Part -2 JavaScript
JavaScript Errors
How to Handle JavaScript Errors

try block

- try ke andar aap woh code likhte ho jisme error aane ka chance ho sakta hai.
- JavaScript pehle try ka code chalata hai.
- Agar koi error nahi aata, toh catch skip ho jaata hai.

catch block

- Agar try block chalate time koi error aata hai, toh control seedha catch block mein chala jaata hai.
- catch ke andar aap bata sakte ho ki error aane par kya karna hai — jaise message dikhana, fallback value dena, etc.

```
try {
  Block of code to try
} catch(err) {
  Block of code to handle errors
}
```

Reference Errors

JavaScript mein ReferenceError tab aata hai jab aap kisi aise variable ka naam use karte ho jo exist hi nahi karta — yaani declare hi nahi hua.

- Agar aap kisi variable ko pehle declare nahi karte, aur directly use karne ki koshish karte ho → ReferenceError milta hai.
- JavaScript bolta hai: “Yeh variable mujhe pata hi nahi hai!”

Error Type	Example	Error
ReferenceError	fname = foo;	foo is not defined
ReferenceError	let x = y; let y = 5;	Cannot access y before initialization

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Errors</h1>
<h2>ReferenceError</h2>
<p id="demo"></p>
<script>
try {
  let x = y;
  let y = 5;
} catch(err) {
  let text = err.name + "<br>" + err.message;
  document.getElementById("demo").innerHTML = text;
}
</script>
</body>
</html>
```

JavaScript Type Errors

JavaScript mein TypeError tab hota hai jab:

1. Aap galat type ki value use kar dete ho,
   ya
2. Aap koi aisi operation ya method call karte ho jo us type ke liye valid hi nahi hai.

Error	Example	Error Message
TypeError	Anna(5);	Anna is not a function
Type Error	let num = 1; num.toUpperCase();	num.toUpperCase is not a function

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Errors</h1>
<h2>The TypeError</h2>
<p>You cannot convert a number to upper case:</p>
<p id="demo"></p>
<script>
```

```
let num = 1;
try {
  num.toUpperCase();
} catch(err) {
  let text = err.name + "<br>" + err.message;
  document.getElementById("demo").innerHTML = text;
}
</script>
</body>
</html>
```

JavaScript Range Errors

JavaScript mein RangeError tab hota hai jab aap koi aisi value use karte ho jo allowed range (limit) ke bahar ho.

Error Type	Example	Error Message
RangeError	new Array(-1);	Invalid array length
RangeError	num.toPrecision(500);	toPrecision() argument must be between 1 and 100

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Errors</h1>
<h2>RangeError</h2>
<p id="demo">
<script>
let num = 1;
try {
  num.toPrecision(500);
} catch(err) {
  let text = err.name + "<br>" + err.message;
  document.getElementById("demo").innerHTML = text;
}
</script>
</body>
```

```
</html>
```

JavaScript URI Errors

JavaScript mein URIError tab hota hai jab aap URL-related (URI) functions — jaise encodeURI(), decodeURI(), encodeURIComponent(), decodeURIComponent() mein galat ya illegal characters use kar dete ho.

- Jab aap kisi string ko encode/decode karte ho
- Aur usme aise characters hote hain jo URI ke rules ke hisaab se allowed nahi hote
  → Tab JavaScript URIError throw karta hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Errors</h1>
<h2>The URIError</h2>
<p>Some characters cannot be decoded with decodeURI():</p>
<p id="demo"></p>
<script>
try {
  decodeURI("%%%");
} catch(err) {
  let text = err.name + "<br>" + err.message;
  document.getElementById("demo").innerHTML = text;
}
</script>
</body>
</html>
```

JavaScript Syntax Errors

- Code likhne mein koi typo,
- Bracket ya quote missing,
- Statement incomplete,
- Ya invalid JavaScript keywords ka use…
  → In sab situations mein SyntaxError aata hai.

Error	Example	Error
SyntaxError	fname = "John);	Invalid or unexpected token)
SyntaxError	Math.round(4.6;	Missing) after argument list

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Errors</h1>
<h2>Syntax Errors</h2>
<p>A syntax error will stop your program. The execution not continue.</p>
<p id="demo"></p>
<script>
// This line will fail
let text = "John Doe);
// This line will not be executed
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

Syntax Errors are Not Catchable

Kyuki SyntaxError code chalne se pehle hi aata hai.
JavaScript interpreter code ko run karne se pehle parse karta hai — yani check karta hai ki grammar/syntax sahi hai ya nahi.
Agar syntax galat hai, toh code start hi nahi hota, isliye try...catch tak control pahuchta hi nahi.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Errors</h1>
<h2>Syntax Errors</h2>
<p>Syntax errors are not catchable by try...catch.</p>
<p id="demo"></p>
<script>
```

```
try {
  let x = Math.round(4.6;)
} catch(err) {
  let text = err.name + " " + err.description;
  document.getElementById("demo").innerHTML = text;
}
</script>
</body>
</html>
```

JavaScript engine script ko pehle PARSE karta hai

- Script run hone se pehle browser JavaScript ko padhta hai.
- Agar parsing ke time koi grammar/syntax galti milti hai → SyntaxError throw hota hai.

Syntax errors try...catch se pehle hi aajate hain

- Try...catch sirf runtime errors handle karta hai.
- Lekin SyntaxError runtime se pehle hi detect ho jaata hai.

Result: script run hi nahi hoti

- Kyuki parsing fail ho gayi, engine script ko execute karna hi band kar deta hai.

JavaScript Silent Errors

JavaScript kabhi-kabhi silent errors produce karta hai — matlab:

- Program me error hota hai
- Lekin code rukta nahi
- Browser koi error message show nahi karta
- Execution normal tarah continue rehti hai

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Silent Errors</h1>
<p>Silent errors will not stop your program.</p>
<p id="demo"></p>
<script>
const user = {};
let result = user.name;
document.getElementById("demo").innerHTML = result;
```

</script>
</body>
</html>

JavaScript Error Statements

The try Statement

JavaScript mein try statement ka use error handle karne ke liye hota hai — taaki agar code mein koi error aaye, toh poora program ruk na jaye.
try allow karta hai:

- Code ko safely test karna
- Agar error aaye toh usko catch karna
- Program ko smoothly chalna dena

```
try {
  // Code that may cause an error
} catch (error) {
  // Code to handle the error
}
```

The Catch Block

- catch block sirf tab execute hota hai jab try block ke andar koi runtime error aata hai.
- Agar try block error-free execute ho jaye → catch skip ho jaata hai.

Error object kya hai?

- catch block ko ek error object milta hai.
- Is object mein error ke baare mein details hoti hain, jaise:
- name → error ka type (TypeError, ReferenceError, etc.)
- message → error ka description

```
try {
  // Code that may cause an error
} catch (error) {
  // Code to handle the error
}
```

The Finally Block (Optional)

- finally block hamesha execute hota hai, chahe try me error aaye ya na aaye.
- Iska main use hai cleanup tasks ke liye, jaise:
- Files close karna
- Loader stop karna
- Database connections release karna

```
try {
  // Code that may cause an error
} catch (error) {
  // Code to handle the error
} finally {
  // Code that always runs, no matter what
}
```

JavaScript Throws Errors

Error hone par JavaScript kya karta hai?

- Jab code me koi error hota hai, JavaScript normally execution rok deta hai.
- Browser ya console me error message show hota hai.

Technical term:

- Is behavior ko kehte hain: JavaScript throws an exception
- Yani JavaScript “error throw kar deta hai”
- Exception = Error ka formal term

The throw Statement

- throw ka use custom error create karne ke liye hota hai.
- Matlab aap apni marzi ka error generate kar sakte ho.
- Ye error runtime me program ko rok sakta hai, aur try…catch me handle kiya ja sakta hai.

Input Validation

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Errors</h1>
<h2>JavaScript try catch</h2>
```

```
<p>Please input a number between 5 and 10:</p>

<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="p01"></p>
<script>
function myFunction() {
const message = document.getElementById("p01");
message.innerHTML = "";
let x = document.getElementById("demo").value;
try {
if(x.trim() == "")  throw "empty";
if(isNaN(x)) throw "not a number";
x = Number(x);
if(x < 5)  throw "too low";
if(x > 10)   throw "too high";
} catch(err) {
message.innerHTML = "Input is " + err;
}
}
</script>
</body>
</html>
```

The Error Object

- JavaScript ka built-in error object hota hai jo error ke baare mein information deta hai.
- Jab try block me error aata hai aur catch me handle karte hain, tab ye object accessible hota hai.

Property	Description
name	Error ka type set ya return karta hai. Example: ReferenceError, TypeError
message	Error ka description set ya return karta hai. Ye hamesha string hota hai.

Error Names

Error Name	Description
EvalError	Deprecated → pehle eval() se related errors dikhata tha. Ab mostly SyntaxError use hota hai.
RangeError	Jab koi number ya value allowed range ke bahar ho jaye.
ReferenceError	Jab aap undefined variable ya illegal reference use karte ho.
SyntaxError	Jab code ka grammar/syntax galat ho.
TypeError	Jab koi operation ya function wrong type pe use ho.
URIError	Jab illegal characters encode/decode kiye jaye URI functions me (encodeURI, decodeURI).

Non-Standard Properties and Methods

Property / Method	Description / Notes
arguments	Deprecated – purane code me use hota tha
caller	Deprecated – function caller track karne ke liye tha
columnNumber	Firefox only – error ka column number batata hai
description	Microsoft only – error ka description deta tha
displayName	Firefox only – error ka friendly name
fileName	Firefox only – error kis file me hua
lineNumber	Firefox only – error kis line me hua
number	Microsoft only – error ka numeric code

Property / Method	Description / Notes
stack	Firefox only – stack trace deta hai (error ka call path)
evalError()	Deprecated – purane eval() errors ke liye
internalError()	Firefox only – internal JS engine errors handle karta tha
toSource()	Non-standard – object ka source code string me return karta tha

Code Debugging

Errors in Programming

- Programming code me do main types of errors ho sakte hain:
- Syntax errors
- Code likhne ka grammar galat ho
- Example: missing parentheses, quotes, etc.

Logical errors

- Code correctly run hota hai, lekin expected result nahi deta
- Example: wrong formula, wrong condition

Difficulties with errors

- Kai baar errors diagnose karna mushkil hota hai.
- Kabhi kuch bhi nahi hota, na error message, na hint
- Programmer ko khud problem find karke fix karna padta hai

Debugging

- Debugging = Programming code me errors dhundna aur fix karna
- Tools: console logs, debuggers, breakpoints, etc.
- Goal: Program ko correct aur expected way se chalana

JavaScript Debuggers

- Debugging is not easy. But fortunately, all modern browsers have a built-in JavaScript debugger.
- Built-in debuggers can be turned on and off, forcing errors to be reported to the user.
- With a debugger, you can also set breakpoints (places where code execution can be stopped), and examine variables while the code is executing.
- Normally (otherwise follow the steps at the bottom of this page), you activate

debugging in your browser with the F12 key, and select "Console" in the debugger menu.

The console.log() Method

```
<!DOCTYPE html>
<html>
<body>
<h2>My First Web Page</h2>
<p>Activate debugging in your browser (Chrome, IE, Firefox) with F12, and select "Console" in the debugger menu.</p>
<script>
a = 5;
b = 6;
c = a + b;
console.log(c);
</script>
</body>
</html>
```

Setting Breakpoints

- Yeh explanation Hinglish me samjha sakte hain:
- Breakpoints set karna:
- Debugger window me aap apne JavaScript code me breakpoints set kar sakte ho.
- Jab code execution kisi breakpoint pe pohchta hai, to JavaScript execution ruk jaata hai, aur aap variables aur values ko check kar sakte ho.
- Values check karne ke baad, aap code execution ko resume kar sakte ho, usually play button press karke.
- Agar chaho, mai ye bhi bata sakta hoon ki breakpoints kaise set karte hain step by step browser me. Chahiye?

The debugger Keyword

- debugger keyword ka use karne se JavaScript execution ruk jaata hai, aur agar debugging function available ho to wo call ho jata hai.
- Iska kaam breakpoint set karne ke same hai.
- Agar debugging available nahi hai, to debugger statement ka koi effect nahi hota.
- Jab debugger on hota hai, to is code me teesi line execute hone se pehle hi execution ruk jaayega, aur aap variables check kar sakte ho.

```
<!DOCTYPE html>
```

```
<html>
<body>
<h2>JavaScript Debugger</h2>
<p id="demo"></p>
<p>With the debugger turned on, the code below should stop executing before it executes the third line.</p>
<script>
let x = 15 * 5;
debugger;
document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>
```

Major Browsers' Debugging Tools

Chrome:

1. Browser open karo.
2. Menu me jaake "More tools" select karo.
3. Tools me se "Developer tools" choose karo.
4. Finally, Console select karo.

Firefox:

1. Browser open karo.
2. Menu me jaake "Web Developer" select karo.
3. Phir "Web Console" choose karo.

Edge:

1. Browser open karo.
2. Menu me jaake "Developer Tools" select karo.
3. Phir Console choose karo.

Opera:

1. Browser open karo.
2. Menu me jaake "Developer" select karo.
3. Phir "Developer tools" choose karo.
4. Finally, Console select karo.

Safari:

1. Safari me jaake Preferences → Advanced open karo.
2. "Enable Show Develop menu in menu bar" check karo.
3. Jab menu me Develop option aa jaye, to usme jaake "Show Error Console" select karo.

JavaScript Events

HTML Events

HTML event kuch bhi ho sakta hai jo browser karta hai ya user karta hai.

Examples of HTML events:

- Ek HTML web page load ho gaya.
- Ek HTML input field me value change hui.
- Ek HTML button click hua.
- Jab events hote hain, to aap aksar chahte ho ki kuch action execute ho.
- Yahan JavaScript ka kaam hai: event detect hone par code run karna.
- Agar chaho, mai ek chhota sa example code bhi de sakta hoon jisme button click hone par JavaScript execute ho.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript HTML Events</h1>
<h2>The onclick Attribute</h2>
<p>Click the button to display the date.</p>
<button onclick="displayDate()">The time is?</button>
<script>
function displayDate() {
 document.getElementById("demo").innerHTML = Date();
}
</script>
<p id="demo"></p>
</body>
</html>
```

Common HTML Events

Event	Description
onchange	Jab koi HTML element change hota hai (jaise input field me value badalna)
onclick	Jab user HTML element pe click karta hai
onmouseover	Jab user mouse ko HTML element ke upar le jaata

Event	Description
	hai
onmouseout	Jab user mouse ko HTML element se hataata hai
onkeydown	Jab user keyboard ka key press karta hai
onload	Jab browser page fully load kar leta hai

JavaScript Event Handlers

- Event handlers ka use user input, user actions, aur browser actions ko handle aur verify karne ke liye kiya ja sakta hai:
- Jo cheezein har baar page load hone par karni chahiye
- Jo cheezein page close hone par karni chahiye
- Action jo tab perform karna ho jab user button click kare
- Content jo verify karna ho jab user data input kare
- Aur bhi bahut kuch…
- JavaScript ko events ke saath kaam karwane ke liye alag-alag methods use kiye ja sakte hain:
- HTML event attributes directly JavaScript code execute kar sakte hain
- HTML event attributes JavaScript functions ko call kar sakte hain
- Aap apne khud ke event handler functions HTML elements ko assign kar sakte hain
- Aap events ko send hone ya handle hone se prevent kar sakte hain.

JavaScript Scope

Scope = Visibility

Scope determines the **accessibility** (visibility) of variables.

JavaScript variables have 3 types of scope:

- Global scope
- Function scope
- Block scope

Global Scope

Jo variables globally declare kiye jaate hain (matlab kisi block ya function ke bahar) unka Global Scope hota hai.

- Global variables ko JavaScript program ke kahin se bhi access kiya ja sakta hai.
- Agar var, let, ya const se variable declare kiya gaya ho block ke bahar, toh ye sabka Global Scope hota hai.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Scope</h2>
<p>A GLOBAL variable can be accessed from any script or function.</p>
<p id="demo"></p>
<script>
let carName = "Volvo";
myFunction()
function myFunction() {
  document.getElementById("demo").innerHTML = "I can display " + carName;
}
</script>
</body>
</html>
```

Function Scope

Har JavaScript function ka apna scope hota hai.

- Jo variables function ke andar define kiye jaate hain, wo function ke bahar access nahi kiye ja sakte (visible nahi hote).
- Agar var, let, ya const se variable function ke andar declare kiya gaya ho, toh ye sab function scope follow karte hain.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Scope</h2>
<p><b>carName</b> is undefined outside myFunction():</p>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
myFunction();
function myFunction() {
```

```
let carName = "Volvo";
 document.getElementById("demo1").innerHTML = typeof carName + " " + carName;
}
document.getElementById("demo2").innerHTML = typeof carName;
</script>
</body>
</html>
```

Local Variables has Function Scope

- Ye variables sirf function ke andar hi access kiye ja sakte hain.
- Function ke bahar ke scripts ya functions inhe access nahi kar sakte.
- Agar bahar ya dusre functions mein same naam ke variables use kiye jaayein, wo problem nahi karte.
- Local variables tab create hote hain jab function start hota hai.
- Local variables delete ho jaate hain jab function complete ho jata hai.
- Function ke arguments (parameters) bhi local variables ki tarah behave karte hain function ke andar.

Automatically Global

- Agar aap kisi variable ko declare kiye bina value assign kar dete ho, toh wo variable automatically GLOBAL ban jaata hai.
- Matlab, function ke andar bhi value assign karne par wo variable global scope le lega.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Global Variables</h2>
<p>If you assign a value to a variable that has not been declared, it will automatically
become a GLOBAL variable:</p>
<p id="demo"></p>
<script>
myFunction();
// code here can use carName as a global variable
document.getElementById("demo").innerHTML = "I can display " + carName;

function myFunction() {
  carName = "Volvo";
}
</script>
</body>
</html>
```

Strict Mode

- JavaScript ka Strict Mode ek special mode hai jo errors ko zyada strictly check karta hai.
- Isse aap accidental globals aur kuch unsafe actions ko avoid kar sakte ho.
- Strict mode enable karne ke liye code ke shuru me ya function ke andar likhte hain:

Global Variables in HTML

- JavaScript me global scope ka matlab hai poora JavaScript environment.
- HTML me global scope ka matlab hai window object.
- Agar aap var keyword se global variable define karte ho, toh wo variable window object` ka property ban jaata hai.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Global Variables</h2
<p>In HTML, global variables defined with <b>let</b>, will not become window variables.</p>
<p id="demo"></p>
<script>
let carName = "Volvo";
// code here can not use window.carName
document.getElementById("demo").innerHTML = "I can not display " + window.carName;
</script>
</body>
</html>
```

The Lifetime of JavaScript Variables

- JavaScript me variable ki lifetime tab start hoti hai jab wo declare kiya jaata hai.
- Function ke andar ke (local) variables tab delete ho jaate hain jab function complete ho jaata hai.
- Web browser me, global variables tab delete ho jaate hain jab aap browser window ya tab close kar dete ho.

JavaScript Hoisting

- Hoisting ka matlab hai ki JavaScript variables aur functions ko unke declaration ke upar “lift” kar deta hai, chahe aap unhe code me baad me likho.
- Iska effect ye hota hai ki aap variable ya function ko declare karne se pehle bhi use kar sakte ho (lekin behavior var, let, aur const me thoda alag hota hai).

```
<!DOCTYPE html>
<html>
```

```
<body>
<p id="demo"></p>
<script>
x = 5; // Assign 5 to x
elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x;          // Display x in the element
var x; // Declare x
</script>
</body>
</html>
```

The let and const Keywords

- let aur const se defined variables block ke top tak hoist hote hain, lekin initialize nahi hote.
- Matlab: block code variable ke existence ko jaanta hai, lekin use tab tak use nahi kar sakta jab tak declare na ho.
- Agar aap let variable ko declare se pehle use karte ho, toh ReferenceError aayega.
- Ye variable “Temporal Dead Zone (TDZ)” me hota hai, jo block ke start se lekar declaration tak chalti hai.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Hoisting</h2>
<p>With <b>const</b>, you cannot use a variable before it is declared.</p>
<p>Try to remove the //.</p>
<p id="demo"></p>
<script>
carName = "Volvo";
//const carName;
document.getElementById("demo").innerHTML = carName;
</script>
</body>
</html>
```

JavaScript Initializations are Not Hoisted

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
```

```
<script>
var x = 5; // Initialize x
var y;   // Declare y
elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x + " " + y;       // Display x and y
y = 7;   // Assign 7 to y
</script>
</body>
</html>
```

Declare Your Variables At the Top

- Hoisting ka behavior kaafi developers ke liye unknown ya overlook hota hai.
- Agar developer hoisting ko samajh nahi paaye, toh program me bugs (errors) aa sakte hain.
- Bugs avoid karne ke liye, hamesha apne variables ko har scope ke shuru me declare karein.
- Ye ek accha rule hai, kyunki JavaScript code ko isi tarah interpret karta hai.

JavaScript Use Strict

- "use strict" directive ECMAScript 5 me introduce hua tha.
- Iska matlab hai ki JavaScript code strict mode me execute hoga.
- Ye statement nahi hai, balki literal expression hai.
- Purane JavaScript versions isko ignore kar dete hain.

```
<!DOCTYPE html>
<html>
<body>
<p>"use strict" in a function will only cause errors in that function.</p>
<p>Activate debugging in your browser (F12) to see the error report.</p>
<script>
x = 3.14;    // This will not cause an error.
myFunction();
function myFunction() {
  "use strict";
  y = 3.14;  // This will cause an error (y is not defined).
}
</script>
</body>
</html>
```

The "use strict"; Syntax

- "use strict"; ka syntax older JavaScript versions ke saath compatible banaya gaya hai.
- Agar aap numeric literal (4 + 5;) ya string literal ("John Doe";) likhte ho, koi side effect nahi hota. Ye simply compile hota hai aur ignore ho jaata hai.
- Iska matlab: "use strict"; sirf naye compilers ke liye matter karta hai, jo iska meaning samajhte hain.

Why Strict Mode?

- Strict mode JavaScript ko secure aur predictable banata hai.
- Bad syntax ko real errors me convert karta hai.
- Mistyped variable names:
- Normal JS: Naya global variable create ho jaata hai.
- Strict mode: Error throw hota hai, accidental globals nahi bante.
- Non-writable property assignments:
- Normal JS: Assignment silently fail ho sakta hai.
- Strict mode: Error throw hota hai agar non-writable, getter-only, non-existing property/variable/object ko assign karein.
- Strict mode bugs aur accidental errors ko reduce karta hai.

```
<!DOCTYPE html>
<html>
<body>
<h2>With "use strict":</h2>
<h3>Inside functions, the "this" keyword is no longer the global object if not specified:</h3>
<script>
"use strict";
function myFunction() {
  alert(this);
}
myFunction();
</script>
</body>
</html>
```

JavaScript Code Blocks

- Curly Braces { } in JavaScript
- Code block / block statement: Statements ka group jo curly braces { } ke andar likha jata hai.
- Importance of code blocks:

- Control execution flow – decide karte hain ki kaunsa code kab chale.
- Define variable scope – variables ka scope block ke andar ya bahar hota hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Loops</h1>
<h2>The While Loop</h2>
<p id="demo"></p>
<script>
let text = "";
let i = 0;
while (i < 10) {
  text += "The number is " + i + "<br>";
  i++;
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

Defining Scope

- Variables jo let aur const se code block ke andar declare hote hain, wo block-scoped hote hain.
- Matlab: Ye variables sirf ussi block ke andar accessible hote hain.
- Benefits:
- Unintended variable overwrites se bacha jaata hai.
- Code ka organization aur readability better hota hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Block Scope</h1>
<p id="demo"></p>
<script>
{
  let x = 10;
  // x is accessible here
}
// x is not accessible here
document.getElementById("demo").innerHTML = "x is " + typeof x;
</script>
```

```
</body>
</html>
```

Standalone Blocks

- Standalone code blocks aise blocks hote hain jo kisi if statement, function, ya loop se attached nahi hote.
- Ye independent { } blocks bhi create kiye ja sakte hain.
- Purpose:
- let aur const variables ke liye temporary scope create karna.
- Code ko organized aur safe rakhna, accidental variable conflicts se bachna.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Block Scope</h1>
<p id="demo"></p>
<script>
{
  let x = 10;
  let y = 100;
  let areal = x * y;
  document.getElementById("demo").innerHTML = "Areal is " + areal;
}
</script>
</body>
</html>
```

Encapsulation:

- Variables sirf block ke andar hi available hote hain.
- Ye global scope ko “pollute” hone se bachata hai.
- Code clean aur safe rehta hai, name collisions kam hote hain.

Temporary Use:

- Agar variables sirf calculation ke liye chahiye, block ke andar declare karo, use karo, aur automatically discard ho jaate hain.

Organized Code:

- Related variables aur statements ko apni scope me group kar sakte ho.
- Accidental name conflicts se bachata hai without forcing a function or object.
- Code readable aur maintainable rehta hai.

JavaScript Coding Conventions

- Style guidelines for programming: Programming ke liye ek set of rules ya guidelines.
- Naming & declaration rules for variables and functions: Variables aur functions ke naam aur unko declare karne ka standard tarika.
- Use of white space, indentation, and comments: Code me spacing, indentation aur comments ka sahi use.
- Programming practices and principles: Best practices aur principles follow karna for clean aur efficient code.
- Improve code readability: Code easily samajh aana aur read karna easy ho jaye.
- Make code maintenance easier: Bugs fix karna aur code update karna simple ho jaye.
- Ensure consistent and high-quality code: Code ka style aur quality consistent rahe, errors kam ho.

Variable Names

- CamelCase for identifiers: W3Schools me variables aur functions ke names ke liye camelCase use hota hai.
- Example: myVariable, calculateSum()
- All names start with a letter: Variable ya function ka naam hamesha letter se start hona chahiye.
- More details: Page ke bottom me naming rules ka detailed discussion diya gaya hai.

Code Indentation

Definition: Code ko properly space aur tab ke saath align karna, taki structure clearly dikh sake.

- Purpose:
  - Code readable aur understandable banaye.
  - Nested blocks aur scope clearly dikhaye.
  - Bugs aur errors dhundna easy ho jaaye.

Statement Rules

- Hamesha semicolon (;) se end karein.
- Examples:
- let x = 10;
- const cars = ["Volvo", "Saab", "Fiat"];
- Complex / Compound Statements:
- Opening bracket { ko first line ke end me rakhein.
- Opening bracket se pehle ek space rakhein.
- Closing bracket } ko new line me rakhein, bina leading spaces ke.

- Complex statements ko semicolon se end nahi karein.

Object Rules

- Opening bracket {: Object name ke same line me rakhein.
- Property-value separator: Colon : ke baad ek space use karein.
- Quotes: String values ke liye quotes " " use karein, numeric values ke liye nahi.
- Last property: Last property-value pair ke baad comma na lagayein.
- Closing bracket }: New line me rakhein, bina leading spaces ke.
- Semicolon: Hamesha object definition ke end me semicolon ; lagayein.

Line Length < 80

- Avoid lines longer than 80 characters for better readability.
- Agar statement ek line me fit nahi hota, toh break karne ka best place hai:
- After an operator (+, -, =, etc.)
- After a comma (,)

```
<!DOCTYPE html>
<html>
<body>
<h2>My Web Page</h2>
<p>The best place to break a code line is after an operator or a comma.</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"Hello Dolly.";
</script>
</body>
</html>
```

JavaScript Naming Conventions

- Consistency: Hamesha same naming convention use karein throughout your code.
- Common practices:
- Variables & functions: camelCase
- Global variables: UPPERCASE (common, but optional)
- Constants (like PI): UPPERCASE
- Naming style choices:
- Hyphens (-)

- Allowed in HTML attributes (data-quantity) and CSS properties (font-size).
- Not allowed in JavaScript variable names (confused with subtraction).
- Underscores (_)
- Used in SQL, PHP, or by some programmers (date_of_birth).
- PascalCase
- Often preferred by C programmers (MyFunction).
- camelCase
- Preferred in JavaScript, jQuery, and most JS libraries (myVariable).

