JAVASCRIPT

→ Javascript is a light weight object oriented programming language which is used by several website for scripting the webpages.

→ It is an interpreted, full fledged programming language that enables dynamic interactivity on website when applied to an HTML document

- Why Javascript Born?
  To create interactive website
  - Client side validation
  - Pop up
  - Event on click etc..

- Where is Javascript now?
  - ~~website~~ website Client side (Js, Jquery, React js ...)
  - website server side (Node Js, Express Js)
  - Mobile development (React Native, Phone Crap, ionic..)
  - Software Development (electronjs Eg:- VScode)

Q. What is ES6, ES7, ES8 and so on?
A. what is ECMAScript?

Ecma International
→ An organization that create standard for technologies

ECMA-262
→ This is standard published by Ecma International. It contains the specification for a general purpose scripting language.

ECMA script

→ The specification define in ECMA-262 for creating a general purpose scripting language

→ ECMA Script provide rules, details and guidlines that a scripting language must observe to be considered ECMAScript complaints.

Javascript

→ A general purpose scripting language that conforms to the ECMA script specification.

→ By reading the ECMAScript specification, you learn how to create a scripting language.

→ By reading the JavaScript documentation, you learn how to use a scripting language.

→ JavaScript implements the ECMAScript specification as described in ECMA-262

- JavaScript engine
  V8 in chrome
  SpiderMonkey in Firefox
  Chakra in Edge

- Javascript in HTML

Note:
also write code in <head> tag

```
<!DOCTYPE html>
<html lang = "en">
<head>
  <meta charset = "UTF-8">
  <meta name = "viewport" content "width=device-width, ...>
  <title> Document </title>
</head>
<body>
  <h1> JS tutorial </h1>
  <script>
    alert ("Hello world");
  </script>
</body>
</html>
```

Or

- Creating different file for js

This file is stored in a folder name js

first.js
```
alert("Hello");
```

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

also write in <body> tag

Note: ① <script async src = "js/first.js"></script>
as jese hi file download hua vese hi execute kar dega

② <script defer src = "js/first.js"></script>
is pehle document padh lega badh me downloading start kar

- Statement and Comment in Javascript.

```
// this will popup 5
var a;
a = 5;
alert(a);
```

→ use // for comment for single line comment
→ use /* ... */ for multi line comment

- Variables → are containers for data.

→ They can be created using var, let or const keyword

```
var x = 10;
console.log(x);
```

Note: x = null;
y ? undefined;

→ var → declares a variable. It has a function-scoped or globally-scoped behavior.
let → introduce block-scoped variables. It's commonly used for variable that may change their value.
const → declare variable that cannot be reassigned. It's block-scoped as well

- console.log is used to ~~print~~ log (print) a message to the console.

var → variable can be re-declared & updated. A global scope & variable.

let → variable cannot be re-declared but can be updated. A block scope variable.

const → variable cannot be re-declared or updated. A block scope variable.

```
let a;                    o/p: undefined
console.log(a);

const a;                  o/p: error
console.log(a);
```

Datatype

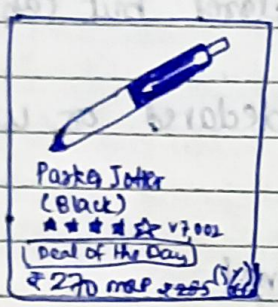
Primitives → Number, String, Boolean, Undefined, Null, BigInt, Symbol

Non-primitives → objects → collection of values

```
const student = {
    fullname : "Kamlesh",
    age : 20,
    isPass : true
};
console.log(student["age"]);
```

console
→ student
* typeof

Q. Create a const object called "product" to store information shown in the picture.




```
const product = {
    title : "Ball Pen"
    rating : 4,
    offer : 5,
    price : 270
};
```

- Operators in JS

1. Arithmetic operators
   +, -, *, /.
   modulus, exponentiation, increment, decrement.
   (%)

```
let a = 5;
let b = 2;
console.log ("a+b=", a+b);      o/p: a+b = 7
console.log ("a-b = ", a-b);    o/p: 3
console.log ("a*b=" , a*b);          10
console.log ("a/b = " , a/b);        2.5
Console.log ("a%b=", a%b);           1
console.log ("a**b =", a**b);        25
```

(5) Ternary operator
   condition ? true output : false output
eg. age > 18 ? "adult" : "not adult";

