Date :
PAGE NO. :
VAANI

JAVASCRIPT

I. use control + L to clear console and ctrl + (+) to increase font size/window size. → In console tab.

* **Using the Console**
  - ↳ uses **REPL**
    - **Read**
    - **Evaluate** (Calculate / Compute)
    - **Print**
    - Loop

use ctrl + ↑ for redo, ctrl + ↓ for undo

* **Data types in JS**

7 Primitive types :-

1. Number
2. Boolean
3. String
4. Undefined
5. Null

} frequently used

6. Bigint
7. Symbol

} less used

typeof : it is an operator which defines the variable/value.

Eg. ⇒ a = 10 ⇒ typeof a ⇒ 'number'

⇒ name = "tony stark" ⇒ typeof name ⇒ 'string'

⇒ typeof 'a' ⇒ 'string'

* **Operations in JS**
  - Modulo 8 % 3 = 2
  - Exponentiation (power) 2 ** 3 = 8

* **NAN in JS**

NAN → not a number

eg. 0/0, NAN - 1, NAN * 2

Note :- typeof NaN is a number.

* **Operator Precedence**

Priority order:

1. ()
2. **
3. *, /, %
4. +, -

Note: if same operator or its level same then Left to right.

* let keyword → syntax of declaring variables. eg. let a = 24;

Note:- if we are updating any variable then there's no need of using let.

const keyword → value of constant can't be changed with re-assignment & they can't be re-declared.

e.g. → const year = 2025;
→ year = 2026 // Error
→ year = year+1 // error

e.g. const pi = 3.14;
e.g. const g = 9.8;

var keyword → old syntax of writing variables. eg var age = 23;

* Q. what is the value of age after this code runs?

let age = 23;
age+2; → gives o/p 25

→ 25 ✗ jab tak assign nhi karoge variable ki value update nhi hogi.
23 ✓ Ans.

∴ age = age + 2; Now age = 25.

* Unary Operator

a+b
a*b
a/2
} binary operators since, two operands are participating.

a = a+1
a += 1
a++
} increment
} dec, decrement
} are called unary operator.

* Note:- • Pre-increment (change, then use)

let age = 10;
let newAge = ++age;
→ 1. age = 11
2. newAge = 11

• Post-increment (use, then change)

let age = 10;
let newage = age++;
1. ~~age = 11~~ newage = age
2. age = 11

e.g. let age = 10;
let newAge = ++age;
age → 11
newAge → 11
~~newage = age++;~~

ye bhi update
iff update

~~NewAge = age++;~~
// age = 10;
newAge = age++;
age → 11
newAge → 10.

Date :
PAGE NO. :
VAANI

Q. What is the value of each variable in each line of code?

```
let num = 5;          num = 5
let new = num++;      new = 5, num = 6
new = ++num;          num = 7, new = 7
```

* Identifier rules:-

- all JS variables must be identified with unique names (identifiers).

Rules:-

- names can contain letters digits, underscores & $ sign (no space). ($ - dollar)
- names must begin with letter.
- names can also begin with $ and _ (underscore).
- names are case sensitive.
- reserved words can't be used as names.

- Ways of writing identifiers :-
  1. camelCase (first word small, next word Capital) e.g. let fullName; → Conventional for JS, JAVA, C++.
  2. snake_case → conventional for Python
  3. PascalCase

- In J.S. type change is possible. e.g. let a = 10;
  ⇒ a = true;
  dynamic typed

* Typescript → designed by microsoft
  ↳ Static typed.
  - e.g. a = 5;
    a = true; ✗ → but correct for JS ∵ it is dynamic typed.
    a = 10; ✓

Q. let Pass = True; // error ∵ there is no word like True in J.S.
let Pass = true; ✓ (boolean) " keyword in JS"
let Pass = 'True'; ✓ acting as string.

Strings

- if single letter we can use both single & double quotes ex char = 'a'.
- if more than letter ie word we must have to use double quotes.
- spaces always included.
- to print this → this is an "apple"

  let str = 'this is an "apple"';
- Note:- let name = "TONY STARK";

  name[1] = 'O'

  name[4] = ' '

  name.length = 10

  name[name.length] = undefined.

  name[name.length-1]; K.

  "pratik".length = 5.

  "pratik"[6] = K

Note:- length of space is 1 and length of empty string "" is 0.

- concatenation :- adding strings.

  ↳ "tony" + "" + "stark" = "tonystark"

  "tony" + 1 = "tony1".