Loading JavaScript in HTML

- JavaScript HTML me 3 main ways se load kiya ja sakta hai
- Inline JavaScript: HTML element ke attributes me directly likhna (onclick, onchange)
- Internal JavaScript: <script> tag ke andar HTML me directly likhna
- External JavaScript: Alag .js file me likhna aur <script src="file.js"></script> se load karna
- <script> tag ko head ya body ke end me place kar sakte hain
- defer aur async attributes se script loading behavior control hota hai

Accessing HTML Elements

- By ID: document.getElementById("id")
- By Class Name: document.getElementsByClassName("className")
- By Tag Name: document.getElementsByTagName("tagName")
- By Query Selector: document.querySelector("selector") (first match)
- By Query Selector All: document.querySelectorAll("selector") (all matches)
- Access Form Elements: document.forms["formName"] or document.forms[0]

File Extensions

- HTML files: .html (or .htm)
- CSS files: .css
- JavaScript files: .js

Use Lower Case File Names

- Zyada tar web servers (Apache, Unix) case sensitive hote hain
- Example: london.jpg ≠ London.jpg
- Kuch servers (Microsoft, IIS) case insensitive hote hain

- Example: london.jpg = London.jpg
- Upper aur lower case mix karne se bahut consistent rehna zaroori hai
- Case-insensitive se case-sensitive server par move karne par chhoti galti bhi site break kar sakti hai

JavaScript Keywords Reference

- JavaScript statements ka start statement identifier se hota hai, jo action define karta hai
- Statement identifiers reserved words hote hain
- Reserved words ko variable, function ya kisi aur cheez ke naam ke liye use nahi kar sakte

Statement	Description
{ }	Statements ka block create karta hai
async function	AsyncFunction object create karta hai
async function*	AsyncGeneratorFunction object create karta hai
await using	Local variables asynchronously dispose karne ke liye declare karta hai
break	Loop ya switch se bahar nikalta hai
class	Class declare karta hai
const	Constant variable declare karta hai
continue	Current loop iteration skip karke next me chala jata hai
debugger	Execution ko stop karta hai aur debugger call karta hai
do...while	Block execute karta hai aur condition true hone tak repeat karta hai
for	Block ko multiple times loop karta hai
for...in	Object properties ke through loop karta hai
for...of	Iterable object ke values ke through loop karta hai
for await...of	Async iterable objects ke values ke through loop karta hai
function	Function declare karta hai
function*	GeneratorFunction object create karta hai
if...else...else if	Condition ke basis par block execute karta hai

Statement	Description
import	Module ka read-only import define karta hai
import attributes	Module ka loading behavior define karta hai
let	Variable declare karta hai
return	Function execution stop karta hai aur value return karta hai
switch	Cases ke basis par block execute karta hai
throw	Error generate karta hai
try...catch...finally	Error handling ke liye block define karta hai
using	Local variables synchronously dispose karne ke liye declare karta hai
var	Variable declare karta hai
while	Condition true hone tak block execute karta hai

JavaScript Keyword Reserved

abstract	arguments	async *	await *
boolean	break	byte	case
catch	char	class *	const *
continue	debugger	default	delete
do	double	else	enum *
eval	export *	extends *	false
final	finally	float	for
function	goto	if	implements
function	import *	in	instanceof

int	interface	let *	long
native	new	null	package
private	protected	public	return
short	static	super *	switch
synchronized	this	throw	throws
transient	true	try	typeof
using *	var	void	volatile
while	with	yield	

JavaScript Versions

Version	Year / Name	Key Features / Notes
ES1	1997	First ECMAScript standard.
ES2	1998	Mainly editorial changes.
ES3	1999	Added things like regular expressions, try/catch, switch, do-while loops.
ES4	—	Not released. The proposal was abandoned.
ES5	2009	Big update: introduced strict mode, JSON support, Array.isArray(), String.trim(), and more.
ES5.1	2011	Minor/editorial update.
ES6 / ES2015	2015	Very significant release. Features: let & const, classes, modules, arrow functions, template literals, promises, for…of, maps/sets, generators, etc.
ES2016 (ES7)	2016	Exponentiation operator (**), Array.prototype.includes. (

Version	Year / Name	Key Features / Notes
ES2017 (ES8)	2017	async / await, Object.entries(), Object.values() and some more.
ES2018 (ES9)	2018	Rest / spread properties, asynchronous iteration, RegExp improvements.
ES2019 (ES10)	2019	Features like Array.prototype.flat(), flatMap(), stable Array.sort(), etc.
ES2020 (ES11)	2020	New primitives & operators: BigInt, nullish coalescing (??), optional chaining (?.), globalThis, Promise.allSettled, dynamic import(), etc.
ES2021 (ES12)	2021	More updates: e.g. String.prototype.replaceAll, Promise.any, logical assignment operators.
…	…	ECMAScript continues to evolve yearly. The 16th edition / ES2025 was released in June 2025.

History of JavaScript

Year	ECMA/JS Version	Browser / Event	Explanation
1995	-	-	JavaScript invent hua by Brendan Eich
1996	JS 1.0	Netscape 2	Netscape 2 me JavaScript 1.0 release hua
1997	ES1	-	JavaScript officially ECMA standard (ECMA-262) bana
1997	ES1	IE4	IE4 ne pehli baar ES1 support kiya
1998	ES2	-	ECMAScript 2 release hua
1998	JS 1.3	Netscape 42	Netscape 42 me JavaScript 1.3 aaya
1999	ES2	IE5	IE5 ne pehli baar ES2 support kiya
1999	ES3	-	ECMAScript 3 release hua
2000	ES3	IE5.5	IE5.5 ne ES3 support kiya
2000	JS 1.5	Netscape 62 / Firefox 1	Netscape 62 aur Firefox 1 me JS 1.5 aaya
2008	ES4	-	ES4 abandoned (release nahi hua)
2009	ES5	-	ECMAScript 5 release hua

Year	ECMA/JS Version	Browser / Event	Explanation
2011	ES5	IE9 / Firefox 4	IE9 ne pehli baar ES5 support kiya; Firefox 4 me JS 1.8.5 aaya
2012	ES5	Safari 6 / IE10 / Chrome 23	Full ES5 support in these browsers
2013	ES5	Firefox 21 / Opera 15	Full ES5 support
2014	ES5	All major browsers	ES5 ka full support sab browsers me ho gaya
2015	ES6	-	ECMAScript 6 release hua (modern JS start)
2016	ES6	Chrome 51 / Opera 38 / Safari 10	Full ES6 support in these browsers
2017	ES6	Firefox 54 / Edge 15	Full ES6 support
2018	ES6	All major browsers	Sab browsers ne full ES6 support provide kiya
Year	ECMA/JS Version	Browser / Event	Explanation (Hinglish)
1995	-	-	JavaScript invent hua by Brendan Eich
1996	JS 1.0	Netscape 2	Netscape 2 me JavaScript 1.0 release hua
1997	ES1	-	JavaScript officially ECMA standard (ECMA-262) bana
1997	ES1	IE4	IE4 ne pehli baar ES1 support kiya
1998	ES2	-	ECMAScript 2 release hua
1998	JS 1.3	Netscape 42	Netscape 42 me JavaScript 1.3 aaya
1999	ES2	IE5	IE5 ne pehli baar ES2 support kiya
1999	ES3	-	ECMAScript 3 release hua
2000	ES3	IE5.5	IE5.5 ne ES3 support kiya
2000	JS 1.5	Netscape 62 / Firefox 1	Netscape 62 aur Firefox 1 me JS 1.5 aaya
2008	ES4	-	ES4 abandoned (release nahi hua)
2009	ES5	-	ECMAScript 5 release hua
2011	ES5	IE9 / Firefox 4	IE9 ne pehli baar ES5 support kiya; Firefox 4 me JS 1.8.5 aaya
2012	ES5	Safari 6 / IE10 / Chrome 23	Full ES5 support in these browsers
2013	ES5	Firefox 21 / Opera 15	Full ES5 support
2014	ES5	All major browsers	ES5 ka full support sab browsers me ho gaya
2015	ES6	-	ECMAScript 6 release hua (modern JS start)

Year	ECMA/JS Version	Browser / Event	Explanation
2016	ES6	Chrome 51 / Opera 38 / Safari 10	Full ES6 support in these browsers
2017	ES6	Firefox 54 / Edge 15	Full ES6 support
2018	ES6	All major browsers	Sab browsers ne full ES6 support provide kiya

JavaScript Classes

- Class ek blueprint hai objects create karne ke liye.
- JavaScript me ES6 (2015) se classes introduce hui.
- Pehle hum constructor functions use karte the, ab class syntax simpler aur readable hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Classes</h1>
<p>Creating two car objects from a car class:</p>
<p id="demo"></p>
<script>
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
}
const myCar1 = new Car("Ford", 2014);
const myCar2 = new Car("Audi", 2019);
document.getElementById("demo").innerHTML =
myCar1.name + " " + myCar2.name;
</script>
</body>
</html>
```

The Constructor Method in JavaScript Classes

1. **Special Method**
   - Constructor ek special function hai inside a class.
   - Iska exact naam “constructor” hona chahiye.
   - Agar aap naam change karoge, wo constructor nahi hoga.

2. **Automatic Execution**
   - Jab bhi aap new object create karte ho using new, constructor automatically run hota hai.
   - Aapko manually call karne ki zarurat nahi.
3. **Initialize Object Properties**
   - Constructor ka main kaam hota hai object ke properties set karna.
   - Ye ensure karta hai ki har naya object apne initial values ke saath ready ho.

Class Methods

- Class methods functions hote hain jo class ke andar define kiye jaate hain.
- Ye objects ke behavior ko define karte hain.
- Syntax bilkul object methods jaisa hota hai.

Syntax & Rules

1. Class banane ke liye class keyword use hota hai.
2. Constructor method add karna mandatory nahi hai, lekin usually use hota hai properties initialize karne ke liye.
3. Methods directly class ke andar define kiye jaate hain, function keyword nahi lagate.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Class Method</h1>
<p>Pass a parameter into the "age()" method.</p>
<p id="demo"></p>
<script>
class Car {
constructor(name, year) {
this.name = name;
 this.year = year;
  }
  age(x) {
    return x - this.year;
  }
}
const date = new Date();
let year = date.getFullYear();
const myCar = new Car("Ford", 2014);
document.getElementById("demo").innerHTML=
"My car is " + myCar.age(year) + " years old.";
</script>
</body>
</html>
```

use strict

- "use strict" enable karta hai strict mode, jisme errors ignore nahi hote.
- JavaScript classes hamesha strict mode me hi run hoti hain.
- Agar strict rules violate hote hain, error throw hota hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Classes uses "strict mode"</h1>
<p>In a JavaScript Class you cannot use variable without declaring it.</p>
<p id="demo"></p>
<script>
class Car {
 constructor(name, year) {
   this.name = name;
   this.year = year;
  }
  age() {
   // date = new Date();  // This will not work
   const date = new Date(); // This will work
   return date.getFullYear() - this.year;
  }
}
const myCar = new Car("Ford", 2014);
document.getElementById("demo").innerHTML =
"My car is " + myCar.age() + " years old.";
</script>
</body>
</html>
```

Class Inheritance

- extends keyword use karke ek class doosri class se inherit kar sakti hai.
- Inherited class ko child class ya subclass kehte hain.
- Jo class inherit ho rahi hai, usse parent class ya superclass kehte hain.
- Child class parent class ke saare methods aur properties automatically inherit kar leti hai.
- Child class apne additional methods aur properties bhi add kar sakti hai.
- Parent class ke constructor ko call karne ke liye super() use karna padta hai.

```
<!DOCTYPE html>
```

```
<html>
<body>
<h1>JavaScript Class Inheritance</h1
<p>Use the "extends" keyword to inherit all methods from another class.</p>
<p>Use the "super" method to call the parent's constructor function.</p
<p id="demo"></p>
<script>
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return 'I have a ' + this.carname;
  }
}
class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model;
  }
}
const myCar = new Model("Ford", "Mustang");
document.getElementById("demo").innerHTML = myCar.show();
</script>
</body>
</html>
```

Getters and Setters

- Getters: Methods jo object properties ko access karte hain, jaise property ho.
- Setters: Methods jo object properties ko modify karte hain, aur values set karne se pehle kuch processing kar sakte hain.
- Use get aur set keywords.
- Useful jab aapko property access ya assignment ke time extra logic lagana ho.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Class Setters</h1>
```

```
<p>When using a setter to set a property value, you do not use parantheses.</p>
<p id="demo"></p>
<script>
class Car {
  constructor(brand) {
    this._carname = brand;
  }
  set carname(x) {
    this._carname = x;
  }
  get carname() {
    return this._carname;
  }
}
const myCar = new Car("Ford");
myCar.carname = "Volvo";
document.getElementById("demo").innerHTML = myCar.carname;
</script>

</body>
</html>
```

Hoisting & Classes

- Class declarations are NOT hoisted.
  - Matlab aap class ko use karne se pehle declare karna zaruri hai.
- Agar aap class ko declare karne se pehle use karenge, to ReferenceError aayega.
- Functions aur variables (var, let, const) se alag behave karte hain:
  - Functions hoist ho jate hain (function declaration).
  - Variables var hoist hote hain but let/const temporal dead zone me hote hain.
  - Classes bilkul hoist nahi hoti.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Classes are not Hoisted</h1>
<p>You will get an error if you try to use a class before it is declared.</p>
<p id="demo"></p>
<script>
//You cannot use the class yet.
//myCar = new Car("Ford") will raise an error.
class Car {
```

```
  constructor(brand) {
    this.carname = brand;
  }
}
/Now you can use the class:
const myCar = new Car("Ford");
document.getElementById("demo").innerHTML = myCar.carname;
</script>
</body>
</html>
```

JavaScript Static Methods

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Class Static Methods</h1>
<p>To use the "myCar" object inside the static method, you can send it as parameter.</p>
<p id="demo"></p>
<script>
class Car {
  constructor(name) {
    this.name = name;
  }
  static hello(x) {
    return "Hello " + x.name;
  }
}
const myCar = new Car("Ford");
document.getElementById("demo").innerHTML = Car.hello(myCar);
</script>
</body>
</html>
```

JavaScript Iteration

- Iteration ka matlab hai kisi sequence ya collection (array, object, string, etc.) ke items ko repeat karke access karna.
- JavaScript me loops aur iteration methods ka use hota hai.

Loop Type	Description
for	Iterates over values and expressions
while	Iterates over a condition
do...while	Iterates over a condition
for...in	Iterates over the properties of an Object
for...of	Iterates over array like objects
forEach()	Iterates over each element in an Array

JavaScript Iterables

- Iterable wo object hai jisko aap for...of loop ke saath iterate kar sakte ho.
- Iterable objects ke paas Symbol.iterator method hota hai jo iterator return karta hai.
- Common Iterables:
  - Arrays
  - Strings
  - Maps
  - Sets
  - Arguments object (ES6+)

Iterating Over a String

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Iterables</h1>
<h2>Iterate over String Elements</h2>
<p id="demo"></p>
<script>
// Create a String
const name = "W3Schools";
// Iterate over the string elements
let text = ""
for (const x of name) {
```

```
  text += x + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

Iterating Over an Array

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Iterables</h1>
<h2>Iterate over Array Elements</h2>
<p id="demo"></p>

<script>
// Create an Array
const numbers = [2,4,6,8];
// Iterate over the array elements
let text = "";
for (const x of numbers) {
  text += x + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

Iterating Over a Set

Result Size: 486 x 495

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Iterables</h1>
<h2>Iterate over Set Elements</h2>
<p id="demo"></p>
<script>
// Create a Set
const letters = new Set(["a","b","c"]);
// Iterate over the set elementss
```

```
let text = "";
for (const x of letters) {
  text += x + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

Iterating Over a Map

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Iterables</h1>
<h2>Iterate over Map Elements</h2>
<p id="demo"></p>
<script>
// Create a Map
const fruits = new Map([
  ["apples", 500],
  ["bananas", 300],
  ["oranges", 200]
]);
// Iterate over the map elements
let text = "";
for (const x of fruits) {
  text += x + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

JavaScript Iterators

- Iterator ek object hai jo elements ko sequentially access karne ka standard way provide karta hai.
- Iterator ka main kaam hai ek ek item return karna aur iteration complete hone ka signal dena.

The next() Method

- next() ek method hai jo iterator objects me hoti hai.
- Har call par ye ek object return karta hai jisme 2 properties hoti hain:
  - value → current iteration ka value
  - done → boolean, false agar next element available ho, true agar iteration khatam ho gayi ho

Symbol.iterator

Note

Technically, iterables must implement the `Symbol.iterator` method.

In JavaScript the following are iterables:

- Strings
- Arrays
- Typed Arrays
- Sets
- Maps

Because their prototype objects have a `Symbol.iterator` method:

Iterator Helper Functions (ES2025)

- JavaScript 2025 ne naye iterator helper methods introduce kiye hain.
  Ye methods iterators ko direct manipulate karne ki power dete hain—
  bina arrays me convert kiye.
- Ye exactly Array methods (map, filter, take, etc.) jaise behave karte hain,
  lekin iterators par directly kaam karte hain, jisse memory bachti hai
  aur performance better hoti hai.

Function	Description
drop(n)	Iterator ke pehle n elements ko skip karta hai, baaki items return karta hai.
every(fn)	Return karta hai true agar saare elements test function ko pass karein.

Function	Description
filter(fn)	Sirf un elements ka iterator return karta hai jo filter/test pass karte hain.
find(fn)	Pehla element return karta hai jo test function satisfy kare.
flatMap(fn)	Har element ko map karta hai aur result ko flatten kar deta hai (1 level flat).
forEach(fn)	Iterator ke har element par function execute karta hai (value return nahi karta).
from(iterable)	Kisi iterable (array, string, set, etc.) se naya iterator object banata hai.
map(fn)	Har element ko transform karke naya iterator return karta hai.
reduce(fn, initialValue)	Sare elements ko combine karke single value return karta hai.
some(fn)	Agar koi ek element bhi test function pass kare to true return karta hai.
take(n)	Sirf pehle n elements ka iterator return karta hai.

The Iterator.from() Method

Iterator.from() kisi bhi iterable (jaise Array, String, Set, Map, Generator, etc.) ko iterator object me convert karta hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Iterators</h1>
<h2>The Iterator.from() Method</h2>
<p id="demo"></p>
<script>
// Create an iterator
const myIterator = Iterator.from([1, 2, 3]);
// Iterate over the elements
let text = ""
for (const x of myIterator) {
  text += x + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

The filter() Method

- filter() naya iterator return karta hai.
- Isme sirf wo elements hote hain jo filter function (test function) ko pass karte hain.
- Ye lazy evaluation use karta hai → elements ek-ek karke process hote hain.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Iterators</h1>
<h2>The filter() Method</h2>
<p id="demo"></p>
<script>
// Create an iterator
const myIterator = Iterator.from([32, 33, 16, 40]);
// Filter the iterator
const filteredIterator = myIterator.filter(x => x > 18);
// Iterate over the filtered elements
let text = "";
for (const x of filteredIterator) {
  text += x + "<br>";
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

The map() Method

- map() naya iterator return karta hai.
- Har element ko transform (modify) kar deta hai using a map function.
- Ye process lazy evaluation ke through hota hai — ek-ek element jab needed ho tab transform hota ha

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Iterators</h1>
<h2>The map() Method</h2>
<p>The map() method returns a new iterator with all elements transformed by a map function.</p>
<p id="demo"></p>
<script>
// Create an iterator
```

```
const myIterator = Iterator.from("123456789");
// Map the Iterator
const mappedIterator = myIterator.map(x => x * 2);
// Iterate over all elements
let text = "";
for (const x of mappedIterator) {
  text += x + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

Iterator.flatMap() Method (ES2025)

- Definition
- flatMap() ek naya iterator return karta hai.
- Ye 2 kaam karta hai:
  - Har element ko map karta hai (transform).
  - Mapping ka result agar iterable ho, to usko flatten karta hai (sirf 1 level flatten).

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Iterators</h1>
<h2>The flatMap() Method</h2>
<p>The flatMap() method returns a new iterator with all elements transformed by a flatmap function.</p>
<p id="demo"></p>
<script>
// Create an iterator
const myIterator = Iterator.from([1, 2, 3, 4, 5, 6])
// Map the Iterator
const mappedIterator = myIterator.flatMap(x => [x, x * 10])
// Iterate over the elements
let text = "";
for (const x of mappedIterator) {
  text += x + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
```

```
</body>
</html>
```

The take() Method

- take(n) ek naya iterator return karta hai.
- Ye iterator maximum n elements hi return karega.
- Agar iterable me zyada elements ho, tab bhi sirf pehle n values milengi.
- Ye lazy evaluation par kaam karta hai → zaroorat padne par hi values generate hoti hain.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Iterators</h1>
<h2>The take Method</h2>
<p>The take() method returns a new iterator that yields a specified number of elements.</p>
<p id="demo"></p
<script>
// Create an iterator
const myIterator = Iterator.from([1, 2, 3, 4, 5, 6])
// Take the first five elements
const firstFive = myIterator.take(5)
// Iterate over all elements
let text = "";
for (const x of firstFive) {
  text += x + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

The drop() Method

- drop(n) ek naya iterator return karta hai.
- Ye pehle n elements ko skip/drop kar deta hai.
- Uske baad ke baaki elements ko yield karta hai.
- Ye bhi lazy evaluation use karta hai → elements tabhi process hote hain jab zaroorat ho.

```
<!DOCTYPE html>
<html>
<body>
```

```
<h1>JavaScript Iterators</h1>
<h2>The flatMap() Method</h2>
<p>The flatMap() method returns a new iterator with all elements transformed by a flatmap function.</p
<p id="demo"></p>
<script>
// Create an iterator
const myIterator = Iterator.from([1, 2, 3, 4, 5, 6]);
// Remove the first five
const firstFive = myIterator.drop(5);
// Iterate over all elements
let text = "";
for (const x of firstFive) {
  text += x + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

The find() Method

- find(fn) iterator ke elements ko ek-ek karke check karta hai.
- Ye sabse pehla element return karta hai jo test function (fn) ko satisfy kare.
- Agar koi matching element na mile → undefined return hota hai.
- Yeh arrays ke .find() jaisa hi behave karta hai, but direct iterator par kaam karta hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Iterators</h1>
<h2>The find() Method</h2>
<p>The find() method returns the value of the first iterator element that passes a test provided by a test function.</p>
<p id="demo"></p>
<script>
// Create an iterator
const myIterator = Iterator.from([3, 10, 18, 30, 20]);
// Find first greater than 18
let result = myIterator.find(x => x > 18);
document.getElementById("demo").innerHTML = result;
</script>
</body>
```

</html>

The reduce() Method

- reduce(reducerFn, initialValue)
  - Iterator ke har element ko ek accumulator ke saath combine karta hai.
  - End result ek single value hota hai.
- Ye arrays ke .reduce() jaisa hi kaam karta hai, bas direct iterator par work karta hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Iterators</h1>
<h2>The reduce() Method</h2>
<p>The reduce() method executes a reducer function for each iterator element.</p>
<p id="demo"></p
<script>
function myFunc(total, num) {
  return total + num;
}
// Create an Iterator
const myIterator = Iterator.from([175, 50, 25]);
// Reduce the Iterator
let result = myIterator.reduce(myFunc);
document.getElementById("demo").innerHTML = "The sum of all items is: " + result;
</script>
</body>
</html>
```

every() Method

every(fn) method true return karta hai agar iterator ke saare elements provided test function ko satisfy karte hain.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Iterators</h1>
<h2>The every() Method</h2>
<p>The every() method teturns true if all elements in an interator pass a test provided by a function.</p>
<p id="demo"></p>
<script>
```

```
// Create an Iterator
const myIterator = Iterator.from("123456789");
// Is every Element greater than 7?
let result = myIterator.every(x => x > 7);
document.getElementById("demo").innerHTML = result;
</script>
</body>
</html>
```

some() Method

some(fn) method true return karta hai agar kam se kam ek element iterator me se provided test function ko satisfy kare.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Iterators</h1>
<h2>The some() Method</h2>
<p>The some() method teturns true if any of the elements in an interator pass a test provided by a function.</p>
<p id="demo"></p>
<script>
// Create Iterator
const myIterator = Iterator.from("123456789");
// Is some Elements  greater than 7?
let result = myIterator.some(x => x > 7);
document.getElementById("demo").innerHTML = result;
</script>
</body>
</html>
```

The forEach() Method

forEach() method har element ke liye provided function ko ek baar execute karta hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Iterators</h1>
<h2>The forEach() Method</h2>
<p id="demo"></p>
<script>
```

```
// Create an iterator
const myIterator = Iterator.from("123456789");
// Iterate over all elements
let text = "";
myIterator.forEach (x => text += "<br>" + x);
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

JavaScript Generators

- A JavaScript function can only return one value.
- A JavaScript generator can return multiple values, one by one.
- A JavaScript generator can yield a stream of data.
- A JavaScript generator can be paused and resumed.

Generator Functions

Generator function ek special JavaScript function hai jo pause aur resume ho sakta hai yield ke through. Ye ek ke bajaye multiple values ek-ek karke return kar sakta hai.

Generator Objects

- Generator Object ek object hai jo generator function call karne par return hota hai.
- Ye iterable aur iterator dono protocols follow karta hai.
- Matlab, isko for...of loop se iterate kiya ja sakta hai.
- Iske paas .next() method hoti hai jo next yield value return karti hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Generators</h1>
<p>A generator returns an object with two properties: value and done.</p>
<p id="demo"></p>
<script>
function* myStream() {
// return {value:1, done:false}
yield 1;
// return {value:2, done:false}
yield 2;
// return {value:3, done:true}
return 3;
```

```
}
// Create a Generator
let myGenerator = myStream();
let text = "";
// Loop the Generator
for (let value of myGenerator) {
  text += value + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

The yield Keyword

- yield keyword execution ko pause karta hai aur ek value return karta hai caller ko.
- Generator ka internal state save ho jata hai, aur next() call karne par resume ho sakta hai wahi se jahan yield pe ruk gaya tha.
- Generators apni state maintain karte hain between yield calls, isliye wo execution continue kar sakte hain from the last paused point.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Generators</h1>
<p>A generator returns an object with two properties: value and done.</p>
<p id="demo"></p>
<script>
function* myStream() {
// return {value:1, done:false}
yield 1;
// return {value:2, done:false}
yield 2;
// return {value:3, done:false}
yield 3;
}
// Create a Generator
let myGenerator = myStream();
let text = "";
// Loop the Generator
for (let value of myGenerator) {
  text += value + "<br>";
```

```
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

Generator Object Methods

Method	Description	Example
next()	Generator ki next execution ko resume karta hai aur next yield value return karta hai.	gen.next()
return()	Generator ko finish karta hai aur specified value return karta hai.	gen.return(10)
throw()	Generator me exception throw karta hai.	gen.throw('Error!')

Custom Iterators

- Generators simplify custom iterators: Complex data structures ya sequences ke liye custom iterators easily create kiye ja sakte hain.
- On-demand value generation: Values efficiently generate ki ja sakti hain jab zarurat ho, suitable for potentially infinite data streams.
- Pause & resume: Execution ko pause aur resume karna possible hai, useful for fine-grained control over program flow.
- Async use (before async/await): Generators ko Promises ke saath use karke asynchronous operations ko sequential style me manage kiya ja sakta tha.

Asynchronous JavaScript

- Asynchronous JavaScript ka matlab hai code block kiye bina execute hona.
- Long-running operations jaise API calls, timers, file reading background me run hote hain.
- Common async techniques:
- Callbacks – function ko argument me pass karke later call karte hain.
- Promises – future value ko represent karte hain, .then() aur .catch() ke saath handle hota hai.
- async/await – Promises ka easy aur sequential-looking syntax.
- Isse web apps ka performance aur responsiveness improve hota hai.

- JavaScript Asynchronous Flow ka matlab hai ki JS long-running tasks (jaise file reading, API calls, ya user input wait karna) ko background me run karta hai, bina baaki code ko block kiye.
- Asynchronous programming ka use karke JS blocking prevent karta hai.
- Isse kuch operations background me execute hote hain, aur unke results tab handle kiye jaate hain jab wo ready ho jate hain.
- Ye approach smooth aur responsive programs banata hai.