2. Assignment operators
   =, +=, -=, *=, %=, **=

3. Comparison operators
   ==, !=, ===(Equal to & type), !==(Not equal to & type)
   >, <=, <, <=

4. Logical operators
   && (and), || (or), ! (not)

- **Conditional statements**

(1.) if statement

```
let age = 25;
if (age > 18){
    console.log("can vote");
}
```

(2.) if else statement

```
let mode = "light";
let color;
if (mode === "dark"){
    color = "black";
}
else{
    color = "white";
}
console.log(color);
```

(3) else if statement

```
if (age < 18) {
    console.log ("junior");
} else if (age > 60) {
    console.log ("senior");
} else {
    console.log ("middle");
}
```

Q. Get user to input a number using prompt ("Enter a number:") check if the number is a multiple of 5 or not

~~let num = prompt ("enter a num~~

```
let num = prompt ("enter a number: ");
if (num % 5 === 0) {
    console.log (num, "is a multiple of 5");
} else {
    console.log (num, "is NOT a multiple of 5");
}
```

Q write a code which can grades to students according to their scores:

80 - 100	A
70 - 89	B
60 - 69	C
50 - 59	D
0 - 49	F

```
let score = prompt ("enter your score (0-100) :");
let grade ;
if (score >= 90 && score <=100) {
    grade = "A";
} else if (score >= 70 && score <=89) {
    grade = "B";
} else if (score >=60 && score <=69) {
    grade = "C";
} else if (score >=50 && score <=59) {
    grade = "D";
} else if (score >=0 && score <=49) {
    grade = "F";
}
console.log ("according to your scores, your grade was:"
    , grade)
```

- loops in JS

① for loop

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

② while loop

```
while ( condition) {
    //do some work
}
```

③ do while loop

```
do{
    // do some work
} while (condition);
```

④ for of loop → loop on strings and arrays.

Syntax

```
for (let val of strVar) {
    // do some work
}
```

Eg.

```
let str = "ApnaCollage";
for (let i of str) {
    console.log(i);
}
```

⑤ for in loop → objects and arrays

Syntax

```
for (let key in objVar) {
    // do some work
}
```

Eg.

```
let student {
    name : "Rahul kumar";
    age : 20
    cgpa : 7.5,
    isPass : true
};
for (let i in student) {
    console.log(i)
}
```

↓ to print values

```
console.log("key =", key, "value =", student[key])
```

O/P: name
age
cgpa
isPass

Q Print all even number from 0 to 100

```
for (let num=0 ; num<=100 ; num++) {
    if (num % 2 !==0) {
        //even number
        console.log("num=", num);
    }
}
```

Q Create a game where you start with any random game number. Ask the user to keep guessing the game number until the user enters correct value

```
let gameNum = 28;
let userNum = prompt("Guess the game number:");
while (userNum != gameNum) {
userNum = prompt("You entered wrong number, Guess again:");
}
console.log("congratulations, you entered right number");
```

- **String in JS**

① Create string
```
let str = "Apna College";
```

② String length
```
str.length
```

③ String Indices
```
str[0], str[1], str[2]
```

- **Template Literals in JS**
  A way to have embedded expressions in string

```
`this is a template literal`
```

Imp. ④ String Interpolation
To create string by doing substitution of placeholders.

Syntax
```
`string text ${ expression } string text`
```

```
//String
let str = "Apna College";
console.log(str[8]);

//Template Literals
let specialString = `This is a template literal`;
console.log(specialString);
```

escape characters
\n → next line
\t → tab space.

```
let obj = {
    item : "pen",
    price : 10,
};
```

→ let output = `the cost of ${obj.item} is ${obj.price} rupees`;
console.log(output)

O/P of both are same → console.log("the cost of", obj.item, "is", obj.price, "rupees");
but using template literals it is easy to understand.

- **String Methods in JS**

→ There are built in function to manipulate a string

- str.toUpperCase()
- str.toLowerCase()
- str.trim() // remove whitespaces

Eg.
```
let str = "ApnaCollege";
let newStr = str.toUpperCase();
console.log(newStr)
```

- str.slice(start, end?) // return part of string (accept -ve value)
- str1.concat(str2) // join str2 with str1
- str.replace(searchVal, newVal)
- str.charAt(idx)

Eg. let str = "012345"
console.log(str.slice(1,3)); O/P: 1,2

Eg. let str = "hello";
console.log(str.replace("lo","p")); O/P: help
let str = "hellololo";
console.log(str.replace("lo","p")); O/P: helplolo
console.log(str.replaceAll("lo","p")); O/P: helppp.

