JavaScript

"JavaScript isn't just a language; it's the engine of the modern web, powering everything from simple scripts to complex applications."

- By Swaraj Phalke

www.linkedin.com/in/swaraj-phalke-b5539b220

JavaScript

How the web works?

Think of the web like an online food delivery system: - client (Browser) = You ordering food
- server = Restuarant
- Internet = Delivery guy

- Steps → 1] You enter website URL (e.g, www.amazon.in) in the browser.
  2] The browser sends a request to the ~~window~~ website server using HTTP/HTTPS.
  3] The server finds the required files (HTML, CSS, JS) and sends them back.
  4] The browser reads these files and renders the webpage you see (This all happens in a matter of milliseconds).

Purpose of a Scripting language.

A scripting language is a lightweight programming language used to automate tasks and control other software (like browsers or servers).

- Ex - On a e-commerce site, when you click "Add"/ "Add to wishlist", JS updates it in real time - this is scripting in action.
- Why scripting languages like JS are used:
  - To make webpage interactive (buttons, forms)
  - To change content dynamically (load more posts without refreshing)
  - To process data (validating forms, storing user preferences)
  - To connect to API's and fetch data from servers.

History of Javascript	
Year	- Event
1995	created by Brendan Eich at netscape in just 10 days.
1996	microsoft introduced a similar version called Jscript.
1997	standardized as ECMAScript
2009	Node.js was released - allowed JS to run on servers.
2015	Major updates ES6 (let/const, arrow fun, promises, etc).
Present	used for frontend, backend, games, mobile apps.

What is Javascript?

Javascript is powerful, high level scripting language used mainly for making web pages interactive.

- features :
  - Runs directly in the browser
  - Adds dynamic behaviour like popups, animations, data fetching.
  - Works with HTML & CSS to create full web experience.
  - Also works on the server using node.js.
- Ex - on YouTube, when you click the play btn, switch between dark and light mode, or recieve video suggestions - all of that is done using JS.

Advantages of Javascript

Feature	Explanation
1] fast execution	JS runs in browser, so its quick and doesn't always need to contact server.
2] Easy to learn.	Clean, simple syntax & beginner-friendly.
3] Works everywhere	Supported by all modern browsers & devices.
4] Interactive websites	Enables UI/UX features like sliders, animations, drag and drop.
5] Used for full stack	works on both front end (React) & backend (Node.js)
6] Dynamic content	update page content without reloading.
7] Large Ecosystem	Tons of libraries, frameworks, and tools.
8] Package managers & modules	Use npm to install and reuse thousands of code packages.

How JavaScript works ?

Everything in JavaScript happens inside an "execution context".

Thread of execution

variable environment → | memory component | code component ←

key : value	
a : 10	
fn : {...}	

contains variables & funⁿ as key value pair. →

← is place where whole JS code is executed.

Execution Context

- "JavaScript is a synchronous single-threaded language".

1] Variables in JavaScript

Variables store data, you can think of variables like containers.

1] var - old, function-scoped
2] let - block-scoped, can be updated
3] const - block-scoped, cannot be updated

- Ex -
```
let name = "John";
const age = " 26;
var country = "India";
```

2] Datatypes in JavaScript

JS has primitive and non-primitive datatypes.

- Primitive Datatypes
  1] String - "hello"
  2] Number - 10, 25, 5.5
  3] Boolean → true, false

4] Operators in Javascript

1] Arithmetic Operators
-used for basic math operations.

a] + (Addition)
5+2 = 7

b] - (substraction)
5-2 = 3

c] * (Multiplication)
5*2 = 10

d] / (Division)
5/2 = 2.5

e] % Modulus (Remainder)
5%2 = 1

f] ++ (increment)
a++ = a+1

g] -- (decrement)
a-- = a-1

2] Comparison Operators
-used for comparing two values.

a] == (equal to) [loose]
5 == '5' → true

b] === (Equal value & type) [strict]
5 === '5' → false

c] != (Not equal) [loose]
5 != '5' → false

d] !== (not equal) [strict]
5 !== '5' → true

e] > (Greater than)
5>3 → true

f] < (less than)
3<5 → true

M T W T F S S
Page No.:
Date:
YOUVA

4] Nested if... else statement

- if inside another if block, useful for multiple-level checks.

• Ex -
```
let username = "admin"
let password = "1234"
if (username === "admin") {
    if (password === "1234") {
        console.log("Login successful");
    } else {
        console.log("wrong password");
    }
} else {
    console.log("Invalid user");
}
```

5] switch statement

- used for multiple exact matches - cleaner than many else if statements.

• Ex -
```
let day = 3;
switch (day) {
    case 1: console.log("Monday");
            break;
    case 2: console.log("Tuesday");
            break;
    case 3: console.log("Wednesday");
            break;
    default: console.log("Another day");
}
```

output → wednesday.