JavaScript Callbacks

- Callback ek function hota hai jo dusre function ke argument ke roop me pass kiya jata hai.
- Ye later execute hota hai, usually jab koi event occur ho ya asynchronous operation complete ho.
- Callback ka use asynchronous tasks ko handle karne ke liye hota hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Functions</h1>
<h2>Callback Functions</h2>

<p>The result of the calculation is:</p>
<p id="demo"></p>
<script>
function myDisplayer(something) {
  document.getElementById("demo").innerHTML = something;
}
function myCalculator(num1, num2, myCallback) {
  let sum = num1 + num2;
  myCallback(sum);
}
myCalculator(5, 5, myDisplayer);
</script>
</body>
</html>
```

JavaScript Promises

- Promise ek object hai jo future me asynchronous operation ka result represent karta hai.

- Ek promise ke 3 states hote hain:
- Pending → Operation abhi complete nahi hua
- Fulfilled → Operation successful complete hua, result available hai
- Rejected → Operation fail hua, error available hai
- Promises ka use callback hell ko avoid karne ke liye hota hai.
- Promises ko .then() aur .catch() ke through handle kiya jata hai.

JavaScript Async/Await

- Async/Await ek modern aur cleaner way hai asynchronous code handle karne ka.
- async function declare karta hai ki ye function asynchronous hai aur promise return karega.
- await keyword ka use karke hum promise ka result wait kar sakte hain without blocking baaki code.
- Ye asynchronous code ko synchronous jaisa readable bana deta hai.

Flow Concept	Description
Synchronous	Line by line execute hota hai, next task block hota hai.
Asynchronous	Dusra code run karne deta hai jab tak tasks complete nahi hote.
Events	Callback function store karta hai jo future me execute hoga.
Promises	Tools jo asynchronous operations ko cleanly handle karte hain.
Async/Await	Modern aur cleaner way asynchronous code handle karne ka.

JavaScript Modules

- JavaScript Modules code ko reusable aur organized parts me divide karte hain.
- Ek module variables, functions, classes export kar sakta hai aur dusre modules me import kiya ja sakta hai.
- Modules global scope ko pollute nahi karte, code clean aur maintainable rehta hai.
- export → kisi variable/function/class ko dusre module me use karne ke liye available banata hai.
- import → exported items ko current module me access karta hai.

Module Files

- JavaScript module usually ek .js file hota hai, lekin ye HTML script bhi ho sakta hai.
- Module file → .js file jo import/export use karti hai.
- Module script → HTML me <script type="module"> use karke import/export implement hota hai.

How to Use Modules

- Modules export aur import ka use karke functionalities share/interchange karte hain.
- HTML script me type="module" use karne se wo script module ke jaise treat hota hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Modules</h1>
<h3>Import add from math.js</h3>
<p id="demo"></p
<script type="module">
import { add } from './math.js';
let result = add(2, 3);
document.getElementById("demo").innerHTML = "The result is " + result;
</script>
</body>
</html>
```

Modules Can Export

- Variables
- Functions
- Objects
- Classes

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Modules</h1>
<h2>Import message from message.js</h2>

<p id="demo"></p>
<script type="module">
import message from "./message.js";
document.getElementById("demo").innerHTML = message();
</script>
</body>
</html>
```

Why Modules?

- Code Organization: Modules help code ko organize karne me, large codebases ko small, self-contained files me break karte hain, har file ek specific task ke liye.

- Prevent Naming Conflicts: Modules ka private scope hota hai, variables aur functions globally exposed nahi hote unless explicitly exported. Ye namespace pollution prevent karta hai.
- Better Readability: Modules code ko readable aur manageable banate hain, especially large projects ya team environments me.
- Reusability: Modules ke functions, variables, aur objects ko ek file se export karke dusri files ya projects me import kiya ja sakta hai, DRY principle follow hota hai.
- Maintainability & Debugging: Modules isolated hote hain, isliye bug fixes aur modifications easily ek module tak limited rehte hain, baaki system pe impact kam hota hai.
- Team Collaboration: Multiple developers ek saath kaam kar sakte hain without conflicts, module boundaries clear hoti hain.
- Encapsulation & Isolation: Variables/functions default private hote hain, only explicitly exported items accessible hote hain. Isse side effects kam hote hain aur code easier to reason hota hai.
- Dependency Management: Modules explicit import/export use karte hain, dependencies easily manage aur track ki ja sakti hain.
- Reliability: Modules script loading order ka manual process avoid karte hain, execution zyada reliable hota hai.

JavaScript Modules Export

The Export Keyword

- export keyword ka use karke module values, functions, ya objects ko dusre files me share karta hai.
- Ek module me multiple named exports ho sakte hain.
- Ek module me optional single default export bhi ho sakta hai.

Named Exports

- Named Exports me har item ko ek unique name diya jata hai.
- Items ko individually export kiya ja sakta hai ya ek sath {} me wrap karke export kar sakte hain.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Modules</h1>
<h3>Import name and age from person.js</h3>
<p id="demo"></p>
<script type="module">
import { name, age } from "./person.js";
let text = "My name is " + name + ", I am " + age + ".";
```

```
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

Name Strictness

- Named exports me exported item ka exact name use karna mandatory during import.
- Agar import name match nahi karta, to error aayega.
- Curly braces {} me exact name dena hota hai.

Toolbox Utilities

- Toolbox Utilities (Helpers) ek collection of small, reusable functions hote hain jo common tasks perform karte hain.
- Agar ek module me multiple helper functions hain, to named exports use karna best practice hai.
- Common utility modules examples:
- math.js → Math-related helper functions
- stringUtils.js → String manipulation helpers
- domHelpers.js → DOM-related helper functions
- dateHelpers.js → Date/time related helpers

When to Use Named Export?

Cases	Why Named Export
Many functions	Saare functions clearly list ho jate hain
Strictness needed	Name typos prevent hote hain
Big projects	Consistency improve hoti hai
Utility sets	Module structure ke saath match karta hai
Tree-shaking	Unused code automatically remove ho jata hai

Default Export

- Default Export ek module se ek main value export karta hai.
- Isse clear intent milta hai ki module ka primary functionality kya hai.
- Agar ek file ka purpose ek main function, class, ya value expose karna hai, to default export use karna best practice hai.

```
<!DOCTYPE html>
<html>
<body>
```

```
<h1>JavaScript Modules</h1>
<h2>Import text from message.js</h2>
<p id="demo"></p>
<script type="module">
import text from "./message.js";
document.getElementById("demo").innerHTML = text();
</script>
</body>
</html>
```

When to Use Default Export?

Point	Explanation
One main purpose	Jab module ka ek hi main kaam hota hai, toh direct import karna easy aur useful hota hai.
Flexible naming	Import karte time aap function/class ka naam change kar sakte ho, jaise as use karke — code zyada flexible ho jata hai.
Cleaner imports	Direct import se code short aur clean lagta hai, jaise sqrt() instead of math.sqrt().
Common pattern	Bohot saare frameworks (Flask, Django, NumPy) yeh import style use karte hain taaki main features ko easy access mil sake.

JavaScript Modules Import

Aap modules ko do tareeke se import kar sakte ho —
yeh depend karta hai ki module ne apni cheezein kaise export ki hain:

1. **Named exports:**
   Module kuch functions/variables ko naam dekar export karta hai.
   Import karte waqt aapko same naam use karna padta hai.
2. **Default exports:**
   Module ek hi main value/export deta hai.
   Import karte waqt aap koi bhi naam rakh sakte ho.

Named Import

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Modules</h1>
<h3>Import PI, add, subtract from math.js</h3>
```

```
<p id="demo"></p>
<script type="module">
import { PI, add, subtract } from './math.js';
document.getElementById("demo").innerHTML = PI + "<br>" + add(3,2) + "<br>" +
subtract(3,2);
</script>
</body>
</html>
```

Default Import

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Modules</h1>
<p id="demo"></p>
<script type="module">
import text from "./message.js";
document.getElementById("demo").innerHTML = text();
</script>
</body>
</html>
```

Default Import vs Named

Feature	Default Import	Named Import
Typical use case	Jab module ka ek hi main feature ho — like “main function/class”.	Jab module multiple functions/variables export karta ho.
Matches export name?	Nahi — aap import karte time koi bhi naam de sakte ho.	Haan — import karte waqt exact same naam use karna hota hai.
Can rename freely?	Hamesha rename kar sakte ho, bina kisi restriction ke.	Rename kar sakte ho, but sirf as ke saath.
Export count	Sirf ek hi default export allowed hota hai module me.	Aap multiple named exports rakh sakte ho.

JS Module Namespace

JavaScript ek module namespace object banata hai.
Yeh ek special object hota hai jo saare exported variables, functions, ya classes ko store karta hai jo us module ne export kiye hote hain.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Modules</h1>
<h2>Modules import</h2>
<p id="demo"></p>
<script type="module">
</script>
<script type="module">
import * as math from "./math_module.js";
let myPi = math.PI;
let result = math.add(2, 3);
document.getElementById("demo").innerHTML = "myPi is: " + myPi + "<br>Result is: " +
result;
</script>
</body>
</html>
```

Module Namespace Support

Module namespace support ka matlab hai ki JavaScript aapko ek module ke saare exports ko ek hi namespace object ke andar import karne ki ability deta hai.

Module Namespace Features

Feature	Explanation
Namespace	Ek special object hota hai jisme module ke saare named exports store hote hain.
Syntax	import * as name from "module" — is syntax se aap pura module ek namespace object ke andar le aate ho.
Purpose	Exports ko organize, safe tarike se access, aur cleanly reference karne ke liye use hota hai.
Read Only	Aap exports read kar sakte ho, par unhe reassign ya modify nahi kar sakte. Namespace object immutable hota hai.
Live Binding	Agar module ke andar koi export value change hoti hai, toh namespace object automatically updated value dikhata hai. Yeh “live link” hota hai.
Enumerable	Namespace object ki properties enumerable hoti hain, matlab for...in ya Object.keys() se easily list ho sakti hain.

Feature	Explanation
Prototype	Namespace ek plain object hota hai jiska koi prototype nahi hota. Isliye: Object.getPrototypeOf(math) === null true hota hai.

Module Namespace Export

Module namespace export ka matlab hai ki aap ek module se jo namespace object import kiya hai, usse hi poora ka poora kisi doosre module se export kar dena.

Namespace Export

Normally, jab aap kisi module ke saare exports import karna chahte ho, toh aap ek module namespace object banate ho.
Iske liye yeh syntax use hota hai:

```
import * as math from "./math.js";
```

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Modules</h1>
<h2>Modules import</h2>
<p id="demo"></p>
<script type="module">
</script>
<script type="module">
import * as math from "./math_module.js";
let myPi = math.PI;
let result = math.add(2, 3);
document.getElementById("demo").innerHTML = "myPi is: " + myPi + "<br>Result is: " +
result;
</script>
</body>
</html>
```

Syntax Overview

Syntax	Explanation / Result
export * from "module"	Module ke saare named exports ko individually re-export kar do. Namespace object nahi banta, har export alag se forward hota hai.
export * as name from "module"	Module ke saare exports ek single namespace object ke andar re-export karo. Dusre module me name.someExport ke through access hota hai.
export { name } from "module"	Module ke specific exports ko hi re-export karo. Bina poore module ko import kiye.

Aggregator Scripts

Aggregator script ek aisa script hota hai jo sirf import aur re-export ka kaam karta hai.

- Normally, agar aapke top module me bohot saare imports hote, toh code messy ho jata.
- Aggregator script me aap sab modules ke exports ko ek jagah collect kar lete ho, aur phir top module me sirf aggregator se import karte ho.

JavaScript Dynamic Modules

Dynamic modules ka matlab hai ki aap modules ko runtime (program chalate waqt) load kar sakte ho, compile-time par nahi.

- Normal import statements static hote hain — compile-time par hi modules ka structure fix ho jata hai.
- Dynamic import se aap condition ke basis par, user input ke basis par, ya kisi event ke time par module load kar sakte ho.

Modern Software & Dynamic Modules

Modern software me imports ko alag-alag chunks me split karte hain.

- Matlab, modules tab hi download hote hain jab zarurat hoti hai.
- Is method ke kai names hain:
  - Dynamic Import
  - Code Splitting
  - Lazy Loading
  - Conditional Loading

Dynamic Import Advantages:

1. Powerful feature – Modular aur efficient code likhne me help karta hai.
2. Flexible location – Unlike static import (jo file ke top par hona chahiye), dynamic import kisi bhi jagah use kar sakte ho:
   - Functions ke andar
   - Conditionals me

- Event handlers me

Type	Example	When Loaded
Static	import { add } from './math.js';	File load hote hi module load ho jata hai.
Dynamic	const math = await import('./math.js');	Module tab load hota hai jab zarurat ho — runtime par.

Improved Performance

Modules performance improve kar sakte hain kyunki ye code splitting allow karte hain.
Code Splitting Ka Matlab:

- Browser ko sirf wo JavaScript modules load karne hote hain jo user abhi use kar raha hai.
- Poore application ka code ek saath load nahi hota.
- Isse initial load fast hota hai aur user experience better hota hai.

Bundlers ke saath Benefits:
Jab modules ko bundlers jaise Webpack ya Rollup ke saath use karte hain, to aur zyada optimizations possible hain:

1. Tree-shaking – Unused code remove ho jata hai.
2. Code splitting – Modules alag chunks me split hote hain, runtime par load hote hain.
3. Minification – Code chhota aur optimized ban jata hai.

How Dynamic Import Works

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Dynamic Modules</h1>
<p id="demo"></p>
<script>
async function run() {
  const module = await import("./math.js");
  let result = module.add(2, 3);
  document.getElementById("demo").innerHTML = result;
}
run();
</script>
</body>
</html>
```

1. **Script starts running**
   - Browser ya Node.js script ko execute karna start karta hai.

2. **It defines and calls run()**
   - Ek function run() define hota hai aur turant call kiya jata hai.
3. **Inside run(), it dynamically loads math.js**
   - import("./math.js") ke through math.js runtime pe load hota hai.
4. **Once loaded, it gets access to the exported function add()**
   - math.js ke exports, jaise add, run() ke andar available ho jate hain.
5. **It calls add(2, 3) and gets 5**
   - Dynamically loaded function ko use karke calculation hota hai.
6. **It displays the result**
   - Result screen ya console me show hota hai (5).

Dynamic Import Key Features

Feature	Hinglish
Lazy Loading	Code ko tabhi load karo jab zarurat ho. Matlab user abhi feature use nahi kar raha → module load nahi hoga.
Performance Optimization	Initial page load ko fast banata hai, kyunki sirf required modules hi load hote hain.
Conditional Imports	Different modules ko conditions ke basis par import kar sakte ho. Jaise user ka role admin hai → admin module load karo, otherwise normal module.

JavaScript Meta Programming

Capability	Explanation
Inspect Object Properties	Object ke properties ko runtime pe dekhna aur analyze karna.
Intercept Operations	Object ke operations (jaise property access) ko rok ya modify karna.
Control Operations	Object ke operations ka behaviour change karna.
Modify Objects	Objects ko runtime me modify karna, naye properties ya methods add karna.
Modify Properties	Object ke properties ke values ya descriptors change karna.

Capability	Explanation
Modify Functions	Functions ke behaviour ko wrap, extend, ya replace karna.
Modify Classes	Classes ke methods ya properties ko runtime pe alter karna.
Generate Code Dynamically	Code ko runtime pe create aur execute karna (jaise eval, dynamic functions, etc.).

Inspecting Object Properties

Object.keys(obj) ek built-in method hai jo object ke saare keys ko array me return karta hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>Metaprogramming</h1>
<h2>The Object.keys() Method</h2>
<p>List all property names:</p>
<p id="demo"></p>
<script>
const user = { name: "Jan", age: 40 };

const myArr = Object.keys(user);
document.getElementById("demo").innerHTML = myArr;
</script>
</body>
</html>
```

Proxies For Metaprogramming

- JavaScript me Proxy ek object hota hai jo dusre object ke operations ko intercept kar sakta hai.
- Matlab, get, set, delete, function call jaise operations ko hum custom behavior ke saath handle kar sakte hain.
- Isliye Proxy metaprogramming ka powerful tool hai.

```
<!DOCTYPE html>
<html>
```

```
<body>
<h1>JavaScript Proxy</h1>
<h2>The set() Method</h2>
<p>Log changes to all property values:</p>
<p id="demo"></p>
<script>
// Create an Object
const user = { name: "Jan", age: 40 };
//Create a Proxy
const proxy = new Proxy(user, {
  set(target, property, value) {
  log(property + ": " + value);
    return target[property];
  }
});
function log(message) {
  const time = new Date().toLocaleTimeString();
  document.getElementById("demo").innerHTML += "[" + time + "] " + message + "<br>";
}
proxy.name = "John";
proxy.age = 45;
proxy.name = "Paul";
</script>
</body>
</html>
```

Why Metaprogramming

(What)	(How, Proxy / Metaprogramming)
Validation	Object ke values ko restrict karna (jaise age negative na ho) using set trap in Proxy.
Logging	Proxy ke get ya set traps use karke operations ko intercept karke log karna.
Debugging	Property read ya change hone par Proxy ke through values ko dynamically display karna.

(What)	(How, Proxy / Metaprogramming)
Reactive frameworks	Vue, MobX, Svelte jaise frameworks Proxy/metaprogramming use karke state changes detect karte hain aur UI update karte hain.
ORM / Database Mapping	Objects ko automatically wrap karna aur fields create karna database schema ke basis pe.
Dynamic APIs	Runtime me functions ya object structures create karna, jisse flexible APIs ban sakti hain.

JavaScript Reflect

Reflect kya hai?

- Reflect ek built-in JavaScript object hai.
- Isme methods hote hain objects ke low-level operations ke liye, jaise property access, setting, deletion, etc.
- Basically, ye JavaScript ke internal operations ko expose karta hai.

2. Reflect ka behavior

- Reflect ka behavior internal object operations ke jaise hota hai.
- Example: agar aap normally obj.prop = value karte ho, internally JS ye kaam [[Set]] operation ke through karta hai.
- Reflect.set(obj, prop, value) same low-level operation ko explicitly perform karta hai.

3. History

- Reflect ES6 (2015) me introduce hua.
- Iska purpose: metaprogramming aur Proxy ke saath low-level operations ko standard way me handle karna.

Why JavaScript Reflect?

Before Reflect:

- in keyword se property existence check karna

- delete keyword se property delete karna
- Object.defineProperty jaise methods use karna
- Internal mechanisms jaise [[Get]] aur [[Set]] directly accessible nahi

Reflect ke benefits:

- Sab object operations ek jagah (get, set, has, deleteProperty, etc.)
- Behavior predictable aur consistent
- Proxy ke saath perfect integration
- Errors aur results standardized (true/false ya result return)

Reflect.has()

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Object </h1>
<h2>The in Keyword</h2>
<p id="demo"></p>
<script>
// Create an Object
const person = {name: "John", lastname: "Doe"};
// Is name in Object
let answer = "name" in person;
document.getElementById("demo").innerHTML = "The answer is " + answer;
</script>
</body>
</html>
```

Reflect.deleteProperty()

```
<!DOCTYPE html>
<html>
<body>
<h1>Object Properties</h1>
<h2>The delete Method</h2>
<p id="demo"></p>
<script>
// Create an Object
const person = {name: "John", lastname: "Doe"};
// Delete the name Property
delete person.name;
document.getElementById("demo").innerHTML = Object.keys(person);
</script>
</body>
```

</html>

The Reflect Object

Method	Description
Reflect.get()	Property ka value get karna
Reflect.set()	Property ka value set karna
Reflect.has()	Check karna ki property exist karti hai
Reflect.deleteProperty()	Property delete karna
Reflect.defineProperty()	Nayi property define karna
Reflect.getOwnPropertyDescriptor()	Property ka descriptor get karna
Reflect.apply()	Function call karna
Reflect.construct()	Object create karna (jaise new)
Reflect.ownKeys()	Saari keys get karna (symbols included)
Reflect.isExtensible()	Check karna ki object grow kar sakta hai
Reflect.preventExtensions()	Object ko grow hone se rokna

Reflect.get()

```
<!DOCTYPE html>
<html>
<body>
<h1>Object Properties</h1>
<p id="demo"></p>
<script>
// Create an Object
const user = {name: "Jan", age: 40};
// Get the value of age
let age = user.age;
document.getElementById("demo").innerHTML = "Age: " + age;
</script>
</body>
</html>
```

Reflect.set()

```
<!DOCTYPE html>
```

```
<html>
<body>
<h1>Object Properties</h1>
<p id="demo"></p>
<script>
// Create an Object
const user = {name: "Jan", age: 40};
// Set the value of a Property
user.age = 41;
document.getElementById("demo").innerHTML = "Age is now " + user.age;
</script>
</body>
</html>
```

Reflect.apply()

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Functions</h1>
<h2>The apply() Method</h2>
<p id="demo"></p>
<script>
// Create a Function
function greet(message) {
  return message + ", " + this.name;
}
// Create an Object
const person = {name: "Jan"};
// Apply the Function to the Object
let msg = greet.apply(person, ["Hello"]);
document.getElementById("demo").innerHTML = msg;
</script>
</body>
</html>
```

Reflect.construct()

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Arrays</h1>
```

```
<h2>The new Constructor</h2>
<p id="demo"></p>
<script>
const colors = new Array(["red", "green", "blue"]);
document.getElementById("demo").innerHTML = colors;
</script>
</body>
</html>
```

Reflect.defineProperty()

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Object</h1>
<h2>The defineProperty() Method</h2>
<p id="demo"></p>
<script>
// Create an Object
const user = {};
// Add a Property
Object.defineProperty(user, "id", {
  value: 123,
  writable: false
});
document.getElementById("demo").innerHTML = "User id is " + user.id;
</script>
</body>
</html>
```

Reflect.ownKeys()

```
<!DOCTYPE html>
<html>
<body>
<h1>The Reflect Object</h1>
<h2>The ownKeys() Method</h2>
<p id="demo"></p>
<script>
const sym = Symbol("secret");
const obj = { a: 1, [sym]: 2 };
```

```
let keys = Object.keys(obj);
let text = "Object keys: <br>";
keys.forEach(myFunction);
function myFunction(value, index, array) {
  let typ = typeof array[index];
  let add = typ + " ";
  if (typ != "symbol") {
    add += value;
  } else {
    add += "secret"
  };
  text += add + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

When to Use Reflect?

Case	Use	Why
Getting / setting values	Yes	Consistent return values
Creating objects like new	Yes	Reflect.construct() Proxy ke saath work karta hai
Calling a function with context	Yes	Reflect.apply() function call ko cleaner banata hai
Meta programming	Yes	Low-level tasks ke liye design kiya gaya hai
Simple object work	No	Normal JS syntax use karo

Reflect with Proxy (Very Common)

- Read a property → property ko access karte waqt intercept karna
- Write a property → property ko set/update karte waqt intercept karna
- Delete a property → property delete karte waqt intercept karna
- Check property existence → in ya similar checks ko intercept karna

```
<!DOCTYPE html>
<html>
<body>
```

```
<h1>JavaScript Proxy</h1>
<h2>Using Reflect</h2>
<p>Log changes to all property values:</p>
<p id="demo"></p>
<script>
// Create an Object
const user = { name: "Jan", age: 40 };
// Create a Proxy
const proxy = new Proxy(user, {
  set(target, property, value) {
    // safe forwarding
    log(property + ": " + value);
    return Reflect.set(target, property, value);
  }
});
function log(message) {
  const time = new Date().toLocaleTimeString();
  document.getElementById("demo").innerHTML += "[" + time + "] " + message + "<br>";
}
proxy.name = "John";
proxy.age = 45;
proxy.name = "Paul";
</script>
</body>
</html>
```

Proxy and Reflect

Order of design:

- Proxy → pehle design hua
- Reflect → baad mein design hua

Proxy ko design karne ka purpose:

- Property access ko intercept karna
- Default property behaviors ko override karna
- Data validate karna
- Functions ko wrap karna
- Objects ko virtualize karna
- Reactive objects create karna

JavaScript Reflect History

Problems before Reflect:

- delete, in, new sirf operators the
- Kuch operations errors throw karte the
- Object operations inconsistent values return karte the
- Generic property access ka koi tariqa nahi tha
- Constructor calls ko safely forward karne ka way nahi tha
- Yeh sab Proxy ko safely implement karna mushkil banata tha

ES6 ne Reflect introduce kiya to:

- Har internal operation ka clean, function-based version provide karna
- Predictable booleans return karna instead of objects
- Unnecessary errors avoid karna
- JavaScript engine ke behavior ko mirror karna
- Proxy traps ke liye reliable forward-path provide karna

What Can a Proxy Do?

- Property read karna → get
- Property set karna → set
- Property delete karna → deleteProperty
- Property existence check karna → has
- Function call karna → apply
- Object construct karna → construct

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Proxy</h1>
<h2>Using Reflect</h2>
<p>Log changes to all property values:</p>
<p id="demo"></p>
<script>
// Create an Object
const user = { name: "Jan", age: 40 };
// Create a Proxy
const proxy = new Proxy(user, {
  set(target, property, value) {
    // safe forwarding
    log(property + ": " + value);
```

```
    return Reflect.set(target, property, value);
  }
});
function log(message) {
  const time = new Date().toLocaleTimeString();
  document.getElementById("demo").innerHTML += "[" + time + "] " + message + "<br>";
}
proxy.name = "John";
proxy.age = 45;
proxy.name = "Paul";
</script>
</body>
</html>
```

Proxy with Reflect (Most Common)

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Proxy</h1>
<h2>Using Reflect</h2>
<p>Log changes to all property values:</p>

<p id="demo"></p>
<script>
// Create an Object
const user = { name: "Jan", age: 40 };
// Create a Proxy
const proxy = new Proxy(user, {
  set(target, property, value) {
    // safe forwarding
    log(property + ": " + value);
    return Reflect.set(target, property, value);
  }
});
function log(message) {
  const time = new Date().toLocaleTimeString();
  document.getElementById("demo").innerHTML += "[" + time + "] " + message + "<br>";
}
proxy.name = "John";
proxy.age = 45;
```

```
proxy.name = "Paul";
</script>
</body>
</html>
```

Why Proxies

Proxies aapko allow karte hain ki aap:

- Logging add karo (property access ya changes track karne ke liye)
- Changes validate karo (data validation)
- Reactive systems create karo (jaise Vue.js)
- Properties auto-generate karo
- Sensitive data protect karo
- Virtual ya computed objects create karo
- Function calls intercept karo

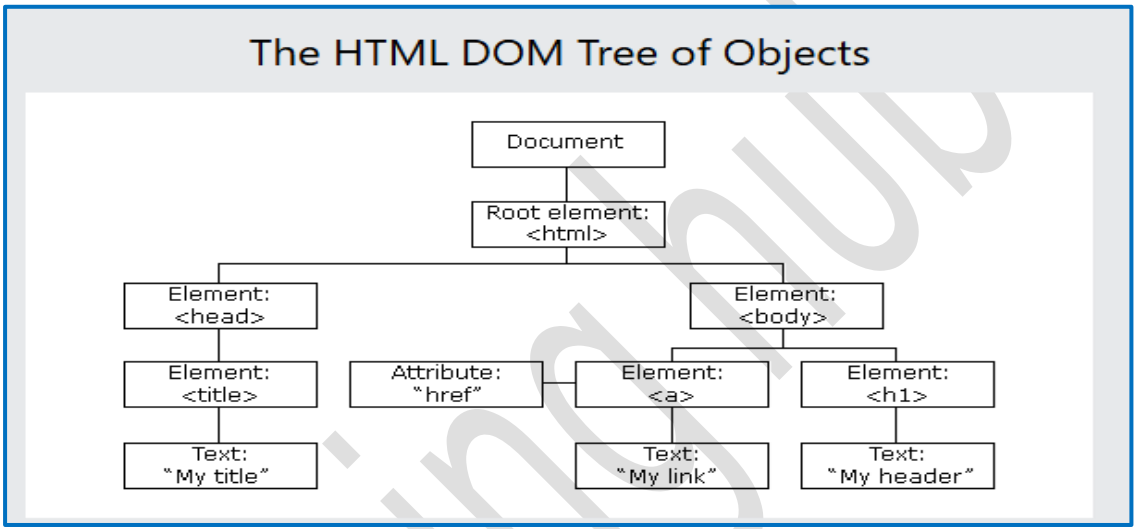
Proxy Traps

Trap Name	Triggered When
get	Jab property read ki ja rahi ho
set	Jab property assign/set ki ja rahi ho
has	in operator use karte waqt
deleteProperty	Jab property delete ki ja rahi ho
apply	Jab function call ho rahi ho
construct	Jab object create ho raha ho (new)
getOwnPropertyDescriptor	Jab property descriptor retrieve kiya ja raha ho
defineProperty	Jab nayi property define ki ja rahi ho
getPrototypeOf	Jab prototype retrieve kiya ja raha ho
setPrototypeOf	Jab prototype set kiya ja raha ho
isExtensible	Jab object extensibility check ki ja rahi ho
preventExtensions	Jab object ko extend karne se roka ja raha ho
ownKeys	Jab properties list ki ja rahi ho
hasOwn	Jab property existence check ki ja rahi ho

HTML DOM (Document Object Model)

Definition: HTML DOM ek programming interface hai jo HTML documents ko tree-like structure me represent karta hai.

- Purpose: Isse aap JavaScript ke through web page ke elements ko access, modify, add, ya delete kar sakte ho.
- Structure:
  - Document → Root node
  - Elements → Nodes (like , <p>, <h1>)
  - Attributes → Properties of elements (like id, class)
  - Text → Content inside elements



What is the DOM?

