Unit-2

- **CLIENT-SIDE PROGRAMMING**
- Introduction to JavaScript – Functions – Objects – Arrays – Built - in Objects - JavaScript Debuggers. Browsers and the DOM - Introduction to the Document Object Model DOM HistoryandLevels-IntrinsicEventHandlingModifyingElementStyle-TheDocumentTree-DOMEventHandling.

- What is JavaScript?
- JavaScript is a scripting language used to make web pages interactive.
- Runs inside the browser (client-side).
- It works together with:
- HTML → Structure (content)
- CSS → Style (design)
- JavaScript → Behavior (actions, interactivity)
- Example: HTML builds a button, CSS styles it, and JavaScript makes it respond when clicked.

- A **scripting language** is a type of programming language designed to automate tasks within a specific software environment or a system. While all scripting languages are programming languages, not all programming languages are scripting languages.
- The primary difference lies in how the code is executed. Scripting languages are typically **interpreted**, meaning the code is read and executed line by line by an interpreter program at runtime. This contrasts with **compiled languages**, which must be translated into machine code by a compiler before they can be run.

- Faster Development Cycle: Because there's no compilation step, developers can make changes and see the results instantly, making it ideal for rapid prototyping and testing.

- Easier to Learn: Scripting languages often have a simpler syntax and fewer rules, making them more accessible to beginners.

- Less Code Intensive: They are designed to automate tasks with fewer lines of code.

- Scripting languages are used in a variety of contexts, including:

- Web Development:

- Client-side scripting (run in the user's browser): JavaScript is the most prominent example, used to create interactive and dynamic web pages.

- Server-side scripting (run on a web server): Examples include PHP and Node.js, which handle tasks like processing user input and interacting with databases.

- System Administration: Shell scripting languages like Bash and PowerShell are used to automate repetitive tasks on operating systems, such as file management and system monitoring.

- Game Development: Many games use scripting languages, like Lua, to handle in-game logic and user-generated content.

- Data Science: Python is a widely used scripting language for data analysis and machine learning due to its simplicity and extensive libraries.

- Features of JavaScript

- Lightweight, interpreted language.

- Case-sensitive (e.g., Name ≠ name).

- Object-oriented features (objects, arrays, functions).

- Event-driven (responds to clicks, key presses, etc.).

- Supported by all modern browsers (Chrome, Edge, Firefox, Safari).

- Adding JavaScript to a Web Page
- (a) Inline JavaScript
- (b) Internal JavaScript
- (c) External JavaScript

(a) Inline JavaScript

- <button onclick="alert('Hello!')">Click Me</button>

(b) Internal JavaScript

```
<!DOCTYPE html>
<html>
<head>
  <title>My Page</title>
  <script>
    function sayHello() {
      alert("Welcome to JavaScript!");
    }
  </script>
</head>
<body>
  <button onclick="sayHello()">Click</button>
</body>
</html>
```

(c) External JavaScript

- <!-- index.html -->
- <script src="script.js"></script>

- // script.js
- alert("Hello from external file!");

- 1. **HTML file** (index.html)
- <!DOCTYPE html>
- <html>
- <head>
- <title>External JavaScript Example</title>
- </head>
- <body>
- <h1 id="heading">Hello, Welcome!</h1>
- <button onclick="changeText()">Click Me</button>

- <!-- Linking external JS file -->
- <script src="script.js"></script>
- </body>
- </html>

- 2. **JavaScript file** (script.js)

- // Function to change heading text
- function changeText() {
- document.getElementById("heading").innerHTML = "You clicked the button!";
- }

- **How it works:**
- The HTML file has a button.
- When the button is clicked, it calls the changeText() function.
- The function is written in **external file (script.js)**.
- It changes the <h1> text dynamically.

- Where to place <script> in an HTML file.
- You can place JavaScript in three main locations inside an HTML document:
- 1. Inside the <head> section
- Code runs before the page content is loaded.
- Useful for defining functions that will be used later.

- 2. Inside the <body> section

- Code runs as the browser reads the page.
- Useful if you want scripts to execute immediately with the content.

```
<!DOCTYPE html>
<html>
<body>
  <h1>Body Example</h1>
  <script>
    document.write("This runs while page loads!");
  </script>
</body>
</html>
```

- 3. At the end of <body> (Most Recommended)

- Code runs after the entire page is loaded.
- Prevents errors when accessing HTML elements (because they’re already loaded).