- uno

Undefined & null in JS

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

↳ represent intentional absence of any object value.

e.g. let a = null;

a = → null

type of a = 'object'.

let b;

→ undefined

Date : 17/07/24
PAGE NO. :
VAANI

J-S : 2

- let pencil = 10;
  let pen = 5;
- (1) → console.log("the total price is:", pencil + erasor, "Rupees"); (in case of Java value used + in separation here ',' or (+) for concatenation)
- (2) → console.log(`the total price is: ${pencil + erasor} Rupees`); (strictly strictly)
  ↳ backtick

- === operator checks no. as well as type.
  e.g. "123" === 123 (false) but "123" == 123 (true)
  0 == '' (true) but not for ===
  null == undefined (true " " " "

J.S : 3

alert & prompts → takes from alert.
↳ for popup msg on top.
give something

- alert("this is a simple text");
- but let first = prompt("enter your first name:");
  " last = " "
  console.log(alert("Welcome" + " " + first + " " + last + " " + "!"));

- let msg = " --- ";
  msg.trim(); → trims spaces at start & end (not from b/w)
  msg.toUpperCase(); → all capital
  " " Lower " → all lower

- message chaining using '.' → [priority from left to right.]
  console.log(msg.trim().toUpperCase()); debugg check left→R or R→L

- msg.slice(5,11); → [5,11) → tak ka index hi bachenge (open at 11 (not included))
  msg.slice(5); → 5 se last tak. or msg.slice(5, msg.length);
  msg.slice(-1); → 1 from last will be o/p e.g. 'e' in college
  msg.replace("Apna", "College"); // msg = Apna college then College college
  "Mera"; // mera college

msg.repeat(4); → repeats msg content 4 times.

- Arrays:-

let a = ["x", "y", "z"];

console.log(a); → ['x', 'y', 'z']

→ here we can make array that can take all 4 types.

let b = ["x", 10, 7.42];

b.push("10"); adds 10 at last index.

b.pop(); removes from last.

b.shift(); pops from start.

↳ used for blocking someone

b.indexOf(7.42); → gives index of value

b.includes(742); → true (checks value)

→ slice in arrays (slice[start, end]) → you know how to use

removes

→ splice in arrays → splice(start, delete count, item1, item2, ...)

removed

colors = ["blue", "red", "pink", "orange", "green", "yellow"];

removed colors.splice(4); → green & yellow ∴ start always included

" " (0,1); → "blue" → removed

" " (0,1, "black", "grey"); → red nikal gya (not grey ∵ written in item but delete count matters).

console.log(colors); → 'black', 'grey', 'pink', 'orange'

→ Sort in arrays:- (directly).

colors.sort(); → used for chars & strings only not nos

J.S. : 4

- Loops

let heroes = [["a", "b", "c"], [-----]];

for(let i=0; i< heroes.length; i++){

console.log(i, heroes[i], heroes.length) // o/p

for(let j=0; j< heroes[i].length; j++){

console.log(`${j} = ${heroes[i][j]}`);

}

}

[${variable name only}]

Date :
PAGE NO. :
VAANI

Use of $ literal :- (last time hai samajh lo)!

```
const name= "Alice";
const age = 30;
console.log("Hello, my name is" + name + "and I'm" + age + "years old");
console.log(`Hello, my name is ${name} and I'm ${age} years old`);
```

for of loop :-

```
let fruits = ["mango", "apple", "guava"];
for (fruit of fruits) {
    console.log(fruit);
}
```

o/p
mango
apple
pineapple
guava

```
for (letter of "apna college") {
    console.log(letter);
}
```

o/p
a
p
n
a
c
o
l
l
e
g
e

To-Do App :

→ Whenever combining JS + HTML dont forget to use script tag at last of body.

```
<h1> To-Do App </h1>
<h3> Enter "list" for show tasks </h3>
"     "  "add" to add    "   "
"     "  "delete" " delete "   "
"     "  "quit"  " quit app "
```

all tasks inside ← 2 lines

app.js

```
let todo = [];
let req = prompt("enter req");
while (true) {
    if (req == "quit") {
        break;
    }
    if (req == "list") {
        console.log("_____");
        for (let i=0; i< todo.length; i++) {
            console.log(i, todo[i]);
        }
        console.log("_____");
```

```
else if (req == "add"){
    let task = prompt ("Enter task");
    todo.push (task);
    console.log ("task added");
else if (req = "delete") {
    let idx = prompt("Enter index to delete");
    todo.splice (idx, 1);
    console.log ("deleted");
}
req = prompt("enter your another req");
}
```

1 item to delete of respective index

J.S. :5

Objects in Javascript

```
let student = {
    name : "XYZ",
    roll : 32,
    marks : 93.2
    null : "not a boolean"
}
```

Two ways to access this individual :-

(1). student["name"]; //XYZ

(2). student.marks; //93.2

Difference b/w these 2 ways are :-

→ we can't store variable in second method

i.e.
```
let rate = "price";
item[rate]; //$32
item.rate; //error
```

```
let item = {
    price : "$32",
    discount : 75,
    color : ["red", "black"],
}
```

Note: everything written inside sq. brackets for accessing particular objects are string i.e. all keys are string type e.g. null here not gives boolean if item["null"] but ! not in case of dot '.' operator.