Eg. let str = "IloveJs";
console.log(str.charAt(2); O/P: o

Eg. let str = "IloveJs";
str[0] = "S";
console.log(str); O/P: IloveJs

∴ we use replace.
str = str.replace("I","S");
console.log(str); O/P: SloveJs

Q Prompt the user to enter their full name. Generate a username for them based on the input. Start username with @, followed with their full name and ending with the fullname length.

eg. user name = "shradhakhapra", username should be "@shradhakhapra13"

Code:

```
let fullName=prompt("enter your fullname without spaces");
let username = "@" + fullName + fullName.length;
console.log(username);
```

- **Arrays in JS**
  → collection of items → linear.

```
let heroes = ["ironman", "hulk", "thor", "batman"]
let marks = [98,75,48,83,66];
let info = ["rahul", 86, "Delhi"];
```

Eg
```
marks = [97,82,75,64,36];
console.log(marks);
console.log(marks.length);
```

O/P:
```
(5) [97,82,75,64,36]
0: 97
1: 82
2: 75
3: 64
4: 36
length: 5
```

~~typeof~~ typeof marks
'object' → Imp.

Array Indices
arr[0], arr[1]
Eg. marks[0]

Strings → immutable
array → mutable

Looping over an Array

```
let heroes = ["ironame", "thor", "hulk"]
for (let idx=0 ; idx < heroes.length ; idx++){
    console.log(heroes[i]);
}

// for of
for ( let hero of heroes) {
    console.log(hero)
}
```

Q For a given array with marks of student → [85, 97, 44, 37, 76, 60]
Find the average marks of entire class.

Ans

```
let marks = [85,97,44,37,76,60]
let sum = 0
for ( let val of marks) {
    sum = sum + val;
}
let avg = sum / marks.length;
console.log (`avg. marks of the class = ${avg}`);
```

Q For a given array with prices of 5 items → [250, 645, 300, 900, 50] All items have an offer of 10% OFF on them. Change the array to store final price after applying offer.

Sol) if item price 300 10% off → 30 Rs. minus hoga

300 - 30 ⇒ 270

```
let items = [250, 645, 300, 900, 50];
let i = 0;
for (let val of items) {
    let offer = val/10;
    items[i] = items[i] - offer;
    console.log("value after offer = ${items[i]}");
    i++;
}
```

using for of loop

using for loop →

```
for (let i=0; i<items.length; i++) {
    let offer = items[i]/10;
    items[i] -= offer;
}
console.log(items);
```

- **Arrays Methods**

→ Push() : add to end
→ Pop() : delete from End & return
→ toString() : convert array to string.

Eg

```
let foodItems = ["potato", "apple", "litchi", "tomato"];
foodItems.push("chips", "burger", "paneer");
console.log(foodItems);
```

```
let foodItems = ["potato", "apple", "litchi", "tomato"]
let deletedItem = foodItems.pop();
console.log(foodItems);
console.log("deleted", deletedItem); // tomato
```

→ concat() : joins multiple arrays & return result
→ unshift() : add to start
→ shift() : delete from start & return
→ slice() : return a piece of the array
  slice(startIdx, endIdx)
→ splice() : change original array (add, remove, replace)
  splice(startIdx, delCount, newEl1, ...)

Q Create an array to store companies → "Bloomberg", "Microsoft", "Uber", "Google", "IBM", "Netflix"
(a) Remove the first company from the array
(b) Remove Uber & Add Ola in its place
(c) Add Amazon at the end.

Sol
```
let companies = ["Bloomberg", "microsoft", "Uber",
"Google", "IBM", "Netflix"];
companies.shift();
companies.splice(2, 1, "Ola");
companies.push("Amazon");
```

redundancy → repeat

Functions in JS.

Block of code that performs a specific task, can be invoked whenever needed.

Function Definition	Function Call

```
functionName();
```

```
function functionName() {
    // do some work
}
```

```
function functionName(param1, param2...) {
    // do some work.
}
```

↳ parameter.

Eg.

```
function myfunction(msg) {
    console.log(msg);
}

myfunction("I love JS");
```

↳ argument

Note

Eg.

```
function sum(x, y) {
    s = x + y;
    console.log("before return");
    return s;
    console.log("after return");
}
```

↳ act as local variable

⇒ It will not print "after return" bcz we can't write code after return stmt.

function parameter → like local variable of function → block scope

```
let val = sum(3,4);
console.log(val);
console.log(x);
```
→ not print becz. it is a local variable we can only access it within its scope.