4] null - intentionally empty
5] undefined - declared but not assigned
6] BigInt - large numbers
7] Symbol - unique identifiers

• Non-Primitive
1] object
2] Array
3] function

• Ex - let str = "hello";
let num = 100;
let isOnline = true;

3] Numbers in Javascript
- Javascript has only one number type : floating-point numbers (both integers and decimals).
- supports arithmetics : +, -, *, /, %, **

• Ex - let a = 5;
let b = 2.5;
let c = a+b;

4] Operators in Javascript
1] Arithmetic = +, -, *, /, %,
2] comparison = ==, ===, != , !==, >, <, >=, <=
3] Logical = &&, ||, !
4] Bitwise = &, |, ^, ~, <<, >>, >>>
5] Assignment = =, +=, -=, *=, /=
6] unary = ++, --
7] Ternary = condition ? value1 : value 2

5] NaN in Javascript
- NaN stands for "Not a Number". Its value that JS returns when a number operation fails.
• Ex - let result = "abc"/ 3; NaN
isNan (result); true

M T W T F S S
Page No.:
Date:
YOUVA

6] Ternary operator
- shorthand for if... else
- condition ? value1 : value2

6] strings in Java script
- A string is sequence of characters used to represent text.

• Ex- let greet = "Hello";
let name = 'Alice';
let message = `Hi, ${name}`;
console.log (message); // Hi Alice

• methods
1] .length
2] .toUpperCase()
3] .toLowerCase()
4] .includes() "hello".includes("ell") → true
5] .slice() "Javascript".slice(4,10) → "script"
6] .replace() "Alice".replace("John") → John
7] .trim() " hello ".trim() → "hello"
8] charAt() "hello".charAt(1) → "e"
9] indexOf() "hello".indexOf("l") → 2
10] lastIndexOf() "hello".lastIndexOf("l") → 3
11] split() "a,b,c".split(",") → ["a","b","c"]

7] null vs undefined

• null	• undefined
meaning - no value	meaning - not assigned value
type - object	Type - undefined
set by - programmer	Set by - Javascript (default)

• Ex - let x = null; x is empty
let y; y is undefined

g] >= (Greater than equal to)
5>=3 → true

h] <= (less than equal to)
4<=3 → false

3] Logical operators
- used to combine multiple conditions
a] && (Logical AND)
true && false → false
b] || (Logical OR)
true || false → true
c] ! (Logical NOT)
!true → false

4] Bitwise Operators
- works on binary representations.
a] & AND
b] | OR
c] ^ XOR
d] ~ NOT
e] << left shift
f] >> right shift
g] >>> zero-fill right shift

5] Assignment operators
- Assign values to variables.
a] = (assign) x=5
b] += (add & assign) x+=3
c] -= (substract & assign) x-=2
d] *= (multiply & assign) x*=2
e] /= (divide & assign) x/=2

8] Conditional statements in Javascript

1] If statement

- conditional statements are used to make decisions in your code.

• Ex -
```
let age = 20;
if (age >= 18) {
    console.log("You are adult");
}
```

2] else if statement

- checks another condition if previous if was false.

• Ex -
```
let score = 75;
if (score >= 90) {
    console.log("A grade");
} else if (score >= 70) {
    console.log("B grade");
} else {
    console.log("C grade");
}
```

3] else statement

- Executes when all above conditions are false.

• Ex -
```
let isRaining = false;
if (isRaining) {
    console.log("Take umbrella");
} else {
    console.log("Enjoy sunshine");
}
```

9] Truthy and falsy values:
- Javascript treats some values as false in condition these are called falsy values. All other are truthy.

1] Truthy values
- "hello" (non-empty string)
- 42 (non-zero number)
- [], {} (empty array/object)
- true

2] Falsy values
- false
- 0
- "" (empty string)
- null
- undefined
- NaN

10] Arrays in Javascript
- An array is a data structure used to store multiple values in a single variable.
- Ex - let fruit = ['Apple', 'Banana', 'Mango'];

1] Arrays are mutable
- You can change elements in array:
- Ex - let colors = ["Red", "Green", "Blue"];
  color[1] = "Yellow"; "Green" to "yellow"
  console.log(colors);
  → ["Red", "Yellow", "Blue"]

2] Array References
- Arrays are reference types. When you assign an array to another variable, both point to the same memory.

- Ex- let a = [1, 2, 3];
  let b = a;
  b[0] = 100;
  console.log(a); [100, 2, 3]

3] Constant Array
- You can change items inside a const array, but you can't ressign the whole array.
- Ex- const fruits = ["Apple", "Banana"];
  fruits [0] = "kiwi"; allowed
  fruits.push ("orange"); allowed
  // fruits. ["New", "Array"]; not allowed

4] Array common methods.
1] Push() - Add item to end
arr.push(4) - [1, 2, 3, 4]

2] pop() - remove last item
arr.pop() - [1, 2]

3] shift() - remove first item
arr.shift() - [2, 3]

4] unshift() - Add item at start
arr.unshift(0) - [0, 1, 2, 3]

5] length - number of items
arr.length - 3

6] includes() - checks if item exists
arr.includes(2) - true.

7] join() - convert array to string
arr.join("-") - "1-2-3"

8] slice - copy part of array (doesn't change original)
arr.slice(1,3) - [2,3]

9] splice - Add/ remove items (changes original)
arr.splice(1,1) - removes 1 item at index 1.

10] reverse() - Reverse the array
arr.reverse() - [3,2,1]

11] sort() - sorts the array
arr.sort() - alphabetical (by default).