Adding / updating / deleting values inside object externally :-

```
item.price = "87";      updated .
item.quality = "good";  added
delete item.price;      // deleted
```

Nested Objects :-

```
let class = {
    aman : {
        marks : 97,
        city : "Delhi"
    },
    xyz : - -
    :
    :
```

access individually as → class.aman.city ;

Arrays of Object :-

```
const bioData = [
    {
        name : "aman",
        grade : "C",
        city : "Delhi",
    },
    {
        - - -
    },
    {
        - - -
    }
];
```

accessing → bioData[0]; gives { name: 'aman', - - - }

bioData[0].grade ; // C

- **Useful Math methods :-**

→ Math.PI ;

→ Math.abs(-23) ; // 23 , floor (you know! (GIF)) , ceil (opp. of GIF) , random [0,1)

→ For random b/w 1 to 10 → Math.random() * 10 ; actually → 0 to 9

for integer specific → Math.floor(math.random() * 10) ;

from 1 to 10 strictly → " " " +1 ;

* **J.S. : 6**

- **functions**

e.g.
```
function print1to10 () {
    for (let i = 1 ; i <= 10 ; i++) {
        console.log (i) ;
    }
}
print1to10 () ; ⟶ calling
```
(for ... console.log(i); } — Body)

e.g.
```
function sum (a,b) {
    return a+b;
}
console.log (sum (36,4)) ;   // 40
```
(sum called)

Nested call → " " (sum (sum (4,16) , 20)) ; // 40

→ func. for Concatenation of string
```
let str = ["Hi", "hello", "bye", "why"] ;
function concat (str) {
    let result = " " ;
    for (let i=0; i< str.length ; i++) {
        result += str[i];
    }
    return result ;
}
```
(starting with undefined if not used this)

call to check the

Date :
PAGE NO. :
VAANI

→ Scoping :-
↳ determine accessibility of variables, objects & func.
→ 1. Function
→ 2. Block
↳ 3. Lexical

1. Function Scope
⇒ Variables defined inside fn. are not accessible from outside func.

eg. let sum = 54; → GLOBAL SCOPE

```
function calsum (a,b) {
    let sum = a+b;          → FUNCTION SCOPE
    console.log (sum);
}
calsum (3,4);
console.log (sum);
```

O/P
7
→ 54

2. Block Scope
⇒ variables declared inside a block cant be accessed from outside block.

3. Lexical Scope
⇒ a variable defined outside a function can be accessed from inside another function defined after the variable declaration.
The opposite is NOT TRUE.

eg.
```
function outer () {
    let x = 5;
    let y = 6;
    function inner () {
        console.log (x);
    }
    inner ();
}
outer ();
```

O/P
10

Must to

Date :
PAGE NO. :
VAANI

Functional Expression

↳ is that type of nameless fn. which is declared with a variable. with .

e.g. let fun = function (a,b){
return a+b;
};
fun(2,3); → variable name is used to call that fn.

High Order Functions

1. which takes multiple arguments 2. returns a fn.

```
function Mgreet (func, n) {
    for (let i=1; i<=n; i++){
        func();
    }
}

let greet = function(){
    console.log("hello");
};

Mgreet(greet, 10);
```

o/p
hello
;10 times

```
function OddEven(request){
    if (request == "odd"){
        return function(n){
            console.log(!(n%2==0));
        };
    } else if (request=="even"){
        return function(n){
            console.log(n%2==0);
        };
    } else {
        console.log("wrong I/P");
    };
let request = "odd";
```

access in console :-

```
let fun = OddEven(request);
fun(5);
```

⇒ True

Methods - functions defined inside an object ★

```
const calculator = {
    num : 55,
    add : function(a,b){
        return a+b;
    },
    sub : function(a,b){
        return a-b;
    },
    mul : ...
};
```

calculator.add(3,9);
= 12

Date :
PAGE NO. :
VAANI

Or directly,

```
const calculator2 = {
    num: 55,
    add(a,b){
        return a+b;
    },
}
```

* J.S : 7

- this keyword → In obj. methods can't directly access key so we use this

eg.
```
const student = {
    name: "aman",
    eng : 96,
    math : 64,
    chem : 89,
    getAvg(){
        let avg = (this.eng + this.math + this.chem)/3;
        console.log(`${name} got Avg mars of ${avg}`);
    }
}
```

if we write, this.(eng + math + chem)/3 → error that eng is not defined

*** ∴ 'this' keyword refers to the object that is executing the current piece of code.

↓

matlab if e.g. alert("hello"); → than by default its written → window.alert("hello");

↓

universal outsider

- TRY AND CATCH

↳ use case: when we get an error then anything written after that doesn't execute!