Arrow function → (used for small function or work)

→ Compact way of writing a function

```
const functionName = (param1, param2...) => {
    //do some work
}
```

Eg
```
const sum = (a,b) => {
    return a+b;
}
```

Eg
```
// simple function
function mul(a,b) {
    return a*b;
}
```

Use in modern JS.
```
// using arrow function
const arrowMul = (a,b) => {
    console.log(a*b);
    //return a*b;
}
```

Q Create a function using "function" keyword that takes a string as an argument & return the number of vowels in the string

```
function countVowels(str) {
    let count = 0;
    for(const char of str) {
        if(char === "a" || char === "e" || char === "i" ||
           char === "o" || char === "u") {
            count++;
        }
    }
    return count;
}
```

Q Create an arrow function to perform the same task.

```
const countVow = (str) => {
    let count = 0;
    for(const char of str) {
        if(char === "a" || char === "e" || char === "i" ||
           char === "o" || char === "u") {
            count++;
        }
    }
    return count;
}
```

Q what is higher order function/methods

⇒ function that take other function as parameter or return some function

↳ not used for storing → give error

- forEach loop in Array

higher order function

arr.forEach(callBackFunction)

CallbackFunction: Here it is a function to execute for each element in the array.

"A callback is a function passed as an argument to another function."

Syntax:

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

Eg

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

or

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

```
arr.forEach((val, idx, arr) => {
    console.log(val.toUpperCase(), idx, arr);
});
```

Q for a given array of numbers, print the square of each value using the forEach loop

```
let nums = [2,3,4,5,6];
nums.forEach((num) => {
    console.log(num * num);    // or num**2
});
```

or

```
let nums = [67,52,39];
let calcSquare = (num) => {
    console.log(num * num);
};
nums.forEach(calcSquare);
```

- **Some more Array Methods**

- **Map**

Create a new array with the results of some operation. The value its callback return are used to form new array.

Syntax

```
arr.map(callbackFnx(value, index, array))
```

```
let newArr = arr.map((val) => {
    return val*2;
})
```

foreach → normal kuch calculation ya print karana ho.
map → is used when ~~re~~using values we want to create new array.

creating a copy of same array.

```
let nums = [67, 52, 39];
let newArr = nums.map ((val) => {
    return (val);
});
```

⇒ Filter

Create a new array of elements that give true for a condition / filter.

Eg. all even elements

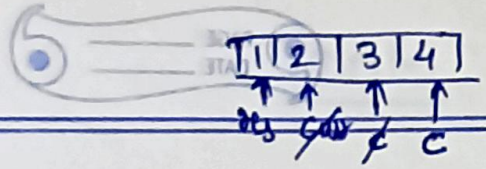
```
let arr = [1,2,3,4,5,6,7]
let evenArr = arr.filter ((val) => {
  return val % 2 ===0;
})

// for odd value. return val %2 !==0;
```

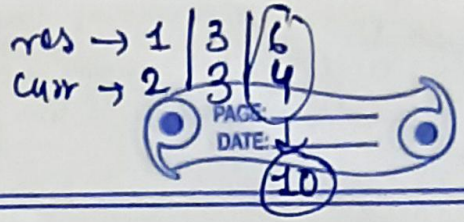
⇒ Reduce

Performs some operations & reduces the array to single value. It returns that single value

Eg

```
const array1 = [1,2,3,4];
// 0 +1+2+3+4
const initialValue = 0;
const sumWithInitial = array1.reduce(
  (accumulator, currentValue) => accumulator + currentValue
  initial value,
);
console.log (sumWithInitial);
// expected o/p: 10
```





Eg let arr = [1,2,3,4];
const output = arr.reduce((res, curr) => {
return res + curr;
});
console.log(output); // output = 10.

Eg To find out largest number.

let arr = [5,6,2,1,3];
Const output = arr.reduce((prev, curr) => {
return prev > curr ? prev : curr;
});
console.log(output);

Q We are given array of marks of students. Filter out of the marks of students that scored 90+.