11] Loops in Javascript
- loops are used to repeat a block of code multiple times, especially when working with arrays or repeating logic.

1] for loop
- used when you know how many times to loop.
• Ex -
```
for (let i = 1; i <= 5; i++) {
    console.log("count :", i);
}
```

2] while loop
- used when you don't know the number of repetitions in advance.
• Ex -
```
let i = 1;
while (i<=3) {
    console.log("i is" + i);
    i++;
}
```

3] do... while loop
- Executes the code at least once, even if the condition is false.
- Ex- let i = 1;
```
do {
    console.log ("Doing :", i);
    i++;
} while (i<=2);
```

4] for...of loop (for arrays, string)
- used to loop through values of iterable objects.
- Ex- let fruits = ["Apple", "Banana", "Mango"].
```
for (let fruit of fruits) {
    console.log (fruits);
}
```

5] for...in loop (for objects)
- used to loop through keys/ properties of an object.
- Ex.- let person = {name : "Alice", age : 25};
```
for (let key in person) {
    console.log (key, ":", person [key]);
}
```

12] Javascript objects.
- JS has several built-in or native or predefined objects.
- These object are accessible anywhere in program
- These objects will work the same way in any browser running in any operation system.

1] Date object

- The Date object is used to work with dates & times.
- Date objects are created with new Date().
- There are two ways of instantiating a date:
- var d = new Date(); var d = new Date(year, month, day, hours, minutes, seconds, milliseconds)

• Methods

1] getDate() - Returns the day of month from (1-31)
2] getDay() - Returns the day of week from (0-6).
3] getFullYear() - Returns the year (four digits).
4] getHours() - Returns the hour (four 0-23)
5] getMilliseconds() - Returns the milliseconds (from 0-999)
6] getMinutes() - Returns the minutes (from 0-59)
7] getMonth() - Returns the month (from 0-11)
8] getseconds() - Returns the seconds (from 0-59)

Ex-
```
var d = newDate();
var dd = d.getDate();
console.log("current date :" + dd);
```

• Methods

1] setDate() - sets the day of the month of a date object.
2] setFullYear() - sets the year (four digits) of date object.
3] setHours() - sets the hour of a date object.
4] setMilliseconds() - sets the milliseconds of a date object.

4] Math.max() - returns largest number
Ex- Math.max (10,5,8) //10

5] Math.min() - returns smallest number
Ex- Math.min (10,5,8) //5

6] Math.random()
- returns random number betn 0 and 1.
Ex- Math.random(); //0.55721

7] Math.pow (x,y)
- returns x raised to power of y.
Ex. Math.pow(2,3) //8

8] Math.sqrt ()
- returns the square root of number.
Ex- Math.sqrt(25); //5

9] Math.abs ()
- returns the absolute value (removes negative sign)
Ex- Math.abs (-5); //5

- Generating random number.

```
let randomNum = Math.floor(Math.random()
                *10) +1;
console.log (randomNum);
```

13] functions in Javascript.

1] functions in Javascript
- functions are reuseable blocks of code used to perform tasks.

5] setMinutes() - sets the milliseconds of a date object.

6] setMonth() - sets the month of a date object.

7] setSeconds() - sets the seconds of a date object.

Ex - var d = new Date();
d.setDate(31);
console.log(d);

2] Number object

3] String object

4] Array object

5] Math object

- The math object allows you to perform mathematical tasks.
- PI - returns PI (Approx 3.14)

1] Math.round()
- rounds a number to the nearest integer.
Ex - Math.round(4.4) //4
Math.round(4.6) //5

2] Math.ceil()
- rounds upward to the nearest integer.
Ex - Math.ceil(4.1) //5
Math.ceil(4.9) //5

3] Math.floor() - rounds downward to nearest integer. Ex - Math.floor(4.9) //4
Math.floor(4.1) //4

Ex- function greet () {
console.log ("Hello")
}
greet () ; // calling the function

2] Parameterless function
- A function with no parameters.
Ex- function sayHello () {
alert ("Hello, user")
}

3] Parameterized function.
- Takes arguments when called.
Ex- function greetUser (name) {
alert ("Hello," + name) ;
}
greetUser ("Alice") ;

4] Passing values via prompt
Eg- addition of two numbers using function

```
<head>
  <script>
    function Add (a,b) {
      var sum ;
      sum = a+b ;
      document.write ("Addition :" +sum);
    }
  </script>
</head>
```

1] this in Javascript
- this refers to the context in which a function is called.

• In global context:
Ex- console.log (this) refers to window (in browser)

• Inside an object method:
Ex- const person = {
name: "Alice",
greet: function () {
console.log ("Hi" + this.name); 'this' refers to person
}
};
person.greet();

2] try & catch in Javascript
- used for error handling without stopping the script.
Ex- try {
let x = y + 10;
} catch (error) {
console.log ("An error occurred", error.message);
}

3] Arrow functions
• syntax:
Ex- const greet = () => {
console.log ("Hello");
}

```
<body>
  <script>
      var n1, n2;
      n1 = parseInt (prompt ("Enter 1st no"));
      n2 = parseInt (prompt ("Enter 2nd no"));
      Add (n1, n2);
  </script>
</body>
```

5] function expressions

- functions stored in variables.