- <!DOCTYPE html>
- <html>
- <body>
- <h1>End of Body Example</h1>
-

- <script>
- //document.getElementById("demo").innerHTML = "Script ran after page loaded!";
- Find the element with id="demo" and change its content to “Script ran after page loaded!”.
- </script>
- </body>
- </html>

- Use <head> if scripts define functions only.
- Use end of <body> if scripts directly work with page content.
- Use external files for big projects.

- **Variables**
- Variables are used to store data.
- var name = "Alice"; // old way
- let age = 20; // modern, preferred
- const pi = 3.1416; // constant, cannot change

- Scope
- var → Function-scoped
- let and const → Block-scoped (inside { })

- Re-declaration
- var → Can be re-declared in the same scope
- let and const → Cannot be re-declared in the same scope.

- let → Use when the value will change.
- const → Use when the value should not change

Data Types

- JavaScript has dynamic typing (no need to declare type).
- String → "Hello", 'World', \Template``
- Number → 10, 3.14, -5
- Boolean → true, false
- Null → empty value (null)
- Undefined → variable declared but no value
- Object → {name: "John", age: 25}
- Array → [10, 20, 30]

- let student = “John"; // string
- let marks = 85; // number
- let isPassed = true; // boolean
- let fruits = ["Apple", "Banana", "Mango"]; // array
- let person = {name:"Ravi", age:21}; // object

Operators

- Arithmetic Operators
- let x = 10, y = 3;
- console.log(x + y); // 13
- console.log(x - y); // 7
- console.log(x * y); // 30
- console.log(x / y); // 3.33
- console.log(x % y); // 1 (remainder)

- Comparison Operators
- x == y // equal (checks value only)
- x === y // strict equal (checks value & type)
- x != y // not equal
- x > y, x < y, x >= y, x <= y

- Logical Operators
- true && false // AND → false
- true || false // OR → true
- !true // NOT → false

- Strings
- let name = "Alice";
- console.log(name.length); // 5
- console.log(name.toUpperCase()); // "ALICE"
- console.log("Hi " + name); // concatenation
- console.log(`Hello, ${name}`); // template literal

```
let age = 18;
if (age >= 18) {
  console.log("You are an adult");
} else if (age > 12) {
  console.log("Teenager");
} else {
  console.log("Child");
}
```

- USER INPUT through keyboard.
- Using `prompt()`
- let name = prompt("Enter your name:"); // user types via keyboard
- document.write("Hello, " + name + "!");

```
<!DOCTYPE html>
<html>
<head>
  <title>Add Two Numbers</title>
</head>
<body>
  <h2>Add Two Numbers</h2>
  <script>
    // Take input from user
    let num1 = prompt("Enter the first number:");
    let num2 = prompt("Enter the second number:");

    // Convert string input to numbers
    num1 = Number(num1);
    num2 = Number(num2);

    // Add the numbers
    let sum = num1 + num2;

    // Display the result
    document.write("The sum of " + num1 + " and " + num2 + " is: " + sum);
  </script>
</body>
</html>
```

- Write a program to check whether a number entered by the user is even or odd.

- let num = 7;
- if (num % 2 === 0) {
- console.log("Even");
- } else {
- console.log("Odd");
- }

- Write a program to check whether a number is positive, negative, or zero.

- let num = -5;

- if (num > 0) {
- console.log("Positive");
- } else if (num < 0) {
- console.log("Negative");
- } else {
- console.log("Zero");
- }

```
<!DOCTYPE html>
<html>
<head>
  <title>Check Positive, Negative, or Zero</title>
</head>
<body>
  <h2>Number Check</h2>

  <script>
    // Ask user for input
    let num = parseFloat(prompt("Enter a number:"));

    // Check conditions
    if (num > 0) {
      alert(num + " is Positive");
    } else if (num < 0) {
      alert(num + " is Negative");
    } else {
      alert("The number is Zero");
    }
  </script>
</body>
</html>
```

- prompt() → asks the user to enter a number.
- parseFloat() → converts the input (string) into a number.
- if…else → checks whether the number is positive, negative, or zero.
- alert() → displays the result.

JavaScript provides several ways to convert user input (string) into a number.

- **1. parseInt()**

- Converts a string to an integer.
- Ignores decimal part.

