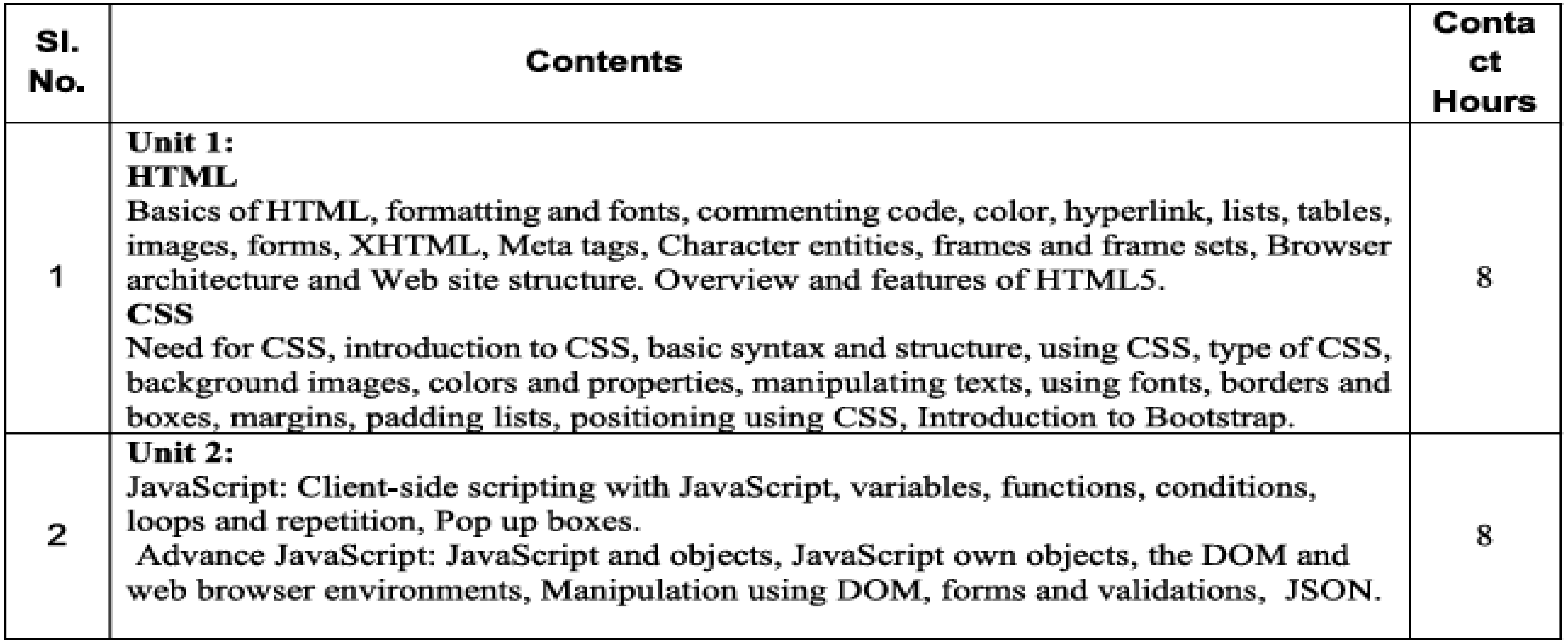
Full Stack Web Development

TCS 693

Full Stack Web Development TCS 693



Sl. No.	Contents	Contact Hours
1	**Unit 1:** **HTML** Basics of HTML, formatting and fonts, commenting code, color, hyperlink, lists, tables, images, forms, XHTML, Meta tags, Character entities, frames and frame sets, Browser architecture and Web site structure. Overview and features of HTML5. **CSS** Need for CSS, introduction to CSS, basic syntax and structure, using CSS, type of CSS, background images, colors and properties, manipulating texts, using fonts, borders and boxes, margins, padding lists, positioning using CSS, Introduction to Bootstrap.	8
2	**Unit 2:** JavaScript: Client-side scripting with JavaScript, variables, functions, conditions, loops and repetition, Pop up boxes. Advance JavaScript: JavaScript and objects, JavaScript own objects, the DOM and web browser environments, Manipulation using DOM, forms and validations, JSON.	8

3	**Unit 3:** **XML**: Introduction to XML, uses of XML, simple XML, XML key components, DTD and Schemas. **Ajax** : Introduction to Ajax , XMLHttpRequest Methods and Properties, JavaScript code for Ajax , Implementing Ajax techniques with a server scripting language , Handling the Response, Ajax with JSon **JQuery**: jQuery Introduction, Install and Use jQuery Library, jQuery Syntax, Ajax with jQuery, Load method, jQuery get and getJson methods.	10
4	**Unit 4:** **PHP** XAMPP Server Configuration, Introduction and basic syntax of PHP, decision and looping with examples, PHP and HTML, Arrays, Functions, Browser control and detection, string, Form processing, Files. Advance Features: Cookies and Sessions, Basic commands with PHP examples, Connection to server, creating database, selecting a database, listing database, listing table names, creating a table, inserting data, altering tables, queries, deleting database, deleting data and tables.	10

5	**Unit 5:** MERN Web Application Deployment, Content Management System (CMS). MERN Stack: MongoDB: Overview , Environment, Data Modelling, Database Operations. Express: Installing ExpressJS, Environment, Routing React: React Intro, , React Lifecycle, Building Forms using React, states and components. Node: Install node, simple server, HTML and JSON Response.	12
	Total	**48**

World Wide Web (WWW)?

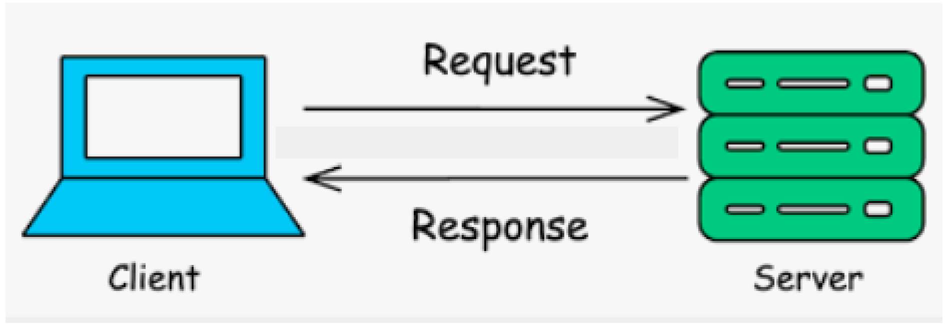
The **World Wide Web (WWW)** is a system of interlinked web pages that are accessed through the Internet using a **web browser**.

