JAVASCRIPT

Topics

1. JavaScript
2. Application of Java
3. Variables
4. Data Types (Primitive and Non-Primitive)
5. Types of Operators
6. Arithmetic, Comparison, Assignment, Logical, String, Ternary, Typeof, Bitwise
7. Control Flow
8. Sequential flow, if..else, Nested if..else, Conditional Flow
9. Loops (for, while, do-while) Break and Continue
10. OOPs
11. Object, Class, Inheritance, Polymorphism, Encapsulation, Abstraction
12. DOM Concept
13. Form Validation
14. Cookie (Expire, Encode and Decode, Secure)
15. CSS
16. Error Handling in JavaScript
17. Ajax

JavaScript

JavaScript is a lightweight, cross-platform, single-threaded, weakly typed scripting language used for both client-side and server-side development.

JavaScript is a **multi-paradigm** language. **A multi-paradigm language** that supports **imperative**, **declarative**, and **object-oriented programming** styles.

History of JavaScript

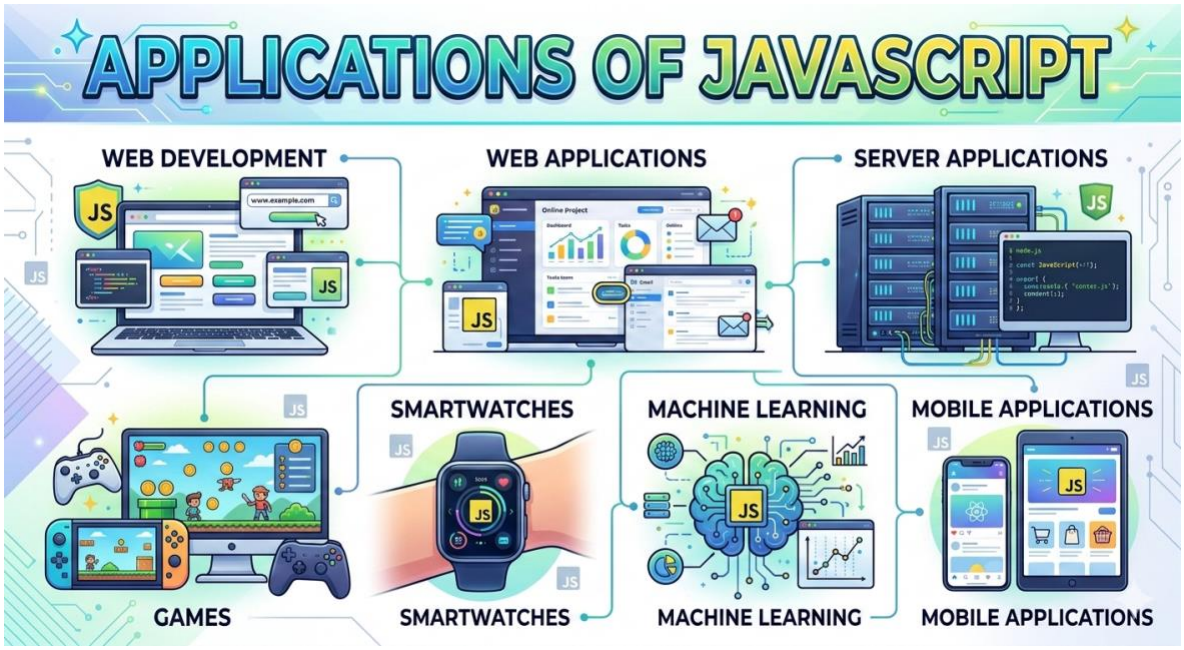
JavaScript, created in 1995 by Brendan Eich at Netscape, was originally named LiveScript and is designed to run as a scripting language within a host environment, typically a browser, without built-in input or output concepts.

Features of JavaScript

According to a recent Stack Overflow survey, JavaScript is the most popular language on Earth.

1. **DOM Manipulation**: JavaScript was initially created to manipulate the DOM (Document Object Model), turning static websites into dynamic, interactive ones.
2. **Functions as Objects**: In JavaScript, functions are also objects, meaning they can have properties and methods and can be passed as arguments to other functions.
3. **Date and Time Handling**: JavaScript can handle date and time operations using the Date object, allowing you to work with and manipulate dates and times easily.
4. **Form Validation**: JavaScript performs client-side form validation, ensuring that user input in forms created with HTML is correct before submission.
5. **No Compiler Needed**: JavaScript is an interpreted language, meaning no separate compilation is required. It runs directly in the browser or on the server (with Node.js).

Application of JavaScript



JavaScript is a compiled or interpreted language

Compiler

High-Level Language

↓

Assembly language

↓

Machine language

int x = 10; - MOV R1, 10 - 01010001 00001010

Interpreter

High-Level Language

Machine language

int x = 10; - 01010001 00001010

Variable

A **variable** is a **named container** used to **store data (values)** in a program.The stored value can be **used, changed, and reused** during program execution.

Example-

```
let price = 100;

let total = price + 20;
```

Types of Variable

4 types of variable in JS.

