JavaScript

Introduction to JavaScript

- **JavaScript** is a high-level, interpreted programming language.
- Used to **make web pages interactive and dynamic**.
- It runs **in the browser** (client-side) but can also run on servers using **Node.js**.

Key Features:

- Lightweight and fast
- Interpreted language
- Event-driven

Purpose and Usage in Web Development

JavaScript is used for:

- Validating user input on forms
- Manipulating HTML/CSS (DOM)
- Creating interactive effects (like sliders, popups)
- Handling browser events
- Storing data locally (localStorage, sessionStorage)

Ways to Include JavaScript in HTML

1. Internal JavaScript

- JavaScript is written **inside a <script> tag** in the **same HTML file**, usually placed in the <head> or <body> section.

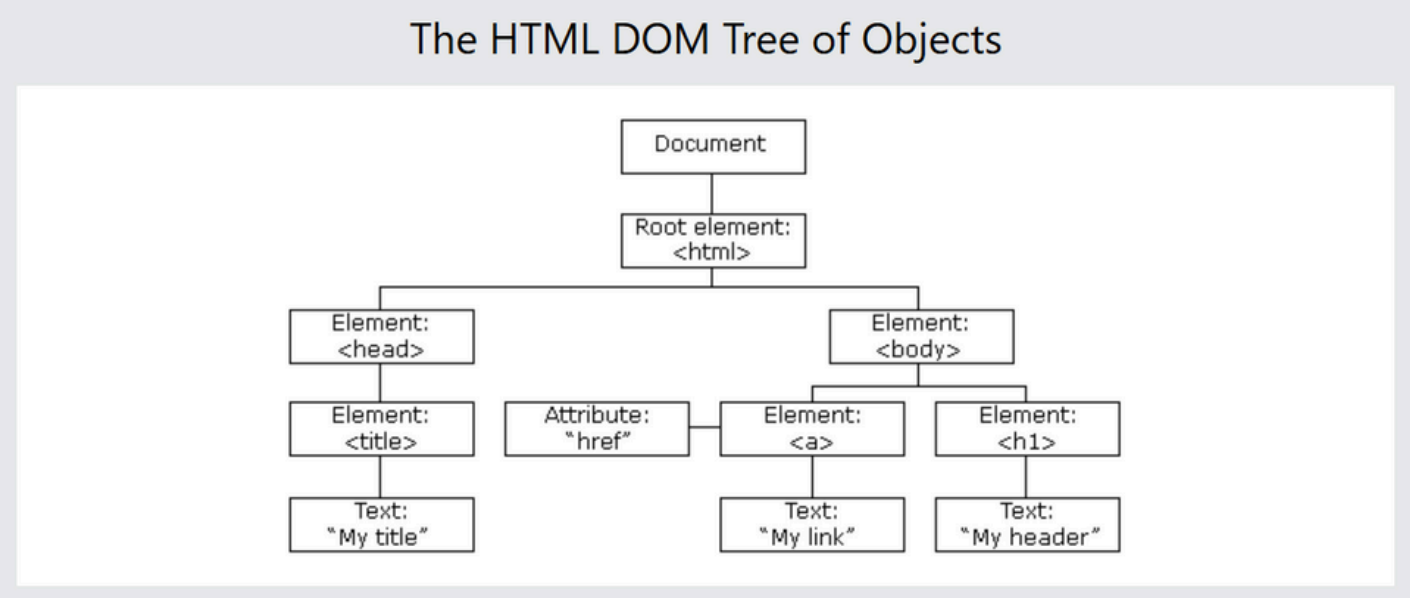
2. External JavaScript

- JavaScript code is written in a separate .js file, and linked using the <script src="filename.js"> tag.

DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

“ The HTML DOM is a standard for how to get, change, add, or delete HTML elements. ”

DOM Methods :

DOM Method	**Description**
getElementById()	Selects an element by its ID
querySelector()	Selects the first element matching a CSS selector
createElement()	Creates a new HTML element
appendChild()	Adds a child element to a parent element
removeChild()	Removes a specified child element

Output Methods in JavaScript

JavaScript provides several ways to display output or communicate with the user. Each method serves a different purpose.