- Invented by Tim Berners-Lee
- Uses HTTP/HTTPS protocol
- Information is stored in the form of web pages
- Web pages are written using HTML

Client–Server Architecture

How a Website Works

- User enters URL in browser
- Browser sends request to server
- Server processes request
- Response sent back to browser
- Browser renders web page



Components

- Client: Browser (Chrome, Edge)
- Server: Apache, Nginx
- Protocol: HTTP / HTTPS

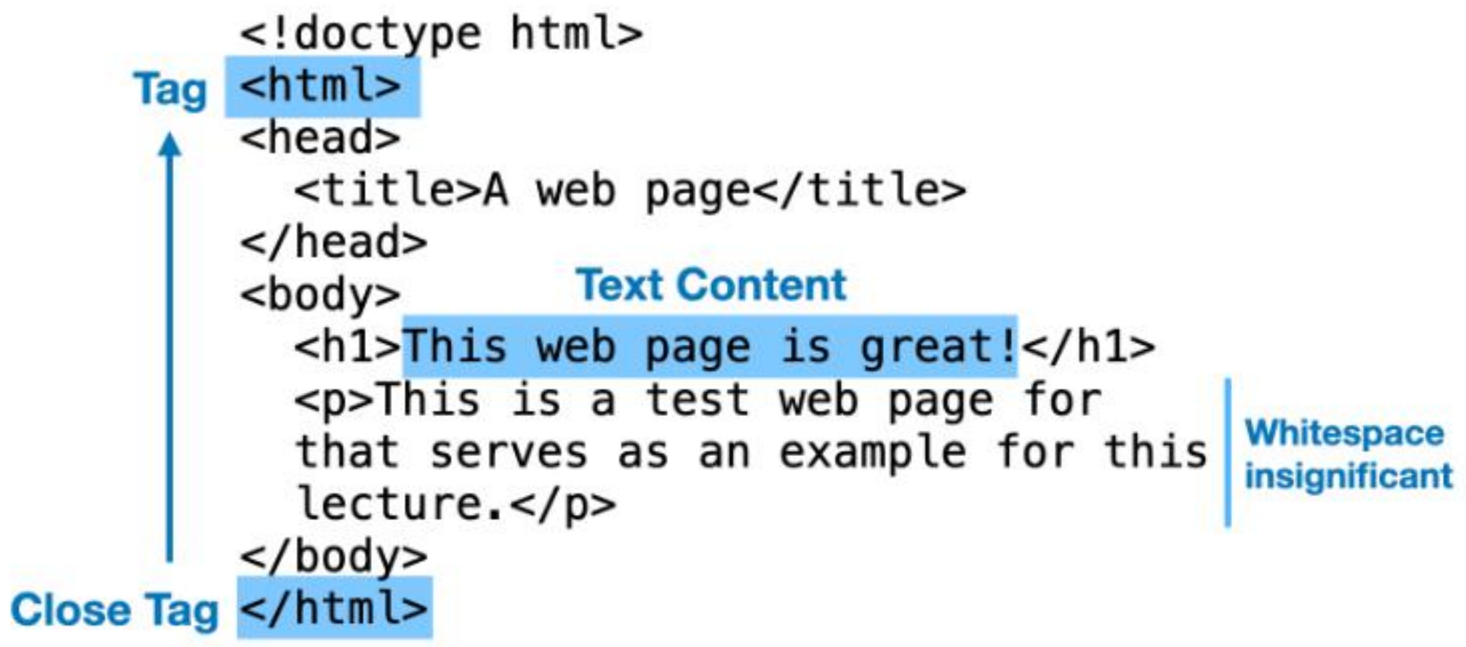
Website Structure

Frontend (Client Side)

- HTML → Structure
- CSS → Styling
- JavaScript → Behavio

Backend (Server Side)

- PHP, Node.js, Python
- Database (MySQL, MongoDB)



```
<!doctype html>
<html>
<head>
  <title>A web page</title>
</head>
<body>
  <h1>This web page is great!</h1>
  <p>This is a test web page for
  that serves as an example for this
  lecture.</p>
</body>
</html>
```

HTML

HTML (HyperText Markup Language) is a markup language used to structure web pages.

Features

- Easy to learn
- Platform independent
- Supports multimedia
- Not a programming language

Markup Language

A markup language is used to structure, format, and present data. It does not perform logic, calculations, or decision-making.

Key Characteristics

- Describes structure and layout
- No variables, loops, or conditions
- Interpreted by browsers or parsers
- Used for content presentation

Examples

- HTML – Web page structure
- XML – Data storage and transfer
- SGML – Standard generalized markup
- XHTML – HTML with XML rules
- Markdown – Lightweight formatting

Scripting Language

A scripting language is used to automate tasks, add interactivity, and control behavior of applications—often inside another environment.

Key Characteristics

- Interpreted (no compilation step)
- Used for **client-side or server-side tasks**
- Supports variables, loops, conditions
- Executes line-by-line

Examples

- **JavaScript** – Client-side scripting
- **PHP** – Server-side scripting
- **Python** – Automation & scripting
- **Ruby** – Web scripting
- **Shell Script** – OS automation

Programming Language

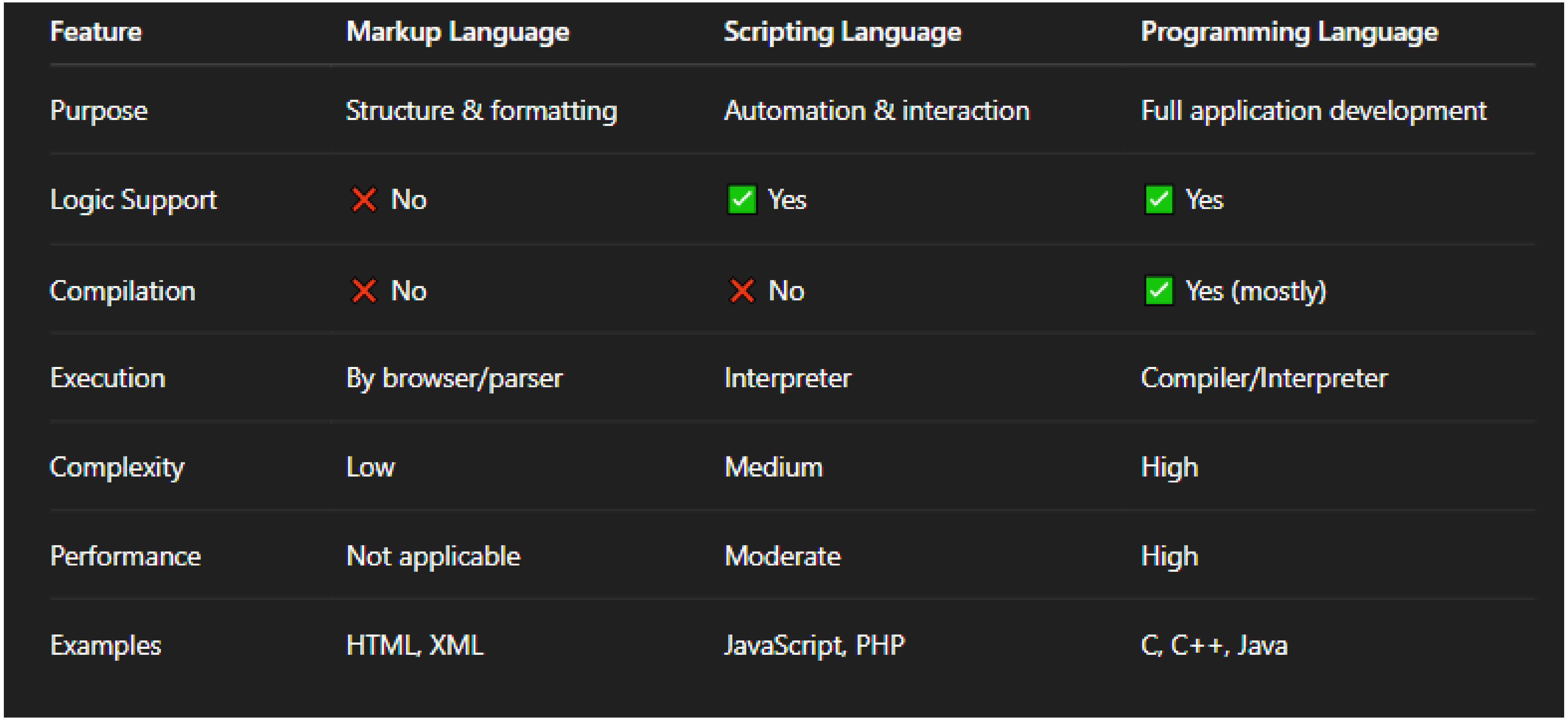
A programming language is used to develop complete software applications, perform complex computations, and build systems from scratch.