- DOM ka full form hai Document Object Model.
- Ye W3C (World Wide Web Consortium) standard hai.
- DOM define karta hai ek standard document access aur manipulation ke liye:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

W3C DOM standard ke 3 parts:

1. **Core DOM** → Standard model for all document types
2. **XML DOM** → Standard model for XML documents
3. **HTML DOM** → Standard model for HTML documents

Here's a **bullet-only version in Hinglish**:

DOM Programming Interface:

- HTML DOM ko JavaScript aur other languages se access kiya ja sakta hai
- Sab HTML elements objects ke form me hote hain
- Har object ke properties aur methods hote hain

Properties:

- Values jo get ya set ki ja sakti hain
- Example: textContent, innerHTML, value

Methods:

- Actions jo perform ki ja sakti hain
- Example: createElement(), appendChild(), removeChild(), getElementById()

```
<!DOCTYPE html>
<html>
<body>
<h2>My First Page</h2>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
</body>
</html>
```

The getElementById Method:

- Sabse common way hai HTML element access karne ka using its id
- getElementById method element ke unique id ke through element ko find karta hai

```
<!DOCTYPE html>
<html>
 <head>
   <title>getElementById Example</title>
  </head>
  <body>
   <p id="demo">Hello World!</p>
    <script>
      // Accessing the element with id="demo"
      const element = document.getElementById("demo");
      // Changing the content of the paragraph
 element.textContent = "Hi there!";
    </script>
  </body>
</html>
```

The innerHTML Property:

- Sabse easy way hai element ka content get ya change karne ka
- innerHTML property HTML element ka content get karne ya replace karne ke liye use hoti hai

```
<!DOCTYPE html>
<html>
```

```
<head>
 <title>innerHTML Example</title>
 </head>
 <body>
  <p id="demo">Hello World!</p>
   <script>
     // Access the element with id="demo"
     const element = document.getElementById("demo");
     // Get the current content of the paragraph
     console.log(element.innerHTML); // Output: Hello World!
     // Replace the content of the paragraph
     element.innerHTML = "Hi there!";
   </script>
 </body>
</html>
```

The HTML DOM document Object:

- document object aapke web page ko represent karta hai
- Agar aap kisi bhi HTML element ko access karna chahte ho, hamesha document se start karte ho
- document object ke through aap elements ko access aur manipulate kar sakte ho

Finding HTML Elements

Method	Description
document.getElementById(id)	Element ko uske id se find karna
document.getElementsByTagName(name)	Elements ko tag name ke through find karna
document.getElementsByClassName(name)	Elements ko class name ke through find karna

Changing HTML Elements

Name / Usage	Description
element.innerHTML = new html content	Element ka inner HTML change karna
element.attribute = new value	HTML element ka attribute value change karna
element.style.property = new style	HTML element ka style change karna
element.setAttribute(attribute, value)	HTML element ka attribute value change karna (method)

Adding and Deleting Elements

Method	Description
document.createElement(element)	Naya HTML element create karna
document.removeChild(element)	HTML element ko remove karna
document.appendChild(element)	HTML element ko add karna
document.replaceChild(new, old)	HTML element ko replace karna
document.write(text)	HTML output stream me text write karna

Adding Events Handlers

Method / Usage	Description
document.getElementById(id).onclick = function(){code}	onclick event ke liye event handler code add karna

Finding HTML Objects

Property	Description	DOM Level
document.anchors	Deprecated. Use mat karo	1
document.applets	Deprecated. Use mat karo	1
document.baseURI	Document ka absolute base URI return karta hai	3
document.body	<body> element return karta hai	1
document.cookie	Document ka cookie return karta hai	1
document.doctype	Document ka doctype return karta hai	3
document.documentElement	<html> element return karta hai	3
document.documentMode	Browser ka mode return karta hai	3
document.documentURI	Document ka URI return karta hai	3
document.domain	Deprecated. Use mat karo	1
document.domConfig	Deprecated. Use mat karo	3
document.embeds	Saare <embed> elements return karta hai	3
document.forms	Saare <form> elements return karta hai	1

Property	Description	DOM Level
document.head	<head> element return karta hai	3
document.images	Saare <img> elements return karta hai	1
document.implementation	DOM implementation return karta hai	3
document.inputEncoding	Document ka encoding return karta hai	3
document.lastModified	Document ke last update ka date/time return karta hai	3
document.links	Saare <a> aur <area> elements with href return karta hai	1
document.readyState	Document ka loading status return karta hai	3
document.referrer	Linking document ka URI return karta hai	1
document.scripts	Saare <script> elements return karta hai	3
document.strictErrorChecking	Error checking enforced hai ya nahi return karta hai	3
document.title	<title> element return karta hai	1
document.URL	Document ka complete URL return karta hai	1

Finding HTML Elements

- By ID: document.getElementById("id") → unique element access
- By Tag Name: document.getElementsByTagName("tag") → elements list access
- By Class Name: document.getElementsByClassName("class") → elements list access
- By CSS Selectors: document.querySelector("selector") or document.querySelectorAll("selector") → single ya multiple elements access
- By HTML Object Collections: Access elements via document.forms, document.images, document.links, etc.

Finding HTML Element by Id

- Sabse common way hai ek unique element access karne ka using its id
- Method: document.getElementById("id")
- Returns single element object

```
<!DOCTYPE html>
<html>
<body>
```

```
<h2>JavaScript HTML DOM</h2>
<p id="intro">Finding HTML Elements by Id</p>
<p>This example demonstrates the <b>getElementsById</b> method.</p>
<p id="demo"></p>
<script>
const element = document.getElementById("intro");
document.getElementById("demo").innerHTML =
"The text from the intro paragraph is: " + element.innerHTML;
</script>
</body>
</html>
```

Finding HTML Elements by Tag Name

- Method: document.getElementsByTagName("tag")
- Returns HTMLCollection of all elements with the specified tag
- Useful when you want to manipulate multiple elements of the same type

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript HTML DOM</h2>

<p>Finding HTML Elements by Tag Name.</p>
<p>This example demonstrates the <b>getElementsByTagName</b> method.</p>
<p id="demo"></p>
<script>
const element = document.getElementsByTagName("p");
document.getElementById("demo").innerHTML = 'The text in first paragraph (index 0) is: ' +
element[0].innerHTML;
</script>
</body>
</html>
```

Finding HTML Elements by Class Name

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript HTML DOM</h2>
<p>Finding HTML Elements by Class Name.</p>
<p class="intro">Hello World!</p>
<p class="intro">This example demonstrates the <b>getElementsByClassName</b>
method.</p>
```

```
<p id="demo"></p>
<script>
const x = document.getElementsByClassName("intro");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro" is: ' + x[0].innerHTML;
</script>
</body>
</html>
```

Finding HTML Elements by CSS Selectors:

- Method:
- document.querySelector("selector") → first matching element
- document.querySelectorAll("selector") → all matching elements
- Supports CSS selectors like #id, .class, tag, [attribute=value], etc.
- Returns NodeList (can loop through using forEach or index)

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript HTML DOM</h2>

<p>Finding HTML Elements by Query Selector</p>
<p class="intro">Hello World!.</p>
<p class="intro">This example demonstrates the <b>querySelectorAll</b> method.</p>
<p id="demo"></p>
<script>
const x = document.querySelectorAll("p.intro");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro" is: ' + x[0].innerHTML;
</script>
</body>
</html>
```

Finding HTML Elements by HTML Object Collections

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript HTML DOM</h2>
```

```
<p>Finding HTML Elements Using <b>document.forms</b>.</p>
<form id="frm1" action="/action_page.php">
  First name: <input type="text" name="fname" value="Donald"><br>
  Last name: <input type="text" name="lname" value="Duck"><br><br>
  <input type="submit" value="Submit">
</form>
<p>These are the values of each element in the form:</p>
<p id="demo"></p>
<script>
const x = document.forms["frm1"];
let text = "";
for (let i = 0; i < x.length ;i++) {
  text += x.elements[i].value + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

Changing HTML Content

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript can Change HTML</h2>
<p id="p1">Hello World!</p>
<script>
document.getElementById("p1").innerHTML = "New text!";
</script>
<p>The paragraph above was changed by a script.</p>
</body>
</html>
```

Changing the Value of an Attribute

- HTML element ke attributes ko change karne ke liye DOM properties ya setAttribute() method use karte hain
- Commonly changed attributes: src, href, id, class, value, etc.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript HTML DOM</h2>
```

```
<img id="image" src="smiley.gif" width="160" height="120">
<script>
document.getElementById("image").src = "landscape.jpg";
</script>
<p>The original image was smiley.gif, but the script changed it to landscape.jpg</p>
</body>
</html>
```

Dynamic HTML content

- Dynamic content ka matlab hai: HTML content ko runtime pe JavaScript ke through change karna
- Isse page reload kiye bina content update ho sakta hai
- Common methods/properties:
- innerHTML → HTML content update karna
- textContent / innerText → plain text update karna
- setAttribute() → attributes dynamically change karna

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Date : " + Date();
</script>
</body>
</html>
```

document.write() Method:

- document.write() se aap directly HTML output page me likh sakte ho
- Mostly page load time pe kaam karta hai
- Agar page load ke baad use kiya jaaye, toh wo existing content ko replace kar deta hai

```
<!DOCTYPE html>
<html>
<body>
<p>Bla, bla, bla</p>
<script>
document.write(Date());
</script>
<p>Bla, bla, bla</p>
</body>
```

```
</html>
```

JavaScript Form Validation:

- Form validation ka matlab hai: user input ko check karna before submitting the form
- Ensure karta hai ki required fields filled ho, proper format ho, aur invalid data na ho
- Done client-side using JavaScript for faster feedback

```
<!DOCTYPE html>
<html>
<head>
<script>
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
</script>
</head>
<body>
<h2>JavaScript Validation</h2>
<form name="myForm" action="/action_page.php" onsubmit="return validateForm()"
method="post">
  Name: <input type="text" name="fname">
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

Automatic HTML Form Validation

- HTML5 me kuch built-in form validation attributes diye gaye hain
- Browser automatically check karta hai user input without JavaScript
- Common attributes:
- required → field must be filled
- type="email" → valid email format
- type="number" → numeric input
- min, max → number range
- pattern → regex pattern match

- maxlength / minlength → character length limit

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Validation</h2>
<form action="/action_page.php" method="post">
  <input type="text" name="fname" required>
  <input type="submit" value="Submit">
</form>
<p>If you click submit, without filling out the text field,
your browser will display an error message.</p>
</body>
</html>
```

Data Validation (JavaScript / Web Forms):

- Definition: Ye process hai jisme ensure kiya jaata hai ki user ka input clean, correct, aur useful ho
- Purpose: Ye confirm karta hai ki user ne jo data submit kiya hai wo valid aur safe ho

Typical Validation Tasks:

- Check karo ki sab required fields fill kiye gaye hain ya nahi
- Check karo ki user ne valid date enter ki hai ya nahi
- Check karo ki user ne text numeric field me nahi dala
- Check karo ki input ka format sahi hai ya nahi (email, phone number, password rules)

Key Points:

- Validation ensure karta hai ki user input correct ho before processing
- Ye kiya ja sakta hai:
  - HTML attributes (required, type, pattern)
  - JavaScript functions (onsubmit, onchange)
  - Server-side validation security ke liye

HTML Constraint Validation:

- HTML5 me form elements ke liye built-in validation rules diye gaye hain
- Ye automatically user input ko check karte hain without JavaScript
- Ye rules ko constraints kehte hain

Constraint Validation HTML Input Attributes

Attribute	Description
disabled	Specify karta hai ki input element disable ho

Attribute	Description
max	Input element ka maximum value specify karta hai
min	Input element ka minimum value specify karta hai
pattern	Input element ka value pattern specify karta hai (regex)
required	Specify karta hai ki input field ko fill karna zaroori hai
type	Input element ka type specify karta hai (text, email, number, etc.)

Constraint Validation CSS Pseudo Selectors

Selector	Description
:disabled	Select karta hai sab input elements jo "disabled" hain
:invalid	Select karta hai sab input elements jin ki values invalid hain
:optional	Select karta hai sab input elements jinke paas "required" nahi hai
:required	Select karta hai sab input elements jinke paas "required" attribute hai
:valid	Select karta hai sab input elements jinke values valid hain

Changing HTML Style

HTML elements ke styles dynamically change karne ke liye style property use hoti hai

- CSS properties ko camelCase me likhna padta hai (e.g., backgroundColor, fontSize

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript HTML DOM</h2>
<p>Changing the HTML style:</p>
<p id="p1">Hello World!</p>
<p id="p2">Hello World!</p>
<script>
document.getElementById("p2").style.color = "blue";
document.getElementById("p2").style.fontFamily = "Arial";
document.getElementById("p2").style.fontSize = "larger";
</script>
</body>
</html>
```

Using Events

- HTML DOM se aap code execute kar sakte ho jab koi event occur ho
- Events browser generate karta hai jab “kuch hota hai” HTML elements ke saath

Common Events:

- Element pe click (onclick)
- Page load (onload)
- Input change (onchange)
- Mouse over / out (onmouseover, onmouseout)

```
<!DOCTYPE html>
<html>
<body>
<h1 id="id1">My Heading 1</h1>
<button type="button"
onclick="document.getElementById('id1').style.color = 'red'">
Click Me!</button>
</body>
</html>
```

JavaScript HTML DOM Animation

- JavaScript ke through aap HTML elements ko animate kar sakte ho
- Animation ka matlab hai element ka position, size, color, opacity, etc. dynamically change karna
- Commonly setInterval() ya requestAnimationFrame() use kiya jaata hai for smooth animations

```
<!DOCTYPE html>
<html>
<style>
#container {
  width: 400px;
  height: 400px;
  position: relative;
  background: yellow;
}
#animate {
  width: 50px;
  height: 50px;
  position: absolute;
  background-color: red;
}
</style>
```

```
<body>
<p><button onclick="myMove()">Click Me</button></p>
<div id ="container">
  <div id ="animate"></div>
</div>
<script>
function myMove() {
  let id = null;
  const elem = document.getElementById("animate");
  let pos = 0;
  clearInterval(id);
  id = setInterval(frame, 5);
  function frame() {
    if (pos == 350) {
      clearInterval(id);
    } else {
      pos++;
      elem.style.top = pos + "px";
      elem.style.left = pos + "px";
    }
  }
}
</script>
</body>
</html>
```

JavaScript HTML DOM Events:

- HTML DOM events allow you to execute code when something happens in the browser
- Events are triggered when user interacts with HTML elements or when the browser state changes

Common Types of Events:

1. **Mouse Events:**
   - onclick → element pe click hone par
   - ondblclick → element pe double click hone par
   - onmouseover → mouse element ke upar move kare
   - onmouseout → mouse element se bahar jaaye
2. **Keyboard Events:**
   - onkeydown → key press karte hi
   - onkeyup → key release hone par

   - onkeypress → key press hone par
3. **Form Events:**
   - onsubmit → form submit hone par
   - onchange → input field change hone par
   - onfocus → input field focus hone par
   - onblur → input field focus chhodne par
4. **Window / Document Events:**
   - onload → page load hone par
   - onresize → window resize hone par
   - onscroll → page scroll hone par

Reacting to Events:

- JavaScript me aap event ko detect karke uske response me code run kar sakte ho
- Is process ko kehte hain reacting to events
- Event reaction ka matlab hai: user ka interaction (click, input, hover) ya browser state change hone par action lena

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript HTML Events</h1>
<h2>The onclick Attribute</h2>
<h2 onclick="changeText(this)">Click on this text!</h2>
<script>
function changeText(id) {
  id.innerHTML = "Ooops!";
}
</script>
</body>
</html>
```

HTML Event Attributes:

- HTML elements ke andar aap directly event handlers define kar sakte ho
- Ye attributes browser ko batate hain ki event hon

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript HTML Events</h1>
<h2>The onclick Attribute</h2>
<p>Click the button to display the date.</p>
<button onclick="displayDate()">The time is?</button>
<script>
function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>
<p id="demo"></p>
</body>
</html>
```

Assign Events Using the HTML DOM:

- HTML DOM se aap JavaScript ke through events assign kar sakte ho
- Ye HTML attributes ke bina event handling allow karta hai
- Do main ways hain: DOM property aur addEventListener()

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript HTML Events</h1>
<h2>The onclick Events</h2>
<p>Click "Try it" to execute the displayDate() function.</p>
<button id="myBtn">Try it</button>
<p id="demo"></p>
<script>
document.getElementById("myBtn").onclick = displayDate;
function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>
</body>
</html>
```

The onload and onunload Events

The onload and onunload Events:

- Ye events window ya HTML elements ke load/unload hone par trigger hote hain
- onload → page ya element fully load hone par code run hota hai
- onunload → page ya element close ya leave hone par code run hota hai

```
<!DOCTYPE html>
<html>
<body onload="checkCookies()">
<h1>JavaScript HTML Events</h1>
<h2>The onload Attribute</h2>
<p id="demo"></p>
<script>
function checkCookies() {
  let text = "";
  if (navigator.cookieEnabled == true) {
    text = "Cookies are enabled.";
  } else {
    text = "Cookies are not enabled.";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>
</body>
</html>
```

The oninput and onchange Event

- oninput event tab trigger hota hai jab user input field me value change karta hai
- Difference onchange se:
- onchange → tab trigger hota hai jab user input field se bahar jaata hai
- oninput → value change hote hi trigger hota hai

```
<!DOCTYPE html>
<html>
<head>
  <title>onchange vs oninput Example</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
    }
```

```
    input {
      padding: 8px;
      margin: 10px 0;
      width: 300px;
      border: 2px solid #ccc;
      border-radius: 4px;
      font-size: 16px;
    }
    p {
      font-weight: bold;
      color: #333;
    }
    .oninput-output {
      color: green;
    }
    .onchange-output {
      color: blue;
    }
  </style>
</head>
<body>
  <h2>oninput vs onchange Example</h2
  <label for="input1">oninput (value changes instantly):</label><br>
  <input type="text" id="input1" placeholder="Type here...">
  <p class="oninput-output" id="output1">Your text will appear here instantly</p>
  <label for="input2">onchange (value updates on blur):</label><br>
  <input type="text" id="input2" placeholder="Type here...">
  <p class="onchange-output" id="output2">Your text will appear here after leaving the
input</p>
  <script>
    const input1 = document.getElementById("input1");
    const output1 = document.getElementById("output1")
    const input2 = document.getElementById("input2");
    const output2 = document.getElementById("output2");
    // oninput → value changes instantly
    input1.oninput = function() {
      output1.textContent = "You typed: " + input1.value;
    };
    // onchange → value updates when input loses focus
    input2.onchange = function() {
      output2.textContent = "You typed: " + input2.value;
```

```
  };
 </script>
</body>
</html>
```

The onmouseover and onmouseout Events:

- Ye events **mouse interaction** ke liye use hote hain
- **onmouseover** → tab trigger hota hai **jab mouse element ke upar move kare**
- **onmouseout** → tab trigger hota hai **jab mouse element se bahar jaaye**
- Useful for **hover effects, tooltips, interactive elements**

```
<!DOCTYPE html>
<html>
<head>
 <title>onmouseover & onmouseout Example</title>
 <style>
  #box {
   width: 200px;
   height: 100px;
   background-color: lightblue;
   text-align: center;
   line-height: 100px;
   font-weight: bold;
   font-size: 18px;
   border-radius: 8px;
   transition: background-color 0.3s, color 0.3s;
   cursor: pointer;
  }
 </style>
</head>
<body>
 <h2>Mouse Hover Example</h2>
 <div id="box">Hover over me!</div>
 <script>
  const box = document.getElementById("box");
  // Mouse enters element
  box.onmouseover = function() {
   box.style.backgroundColor = "orange";
   box.style.color = "white";
   box.textContent = "Mouse is over me!";
```

```
  }
  // Mouse leaves element
  box.onmouseout = function() {
    box.style.backgroundColor = "lightblue";
    box.style.color = "black";
    box.textContent = "Hover over me!";
  };
 </script>
</body>
</html>
```

Mouse Events in JavaScript HTML DOM:

1. onmousedown → tab trigger hota hai jab mouse button press hota hai on an element
2. onmouseup → tab trigger hota hai jab mouse button release hota hai on an element
3. onclick → tab trigger hota hai jab mouse button press aur release dono complete ho jaaye on an element

```
<!DOCTYPE html>
<html>
<head>
 <title>Mouse Events Example</title>
 <style>
  #btn {
   padding: 15px 30px;
   font-size: 18px;
   background-color: lightblue;
   border: none;
   border-radius: 8px;
   cursor: pointer;
   transition: background-color 0.3s;
  }
 </style>
</head>
<body>
 <h2>Mouse Events Example</h2>
 <button id="btn">Click Me</button>
 <p id="output">Event Output</p>
 <script>
  const button = document.getElementById("btn");
  const output = document.getElementById("output");
  button.onmousedown = function() {
   output.textContent = "Mouse button pressed (onmousedown)";
```

```
    button.style.backgroundColor = "orange";
  };
  button.onmouseup = function() {
    output.textContent = "Mouse button released (onmouseup)";
    button.style.backgroundColor = "lightgreen";
  };
  button.onclick = function() {
    output.textContent = "Button clicked (onclick)";
    button.style.backgroundColor = "lightblue";
  };
 </script>
</body>
</html>
```

JavaScript HTML DOM EventListener

Definition:

- addEventListener() method se aap event ko element pe attach kar sakte ho
- Modern aur recommended way hai HTML DOM events handle karne ka
- Allows multiple event listeners on same element

The addEventListener() Method

Definition:

- addEventListener() method se aap HTML elements pe multiple events attach kar sakte ho
- Recommended method for modern JavaScript event handling
- Keeps HTML clean (no inline event attributes needed)

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript addEventListener()</h2>
<p>This example uses the addEventListener() method to attach a click event to a button.</p>
<button id="myBtn">Try it</button>
<p id="demo"></p>
<script>
document.getElementById("myBtn").addEventListener("click", displayDate);
function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>
</body>
```

```
</html>
```

Add an Event Handler to an Element

- Event handler ek function hota hai jo execute hota hai jab specific event occur hota hai
- JavaScript me event handler assign karne ke 2 main ways hain:
    1. Using DOM property (element.onclick = function(){...})
    2. Using addEventListener() (recommended)

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript addEventListener()</h2>
<p>This example uses the addEventListener() method to execute a function when a user clicks on a button.</p>
<button id="myBtn">Try it</button
<script>
document.getElementById("myBtn").addEventListener("click", myFunction);

function myFunction() {
  alert ("Hello World!");
}
</script>
</body>
</html>
```

Add Many Event Handlers to the Same Element

- Ek element pe ek se zyada events assign karna hota hai
- Agar aap DOM property (element.onclick) use karte ho, to pehla event overwrite ho jaata hai
- addEventListener() se multiple events attach kiye ja sakte hain without overwriting

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript addEventListener()</h2
<p>This example uses the addEventListener() method to add many events on the same button.</p>
```

```
<button id="myBtn">Try it</button>
<p id="demo"></p
<script>
var x = document.getElementById("myBtn");
x.addEventListener("mouseover", myFunction);
x.addEventListener("click", mySecondFunction);
x.addEventListener("mouseout", myThirdFunction);
function myFunction() {
  document.getElementById("demo").innerHTML += "Moused over!<br>";
}
function mySecondFunction() {
  document.getElementById("demo").innerHTML += "Clicked!<br>";
}

function myThirdFunction() {
  document.getElementById("demo").innerHTML += "Moused out!<br>";
}
</script>
</body>
</html>
```

Add an Event Handler to the Window Object

- window object represents the browser window
- Aap window pe events attach kar sakte ho jaise:
- Page load (load)
- Page unload (beforeunload / unload)
- Resize (resize)
- Scroll (scroll)
- Event handler assign karne ke 2 main ways hain:
- Using DOM property (window.onload = function(){})
- Using addEventListener() (recommended)

```
<!DOCTYPE html>
<html>
<head>
  <title>Window Event Handlers Example</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      margin: 50px;
```

```
  }
  h1 {
    color: #333;
  }
  p {
    font-weight: bold;
    color: #555;
  }
 </style>
</head>
<body>
 <h1>Window Event Handlers Example</h1>
 <p id="output">Event Output</p>

 <script>
  const output = document.getElementById("output");

  // 1. Using DOM property: onload
  window.onload = function() {
    output.textContent = "Page has fully loaded (onload)!";
  };

  // 2. Using addEventListener: load
  window.addEventListener("load", function() {
    console.log("Page loaded via addEventListener!");
  });

  // 3. Using addEventListener: resize
  window.addEventListener("resize", function() {
    output.textContent = "Window resized to " + window.innerWidth + " x " +
window.innerHeight;
  });

  // 4. Using addEventListener: scroll
  window.addEventListener("scroll", function() {
    output.textContent = "You scrolled! Scroll position: " + window.scrollY + "px";
  });

  // 5. Using addEventListener: beforeunload
  window.addEventListener("beforeunload", function(e) {
    e.preventDefault(); // Some browsers require this
```

```
    e.returnValue = ''; // Message may or may not appear
  });
 </script>
 <div style="height: 1500px;"></div> <!-- Just to enable scrolling -->
</body>
</html>
```

Passing Parameters

Jab hum function banate hain, hum input values ko function ke andar bhejte hain, taki function un values ke saath kaam kar sake. Ye input values ko parameters ya arguments kehte hain.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript addEventListener()</h2>
<p>This example demonstrates how to pass parameter values when using the addEventListener()
method.</p>
<p>Click the button to perform a calculation.</p>
<button id="myBtn">Try it</button>
<p id="demo"></p>
<script>
let p1 = 5;
let p2 = 7;
document.getElementById("myBtn").addEventListener("click", function() {
 myFunction(p1, p2);
});
function myFunction(a, b) {
 document.getElementById("demo").innerHTML = a * b;
}
</script>
</body>
</html>
```

Event Bubbling (Bubble Up)

- Kya hota hai: Jab kisi child element pe event trigger hota hai (jaise click), wo event sabse pehle child element pe handle hota hai, phir parent elements tak upar ki taraf propagate hota hai.
- Order: Child → Parent → Grandparent → Body → Document

Event Capturing (Capture Phase)

- Kya hota hai: Isme event parent se start hota hai aur child tak propagate hota hai. Matlab event upar se neeche ki taraf jata hai.

- Order: Document → Body → Grandparent → Parent → Child

```
<!DOCTYPE html>
<html>
<head>
<style>
#myDiv1, #myDiv2 {
  background-color: coral;
  padding: 50px;
}
#myP1, #myP2 {
  background-color: white;
  font-size: 20px;
  border: 1px solid;
  padding: 20px;
}
</style>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
</head>
<body>
<h2>JavaScript addEventListener()</h2>
<div id="myDiv1">
  <h2>Bubbling:</h2>
  <p id="myP1">Click me!</p>
</div><br>
<div id="myDiv2">
  <h2>Capturing:</h2>
  <p id="myP2">Click me!</p>
</div>
<script>
document.getElementById("myP1").addEventListener("click", function() {
  alert("You clicked the white element!");
}, false);
document.getElementById("myDiv1").addEventListener("click", function() {
  alert("You clicked the orange element!");
}, false);
document.getElementById("myP2").addEventListener("click", function() {
  alert("You clicked the white element!");
}, true);
document.getElementById("myDiv2").addEventListener("click", function() {
  alert("You clicked the orange element!");
}, true);
</script>
</body>
</html>
```

removeEventListener()

Ye method use hoti hai event listener ko remove karne ke liye, jo pehle addEventListener() ke through add kiya gaya tha.

- Matlab agar aap chahte ho ki kisi element par event aur respond na kare, to aap ye method use karte ho.

```
<!DOCTYPE html>
<html>
<head>
<style>
#myDIV {
  background-color: coral;
  border: 1px solid;
  padding: 50px;
  color: white;
  font-size: 20px;
}
</style>
</head>
<body>
<h2>JavaScript removeEventListener()</h2>
<div id="myDIV">
  <p>This div element has an onmousemove event handler that displays a random number every time you move your mouse inside this orange field.</p>
  <p>Click the button to remove the div's event handler.</p>
  <button onclick="removeHandler()" id="myBtn">Remove</button>
</div>
<p id="demo"></p>
<script>
document.getElementById("myDIV").addEventListener("mousemove", myFunction);
function myFunction() {
  document.getElementById("demo").innerHTML = Math.random();
}
function removeHandler() {
  document.getElementById("myDIV").removeEventListener("mousemove", myFunction);
}
</script>
</body>
</html>
```

DOM Navigation

DOM (Document Object Model) ke andar, har HTML element tree structure me hota hai.
DOM Navigation ka matlab hai ek element se dusre element tak travel karna — parent, child, sibling, etc.

DOM Nodes

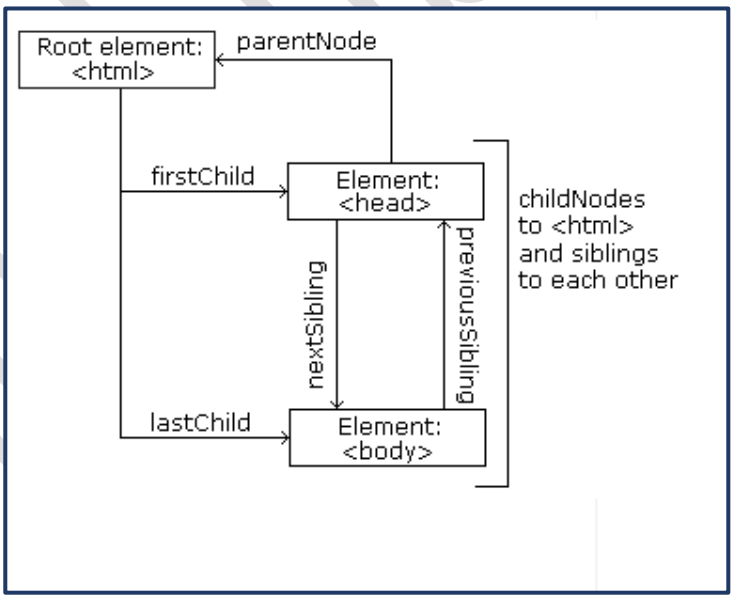
Document Node
- Represents the entire HTML document.
- Access via document object.
- nodeType = 9

Element Node
- Har HTML tag ek element node hota hai.
- Examples: , <p>, <a>
- nodeType = 1

Text Node
- HTML element ke andar jo text hota hai, wo text node hai.
- nodeType = 3
- Access text via nodeValue

Attribute Node *(deprecated)*
- Har HTML attribute ek attribute node tha.
- Example: id="myDiv"
- Modern JS me element.getAttribute() ya element.property use karna better hai.
- nodeType = 2

Comment Node
- HTML comments bhi node hote hain.
- Example: <!-- This is a comment -->
- nodeType = 8
- Access comment text via nodeValue



Navigating Between Nodes

Nodes ke Beech Navigate Karna
JavaScript me aap nodes ke beech navigate karne ke liye neeche diye gaye node properties ka use kar sakte ho:
- parentNode
- childNodes[nodenumber]
- firstChild
- lastChild
- nextSibling

- previousSibling

```
<!DOCTYPE html>
<html>
<body>
<h1 id="id01">My First Page</h1>
<p id="id02"></p>
<script>
document.getElementById("id02").innerHTML =
document.getElementById("id01").childNodes[0].nodeValue;
</script>
</body>
</html>
```

innerHTML

Is tutorial me hum innerHTML property ka use karte hain ek HTML element ka content retrieve karne ke liye.
Lekin, upar diye gaye other methods ko seekhna bhi useful hai, kyunki ye DOM ke tree structure aur navigation ko samajhne me help karta hai.