- let num = parseInt("42.8");
- console.log(num); // 42

- **2. parseFloat()**

- Converts a string to a floating-point number (decimal allowed).

- let num = parseFloat("42.8");
- console.log(num); // 42.8

- 3. Number()

- Converts the entire string to a number.
- Works for both integer and float.

- let num1 = Number("42");
- let num2 = Number("42.8");
- console.log(num1); // 42
- console.log(num2); // 42.8

- 4. Unary Plus Operator (+)

- A quick way to convert string to number.

- let num = +"42.8";
- console.log(num); // 42.8

- 5. Math.floor() / Math.ceil() / Math.round()

- These don’t convert directly, but after Number() or parseFloat(), they adjust values.

- let num = Math.floor("42.9"); // Converts and rounds down
- console.log(num); // 42

- Best practice for user input via prompt():
- **Use Number() → if you want both integers & floats.**
- **Use parseInt() → if only integers are allowed.**
- **Use parseFloat() → if decimals are allowed.**

- 1.Write a program to find the largest of three numbers.
- 2.Write a program that assigns grades based on marks.

- Marks ≥ 90 → Grade A

- Marks ≥ 75 → Grade B

- Marks ≥ 50 → Grade C

- Else → Fail
- 3.Write a program to check whether a given year is a leap year.
- 4.Write a program to check if a person is eligible to vote (age ≥ 18).

Write a program to find the largest of three numbers.(USER INPUT)

```
<html>
<head>
  <title>Largest of Three Numbers</title>
</head>
<body>
  <h2>Find the Largest Number</h2>

  <script>
    // Take user inputs
    let num1 = Number(prompt("Enter first number:"));
    let num2 = Number(prompt("Enter second number:"));
    let num3 = Number(prompt("Enter third number:"));

    let largest;

    // Compare numbers
    if (num1 >= num2 && num1 >= num3) {
      largest = num1;
    } else if (num2 >= num1 && num2 >= num3) {
      largest = num2;
    } else {
      largest = num3;
    }

    // Display result
    alert("The largest number is: " + largest);
  </script>
</body>
</html>
```

- Loops
- For Loop
- for (let i = 1; i <= 5; i++) {
- console.log(i);
- }

- Write a program to print the multiplication table of a given number using a for loop.

- let num = 5;
- for (let i = 1; i <= 10; i++) {
- console.log(num + " x " + i + " = " + (num * i));
- }

- While Loop
- let i = 1;
- while (i <= 5) {
- console.log(i);
- i++;
- }

- Write a JavaScript program using a while loop to print all even numbers between 1 and 20.
- let i = 1;

- while (i <= 20) {
- if (i % 2 === 0) {
- console.log(i);
- }
- i++;
- }

Functions

- Reusable blocks of code.

- function greet(name) {
- return "Hello, " + name;
- }
- console.log(greet("Ravi")); // Hello, Ravi

- Function with Parameters
- function add(a, b) {
- return a + b;
- }

- console.log(add(5, 3)); // 8
- console.log(add(10, 20)); // 30

- Function Expression

- You can also store a function in a variable.

- let multiply = function(x, y) {
- return x * y;
- };

- console.log(multiply(4, 5)); // 20

- **Arrow Function**: A shorter way to write functions.
- const greet = (name) => `Hello, ${name}`;
- console.log(greet("Ravi"));

- let square = (n) => n * n;
- console.log(square(6)); // 36

- Functions Returning Values
- function isEven(num) {
- if (num % 2 === 0) {
- return true;
- } else {
- return false;
- }
- }

- console.log(isEven(10)); // true
- console.log(isEven(7)); // false

Function Example (Addition of Two Numbers)

```
<script>
    // Define a function
    function addNumbers(a, b) {
      return a + b;  // Returns the sum
    }

    // Call the function with user input
    let num1 = Number(prompt("Enter first number:"));
    let num2 = Number(prompt("Enter second number:"));

    let result = addNumbers(num1, num2);

    alert("The sum is: " + result);
  </script>
```

Function with User Input

```
<script>
    // Function to find square
    function square(num) {
      return num * num;
    }

    // Get user input
    let n = Number(prompt("Enter a number:"));

    // Call function
    let result = square(n);

    // Show result
    alert("The square of " + n + " is " + result);
  </script>
```