- a=10; // implicit variable (not recommended)
- Var b=10; // old way to declare variable
- Let c=10; //Modern way to declare variables
- Const d=10; //Used for **fixed (constant) values**

a=10;

- Implicit variable (not recommended)
- Its global variable

Example-

```
function test() {
    a = 10;   // no var / let / const
    }
  test();
console.log(a);
```

Var b=10;

- Old way to declare variables
- Function scoped
- Can be re-declared and updated
- May cause bugs, so not recommended now

Example-

```
var x = 10;
  var x = 20; // allowed
    x = 30;     // allowed
```

Var b=10;

```
var a = 10;

function test() {
  console.log(a);
}

test();       // 10
console.log(a); // 10
```

```
function test() {
  var b = 20;
  console.log(b);
}

test();       // 20
console.log(b); // ❌
Error
```

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

Let c=10;

- Modern way to declare variables
- Block scoped
- Cannot be re-declared, but can be updated
- Recommended for variables that change

Example-

```
let y = 10;
// let y = 20; ❌ not allowed
 y = 20;     // allowed
```

Let c=10;

Block scoped-
function test() {
  if (true) {
  let b = 20;
  }
  console.log(b); // ❌ Error
}
test();
	if (true) {
	  let b = 20;
	  console.log(b); // ✅ 20
	}
	
	console.log(b); // ❌ Error

Let c=10;

function test() {
  let a = 10;
  console.log(a); // ✅ 10
}
 	
test();
console.log(a); // ❌ Error (a is not defined)
	if (true) {
	  let b = 20;
	  console.log(b); // ✅ 20
	}
	
	console.log(b); // ❌ Error

Const d=10;

Used for fixed (constant) values

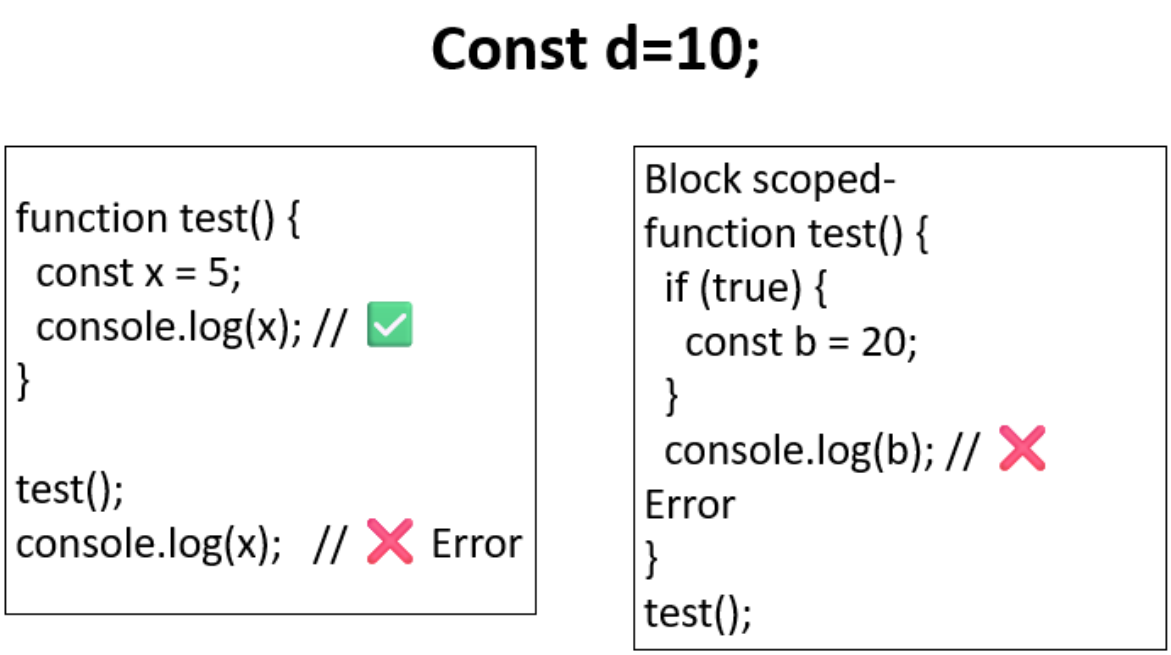
Block scoped

Cannot be re-declared or updated

Value must be assigned at declaration

Example –

```
const z = 10;
// z = 20; ❌ not allowed
```



Data types

A **data type** defines the **type of data** a variable can store.
It tells the programming language **what kind of value** (number, text, true/false, etc.) is stored in a variable and **how it should be used in memory and operations**.

Example-

```
let age = 20;       // Number
let name = "Name1"; // String
let isPass = true;  // Boolean
```

Types of Data types

- **Primitive Data Types (7)**
- **Non-Primitive / Reference Data Type (1)**

Primitive Data Types

- Number
- String
- Boolean
- Undefined
- Null
- BigInt
- Symbol

Number

- Used to store all types of numbers (integer and decimal)

  Example-

```
let a = 10;
let b = 3.14;
```

String

- Used to store text or characters

  Example-

```
let name = "N1";
let city = 'Delhi';
```

Boolean

- Stores only two values: true or false
- Example-

```
let isPass = true;
let isFail = false;
```

Undefined

- A variable is declared but no value is assigned
- Example-

```
let x;
```

Null

- Used to assign an empty or no value intentionally
- Example –

```
let y = null;
```

BigInt

- Used to store very large integer values
- Example-

```
let bigint = 123456789012345678901n;
```

Symbol

- Used to create unique values (mainly for advanced usage)
- Example-

```
let id = Symbol("id");
```

Non-Primitive (Reference) Data Type

- **Object**
- **Array** (A type of Object)
- **Function** (A type of Object)
- **typeof Operator**

Object

- Used to store multiple values in key–value pairs (Array and Function are also types of objects)

Example-

```
let student = {
        name: "CSA",
        age: 20
            };
```