DOM Root Nodes

Do special properties hain jo poore document tak access deti hain:

- document.body – Document ka body
- document.documentElement – Poora document

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript HTMLDOM</h2>
<p>Displaying document.documentElement</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = document.documentElement.innerHTML;
</script>
</body>
</html>
```

The nodeName Property

nodeName property kisi node ka name specify karti hai.

- nodeName read-only hota hai
- Element node ka nodeName uska tag name hota hai
- Attribute node ka nodeName us attribute ka naam hota hai
- Text node ka nodeName hamesha #text hota hai
- Document node ka nodeName hamesha #document hota hai

```
<!DOCTYPE html>
<html>
<body>
<h1 id="id01">My First Page</h1>
<p id="id02"></p>
<script>
document.getElementById("id02").innerHTML = document.getElementById("id01").nodeName;
</script>
</body>
</html>
```

nodeValue Property

nodeValue property kisi node ki value specify karti hai.

- **Element nodes ke liye nodeValue null hoti hai**
- **Text nodes ke liye nodeValue wahi text hota hai**
- **Attribute nodes ke liye nodeValue us attribute ki value hoti hai**

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Hello World</p>
<script>
// Element node
let elementNode = document.getElementById("demo");
console.log("Element nodeName:", elementNode.nodeName);   // Output: P
console.log("Element nodeValue:", elementNode.nodeValue); // Output: null
// Text node
let textNode = elementNode.childNodes[0];
console.log("Text nodeName:", textNode.nodeName);   // Output: #text
console.log("Text nodeValue:", textNode.nodeValue); // Output: Hello Worl
// Attribute node
let attrNode = elementNode.getAttributeNode("id");
console.log("Attribute nodeName:", attrNode.nodeName);   // Output: id
console.log("Attribute nodeValue:", attrNode.nodeValue); // Output: demo
</script>
</body>
</html>
```

nodeType Property

Node Type	Description	nodeType Number
Element Node	HTML tag jaise <p>,	1

Node Type	Description	nodeType Number
Attribute Node	HTML attribute jaise id="test"	2 (but modern DOM me rarely used)
Text Node	Text content	3
Comment Node	<!-- comment -->	8
Document Node	Pure document	9

```
<!DOCTYPE html>
<html>
<body
<h1 id="id01">My First Page</h1>
<p id="id02"></p>
<script>
document.getElementById("id02").innerHTML =
document.getElementById("id01").nodeType;
</script>
</body>
</html>
```

Property

Node	Type Number	Example / Explanation
ELEMENT_NODE	1	<h1 class="heading">W3Schools</h1> — Yeh ek HTML element node hai
ATTRIBUTE_NODE	2	class="heading" — Attribute node (ab deprecated mana jata hai)
TEXT_NODE	3	W3Schools — Element ke andar ka text
COMMENT_NODE	8	<!-- This is a comment --> — Yeh HTML comment node hai
DOCUMENT_NODE	9	Pure HTML document khud, jaise <html> ka parent
DOCUMENT_TYPE_NODE	10	<!DOCTYPE html> — Doctype declaration

Creating New HTML Elements (Nodes)

- document.createElement("tag") → naya HTML element banata hai
- document.createTextNode("text") → naya text node banata hai
- element.textContent = "text" → element ke andar text daalne ka easy tareeka

- element.appendChild(child) → element ke andar child node add karta hai
- parent.appendChild(newElement) → new element ko DOM me insert karta hai
- element.classList.add("className") → class add karta hai
- element.id = "someId" → ID set karta hai
- element.setAttribute("name", "value") → koi bhi attribute set karna
- parent.insertBefore(newElement, referenceElement) → specific jagah par insert karna
- element.remove() → element ko DOM se hataana
- element.innerHTML = "HTML" → element ke andar HTML add karna (use carefully)

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript HTML DOM</h2>
<p>Add a new HTML Element.</p>
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
<script>
const para = document.createElement("p");
const node = document.createTextNode("This is new.");
para.appendChild(node);
const element = document.getElementById("div1");
element.appendChild(para);
</script>
</body>
</html>
```

Creating new HTML Elements - insertBefore()

- insertBefore() DOM ka method hai jo naya element kisi specific element se pehle insert karta hai
- Syntax:
  parent.insertBefore(newElement, referenceElement)
- parent → jiske andar dono elements honge
- newElement → jo element aap insert karna chahte ho
- referenceElement → jiske pehle naya element add hoga
- Agar referenceElement null ho, to insertBefore() appendChild() jaisa behave karta hai (end me add karega)
- Pehle createElement() se naya element banao
- Phir text add karo (textContent ya createTextNode())

- Phir insertBefore() se specific jagah par insert karo

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript HTML DOM</h2>
<p>Add a new HTML Element.</p
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
<script>
const para = document.createElement("p");
const node = document.createTextNode("This is new.");
para.appendChild(node);
const element = document.getElementById("div1");
const child = document.getElementById("p1");
element.insertBefore(para,child);
</script>
</body>
</html>
```

Removing Existing HTML Elements

- remove() method kisi bhi HTML element ko directly DOM se hata deta hai
- Syntax:
  element.remove()
- Element ko remove karne ke liye parent element ki zaroorat nahi hoti, seedha element par remove() call hota hai
- Pehle element ko select karo (getElementById, querySelector, etc.)
- Phir us element par .remove() call kar do
- Element remove hone ke baad DOM me exist nahi karta
- remove() modern browsers me supported hai (IE me nahi)

Removing a Child Node

- Child node ko remove karne ke liye parent element ki zaroorat hoti hai
- Method: parent.removeChild(child)
- Syntax:
- parentElement.removeChild(childElement);
- parentElement → jiske andar child hai

- childElement → jo element remove karna hai
- Pehle child element ko select karo (getElementById, querySelector, etc.)
- removeChild() call karne ke baad child element DOM se hata diya jata hai
- Ye method older browsers me bhi supported hai

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript HTML DOM</h2>
<p>Remove Child Element</p>
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
<script>
const parent = document.getElementById("div1");
const child = document.getElementById("p1");
parent.removeChild(child);
</script>
</body>
</html>
```

JavaScript HTML DOM Collections

Common DOM Collections

- document.forms → page ke saare forms ka collection
- document.images → page ke saare images ka collection
- document.links → page ke saare links (a tags) ka collection
- document.scripts → page ke saare script tags ka collection
- document.anchors → page ke anchor elements (name attribute ke sath) ka collection

Element-specific Collections

- element.children → element ke direct child elements
- element.childNodes → element ke saare child nodes (text, comment, elements)
- element.getElementsByTagName("tag") → live collection of elements with given tag
- element.getElementsByClassName("class") → live collection of elements with given class
- element.getElementsByName("name") → live collection of elements with given name

```
<!DOCTYPE html>
<html>
<body>
```

```
<h2>JavaScript HTML DOM</h2>
<p>Hello World!</p>
<p>Hello Norway!</p>
<p>Click the button to change the color of all p elements.</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {
  const myCollection = document.getElementsByTagName("p");
  for (let i = 0; i < myCollection.length; i++) {
    myCollection[i].style.color = "red";
  }
}
</script>
</body>
</html>
```

NodeList Basics

- NodeList → JavaScript object jo nodes ka collection return karta hai
- Nodes = elements, text nodes, comment nodes, etc.
- NodeLists live ya static ho sakte hain (depending on method)
- NodeList array nahi hai, lekin for, for...of ya Array.from() se iterate kiya ja sakta hai

Kaunse methods NodeList return karte hain

- document.querySelectorAll("selector") → static NodeList
- element.childNodes → live NodeList (including text, comment nodes)
- element.querySelectorAll("selector") → static NodeList of matching elements

NodeList vs HTMLCollection

- NodeList → nodes ka general collection (text, comment, element)
- HTMLCollection → sirf element nodes, usually live
- NodeList ko index se access kar sakte hain: nodelist[0]
- length property dono me hoti hai: nodelist.length

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript HTML DOM</h2>
<p>Hello World!</p>
<p>Hello Norway!</p>
<p>Click the button to change the color of all p elements.</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {
  const myNodelist = document.querySelectorAll("p");
```

```
  for (let i = 0; i < myNodelist.length; i++) {
    myNodelist[i].style.color = "red";
  }
}
</script>
</body>
</html>
```

The Browser Object Model (BOM)

BOM ek aisa concept hai jisme browser apne features (jaise window, screen, location, history, navigator, etc.) ko JavaScript ke through access karne deta hai.
Chahe official standard nahi hain, par browsers inhe similar tarike se support karte hain.

The Window Object

Jo bhi cheezein aap bina kisi object ke directly JavaScript me likhte ho (global cheezein), woh sab window object ke andar store ho jaati hain.
Aur document, jisse hum HTML ko access/update karte hain, woh bhi window ka hi part hai.

```
<!DOCTYPE html>
<html>
<body>
<h2>Window Object Example</h2>
<script>
// Global variable
var myVar = 100;
// Global function
function myFunction() {
   console.log("This is a global function!");
}
// Accessing global variable through window
console.log(window.myVar); // 100
// Accessing global function through window
window.myFunction(); // "This is a global function!"
// Accessing document through window
console.log(window.document); // Shows the document object
</script>
</body>
</html>
```

Window Size

Browser window ka size jaanne ke liye JavaScript me do properties hoti hain:

1. window.innerHeight
   → Browser window ki inner height (upar se neeche tak), pixels me.
2. window.innerWidth
   → Browser window ki inner width (left se right tak), pixels me.

Ye dono values pixels me milti hain, aur window resize hone par change bhi ho jaati hain.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Window</h2>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"Browser inner window width: " + window.innerWidth + "px<br>" +
"Browser inner window height: " + window.innerHeight + "px";
</script>
</body>
</html>
```

Other Window Methods

JavaScript me window object ke kuch aur useful methods hote hain jo browser window ke saath interact karne ke kaam aate hain:

1. window.open()
   → Ek nayi browser window/tab open karta hai.
2. window.close()
   → Current window ko close karta hai.
   (Note: Sirf woh windows close hoti hain jo JavaScript ne khud open ki ho.)
3. window.moveTo()
   → Browser window ko screen par kisi specific position par move karta hai.
   (Security reasons ki wajah se modern browsers me limited support hoti hai.)
4. window.resizeTo()
   → Window ka size change karta hai.
   (Ye bhi modern browsers me mostly restricted hota hai.)

Window Screen

window.screen object browser ke device screen ke baare me information deta hai.
Isse aap window prefix ke bina bhi use kar sakte ho — bas screen likhna kaafi hai.

Important Properties:

1. screen.width
   → Screen ki total width (pixels me)
2. screen.height
   → Screen ki total height (pixels me)
3. screen.availWidth
   → Usable width (taskbar, dock wagaira ko exclude karke)
4. screen.availHeight
   → Usable height (taskbar, dock wagaira ko exclude karke)
5. screen.colorDepth
   → Screen kitne bits per pixel use karti hai (usually 24 or 32 bit)
6. screen.pixelDepth
   → Same as colorDepth (zyadatar browsers me same value hoti hai)

Window Screen Width

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"Screen width is " + screen.width;
</script>
</body>
</html>
```

Window Screen Height

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"Screen height is " + screen.height;
</script>
</body>
</html>
```

Window Screen Available Width

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"Available screen width is " + screen.availWidth;
</script>
</body>
</html>
```

Window Screen Available Height

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"Available screen height is " + screen.availHeight;
</script>
</body>
</html>
```

Window Screen Color Depth

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"Screen color depth is " + screen.colorDepth;
</script>
</body>
</html>
```

Window Screen Pixel Depth

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"Screen pixel depth is " + screen.pixelDepth;
</script>
</body>
</html>
```

JavaScript Window Location

window.location object browser me currently loaded webpage ki URL information deta hai. Aap isse window prefix ke bina bhi use kar sakte ho — sirf location likhna bhi kaafi hai.

Important Properties & Methods:

- location.href
  → Current page ka full URL return karta hai.
  Example: https://example.com/home/page.html
- location.hostname
  → Website ka domain name deta hai.
  Example: example.com
- location.pathname
  → Current page ka path + filename deta hai.
  Example: /home/page.html
- location.protocol
  → Web protocol batata hai:
  Example: https: or http:
- location.assign()
  → Kisi nayi URL par redirect karta hai (page load karta hai).
  Example: location.assign("https://google.com");

Window Location Href

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript</h2>
<h3>The window.location object</h3>
```

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"The full URL of this page is:<br>" + window.location.href;
</script>
</body>
</html>
```

Window Location Hostname

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript</h2>
<h3>The window.location object</h3>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"Page hostname is: " + window.location.hostname;
</script>
</body>
</html>
```

Window Location Pathname

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript</h2>
<h3>The window.location object</h3>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"Page path is: " + window.location.pathname;
</script>
</body>
</html>
```

Window Location Protocol

```
<!DOCTYPE html>
```

```
<html>
<body>
<h2>JavaScript</h2>
<h3>The window.location object</h3>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"The page protocol is: " + window.location.protocol;
</script>
</body>
</html>
```

Window Location Port

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript</h2>
<h3>The window.location object</h3>
<p id="demo"></p>
<p><b>Note: </b>If the port number is default (80 for http and 443 for https), most browsers
will display 0 or nothing.</p>
<script>
document.getElementById("demo").innerHTML =
"The URL port number of the current page is: " + window.location.port;
</script>
</body>
</html>
```

Window Location Assign

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript</h2>
<h3>The window.location object</h3>
<input type="button" value="Load new document" onclick="newDoc()">
<script>
function newDoc() {
  window.location.assign("https://www.w3schools.com")
}
```

```
</script>
</body>
</html>
```

JavaScript Window History

window.history object browser ke history (previous pages aur next pages) ko control karne ke kaam aata hai.
Aap isse window prefix ke bina bhi use kar sakte ho — sirf history likhna kaafi hai.
Lekin privacy reasons ki wajah se JavaScript puri history dekh ya read nahi kar sakti — sirf limited actions allowed hote hain.

Important Methods:

1. history.back()
   → Browser ke Back button jaisa kaam karta hai.
   → User ko pichle page par le jaata hai.
2. history.forward()
   → Browser ke Forward button jaisa kaam karta hai.
   → User ko aage wale page par le jaata hai (agar available ho).

Window History Back

```
<!DOCTYPE html>
<html>
<body>
<h2>Window History Back Example</h2>
<button onclick="goBack()">Go Back</button>
<script>
function goBack() {
  history.back();  // Goes to the previous page
}
</script>
</body>
</html>
```

Window History Forward

```
<!DOCTYPE html>
<html>
<body>
<h2>Window History Forward Example</h2>
```

```
<button onclick="goForward()">Go Forward</button>
<script>
function goForward() {
   history.forward();  // Goes to the next page in the browser history
}
</script>
</body>
</html>
```

JavaScript Window Navigator

navigator object browser ke baare me information provide karta hai — jaise browser ka naam, version, operating system, aur user-agent details.
Aap isse window.navigator ya sirf navigator likh kar access kar sakte ho.

Common Properties of Navigator:

1. navigator.appName → Browser ka naam
2. navigator.appVersion → Browser ka version
3. navigator.userAgent → Browser aur OS ka detail string
4. navigator.platform → Operating system ka platform
5. navigator.language → Browser ki default language

Browser Cookies

Browser Cookies chhoti text files hoti hain jo browser me store hoti hain aur website user ke baare me information ya preferences save karne ke liye use karti hai.
Cookies ka use mostly:

- Login sessions maintain karne ke liye
- User preferences ya settings save karne ke liye
- Tracking aur analytics ke liye

```
<!DOCTYPE html>
<html>
<body>
<h1>The Navigator Object</h1>
<h2>The cookieEnabled Property</h2>
<p>The cookieEnabled property returns true if cookies are enabled:</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"navigator.cookieEnabled is " + navigator.cookieEnabled;
</script>
</body>
```

</html>

The Browser Language

Browser Language ka matlab hai ki user ka browser ka default language setting kya hai. JavaScript me is information ko navigator.language property ke through access kiya ja sakta hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>The Navigator Object</h1>
<h2>The language Property</h2>
<p>The language property returns the browser's language:</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"navigator.language is " + navigator.language;
</script>
</body>
</html>
```

Is The Browser Online

JavaScript me aap check kar sakte ho ki browser currently online hai ya offline.
Iske liye navigator.onLine property use hoti hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>The Navigator Object</h1>
<h2>The online Property</h2>
<p>The onLine property returns true if the browser is online:</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"navigator.onLine is " + navigator.onLine;
</script>
</body>
</html>
```

Browser Application Name

Browser Application Name se matlab hai ki user ka browser ka name kya hai.
JavaScript me isse navigator.appName property ke through access kiya ja sakta hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>The Navigator Object</h1>
<h2>The appName Property</h2>
<p>The appName property is removed (deprecated) in the latest web standard.
Most browsers (Chrome, Edge, Firefox, Safari) returns Netscape as the appName:</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"navigator.appName is " + navigator.appName;
</script>
</body>
</html>
```

Browser Application Code Name

Browser Application Name se matlab hai ki user ka browser ka name kya hai.
JavaScript me isse navigator.appName property ke through access kiya ja sakta hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>The Navigator Object</h1>
<h2>The appCodeName Property</h2
<p>The appCodeName property returns the code name of the browser.</p>
<p>This property is removed (deprecated) in the latest web standard.
Most browsers (Chrome, Edge, Firefox, Safari) returns Mozilla as appCodeName.</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"navigator.appCodeName is " + navigator.appCodeName;
</script>
</body>
</html>
```

The Browser Engine

Browser Engine ek software component hai jo browser me web content render karne ke liye use hota hai.
Ye HTML, CSS, JavaScript aur images ko browser window me display karta hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>The Navigator Object</h1>
<h2>The product Property</h2>
<p>This property is removed (deprecated) in the latest web standard.
Most browsers returns <b>Gecko</b> as product name.</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"navigator.product is " + navigator.product;
</script>
</body>
</html>
```

The Browser Version

Browser Version se matlab hai ki user ka browser kaunsa version use kar raha hai.
JavaScript me iske liye navigator.appVersion property use hoti hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>The Navigator Object</h1>
<h2>The appVersion Property</h2>
<p>This property is removed (deprecated) in the latest web standard.
Do not rely on appVersion to return the correct browser version:</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Browser version is: " +
navigator.appVersion;
</script>
</body>
</html>
```

The Browser Agent

Browser Agent ka matlab hai User Agent, jo browser ka ek string hota hai jo server ko bataata hai ki client (user) ka browser aur operating system kya hai.
JavaScript me isse navigator.userAgent ke through access kiya ja sakta hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>The Navigator Object</h1>
<h2>The userAgent Property</h2>
<p>The userAgent property returns the user-agent header sent by the browser to the server:</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
navigator.userAgent;
</script>
</body>
</html>
```

The Browser Platform

Browser Platform se matlab hai ki browser kaunsa operating system ya hardware platform use kar raha hai.
JavaScript me iske liye navigator.platform property use hoti hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>The Navigator Object</h1>
<h2>The platform Property</h2>
<p>The platform property returns the browser platform (operating system):</p>
<p id="demo"></p>
<p>This property is removed (deprecated) in the latest web standard.
Do not rely on platform to return the correct browser platform in all browsers.</p>
<script>
document.getElementById("demo").innerHTML =
"navigator.platform is " + navigator.platform;
</script>
</body>
</html>
```

Is Java Enabled

JavaScript me aap check kar sakte ho ki browser me Java enabled hai ya nahi.
Iske liye navigator.javaEnabled() method use hota hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>The Navigator Object</h1>
<h2>The javaEnabled() Method</h2>
<p>This method is removed (deprecated) in the latest web standard.</p>
<p>The javaEnabled() method always returns false:</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"javaEnabled is " + navigator.javaEnabled();
</script>
</body>
</html>
```

JavaScript Timing Events

window object ke through aap JavaScript me code ko specific time ke baad ya repeatedly execute kar sakte ho.
Ye timing events kehlate hain.

Key Methods:

1. setTimeout(function, milliseconds)
   - Iska kaam hai ek function ko specified time ke baad execute karna.
   - Example: Agar aap 3 seconds ke baad koi message show karna chahte ho, to setTimeout use karoge.
2. setInterval(function, milliseconds)
   - Ye setTimeout jaisa hai, lekin repeat karta hai function ko continuously har specified interval me.
   - Example: Agar aap har 2 seconds me clock update karna chahte ho, to setInterval use karoge.

The setTimeout() Method

setTimeout() ek timing event method hai jo window object ka part hai.
Iska kaam hai: specified time ke baad ek function execute karna.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Timing</h2>
<p>Click "Try it". Wait 3 seconds, and the page will alert "Hello".</p>
<button onclick="setTimeout(myFunction, 3000);">Try it</button>
<script>
function myFunction() {
  alert('Hello');
}
</script>
</body>
</html>
```

How to Stop the Execution?

Stop setTimeout()

- setTimeout() se scheduled function ko stop karne ke liye ID store karo.
- Use clearTimeout(timeoutID) to cancel execution.
- Example:
- let timeoutID = setTimeout(myFunction, 5000);
- clearTimeout(timeoutID); // Cancels the function

Stop setInterval()

- setInterval() se repeated execution ko stop karne ke liye ID store karo.
- Use clearInterval(intervalID) to stop execution.
- Example:
- let intervalID = setInterval(myFunction, 2000);
- clearInterval(intervalID); // Stops the repeated execution

Key Points

- Both methods return an ID which is used to cancel execution.
- Useful when you want to stop delayed or repeated tasks based on user action or conditions.
- Always store the ID if you think you may need to cancel the function later.

The clearTimeout() method

```
<!DOCTYPE html>
<html>
<body>
```

```
<h2>JavaScript Timing</h2>

<p>Click "Try it". Wait 3 seconds. The page will alert "Hello".</p>
<p>Click "Stop" to prevent the first function to execute.</p>
<p>(You must click "Stop" before the 3 seconds are up.)</p>
<button onclick="myVar = setTimeout(myFunction, 3000)">Try it</button>
<button onclick="clearTimeout(myVar)">Stop it</button>
<script>
function myFunction() {
  alert("Hello");
}
</script>
</body>
</html>
```

The setInterval() Method

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Timing</h2>
<p>A script on this page starts this clock:</p
<p id="demo"></p>
<script>
setInterval(myTimer, 1000)
function myTimer() {
  const d = new Date();
  document.getElementById("demo").innerHTML = d.toLocaleTimeString();
}
</script>
</body>
</html>
```

JavaScript Cookies

Cookies chhoti text files hoti hain jo user ke computer/browser me store hoti hain.
Ye websites ko user-specific information ya preferences yaad rakhne me help karti hain.

Why Cookies Are Needed:

- Jab browser server se web page request karta hai, connection band ho jaata hai.
- Server user ke baare me kuch yaad nahi rakhta by default.

- Cookies solve karte hain ye problem: “Kaise user ki information yaad rakhe?”

Example:

1. User pehli baar website visit karta hai → Browser me user ka naam store ho jaata hai.
2. Agli baar user website visit kare → Cookie uska naam yaad rakh leti hai.

Create a Cookie with JavaScript

JavaScript me cookie create karne ke liye document.cookie property use hoti hai.
Ek cookie me aap name, value, expiry date, aur path define kar sakte ho.

```
document.cookie = "name=value; expires=Date; path=/";
```

```
<!DOCTYPE html>
<html>
<body>
<h2>Create a Cookie Example</h2>
<button onclick="createCookie()">Set Cookie</button>
<script>
function createCookie() {
   // Create a cookie named "username" with value "John", expires after 7 days
   let d = new Date();
   d.setTime(d.getTime() + (7*24*60*60*1000)); // 7 days in milliseconds
   let expires = "expires=" + d.toUTCString();
   document.cookie = "username=John; " + expires + "; path=/";
   alert("Cookie has been created!");
}
</script>
</body>
</html>
```

Read a Cookie with JavaScript

JavaScript me cookie read karne ke liye document.cookie property use hoti hai.
Ye current page ki saari cookies ek single string ke form me return karti hai.

```
<!DOCTYPE html>
<html>
<body>
<h2>Read Cookie Example</h2>

<button onclick="readCookie()">Show Cookies</button
<p id="cookieDisplay"></p>
```

```
<script>
function readCookie() {
   let allCookies = document.cookie; // Get all cookies as a string
   if(allCookies) {
      document.getElementById("cookieDisplay").innerHTML = "Cookies: " + allCookies;
   } else {
      document.getElementById("cookieDisplay").innerHTML = "No cookies found!";
   }
}
</script>
</body>
</html>
```

Change a Cookie with JavaScript

JavaScript me cookie ko change/modify karne ke liye document.cookie use karke same cookie name ke saath new value set kar dete hain.
Browser automatically purani cookie ko replace kar deta hai.

```
<!DOCTYPE html>
<html>
<body>
<h2>Change Cookie Example</h2>
<button onclick="changeCookie()">Change Cookie</button>
<p id="cookieDisplay"></p>
<script>
// Initially create a cookie
document.cookie = "username=John; path=/";
// Function to change cookie
function changeCookie() {
   // Change the value of "username" cookie
   document.cookie = "username=Alice; path=/";
   document.getElementById("cookieDisplay").innerHTML = "Cookie changed: " +
document.cookie;
}
</script>
</body>
</html>
```

Delete a Cookie with JavaScript

JavaScript me cookie delete karne ke liye uska expiry date past me set kar diya jaata hai.
Browser dekh kar cookie ko remove kar deta hai.

```
<!DOCTYPE html>
<html>
<body
<h2>Delete Cookie Example</h2>
<button onclick="deleteCookie()">Delete Cookie</button>
<p id="cookieDisplay"></p>
<script>
// Initially create a cookie
document.cookie = "username=John; path=/";
// Function to delete cookie
function deleteCookie() {
   document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/";
   document.getElementById("cookieDisplay").innerHTML = "Cookie deleted! Current
cookies: " + document.cookie;
}
</script>
</body>
</html>
```

The Cookie String

document.cookie ek property hai jo browser ke cookies ko handle karti hai. Lekin ye normal string ki tarah behave nahi karti.

Read karne par:

Agar aap document.cookie se cookie read karte ho, to aapko sirf name=value pairs hi dikhte hain.

Example:

- document.cookie = "username=John";
- console.log(document.cookie); // Output: "username=John"

Write karne par:

Agar aap ek nayi cookie set karte ho, purani cookies overwrite nahi hoti. Nayi cookie simply add ho jati hai.

Example:

- document.cookie = "age=25";
- console.log(document.cookie); // Output: "username=John; age=25"

Key point:

- document.cookie ek concatenated string of all name=value pairs return karta hai.
- Har nayi cookie add hoti hai, purani delete nahi hoti (jab tak explicitly expire na kar do).