(always mention both).

// console.log(a); error 'a' is not defined

∴ try {
    console.log(a);
}
catch (err) {
    console.log ("caught error in a");
    console.log (err); // reference error 'a' is not defined.
}
console.log("hello"); → executable ✓

e|= element

- **ARROW FUNCTIONS / CALLBACKS**

```
const sum = (a,b) => {
        console.log (a+b);
};
const pow = (a,b) => {
        return a**b;
};
const cube = n => {          → brackets are not necessary for single parameter.
        return n*n*n;
};
```

use basic call to access e.g. sum(2,3);

→ implicit return => const mul = (a,b) => (a*b);

- **Set Timeout** → (used for API calls)

// setTimeout (func, 4000); → syntax
(func → callback function; 4000 → time in (ms))

=> setTimeout(() => {
        console.log(" mera college");
}, 4000); → shown after ~~4000~~ 4 seconds.

console.log ("welcome to"); → shown first than above = immediate

Date
PAGE NO
VAANI

Or directly,

```
const calculator2 = {
    num: 55,
    add(a,b){
        return a
    },
}
```

- **Set Interval**

```
let id1 = setInterval(()=>{
    console.log("...");
},2000);
console.log(id1); → to access
```

↓

repeats after every 2 seconds

∴ clearInterval(id1) to stop.

↳ in console.

- **'This' in arrow func.**

gets apne 'ex' bahar ka "~~parent~~ scope"

```
const classmate ={
    name : "aman",
    age : 43,
    getName : function(){
        console.log(this); // gives for classmate obj not windows
        return this.name; // aman
    },
* getMarks : ()=>{
        console.log(this); // window
        return this.marks; // undefined
    }
```

reason: 'this' for arrow function do not gets ~~parent~~ scope rather ~~parent~~ ~~of~~ parent's scope.

here scope → classmate , parent scope → window

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

parsing arrow func. inside callback →

∵ scope = function
p. ∞ = classmate

```
* getInfo2 : function(){
    setTimeOut(function(){
        console.log(this); //window
    },2000);
}
```

solo

Date :
PAGE NO
VAANI

* J.S. # 8

FOREACH

```
let arr = [1,2,3];
let print = function(el){          el = element
        console.log(el);
};
arr.forEach(print);   → O/P 1 2 3
```

or directly,

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

O/P 1 2 3

- forEach for Objects :-

```
let student = [                    → obj. of array.
{
  name : "aman",
  marks : 23
},
{ ==
},
{ ==
}
];
student.forEach((keys) => {
        console.log(keys.marks);   → gives marks of all students
});
```

- Map and Filter

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

O/P access → double ↵ [2,4,6]

→ filter is used to make some changes

```
let edit = nums.filter((el) => {
        return el%2 == 0;
});
```

O/P 2

* every and some

AND OR

eg. In console tab :-

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

[2,4]. » » » » ; // true

Ily some → gives true for both until we take only odd nos.

(You know the rizz)

- Reduce

↳ reduce the array to single value

used for min, max, sum, etc of array

Sum → eg. let int = [1,2,3,4,5];

let sum = int.reduce((acc, el) => acc+el); (el → element.)

console.log(sum); //15. (acc → accumulator)

Proof => initial acc=0, el = element at 0th index

∴ acc+el = 0+1 = 1 ∴ acc updates to 1.

∴ 1+2 = 3 ∴ acc=2

3+3 = 6

6+4 = 10 → 10+5 = 15 ans

MAX → e.g. let arr = [4,3,6,8,2,1];

Basic approach

let max = arr[0];

for(let i=0; i<arr.length; i++){

if(max < arr[i]){

max = arr[i];

}

console.log(max);

Date :
PAGE NO. :
VAANI

MAX using reduce eg. let arr = [4, 3, 6, 8, 2, 1];

```
let max = arr.reduce((max<el) => {
    if (max < el){
        return el;
    }else{
        return max;
    }
});
console.log(max);
```

- **Default Parameters**

```
function sum(a,b = 3){
    console.log(a+b);
}

sum(2); → // 5

sum(2,4); → 6  ∵ default ई if nhi hoga tabhi use hoga
                but 'a' mention mandatory hai.
```

eg.
```
function sum1(a=3,b){
    console.log(a+b);
}
sum(2); → error (b का mention करना is mandatory)
```

- **SPREAD**

is use for selecting & printing nos. Individually → same as VarArgs

```
let arr = [2,4,6,7,8,9,10];
// math.min(array[0], array[1], .....);
directly console.log(Math.min(...array));
```

```
console.log("...mera");   O/P
                          m
                          e
                          r
                          a
```

Day-1 Time: 20/7/24, 8:00pm
☆ of No call pickup till Day-7.