Array (A type of Object)

- Used to store multiple values in a single variable
- **Example-**

```
let marks = [70, 80, 90];
```

Function (A type of Object)

- Used to perform a specific task
- **Example-**

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

Typeof Operator

- Used to check the data type of a value
- **Example-**

```
typeof 10       // number
typeof "Hi"     // string
typeof true     // boolean
typeof null     // object (JavaScript bug)
typeof undefined // undefined
```

Types of operator

- Arithmetic Operators.
- Comparison Operators
- Assignment Operators
- Logical Operators
- String Operators
- Ternary Operator
- Type Operators
- Bitwise Operators

Arithmetic Operators

Operator	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
**	Exponentiation
++	Increment
--	Decrement

Arithmetic Operators

Operators	Meaning	Example	Result
+	Addition	4+2	6
-	Subtraction	4-2	2
*	Multiplication	4*2	8
/	Division	4/2	2
%	Modulus operator to get remainder in integer division	5%2	1
+ +	Increment	A = 10; A+ +	11
− −	Decrement	A = 10; A− −	9

Comparison Operators

Operator	Name
==	Equal to
===	Strictly equal to
!=	Not equal to
!==	Strictly not equal to
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal

Relational Operators

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
! =	Not equal to	5! =2	True
===	Equal value and same type	5 === 5	True
		5 === "5"	False
! ==	Not Equal value or Not same type	5 ! == 5	False
		5 ! == "5"	True

Assignment Operators

Operator	Name
=	Assignment
+=	Add and assign
-=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Modulus and assign

Logical Operators

Operator	Name
&&	Logical AND
||	Logical OR
!	Logical NOT

String Operators

Operator	Name
+	Concatenation
+=	Concatenation and assign

Ternary Operator

Operator	Name
? :	Ternary Operator

Type Operators

Operator	Name
typeof	Type of a variable
instanceof	Instance of a class

Bitwise Operators

Operator	Name
&	AND
|	OR
~	NOT
^	XOR
<<	LEFT SHIFT
>>	RIGHT SHIFT
>>>	UNSIGNED RIGHT SHIFT

BITWISE OPERATIONS IN JavaScript ex.html

```
//Bitwise Operations
// bitwise AND
a = 5 & 6;
//        5=101,
//        6=110
// output:- 100: 4
document.write("Bitwise AND of 5 & 6 is: " + a + "<br>");

//bitwise OR
b = 7 | 3;
//        7=111,
//        3=011
// output:- 111 = 7
document.write("Bitwise OR of 7 | 3 is: " + b + "<br>");

//bitwise XOR
c = 11 ^ 3;
//       11=1011,
//       3= 0011
// output:- 1000 = 8
document.write("Bitwise XOR of 11 ^ 3 is: " + c + "<br>");



//bitwise NOT
d = ~4;
// 4=100
// output: ~100 :- 011 = 1011:= -8+3 = -5
document.write("Bitwise NOT of ~4 is: " + d + "<br>");

i = ~-9;
// -9=...11111111111111111111111111110111
// output: ~...11111111111111111111111111110111 :- 00000000000000000000000000001000 = 8
// 2^31 = -2147483648,
// 2^30 =  1073741824
// Equal=> 2147483648 - 1073741824 = 1073741824 - 8 = 1073741816
document.write("Bitwise NOT of ~-9 is: " + i + "<br>");
```

```
//left shift
e = 9 << 2;
// 9=1001,
// output: 1001 << 2 :- 100100 = 36
document.write("Left Shift of 9 << 2 is: " + e + "<br>");

//right shift
f = 22 >> 2;
// 22=10110
// output: 10110 >> 2 :- 101 = 5
document.write("Right Shift of 22 >> 2 is: " + f + "<br>");

//Unsigned zero fill right shift
g = 15 >>> 2;
// 15=1111
// output: 1111 >>> 2 :- 11 = 3
document.write("Zero Fill Right Shift of 15 >>> 2 is: " + g + "<br>");

//Unsigned zero fill right shift with negative number
h = -11 >>> 2;
// -11=11111...100100
// output: 11111...100100 >>> 2 :- 1111...1100 = -3
document.write("Zero Fill Right Shift of -11 >>> 2 is: " + h + "<br>");
```

Ex- Practical Bitwise Not(~) Operator

```
//Not bitwise
a = ~5; // 5 = 101
        //    = 010 := 2
       //     = 1010 := 8
      //    = -8+2 = -6
console.log(a); // -6

b = ~9; // 9 = 1001
        //    = 0110 := 6
       //     = 10110 := 16
      //    = -16+6 = -10
console.log(b); // -10
```

```
// Unsigned right shift
c = 5 >>> 1; // 5 = 101
             //   = 10 := 2
console.log(c); // output = 2

// Unsigned Left shift
d = 5 << 1; // 5 = 101
            //   = 1010 := 10
console.log(d); // output = 10
```