1. alert()

- Displays a popup alert box with a message.

```
alert("Hello! This is an alert.");
```

2. console.log()

- Displays output in the **browser's developer console**.

```
console.log("This is a console log.");
```

3. document.write()

- Writes directly to the **HTML document**.

```
document.write("Hello from document.write!");
```

4. innerHTML

- Dynamically changes the content of a specific **HTML element**.

```
<p id="output"></p>

<script>
  document.getElementById("output").innerHTML = "Hello from innerHTML!";
</script>
```

JavaScript Variables

A **variable** is a named container used to store data that can be used and manipulated throughout a program.

JavaScript provides **three** ways to declare variables:

i. var
ii. let
iii. const

var

- Function scoped

- Can be redeclared and updated

```
function test() {
  if (true) {
    var x = 10;
  }
  console.log(x); // ✅ Output: 10
}
test();
```

```
var name = "Alice";
var name = "Bob";  // ✅ No error!
console.log(name); // Output: Bob
```

```
var age = 20;
age = 25;           // ✅ Allowed
console.log(age);   // Output: 25
```

let

- Block scoped (only accessible within {})
- Cannot be redeclared in the same scope
- **Re-assignable :** You can change the value of a let variable after it has been declared.

```
// Block Scope
{
  let a = 10;
  console.log(a); // ✅ 10
}
// console.log(a); // ❌ ReferenceError: a is not defined

// Re-assignable
let name = "Alice";
name = "Bob"; // ✅ Allowed

// Cannot Redeclare
let x = 5;
// let x = 10; // ❌ Error: Identifier 'x' has already been declared
```

const

- Used to declare constants (values that do not change).
- Block scoped – only accessible within {} where declared.
- Cannot be re-assigned or re-declared in the same scope.
- Must be initialized

```
// Block Scope
{
  const pi = 3.14;
  console.log(pi); // ✅ 3.14
}
// console.log(pi); // ❌ ReferenceError: pi is not defined

// Re-assignment not allowed
const name = "Alice";
// name = "Bob"; // ❌ Error: Assignment to constant variable

// Must be initialized
// const x; // ❌ Error: Missing initializer in const declaration
```

Data Types

A **data type** defines the **type of data** stored in a variable.

JavaScript data types are broadly classified into:

a. Primitive Data Types
b. Non-Primitive (Reference) Data Types

Primitive Data Types

These store **single values** and are **immutable**.

Data Type	Description	Example
String	Represents textual data	"Hello, world!"
Number	Represents both integer and float values	42, 3.14
Boolean	Logical value: true or false	true, false
Undefined	Variable declared but not assigned a value	let x;
Null	Represents intentional absence of value	let y = null;
BigInt	For integers larger than 2^53 - 1	12345678901234567890n

Non-Primitive (Reference) Data Types

Data Type	Description	Example
Object	Collection of key-value pairs	{ name: "Alice", age: 25 }
Array	Ordered list of values (objects)	[1, 2, 3]

Arrays and Array Methods

1. Arrays store multiple values in a single variable.
2. Values can be of any data type (numbers, strings, objects, etc.).
3. Arrays are zero-indexed (first element is at index 0).

let fruits = ["apple", "banana", "cherry"];

Accessing and Modifying Elements

```
let colors = ["red", "green", "blue"];

console.log(colors[1]);  // Output: green

colors[2] = "yellow";    // Modifying
colors[3] = "purple";    // Adding new element
```

Array Methods :

push(): Adds item to end

pop(): Removes last item

shift(): Removes first item

unshift(): Adds item to start

splice(): Add/Remove items at specific index

```
// Insert at index 1, delete 0 items
fruits.splice(1, 0, "kiwi");

// Delete 2 items from index 0
fruits.splice(0, 2);
```

sort(): Sorts array alphabetically

concat(): Merge arrays

```
let arr1 = [1, 2];
let arr2 = [3, 4];
let merged = arr1.concat(arr2);
```

slice(): Copy part of array

```
let newArr = fruits.slice(1, 3); // index 1 to 2
```

includes(): Check if value exists

```
fruits.includes("apple"); // true or false
```

indexOf(), lastIndexOf(): Get position of element

```
fruits.indexOf("banana");
fruits.lastIndexOf("banana");
```