Key Characteristics

- Can be compiled or interpreted
- Supports **OOP, data structures, algorithms**
- Used to build **desktop, mobile, web, and system software**
- High performance and scalability

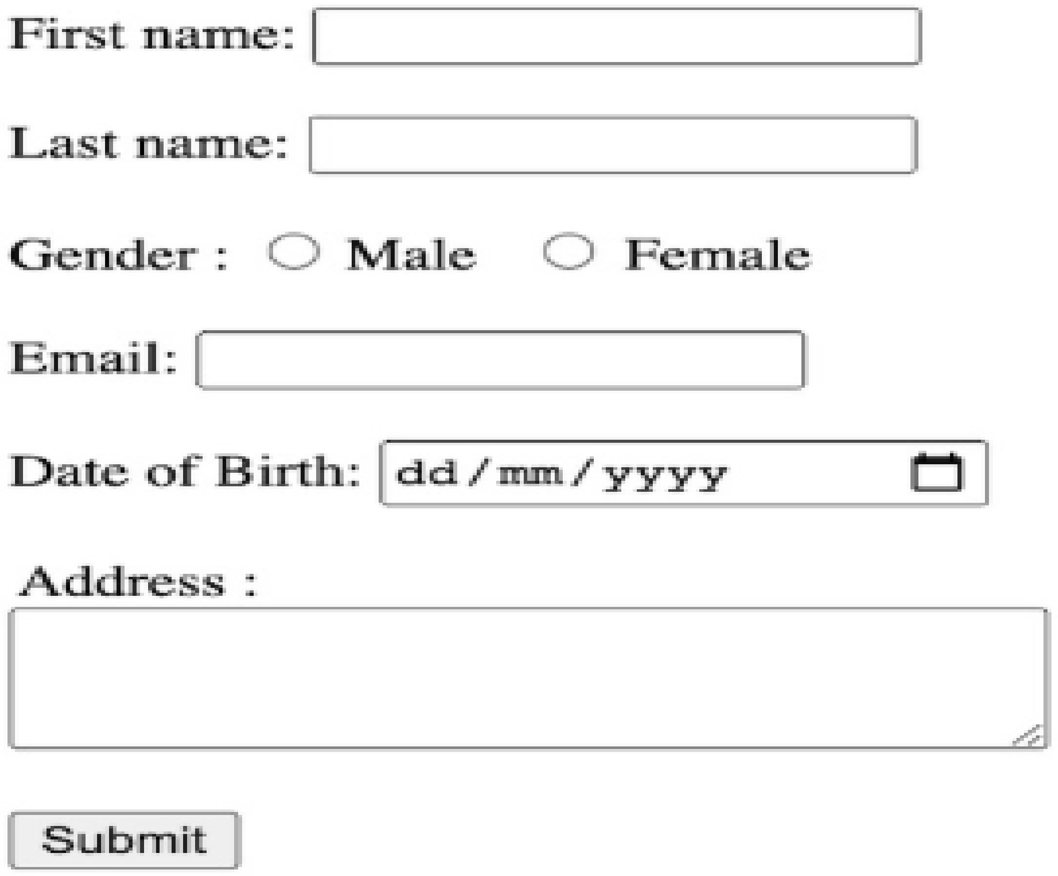
Examples

- **C** – System programming
- **C++** – High-performance apps
- **Java** – Enterprise & Android apps
- **Python** – General-purpose programming
- **C#** – Windows & game development



Feature	Markup Language	Scripting Language	Programming Language
Purpose	Structure & formatting	Automation & interaction	Full application development
Logic Support	❌ No	✅ Yes	✅ Yes
Compilation	❌ No	❌ No	✅ Yes (mostly)
Execution	By browser/parser	Interpreter	Compiler/Interpreter
Complexity	Low	Medium	High
Performance	Not applicable	Moderate	High
Examples	HTML, XML	JavaScript, PHP	C, C++, Java

•**Markup language** defines *what the content is*.
•**Scripting language** defines *how the content behaves*.
•**Programming language** defines *how the system works*.


First name:

Last name:

Gender : ○ Male ○ Female

Email:

Date of Birth: dd / mm / yyyy

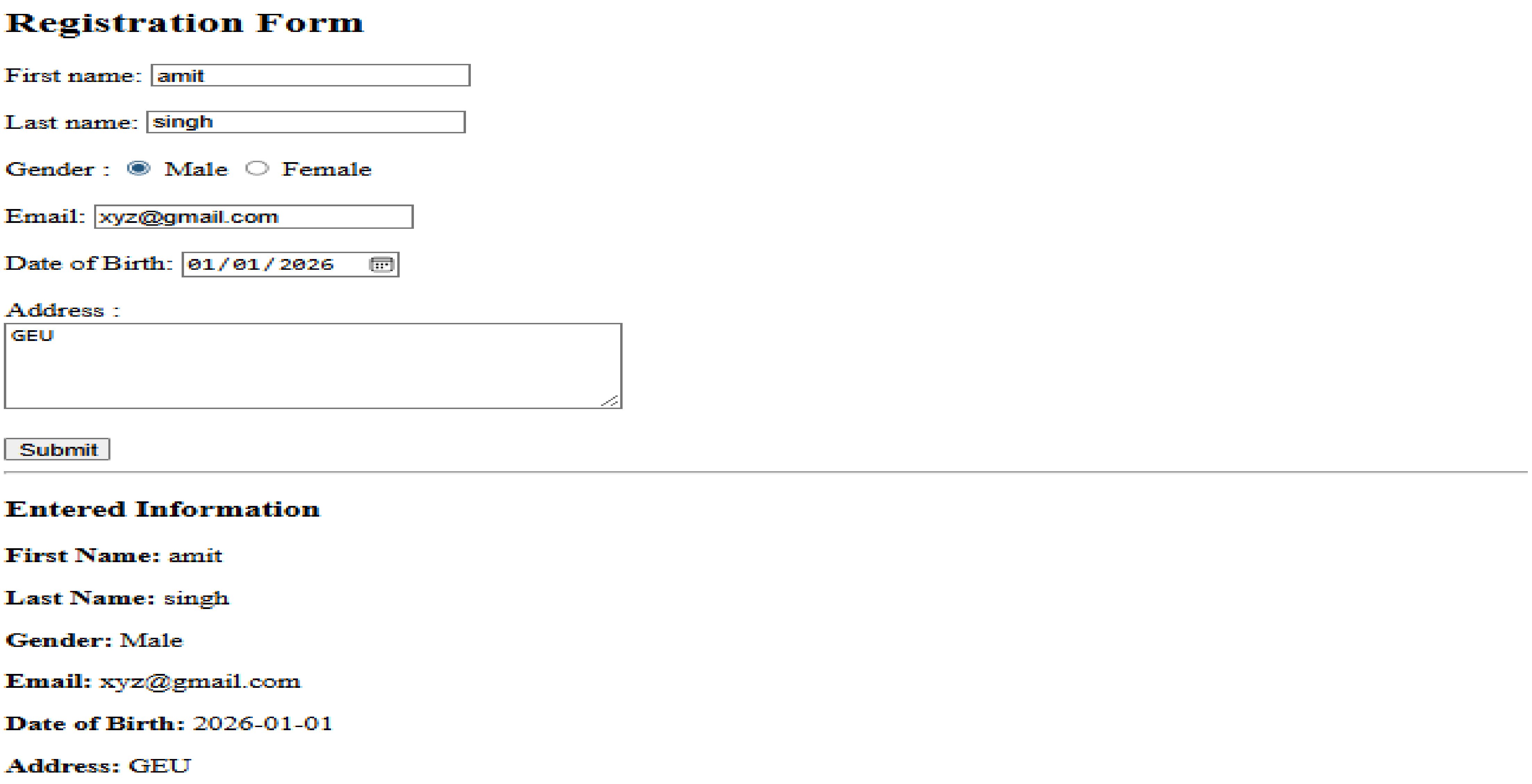
Address :