BITWISE OPERATIONS IN JavaScript ex.js

```
// Bitwise Operators in JavaScript

// Bitwise AND (&)
let a = 5 & 6;
     // 5: 101,
   // Result: 100 (4 in decimal)

console.log(a); // Output: 4

// Bitwise OR (|)
let b = 7 | 3;
     // 7: 111,
     // 3: 011
// Result: 111 (7 in decimal)
console.log(b); // Output: 7

// Bitwise XOR (^)
let c = 11 ^ 3; // 11: 1011, 3: 0011
// Result: 1000 (8 in decimal)
console.log(c); // Output: 8

// Bitwise NOT (~)
let d = ~4; // 4: 0100
// Result: 1011 (-8+3 = -5 in decimal)
console.log(d); // Output: -5

// Left Shift (<<)
let e = 9 << 2;
// 9: 1001
// => 1001 << 2 : 100100 (36 in decimal)
// Result: 100100 (36 in decimal)
console.log(e); // Output: 36
```

```
// Right Shift (>>)
let f = 22 >> 2; // 22: 10110
// Result: 101 (5 in decimal)
console.log(f); // Output: 5

// Unsigned Right Shift (>>>)
let g = 15 >>> 2; // 15: 1111
// Result: 11 (3 in decimal)
console.log(g); // Output: 3


// Unsigned Right Shift with Negative Number (>>>)
let h = -11 >>> 2; // -11: 11111...100100
// Result: 1111...1100 (large positive number in decimal)
console.log(h); // Output: 1073741821
```

CONTROL FLOW

control flow refers to the order in which statements are executed in a program. It includes constructs like conditionals, loops, and functions to determine the path of execution based on certain conditions or repetitive tasks.

Sequential flow

The default flow where statements execute one after the other in the order they are written.

Example:

```
let a=5;
let b=6;
let c=a+b;
console.log(c);
// output: 11
```

if....else

Executes different blocks of code based on a condition.

Example:

```
let a = 5;
if (a > 0) {
  console.log("The number is positive.");
} else {
  console.log("The number is non-positive.");
}
// Output: The number is positive.
```

if....else if

The else if statement allows you to check multiple conditions in sequence.

Example:

```
score = 95; // Example score
grade = "";

if (score >= 90) {
    grade = "A";
} else if (score >= 80) {
    grade = "B";
} else if (score >= 70) {
    grade = "C";
} else {
    grade = "F";
}
console.log("Your grade is: " + grade);
// output: Your grade is: A
```

Nested if...else

Nested if-else means having one if-else statement inside another if-else statement.

Example:

```
let age = 25;
let hasLicense = true;
if (age >= 18) {
    console.log("Adult hai");
    if (hasLicense) {          // ! NESTED IF
        console.log("License hai - drive kar sakte ho");
    } else {
        console.log("Adult hai par license nahi hai");
    }
    } else {
    console.log("Minor hai");
    }    // Output: - "Adult hai"
        // "License hai - drive kar sakte ho"
```

Selects one of many blocks of code based on a variable's value.

Example:

```
let day = 2;
switch (day) {
  case 1:
    console.log("Monday");
    break;
  case 2:
    console.log("Tuesday");
    break;
  default:
    console.log("Invalid day");
   }
  // Output: Tuesday
```

Conditional Flow

```
let a = 10;
let b = 20;
console.log(a > b ? "a is greater than b" : a < b ? "b is greater than a" : "a and b are equal");
// Output: b is greater than a
```

Loops-for

Executes a block of code a specified number of times.

Example:

```
for (let i = 1; i <= 4; i++) {
console.log("Iteration:", i);
}
// Output: Iteration: 1
// Output: Iteration: 2
// Output: Iteration: 3
// Output: Iteration: 4
```

Loops – while

Repeats a block of code as long as the condition is true.

Example:

```
let count = 1;
while (count <= 3) {
console.log("Count:", count);
count++;
}
// Output:
// Count: 1
// Count: 2
// Count: 3
```

Loops- do-while

Executes a block of code at least once, then continues based on the condition.

Example:

```
let n = 0;
do {
console.log("Number:", n);
n++;
} while (n < 3);
console.log("Loop finished.");
// Output:
// Number: 0
// Number: 1
// Number: 2
// Loop finished.
```

Break and continue

Break: Exits the loop immediately.

Continue: Skips the current iteration and moves to the next one.

Example:

Break	Continue

for (let i = 1; i <= 10; i++)
{
if (i === 5) {
break;
}
console.log(i);
}
	for (let i = 1; i <= 10; i++)
	{
	if (i === 5) {
	continue;
	}
	console.log(i);
	}

Break Ex-

```
for (let i = 1; i <= 10; i++) {
  if (i === 5) {
    break;
  }
  console.log(i);
}
// Output: 1, 2, 3, 4
```

Continue Ex-

```
for (let i = 1; i <= 10; i++) {
  if (i === 5) {
    continue;
}
  console.log(i);
}
// Output: 1, 2, 3, 4, 6, 7, 8, 9, 10
```

Functions

Encapsulates reusable blocks of code.

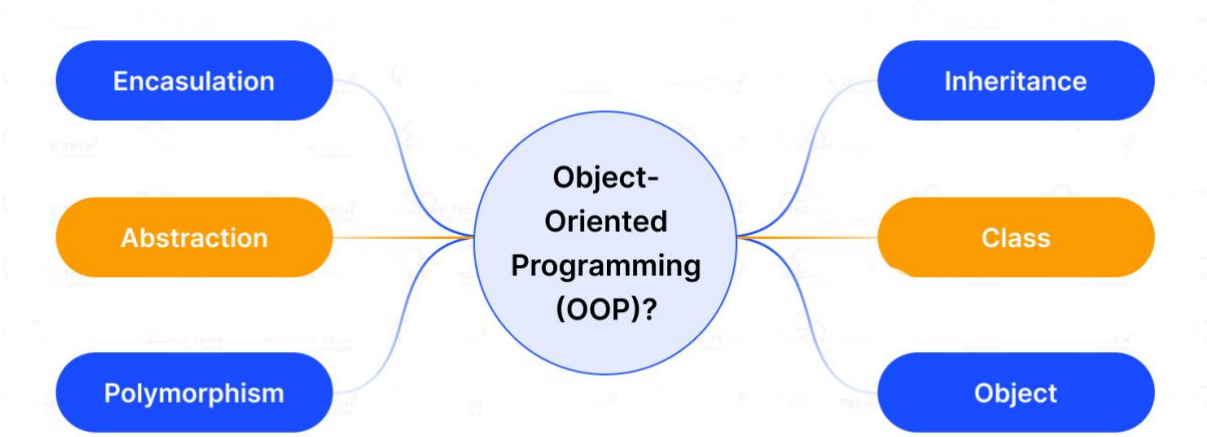
```
function greet(name) {
    if (name) {
        console.log('Hello, ${name}');
    } else {
        console.log("Hello, stranger");
    }}
  greet("CSA"); // output: Hello, CSA
  greet();      // output: Hello, stranger
```