A Function to Set a Cookie

```
function setCookie(cname, cvalue, exdays) {
  const d = new Date();
  d.setTime(d.getTime() + (exdays*24*60*60*1000));
  let expires = "expires="+ d.toUTCString();
  document.cookie = cname + "=" + cvalue + ";" + expires +
";path=/";
}
```

A Function to Get a Cookie

```
function getCookie(cname) {
  let name = cname + "=";
  let decodedCookie = decodeURIComponent(document.cookie);
  let ca = decodedCookie.split(';');
  for(let i = 0; i <ca.length; i++) {
    let c = ca[i];
    while (c.charAt(0) == ' ') {
      c = c.substring(1);
    }
    if (c.indexOf(name) == 0) {
      return c.substring(name.length, c.length);
    }
  }
  return "";
}
```

A Function to Check a Cookie

```
function checkCookie() {
  let username = getCookie("username");
  if (username != "") {
   alert("Welcome again " + username);
  } else {
    username = prompt("Please enter your name:", "");
    if (username != "" && username != null) {
      setCookie("username", username, 365);
    }
  }
}
```

```
<!DOCTYPE html>
<html>
<head>
<script>
function setCookie(cname,cvalue,exdays) {
  const d = new Date();
  d.setTime(d.getTime() + (exdays*24*60*60*1000));
  let expires = "expires=" + d.toUTCString();
  document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";
}
function getCookie(cname) {
  let name = cname + "=";
  let decodedCookie = decodeURIComponent(document.cookie);
  let ca = decodedCookie.split(';');
  for(let i = 0; i < ca.length; i++) {
    let c = ca[i];
    while (c.charAt(0) == ' ') {
      c = c.substring(1);
    }
    if (c.indexOf(name) == 0) {
      return c.substring(name.length, c.length);
    }
  }
  return "";
}
function checkCookie() {
```

```
 let user = getCookie("username");
 if (user != "") {
  alert("Welcome again " + user);
 } else {
   user = prompt("Please enter your name:","");
   if (user != "" && user != null) {
    setCookie("username", user, 30);
   }
 }
}
</script>
</head>
<body onload="checkCookie()">
</body>
</html>
```

Web APIs

1. **Browser ki functionality extend karna** – Jaise geolocation, audio, ya graphics add karna.
2. **Complex tasks ko simplify karna** – Mushkil operations ko simple function calls se easy banana.
3. **Easy aur clean syntax provide karna** – Pehle jo complex tha, ab readable aur simple code me possible.
4. **Hardware ke saath interact karna** – Camera, microphone, sensors access karna.
5. **Modern web apps ko support karna** – Dynamic aur interactive websites banane ke liye essential.

API ka matlab hai Application Programming Interface – basically, yeh ek set of rules or tools hai jo allow karta hai do applications ko ek dusre se baat karne ke liye.

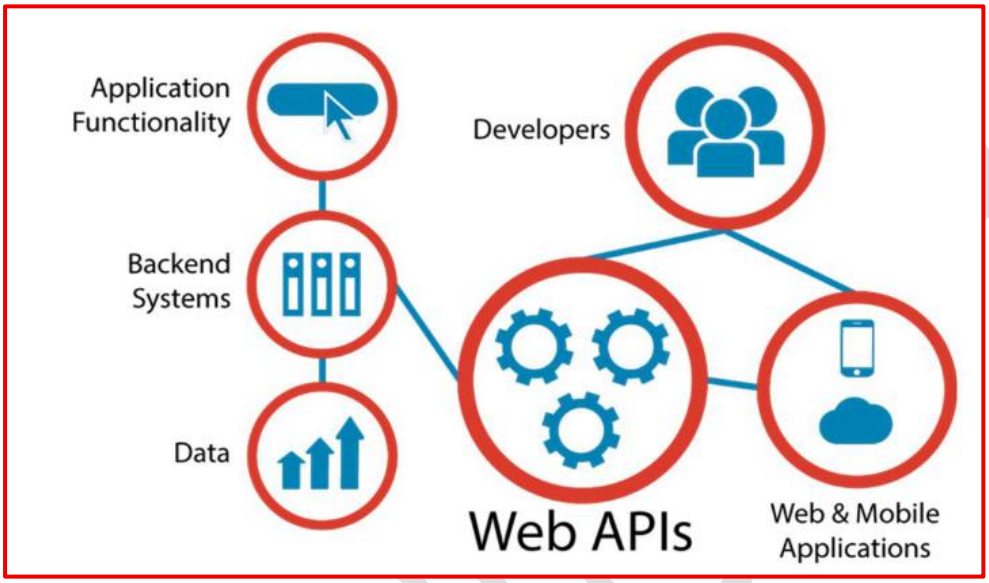
Web API:

- Jab API Web ke liye hota hai, matlab yeh internet ke through applications ke beech communication allow karta hai.
- Example: Agar aapka app weather information chahiye, toh aap ek Web API ke through Weather service se data le sakte ho.

Types of Web API:

1. **Browser API**
   - Yeh APIs web browser ke features ko extend karte hain.
   - Matlab browser ke andar extra functionalities add karna.
   - Example: Geolocation API jo browser ko location detect karne deta hai, ya LocalStorage API jo browser me data save karne deta hai.

2. **Server API**
   - Yeh APIs web server ke functionalities ko extend karte hain.
   - Matlab server se data ya services ko access karna aur use control karna.
   - Example: REST API ya GraphQL API jo server se data fetch karte hain aur app me dikhate hain.



Third Party APIs

Ye wo APIs hain jo aapke browser ya server me by default nahi hote.

- Matlab agar aapko ye use karna hai, toh aapko web se download karna padega ya access ke liye register karna padega.

Key Points:

1. **Built-in nahi hote** – Browser ya server ke default tools me ye nahi aate.
2. **Web se leke use karna padta hai** – Documentation follow karke aap inhe apne project me integrate kar sakte ho.
3. **Examples:**
   - **YouTube API** → Website me videos embed karne ke liye.
   - **Twitter API** → Tweets display karne ke liye.
   - **Facebook API** → Facebook ka info show karne ke liye.

JavaScript Validation API

- Ye web forms me data validate karne ke liye JavaScript ka built-in API hai.
- Matlab user ne form me jo input diya hai, wo sahi format me hai ya nahi, ye check karne ke liye use hota hai.
- Validation ka matlab: "Data ko check karna before server ko send karna".

Constraint Validation DOM Methods

Property / Method	Description (Hinglish)	Example / Use
checkValidity()	Input field valid hai ya nahi check karta hai. True return karta hai agar valid, false agar invalid.	if(!email.checkValidity()){ alert("Invalid Email!"); }
setCustomValidity()	Input field ke liye custom error message set karta hai. Default browser message replace hota hai.	password.setCustomValidity("Password must be at least 6 characters!");

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Validation</h2>
<p>Enter a number and click OK:</p>
<input id="id1" type="number" min="100" max="300" required>
<button onclick="myFunction()">OK</button>
<p>If the number is less than 100 or greater than 300, an error message will be
displayed.</p>
<p id="demo"></p>
<script>
function myFunction() {
  const inpObj = document.getElementById("id1");
  if (!inpObj.checkValidity()) {
    document.getElementById("demo").innerHTML = inpObj.validationMessage;
  } else {
    document.getElementById("demo").innerHTML = "Input OK";
  }
}
</script>
</body>
</html>
```

Constraint Validation DOM Properties

Property	Description (Hinglish)	Example / Use
validity	Input element ke validity ke boolean properties contain karta hai (jaise valid, valueMissing, typeMismatch etc.).	if(email.validity.typeMismatch){ alert("Invalid email format!"); }
validationMessage	Wo message jo browser invalid input par show karega.	alert(email.validationMessage);
willValidate	Check karta hai ki ye input element validate hoga ya nahi.	if(password.willValidate){ console.log("Password will be validated."); }

Validity Properties

- validity object ke andar ye boolean properties hoti hain jo input ke valid/invalid status ko check karte hain.
- Har property true/false return karti hai.

Property	Description (Hinglish)
customError	True hoga agar input me custom validity message set kiya gaya ho (setCustomValidity()).
patternMismatch	True hoga agar input ka value pattern attribute se match nahi karta.
rangeOverflow	True hoga agar input ka value max attribute se bada hai.
rangeUnderflow	True hoga agar input ka value min attribute se chhota hai.
stepMismatch	True hoga agar input ka value step attribute ke rule ko follow nahi karta.
tooLong	True hoga agar input ka value maxlength se zyada lamba hai.
tooShort	True hoga agar input ka value minlength se chhota hai.
typeMismatch	True hoga agar input ka value type (email, URL, etc.) ke format me invalid hai.
valueMissing	True hoga agar required input field empty hai.
valid	True hoga agar input ka value completely valid hai.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Validation</h2>
<p>Enter a number and click OK:</p
```

```
<input id="id1" type="number" max="100">
<button onclick="myFunction()">OK</button>
<p>If the number is greater than 100 (the input's max attribute), an error message will be displayed.</p>
<p id="demo"></p>
<script>
function myFunction() {
  let text;
  if (document.getElementById("id1").validity.rangeOverflow) {
    text = "Value too large";
  } else {
    text = "Input OK";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>
</body>
</html>
```

Web History API kya hai

- Ye browser ka API hai jo web page ke history ko manipulate karne aur access karne ke liye use hota hai.
- Isse aap current page ka URL change kar sakte ho, without reloading the page.
- Mainly Single Page Applications (SPA) me use hota hai.

Yes	Yes	Yes	Yes	Yes

The History back() Method

```
<!DOCTYPE html>
<html>
<head>
  <title>History Back Example</title>
</head>
```

```
<body>
  <h1>History Back() Method Example</h1>
  <p>Click the button to go back to the previous page in your browser history.</p>
  <button id="backBtn">Go Back</button>
  <script>
    // Select the button
    const backButton = document.getElementById("backBtn");
    // Add click event to go back one step in history
    backButton.addEventListener("click", function() {
      history.back(); // Goes back one page in the browser history
    });
  </script>
</body>
</html>
```

The History go() Method

- Ye method browser history me multiple steps move karne ke liye use hota hai.
- Positive number → forward in history
- Negative number → backward in history
- Zero (0) → Current page reload karta hai

```
<!DOCTYPE html>
<html>
<head>
  <title>History go() Example</title>
</head>
<body>
  <h1>History go() Method Example</h1>
  <button id="backBtn">Go Back 1 Page</button>
  <button id="forwardBtn">Go Forward 1 Page</button>
  <button id="reloadBtn">Reload Page</button>
  <script>
    document.getElementById("backBtn").addEventListener("click", function() {
      history.go(-1); // Ek page back
    });
    document.getElementById("forwardBtn").addEventListener("click", function() {
      history.go(1); // Ek page forward
    });
    document.getElementById("reloadBtn").addEventListener("click", function() {
      history.go(0); // Current page reload
    });
```

```
  </script>
</body>
</html>
```

History Object Methods

Method	Description (Hinglish)
back()	Browser ko previous URL pe le jaata hai (jaise back button click).
forward()	Browser ko next URL pe le jaata hai (jaise forward button click).
go()	Browser history me specific number of steps move karta hai. (go(-1) = back, go(1) = forward, go(0) = reload current page)

Web Storage API

- Ye ek browser-based storage system hai jo web applications ko data client-side (user ke browser me) store karne allow karta hai.
- Iska data browser me save hota hai, aur server pe send nahi hota (jab tak explicitly na bheja jaye).
- Web Storage cookies ke alternative ke roop me use hota hai, lekin zyada space aur easy API provide karta hai.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
localStorage.setItem("name","John Doe");
document.getElementById("demo").innerHTML = localStorage.getItem("name");
</script>
</body>
</html>
```

localStorage Object

- localStorage ek Web Storage API ka object hai jo browser me data permanently store karta hai.
- Data specific website/domain ke liye store hota hai.
- Data ka koi expiration date nahi hota → browser close hone ke baad bhi data safe rehta hai.
- Ye data days, weeks, ya even years tak available rahega, jab tak manually delete na ho.

setItem() Method

- Ye method localStorage me data store karne ke liye use hota hai.
- Data key-value pair ke form me store hota hai.
- Agar same key pe pehle se data hai, to old value replace ho jaati hai.

The getItem() Method

- Ye method localStorage me stored data ko read (retrieve) karne ke liye use hota hai.
- Data key-value pair ke form me store hota hai, aur getItem() se aap value ko key ke through access karte ho.
- Agar key exist nahi karti, to method null return karta hai.

Storage Object Properties and Methods

Property / Method	Description
key(n)	Storage me nth key ka name return karta hai.
length	Storage me total stored items ki sankhya return karta hai.
getItem(keyname)	Specified key ka value return karta hai.
setItem(keyname, value)	Ek key-value pair add karta hai, ya agar key already exist karti hai to value update karta hai.
removeItem(keyname)	Specified key ko storage se delete karta hai.
clear()	Saare keys ko storage se remove kar deta hai.

Related Pages for Web Storage API

Property / Object	Description
window.localStorage	Browser me key/value pairs permanently store karne ke liye. Data ka koi expiration date nahi hota.
window.sessionStorage	Browser me key/value pairs temporary store karne ke liye. Data tab tak rahega jab tak session (tab) open hai.

What is a Web Worker

- Jab aap JavaScript scripts HTML page me run karte ho, page tab tak unresponsive ho sakta hai jab tak script finish na ho jaye.
- Web Worker ek JavaScript script hai jo background me run hoti hai, main page ke scripts se independent.

- Matlab: Page freeze nahi hota, aap click, select, scroll, ya interact kar sakte ho, while worker kaam kar raha hai.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Web Workers API</h2>
<p>Count numbers: <output id="result"></output></p>
<button onclick="startWorker()">Start Worker</button>
<button onclick="stopWorker()">Stop Worker</button>
<script>
let w;
function startWorker() {
  if(typeof(w) == "undefined") {
    w = new Worker("demo_workers.js");
  }
  w.onmessage = function(event) {
    document.getElementById("result").innerHTML = event.data;
  };
}
function stopWorker() {
  w.terminate();
  w = undefined;
}
</script>
</body>
</html>
```

What is a Worker File?

- Worker file ek separate JavaScript file hoti hai jo Web Worker ke liye banayi jaati hai.
- Ye file background me independently run hoti hai, main HTML page ke scripts se alag thread me.
- Worker file me aap time-consuming tasks ya continuous operations perform kar sakte ho without freezing the page.
- Ye directly DOM access nahi kar sakti, data communicate karne ke liye postMessage() aur onmessage use hota hai.

What is a Web Worker Object?

- Web Worker object main thread se communicate karne ke liye use hota hai.
- Isse aap background thread (worker) create kar sakte ho aur main page freeze hone se bachate ho.

Why Terminate a Web Worker?

- Jab aapko background task complete ho jaye ya worker ki zarurat na ho, tab aap worker ko terminate kar sakte ho.
- Terminating a worker frees up memory and CPU resources.

What is Fetch API?

- Fetch API ek modern JavaScript method hai jo HTTP requests (GET, POST, PUT, DELETE, etc.) send karne ke liye use hoti hai.
- Ye Promise-based hai → asynchronous requests ke liye easy aur readable syntax provide karta hai.
- Previously hum XMLHttpRequest (XHR) use karte the, lekin fetch API simpler aur cleaner hai.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Fetch a file to change this text.</p>
<script>
getText("fetch_info.txt");
async function getText(file) {
  let myObject = await fetch(file);
  let myText = await myObject.text();
  document.getElementById("demo").innerHTML = myText;
}
</script>
</body>
</html>
```

What is Web Geolocation API?

- Geolocation API ek JavaScript API hai jo user ke device ki location (latitude & longitude) access karne ke liye use hoti hai.
- Location GPS, Wi-Fi, or IP address ke through detect hoti hai.
- User ki permission chahiye hoti hai location access karne ke liye.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Geolocation API</h2>
<p>Click the button to get your coordinates.</p>
<button onclick="getLocation()">Try It</button>
<p id="demo"></p>
```

```
<script>
const x = document.getElementById("demo");
function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(showPosition);
  } else {
    x.innerHTML = "Geolocation is not supported by this browser.";
  }
}
function showPosition(position) {
  x.innerHTML = "Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
}
</script>
</body>
</html>
```

Handling Errors in Fetch API

- Fetch API is Promise-based, so errors can be handled using .catch() or try…catch with async/await.
- Common errors: network failure, invalid URL, server error, JSON parsing errors.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Geolocation API</h2>
<p>Click the button to get your coordinates.</p>
<button onclick="getLocation()">Try It</button>
<p id="demo"></p>
<script>
const x = document.getElementById("demo");
function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(showPosition, showError);
  } else {
    x.innerHTML = "Geolocation is not supported by this browser.";
  }
}
function showPosition(position) {
  x.innerHTML = "Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
```

```
}
function showError(error) {
  switch(error.code) {
    case error.PERMISSION_DENIED:
      x.innerHTML = "User denied the request for Geolocation."
      break;
    case error.POSITION_UNAVAILABLE:
      x.innerHTML = "Location information is unavailable."
      break;
    case error.TIMEOUT:
      x.innerHTML = "The request to get user location timed out."
      break;
    case error.UNKNOWN_ERROR:
      x.innerHTML = "An unknown error occurred."
      break;
  }
}
</script>
</body>
</html>
```

Displaying the Result in a Map

```
<!DOCTYPE html>
<html>
<head>
  <title>Display Location on Google Map</title>
  <style>
    #map {
      height: 400px;
      width: 100%;
    }
  </style>
</head>
<body>
  <h1>My Current Location on Map</h1>
  <div id="map"></div>
  <!-- Google Maps API Script -->
  <script src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY"></script>
  <script>
    if (navigator.geolocation) {
```

```
    navigator.geolocation.getCurrentPosition(
      function(position) {
        const lat = position.coords.latitude;
        const lng = position.coords.longitude;

        // Initialize map
        const map = new google.maps.Map(document.getElementById('map'), {
          center: { lat: lat, lng: lng },
          zoom: 15
        });

        // Add marker
        const marker = new google.maps.Marker({
          position: { lat: lat, lng: lng },
          map: map,
          title: "You are here!"
        });
      },
      function(error) {
        console.error("Error getting location:", error.message);
      }
    );
  } else {
    alert("Geolocation is not supported by your browser.");
  }
 </script>
</body>
</html>
```

Location-specific Information

- Jab aap user ki location access kar lete ho using Geolocation API, aap us location ke basis par custom information show kar sakte ho.
- Ye feature maps, apps, and websites me bahut useful hai.

The getCurrentPosition() Method - Return Data

Property	Returns
coords.latitude	The latitude as a decimal number (always returned)

Property	Returns
coords.longitude	The longitude as a decimal number (always returned)
coords.accuracy	The accuracy of the position in meters (always returned)
coords.altitude	The altitude in meters above mean sea level (returned if available)
coords.altitudeAccuracy	The altitude accuracy of the position in meters (returned if available)
coords.heading	The heading in degrees clockwise from North (returned if available)
coords.speed	The speed in meters per second (returned if available)
timestamp	The date/time of the response (returned if available)

Geolocation Object - Other interesting Methods

- watchPosition()
  - Continuously tracks user's location as they move
  - Returns a watchId to manage tracking
  - Useful for GPS, navigation, or real-time location updates
- clearWatch()
  - Stops tracking started by watchPosition()
  - Takes the watchId as parameter
  - Helps save battery and resources when tracking is no longer needed

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Geolocation API</h2>
<p>Click the button to get your coordinates.</p>
<button onclick="getLocation()">Try It</button>
<p id="demo"></p>
<script>
const x = document.getElementById("demo");
function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.watchPosition(showPosition);
  } else {
    x.innerHTML = "Geolocation is not supported by this browser.";
  }
}
```

```
function showPosition(position) {
   x.innerHTML="Latitude: " + position.coords.latitude +
   "<br>Longitude: " + position.coords.longitude;
}
</script>
</body>
</html>
```

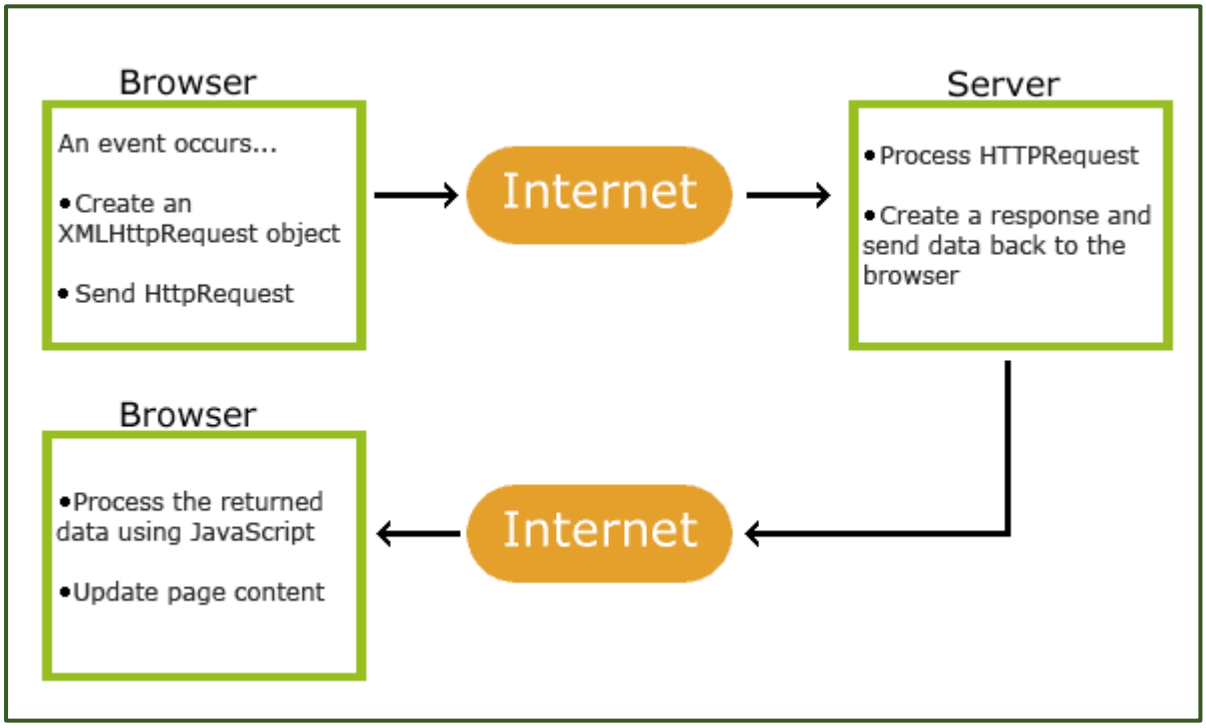
AJAX Introduction

- Read data from a web server – Page load hone ke baad bhi server se data fetch kar sakte ho
- Update a web page without reloading – User experience smooth ho jata hai, page refresh ki zarurat nahi
- Send data to a web server in the background – Form submit ya user actions ke liye background me data bhej sakte ho

```
<!DOCTYPE html>
<html>
<body>
<div id="demo">
<h2>The XMLHttpRequest Object</h2>
<button type="button" onclick="loadDoc()">Change Content</button>
</div>
<script>
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    document.getElementById("demo").innerHTML =
    this.responseText;
  }
  xhttp.open("GET", "ajax_info.txt");
  xhttp.send();
}
</script>
</body>
</html>
```

- AJAX = Asynchronous JavaScript And XML
- Not a programming language – Ye sirf techniques ka combination hai
- Uses:
- XMLHttpRequest object → Browser se server se data fetch karne ke liye

- JavaScript & HTML DOM → Fetched data ko page me display ya use karne ke liye



Fetch API

- Fetch API modern replacement hai AJAX (XMLHttpRequest) ka
- Built-in hai sab modern browsers me → Chrome, Firefox, Edge, Safari
- Promise-based hai → Asynchronous code easy aur readable hota hai
- Supports GET, POST, PUT, DELETE requests
- Work karta hai JSON, text, blobs, FormData, etc. ke saath

What is XMLHttpRequest (XHR)?

- XMLHttpRequest ek JavaScript object hai jo AJAX requests bhejne ke liye use hota hai
- Server se data fetch karne aur page update karne me help karta hai without reloading the page
- Data format: XML, JSON, HTML, text

Create an XMLHttpRequest Object

- XMLHttpRequest ek built-in JavaScript object hai

- Ye object AJAX requests bhejne aur server se response lene ke liye use hota hai
- xhr naam ka variable ab request aur response handle kar sakta hai

Define a Callback Function

What is a Callback Function

- Callback function ek function hai jo dusre function ko argument ke roop me pass kiya jata hai
- Ye tab call hota hai jab dusra function complete ho jaye
- Mostly asynchronous operations (like AJAX, timers, events) me use hota hai

```
<!DOCTYPE html>
<html>
<body>
<h2>The XMLHttpRequest Object</h2>
<div id="demo">
<p>Let AJAX change this text.</p>
<button type="button" onclick="loadDoc()">Change Content</button>
</div>
<script>
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    document.getElementById("demo").innerHTML = this.responseText;
  }
  xhttp.open("GET", "ajax_info.txt");
  xhttp.send();
}
</script>
</body>
</html>
```

What is Access Across Domains

- Access Across Domains ka matlab hai: ek web page ek different domain (origin) se data fetch karna.
- Same-Origin Policy (SOP) ke wajah se browser normally restrict karta hai cross-domain requests for security.
- Agar A.com page se B.com ka data access karna ho, toh special techniques use karni padti hain.

XMLHttpRequest Object Methods

Method	Description
new XMLHttpRequest()	Ek naya XMLHttpRequest object create karta hai
abort()	Current request ko cancel karta hai
getAllResponseHeaders()	Server se aayi saari header information return karta hai
getResponseHeader()	Server se aayi specific header information return karta hai
open(method, url, async, user, psw)	Request ko specify karta hai • method: GET ya POST • url: file location • async: true/false • user: optional username • psw: optional password
send()	Request ko server ko bhejta hai (GET requests ke liye)
send(string)	Request ko server ko bhejta hai aur data POST kar sakte ho (POST requests ke liye)
setRequestHeader()	Header me label/value pair add karta hai jo server ko bheja jayega

XMLHttpRequest Object Properties

Property	Description
onload	Jab request complete ho jaye aur response aa jaye, tab jo function chalana hai wo yahan define karte hain.
onreadystatechange	Jab bhi readyState change hota hai, yeh function call hota hai. Matlab request ki progress track karne ke kaam aata hai.
readyState	Request ki current state batata hai: 0: Request start bhi nahi hui 1: Server se connection ban gaya 2: Request server ko mil gayi 3: Server request process kar raha hai 4: Response ready hai
responseText	Server ka response string form me deta hai (mostly JSON ya text).
responseXML	Server ka response XML format me deta hai.
status	HTTP status code batata hai: 200: OK (sab set) 403: Forbidden 404: Not Found
statusText	Status code ka text version, jaise "OK", "Not Found".

The onload Property

- onload tab trigger hota hai jab readyState = 4 ho aur status 200 (OK) ho.
- Iske andar aap response ko process kar sakte ho.

```
<!DOCTYPE html>
<html>
<body>
<div id="demo">
<h2>The XMLHttpRequest Object</h2>
<button type="button" onclick="loadDoc()">Change Content</button>
</div>
<script>
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    document.getElementById("demo").innerHTML =
    this.responseText;
  }
  xhttp.open("GET", "ajax_info.txt");
  xhttp.send();
}
</script>
</body>
</html>
```

Multiple Callback Functions

- Ek common function banao jo XMLHttpRequest banaye aur execute kare.
- Har AJAX task ke liye alag callback function banao jo response ko process kare.

The onreadystatechange Property

- onreadystatechange XMLHttpRequest (XHR) object ka ek property hai.
- Yeh ek function assign karta hai jo XHR ke state change hone par call hota hai.
- XHR ke readyState ka status change hone par yeh function execute hota hai.

Property	Description	Notes / Values
onreadystatechange	Ek function define karta hai jo readyState change hone par call hota hai	Isme hum response handle karte hain
readyState	XMLHttpRequest ka current status batata hai	0: request not initialized1: server connection established2: request

Property	Description	Notes / Values
		received3: processing request4: request finished and response ready
status	HTTP response ka status code	200: OK403: Forbidden404: Page not found…complete list.
statusText	HTTP response ka status message	Example: "OK", "Not Found"

```
<!DOCTYPE html>
<html>
<body>
<div id="demo">
<h2>The XMLHttpRequest Object</h2>
<button type="button" onclick="loadDoc()">Change Content</button>
</div>
<script>
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt");
  xhttp.send();
}
</script>
</body>
</html>
```

AJAX - XMLHttpRequest

AJAX ek technique hai jo web page ko asynchronously server se data fetch karne ya send karne me help karti hai, bina page reload kiye.