Ex-
```
const add = function (a,b) {
    return a+b;
}
console.log (add (2,5));
```

6] Higher order functions.

- functions that take other functions as arguments or return functions.

Ex-
```
function greet (name, callback) {
    console.log ("Hello " + name);
    callback ();
}
function aftergreet () {
    console.log ("Welcome to JS!");
}
greet ("Alice", aftergreet);
```

7] JavaScript DOM.

- DOM (Document object Model) is programming interface that allows Javascript to interact with and manipulate HTML and CSS.

• with parameters.

Ex- const add = (a,b) => {
    return a+b;
}

4] Implicit Return in Arrow functions.

- when there is only one expression, no need for return or { }.

Ex- const square = (n) => n*n;
    console.log (square (4));

5] setTimeout.

- Runs code after a delay (once).

Ex- setTimeout (() => {
        console.log ("Runs after 2 seconds");
    }, 2000);

6] setInterval

- Runs code repeatedly after a delay.

Ex- setInterval (() => {
        console.log ("Runs every 3 seconds");
    }, 3000);

7] this with arrow function

- Arrow functions do not have their own this. They use the this value of their enclosing scope.

Ex- const user = {
        name: "Bob",
        greet : function () {
            setTimeout (() => {
                console.log ("Hi" + this.name) works fine
            }, 1000);

```
    }
};
user.greet();
```

14] Array methods.

1] for Each()
- loops through each item in array.
- Does not return anything.

Ex- let arr = [1,2,3,4,5].

```
arr.forEach((el) => {      //using arrow funⁿ
    console.log(el);
}
arr.forEach(function (el) {  //using normal funⁿ
    console.log(el);
}
```

output → 1, 2, 3, 4, 5.

2] Map()
- Transform each item and return a new array.
- Does not change the original array.

Ex- let num = [1, 2, 3, 4];

```
let double = num.map((el) => {
    return el *2;
}
```

output → 2, 4, 6, 8

3] filter()
- returns a new array containing items that match the condition.

Ex- let ages = [12, 18, 25, 15];

```
let adults = ages.filter((el) => {
    return ~~age~~ el >= 18;
}
```

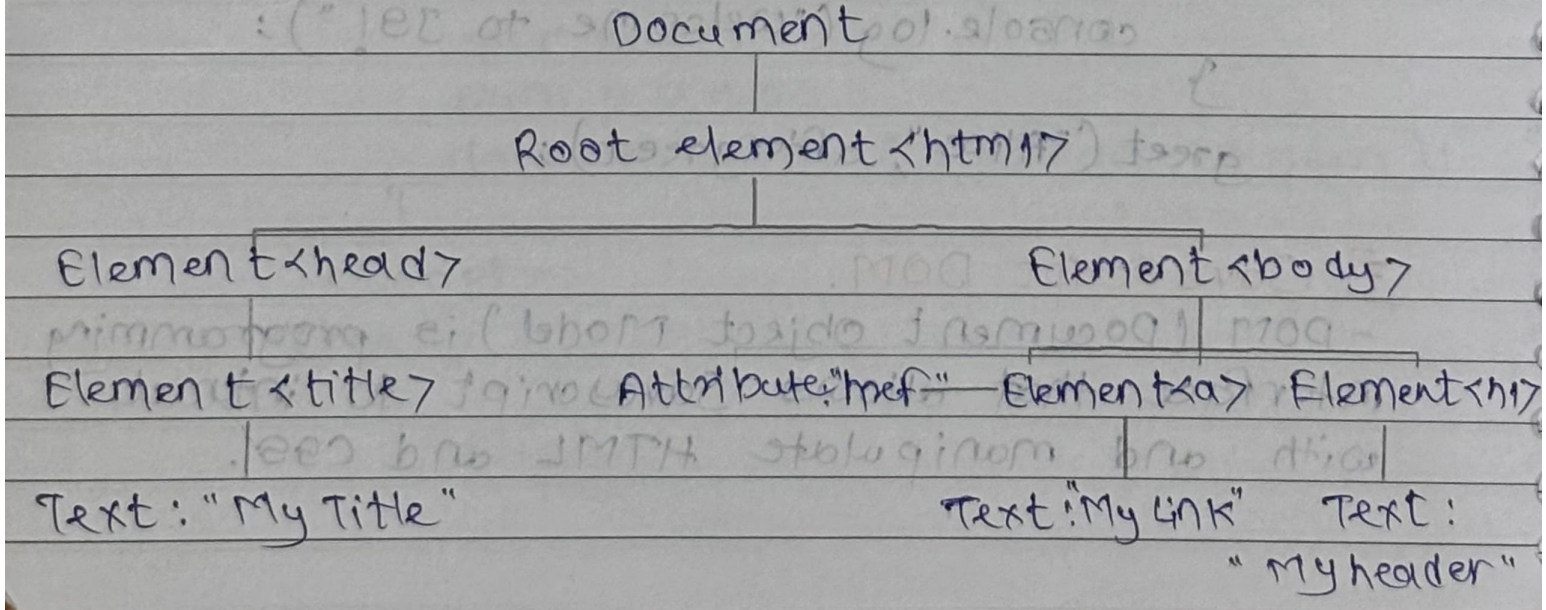
o/p → 18, 25.