~~let output = marks = [97,64,32,49,99,96,86];~~
~~let toppers = marks.filter((val) => {~~
return
let marks = [97,64,32,49,99,96,86];
let toppers = marks.filter((val) => {
return val > 90;
});
console.log(toppers);

Q. Take a number n as input from user. Create an array of numbers from 1 to n. Use the reduce method to calculate sum of all numbers in the array. Use the reduce method to calculate product of all numbers in the array.

```
let n = prompt ("enter a number:");
let arr = [];
for(let i=0; i<=n.; i++){
    arr[i-1] = i;
}

console.log(arr);
arr.reduce((res, curr)
let sum = arr.reduce ((res, curr) => {
    return res + curr;
});
console.log("sum =", sum);
let factorial = arr.reduce ((res, curr) => {
    return res * curr;
});
console.log ("factorial =", factorial);
```

Output: 4

(4) [1, 2, 3, 4]

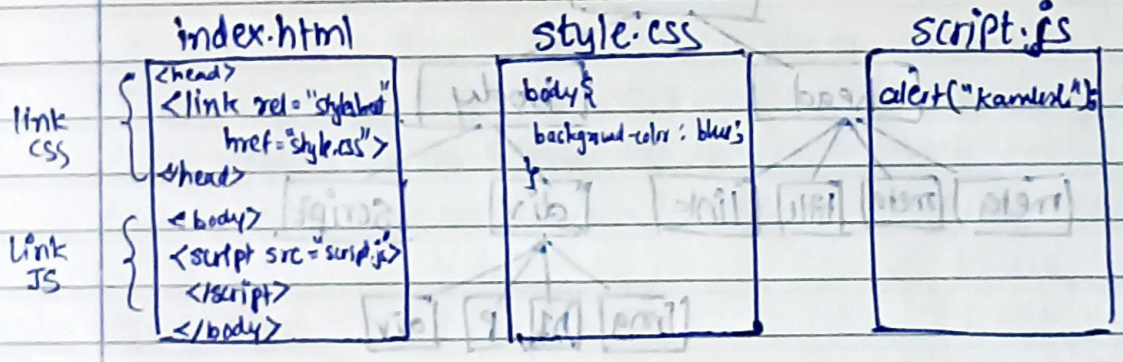
sum = 10

factorial = 24

DOM

3 Musketeers of Web Dev
- HTML (structure)
- CSS (style)
- JS (logic)

Starter code
<style> tag connect HTML with CSS
<script> tag connect HTML with JS



→ If we write JS file differently it helps in
- Readability
- Modularity
- Browser caching

window Object
→ The window object represents an open window in a browser. It is browser's object (not JavaScript's) & is automatically created by browser.
→ It is global object with lots of properties & methods

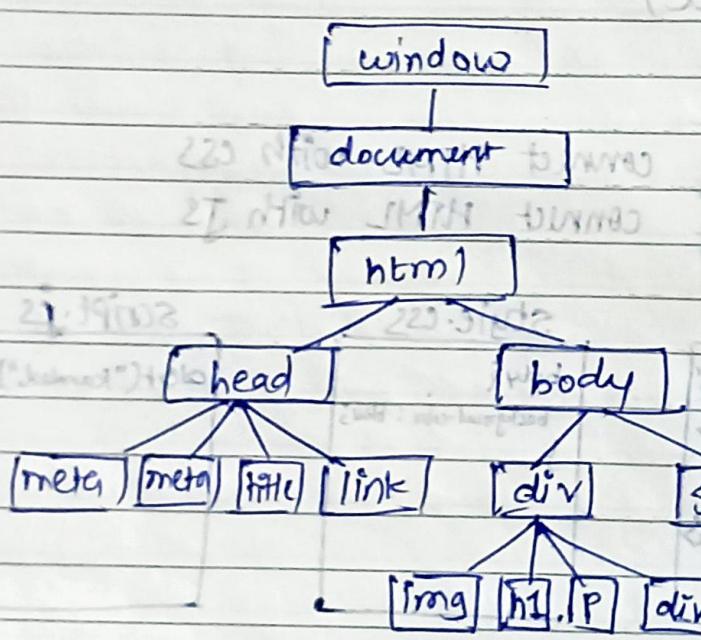
HTML → JS access window.document

all elements of HTML comes in JS and form object and we name that object as document which is available on window object

PAGE
DATE

What is DOM?

→ When a web page is loaded, the browser creates a Document Object Model (DOM) of the page.



console.log → print
to print object
↳ console.dir → document
↓
properties and method printed

DOM Manipulation

- Selecting with id
  document.getElementById("myId")
- Selecting with class
  document.getElementsByClassName("myClass")
- Selecting with tag
  document.getElementsByTagName("p")

#myId .myclass
dot

- Query Selector

document.querySelector("myId / myclass / tag")
//return first element

give a Node List which we can use as array but we can't use methods like map etc

document.querySelectorAll("myId/myclass/tag")
// return a Nodelist

Properties

tagName : return tag for element nodes
innerText : return the text content of the element and all its children
innerHTML : return the plain text or HTML contents in the element
textContent : return textual content even for hidden elements