Submit

```html
<form>
   <label>First name:</label>
   <input type="text" name="firstname"><br><br>
   <label>Last name:</label>
   <input type="text" name="lastname"><br><br>
   <label>Gender :</label>
   <input type="radio" name="gender" value="Male"> Male
   <input type="radio" name="gender" value="Female"> Female
   <br><br>
   <label>Email:</label>
   <input type="email" name="email"><br><br>
   <label>Date of Birth:</label>
   <input type="date" name="dob"><br><br>
   <label>Address :</label><br>
   <textarea name="address" rows="4" cols="40"></textarea>
   <br><br>
   <input type="submit" value="Submit">
</form>
```

```
<form id="myForm" onsubmit="displayData(); return false;">
   <label>First name:</label>
   <input type="text" id="firstname"><br><br>
   <label>Last name:</label>
   <input type="text" id="lastname"><br><br>
   <label>Gender :</label>
   <input type="radio" name="gender" value="Male"> Male
   <input type="radio" name="gender" value="Female"> Female
   <br><br>
   <label>Email:</label>
   <input type="email" id="email"><br><br>
   <label>Date of Birth:</label>
   <input type="date" id="dob"><br><br>
   <label>Address :</label><br>
   <textarea id="address" rows="4" cols="40"></textarea>
   <br><br>
   <input type="submit" value="Submit">
</form>
```

```
<hr>
<h3>Entered Information</h3>
<div id="output"></div>
<script>
function displayData() {
    var fname = document.getElementById("firstname").value;
    var lname = document.getElementById("lastname").value;
    var email = document.getElementById("email").value;
    var dob = document.getElementById("dob").value;
    var address = document.getElementById("address").value;
    var gender = "";
    var genders = document.getElementsByName("gender");
    for (var i = 0; i < genders.length; i++) {
        if (genders[i].checked) {
            gender = genders[i].value;
        }
    }
document.getElementById("output").innerHTML =
        "<p><b>First Name:</b> " + fname + "</p>" +
        "<p><b>Last Name:</b> " + lname + "</p>" +
        "<p><b>Gender:</b> " + gender + "</p>" +
        "<p><b>Email:</b> " + email + "</p>" +
        "<p><b>Date of Birth:</b> " + dob + "</p>" +
        "<p><b>Address:</b> " + address + "</p>";
}
</script>
</body>
</html>
```



Registration Form

First name: amit

Last name: singh

Gender : [✓ Male] Female

Email: xyz@gmail.com

Date of Birth: 01/01/2026

Address :
GEU

Submit

Entered Information

First Name: amit

Last Name: singh

Gender: Male

Email: xyz@gmail.com

Date of Birth: 2026-01-01

Address: GEU

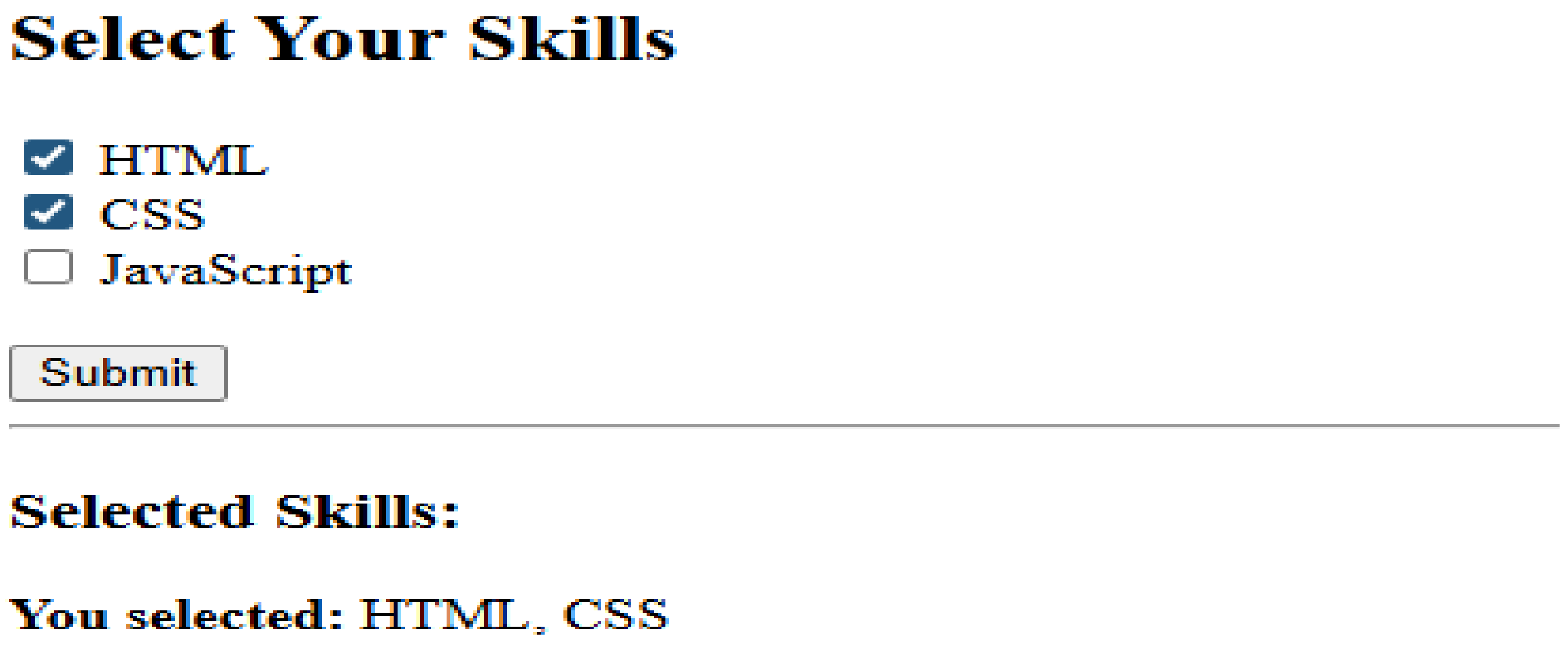
Check Boxes

```
<html>
<head>    <title>Checkbox Example</title> </head>
<body>
<h2>Select Your Skills</h2>
<form onsubmit="showSkills(); return false;">
    <input type="checkbox" id="html" name="skills" value="HTML">
    <label for="html">HTML</label><br>
    <input type="checkbox" id="css" name="skills" value="CSS">
    <label for="css">CSS</label><br>
    <input type="checkbox" id="js" name="skills" value="JavaScript">
    <label for="js">JavaScript</label><br><br>
    <input type="submit" value="Submit">
</form>
<hr><h3>Selected Skills:</h3>
<div id="output"></div>
```

```
<script>
function showSkills() {
    var skills = document.getElementsByName("skills");
    var selectedSkills = [];

    for (var i = 0; i < skills.length; i++) {
        if (skills[i].checked) {
            selectedSkills.push(skills[i].value);
        }
   }
    if (selectedSkills.length === 0) {
        document.getElementById("output").innerHTML =
            "<p>No skill selected</p>";
    } else {
        document.getElementById("output").innerHTML =
            "<p><b>You selected:</b> " + selectedSkills.join(", ") + "</p>";
    }
}
</script>
</body>
</html>
```