OOPs

Object-Oriented Programming System

In JavaScript, Object-Oriented Programming (OOP) is a programming pattern that organizes data and its associated actions into objects. This approach makes the code more modular, reusable, and maintainable.

fundamental features of OOPs



Object

In JavaScript, an object is a collection of key-value pairs (properties). Objects are used to store multiple values as a single entity. Keys are strings (or Symbols), and values can be any data type.

```
const person = {
name: "CSA",
age: 30,
address: "howrah",
mobile: 9999999999
};
console.log(person.name); // Output: CSA
console.log(person.age);  // Output: 30
console.log(person.address); // Output: howrah
console.log(person.mobile); // Output: 9999999999
```

Class

A class is a template for creating objects. It encapsulates data and functions that operate on that data. Classes were introduced in ES6 (2015) and provide a cleaner, more structured way to create objects compared to constructor functions.

```
class Person {
constructor(name, age) {
this.name = name;
this.age = age;
}};
console.log(new Person("CSA", 30));
// Output: Person { name: 'CSA', age: 30 }
```

Class and object

```
class Tranee {
    constructor(n1, m1) {
        this.name = n1;
        this.mobile = m1;
    }};
// Creating an instance
const Tranee1 = new Tranee("Jit",9999900000);
console.log(Tranee1.mobile); //Output: 9999900000
console.log(Tranee1.name);  //Output: Jit
console.log(Tranee1); //Output: Tranee { name: 'Jit', mobile: 9999900000 }
```

Inheritance

Inheritance is a mechanism that allows one class to inherit properties and methods from another class. It enables code reusability and establishes a parent-child relationship between classes.

```
class Animal {
    constructor(name) {
    this.name = name;
    };};
class Dog extends Animal {
   constructor(name, breed) {
    super(name);
    this.breed = breed;
};};
const myDog = new Dog("Buddy", "Golden Retriever");
console.log(myDog.name);    // Output: Buddy
console.log(myDog.breed); // Output: Golden Retriever
```

Polymorphism

Polymorphism (Greek: "many forms") is the ability of different objects to respond to the same method or property in different ways.

➢ Polymorphism

```
class animal {
sound (){
   console.log("animal sounds");
}
}
class dog {
sound (){
   console.log("dog bark");
}
}
```

```
class cat {
sound (){
   console.log("meow meow");
}
}
let Animal = new animal();
let Dog = new dog();
let Cat = new cat();
Animal.sound();
Dog.sound();
Cat.sound();
```

Encapsulation

Encapsulation is the bundling of data (properties) and methods (functions) that operate on that data within a single unit (object/class), while restricting direct access to some of the object's components.

Its not Encapsulated

```
class studentDetails {
    constructor(name,trade,age){
        this.name=name;
        this.trade=trade;
        this.age=age;
    }
}
let student1= new studentDetails("S1","csa",21);
console.log(student1);
// Output: studentDetails { name: "S1", trade: "csa", age: 21 }
let student2= new studentDetails("S2","csa",22);
console.log(student2);
// Output: studentDetails { name: "S2", trade: "csa", age: 22 }
```

Ex- 2

```
class employeeDetails {
    #name;  //private variables declaration
    #address;
constructor(name,addresss,age){
        this.#name=name;
        this.#address=addresss;
        this.age=age
    }
    getEmployeeDetails (){
        return this.name=this.#name;
    };};
let employee1= new employeeDetails("N1","howrah",21);
console.log(employee1.getEmployeeDetails());
//Output: N1
```

Abstraction

Abstraction means hiding complex implementation details and showing only essential features to the user. It's like a TV remote - you just press buttons, you don't need to know how signals are transmitted.

```
class Vehicle {
    constructor(name) {
      if (new.target === Vehicle) {
        throw new Error("Cannot instantiate an abstract class.");
      }
      this.name = name;
    }
    startEngine() {
          throw new Error("Method 'startEngine()' must be   implemented.");
    }
  }
console.log("Vehicle class created successfully.");
```

Ex- 2

```
class Car extends Vehicle {
   startEngine() {
     console.log(`${this.name}'s engine is starting... Vroom Vroom!`);
   }}
 // Subclass
 class Bike extends Vehicle {
   startEngine() {
     console.log(`${this.name}'s engine is starting... Zoom Zoom!`);
   }}
```