Arrays

- An array is a special variable used to store multiple values in a single variable.Each value is stored at an index (starting from 0)
- let fruits = ["Apple", "Banana", "Mango"];
- console.log(fruits[0]); // Apple
- fruits.push("Orange"); // add
- fruits.pop(); // remove last
- console.log(fruits.length); // count

- 1.Creating Arrays
- let numbers = [10, 20, 30, 40];
- let colors = new Array("Red", "Green", "Blue");

- 2. Accessing Array Elements
- let fruits = ["Apple", "Banana", "Mango"];
- console.log(fruits[1]); // Banana

- 3. Modifying Arrays
- let fruits = ["Apple", "Banana", "Mango"];
- fruits[1] = "Orange";
- console.log(fruits); // ["Apple", "Orange", "Mango"]

- Array Properties
- let fruits = ["Apple", "Banana", "Mango"];
- console.log(fruits.length); // 3

- Array Methods
- Add/Remove Elements
- let fruits = ["Apple", "Banana"];
- fruits.push("Mango"); // Add at end
- console.log(fruits); // ["Apple", "Banana", "Mango"]

- fruits.pop(); // Remove last
- console.log(fruits); // ["Apple", "Banana"]

- fruits.unshift("Grapes"); // Add at beginning
- console.log(fruits); // ["Grapes", "Apple", "Banana"]

- fruits.shift(); // Remove first
- console.log(fruits); // ["Apple", "Banana"]

- Iterating (Looping through Arrays)

- Using for loop

- let numbers = [10, 20, 30];
- for (let i = 0; i < numbers.length; i++) {
- console.log(numbers[i]);
- }

- Using for…of

- let colors = ["Red", "Green", "Blue"];
- for (let color of colors) {
- console.log(color);
- }

- Using forEach

- let numbers = [1, 2, 3];
- numbers.forEach(num => console.log(num));

- Useful Array Methods
- let nums = [1, 2, 3, 4, 5];

- console.log(nums.includes(3)); // true
- console.log(nums.indexOf(4)); // 3
- console.log(nums.join("-")); // "1-2-3-4-5"

To DO:

- Create an array of 5 animals and print them.
- Print the first and last element of an array.
- Add a new element to the end of an array. push()
- Remove the last element from an array. pop()
- Find the length of an array.
- Loop through an array and print all elements.
- Sort an array of numbers. Sort()
- Check if an element exists in an array.[use:includes()]
- Reverse an array.[use:reverse()]
- Merge two arrays.[use: concat()]

Objects

- An object is a collection of properties.
- Each property is a key-value pair (also called name-value pair).
- Keys are always strings (or symbols), and values can be any datatype (string, number, array, function, another object, etc.).

```
let objectName = {
  key1: value1,
  key2: value2,
  key3: value3
};
```

```
let student = {
  name: "Rahul",
  rollNo: 101,
  marks: 85,
  isPassed: true
};

// Accessing object properties
console.log(student.name);      // Dot notation → "Rahul"
console.log(student["marks"]);  // Bracket notation → 85
```

- Adding New Properties
- student.grade = "A";   // adds new property
- console.log(student.grade); // "A"
- Updating Properties
- student.marks = 90; // updates value
- console.log(student.marks); // 90

- Using Object Literal (Most Common & Simple)
- let student = {
- name: "Rahul",
- rollNo: 101,
- marks: 85
- };

- console.log(student.name);

- Using new Object() Constructor
- let person = new Object();
- person.name = "Anjali";
- person.age = 25;
- person.city = "Delhi";

- console.log(person.city); // "Delhi"

- Using a Function Constructor
- function Employee(id, name, salary) {
- this.id = id;
- this.name = name;
- this.salary = salary;
- }

- // Create objects
- let emp1 = new Employee(101, "Arun", 50000);
- let emp2 = new Employee(102, "Meera", 60000);

- console.log(emp1.name); // "Arun"
- console.log(emp2.salary); // 60000

- Using ES6 Classes (Modern Approach)
- class Car {
- constructor(brand, model, year) {
- this.brand = brand;
- this.model = model;
- this.year = year;
- }

- start() {
- console.log(this.brand + " " + this.model + " is starting...");
- }
- }

- // Create objects
- let car1 = new Car("Toyota", "Camry", 2020);
- let car2 = new Car("Honda", "City", 2022);

- car1.start(); // "Toyota Camry is starting..."
- car2.start(); // "Honda City is starting..."