Output



Select Your Skills

☑ HTML
☑ CSS
☐ JavaScript

Submit

Selected Skills:

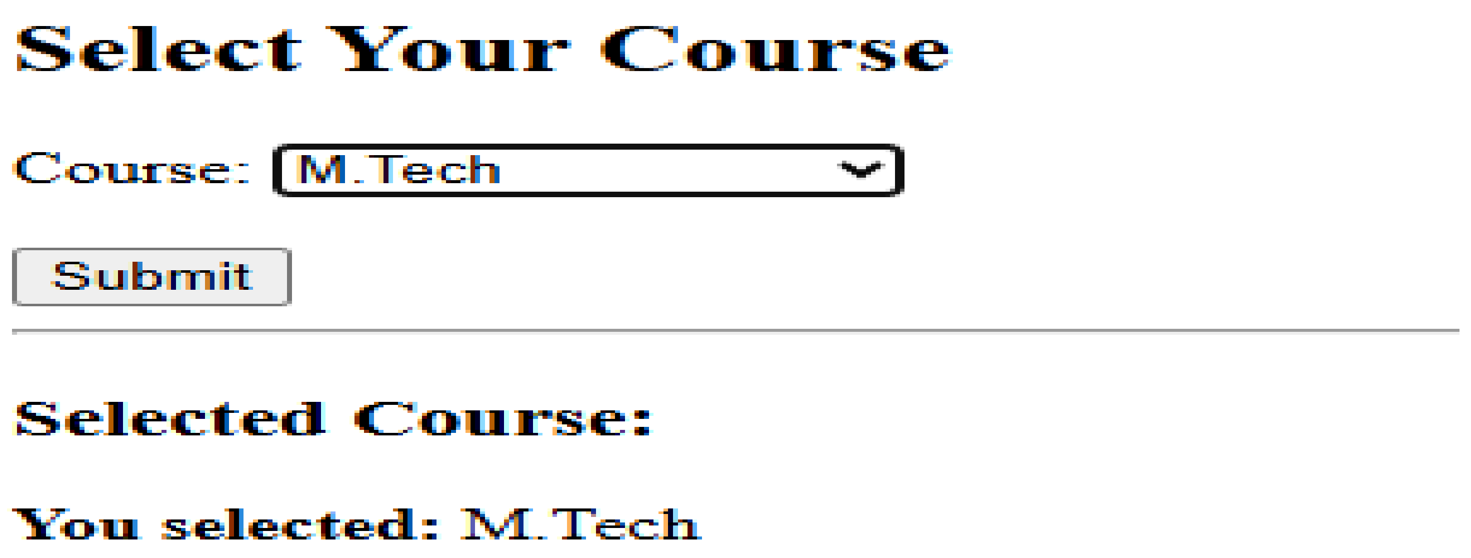
You selected: HTML, CSS

Dropdown

```
<html>
<head>    <title>Dropdown Example</title> </head>
<body> <h2>Select Your Course</h2>
<form onsubmit="showCourse(); return false;">
    <label for="course">Course:</label>
    <select id="course">
        <option value="">-- Select Course --</option>
        <option value="B.Tech">B.Tech</option>
        <option value="M.Tech">M.Tech</option>
        <option value="BCA">BCA</option>
        <option value="MCA">MCA</option>
    </select>    <br><br>
    <input type="submit" value="Submit">
</form>
```

```
<h3>Selected Course:</h3>
<div id="output"></div>
<script>
function showCourse() {
    var course = document.getElementById("course").value;

    if (course === "") {
        document.getElementById("output").innerHTML =
            "<p>No course selected</p>";
    } else {
        document.getElementById("output").innerHTML =
            "<p><b>You selected:</b> " + course + "</p>";
    }
}
</script>
</body>
</html>
```



Select Your Course

Course: M.Tech

Submit

Selected Course:

You selected: M.Tech

Popup boxes

Alert

- Displays a **simple message**
- No user input
- Only **OK** button

```
<html><body>
<h2>Alert Box Example</h2>
<button onclick="showAlert()">Click Me</button>
<script>
function showAlert() {
    alert("Form submitted successfully!");
}
</script>
</body></html>
```

Confirm Box

- Takes Yes / No decision
- Returns true or false

```
<html> <body><h2>Confirm Box Example</h2>
<button onclick="showConfirm()">Delete Record</button>
<p id="result"></p>
<script>
function showConfirm() {
  var choice = confirm("Are you sure you want to delete?");

  if (choice) {
    document.getElementById("result").innerHTML = "Record deleted";
  } else {
    document.getElementById("result").innerHTML = "Action cancelled";
  }
}
</script> </body> </html>
```

Prompt Box

- Takes user input
- Returns entered value or null

```
<html> <body><h2>Prompt Box Example</h2>
<button onclick="showPrompt()">Enter Name</button>
<p id="output"></p>
<script>
function showPrompt() {
    var name = prompt("Enter your name:");

    if (name === null || name === "") {
        document.getElementById("output").innerHTML = "No name entered";
    } else {
        document.getElementById("output").innerHTML =
            "Hello, " + name;
    }
}
</script> </body> </html>
```

Basic Form Validation

Form validation ensures that user input is correct and complete before submitting data to the server.

It can be done using:

- HTML5 attributes (client-side)
- JavaScript logic (custom client-side)

Validation improves:

- Data accuracy
- User experience

```
<form onsubmit="return validateForm()">
    Name: <input type="text" id="name"><br><br>
    <input type="submit" value="Submit">
</form>
<script>
function validateForm() {
    var name = document.getElementById("name").value;

    if (name === "") {
        alert("Name is required");
        return false;
    }
    return true;
}
</script>
```

Required Fields

A required field must be filled before form submission.