Ex- 3

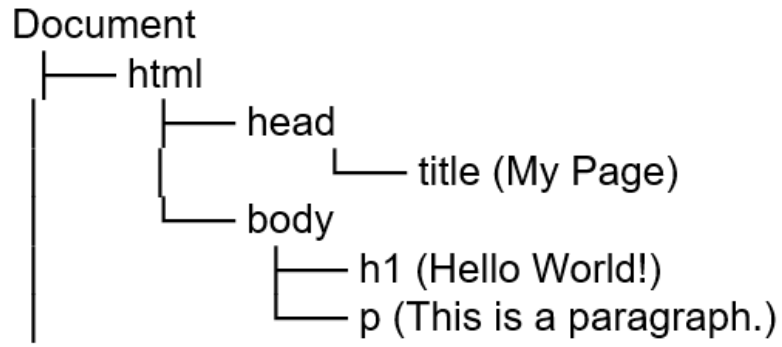
```
const car = new Car("Toyota");
 car.startEngine(); // Output: Toyota's engine is starting... Vroom Vroom!


 const bike = new Bike("Yamaha");
 bike.startEngine(); // Output: Yamaha's engine is starting... Zoom Zoom!
```

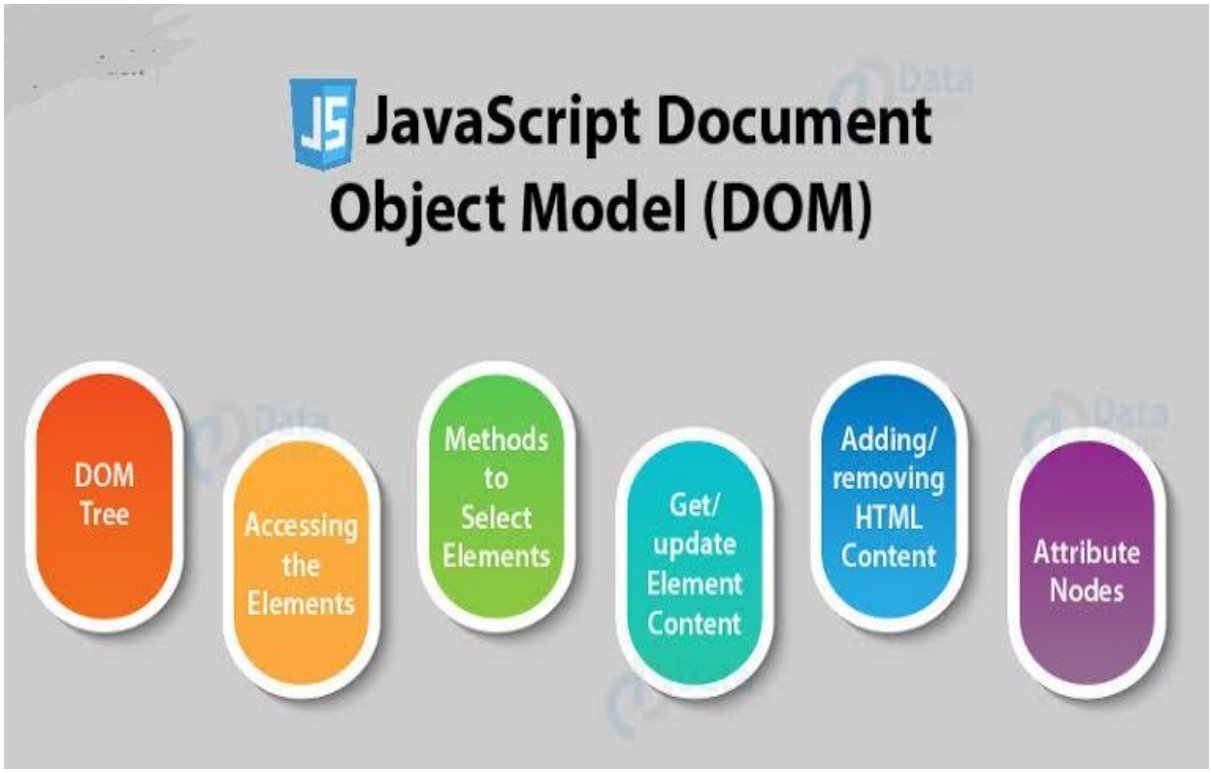
```
class Vehicle {
    constructor(name) {
      if (new.target === Vehicle) {
        throw new Error("Cannot instantiate an abstract class.");
      }
      this.name = name;
    }
    startEngine() {
            throw new Error("Method 'startEngine()' must be   implemented.");
    }}

class Car extends Vehicle {
    startEngine() {
      console.log(`${this.name}'s engine is starting... Vroom Vroom!`);
    }}
  // Subclass
  class Bike extends Vehicle {
    startEngine() {
      console.log(`${this.name}'s engine is starting... Zoom Zoom!`);
    }}
    // Create instances
const myCar = new Car("Toyota");
const myBike = new Bike("Harley");
// Start engines
myCar.startEngine();
myBike.startEngine();
```

DOM CONCEPT

The DOM (DOCUMENT OBJECT MODEL) is a programming interface that represents a web page in a structured form.