Create an object student with properties name, rollNo, and marks. Print all properties.

```
let student = {
  name: "Rahul",
  rollNo: 101,
  marks: 85
};

console.log("Name:", student.name);
console.log("Roll No:", student.rollNo);
console.log("Marks:", student.marks);
```

Write a program to add a new property grade to the student object and print it.

```
const student = {
  name: 'John Doe',
  age: 20,
  major: 'Computer Science'
};
console.log('Original student object:');
console.log(student);

student.grade = 'A';

console.log('Updated student object with new grade property:');
console.log(student);
```

Create an object car with properties brand, model, and year. Write a method inside the object to display car details.

```
const car = {
  brand: 'Toyota',
  model: 'Camry',
  year: 2022,

displayDetails: function() {
    console.log(`Car Details:`);
    console.log(`Brand: ${this.brand}`);
    console.log(`Model: ${this.model}`);
    console.log(`Year: ${this.year}`);
  }
};

car.displayDetails();
```

Built - in Objects

- JavaScript built-in objects make programming easier by providing ready-made functionality for:
- Dates (Date)
- Numbers (Number, Math)
- Strings (String)
- Logic (Boolean, Symbol)
- Data storage (Array, Map, Set)
- JSON handling (JSON)
- Asynchronous tasks (Promise)

- Built-in objects are essential components of the **JavaScript** language that are available globally when a script starts executing.
- They provide a standardized way to work with common tasks like handling text, dates, mathematical calculations, and collections of data.

Core Built-in Objects

Object	Description	Example Usage
`Object`	The base object for all other objects. Used to create generic objects.	`const myObject = new Object();` or more commonly, `const myObject = {};`
`Function`	The base constructor for all JavaScript functions.	`const myFunction = new Function('a', 'b', 'return a + b');` (Less common, usually use function declaration/expression)
`Boolean`	A wrapper object for primitive boolean values (true and false).	`const isTrue = new Boolean(true);`
`Symbol`	A unique and immutable data type often used to make object properties private.	`const uniqueKey = Symbol('description');`
`Error`	Base object for standard and custom error objects.	`throw new Error('Something went wrong!');`

- **Text and Collections**
- These objects are used for manipulating strings and managing groups of data.
- **String**
- A wrapper object for primitive string values. Provides methods for inspecting, searching, and manipulating text.
- **Key Methods:**
  - **.length**: Property to get the string length.
  - **.toUpperCase(), .toLowerCase()**: Change case.
  - **.indexOf(), .includes()**: Search within the string.
  - **.slice(), .substring()**: Extract parts of the string.
  - **.trim()**: Remove whitespace from both ends.
  - **.split()**: Convert a string into an array of substrings.
- Think of the string object as a **toolbox** for text.

- **Array**
- Global object used to construct arrays, which are **list-like objects** that hold multiple values (of any type).
- **Key Methods:**
  - **.length**: Property to get the number of elements.
  - **.push(), .pop()**: Add/remove elements from the *end*.
  - **.unshift(), .shift()**: Add/remove elements from the *beginning*.
  - **.map(), .filter(), .reduce()**: Methods for iteration and transformation (crucial for functional programming).
  - **.concat()**: Merge arrays.
  - **.includes()**: Check if an element exists.
- An array is like a **numbered list** or a **train** where each car holds a value.

- **RegExp (Regular Expression)** Used for pattern matching with strings. Essential for complex search and replace operations.
- **Key Methods:**
  - **.test()**: Checks if a pattern exists in a string (returns true or false).
  - **.exec()**: Executes a search for a match in a specified string.

- **Math**
- A **static object** (you never *instantiate* it with new) that provides constants and functions for mathematical operations.
- **Key Constants:**
  - **Math.PI**, **Math.E** (Euler's constant)
- **Key Methods:**
  - **Math.random()**: Generates a pseudo-random number between 0 (inclusive) and 1 (exclusive).
  - **Math.round(), Math.floor(), Math.ceil()**: Rounding functions.
  - **Math.abs()**: Absolute value.
  - **Math.pow(), Math.sqrt()**: Powers and square root.
  - **Math.max(), Math.min()**: Find the largest/smallest of zero or more numbers.
- The **Math** object is a **scientific calculator**.