Date: 20/07/24
PAGE NO.:
VAANI

* Spread for Array Literals

eg. let newArr = [1,2,3,4,5];
let copy = [...newArr];
copy.push(6);
console.log(copy); [1,2,3,4,5,6];

eg. let odd = [1,3,5,7];
let even = [2,4,6,8];
~~console.log(...odd;~~ let num = [...odd, ...even];
console.log(num); // [1,3,5,7,2,4,6,8]. ∴ order matters

- Spread for Object Literals:-

eg. const data = {
email: "abc@gmail.com",
password: "32456"
};
const dataCopy = {...data, id:23, country: "India"};
console.log(dataCopy); // {email: 'abc@gmail.com', password: ...
id:23, country: India}.

eg. let obj2 = {..."hello"};
console.log(obj2); // {0:'h', 1:'e', 2:'l', 3:'l', 4:'o'}

- REST

DOUBT

function sum(...args){
for(let i=0; i<args.length; i++){
console.log("you gave:", args[i]);
}}
sum(1,3,4,5,2);

O/P
you gave 1
" " 3
" "

• Destructing

let names = ["alice", "anay", "aman", "robot"];

without destructors :→ let winner = names[0];
" second = " [2];
...

using " → let [winner, runnerUp, third, ...other] = names;

In console → winner ↵ // alice → values indexwise

→ using object literals &-

```
const friend = {
    name : ...,
    age : ...,
    password = ...,
    subject = ["hindi", "maths"];
}; city = "Delhi".
```

let { password : secret, city = "Mumbai", city1 place = "xyz", area = "Brij nagar" } = friend;

Now password can be accessed via ~~let~~ secret, city, place bcz we give Delhi, in destructor we give default only

* J.S. &g

• DOM (document object Model)

↳ represent a doc. with a logical tree, it allows us to change webpage (HTML) content.

use console.dir(document); to check directory

• Selecting elements

1. By id

document.getElementById("mainImg"); (double quote ∵ it is variable not object)

let objImg = " " " " " " " .

check directory using ~~a~~ objImg. console.dir(objImg);

(from directory, select → objImg.src // get link by checking)

Date :
PAGE NO. :
VAANI

- selecting description with id

document.getElementById("description");

console.dir(" " " " " " " " " "); check objects to use

" " (" " " " ("abc")); //null :- no Obj. like this

2. By Classname

document.getElementsByClassName("OldImg")[0]; gets img tag of 0th indx image

" " " " " " " " [1]; 1st

or directly :-

let smallImages = document.getElementsByClassName("OldImg");

for(let i=0; i< smallImges.length; i++){

console.dir(smallImages[i]); // get obj. of all 3 img.

" " (" .src);

smallImges[i].src = "newimg.png";

console.log(`value of image no. ${i} is changed`);

}

3. By Tagname

document.getElementsByTagName("p"); //we get 2 elements 0th index of About section & 1st index of CfD (no id) (id = #description)

To change :-

document.getElementByTagName("p")[1].innerText = "abc"; content changed inside CfD.

" " " " " ("span"); //o/p :- no tag like this.

Date 22/07/24
PAGE NO
VAANI

Query Selectors

→ console.dir(document. querySelector ('p')); // selects first p element

(''#description'); // selects first element with id=description)

" " " " ('.oldImg'); // selects first element with class = oldImg

" " " " ("div a"); // selects first anchor tag inside first p tag.

To use all:- " " " .querySelectorAll('p'); // selects all p tag

" " " " ("div a"); // " " anchor ".

→ 3 most important properties:- 1. InnerText 2. textContent 3 innerHTML

let para = document. query Selector ('p');

para. innerText ; // print text inside first p tag

" " HTML ; // " " " " " " but helps in changes tags inside it.

" textContent; // defines how text is defined inside our HTML content not on webpage.

eg. style display-none can be seen here apart from others.

para. innerText = " Hi, there! " ; // changes to this

" " " = "Hi, there ! " ; // no change

" " HTML = " " " ; // now 'there' will be bold

→ to underline heading:-

let heading = document. query selector ('h1');

heading. innerHTML = " <u> Spider man </u> ";

Date :
PAGE NO. :
VAANI

- **Manipulating Attributes** (Getters & Setters) — backtick

```
let img = document.querySelector(`img`);
img.getAttribute('id');
img.setAttribute('id', 'spidermanImg'); // id name changes
```

Note: leads to user styling removed ∵ we desire class name & assign this.

eg. `img.getAttribute(`class`);` — class

`img.setAttribute(`class`, `images`);` // id name changes.