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Page</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```





DOM CONCEPT IN JS

```
//Selecting by  ID
```

- let heading = document.getElementById("myHeading");

```
//  Selecting by Class
```

- let paragraphs = document.getElementsByClassName("myPara");

```
// selecting by TAG
```

- let divs = document.getElementsByTagName("div");

```
// Selecting by Query Selector
```

- let firstPara=document.querySelector("p");// first para is selected <p>
- let allParas = document.querySelectorAll("p"); // All para is selected <p>

```
 //change inner text and inner html
 document.getElementById("myHeading").innerText = "new text";
document.getElementById("myHeading").innerHTML = "<b>Bold text</b>";
//add new element
let newPara = document.createElement("p");  // <p> create a tag
newPara.innerText = "This is new paragra";  // add text
document.body.appendChild(newPara); // add to body
//Remove element
let elementToRemove = document.getElementById("myPara");
elementToRemove.remove();
```

Form Validation

Form Validation is the process of **checking user input in a form before submitting it to the server**. It ensures that the user enters **correct and complete information**.

Why Form Validation is Important

- Prevents **empty fields**
- Ensures **correct data format**
- Improves **data accuracy**
- Enhances **website security**

Types of Form Validation

1. **Client-side Validation** – Done in the browser using JavaScript.
2. **Server-side Validation** – Done on the server using languages like PHP, Python, etc.

Form Validation

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Form Validation</h1>
    <h2>Form</h2>

    <form onsubmit="return ValidateForm()">

        Name: <br>
        <input type="text" id="name"><br><br>

        Email: <br>
        <input type="email" id="email"><br><br>

        Phone: <br>
        <input type="text" id="mobile"><br><br>

        <input type="submit" value="submit">
    </form>
    <script>
        function ValidateForm(){
            var name = document.getElementById("name").value;
            var email = document.getElementById("email").value;

            // Name Validation
            if(name ==""){
                alert("Please enter your name");
                return false;
            }

            //Email validation
            if(email="" || !email.includes("@") || !email.includes(".")){
                alert("Please enter your email");
                return false;
            }

            //Mobile Validation
            if(mobile.length != 10 || isNaN(mobile)){
                alert("Please enter valid 10-digit mobile number")
                return false;
            }
            alert("Form submitted Successfully!")
            return true;
        }
    </script>
</body>
</html>
```

COOKIES

Cookies are small data files that websites store in your browser. They are used to track website information, store user settings, or maintain login status.

Ex-1

Cookie

```
<script>
    document.cookie = "name=CSA";
    let a = document.cookie;
    document.write(a);
</script>
```

Cookie Expire

```
<script>
    document.cookie="name=Day"
    document.cookie="name1=Special Day; expires=sat, 14 Feb 2026 23:59:59 UTC";

    document.write(document.cookie);
</script>
```

Cookie Encode and Decode

```
<script>
    document.cookie="name=jit"
    document.cookie="name1=jit1";

    document.cookie=encodeURIComponent ("name2=jit2;path")
    document.write(document.cookie)

    document.cookie=encodeURIComponent("Name=jit;path;")
    document.write(decodeURIComponent(document.cookie))
</script>
```

Cookie Secure

```
<script>
document.cookie="name=csa"
document.cookie="name1=cs"
document.cookie="name1=csa;path=/;secure;"

document.write(document.cookie);
</script>
```

CSS

CSS (Cascading Style Sheets) is a language used to **style and design web pages**. It is used with **HTML** to control how elements look on a website.

CSS is used to **change the appearance of a webpage**, such as colors, fonts, spacing, layout, and animations.

- **HTML** creates the **structure** of a webpage (like headings, paragraphs, images).
- **CSS** makes the webpage **look attractive** by adding style.

Class in CSS

A **class** is used to apply the **same style to multiple HTML elements**.

- It is written using a **dot (.)** in CSS.
- In HTML, it is written using the **class attribute**.
- Many elements can use the **same class name**.

Class Selector

HTML

```
<p class="text">This is text</p>
```

CSS

```
.text{
  color: green;
}
```

ID in CSS

An **ID** is used to apply style to **one unique element**.

- It is written using a **hash (#)** in CSS.
- In HTML, it is written using the **id attribute**.
- One ID should be used **only once in a page**.

ID Selector

```
HTML

<h1 id="title">Hello</h1>
```

```
CSS

#title{
   color: red;
}
```

Types of CSS

1. Inline CSS
2. Internal
3. External

Inline CSS

Inline CSS is written **inside the HTML tag** using the style attribute.
It affects **only that specific element**.

Ex-

```
HTML

<p style="color:red; font-size:20px;">This is a paragraph.</p>
```

Internal CSS

Internal CSS is written **inside the <style> tag** in the <head> section of the HTML page.
It applies **style to the whole page**.

```
</> HTML

<!DOCTYPE html>
<html>
<head>
<style>
p{
  color: blue;
  font-size: 18px;
}
</style>
</head>
<body>

<p>This is a paragraph.</p>

</body>
</html>
```

External CSS

External CSS is written in a **separate file with .css extension** and linked to HTML.

style.css

```css
p{
  color: green;
  font-size: 20px;
}
```

HTML File

```html
<link rel="stylesheet" href="style.css">
```

This method is **most commonly used in websites.**

Error Handling in JavaScript

Error handling in JavaScript is a technique used to **detect and manage runtime errors using try, catch, finally, and throw statements**, so the program can continue running properly.

When an error occurs, JavaScript provides special statements to catch and handle the error.

- **try** → contains code that may produce an error
- **catch** → runs if an error occurs
- **error.message** → shows the error message

Ex - 1

JavaScript

```
try {
  let x = y + 10;   // y is not defined
}
catch(error) {
  console.log("An error occurred:", error.message);
}
```

JavaScript

```
try {
  let num = 10 / 2;
  console.log(num);
}
catch(error) {
  console.log("Error occurred");
}
finally {
  console.log("Program finished");
}
```

Ex- throw Statement

The throw statement is used to create custom errors.