<u>HTML5 Method</u>

```
<form>
    Name: <input type="text" required><br><br>
    <input type="submit">
</form>
```

JavaScript Method

```
<input type="text" id="fname">
<button onclick="check()">Submit</button>
<script>
function check() {
    if (document.getElementById("fname").value === "") {
        alert("Field cannot be empty");
    }
}
</script>
```

Email & Number Validation

Using HTML5

```
<input type="email" required>
```

Using JavaScript

```
<input type="text" id="email">
<button onclick="validateEmail()">Check</button>

<script>
function validateEmail() {
   var email = document.getElementById("email").value;

   if (!email.includes("@")) {
      alert("Invalid email");
   }
}
</script>
```

Number Validation

<u>Using HTML5</u>

```
<input type="number" min="1" max="100" required>
```

<u>Using JavaScript</u>

```
<input type="text" id="age">
<button onclick="validateAge()">Check</button>
<script>
function validateAge() {
    var age = document.getElementById("age").value;
    if (isNaN(age)) {
        alert("Enter a valid number");
    }
}
</script>
```

Pattern Attribute

The pattern attribute uses regular expressions to restrict input format.

Mobile Number

```
<input type="text"  pattern="[0-9]{10}"  title="Enter 10 digit mobile number“
required>
```

Password

```
<input type="password"
    pattern="(?=.*[A-Z])(?=.*[0-9]).{6,}"
    title="Must contain uppercase letter and number">
```

Example

```
<form onsubmit="return validate()">
    Email:
    <input type="email" id="email" required><br><br>
    Age:
    <input type="number" id="age" required><br><br>
    <input type="submit">
</form>
<script>
function validate() {
    var age = document.getElementById("age").value;
    if (age < 18) {
        alert("Age must be 18 or above");
        return false;
    }
    return true;
}
</script>
```

What are HTML5 Input Types?

HTML5 input types provide built-in validation, better UI controls, and improved user experience by restricting the type of data a user can enter.

```
<input type="text" placeholder="Enter your name">
<input type="password" placeholder="Enter password">
//Hides characters for sensitive data.
<input type="email" placeholder="Enter email" required>
//Validates email format automatically.
<input type="number" min="1" max="100" placeholder="Enter age">
//Accepts only numeric values.
<input type="date"> //Provides a calendar for date selection.
<input type="time"> //Allows selection of time.
<input type="color"> //Opens a color picker.
<input type="range" min="0" max="100">//Creates a slider control for numeric range.
<input type="file">//Allows file upload.
```

Variables & Data Types

Variables store data values.

JavaScript has dynamic typing (type can change).

Main data types: Number, String, Boolean, Undefined, Null, BigInt, Symbol, Object.

```
<script>
let a = 10;          // Number
let b = "GEU";       // String
let c = true;        // Boolean
let d;               // Undefined
let e = null;        // Null
console.log(typeof a, typeof b, typeof c, typeof d, e);
</script>
```

var, let, const

var → function-scoped, can be re-declared (older, avoid)

let → block-scoped, can be reassigned

const → block-scoped, cannot be reassigned (value fixed)

```
<script>
var x = 5;
var x = 10; // allowed (re-declare)
let y = 5;
// let y = 10; // not allowed (re-declare)
y = 10; // allowed (reassign)
const z = 5; // z = 10; // not allowed (reassign)
</script>
```

Primitive vs Non-Primitive Types

Primitive: stored by value (copy)

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

Non-primitive: stored by reference

- Object, Array, Function

```
<script>
// Primitive (copy)
let p1 = 10;
let p2 = p1;
p2 = 20;
console.log(p1, p2); // 10 20

// Non-primitive (reference)
let arr1 = [1,2];
let arr2 = arr1;
arr2.push(3);
console.log(arr1, arr2); // [1,2,3] [1,2,3]
</script>
```

Operators

<u>Arithmetic + - * / % ** ++ --</u>

```
<script>
let a = 10, b = 3;
console.log(a+b, a-b, a*b, a/b, a%b, a**b);
</script>
```

<u>Relational (Comparison)> < >= <= == != === !==</u>

```
<script>
console.log(10 > 5);   // true
console.log(10 <= 5);  // false
</script>
```

Logical && || !

```
<script>
let age = 20;
console.log(age >= 18 && age <= 60); // true
console.log(!(age < 18));            // true
</script>
```

== vs ===

== compares values only (type conversion happens)

=== compares value + type (strict)

```
<script>
console.log(5 == "5");   // true (type conversion)
console.log(5 === "5");  // false (number vs string)
</script>
```

Switch

```
<script>
let day = 3;
switch(day) {
  case 1: console.log("Monday"); break;
  case 2: console.log("Tuesday"); break;
  case 3: console.log("Wednesday"); break;
  default: console.log("Invalid day");
}
</script>
```

Loops

<u>For</u>

```
<script>
for (let i = 1; i <= 5; i++) {
  console.log(i);
} </script>
```

<u>While</u>

```
<script>
let i = 1;
while (i <= 5) {
  console.log(i);
  i++;
}</script>
```

do-while

```
<script>
let j = 1;
do {
  console.log(j);
  j++;
} while (j <= 5);
</script>
```

Loop with Form Inputs

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

<h3>Select Subjects</h3>

<form onsubmit="showSubjects(); return false;">
  <input type="checkbox" name="sub" value="DBMS"> DBMS <br>
  <input type="checkbox" name="sub" value="OS"> OS <br>
  <input type="checkbox" name="sub" value="CN"> CN <br><br>
  <input type="submit" value="Submit">
</form>

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

```
<script>
function showSubjects() {
  let boxes = document.getElementsByName("sub");
  let selected = [];

  for (let i = 0; i < boxes.length; i++) {
    if (boxes[i].checked) {
      selected.push(boxes[i].value);
    }
  }
  document.getElementById("out").innerHTML =
    (selected.length === 0) ? "No subject selected" : "Selected: " + selected.join(", ");
}
</script>
</body>
</html>
```

Document Object Model (DOM) – Window Object

The Document Object Model (DOM) represents an HTML page as a structured tree of objects. It allows JavaScript to access, manipulate, and control web page elements dynamically. At the top level of the DOM hierarchy is the window object`, which represents the browser window or tab. All global JavaScript objects, functions, and variables automatically becom

Window Object

The window object is the global object in the browser environment. It controls browser-related features such as alerts, navigation, timing, screen size, and history.

- Parent of all other objects (document, navigator, screen, history)
- Represents the browser window
- Provides methods for interaction with the browse members of the window object.