- DOM is the object representation of HTML that allows JS to read and modify webpage content.

- Think of the DOM as a tree structure of your webpage where each element is node (like div, h1, button, etc.).

- Accessing Elements in DOM.
  1] getElementById ()
  - get a single element by its ID.
  2] getElementByClassName ()
  - returns a collection of elements with the given class name.
  3] getElementByTagName ()
  - get all elements with a specific tag name.
  4] querySelector ()
  - selects the first element matching a CSS selector.
  5] querySelectorAll ()
  - selects all elements matching a CSS selector.

- Modifying Content
  1] innerText
  - change the text
  2] innerHTML
  - change HTML inside the element.



M T W T F S S
Page No.:
Date:
YOUVA

* using promises.

• Ex- function saveto Db (data) {
return new Promise ((resolve, reject) => {
let internetspeed = Math. floor (Math. random
() * 10) + 1;
if (internetspeed > 4) {
resolve ("success : data saved");
} else {
failure ("failure: weak connection");
}
});
}
saveto Db ("Profound");

5] . then () and . catch () methods
- . then () handles resolved (success) results
- . catch () handles rejected (error) results.

• Ex- saveto Db ("apna college")
. then (() => {
console. log ("promise was resolved");
})
. catch (() => {
console. log ("promise was rejected");
});

→ this . then and . catch methods are performed on the above code while calling the "saveto Db" function.

```
.then ((result) => {
    console.log ("data 2 saved");
    console.log ("result of promise :", result);
})
.catch ((error) => {
    console.log ("promise was rejected");
    console.log ("error of promise :" + error);
});
```

- result and error in promises are used on the same function named as "saveToDb".

- Example - color example

```
h2 = document.querySelector ("h2");
function changeColor (color, delay) {
  return new promise ((resolve, reject) => {
    setTimeout (() => {
        h2.style.color = color;
        resolve ("color changed !");
    }, ~~1000~~ delay);
  });
}
changeColor ("red", 1000)
.then (() => {
    console.log ("red color completed");
    return changeColor ("orange", 1000);
}).
.then (() => {
    console.log ("orange color completed");
});
```

2] Await Keyword

- pauses the execution of its surrounding async function until the promise is settled (resolve or reject).
- used inside async function only to wait for promise to settled.

• Ex-
```
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();
  await getNum();
  getNum();
}
```

• Ex - changing colors.
```
async function demo(){
  await changeColor("red", 1000);
  await changeColor("orange", 1000);
  await changeColor("green", 1000);
  changeColor("blue", 1000);
}
```

- async-await keyword applyed on the color changing problem.

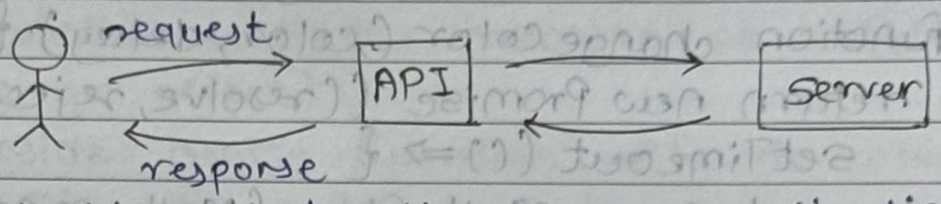
3] Handling Rejections

- Ex -

```
h1 = document.querySelector("h1");
function changeColor (color, delay) {
  return new Promise ((resolve, reject) => {
    setTimeout (() => {
      let num = Math.floor (Math.random () * 5) + 1;
      if (num > 3) {
        reject ("promise rejected");
      }
      h1.style.color = color;
      console.log (`color changed to ${color}`);
      resolve ("color changed");
    }, delay);
  });
}
async function demo () {
  try {
    await changeColor ("red", 1000);
    await changeColor ("orange", 1000);
    await changeColor ("green", 1000);
    await changeColor ("blue", 1000);
  } catch (err) {
    console.log ("error caught");
    console.log (err);
  }
}
```

- used try... catch with async/await to catch errors.

4] API (Application Programming Interface)



- Its a bridge that allows two application to talk to each other.

They allow developers to:
- get data from a server (weather, news, users)
- send data to server (register form, upload files)
- connect with third-party services (google maps)

5] What is JSON?

JSON - Javascript object Notation.
- It is the format in which APIs send and recieve data.

• Ex-
```
{
  "name": "Alice",
  "age": 25
}
```

- Looks like JS object, but it's in string form.

1] JSON.parse(data) Method
→ To parse a string data into a JS object

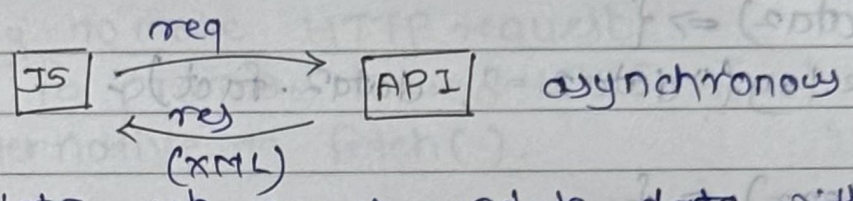
2] JSON.stringify(json) Method
→ To parse a JS object into JSON

6] Testing API requests.