Key Points:

- Page reload nahi hota → fast user experience
- Server se data fetch ya send kar sakte ho
- Response handle karne ke liye JavaScript callbacks use hote hain

Send a Request To a Server

Method	Description	Notes
open(method, url, async)	Request ka type specify karta hai	- method: "GET" ya "POST"- url: server ya file ka location- async: true (asynchronous) ya false (synchronous)
send()	Request server ko bhejta hai (GET request ke liye)	GET request me parameters URL me hote hain
send(string)	Request server ko bhejta hai (POST request ke liye)	POST request me parameters body me bheje jaate hain, usually string format ya JSON

The url - A File On a Server

```
xhttp.open("GET", "ajax_test.asp", true);
```

Asynchronous - True or False?

```
xhttp.open("GET", "ajax_test.asp", true);
```

GET or POST

Definition:

- GET request server se data fetch karne ke liye use hoti hai.
- URL me parameters attach hote hain: example.com?name=Rahul&age=25

Advantages:

- Simple aur fast
- Browser easily cache kar sakta hai
- Bookmarkable URL

Limitations:

- Data size limited hota hai (URL length limit)
- Sensitive data ke liye safe nahi (URL me visible)

- Server side file update ya database update ke liye suitable nahi

Use Case:

- Search queries
- Data fetching
- Public APIs

POST Request

Definition:

- POST request server ko data send karne ke liye use hoti hai.
- Parameters request body me hote hain, URL me nahi

Advantages:

- Large data bheja ja sakta hai
- User input (text, JSON) secure aur robust send kar sakte ho
- File update, database update ke liye safe

Limitations:

- Slightly slower than GET
- URL me data visible nahi → bookmarkable nahi

Use Case:

- Form submit
- Database update
- File upload
- Sensitive data sen

Get request

```
<!DOCTYPE html>
<html>
<body>
<h2>The XMLHttpRequest Object</h2>
<button type="button" onclick="loadDoc()">Request data</button>

<p id="demo"></p>
<script>
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    document.getElementById("demo").innerHTML = this.responseText;
  }
  xhttp.open("GET", "demo_get2.asp?fname=Henry&lname=Ford");
  xhttp.send();
}
</script>
</body>
```

```
</html>
```

POST Request

```
<!DOCTYPE html>
<html>
<body>
<h2>The XMLHttpRequest Object</h2>
<button type="button" onclick="loadDoc()">Request data</button>
<p id="demo"></p>
<script>
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    document.getElementById("demo").innerHTML = this.responseText;
  }
  xhttp.open("POST", "demo_post.asp");
  xhttp.send();
}
</script>
</body>
</html>
```

Method	Description	Parameters / Notes
setRequestHeader(header, value)	Adds HTTP headers to the request	header: header ka namevalue: header ka value

Synchronous Request

- Synchronous request me JavaScript wait karta hai server response ke liye before executing the next line of code.
- Page block ho jata hai jab tak server response nahi aata.
- Old-school approach, rarely use hoti hai, kyunki user experience slow ho jata hai.

```
<!DOCTYPE html>
<html>
<body>
<h2>The XMLHttpRequest Object</h2>
<p id="demo">Let AJAX change this text.</p>
<button type="button" onclick="loadDoc()">Change Content</button>
<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
```

```
  xhttp.open("GET", "ajax_info.txt", false);
  xhttp.send();
  document.getElementById("demo").innerHTML = xhttp.responseText;
}
</script>
</body>
</html>
```

AJAX - Server Response

Property	Description
responseText	Server se aaya hua response string format me return karta hai
responseXML	Server se aaya hua response XML format me return karta hai

The responseText Property

```
<!DOCTYPE html>
<html>
<body>
<div id="demo">
<h2>The XMLHttpRequest Object</h2>
<button type="button" onclick="loadDoc()">Change Content</button>
</div>
<script>
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    document.getElementById("demo").innerHTML =
    this.responseText;
  }
  xhttp.open("GET", "ajax_info.txt");
  xhttp.send();
}
</script>
</body>
</html>
```

The responseXML Property

- responseXML XMLHttpRequest ka property hai jo server se aaya hua response XML format me return karta hai.
- Agar server valid XML return kare → ye ek XML Document object banata hai, jisse DOM methods ke through access kiya ja sakta hai.

```
<!DOCTYPE html>
<html>
<body>
<h2>The XMLHttpRequest Object</h2>
<p id="demo"></p>
<script>
const xhttp = new XMLHttpRequest();
xhttp.onload = function() {
  const xmlDoc = this.responseXML;
  const x = xmlDoc.getElementsByTagName("ARTIST");
  let txt = "";
  for (let i = 0; i < x.length; i++) {
    txt = txt + x[i].childNodes[0].nodeValue + "<br>";
  }
  document.getElementById("demo").innerHTML = txt;
}
xhttp.open("GET", "cd_catalog.xml");
xhttp.send();
</script>
</body>
</html>
```

Server Response Methods

Method	Description	Usage
getResponseHeader(headerName)	Server ke response me se specific header ka value return karta hai	xhr.getResponseHeader("Content-Type")
getAllResponseHeaders()	Server ke response ke saare headers ek string me return karta hai	console.log(xhr.getAllResponseHeaders())

The getAllResponseHeaders() Method

```
<!DOCTYPE html>
<html>
<body>
<h2>The XMLHttpRequest Object</h2>
<p>The getAllResponseHeaders() function returns all the header information of a resource, like length, server-type, content-type, last-modified, etc:</p>
<pre id="demo"></pre>
<script>
const xhttp = new XMLHttpRequest();
xhttp.onload = function() {
  document.getElementById("demo").innerHTML =
  this.getAllResponseHeaders();
}
xhttp.open("GET", "ajax_info.txt");
xhttp.send();
</script>
</body>
</html>
```

The getResponseHeader() Method

```
<!DOCTYPE html>
<html>
<body>
<h2>The XMLHttpRequest Object</h2>
<p>The getResponseHeader() function is used to return specific header information from a resource, like length, server-type, content-type, last-modified, etc:</p>
<p>Last modified: <span id="demo"></span></p>
<script>
const xhttp=new XMLHttpRequest();
xhttp.onload = function() {
  document.getElementById("demo").innerHTML =
  this.getResponseHeader("Last-Modified");
}
xhttp.open("GET", "ajax_info.txt");
xhttp.send();
</script>
</body>
```

```
</html>
```

AJAX XML

- AJAX ka full form: Asynchronous JavaScript and XML
- XML ek data format hai jo structured data ko represent karta hai
- AJAX me XML use hota hai jab client ko server se structured data fetch karna ho

```
<!DOCTYPE html>
<html>
<style>
table,th,td {
  border : 1px solid black;
  border-collapse: collapse;
}
th,td {
  padding: 5px;
}
</style>
<body>
<h2>The XMLHttpRequest Object</h2>
<button type="button" onclick="loadDoc()">Get my CD collection</button>
<br><br>
<table id="demo"></table>
<script>
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    myFunction(this);
  }
  xhttp.open("GET", "cd_catalog.xml");
  xhttp.send();
}
function myFunction(xml) {
  const xmlDoc = xml.responseXML;
  const x = xmlDoc.getElementsByTagName("CD");
  let table="<tr><th>Artist</th><th>Title</th></tr>";
  for (let i = 0; i <x.length; i++) {
    table += "<tr><td>" +
    x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
    "</td><td>" +
    x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
```

```
    "</td></tr>";
  }
  document.getElementById("demo").innerHTML = table;
}
</script>
</body>
</html>
```

AJAX with PHP

AJAX: Asynchronous JavaScript and XML (now JSON bhi common)

- PHP: Server-side scripting language
- AJAX ke through JavaScript asynchronously request bhejta hai PHP server ko, aur PHP response return karta hai without page reload.

```
<!DOCTYPE html>
<html>
<body>
<h2>The XMLHttpRequest Object</h2>
<h3>Start typing a name in the input field below:</h3>
<p>Suggestions: <span id="txtHint"></span></p>
<p>First name: <input type="text" id="txt1" onkeyup="showHint(this.value)"></p>
<script>
function showHint(str) {
  if (str.length == 0) {
    document.getElementById("txtHint").innerHTML = "";
    return;
  }
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    document.getElementById("txtHint").innerHTML =
    this.responseText;
  }
  xhttp.open("GET", "gethint.php?q="+str);
  xhttp.send();
}
</script>
</body>
</html>
```

AJAX with ASP

AJAX: Asynchronous JavaScript and XML
- ASP (Active Server Pages): Microsoft ka server-side scripting technology
- AJAX ke through JavaScript asynchronously request bhejta hai ASP page ko, aur ASP response return karta hai without page reload.

Use Cases:
- Form submit dynamically
- Database records fetch/update
- Real-time content update
- Dynamic dropdowns, live search

```
<!DOCTYPE html>
<html>
<body>
<h2>The XMLHttpRequest Object</h2>
<h3>Start typing a name in the input field below:</h3>
<p>Suggestions: <span id="txtHint"></span></p>
<p>First name: <input type="text" id="txt1" onkeyup="showHint(this.value)"></p>
<script>
function showHint(str) {
  if (str.length == 0) {
    document.getElementById("txtHint").innerHTML = "";
    return;
  }
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    document.getElementById("txtHint").innerHTML =
    this.responseText;
  }
  xhttp.open("GET", "gethint.asp?q="+str);
  xhttp.send();
}
</script>
</body>
</html>
```

AJAX with Database

AJAX: Asynchronous JavaScript and XML (ya JSON)
- Database: Server-side storage (MySQL, MSSQL, etc.)
- AJAX ke through JavaScript server ko request bhejta hai, server database query run karta hai, aur response return karta hai without page reload.

```
<!DOCTYPE html>
```

```
<html>
<style>
th,td {
  padding: 5px;
}
</style>
<body>
<h2>The XMLHttpRequest Object</h2>
<form action="">
  <select name="customers" onchange="showCustomer(this.value)">
    <option value="">Select a customer:</option>
    <option value="ALFKI">Alfreds Futterkiste</option>
    <option value="NORTS ">North/South</option>
    <option value="WOLZA">Wolski Zajazd</option>
  </select>
</form>
<br>
<div id="txtHint">Customer info will be listed here...</div>

<script>
function showCustomer(str) {
  if (str == "") {
    document.getElementById("txtHint").innerHTML = "";
    return;
  }
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    document.getElementById("txtHint").innerHTML = this.responseText;
  }
  xhttp.open("GET", "getcustomer.php?q="+str);
  xhttp.send();
}
</script>
</body>
</html>
```

XML Applications

XML (eXtensible Markup Language) ek structured data format hai jo data ko organize aur share karne ke liye use hota hai.

Key Applications of XML

Application Area	Description / Use
Data Transport	XML ka use client aur server ke beech data exchange ke liye hota hai (AJAX, Web Services)
Configuration Files	Software aur web applications me settings/configuration store karne ke liye XML use hota hai (e.g., web.config, pom.xml)
Data Storage	XML files me data store karke portable aur structured format maintain kiya jata hai
Web Services	SOAP (Simple Object Access Protocol) aur other web services me XML standard format hota hai
RSS Feeds	Websites me news feed, blog updates XML ke through provide kiye jaate hain
Document Representation	Structured documents like Microsoft Office, OpenOffice XML format use karte hain
Database Integration	Database se XML output generate karke other applications me share kiya ja sakta hai
Mobile & Embedded Systems	Lightweight structured data format ke liye XML ka use hota hai

Display XML Data in an HTML Table

```
<!DOCTYPE html>
<html>
<style>
table,th,td {
  border : 1px solid black;
  border-collapse: collapse;
}
th,td {
  padding: 5px;
}
</style>
<body>
<button type="button" onclick="loadXMLDoc()">Get my CD collection</button>
<br><br>
<table id="demo"></table>
<script>
function loadXMLDoc() {
```

```
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    const xmlDoc = xhttp.responseXML;
    const cd = xmlDoc.getElementsByTagName("CD");
    myFunction(cd)
  }
  xhttp.open("GET", "cd_catalog.xml");
  xhttp.send();
}
function myFunction(cd) {
  let table="<tr><th>Artist</th><th>Title</th></tr>";
  for (let i = 0; i < cd.length; i++) {
    table += "<tr><td>" +
    cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
    "</td><td>" +
    cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
    "</td></tr>";
  }
  document.getElementById("demo").innerHTML = table;
}
</script>
</body>
</html>
```

Display the First CD in an HTML div Element

```
<!DOCTYPE html>
<html>
<body>
<div id='showCD'></div>
<script>
const xhttp = new XMLHttpRequest();
xhttp.onload = function() {
  const xmlDoc = xhttp.responseXML;
  const cd = xmlDoc.getElementsByTagName("CD");
  myFunction(cd, 0);
}
xhttp.open("GET", "cd_catalog.xml");
xhttp.send();
function myFunction(cd, i) {
  document.getElementById("showCD").innerHTML =
```

```
  "Artist: " +
  cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
  "<br>Title: " +
  cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
  "<br>Year: " +
  cd[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue;
}
</script>
</body>
</html>
```

Navigate Between the CDs

```
<!DOCTYPE html>
<html>
<body>
<div id='showCD'></div><br>
<input type="button" onclick="previous()" value="<<">
<input type="button" onclick="next()" value=">>">
<script>
let i = 0;
let len;
let cd;
const xhttp = new XMLHttpRequest();
xhttp.onload = function() {
  const xmlDoc = xhttp.responseXML;
  cd = xmlDoc.getElementsByTagName("CD");
  len = cd.length;
  displayCD(i);
}
xhttp.open("GET", "cd_catalog.xml");
xhttp.send();
function displayCD(i) {
  document.getElementById("showCD").innerHTML =
  "Artist: " +
  cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
  "<br>Title: " +
  cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
  "<br>Year: " +
  cd[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue;
}
```

```
function next() {
  if (i < len-1) {
    i++;
    displayCD(i);
  }
}

function previous() {
  if (i > 0) {
    i--;
    displayCD(i);
  }
}
</script>
</body>
</html>
```

Show Album Information When Clicking On a CD

```
<!DOCTYPE html>
<html>
<head>
<style>
table,th,td {
  border : 1px solid black;
  border-collapse: collapse;
}
th,td {
  padding: 5px;
}
</style>
</head>
<body>
<p>Click on a CD to display album information.</p>
<p id='showCD'></p>
<table id="demo"></table>
<script>
const xhttp = new XMLHttpRequest();
let cd;
```

```
xhttp.onload = function() {
 const xmlDoc = xhttp.responseXML;
 cd = xmlDoc.getElementsByTagName("CD");
 loadCD();
}
xhttp.open("GET", "cd_catalog.xml");
xhttp.send();

function loadCD() {
  let table="<tr><th>Artist</th><th>Title</th></tr>";
  for (let i = 0; i < cd.length; i++) {
    table += "<tr onclick='displayCD(" + i + ")'><td>";
    table += cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue;
    table += "</td><td>";
    table += cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue;
    table += "</td></tr>";
  }
  document.getElementById("demo").innerHTML = table;
}
function displayCD(i) {
  document.getElementById("showCD").innerHTML =
  "Artist: " +
  cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
  "<br>Title: " +
  cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
  "<br>Year: " +
  cd[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue;
}
</script>
</body>
</html>
```

JavaScript JSON

JSON ka full form JavaScript Object Notation hota hai.
Yeh ek plain text format hai jo data ko store aur transport karne ke kaam aata hai.
JSON ka syntax bahut had tak JavaScript objects jaisa hota hai—matlab key-value pairs, curly braces {}, arrays [], etc.
JSON ko hum mostly data send karne, receive karne, aur database ya files me store karne ke liye use karte hain.

Isko samajhna aur use karna easy hota hai, isi liye web development me bahut popular hai.

Why JSON?

- Padna aur likhna easy hota hai
- Format lightweight hota hai
- Har programming language support karti hai
- Data exchange bahut fast hota hai
- Humans aur machines dono ke liye friendly
- APIs aur web development me widely use hota hai
- Simple key-value structure hoti hai
- Parsing aur processing bahut quick hoti hai

JSON and JavaScript

- JSON ka format bilkul JavaScript objects banane wale syntax jaisa hota hai.
- Is wajah se JavaScript program JSON data ko easily native JavaScript objects me convert kar sakta hai.
- JavaScript me ek built-in function hota hai jo JSON string ko JavaScript object me convert karta hai: JSON.parse()
- Aur ek aur built-in function hota hai jo JavaScript object ko JSON string me convert karta hai: JSON.stringify()

Storing Data

JSON ka use data ko store karne ke liye bhi bahut hota hai, especially jab hume structured format chahiye hota hai.

- JSON ek text-based format hai, isliye files me easily store ho jata hai.
- Database, configuration files, ya local storage me JSON ko bahut commonly use kiya jata hai.
- JSON me data key-value pairs ki form me store hota hai, jo readable aur manageable hota hai.
- Web apps aur mobile apps dono JSON ko data store karne ke liye prefer karti hain.

JSON Data - A Name and a Value

JSON Data – A Name and a Value (Hinglish Explanation)
JSON data hamesha name/value pair ke format me hota hai.

- Name ko hum key bhi bolte hain
- Value wo data hota hai jo key ke against store hota hai

JSON Objects

- JSON object ek curly braces { } ke andar likha hua name/value pairs ka collection hota hai.
- Har pair comma se separate hota hai.

JSON array

- JSON array ek list hoti hai jisme multiple values ko square brackets [ ] ke andar store kiya jata hai.
- Values comma se separate hoti hain.

Converting a JSON Text to a JavaScript Object

```
<!DOCTYPE html>
<html>
<body>
<h2>Create Object from JSON String</h2>
<p id="demo"></p>
<script>
let text = '{"employees":[' +
'{"firstName":"John","lastName":"Doe" },' +
'{"firstName":"Anna","lastName":"Smith" },' +
'{"firstName":"Peter","lastName":"Jones" }]}';
const obj = JSON.parse(text);

document.getElementById("demo").innerHTML =
obj.employees[1].firstName + " " + obj.employees[1].lastName;
</script>
</body>
</html>
```

JSON Syntax

- Data is in name/value pairs → JSON mein data key : value format mein hota hai, jaise "name": "Rahul"
- Data is separated by commas → Multiple key-value pairs ko comma (,) se separate karte hain, jaise "name": "Rahul", "age": 25
- Curly braces hold objects → { } ka use object banane ke liye hota hai, jaise { "name": "Rahul", "age": 25 }
- Square brackets hold arrays → [ ] ka use list/array banane ke liye hota hai, jaise ["apple", "banana", "mango"]

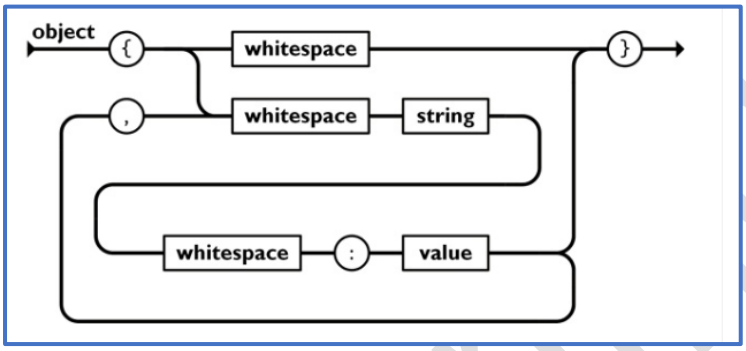
- Objects inside arrays → Arrays mein objects bhi ho sakte hain, jaise [{ "name": "Rahul" }, { "name": "Priya" }]

JSON Values

In JSON, *values* must be one of the following data types:

- a string
- a number
- an object
- an array
- a boolean
- null

JavaScript Objects




```
<!DOCTYPE html>
<html>
<body>
<h2>Modify a JavaScript Object</h2>
<p id="demo"></p>
<script>
const myObj = {name:"John", age:30, city:"New York"};
myObj["name"] = "Gilbert";
document.getElementById("demo").innerHTML = myObj.name;
</script>
</body>
</html>
```

JavaScript Arrays as JSON

- JavaScript arrays as JSON → Bilkul waise hi jaise JavaScript objects ko JSON mein likha ja sakta hai, arrays ko bhi JSON format mein likha ja sakta hai.

- Square brackets → JSON arrays ko [ ] mein likhte hain.
- Comma separation → Array ke elements ko comma (,) se alag karte hain.
- Arrays can contain objects → JSON arrays ke andar objects bhi ho sakte hain, jaise [{"name": "Rahul"}, {"name": "Priya"}].

JSON is Like XML Because

- Self-describing (Human readable) → JSON aur XML dono easily samajh aate hain, data ko read karna straightforward hai.
- Hierarchical structure → Dono ka structure nested values ka support karta hai, matlab values ke andar aur values ho sakti hain.
- Language support → JSON aur XML dono ko bahut saari programming languages parse aur use kar sakti hain.
- Fetchable via XMLHttpRequest → Dono ko web se fetch kiya ja sakta hai using JavaScript ke XMLHttpRequest ya modern fetch API.

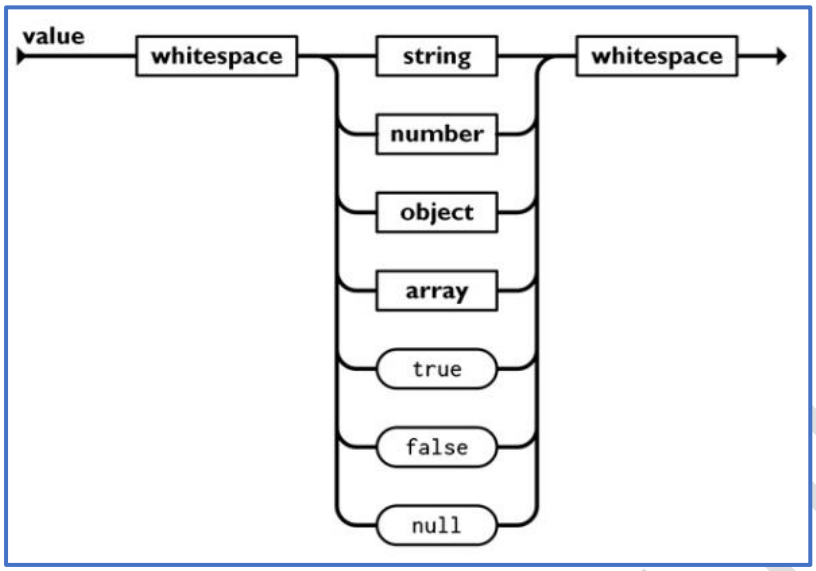
Agar chaho toh main ek chhota example de sakta hoon jisme same data JSON aur XML mein dikhaya gaya ho, taaki similarity aur clear ho jaye.

Why JSON is Better Than XML

- Lightweight → JSON XML se chhota aur concise hota hai, extra tags nahi hote.
- Easier to read and write → Simple { } aur [ ] syntax, XML ke <tag></tag> se easy.
- Faster parsing → JavaScript mein JSON.parse() se directly parse hota hai, XML parsing complex hoti hai.
- Supports native data types → Numbers, strings, booleans, arrays, objects directly support, XML mein sab strings hote hain.
- Better for web apps → Modern web APIs aur JavaScript apps JSON prefer karte hain.
- Less bandwidth → JSON chhota hota hai, network pe data transfer fast aur efficient hota hai.

JSON Data Types

- Valid Data Types
- In JSON, values must be one of the following data types:
- a string
- a number
- an object (JSON object)
- an array
- a boolean
- null



JSON.parse()

- Data exchange → JSON ka common use web server se data bhejne aur receive karne ke liye hota hai.
- Data as string → Web server se jo data aata hai, woh hamesha string ke form mein hota hai.
- Parsing JSON → JSON.parse() ka use karke string data ko JavaScript object mein convert karte hain.
- Use in JS → Parse karne ke baad data ko normal JS object ki tarah access aur manipulate kar sakte hain.
- Agar chaho toh main chhota example bhi de sakta hoon jisme JSON string ko parse karke object banaya gaya ho.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript JSON</h1>
<h2>Creating an Object from JSON</h2>
<p id="demo"></p>
<script>
const txt = '{"name":"John", "age":30, "city":"New York"}'
const myObj = JSON.parse(txt);
document.getElementById("demo").innerHTML = myObj.name + ", " + myObj.age;
```

```
</script>
</body>
</html>
```

Array as JSON

- JSON Array → JavaScript ki array ko JSON format mein likha ja sakta hai.
- Square brackets → Arrays hamesha [ ] mein likhe jaate hain.
- Elements separated by commas → Array ke elements ko comma (,) se alag karte hain.

```
<!DOCTYPE html>
<html>
<body>
<h2>Parsing a JSON Array.</h2>
<p>Data written as an JSON array will be parsed into a JavaScript array.</p>
<p id="demo"></p>
<script>
const text = '[ "Ford", "BMW", "Audi", "Fiat" ]';
const myArr = JSON.parse(text);
document.getElementById("demo").innerHTML = myArr[0];
</script>
</body>
</html>
```

Parsing Dates

JSON format → JSON mein dates ko directly date type ke form mein store nahi kiya ja sakta, usually string format mein store karte hain.

```
<!DOCTYPE html>
<html>
<body>
<h2>Convert a string into a date object.</h2>
<p id="demo"></p>
<script>
const text = '{"name":"John", "birth":"1986-12-14", "city":"New York"}';
const obj = JSON.parse(text);
obj.birth = new Date(obj.birth);
document.getElementById("demo").innerHTML = obj.name + ", " + obj.birth;
</script>
</body>
</html>
```

Parsing Functions

- Functions in JSON → JSON directly functions store nahi kar sakta, kyunki JSON sirf data ka format hai (strings, numbers, arrays, objects, booleans, null).
- Receiving functions → Agar server se function ka code string ke form mein aata hai, toh use JavaScript function mein convert karna padta hai.
- Parsing function string → eval() ya Function() constructor ka use karke string ko function banaya ja sakta hai (caution: security risk).

```
<!DOCTYPE html>
<html>
<body>
<h2>Convert a string into a function.</h2>
<p id="demo"></p>
<script>
const text = '{"name":"John", "age":"function() {return 30;}", "city":"New York"}';
const obj = JSON.parse(text);
obj.age = eval("(" + obj.age + ")");
document.getElementById("demo").innerHTML = obj.name + ", " + obj.age();
</script>
</body>
</html>
```

JSON.stringify()

JSON.stringify() ka use JavaScript object ya array ko JSON string mein convert karne ke liye hota hai.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript JSON</h1>
<h2>Create a JSON string from ant object.</h2>
<p id="demo"></p>
<script>
const myObj = {name: "John", age: 30, city: "New York"};
const myJSON = JSON.stringify(myObj);
document.getElementById("demo").innerHTML = myJSON;
</script>
</body>
</html>
```

Stringify a JavaScript Array

```
<!DOCTYPE html>
```

```
<html>
<body>
<h2>Create a  JSON string from a JavaScript array.</h2>
<p id="demo"></p>
<script>
const arr = ["John", "Peter", "Sally", "Jane"];
const myJSON = JSON.stringify(arr);
document.getElementById("demo").innerHTML = myJSON;
</script>
</body>
</html>
```

Storing Data

```
<!DOCTYPE html>
<html>
<body>
<h2>Store and retrieve data from local storage.</h2>
<p id="demo"></p>
<script>
// Storing data:
const myObj = { name: "John", age: 31, city: "New York" };
const myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);
// Retrieving data:
let text = localStorage.getItem("testJSON");
let obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
</script>
</body>
</html>
```

Stringify a Date

```
<!DOCTYPE html>
<html>
<body>
<h2>JSON.stringify() converts date objects into strings.</h2>
<p id="demo"></p>
<script>
const obj = {name: "John", today: new Date(), city: "New York"};
```

```
const myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
</script>
</body>
</html>
```

Stringify a Function

```
<!DOCTYPE html>
<html>
<body>
<h2>JSON.stringify() will remove any functions from an object.</h2>
<p>Convert functions into strings to keep them in the JSON object.</p>
<p id="demo"></p>
<script>
const obj = {name: "John", age: function () {return 30;}, city: "New York"};
obj.age = obj.age.toString();
const myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
</script>
</body>
</html>
```

JSON Array Literals

- Definition → JSON array literal ek square brackets [ ] mein likha gaya list of values hota hai.
- Elements → Array ke elements strings, numbers, booleans, objects, arrays, null ho sakte hain.
- Comma separation → Elements ko comma (,) se alag karte hain.
- Example (simple array)

```
<!DOCTYPE html>
<html>
<body>
<h2>Looping an Array</h2>
<p id="demo"></p>
<script>
const myJSON = '{"name":"John", "age":30, "cars":["Ford", "BMW", "Fiat"]}';
const myObj = JSON.parse(myJSON);
let text = "";
```

```
for (let i = 0; i < myObj.cars.length; i++) {
  text += myObj.cars[i] + ", ";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

JSON Server

JSON ka ek common use hota hai data ko web server se bhejne ya receive karne ke liye.
Jab hum server se data receive karte hain, toh data hamesha string format me hota hai — matlab ek normal text jaisa.
Ab agar humein is string ko apne JavaScript code me use karna ho, toh humein usko object me convert karna padta hai.

```
<!DOCTYPE html>
<html>
<body>
<h2>Convert a JavaScript object into a JSON string, and send it to the server.</h2>
<script>
const myObj = { name: "John", age: 31, city: "New York" };
const myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;
</script>
</body>
</html>
```

Receiving Data

Server se jo information aapki website ya app ko milti hai, usko receive karna.

```
<!DOCTYPE html>
<html>
<body>
<h2>Convert a JSON string into a JavaScript object.</h2>
<p id="demo"></p>
<script>
const myJSON = '{"name":"John", "age":31, "city":"New York"}';
const myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.name;
</script>
</body>
```

```
</html>
```

JSON From a Server

Server hamesha JSON data ko string ki tarah send karta hai. JavaScript me directly us string ko use nahi kar sakte, isliye hume use object me convert karna padta hai.

```
<!DOCTYPE html>
<html>
<body
<h2>Fetch a JSON file with XMLHttpRequest</h2>
<p id="demo"></p>
<script>
const xmlhttp = new XMLHttpRequest();
xmlhttp.onload = function() {
  const myObj = JSON.parse(this.responseText);
  document.getElementById("demo").innerHTML = myObj.name;
}
xmlhttp.open("GET", "json_demo.txt");
xmlhttp.send();
</script>
</body>
</html>
```

Array as JSON

Aap ek JavaScript array ko bhi JSON format me store ya send kar sakte ho. JSON me array ko bilkul JavaScript array ki tarah hi likha jata hai, bus wo string format me hota hai.