join(): Converts array to string

```
fruits.join(", ");
```

map(): Transform each item, return new array

```
let nums = [1, 2, 3];
let squared = nums.map(n => n * n);  // [1, 4, 9]
```

forEach(): Loop over items

```
fruits.forEach(fruit => console.log(fruit));
```

filter(): Keep items that pass condition

```
let even = nums.filter(n => n % 2 === 0);  // [2]
```

reduce(): Reduce array to a single value

```
let sum = nums.reduce((acc, curr) => acc + curr, 0);
```

find(): Returns first match

```
let found = nums.find(n => n > 1); // 2
```

findIndex(): Index of first match

```
let idx = nums.findIndex(n => n > 1); // 1
```

from(): Converts iterable to array

```
Array.from("hello"); // ['h', 'e', 'l', 'l', 'o']
```

isArray(): Check if value is an array

```
Array.isArray([1, 2, 3]); // true
```

JavaScript Operators

Operators in JavaScript are special symbols used to **perform operations** on values (operands).

Category	Description
Arithmetic Operators	Perform basic mathematical operations
Assignment Operators	Assign values to variables
Comparison Operators	Compare two values
Logical Operators	Combine or invert logical values
Ternary Operator	Shortcut for if-else

1. Arithmetic Operators

Operator	Name	Example	Result
+	Addition	5 + 2	7
-	Subtraction	5 - 2	3
*	Multiplication	5 * 2	10
/	Division	5 / 2	2.5
%	Modulus	5 % 2	1
**	Exponentiation	2 ** 3	8
++	Increment	x++	Increases by 1
--	Decrement	x--	Decreases by 1

2. Assignment Operators

Operator	Example	Meaning
=	x = 5	Assigns 5 to x
+=	x += 2	x = x + 2
-=	x -= 2	x = x - 2
*=	x *= 2	x = x * 2
/=	x /= 2	x = x / 2
%=	x %= 2	x = x % 2

3. Comparison Operators

Used to compare values (returns true or false).

Operator	Name	Example	Result
==	Equal to	5 == "5"	true
===	Strict equal	5 === "5"	false
!=	Not equal	5 != "5"	false
!==	Strict not equal	5 !== "5"	true
>	Greater than	5 > 2	true
<	Less than	5 < 2	false
>=	Greater or equal	5 >= 5	true
<=	Less or equal	5 <= 2	false

4. Logical Operators

Used to combine Boolean expressions.

Operator	Name	Example	Result
&&	Logical AND	true && false	false
||	Logical OR	true && false	true
!	Logical NOT	!true	false

Eg :

```
const x=5,y=3
console.log( (x<6) && (y<5) )
```

5 . Ternary Operator :

Syntax is :

```
condition ? expression1 : expression2
```

The ternary operator evaluates the test condition.

Conditional Statements

Conditional Statements

Used to perform different actions based on different conditions.

1. if Statement
2. if...else Statement
3. else if... Statement
4. switch Statement

if Statement :

Executes a block of code if the condition is true.

```
let age = 18;
if (age >= 18) {
  console.log("You are an adult");
}
```

if...else Statement

Runs one block if the condition is true, another if false.

```
let num = 5;
if (num % 2 === 0) {
  console.log("Even");
} else {
  console.log("Odd");
}
```

else if Statement

Checks multiple conditions in sequence.

```
let marks = 85;
if (marks >= 90) {
  console.log("Grade A");
} else if (marks >= 75) {
  console.log("Grade B");
} else {
  console.log("Grade C");
}
```

switch Statement

The switch statement is used to perform different actions based on different values of a variable or expression.

```
let day = 3;
let dayName;

switch (day) {
  case 1:
    dayName = "Monday";
    break;
  case 2:
    dayName = "Tuesday";
    break;
  case 3:
    dayName = "Wednesday";
    break;
  case 4:
    dayName = "Thursday";
    break;
  case 5:
    dayName = "Friday";
    break;
  case 6:
    dayName = "Saturday";
    break;
  case 7:
    dayName = "Sunday";
    break;
  default:
    dayName = "Invalid day";
}

console.log("Today is " + dayName);
```

Loops

Loops are used to repeat a block of code multiple times.