- **Manipulating Styles**

```
let heading = document.querySelector('h1');
heading.style.color = 'purple'; // peter parker in purple.
" " .backgroundColor = 'yellow';
```

camelcase → Note :- in CSS we have hyphen (-)

- **ClassList** → gives no. of classes in an element.

```
let heading1 = document.querySelector('h1');
heading1.classList.add('green');
```

↳ added a class to h1 tag. Now if we have styling as .green{ } then its color changes.

```
" " " " ('underline'); → class added.
" " " .remove('green'); class removed
```

~~heading1.classlist~~

```
heading.setAttribute('class', 'green'); // green named class added, earlier underline named class removed
```

not preferred use classlist

```
heading1.classList.contains("underline"); //true/false
```

- **toggle** → if yes ↔ no.

use `heading1.classList;` to check how many are present.

```
heading1. " .toggle("green"); // true (now added)
" " " ("underline"); // false.
```

- **Navigation**

```
let h4 = document.querySelector('h4');
h4.parentElement ; // gives <div class="box">...</div>
h4.children // nothing :: nothing inside H4 tag.
let box = document.querySelector(".box");
box.children ; //2 h4, ul
box.childElementCount; // 2.
let ul = ..
ul.parentElement ; //div class of box.
ul.childrenElementCount ; //3  li li li
ul..children[2].previousElementSibling // ul.children[1]
```

```
ue :-  let img = docum. - ...
img.previousElementSibling ;  //<h1> spiderman </h1>
"  "  "  "  ".style.color = "orange" ; // color change to orange.
```

- **Adding Elements** → used to create element & after creating insert them.

```
let newP = document.createElement('p');
console.dir(newP) ;  // null in all item.
newP.innerText = "Hi! there";
let body = document.querySelector('body');
body.appendChild(newP);
```
↳ to add
prepend → shifts

```
eg.  let btn1 = document.createElement('button');
btn1.innerHTML = "New Button !!!";
let p = document.querySelector('p');
p.insertAdjacentElement("beforebegin", btn1); //to insert in begin of para.
"  ("after" , " );
beforeend
afterend
```

to remove this btn → `let body1 = document.querySelector('body');`
`body1.removeChild(btn1);`

p.remove(); whole para removed

body. " " body ".

* J.S. 810

→ Dom Events

↳ Signals that something has occured after a user action

using js in inline : <button onclick = "console.log('button was clicked');
console.log('mera college')" > click me </button

↳ not beneficiary :. for 100 likes we can't make 100 inlines

→ for first button: let btn = document.querySelector("button");
console.dir(btn);
btn.onclick = function(){
alert("button was clicked");
console.log("button was clicked");
}

→ for many buttons: let btns = - - - - - - - - - - All("button");
for(btn of btns){
btn.onclick = sayHello; // don't sayHello(); → execute all.
" = college;
btn.addEventListener("click", sayHello);
" " " " (" , college);
" " " " ("dblclick", function(){
console.log("you double clicked");
});

for others google

Q mdn mouse click elements

btn.onmouseenter = function(){
console.log("you selected"); → whenever you hover on button with cursor
}

console.dir(btn); // find all ...
}

```
function sayHello(){
        alert('hello');
}
" college "
"       " "college"
"
```

→ check activity on file for generating random color.

- Event listener for elements (not just buttons)

```
let p = document.query....
p.addEventListener("click", function(){
        console.log("para clicked");
});   dly. for mouseenter, dblclick, etc.
```

~~eg. let~~

- 'this' in Event listener → used inside callback.

```
let newBtn = dqs.("button");
newBtn.addEventListener("click", function(){
        console.log(this);
        " dir(" )
        "  "  (".innerText);
        this.style.backgroundColor = "blue";
});
```

dly using this we do for "p", "h3", "h1" etc.

but to remove this repeating (redundancy) :-

```
let newBtn = dqs("button");
let p      = "("p");
" h3
" h1
function changeColor(){
        console.dir(this.innerText);
        this.style.backgroundcolor = "blue";
}
newBtn.addEventListener("click", changeColor);
p.
h3:
```

Keyboard Events

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

let inp = dqs("input");
inp.aEL("keydown", function(){
        console.log("key pressed");
});

- - - "keyup" - - -
        - - - "key released"

inp.aEL("keydown", function(event){
        console.log("key:", event.key);   // anything typed.
        " " ("key code =", event.code);
        " " ("key pressed");
```

Used for moving characters ↑ → ↓

```
e.g. inp.aEL("keydown", function(event){
        console.log("code =", event.code); //ArrowUp
     if(event.code == "ArrowUp"){
        console.log("character moves up")
     }
});
```

" Down
" Right
" Left

- FORM EVENTS

```
let form = doc.qs ("form");
form.aEL ("submit", function(event){
        event.preventDefault(); → to stop action that occur after clicking on submit.
        alert ("form submitted");
let inp = dqs ("input");
console.dir (inp); → check. where is value.
console.dir (inp.value); → value that typed after submitting
let user = dqs ("#user");
pass          #pass

console.log (user.value);
"     pass     "

alert→ alert (`Hi ${user.value}, your password is ${pass.value}`)
```

- More Events

1. change Event → occurs when value of element has changed only works on <input>, <textarea> & <select>.