- **Number :** A wrapper object for numeric primitive values. Contains constants for numeric limits and methods for formatting.
- **Key Constants:**
  - **Number.MAX_VALUE, Number.MIN_VALUE**
  - **Number.isInteger()**: Checks if a value is a whole number.
- **Key Methods:**
  - **.toFixed()**: Formats a number to a specific number of decimal places.
  - **.toString()**: Converts the number to a string (can specify the base, e.g., base 16 for hex).

- **Date**
- Object used to work with dates and times.
- **Key Methods:**
  - **new Date()**: Creates a new date object, representing the current date and time.
  - **.getFullYear(), .getMonth()** (0-indexed!), **.getDate()**, **.getHours()**, etc.
  - **.getTime()**: Returns the number of milliseconds since the **Epoch** (January 1, 1970, 00:00:00 UTC). This is useful for comparing times.

- **Structured Data**
- **Map and Set Purpose:** provide new, more efficient ways to handle data collections.
- **Map**: A collection of **key/value pairs** where the keys can be of *any* type (unlike plain objects where keys are usually strings/symbols).
  - **Key Methods:** .set(key, value), .get(key), .has(key), .delete(key).
- **Set**: A collection of **unique values** (no duplicates).
  - **Key Methods:** .add(value), .has(value), .delete(value).

JavaScript Debuggers

- **What is a Debugger?**
- A **debugger** is a tool or feature that helps you **find and fix errors (bugs)** in your JavaScript code.
  It allows you to:
- Pause code execution at specific points (breakpoints)
- Inspect variable values
- Step through your code line by line
- Understand program flow
-

Ways to Debug JavaScript

Method	Description	Example
`console.log()`	Prints information to the console	`console.log("Debug:", x);`
`console.error()`	Displays error message	`console.error("Something went wrong!");`
`console.warn()`	Displays warning message	`console.warn("Check this value");`
`console.table()`	Displays data in table format	`console.table(students);`

- **Using the debugger Statement**
- The debugger keyword acts like a **breakpoint** in code.
- **Example:**
- let x = 5;
- let y = 10;
- let sum = x + y;
-
- debugger; // Execution pauses here in the browser's developer tools
-
- console.log("Sum:", sum);
- When the browser encounters debugger;, it **pauses execution** and opens the **DevTools Sources panel**, allowing you to inspect variables and step through code.
-

- **Browser Developer Tools (DevTools)**
- Most browsers like **Google Chrome**, **Firefox**, **Edge**, and **Safari** have **built-in debugging tools**.
-

- **Using IDEs / Code Editors**
- Modern editors like **VS Code**, **WebStorm**, and **Sublime** provide:
- Integrated debuggers
- Breakpoints
- Watch variables
- Step controls
- Example (VS Code):
- Press F5 → Choose **Chrome Debugging**
- Add **breakpoints** in code
- Use **Debug Console** to inspect variables
-

- **Linting Tools (Prevent Bugs Early)**
- While not debuggers, **linters** like **ESLint** detect issues **before runtime**.
- npm install eslint --save-dev
- They highlight:
- Syntax errors
- Unused variables
- Bad practices
-

Java Debuggers-Summary

Method	Description	Use Case
`console.log()`	Print values for quick check	Small scripts
`debugger` statement	Pause execution at runtime	Simple apps
Browser DevTools	Inspect, step, watch variables	Most powerful & visual
VS Code Debugger	Integrated debugging	Professional development
ESLint	Prevent errors early	Code quality

DOM (Document Object Model)

- The DOM is a **tree-like structure** of an HTML page.
- Each HTML tag becomes a **node** (element).
- The DOM lets JavaScript **access and modify** HTML elements.

DOM HistoryandLevel

DOM Level	Year	Key Features
DOM 0	Pre-1998	Browser-specific, no standard
DOM 1	1998	Standardized Core & HTML DOM, element access
DOM 2	2000	Events, CSS, Traversal, Namespaces
DOM 3	2004	XPath, Load/Save, Keyboard events
DOM 4	Ongoing	Living standard, modern APIs, querySelector

Structure Example (HTML → DOM Tree)

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

```
Document
└── html
    ├── head
    │   └── title
    │       └── "My Page"
    └── body
        ├── h1
        │   └── "Hello World"
        └── p
            └── "This is a paragraph."
```

Explanation:

Element	Description
Document	Root of the DOM tree (represents the entire HTML page)
html	Root element of the HTML document
head	Contains metadata (like title)
title	Defines the title shown in the browser tab
body	Contains visible page content
h1, **p**	Child elements of body — actual content

- **Relationship Summary:**
- **Parent**: Node that contains other nodes
  (e.g. body is the parent of h1 and p)
- **Child**: Node inside another node
  (e.g. h1 is a child of body)
- **Sibling**: Nodes with the same parent
  (e.g. h1 and p are siblings)
-

DOMEventHandling

- **What is an Event?**
- An **event** is an action or occurrence in the browser that JavaScript can respond to.
  Examples:
- A **click** on a button
- A **keypress** on the keyboard
- A **mouse over** an image
- A **form submission**
- **Event Handling** means writing JavaScript code that **responds to these events**.
-

- **Why Use Event Handling?**
- To make web pages **interactive and dynamic**:
- Validate forms
- Respond to user actions
- Update content dynamically

Types of Event Handling Methods

Method	Description	Example
1. Inline Event Handling	Write code inside HTML tag	`<button onclick="sayHello()">Click</button>`
2. DOM Level 0 (Traditional)	Assign function to element property	`btn.onclick = function() {...}`
3. DOM Level 2 (Modern)	Use `addEventListener()` method	`btn.addEventListener("click", handler)`

Inline Event Handling

```
<!DOCTYPE html>
<html>
<head>
  <title>Inline Event Example</title>
</head>
<body>
  <h2>Inline Event Handling</h2>
  <button onclick="sayHello()">Click Me</button>

  <script>
    function sayHello() {
      alert("Hello! You clicked the button.");
    }
  </script>
</body>
</html>
```

- **Explanation:**
- The onclick attribute is placed **directly inside the HTML**.
- When the button is clicked, the **sayHello()** function is called.

DOM Level 0 (Traditional Method)

```
<!DOCTYPE html>
<html>
<head>
  <title>DOM Level 0 Example</title>
</head>
<body>
  <h2>DOM Level 0 Event Handling</h2>
  <button id="btn">Click Me</button>

  <script>
    let button = document.getElementById("btn");
    button.onclick = function() {
      alert("Button clicked using DOM Level 0!");
    };
  </script>
</body>
</html>
```

- We use **document.getElementById()** to get the element.
- Assign the event handler to the **.onclick** property.
- **Only one** event handler per event can be assigned this way.
-

DOM Level 2 (Modern addEventListener) (Recommended)

```
<!DOCTYPE html>
<html>
<head>
  <title>DOM Level 2 Example</title>
</head>
<body>
  <h2>DOM Level 2 Event Handling</h2>
  <button id="btn2">Click Me</button>

  <script>
    let btn2 = document.getElementById("btn2");
    btn2.addEventListener("click", showMessage);

    function showMessage() {
      alert("Button clicked using addEventListener!");
    }
  </script>
</body>
</html>
```

- You can attach **multiple handlers** to the same event.
- Syntax:
- element.addEventListener("eventType", handlerFunction);
-

Button Click Using addEventListener()

```
<!DOCTYPE html>
<html>
<head>
  <title>DOM Level 2 Event Handling</title>
</head>
<body>
  <h2>DOM Level 2 Example</h2>

  <button id="myButton">Click Me</button>

  <script>
    // Step 1: Get the element
    const btn = document.getElementById("myButton");

    // Step 2: Attach event listener
    btn.addEventListener("click", showMessage);

    // Step 3: Define the function
    function showMessage() {
      alert("Hello! You clicked the button.");
    }
  </script>
</body>
</html>
```

- getElementById("myButton") → Selects the button.
- addEventListener("click", showMessage) → Listens for a click event.
- When the button is clicked, the showMessage() function runs.

Multiple Events on Same Element

```
<!DOCTYPE html>
<html>
<head>
  <title>Multiple Events Example</title>
</head>
<body>
  <h2>Multiple Events Example</h2>
  <button id="multi">Hover or Click Me</button>

  <script>
    let btn = document.getElementById("multi");

    btn.addEventListener("click", function() {
      alert("You clicked the button!");
    });

    btn.addEventListener("mouseover", function() {
      console.log("Mouse is over the button!");
    });
  </script>
</body>
</html>
```

addEventListener() allows you to add multiple events.