DOM Tree
- text nodes
- comment node
- elements node

Q. Create a H2 heading with text = "Hello Javascript" Append "from Apna College Student" to this text using JS.

Sol
1. access element
2. property → change

index.html

```
<html>
<head>
</head>
<body>
<h2> Hello Javascript </h2>
<script src="script.js"></script>
</body>
</html>
```

script.css

```
let h2 = document.querySelector("h2");   //access h2
console.dir(h2.innerText);
h2.innerText = h2.innerText + "from Apna College Shda
```

Q Create 3 divs with common class name "box". Access them & add some unique text to each of them.

index.html

```
<body>
  <div class="box"> first div </div>
  <div class="box"> second div. </div>
  <div class="box"> third div </div>
</body>
```

style.css

```
.box {
  height: 100px;
  width: 100px;
  border: 1px solid black;
  margin: 5px;
  text-align: center;
  background-color: red;
}
```

script.js

```
let divs = document.querySelectorAll(".box")
div[0].innerText = "new unique value";
div[1].innerText = "new unique value";
div[2].innerText = "new unique value";
```

or using for loop.

```
let idx = 1;
for (div of divs) {
    div.innerText = `new unique value ${idx}`;
    idx ++;
}
```

Attributes

- getAttribute(attr) // to get the attribute value
- setAttribute(attr, value) // to set the attribute val th

Style

- node.style

Insert Element

- node.append(el) // adds at the end of node (inside)
- node.prepend(el) // adds at the start of node (inside)
- node.before(el) // adds before the node (outside)
- node.after(el) // adds after the node (outside)

Delete Element

- node.remove()

Q. Create a new button element. Give it a text "click me", background color of red and text color white.
Insert the button as the first element inside the body tag.

Sol

```
let newBtn = document.createElement("button");
newBtn.innerText = "Click me"
newBtn.style.color = "white";
newBtn.style.backgroundColor = "red";
document.querySelector("body").prepend(newBtn);
```

Q. Create a <p> tag in html, give it a class & some styling. Now create a new class in css and try to append this class to the <p> element. Did you notice, how you overwrite the class name when you add a new one? Solve this problem using classList

Sol

HTML

```
<body>
  <p class = "content"> I am paragraph </p>
</body>
```

CSS

```
.content {
  color = 'red';
}
```

```
.newClass {
  background-color : ~~red~~ green;
}
```

JS

```
let para = document.querySelector("p");

console.
  para.getAttribute("class");
o/p: 'content'

para.setAttribute("class", "newClass");
o/p: undefined

para.classList.add("newClass")
para.classList
para.classList.remove("newClass");
~~para.classli~~
```

Events in JS

→ The change in the state of an object is known as an Event

→ Event are fired to notify code of "interesting changes" that may affect code execution.

- Mouse events (click, doubleclick etc..)
- Keyboard events (keypress, keyup, keydown)
- Form events (submit etc..)
- Print events & many more

Event object-

It is a special object that has details about the event.

All event handlers have access to ~~the~~ Event object's properties and methods.

```
node.event = (e) => {
    // handle here
}
```

e.target, e.type, e.clientX, e.clientY

Best way to handle event is Event ~~Listeners~~ Listeners

node.addEventListener(event, callback)

node.removeEventListener(event, callback)

Note: the call back reference should be same to remove.

Q. Create a toggle button that change the screen to dark-mode when clicked & light mode when clicked again.

```
<body>
    <button id="mode"> change mode </button>
</body>
```

or CSS
.dark {
background-color: black;
}

JS change document class
body.classList.add("dark");

JS

```
let modeBtn = document.querySelector("#mode");
let currMode = "light"

modeBtn.addEventListener("click", () => {
    if (currMode === "light") {
        currMode = "dark";
        document.querySelector("body").style.backgroundColor = "black";
    } else {
        currMode = "light";
        document.querySelector("body").style.backgroundColor = "white";
    }
    console.log(currMode);
});
```

0 Classes & Objects

→ Prototypes in JS
→ A javascript object is an entity having state and behaviour (properties and method).
→ JS object have a special property called prototype.
→ we can set prototype using --proto--

Note: If object & prototype have same method, object's method will be used

Eg. const student = {
fullname: "shradha khapra",
marks : 94.4,
printMarks: function () {
console.log ("marks =", this.marks);
};

Classes in JS

→ Class is a program-code template for creating objects
→ Those object will have same state (variable) & some behaviours (functions) inside it

Syntax:
```
class MyClass {
  constructor() {...}
  myMethod() {...}
}
let myObject = new MyClass();
```

Constructor() method is:
- automatically invoked by new
- initializes object

Eg.
```
class ToyotaCar {
  constructor() {
    console.log ("creating new object");
  }
  start() {
    console.log ("start");
  }
```

```
    stop() {
        console.log("stop");
    }
    setBrand(brand) {
        this.brand = brand;
    }
}
let fortuner = new Toyotacar();
fortuner.setBrand = ("fortuner");
let lexus = new Toyotacar();
lexus.setBrand = ("lexus");
```

Inheritance in JS

→ Inheritance is passing down properties & methods from parent class to child class.