```
let form = dqs ("form");
form.aEL ("submit", function (event){
        event.preventDefault ();
    });
let user = dqs ("#user");
user.aEL = dqs ("#user ("change", function (){
        console.log ("change event");
        "     ("final value :", this.value);
```

2. input event → it fires when value of <input>, <"">, & <""> elements have been changed.

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

Date :
PAGE NO. :
VAANI

J.S. #11

Visualising Call stack

For debugging
↓
sources tab
↓
breakpoints
↓
select breakpts. three()
↓
refresh page
↓
check call stack using ⬇

```
function one () {
    return 1;
}
function two () {
    return one() + one();
}
function three () {
    let ans = two() + one();
    console.log(ans);
}
three(); // 3. easy
```

- JS is single threaded → i.e. if ~~only one~~ any code written then one time only one execute.

1. Synchronous Nature

```
let a = 10;
console.log(a);   ↓ line by line execution.
let b = 20;
```

2. Asynchronous Nature → achieved via Callstack
   ↓
   controls api req. + DBMs
   Multithreaded ← (browser end)

e.g. setTimeOut

→ Drawback :-

* Callback Hell

```
let h1 = document.querySelector("h1");
setTimeOut(() => {
    h1.style.color = "red";
}, 1000);
"orange"
, 2000);
"blue"
, 3000);
```

↳ bad way!

∴ function changeColor (color, delay){
setTimeOut (() => {
hl.style.color = color;
}, delay);
}
changeColor ("red", 1000);
" "orange", 2000);
" "blue", 3000);

bad! keep note of each time to process.

∴ use callback nesting

function changeColor (color, delay, nextColorChange){
setTimeOut (() => {
hl.style.color = color;
if (nextColorChange) nextColorChange ();
}, delay);
}
changeColor ("red", 1000, () => {
changeColor ("orange", 1000, () => {
- - - - - "yellow", 1000, - - -{
- - - - - "blue" - - {
- - - - "green - - - -{
});
});
});
});
});

→ Hell!

eg. function saveToDb (data, success, failure) {
let internetSpeed = Math.floor (Math.random()*10)+1;
if (" < 4){
success();
} else {
failure ();
}
}

Date :
PAGE NO. :
VAANI

```
saveToDb(
    "mera college",
    ()=>{
        console.log("succes : data saved");
        saveToDb(
            "hello",
            ()=>{
                ... ("succes 2 : data 2 saved");
                saveToDb(
                    "Pinku",
                    ()=>{ console.log("succes 3 : n 3 n n");
                    },
                    () => {
                        console.log("failure 3: weak connection");
                    });
                },
                ()=>{
                    ... ... ("2 : n n");
                });
            },
            ()=> {
                console.log("failure : weak connection");
            });
```

O/Ps

failure, data 1 saved failure 2, data 1, 2 saved failure 3, 1,2,3 all saved

- ∴ To resolve this we use Promises

  ↓

  eventual completion of asynchronous op^n & resulting value.

e.g. function saveToDb(data){

```
return new Promise((succes, failure)=>{
    let internetspeed = Math.floor(...)
    if(internetspeed >4){
        succes("succes : data saved");
    } else{
        failure("failed");
    }
});
```

} use saveToDb("mera college"); → to check.

Fulfilled → resolved.

Date: 22/07/24
PAGE NO.:
VAANI

- Then and catch
  - When promise fulfilled
  - " " rejected

```
let request = saveToDb("mera College");
request.then(() => {
    console.log("promise resolved");
    console.log(request);
});
.catch(() => {
    " " " rejected
    " " "
});
```

// can do nesting here also.

Improved

```
saveToDb("apna College")
.then((result) => {
    console.log("data1 saved");
    " " ("result of promise:", result);
    return saveToDb("hello");
})
.then(() => {
    .....2.....
    .....3.....
})
.catch((error) => {
    " " ("promise rejected");
    " " ("result of promise:", ~~result~~ error);
});
```

- applying Promises to callback hell :-

```
h1 = doc.qS("h1");
function changeColor(color, delay){
    return new Promise((resolve, reject) => {
        let Timeout(() => {
            h1.style.color = color;
            resolve("color changed");
        }, delay);
    });
}
```

M-1 : hell → changeColor ("red", 1000, () => {
changeColor ("orange", 1000, () => {
. . .

M-2 : Prevented !