• Tools
- Hoppscoth
- postman

7] What is AJAX?

AJAX - Asynchronous Javascript and XML.



- It lets web pages update data without refreshing.

8] HTTPS Verbs

- GET → Read data
- POST → Add new data
- PUT → update existing data
- DELETE → Remove data.

9] Status codes.

- Helps understand what went wrong or right.

1] 200 - OK
2] 201 - Created
3] 400 - Bad Request
4] 401 - Unauthorized
5] 404 - Not found
6] 505 - Server Error

10] first API request

• Ex- let url = "https://catfact.ninja/fact";

```
fetch(url)  → returns promise
.then((res) => {
    return res.json();  res.json() → returns promise
})
.then((data) => {
    console.log("data1 =", data.fact);
    return fetch(url);  → return promise
})
```

- finding maximum in array

Ex- let arr = [1, 4, 2, 5, 6, 7, 2 9, 2];

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

15] Promises and Callbacks

1] Callstack in Javascript

- The callstack is where JS keeps track of function calls.

Ex-
```
function hello() {
    console.log("inside hello fnx")
    console.log("hello")
}
function demo() {
    console.log("calling hello fnx");
    hello(); // pushed in the stack
}
console.log("calling demo fnx");
demo(); // callstack : demo() -> hello()
console.log("done, bye");
```

- It's LIFO - the last function pushed is the first one popped.

Callback → function inside function

M	T	W	T	F	S	S

Page No.:
Date:
YOUVA

2] JS is single threaded.

- Javascript executes one thing at a time → it's single threaded.
- But thanks to the browser's Web API, we can do async things like setTimeout, fetch etc.
- After completing the setTimeout the browser places the setTimeout in the JS callstack after completing its time.

• Ex.
```
setTimeout (() => {
    console.log ("hello world");
  }, 2000);
  console.log ("hello...");
```

3] Callback Hell (callback inside callback inside callback)

- callback hell happens when multiple asynchronous functions are nested inside each other, making the code.

Ex-
```
h1 = document.querySelector ("h1");
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);
      });
    });
  });
});
```

4] every()

- returns true if all elements match the condition. Else returns false.

Ex- let scores = [80, 90, 100];
let allPassed = scores.every((el) => {
return ~~score~~ el >= 35;
}
console.log(allPassed);
o/p → true.

5] some()

- Returns true if at least one element matches the condition, Else return false.

Ex- let scores = [45, 60, 90];
let hasFail = scores.some((el) => {
return el < 50;
}
console.log(hasFail);
o/p → true.

6] reduce()

- reduces the array to a single value.
- Takes two arguments: accumulator & current value / element.

Ex- let nums = [1, 2, 3, 4];
let total = nums.reduce((acc, curr) => {
return acc + curr;
}
console.log(total);
o/p → 10.

(0, 1) → 1
(1, 2) → 3
(3, 3) → 6
(6, 4) → 10

- Synchronous - executes tasks sequentially, where one task must finish before the next begins.

- Asynchronous - allows tasks to run concurrently, without waiting for previous tasks to complete.

16] Async-Await in Javascript

1] Async keyword

- creates an Async function
- returns an promise automatically
- async function allows you to write asynchronous code that looks like synchronous code.

- Ex -
```
async function greet () {
  // throw "404 page not found";
  return "hello";
}

greet ()
.then (() => {
  console.log ("promise was resolved");
})
.catch (() => {
  console.log ("promise was rejected");
});
```

- Ex -
```
let demo = async () => {
  return 5;
}
```

- after calling the demo function it will also return the promise.

4] setting up for promises

- The Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

• Ex-
```
function saveToDb (data, success, failure) {
    let internetSpeed = Math.floor(Math.random()*10)+1;
    if (internetspeed > 4) {
        success();
    } else {
        failure();
    }
}

saveToDb ("apna college",
    () => {
        console.log("success: data saved");
        saveToDb ("swaraj",
            () => {
                console.log("success 2: data saved");
            },
            () => {
                console.log("failure 2: weak connection");
            }
        );
    },
    () => {
        console.log("failure: weak connection");
    }
);
```

6] Promise chaining.

- you can return a value or another promise in .then().

• Ex-
```
saveToDb("apna college")
.then(() => {
    console.log("data1 saved");
    return saveToDb("hello world");
})
.then(() => {
    console.log("data2 saved");
    return saveto Db("swaraj");
})
.then(() => {
    console.log("data3 saved");
})
.catch(() => {
    console.log("promise was rejected");
});
```

- Promise chaining is performed on the previous "saveto Db" function.

7] Results and Errors in Promises.