Common Window Object Properties

Property	Description
window.document	Refers to the HTML document
window.location	Current URL of the page
window.history	Browser history
window.navigator	Browser information
window.screen	Screen resolution info
window.innerWidth	Width of browser window
window.innerHeight	Height of browser window

alert() – Display Message Box

```
<script>
  window.alert("Welcome to DOM!");
</script>
```

confirm() – Confirmation Dialog

```
<script>
  let result = window.confirm("Do you want to continue?");
  if(result){
    document.write("User clicked OK");
  } else {
    document.write("User clicked Cancel");
  }
</script>
```

prompt() – Input Dialog

```
<script>
  let name = window.prompt("Enter your name:");
  document.write("Hello " + name);
</script>
```

open() – Open New Window/Tab

```
<script>
  window.open("https://www.google.com");
</script>
```

close() – Close Window

```
<script>
  window.close();
</script>
```

setTimeout() – Execute After Delay

```
<script>
  setTimeout(function(){
    alert("This appears after 3 seconds");
  }, 3000);
</script>
```

setInterval() – Repeated Execution

```
<script>
  setInterval(function(){
    console.log("Running every 2 seconds");
  }, 2000);
</script>
```

history Object – Browser History

```
<script>
  window.history.back();    // Go to previous page
  window.history.forward(); // Go to next page
</script>
```

navigator Object – Browser Information

```
<script>
  document.write(navigator.appName);
  document.write("<br>");
  document.write(navigator.userAgent);
</script>
```

- window is the global object
- You can write alert() instead of window.alert()
- DOM manipulation is done through window.document
- The Window object is the top-level object in the DOM hierarchy and represents the browser window. It provides methods like alert(), prompt(), setTimeout() and properties like location, history, and navigator to control browser behavior using JavaScript.

Window Object Hierarchy

Window → Document → HTML → Head / Body → Elements

→ Location

→ History

→ Navigator

→ Screen

- window is the root object of the browser environment
- document controls the webpage content
- location manages URL navigation
- history manages browser navigation
- navigator gives browser & device info
- screen provides display details
- All JavaScript global variables & functions belong to window

DOM Manipulation (Document Object Model)

DOM represents the HTML page as a tree of objects.

JavaScript uses DOM methods to access and modify HTML elements dynamically.

getElementById()

Selects one element using its unique id.

```
<p id="demo">Hello</p>
<button onclick="changeText()">Click</button>
<script>
function changeText() {
    document.getElementById("demo").innerHTML = "Welcome to GEU";
}
</script>
```

getElementsByName()

Selects multiple elements with same name (used for radio/checkbox).

```
<input type="radio" name="gender" value="Male"> Male
<input type="radio" name="gender" value="Female"> Female
<button onclick="showGender()">Submit</button>
<script>
function showGender() {
    var g = document.getElementsByName("gender");
    for (var i = 0; i < g.length; i++) {
        if (g[i].checked) {
            alert(g[i].value);
        }
    }
}
</script>
```

getElementsByClassName()

Selects all elements of same class.

```
<p class="text">One</p>
<p class="text">Two</p>
<button onclick="changeAll()">Change</button>

<script>
function changeAll() {
    var x = document.getElementsByClassName("text");
    for (var i = 0; i < x.length; i++) {
        x[i].style.color = "red";
    }
}
</script>
```

querySelector()

Selects first matching element using CSS selector.

```
<p class="msg">Hello</p>
<button onclick="changeMsg()">Click</button>
<script>
function changeMsg() {
    document.querySelector(".msg").innerHTML = "DOM is Powerful";
}
</script>
```

JavaScript Events

Events occur due to user actions (click, type, move mouse).

onclick

```
<button onclick="sayHi()">Click Me</button>
<script>
function sayHi() {
   alert("Button Clicked");
}
</script>
```

Onchange

```
<select onchange="showCourse(this.value)">
  <option value="">Select</option>
  <option value="B.Tech">B.Tech</option>
  <option value="MCA">MCA</option>
</select>

<script>
function showCourse(val) {
    alert("Selected: " + val);
}
</script>
```

onmouseover

```
<p onmouseover="changeColor(this)">Hover Me</p>
<script>
function changeColor(x) {
   x.style.color = "blue";
}
</script>
```

onkeyup

```
<input type="text" onkeyup="showText(this.value)">
<p id="out"></p>
<script>
function showText(val) {
    document.getElementById("out").innerHTML = val;
}
</script>
```

Dynamic HTML

Dynamic HTML means changing content, style, and structure at runtime using JavaScript.

Change Text

```
<p id="t">Old Text</p>
<button onclick="change()">Change</button>
<script>
function change() {
    document.getElementById("t").innerHTML = "New Text";
}
</script>
```

Change Color

```
<p id="c">Color Me</p>
<button onclick="color()">Red</button>

<script>
function color() {
  document.getElementById("c").style.color = "red";
}
</script>
```

Change Visibility

```
<p id="p">Hide Me</p>
<button onclick="hide()">Hide</button>

<script>
function hide() {
   document.getElementById("p").style.display = "none";
}
</script>
```

Password Strength Checker

```
Password:
<input type="password" id="pwd" onkeyup="checkStrength()">
<p id="strength"></p>

<script>
function checkStrength() {
    let pwd = document.getElementById("pwd").value;
    let strength = "Weak";
    if (pwd.length >= 8 && /[A-Z]/.test(pwd) && /[0-9]/.test(pwd)) {
        strength = "Strong";
    } else if (pwd.length >= 6) {
        strength = "Medium";
    }
    document.getElementById("strength").innerHTML =
        "Password Strength: " + strength;
}
</script>
```

Show / Hide Password

Allows users to toggle password visibility for better usability.

```
Password:
<input type="password" id="pass">
<input type="checkbox" onclick="toggle()"> Show Password
<script>
function toggle() {
    let p = document.getElementById("pass");
    p.type = (p.type === "password") ? "text" : "password";
}
</script>
```

Character Counter in Textarea

Displays remaining or used characters while typing.

```
<textarea id="msg" rows="4" cols="30" onkeyup="countChar()"></textarea>
<p id="count">Characters: 0</p>
<script>
function countChar() {
    let text = document.getElementById("msg").value;
    document.getElementById("count").innerHTML =
        "Characters: " + text.length;
}
</script>
```

JAVASCRIPT FUNCTIONS & ARRAYS

Functions

A function is a reusable block of code that performs a specific task.

```
<script>
function greet() {
    alert("Welcome to JavaScript");
}
greet();
</script>
```

Parameterized Functions

A function that accepts parameters (inputs).

```
<script>
function greetUser(name) {
    alert("Hello " + name);
}
greetUser(“Amit");
</script>
```

Return Values

Functions can return a value using return.

```
<script>
function add(a, b) {
    return a + b;
}
let result = add(5, 3);
console.log(result);
</script>
```

Arrays

An array stores multiple values in a single variable.

```
<script>
let subjects = ["DBMS", "OS", "CN"];
console.log(subjects);
Document.write(subjects);
</script>
```

Array Methods

```
<script>
let arr = ["A", "B"];
arr.push("C");//push() – Add at end
console.log(arr);
</script>

<script>
arr.pop();//pop() – Remove last
console.log(arr);
</script>
```

shift() – Remove first

```
<script>
arr.shift();
console.log(arr);
</script>
```

join() – Convert array to string

```
<script>
let courses = ["B.Tech", "MCA", "MBA"];
console.log(courses.join(", "));
</script>
//length – Number of elements
<script>
console.log(courses.length);
</script>
```