```
changeColor("red", 1000)
.then (() => {
    console.log("red color");
    return changeColor("orange", 1000);
})
.then
.then
});
```

* J.S 812

• Async keyword

```
async function greet() {
    throw "404 not found";
    return "hello";
}
greet()
.then ((result) => {
    console.log("promise was resolved");
    " " ("result was", result);
})
.catch ((err) => {
    console.log("promise rejected :", err);
});
```

- **await keyword**
  - always use with async

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

async function demo(){
    await getNum();
    "      "
    "      "
}
//call via demo();
```

- **Handling Rejections**

e.g. If after selecting h1 & performing color changes using Promise then →

```
async function demo1(){
    await changeColor("red", 1000);
    :
    let a = 5
    console.log(a);
}
demo1();
```

→ not depended on any color change still will not execute if any Promise rejected.

∴ use try and catch!

⇒

```
async function demo1(){
    try{
        await changeColor("red", 1000);
        :
    }
    catch(err){
        console.log("error caught!", err);
    }
    let a=5;
    console.log(a);
}
```

→ rejection resolved!

- **API** (application Programming Interface)

e.g. when we like a photo in Insta then without reloading, likes count increases.

→ APIs dealing with http : WebApi

e.g. catfact.ninja/fact
boredapi.com/api/activity

- **JSON** (javascript online notation)

when we return data it is in json format only only.

Similar to JS objects, except 2 differences :-

1. key written in JSON format are in strings.
2. ~~no~~ undefined - value throws error in json but not in js.

- **Parsing JSON data** : (json data → js obj)

```
let jsonRes = `{"fact": ..... random copied from catfact.ninja...}`;
let validRes = JSON.parse(jsonRes); ☆
console.log(validRes.fact);
```

- **Stringify JSON** : (js obj → json data)

```
let ~~jsonRes~~ student={
        name : "aman";
        marks : 98
    }
in console → JSON.stringify(student); ✓
```

- **Testing API requests** (tools)

1. Hoppscotch 2. Postman

- AJAX (asynchronous javascript and xml) → overall process in which we request our api & get data.

  why?

  when we req. API through JS then APIs are working asynchronously & return response in JSON format (earlier it returns in XML ∴ AJAX but actually it is AJAJ).

- HTTP Verbs

  1. GET : whenever we put valid api link in https format in hollscoth it converts into json format, this process is GET.
  2. POST : along with the api when we post our data also.
  3. DELETE : we use this to delete our api request.

  We'll use this when we make our custom API.

- Status Codes

  1. 200 : OK (when api call is succes)
  2. 404 : Not found (" " " does not exist)
  3. 400 : Bad request
  4. 500 : internal server error (error from serverend)

- Additional Info. in URLs

  putting extra for search directly using link

  e.g. https://www.google.com/search?q=harry+potter

  (? : key, harry+potter : value)

- ```
  let url = "https://catfact.ninja/fact";
  async function getFacts(){
      try {
          let res = await fetch(url);
          let data = await res.json();
          console.log(data.fact);
      }
      catch(e){
          console.log("error - ", e);
      }
  ```
  ```

Date :
PAGE NO. :
VAANI

JS : 13

in fetch - when we getting data that data was not exact json data ∴ we have to parse it first. then

- AXIOS — library to make HTTP request
  ∴ mention script src in index first to access.

```
let url = " - - - - - ";
let btn = dqs ("button");
btn.aEL (" click", async() ⇒ {
        let fact = await getFacts ();
        console.log (fact);
        let p = dqs ("#output");
        p.innerText = fact;
        });
async function getFacts (){
        try {
            let ~~resp~~ res = await axios.get(url);
            return res.data.fact;
        }
        catch(e){
            console.log ("error - ",e);
            return "No fact found";
        }
```

Ally, get for dog images.

- Activity using query strings in axios.
  getting university names :-

```
let url = "http://universities.hipolabs.com/search?name=";
let btn = dqs ("button");
btn.aEL ("click", async ()⇒ {
        let country = dqs ("input").value;
        console.log(country);
let colArr = await getColleges(country);
console.log(colArr);
show(colArr);
});
```

Date :
PAGE NO. :
VAANI

```
function show(colArr){
    let list = das("#list");
    list.innerText = "";   → to not let other search appendout
for(col of colArr){
    console.log(col.name);
    let li = document.createElement("li");
    li.innerText = colname;
    list.appendChild(li);
}}
async function getColleg(country){
    try{
    let res = await axios.get(url+country);
    return res.data;
    }catch(e){
        console.log("error:", e);
        return[];
    }}
```