→ Syntax:

```
class Parent {
}

class child extends Parent {
}
```

Note: If child & Parent have same method, child's method will be used (Method overriding).

Eg.
```
class Person{
    eat(){
        console.log("eat");
    }
    sleep(){
        console.log("sleep");
    }
    work{
        console.log("do nothing");
    }

class Engineer extends Person{
    work(){
        console.log("solve problems, build something");
    }
}

class Doctor extends person{
    work(){
        console.log("treat patients");
    }

let engobj = new Engineer();
```

super keyword

→ The super keyword is used to call the constructor of its parent class to access the parent's properties and methods.

Eg.
```
class Person {
    constructor () {
        this.species = "homo sapiens";
    }
    eat () {
        console.log ("eat");
    }
}
class Engineer extends Person {
    constructor (branch) {
        super (); // to invoke parent class constructor
        this.branch = branch;
    }
    work () {
        console.log ("solve problem, build something");
    }
}
let engObj = new Engineer ("chemical engg");
```

→ we can also pass if child class constructor want to pass some important information to parent class constructor so it can be done using super keyword.

Eg.
```
class person{
    constructor (name){
        this.name = name;
    }

    eat (){
        console.log("eat")
    }
}

class Engineer extends Person{
    constructor (name) {
        super(name);
    }
    work(){
        console.log("solve problem");
    }
}

let obj = new Engineer("kamlesh");
```

Note: (imp)
whenever in child class if we create constructor then we need to call parent constructor using super keyword.

Q You are creating a website for your collage. Create a class user with 2 properties, name & email. It also has a method called viewData() that allows user to view website data. Create a new class Admin which inherits from User. Add a new method editData to Admin that allows it to edit website data

Soln

```
let DATA = "secret information";
class User {
    constructor (name, email) {
        this.name = name;
        this.email = email;
    }
    viewData () {
        console.log ("data", DATA);
    }
class Admin extends User {
    constructor (name, email) {
        super (name, email);
    }
    editData () {
        DATA = "some new value"
    }
}
let student1 = new User ("kamlesh", "kam@gmail.com");
let student2 = new User ("Dev", "dev@gmail.com");
```

- **Error Handling**

try-catch blocks.

Syntax:
```
try {
    ... normal code
}
code (err) { // err is error object
    ... handling error
}
```

Eg.
```
let a = 5;
let b = 10;
console.log ("a = ", a);
console.log ("b = ", b);
try {
    console.log ("a+b = ", a+c);
}
catch (err) {
    console.log (err);
}
console.log ("a+b = ", a+b);
```

- **Sync in JS**

→ Synchronous
↳ means the code run in a particular sequence

of instructions given in the program. Each instruction wait for the previous intruction to complete its execution.

→ Asynchronous

↳ Due to synchronous programming, ~~something~~ sometimes imp instructions get blocked due to some previous instruction, which causes a delay in the UI. Asynchronous code execution allows to execute next instructions immediately and doesn't block the flow.

- Callbacks

→ A callback is a function passed as an argument to another function.

Eg.
```
function sum(a,b){
    console.log(a+b);
}
function calculator(a,b, sum(callback){
    sumCallback(a,b);
}
calculator(1,2, sum);
// we can also do it using arrow function
// calculator(1,2,(a,b)) => {
    console.log(a+b);
};
```

Eg. const hello = () => {
    console.log("hello");
};
setTimeout(hello, 3000); // give output after 3sec.

- Callback Hell → (Problem in JS)

→ Callback Hell : Nested callbacks stacked below one another forming a pyramid structure. (Pyramid of Doom)

→ This style of programming becomes difficult to understand & ~~image~~ manage.

```
function getData(dataId, getNextData){
    setTimeout(() => {
        console.log("data", dataId);
        if(getNextData){
            getNextData();
        }
    }, 2000);
}
// callback hell.
getData(1, () => {
    getData(2, () => {
        getData(3, () => {
            getData(4);
        });
    });
});
```

(after every 2 sec.)

o/p: data 1
data 2
data 3
data 4

Promises

pending state | fulfilled state [resolved] | rejected state ← Eg. Amazon order

Promises

→ Promises is for "eventual" completion of task.
→ It is an object in JS.
→ It is is a solution to callback hell.