1. for Loop
2. while Loop
3. do...while Loop
4. for...in Loop
5. for of Loop
6. forEach()

for Loop

Syntax :

```
for (initialization; condition; increment/decrement) {
  // code block to execute
}
```

Example :

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

while Loop

Syntax :

```
while (condition) {
  // code block to execute
}
```

Example :

```
let i = 1;
while (i <= 5) {
  console.log("Number:", i);
  i++;
}
```

do...while Loop

Syntax :

```
do {
  // code block to execute
} while (condition);
```

Example :

```
let i = 6;
do {
  console.log("Value:", i);
  i++;
} while (i <= 5);
```

for...in Loop

The for...in loop is used to **iterate over the enumerable properties** (keys) of an **object**.

Syntax :

```
for (let key in object) {
  // code block using object[key]
}
```

key – a variable that holds the current property name (as a string).

object – the object whose properties you want to iterate through.

Example :

```
let student = { name: "John", age: 20, course: "JS" };

for (let key in student) {
  console.log(key + ":", student[key]);
}
```

for...of Loop

The for...of loop is used to iterate over iterable objects like:

```
for (let variable of iterable) {
  // code to execute
}
```

- variable – the current element/value in the iterable.
- iterable – a collection (like array, string, etc.) you want to loop through

Example 1: Loop through array values

```
let colors = ["red", "green", "blue"];

for (let color of colors) {
  console.log(color);
}
```

Example 2: Loop through characters in a string

```
let word = "Hello";

for (let char of word) {
  console.log(char);
}
```

for...in vs for...of

Feature	for...in	for...of
Iterates over	Keys (property names)	Values of iterable items
Works with	Objects, Arrays	Arrays, Strings, Sets, Maps, etc.
Use for	Object properties	Array or string values

Summary

Statement	Use For
if	One condition
if...else	Two possible outcomes
else if	Multiple conditions
switch	Clean alternative to many if...else
for	Known number of iterations
while	Unknown number of iterations
do...while	Run first, check later
for...in	Iterate over object properties
break	Exit loop early
continue	Skip one iteration in loop

forEach() in JavaScript

The forEach() method is used to execute a function once for each element in an array.

```
array.forEach(function(element, index, array) {
  // code to execute for each element
});
```

Example

```
let fruits = ["apple", "banana", "cherry"];

fruits.forEach(function(fruit) {
  console.log(fruit);
});
```

FUNCTIONS

In JavaScript, a function is a reusable block of code designed to perform a specific task or calculate a value.

A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().

```
function functionName(parameters) {
    // code to execute
}
```

You call (run) a function using:

```
functionName(arguments);
```

What Are **Parameters**?

- **Parameters** are variables listed **inside the function definition**.
- They act like placeholders for values the function will receive.

What Are **Arguments**?

- **Arguments** are the **actual values** you pass when you call the function.

Term		Example
Parameter	In the function definition	function add(a, b)
Argument	In the function call	add(5, 3)

Example 1: A Simple Function (No Parameters)

```
function greet() {
  console.log("Hello, students!");
}
greet();
```

Example 2: Function with Parameters

```
function add(a, b) {
  console.log(a + b);
```

```
}

add(5, 3);
```

Example 3: Function with Return Value

```
function multiply(x, y) {
  return x * y;
}

let result = multiply(4, 5);

console.log(result);
```

Closures in JavaScript

Closures in JavaScript occur when a function is defined within another function, allowing the inner function to access and "close over" the variables of the outer function.

This unique behavior enables the inner function to retain access to the outer function's variables even after the outer function has completed execution.

Callback Function in JavaScript

A callback function is a **function passed as an argument to another function** , which is then invoked inside the outer function to complete some action.

```
function greet(name, callback) {
  console.log("Hello, " + name);
  callback();
}

function sayBye() {
  console.log("Goodbye!");
}

greet("Megha", sayBye);
```

Anonymous Function

An **anonymous function** is a function **without a name**. It is often used when the function is not going to be reused and is passed as an argument to another function — making it ideal for callbacks.