```
<!DOCTYPE html>
<html>
<body>
<h2>Fetch a JSON file with XMLHttpRequest</h2>
<p>Content written as an JSON array will be converted into a JavaScript array.</p>
<p id="demo"></p>
<script>
const xmlhttp = new XMLHttpRequest();
xmlhttp.onload = function() {
  const myArr = JSON.parse(this.responseText);
  document.getElementById("demo").innerHTML = myArr[0];
}
xmlhttp.open("GET", "json_demo_array.txt", true);
```

```
xmlhttp.send();
</script>
</body>
</html>
```

JSON PHP

JSON ka ek common use hai web server se data read karna, aur phir us data ko web page par display karna.
Yaani client (browser) server se JSON format me data leta hai, use JavaScript se parse karta hai, aur HTML me show kar deta hai.

The PHP File

Jab hum client (browser) aur server (PHP) ke beech JSON data exchange karte hain, toh PHP file ka main kaam hota hai JSON format me data prepare karna aur client ko send karna.
PHP file backend me data generate karta hai—ye data database se aa sakta hai, array se, ya manually banaya hua ho sakta hai.

```
<?php
$myObj = new stdClass();
$myObj->name = "John";
$myObj->age = 30;
$myObj->city = "New York";
$myJSON = json_encode($myObj);
echo $myJSON;
?>
```

The Client JavaScript

Client side JavaScript ka kaam hota hai PHP server se JSON data fetch karna, use JavaScript object me convert karna, aur phir web page par display karna.
Yaani browser → PHP server → JSON data → JavaScript → Webpage.

```
<!DOCTYPE html>
<html>
<body>
<h2>Get JSON Data from a PHP Server</h2>
<p id="demo"></p>
<script>
const xmlhttp = new XMLHttpRequest();

xmlhttp.onload = function() {
  const myObj = JSON.parse(this.responseText);
```

```
  document.getElementById("demo").innerHTML = myObj.name;
}
xmlhttp.open("GET", "demo_file.php");
xmlhttp.send();
</script>
</body>
</html>
```

PHP Array

PHP me array ek data structure hota hai jisme aap multiple values store kar sakte ho—jaise list, items, key-value pairs, etc.
JSON bhejne se pehle hum aksar PHP array banate hain, phir usko JSON me convert karke client ko send karte hain.
PHP me do common types ke arrays hote hain:

1. Indexed Array (normal list jisme index 0,1,2… hote hain)
2. Associative Array (key–value pair jaisa object hota hai)

```
<?php
$myArr = array("John", "Mary", "Peter", "Sally");
$myJSON = json_encode($myArr);
echo $myJSON;
?>
```

The Client JavaScript

Client-side JavaScript ka main kaam hota hai PHP server se JSON data fetch karna, us data ko JavaScript object me convert karna, aur phir webpage par show karna.
Yeh poora process browser me hota hai.

```
<!DOCTYPE html>
<html>
<body>
<h2>Get JSON Data from a PHP Server</h2>
<p>Convert the data into a JavaScript array:</p>
<p id="demo"></p>
<script>
const xmlhttp = new XMLHttpRequest();
xmlhttp.onload = function() {
  const myObj = JSON.parse(this.responseText);
  document.getElementById("demo").innerHTML = myObj[2];
}
xmlhttp.open("GET", "demo_file_array.php");
xmlhttp.send();
```

```
</script>
</body>
</html>
```

PHP Database

PHP ek server-side language hai, jiska use hum database access karne ke liye karte hain.
Maan lo server par ek database hai jisme customers naam ka table hai, aur aap client (browser) se request bhejna chahte ho ki:
“Mujhe customers table ki pehli 10 rows chahiye.”
Iske liye client side JavaScript pe hum ek JSON object banate hain jo batata hai ki kitni rows chahiye.
Phir JSON object ko string me convert karke PHP page ke URL me parameter ke taur pe send kar dete hain.

```
<!DOCTYPE html>
<html>
<body>
<h2>Get JSON Data from a PHP Server</h2>
<p>The JSON received from the PHP file:</p
<p id="demo"></p>
<script>
const dbParam = JSON.stringify({"limit":10});
const xmlhttp = new XMLHttpRequest();
xmlhttp.onload = function() {
  document.getElementById("demo").innerHTML = this.responseText;
}
xmlhttp.open("GET", "json_demo_db.php?x=" + dbParam);
xmlhttp.send();
</script>
</body>
</html>
```

PHP Method = POST

Jab hum server ko data bhejte hain, toh POST method aksar better hota hai.
POST use karne ka reason:

- URL me data nahi dikhata (safer)
- Large data bhejna possible
- Form data aur JSON data easily send kar sakte ho

AJAX request me POST method use karne ke liye:

1. method: "POST" specify karna

2. Correct Content-Type header set karna (application/json)
3. Data ko send() method ka argument banakar bhejna

```
<!DOCTYPE html>
<html>
<body>
<h2>Use HTTP POST to Get JSON Data from a PHP Server</h2>
<p id="demo"></p>
<script>
const dbParam = JSON.stringify({"limit":10});
const xmlhttp = new XMLHttpRequest();
xmlhttp.onload = function() {
  myObj = JSON.parse(this.responseText);
  let text = "";
  for (let x in myObj) {
    text += myObj[x].name + "<br>";
  }
  document.getElementById("demo").innerHTML = text;
}
xmlhttp.open("POST", "json_demo_db_post.php");
xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xmlhttp.send("x=" + dbParam);
</script>
<p>Try changing the "limit" property from 10 to 5.</p>
</body>
</html>
```

JSON HTML

JSON ka main use hota hai data exchange ke liye, lekin HTML me hum JSON ko display kar sakte hain ya HTML elements ke saath dynamically use kar sakte hain.
Yaani:

- Server se JSON data aata hai
- JavaScript us JSON ko parse karta hai
- HTML page me data show karta hai

HTML Table

Make an HTML table with data received as JSON:

```
<!DOCTYPE html>
<html>
<body>
```

```
<h2>Make a table based on JSON data.</h2>
<p id="demo"></p>
<script>
const dbParam = JSON.stringify({table:"customers",limit:20});
const xmlhttp = new XMLHttpRequest();
xmlhttp.onload = function() {
  const myObj = JSON.parse(this.responseText);
  let text = "<table border='1'>"
  for (let x in myObj) {
    text += "<tr><td>" + myObj[x].name + "</td></tr>";
  }
  text += "</table>"
  document.getElementById("demo").innerHTML = text;
}
xmlhttp.open("POST", "json_demo_html_table.php");
xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xmlhttp.send("x=" + dbParam);
</script>
</body>
</html>
```

HTML Drop Down List

Make an HTML drop down list with data received as JSON:

```
<!DOCTYPE html>
<html>
<body>
<h2>Make a drop down list based on JSON data.</h2>
<p id="demo"></p>
<script>
const dbParam = JSON.stringify({table:"customers",limit:20});
const xmlhttp = new XMLHttpRequest();
xmlhttp.onload = function() {
  const myObj = JSON.parse(this.responseText);
  let text = "<select>"
  for (let x in myObj) {
    text += "<option>" + myObj[x].name + "</option>";
  }
  text += "</select>"
  document.getElementById("demo").innerHTML = text;
}
xmlhttp.open("POST", "json_demo_html_table.php");
```

```
xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xmlhttp.send("x=" + dbParam);
</script>
</body>
</html>
```

JSONP

JSONP ka full form hai JSON with Padding.
Ye ek technique hai jo browser me cross-domain requests ko allow karne ke liye use hoti hai, jab normal AJAX (XMLHttpRequest/fetch) same-origin policy ke karan fail ho jata hai.

1 Problem

- Browser security ke rules ke according, JavaScript normally sirf usi domain se data fetch kar sakta hai jahan se page load hua hai.
- Agar aap kisi dusre domain (cross-domain) se JSON data lena chahte ho, normal AJAX fail ho jata hai.

2 JSONP Ka Idea

- JSONP me server data ko JavaScript function call ke form me send karta hai.
- Client is function ko define karta hai aur data function argument me milta hai.
- Browser script tag ke through JSONP file load kar leta hai, aur function automatically call ho jata hai.

The Server File

Server file ka kaam hota hai client ke request ko handle karna, data fetch karna (database ya array se), aur phir JSON ya JSONP format me client ko send karna.

```
<?php
$myJSON = '{"name":"John", "age":30, "city":"New York"}';
echo "myFunc(".$myJSON.");";
?>
```

The JavaScript function

Client-side JavaScript function ka kaam hota hai server se aane wale JSON ya JSONP data ko handle karna aur phir HTML ya console me display karna.
Ye function JSONP ke case me specially important hota hai, kyunki server data ko function call ke form me bhejta hai.

```
<!DOCTYPE html>
<html>
<body>
```

```
<h2>Request JSON using the script tag</h2>
<p>The PHP file returns a call to a function that will handle the JSON data.</p>
<p id="demo"></p>
<script>
function myFunc(myObj) {
  document.getElementById("demo").innerHTML = myObj.name;
}
</script>
<script src="demo_jsonp.php"></script>
</body>
</html>
```

Creating a Dynamic Script Tag

JSONP me hum cross-domain requests karte hain, aur uske liye dynamic script tag create karna padta hai.
Yaani hum JavaScript se programmatically <script> element banate hain jo server se JSONP data load kare.

```
<!DOCTYPE html>
<html>
<body>
<h2>Click the Button.</h2>
<p>A script tag with a src attribute is created and placed in the document.</p>
<p>The PHP file returns a call to a function with the JSON object as a parameter.</p>
<button onclick="clickButton()">Click me!</button>
<p id="demo"></p>
<script>
function clickButton() {
  let s = document.createElement("script");
  s.src = "demo_jsonp.php";
  document.body.appendChild(s);
}
function myFunc(myObj) {
  document.getElementById("demo").innerHTML = myObj.name;
}
</script>
</body>
</html>
```

Dynamic JSONP Result

Ab tak ke JSONP examples static data use kar rahe the, matlab server hamesha same data bhej raha tha.
“Dynamic JSONP” ka matlab hai:
Server real-time data bhej raha hai, jo client ke request parameters par depend karta hai.
Yaani, har request ke according JSON data dynamically change hota hai.

```
<!DOCTYPE html>
<html>
<body>
<p>A script tag with a src attribute is created and placed in the document.</p>
<p>The PHP file returns a call to a function with an object as a parameter.</p>
<p id="demo"></p>
<p>Try changing the table property from "customers" to "products".</p>
<script>
const obj = { table: "customers", limit: 10 };
let s = document.createElement("script");
s.src = "jsonp_demo_db.php?x=" + JSON.stringify(obj);
document.body.appendChild(s);
function myFunc(myObj) {
  let txt = "";
  for (let x in myObj) {
    txt += myObj[x].name + "<br>";
  }
  document.getElementById("demo").innerHTML = txt;
}
</script>
</body>
</html>
```

Callback Function

JSONP me callback function ka role bahut important hai.
Server JSON data ko directly return nahi karta, balki function call ke form me wrap karke client ko bhejta hai.

1 Concept

1. Client ek JavaScript function define karta hai, jise hum callback function kehte hain.
2. Client dynamic <script> tag create karta hai aur server URL me callback function ka naam bhejta hai.
3. Server JSON ko callbackFunctionName({...}) ke form me return karta hai.
4. Browser script load karte hi callback function automatically call ho jata hai aur data argument me pass ho jata hai.

```
<!DOCTYPE html>
<html>
<body>
```

```
<h2>Request With a Callback Function</h2>
<p>The PHP file returns a call to the function you send as a callback.</p>
<p id="demo"></p>
<script>
let s = document.createElement("script");
s.src = "demo_jsonp2.php?callback=myDisplayFunction";
document.body.appendChild(s);
function myDisplayFunction(myObj) {
  document.getElementById("demo").innerHTML = myObj.name;
}
</script>
</body>
</html>
```

jQuery vs JavaScript

jQuery

- Created by John Resig in 2006
- Simplifies:
  - HTML DOM manipulation
  - Event handling
  - Animations
  - Ajax requests
- Handles browser incompatibilities automatically
- Most popular JS library for 10+ years
- Example: $("#myDiv").hide();

Vanilla JavaScript (Modern JS)

- After JavaScript ES5 (2009), most jQuery features can be done with standard JS
- Better browser support in modern browsers
- No extra library required → lightweight
- Example: document.getElementById("myDiv").style.display = "none";

Finding HTML Element by Id

Agar aapko specific element select karna ho jiska unique ID ho, toh JavaScript me getElementById() use karte hain.

```
<!DOCTYPE html>
<html>
<body>
<h2>Finding HTML Elements by Id</h2>
<p id="id01">Hello World!</p>
<p id="id02">Hello Sweden!</p>
```

```
<p id="id03">Hello Japan!</p>
<p id="demo"></p>
<script>
const myElement = document.getElementById("id01");
document.getElementById("demo").innerHTML = "The text from the id01 paragraph is: " +
myElement.innerHTML;
</script>
</body>
</html>
```

Finding HTML Elements by Tag Name

HTML me agar aapko specific tag ke elements select karne hain (jaise all <p> ya), toh JavaScript me getElementsByTagName() ka use hota hai.

```
<!DOCTYPE html>
<html>
<body>
<h2>Finding HTML Elements by Tag Name</h2>
<p>Hello World!</p>
<p>Hello Sweden!</p>
<p>Hello Japan!</p>
<p id="demo"></p>
<script>
const myElements = document.getElementsByTagName("p");
document.getElementById("demo").innerHTML = "The text in the first paragraph is: " +
myElements[0].innerHTML;
</script>
</body>
</html>
```

Finding HTML Elements by Class Name

Agar aapko ek ya ek se zyada elements select karne ho jinka common class ho, toh JavaScript me getElementsByClassName() use karte hain.

```
<!DOCTYPE html>
<html>
<body>
<h2>Finding HTML Elements by Class Name</h2>
<p class="intro">Hello World!</p>
<p class="intro">Hello Sweden!</p>
<p class="intro">Hello Japan!</p>
<p id="demo"></p>
<script>
```

```
const myElements = document.getElementsByClassName("intro");
document.getElementById("demo").innerHTML = "The first paragraph with class='intro' is: " +
myElements[0].innerHTML;
</script>
</body>
</html>
```

Finding HTML Elements by CSS Selectors

JavaScript me agar aapko complex selection karni ho (jaise id, class, tag, nested elements, attribute selectors), toh CSS selectors ka use karke elements select kar sakte ho.

- Iske liye do main methods hain:
  1. querySelector() → pehla matching element return karta hai
  2. querySelectorAll() → sab matching elements return karta hai (NodeList)

```
<!DOCTYPE html>
<html>
<body>

<h2>Finding HTML Elements by Query Selector</h2>
<p class="intro">Hello World!</p>
<p class="intro">Hello Sweden!</p>
<p class="intro">Hello Japan!</p>
<p id="demo"></p>
<script>
const myElements = document.querySelectorAll("p.intro");
document.getElementById("demo").innerHTML =
"The first paragraph with class='intro' is: " + myElements[0].innerHTML;
</script>
</body>
</html>
```

Set Text Content

Agar aap HTML element ka text content change karna chahte ho, toh JavaScript me textContent ya innerText use kar sakte ho.

```
<!DOCTYPE html>
<html>
<body>
<h2>Setting Text Content</h2>
<p id="01">Hello World!</p>
<p id="02">Hello Sweden!</p>
<script>
const myElement = document.getElementById("01");
```

```
myElement.textContent = "Hello Sweden!";
</script>
</body>
</html>
```

Get Text Content

Agar aapko HTML element ka current text read karna hai, toh JavaScript me textContent ya innerText use kar sakte ho.

```
<!DOCTYPE html>
<html>
<body>
<h2>Get Text Content</h2>
<p id="01">Hello World!</p>
<p id="02">Hello Sweden!</p>
<p id="03">Hello Japan!</p>

<p id="demo"></p>
<script>
const myText = document.getElementById("02").textContent;
document.getElementById("demo").innerHTML = myText;
</script>
</body>
</html>
```

Set HTML Content

Agar aap HTML element ke andar ka content (including HTML tags) dynamically change karna chahte ho, toh innerHTML use karte hain.

```
<!DOCTYPE html>
<html>
<body>
<h2>Set HTML</h2>
<div id="01"><p>Hello World!</p></div>
<div id="02"><p>Hello Sweden!</p></div>
<p id="demo"></p>
<script>
document.getElementById("02").innerHTML = "<p>Hello World!</p>";
</script>
</body>
</html>
```

Get HTML Content

Agar aapko HTML element ke andar ka content including HTML tags read karna hai, toh innerHTML use karte hain.

```
<!DOCTYPE html>
<html>
<body>
<h2>Setting HTML</h2>
<div id="01"><p>Hello World!</p></div>
<div id="02"><p>Hello Sweden!</p></div>
<script>
const content = document.getElementById("02").innerHTML;
document.getElementById("01").innerHTML = content;
</script>
</body>
</html>
```

Hiding HTML Elements

Using style.display

- display: none → element completely hide ho jata hai (space bhi remove ho jata hai)
- display: block ya original value → show karne ke liye.

```
<!DOCTYPE html>
<html>
<body>
<h2>Hide HTML Elements</h2>
<p id="01">Hello World!</p>
<p id="02">Hello Sweden!</p>
<p id="03">Hello Japan!</p>
<script>
document.getElementById("02").style.display = "none";
</script>
</body>
</html>
```

Showing HTML Elements

Using style.display

- Agar element display: none se hidden hai, wapas show karne ke liye original display value set karo (block, inline, etc.)

```
<!DOCTYPE html>
```

```
<html>
<body>
<h2>Show HTML Elements</h2>
<p id="01" style="display:none">Hello World!</p>
<p id="02" style="display:none">Hello Sweden!</p>
<p id="03" style="display:none">Hello Japan!</p>
<script>
document.getElementById("02").style.display = "";
</script>
</body>
</html>
```

Styling HTML Elements

JavaScript me aap HTML elements ke styles dynamically change kar sakte ho using the style property ya CSS classes.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Change the style of an HTML element.</p>
<script>
document.getElementById("demo").style.fontSize = "35px";
</script>
</body>
</html>
```

jQuery HTML DOM
Removing HTML Elements

JavaScript me aap existing HTML elements ko completely DOM se remove kar sakte ho using remove() ya removeChild().

```
<!DOCTYPE html>
<html>
<body>
<h2>Remove an HTML Element</h2>
<p id="id01">Hello World!</p>
<p id="id02">Hello Sweden!</p>
<script>
```

```
document.getElementById("id02").remove();
</script>
</body>
</html>
```

Get Parent Element

- Returns the parent node of an element
- Usually same as parentElement for HTML elements

```
<!DOCTYPE html>
<html>
<body>
<h2>Get Parent HTML Element</h2>
<p id="01">Hello World!</p>
<p id="02">Hello Sweden!</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
document.getElementById("02").parentNode.nodeName;
</script>
</body>
</html>
```

JavaScript Graphics

Graphic Libraries
JavaScript libraries to use for all kinds of graphs:

- Plotly.js
- Chart.js
- Google Chart

HTML Canvas

HTML Canvas ek HTML element hai jisme hum graphics draw kar sakte hain, jaise ki: shapes, lines, images, aur even animations.
Socho yeh ek blank drawing board ki tarah hai jo browser mein dikhta hai.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
<title>Canvas Multi-Ball Animation</title>
<style>
  body {
    text-align: center;
    margin-top: 50px;
    background-color: #f0f0f0;
  }
  canvas {
    border: 2px solid #000;
    background-color: #fff;
  }
</style>
</head>
<body>

<h2>HTML Canvas: Multi-Ball Bouncing Animation</h2>
<canvas id="myCanvas" width="600" height="400"></canvas>
<script>
// Canvas and context
const canvas = document.getElementById("myCanvas");
const ctx = canvas.getContext("2d");
// Ball class
class Ball {
  constructor(x, y, dx, dy, radius, color) {
    this.x = x;
    this.y = y;
    this.dx = dx;
    this.dy = dy;
    this.radius = radius;
    this.color = color;
  }
  draw() {
    ctx.beginPath();
    ctx.arc(this.x, this.y, this.radius, 0, Math.PI*2);
    ctx.fillStyle = this.color;
    ctx.fill();
    ctx.strokeStyle = "#000";
    ctx.stroke();
    ctx.closePath();
  }
  update() {
```

```
    // Bounce off walls
    if (this.x + this.radius > canvas.width || this.x - this.radius < 0) {
      this.dx = -this.dx;
    }
    if (this.y + this.radius > canvas.height || this.y - this.radius < 0) {
      this.dy = -this.dy;
    }
    // Move ball
    this.x += this.dx;
    this.y += this.dy;

    this.draw();
  }
}

// Generate random color
function getRandomColor() {
  const r = Math.floor(Math.random()*256);
  const g = Math.floor(Math.random()*256);
  const b = Math.floor(Math.random()*256);
  return `rgb(${r},${g},${b})`;
}
// Create multiple balls
const balls = [];
for (let i = 0; i < 10; i++) {
  let radius = Math.floor(Math.random() * 20) + 10;
  let x = Math.random() * (canvas.width - 2*radius) + radius;
  let y = Math.random() * (canvas.height - 2*radius) + radius;
  let dx = (Math.random() - 0.5) * 4; // horizontal speed
  let dy = (Math.random() - 0.5) * 4; // vertical speed
  let color = getRandomColor();
  balls.push(new Ball(x, y, dx, dy, radius, color));
}
// Animation loop
function animate() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  balls.forEach(ball => ball.update());
  requestAnimationFrame(animate);
}

// Start animation
```

```
animate();
</script>
</body>
</html>
```

Plotly.js kya hai?

- Plotly.js ek JavaScript charting library hai.
- Isse hum interactive aur high-quality charts browser me directly bana sakte hain.
- Ye open-source aur free hai under MIT license.

Charts ke types

Plotly.js me 40+ chart types available hain. Main popular ones yeh hain:

1. Bar Charts – Horizontal & Vertical
2. Pie & Donut Charts
3. Line Charts – Trend dikhane ke liye
4. Scatter & Bubble Plots – Correlation ya distribution ke liye
5. Equation Plots – Mathematical functions plot karne ke liye
6. 3D Charts – Surface, mesh, 3D scatter
7. Statistical Graphs – Box plot, histogram, violin plot
8. SVG Maps – Choropleth maps ya geographical plots

…and bohot saare aur charts bhi available hain.

Plotly.js kyun use karein

- Interactive – Zoom, pan, hover, click events built-in
- Customizable – Colors, labels, styles easily change kar sakte ho
- Integrates easily – React, Angular, Vue ya plain JS ke saath use kar sakte ho
- Open-source – GitHub pe contribute kar sakte ho, issues report kar sakte ho

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Plotly.js Example</title>
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
</head>
<body>
<h2>1. Line Chart</h2>
<div id="lineChart" style="width:600px;height:400px;"></div>
<h2>2. Bar Chart</h2>
<div id="barChart" style="width:600px;height:400px;"></div>
<h2>3. Pie Chart</h2>
<div id="pieChart" style="width:600px;height:400px;"></div
```

```
<script>
// ========================
// 1. Line Chart
// ========================
var lineData = [{
    x: [1, 2, 3, 4, 5],      // X-axis values
    y: [10, 15, 13, 17, 21], // Y-axis values
    type: 'scatter',         // scatter = line chart
    mode: 'lines+markers',    // line ke saath points dikhaye
    name: 'Sales'
}];

Plotly.newPlot('lineChart', lineData, {title: 'Line Chart Example'});

// ========================
// 2. Bar Chart
// ========================
var barData = [{
    x: ['Apples', 'Bananas', 'Cherries', 'Dates'],
    y: [12, 19, 9, 25],
    type: 'bar',
    name: 'Fruits'
}];
Plotly.newPlot('barChart', barData, {title: 'Bar Chart Example'});

// ========================
// 3. Pie Chart
// ========================
var pieData = [{
    labels: ['Red', 'Blue', 'Yellow', 'Green'],
    values: [10, 15, 30, 45],
    type: 'pie',
    name: 'Colors'
}];
Plotly.newPlot('pieChart', pieData, {title: 'Pie Chart Example'});
</script>
</body>
</html>
```

Chart.js kya hai?

- Chart.js ek free JavaScript library hai charts banane ke liye.
- Ye HTML <canvas> element ke upar charts draw karta hai.
- Ye simple aur beginner-friendly hai, aur interactive charts bhi bana sakte ho.

Built-in chart types

Chart.js me kaafi chart types built-in hote hain:

1. Scatter Plot – X-Y points dikhane ke liye
2. Line Chart – Trend ya time series data ke liye
3. Bar Chart – Vertical ya horizontal bars
4. Pie Chart – Percentage distribution ke liye
5. Donut Chart – Pie ka variation, center hole ke saath
6. Bubble Chart – X-Y ke saath size represent karne ke liye
7. Area Chart – Line chart ka filled version
8. Radar Chart – Multiple variables ka comparison
9. Mixed Chart – Bar + Line ya multiple chart types ek saath

Chart.js ke advantages

- Simple & Lightweight – Easy to learn, beginner-friendly
- Responsive – Mobile aur browser friendly
- Customizable – Colors, labels, tooltips, animation easily change kar sakte ho
- Free & Open Source – MIT license

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Line Chart Example</title>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
<h2>Line Chart</h2>
<canvas id="myChart" width="400" height="200"></canvas>
<script>
new Chart(document.getElementById('myChart'), {
  type: 'line', // chart type
  data: {
    labels: ['Jan','Feb','Mar','Apr','May'], // X-axis labels
    datasets: [{
      label: 'Sales',          // chart label
      data: [12, 19, 15, 17, 22], // Y-axis values
      borderColor: 'blue',      // line color
      backgroundColor: 'rgba(0,0,255,0.1)', // fill color
      fill: true               // line ke neeche fill
    }]
```

```
  }
});
</script>
</body>
</html>
```

Function Graphs

```
<!DOCTYPE html>
<html>
<script src="https://cdn.jsdelivr.net/npm/chart.js@4.5.0"></script>
<body>
<canvas id="myChart" style="width:100%;max-width:600px"></canvas>

<script>
const xValues = [];
const yValues = [];
generateData("Math.sin(x)", 0, 10, 0.5);
const ctx = document.getElementById('myChart');
new Chart(ctx, {
type: 'line',
data: {
  labels: xValues,
  datasets: [{
    label: 'y = sin(x)',
    data: yValues,
    borderColor: 'rgba(0,0,255,0.5)',
    pointRadius: 1,
    tension: 0.4
  }]
},
options: {
  plugins: {
    legend: { display: false },
    title: {
      display: true,
      text: 'y = sin(x)',
      font: { size: 16 }
    },
  },
  scales: {
```

```
    x: {title: { display: true, text: 'x' }},
    y: {title: { display: true, text: 'sin(x)'}
   }
}
}});
function generateData(value, i1, i2, step = 1) {
  for (let x = i1; x <= i2; x += step) {
    yValues.push(eval(value));
    xValues.push(x);
  }
}
</script>
</body>
</html>
```

Google Chart

- Google Charts ek free JavaScript library hai charts banane ke liye.
- Ye interactive aur customizable charts create karta hai directly browser me.
- Data arrays ya Google Sheets se le sakte ho.
- Charts SVG ya HTML5 canvas ke upar render hote hain.

Supported Chart Types

Kuch popular chart types:

1. Line Chart
2. Bar Chart / Column Chart
3. Pie Chart
4. Donut Chart
5. Area Chart
6. Scatter Chart
7. Combo Chart (Mixed charts)
8. Geo Chart / Map

D3.js

- D3.js ka full form hai Data-Driven Documents.
- Ye ek JavaScript library hai jo data ko HTML, SVG, aur CSS ke saath bind karke interactive visualizations banata hai.
- Ye bohot flexible hai, matlab tum almost koi bhi custom chart create kar sakte ho.
- Unlike Chart.js ya Google Charts, D3 built-in chart types nahi deta, lekin tum apna chart design kar sakte ho.

```
<!DOCTYPE html>
<html>
```

```
<script src="https://d3js.org/d3.v4.js"></script>
<body>
<h2>D3.js Scatter-Plot</h2>
<svg id="myPlot" style="width:500px;height:500px"></svg
<script>
// Set Dimensions
const xSize = 500;
const ySize = 500;
const margin = 40;
const xMax = xSize - margin*2;
const yMax = ySize - margin*2;
// Create Random Points
const numPoints = 100;
const data = [];
for (let i = 0; i < numPoints; i++) {
  data.push([Math.random() * xMax, Math.random() * yMax]);
}
// Append SVG Object to the Page
const svg = d3.select("#myPlot")
  .append("svg")
  .append("g")
  .attr("transform","translate(" + margin + "," + margin + ")");

// X Axis
const x = d3.scaleLinear()
  .domain([0, 500])
  .range([0, xMax]);
svg.append("g")
  .attr("transform", "translate(0," + yMax + ")")
  .call(d3.axisBottom(x));
// Y Axis
const y = d3.scaleLinear()
  .domain([0, 500])
  .range([ yMax, 0])
svg.append("g")
  .call(d3.axisLeft(y))
// Dots
svg.append('g')
  .selectAll("dot")
  .data(data).enter()
  .append("circle")
```

```
  .attr("cx", function (d) { return d[0] } )
  .attr("cy", function (d) { return d[1] } )
  .attr("r", 3)
  .style("fill", "Red");
</script>
</body>
</html>
```