- If resolved → goes to .then().
- If rejected or an error occurs → goes to .catch()

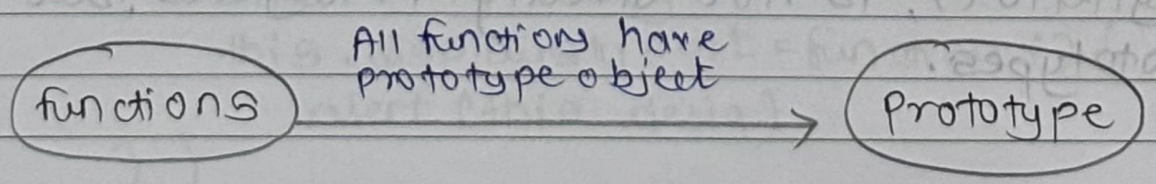
• Ex-
```
saveto Db("apna college")
.then((result) => {
    console.log("data 1 saved");
    console.log("result of promise :", result);
    return saveto Db("hello world");
});
```

- object created using literal.
  - object created using literal are constants.
- Ex - var student = {
  'rno': 10,
  'nm': 'purvesh',
  'details': function () {
  alert (this.rno+" "+this.nm)
  }
  }
  ~~console.log~~ (student. details ();

- object as function.
  - JS classes are declared using function keyword.
- Ex - function student (nm) {
  this.nm = nm;
  this. details = function () {
  alert (this.nm)
  }.
  }

  var s1 = new student ("swaraj").
  s1. detail ();

3] Prototypes.
- In Javascript, every function and object has a hidden property called Prototype.
  (accessed via .prototype in functions or __proto__ in objects).
- whenever we create a JS function, Javascript adds a prototype property to that function.



12 Axios

- library to make HTTP request.
- Axios is promise based HTTP client for Javascript an alternative to fetch().

- Use
  - simpler syntax than fetch
  - Automatically converts response to JSON
  - Better error handling
  - supports older browsers.
- installation
  - npm install axios
- Ex - HTML file →

```
<html>
 <head>
  </head>
  <body>
    <script src="https://cdn.jsdeliver.net/npm/axios
      @1.1.2/dist/axios.min.js"></script>.
    <h1> Get random cat facts </h1>
    <button> show new fact </button>
  <p id="result"></p>
  <script src="app.js"></script>
  </body>
 </html>
```

JS file →

```
let btn = document.querySelector("button");
btn.addEventListener("click", async () => {
   let fact = await getfacts();
   let p = document.querySelector("#result");
   p.innerText = fact;
});
```

- OOPs in Javascript
- Javascript is object based language.
- Everything is an object in Javascript.
- Javascript is template based not class based.
- There is no need to create class every time to get the object. But, we can directly create objects.

2] Javascript objects.
- An object is instance of class.
- An object is an entity that contains properties/states and methods/functions.
- Eg. car, fruit, person, bank account.
- The characteristics of an object are called properties.
- The actions of an object are called methods.

- object - An object is a single instance of class. When a class is defined, no memory is allocated but when it is initiantiated / created memory is allocated.
- One can create many objects from the same class type.

- Three way to create object.
- by creating instance of object directly (using new keyword).
- by object literal.
- by using an object constructor (using new keyword).

```
let url = "https://cat fact.ninja/fact";
async function getfacts () {
  try {
    let res = await axios.get (url);
    return res.data.fact;
  } catch (e) {
    console.log ("error -", e);
    return "No fact found";
  }
}
```

17.] Javascript classes and objects

1] Javascript class

- In Javascript, classes are special type of functions which contain properties & methods.
- objects are variables of type class.
- we can define the class just like function declarations and function expressions.
- A class keyword is used to declare a class with any particular name.

- class - classes are building block of OOP. It is a user defined datatype, which holds its own data member and member functions.
  - It can be accessed and used by creating an instance of that class. Data members are the data variables and member functions are the functions used to manipulate these variables together, these data members and member functions define the properties and behaviour of the object in a class.

5] call(), apply(), bind() - (built-in functions)

1] call() - takes arguments seperately
- calls a function with a given 'this' value and arguements seperately.
- call() method is predefined JS function.
- call() accepts arguement list
• syntax
- person.fullName.call(student)
- person.fullName.call(student, 100, "ABC")
• Ex -

```
const person = {
    fullName: function (city, country) {
        return this.firstName + " " + this.lastName
        + " from " + city + "," + country;
    }
}
const user = {
    firstName = "John",
    lastName = "Doe"
}
// Borrowing function
console.log (person.fullName.call (user, "Pune",
                                   "india"));
// John Doe from Pune, India
```

2] apply() - takes arguements in array
- same as call(), but arguments are passed as an array.
- apply methods is very handy if you want to use an array instead of an argument list.

```
.then ((res) => {
    return res.json() ; res.json() - returns promise
})
.then ((data2) => {
    console.log("data2 =", data2.fact) ;
})
.catch ((err) => {
    console.log ("Error -", err);
});
console.log ("i am happy");
```

- The above code runs asynchronously because we have used promises.

ii] using fetch() with async/await

- Ex -

```
let url = "https://catfact.ninja/fact";

async function getfacts() {
  try {
    let res = await fetch (url);
    let data = await res.json();
    console.log (data.fact);

    let res2 = await fetch(url);
    let data2 = await res2.json();
  } catch (e) {
      console.log ("error -", e);
  }
  console.log ("bye");
}
```

```
student.prototype = new person() //inheritance
student.prototype.constructor = student.

var std = new student ("Jay", "Kale", "Pune",
                        "SR")
std.display()
std.~~student~~ displaystudent()
alert (std.name)
  //we are defining person class is students
  prototype.
```