Syntax :

```
function() {
```

```
    // code block
}
```

Examples

```
let greet = function() {
  console.log("Hello!");
};

greet(); // Output: Hello!
```

Arrow Function in JavaScript

- An **arrow function** is a **shorter syntax** for writing functions in JavaScript, introduced in **ES6**.
- It's commonly used for **inline**, **anonymous**, and **callback functions**.

```
// Traditional Function
function add(a, b) {
  return a + b;
}

// Arrow Function
const add = (a, b) => a + b;
```

Examples

```
const greet = () => {
  console.log("Hello!");
};

greet(); // Output: Hello!
```

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

console.log(multiply(3, 4)); // Output: 12
```

Map() Method in JavaScript

- In JavaScript, the `map` method is a powerful and versatile array method used for transforming elements of an array.
- The `map` function creates a new array by applying a provided callback function to each element in the original array.

```
let numbers = [1, 2, 3, 4];

let doubled = numbers.map(function(num) {
  return num * 2;
});

console.log(doubled); // Output: [2, 4, 6, 8]
```

```
let names = ["alice", "bob", "charlie"];

let upperNames = names.map(name => name.toUpperCase());

console.log(upperNames); // Output: ["ALICE", "BOB", "CHARLIE"]
```

- map() is used to **create a new array**.
- Each element is **transformed** using the provided function.
- Original array is **not modified**.

filter() Method in JavaScript

The filter() method is used to create a new array with only the elements that pass a specific condition.

```
let numbers = [5, 12, 8, 20, 3];

let result = numbers.filter(num => num > 10);

console.log(result); // Output: [12, 20]
```

```
let nums = [1, 2, 3, 4, 5, 6];

let evens = nums.filter(n => n % 2 === 0);

console.log(evens); // Output: [2, 4, 6]
```

```
let students = [
  { name: "Megha", marks: 85 },
  { name: "Anu", marks: 78 },
  { name: "Neha", marks: 90 }
];

let toppers = students.filter(student => student.marks >= 80);

console.log(toppers);
// Output: [
//   { name: "Megha", marks: 85 },
//   { name: "Neha", marks: 90 }
// ]
```

```
let names = ["Alice", "Bob", "Anu", "Charlie"];

let aNames = names.filter(name => name.startsWith("A"));

console.log(aNames); // Output: ["Alice", "Anu"]
```

reduce() Method in JavaScript

The **reduce()** method is used to **reduce** an array to a **single value** (like sum, product, average, object, etc.) by **accumulating** results across elements.

// sum of numbers

```
let numbers = [1, 2, 3, 4, 5];

let sum = numbers.reduce(function(acc, curr) {
  return acc + curr;
}, 0); // initial value is 0

console.log(sum); // Output: 15
```

Event Handling

- Event handling in web development refers to the process of responding to user interactions or actions on a webpage.
- Events can include mouse clicks, keyboard inputs, page loading, and more.
- JavaScript is commonly used to implement event handling, allowing developers to define functions that execute in response to specific events.

Category	Examples
Mouse Events	click, dblclick, mouseover, mouseout, mousedown, mouseup
Keyboard Events	keydown, keyup, keypress
Form Events	submit, change, focus, blur, input
Window Events	load, resize, scroll, unload
Clipboard Events	copy, cut, paste

What is **addEventListener** in JavaScript?

addEventListener() is a method in JavaScript used to **attach event handlers** to HTML elements — like buttons, inputs, or even the window.

It allows you to specify:

- The **type of event** (like click, mouseover, keydown)

Syntax :

```
element.addEventListener("event", function);
```

Example :

```
<button id="myBtn">Click Me</button>
    <script>
     var btn = document.getElementById("myBtn");

     btn.addEventListener("click", function() {
      alert("Button clicked!");
     });
    </script>
```

Keyboard events in JavaScript are triggered when a user interacts with the keyboard. These events can be used to detect key presses, releases, or when a key is being held down.

Event	Description
keydown	Fired when a key is **pressed down**.
keyup	Fired when a key is **released**.
keypress ❌	(Deprecated) Fired when a key is pressed.

Form events are triggered when users interact with form elements like <input>, <textarea>, <select>, <form>, etc.

Event	Description
submit	Triggered when a form is submitted
change	Triggered when input/selection changes & loses focus
input	Triggered immediately as the user types
focus	Triggered when an element gains focus
blur	Triggered when an element loses focus
reset	Triggered when a form is reset