```javascript
let age = 15;

try {
  if(age < 18){
    throw "You are not allowed";
  }
}
catch(error){
  console.log(error);
}
```

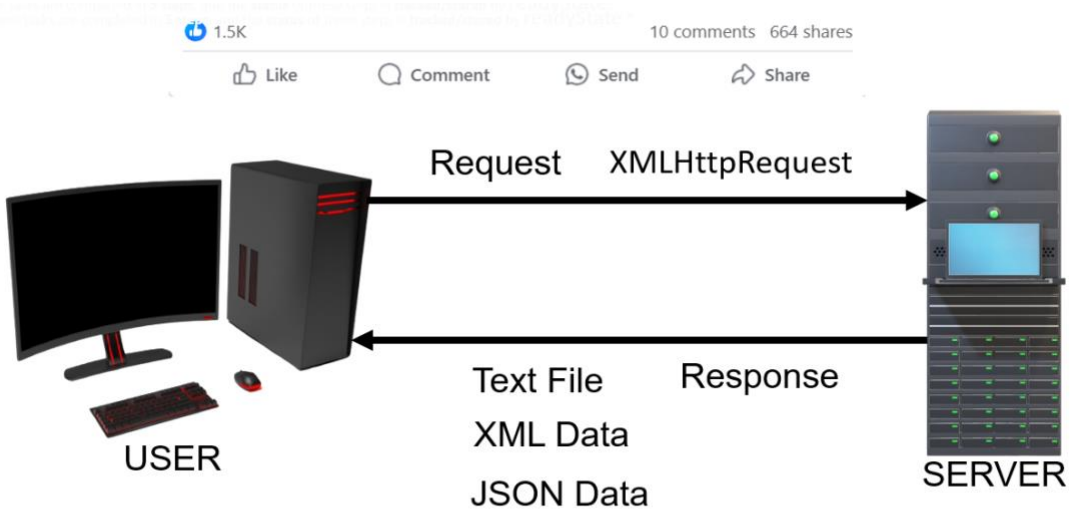
Types of JavaScript Errors

Some common errors in JavaScript:

1. **Syntax Error** – Incorrect code syntax
2. **Reference Error** – Using an undefined variable
3. **Type Error** – Using wrong data type
4. **Range Error** – Value out of range

AJAX

AJAX is a technique for creating fast and dynamic web pages.



All these tasks are completed in 5 steps, and the status of these 5 steps is stored by readyState."

ReadyState 5 State Are-

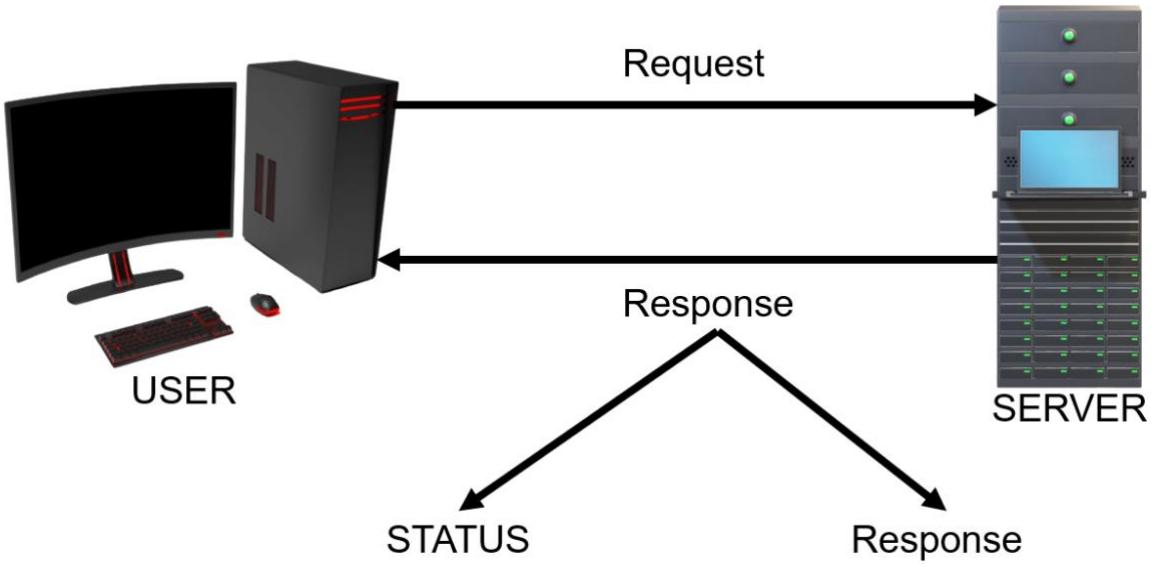
0 (UNSENT):-your XMLHttpRequest is created but open method is not called.

1 (OPENED):-open method is called.

2 (HEADERS RECEIVED):- send method is called and headers id received.

3 (LOADING):- your responseText is loading.

4 (DONE):- request is completed and your response is received.



STATUS		Response
200:-"OK"	your response is successfully received.	TEXT XML
403:-"FORBIDDEN"	Your server is not respond.	JSON HTML
404:-"NOT FOUND"	Your file is not found in server.	BLOB etc.

Ajax Syntax

GET Method

```
let A=new XMLHttpRequest();
A.open("GET","filename.txt",true);

A.onreadystatechange=function(){
if(this.readyState ==4 && this.status==200){

document.getElementById("demo").innerHTML=this.responseText;
}
}
A.send();
```