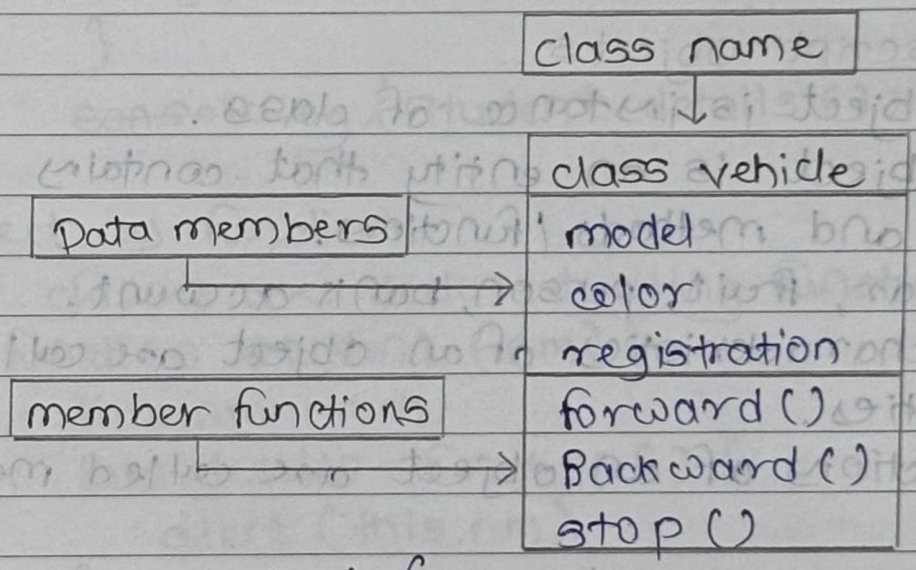
- Ex- Inheritance - extend keyword.

```
class Animal { // Parent class
  constructor (name) {
    this.name = name
  }
  speak () {
    console.log (`${this.name} make a sound.`)
  }
}

class Dog extends Animal { // child class
  bark () {
    console.log (`${this.name} barks!`);
  }
}

  // creating an object
const myDog = new Dog ("Tommy");
myDog.speak ();  // Tommy makes sound
myDog.bark ();   // Tommy barks!
```

- key features
1] constructor () - special method that runs when the object is created.
2] Methods - functions inside the class.
3] new keyword - used to create instances (object) of the class.
4] this keyword - refers to the current object.



- Ex -
```
class student {
    constructor(rno, name) { // initializing an object
        this.rno = rno;
        this.name = name;
    }
    detail () { // Declaring method
        document.write (this.rno+" "+this.name)
    }
}
// creating object
var std = new student (101, "ABC");
std.detail (); // calling method
```

- prototype is base class for all objects, All JS objects inherit properties and methods from a prototype, and it also helps to achieve inheritance in JS.

4] Inheritance
1] prototypes (old way - before ES6)
2] extends keyword (modern way - ES6 and later).

1] Prototypal Inheritance
- Inheritance is one of the important concept in OOP.
- In classical inheritance, derived class can use methods & properties of base class.
- In JS, inheritance is supported by using prototype object, it is called as 'Prototypal Inheritance' or 'Behaviour Delegation'.

```
Ex - function person(nm,add,ph){
        this.name=nm;
        this.address = add;
        this.phone=ph;
     } //setting common functions for all objects
     person.prototype.display=function(){
        alert(this.name+this.address+this.phone)
     }
     //Student class is defined as
     function student(nm,add,ph,desig){
        //it calls person contructor
        student.call(this,nm,add,ph);
        this.desig=desig;
        this.displaystudent=function(){
            alert(this.desig)
     }}
```

- syntax
  - person.fullName.apply (student)
  - person.fullName.apply (this, [100, "ABC"]).
- Ex -

```
console.log (person.fullName.apply (user,
                    ["Pune", "India"]).
//above code is in the call() method.
```

3] bind ()
- The bind function doesn't calls like call() and apply(), The bind() method creates a new function.
- bind function creates a copy with new value.
- We use bind() method primarily to call a fun^n with the this value set explicitly.
- syntax.
  - var displayStudent = person.display.bind (student).

```
displayStudent();  // it returns function.
```

- Ex -

```
const newfunction = person.fullName.bind
                    (user, "Pune", "India");
console.log (newfunction());
//returned a function.
```

- prototype can be accessible, modifiable, and it can add new variables and methods to the existing object and shares accross all the instances of its constructor function.
- syntax for adding new variable:

```
ClassName.prototype.variableName = value
```

- syntax for adding new method:

```
ClassName.prototype.functionname = function () {
    return;
}
```

• Ex 1 -

```
let animal = {
    eats: true
}
let rabbit = {
    jumps: true
}
rabbit.__proto__ = animal;
```

• Ex 2 -

```
function Person(name) {
    this.name = name
}
Person.prototype.sayHello = function () {
    console.log ("Hi, Im" + this.name);
}

let p1 = new Person ("swaraj").
p1.sayHello ();  // Hi iam swaraj
```

- sayHello(), is not directly in objects but in prototypes.