Syntax:

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

function with 2 handlers

Note: resolve and reject are callbacks provided by JS.

Eg.

```
let promise = new Promise((resolve, reject) => {
    console.log("I am a promise");
    resolve(123);
    // resolve("success");
    // reject("some error occured")
});
```

→ A Javascript Promise object can be:
- Pending : the result is undefined
- ~~Result~~ Resolved : the result is a value (fulfilled) resolve(result)
- Rejected : the result is an error object reject(error)

→ In general programming we do not need to create promise object if we request data from api then that api ~~send us~~ return a promise and then we ~~deals~~ handle it

```
function getData (dataId, getNextData) {
    return new Promise ((resolve, reject) => {
        setTimeout (() => {
            //console.log ("data", dataId);
            //resolve ("succes");
            reject ("error");
            if (getNextData) {
                getNextData();
            }
        }, 5000);
    });
}
```

→ .then() & .catch()

```
promise.then ((res) => {...})
promise.catch ((err)) => {...})
```

Eg:
```
const getPromise = () => {
    return new Promise ((resolve, reject) => {
        console.log ("I am promise");
        resolve ("success");
    }); //reject ("error")
};
```

```
let promise = getPromise();
promise.then((res) => {
        console.log("promise fulfilled", res);
});

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

→ Promise chaining

Eg.1:

```
function asyncFunc1() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            console.log("data 1");
            resolve("success");
        }, 4000);
    });
}

function asyncFunc2() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            console.log("data 2");
            resolve("success");
        }, 4000);
    });
}
```

```
console.log("fetching data1....");
let p1 = asyncFunc1();
p1.then((res) => {
    console.log("fetching data2....");
    let p2 = asyncFunc2();
    p2.then((res) => { });
});
```

O/p: fetching data1....
data1 // after 4 sec it will give o/p.
fetching data2...
data2 // after 4 sec it will give o/p.

Eg2:

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

// Promise Chain
getData(1).then((res) => {
    console.log(res);
    return
```

```
getData(1)
    .then((res) => {
        return getData(2);
    })
    .then((res) => {
        return getData(3);
    })
    .then((res) => {
        return getData(4)
        console.log(res);
    });
```

- Async-Await

→ async function always return a promise.

Syntax : async function myFunc() {...}

→ await pauses the execution of its surrounding async function until the promise is settled.

Eg

```
function api() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            console.log("weather data");
            resolve(200);
        }, 2000);
    });
}
```

```
async function getWeatherData(){
        await api();
}
```

Eg.
```
funtion getData (dataId){
    return new Promise ((resolve, reject) =>{
        setTimeout (()=>{
            console.log ("data , dataId");
            resolve ("succes");
        },2000);
    });
}

// Async- Await.
async function getAllData() {
    console.log ("getting data 1 ....");
    await getData(1);
    console.log ("getting data2 ....");
    await getData(2);
    console.log ("getting data3 ....");
    await getData(3);
```

- IIFE : Immediately Invoked function Expression
  → IIFE is a function that is called immediately as soon as it is defined.

Syntax:

```
(function () {
    // ...
})();

(() => {
    // ...
})();

(async() => {
    // ...
})();
```

- **Fetch API**

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

```
let promise = fetch(url, [options])
```

NOTE:
→ If we are not giving any option in a fetch method. then it create a particular type of request called GET request

Eg. const URL = "https://cat-fact.herokuapp.com/facts";
const getfacts = async () => {
console.log("getting data.....");
let response = await fetch(URL);
console.log(response); // JSON format
let data = await response.json(); } give usable data
console.log(data);
}; //console.log(data[0].text);

→ AJAX is Asynchronous JS & XML
→ JSON is Javascript object notation
→ json() method : return a second promise that resolves with the result of parsing the response body text as JSON. (Input is JSON, output is JS object).

- HTTP Request & Response.

→ HTTP Verb → (GET, HEAD, PUT, POST, DELETE, CONNECT, OPTION etc.)
→ HTTP response status code
Information responses (100-199)
Successful responses (200-299)
Redirection responses (300-399)
Client error responses (400-499)
Server error responses (500-599)

NOTE:
→ HTTP response header also contain details about the responses, such as content type, HTTP status code etc..