Here, one for **click**, one for **mouseover**.

•

Event Object (Getting Event Details)

```
<!DOCTYPE html>
<html>
<head>
  <title>Event Object Example</title>
</head>
<body>
  <h2>Event Object Example</h2>
  <button id="eventBtn">Click Me</button>

  <script>
    document.getElementById("eventBtn").addEventListener("click", function(event) {
      console.log("Event Type:", event.type);
      console.log("Target Element:", event.target);
      alert("You triggered a " + event.type + " event!");
    });
  </script>
</body>
</html>
```

- The **event object** gives details about the event:
- event.type → type of event (click, mouseover)
- event.target → element that triggered event
-

Form Validation Using Event Handling

```
<html>
<head>
 <title>Form Validation</title>
</head>
<body>
 <h2>Form Validation Example</h2>
 <form id="myForm">
  <input type="text" id="name" placeholder="Enter your name">
  <button type="submit">Submit</button>
 </form>

 <script>
  document.getElementById("myForm").addEventListener("submit", function(event) {
   let name = document.getElementById("name").value;
   if (name === "") {
    alert("Please enter your name.");
    event.preventDefault(); // stops form submission
   } else {
    alert("Form submitted successfully!");
   }
  });
 </script>
</body>
</html>
```

- **Explanation:**
- Used submit event to validate input.
- event.preventDefault() stops form from submitting if input is empty.
-

Event Handling Method	Syntax	Multiple Handlers	Recommended
Inline	`onclick="func()"`	No	Not Recommended
DOM Level 0	`element.onclick = func`	No	Limited
DOM Level 2	`element.addEventListener()`	Yes	Recommended

Check whether a number is even or odd

```
<!DOCTYPE html>
<html>
<body>
  <script>
    let num = 7; // change value to test
    if (num % 2 === 0) {
      console.log(num + " is Even");
    } else {
      console.log(num + " is Odd");
    }
  </script>
</body>
</html>
```

Even or Odd (User Input)

```
<!DOCTYPE html>
<html>
<body>
  <script>
    // Ask user for input
    let num = prompt("Enter a number:");

    // Convert input to number
    num = Number(num);

    // Check if even or odd
    if (num % 2 === 0) {
      alert(num + " is Even");
    } else {
      alert(num + " is Odd");
    }
  </script>
</body>
</html>
```

- Ask input → prompt()
- Convert input → Number()
- Check condition → if...else
- Show result → alert()

- prompt("Enter a number:") → Opens a popup box asking the user to type something.
- Whatever the user types is stored in the variable num.
- At this stage, num is a string, not a number.
- Number(num) converts the string into a number.
- For example:
- If user types "10" → it becomes 10 (number).
- If user types "7" → it becomes 7.
- num % 2 → calculates the remainder when the number is divided by 2.
- If remainder = 0 → the number is even.
- Otherwise → the number is odd.
- alert(...) → shows a popup message with the result.

Print the multiplication table of a number using a loop

```
<!DOCTYPE html>
<html>
<body>
  <script>
    let n = 5; // number to print table
    for (let i = 1; i <= 10; i++) {
      console.log(n + " x " + i + " = " + (n * i));
    }
  </script>
</body>
</html>
```

Create an object student with properties and display it

```
<!DOCTYPE html>
<html>
<body>
  <script>
    let student = {
      name: "Ravi",
      rollNo: 101,
      marks: 88
    };

    console.log("Name: " + student.name);
    console.log("Roll No: " + student.rollNo);
    console.log("Marks: " + student.marks);
  </script>
</body>
</html>
```

Function that takes 2 numbers and returns the maximum

```
<!DOCTYPE html>
<html>
<body>
  <script>
    function getMax(a, b) {
      return (a > b) ? a : b;
    }

    console.log("Maximum is: " + getMax(10, 25));
  </script>
</body>
</html>
```

Button that changes the background color of the page when clicked

```
<!DOCTYPE html>
<html>
<body>
  <button onclick="changeColor()">Change Background</button>

  <script>
    function changeColor() {
      // pick a random color
      let colors = ["lightblue", "lightgreen", "yellow", "pink", "orange"];
      let randomIndex = Math.floor(Math.random() * colors.length);
      document.body.style.backgroundColor = colors[randomIndex];
    }
  </script>
</body>
</html>
```