STRING METHODS

```
//toUpperCase()
```

Converts string to uppercase.

```
<script>
let name = "graphic era";
console.log(name.toUpperCase());
</script>
//trim() Removes extra spaces from both ends.
<script>
let text = "   hello world   ";
console.log(text.trim());
</script>
```

substring()

Extracts part of a string.

```
<script>
let str = "JavaScript";
console.log(str.substring(0, 4)); // Java
</script>
```

Page Redirection

Page redirection is used to navigate the user to another page automatically or after an event using JavaScript.

```
<button onclick="redirect()">Go to Google</button>

<script>
function redirect() {
   window.location.href = "https://www.google.com";
}
</script>
```

Form Reset using JavaScript

Form reset clears all input fields and restores default values.

```
<form id="myForm">
    Name: <input type="text"><br><br>
    Email: <input type="email"><br><br>

    <button type="button" onclick="resetForm()">Reset</button>
</form>

<script>
function resetForm() {
    document.getElementById("myForm").reset();
}
</script>
```

JavaScript Objects & Classes

JavaScript Objects

- A JavaScript object is a real-world entity representation that stores related data (properties) and behaviour (methods) in the form of key–value pairs.
- Objects are used to model:
- Student, Employee, Product, User, etc.

Key Characteristics

- Properties → variables inside object
- Methods → functions inside object
- Accessed using **dot (.) operator**

```
<script>
let student = {
    name: “Amit",
    roll: 101,
    course: "B.Tech",
    display: function () {
        return this.name + " - " + this.course;
    }
};

console.log(student.name);
console.log(student.display());
</script>
```

Ways to Create Objects

- Using Object Literal

```
let obj = { key: value };
```

- Using Constructor Function

```
<script>
function Student(name, roll) {
   this.name = name;
   this.roll = roll;
}

let s1 = new Student("Amit", 102);
</script>
```

JavaScript Classes

A class is a blueprint or template used to create multiple objects with similar properties and methods.

- class keyword
- constructor() initializes object
- this refers to current object
- Methods defined without function keyword

```
<script>
class Student {
    constructor(name, course) {
        this.name = name;
        this.course = course;
    }
    info() {
        return this.name + " studies " + this.course;
    }
}
let st1 = new Student("Neha", "MCA");
console.log(st1.info());
</script>
```

Class Inheritance

```
<script>
class Person {
    constructor(name) {
        this.name = name;
    }
    greet() {
        return "Hello " + this.name;
    }
}
class Teacher extends Person {
    constructor(name, subject) {
        super(name);
        this.subject = subject;
    }

    details() {
        return this.greet() + ", Subject: " + this.subject;
    }
}
let t1 = new Teacher("Dr. Singh", "DBMS");
console.log(t1.details());
</script>
```

Full Stack vs MERN Stack

Full Stack (broad concept)

A full stack developer works on both:

- Frontend (UI): HTML, CSS, JavaScript + frameworks (React/Angular/Vue)
- Backend (server + APIs): Node.js / Java / Python / .NET / PHP, etc.
- Database: MySQL / PostgreSQL / MongoDB / Oracle, etc.
- So Full Stack = any combination of frontend + backend + database + deployment.

Examples of full stack combos

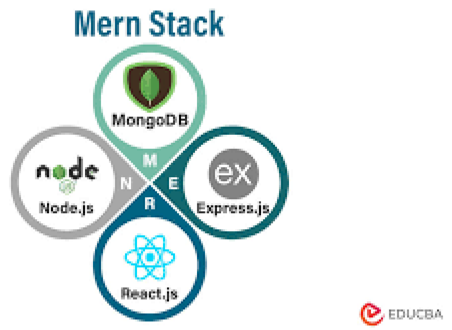
- React + Java Spring Boot + MySQL
- Angular + .NET + SQL Server
- Vue + Django + PostgreSQL
- React + Node + MongoDB (this one is MERN)

MERN Stack

MERN is a specific full-stack technology stack:

- **M** = MongoDB (database)
- **E** = Express.js (backend framework on Node)
- **R** = React (frontend)
- **N** = Node.js (runtime/backend)

So MERN ⊂ Full Stack (MERN is a subset of full stack).



Key differences

- **Scope**
  - Full Stack: wide, many tech options
  - MERN: fixed set (MongoDB + Express + React + Node)
- **Flexibility**
  - Full Stack: more flexible (you can mix any tools)
  - MERN: less flexible but very consistent as a package
- **Learning path**
  - Full Stack: can be broader depending on chosen tech
  - MERN: smoother learning path (all JavaScript-based)
- **Best for**
  - Full Stack: enterprise stacks, varied projects
  - MERN: modern web apps, startups, rapid development, JS-heavy teams

Regular Expression

A Regular Expression (RegEx) is a sequence of characters that defines a search pattern.
In JavaScript, regular expressions are mainly used for string validation, pattern matching, searching, and replacing text.

<u>RegEx in JavaScript</u>

In JavaScript, a regular expression can be created using:

Literal notation: /pattern/

RegExp object: new RegExp("pattern")

The most commonly used method is:

pattern.test(string)

Explanation of Password RegEx

```
/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&]).{8,}$/
```

Meaning:

```
^ → Start of string
(?=.*[a-z]) → At least one lowercase letter
(?=.*[A-Z]) → At least one uppercase letter
(?=.*\d) → At least one digit
(?=.*[@$!%*?&]) → At least one special character
{8,} → Minimum 8 characters
$ → End of string
```

Password At least 6 characters, letters and numbers:

```
let simplePasswordPattern = /^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{6,}$/;
let pass = "abc123";
console.log(simplePasswordPattern.test(pass)); // true
```

URL Validation

```
let urlPattern = /^(https?:\/\/)?([\w\-])+\.{1}[a-zA-Z]{2,}(\/\S*)?$/;
let url = "https://www.geu.ac.in";
console.log(urlPattern.test(url)); // true
```

Date Format (DD/MM/YYYY)

```
let datePattern = /^(0[1-9]|[12][0-9]|3[01])\/(0[1-9]|1[0-2])\/\d{4}$/;
let date = "29/01/2026";
console.log(datePattern.test(date)); // true
```

Only Numbers

```
let numberPattern = /^\d+$/;
let value = "12345";
console.log(numberPattern.test(value)); // true
```

PIN Code – 6 Digits

```
let pinPattern = /^\d{6}$/;
let pincode = "248001";
console.log(pinPattern.test(pincode)); // true
```

Email Address Validation

```
let emailPattern = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
let email = "user@example.com";
console.log(emailPattern.test(email)); // true
```

Password Validation

```
let passwordPattern =/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-
z\d@$!%*?&]{8,}$/;
let password = "Admin@123";
console.log(passwordPattern.test(password)); // true
```

JSON

JSON (JavaScript Object Notation) is a lightweight, text-based data interchange format used to store and exchange data between a client (browser/JavaScript) and a server.

Although it is derived from JavaScript object syntax, JSON is language-independent, meaning it works with Java, Python, PHP, C#, etc.

<u>JSON vs JavaScript</u>

JavaScript Object	JSON
{ name: "Amit", age: 40 }	{ "name": "Amit", "age": 40 }
Keys can be unquoted	Keys must be in double quotes
Can store functions	✕ No functions
Used inside JS	Used for data exchange

Characteristics of JSON

- Text-based format: Written in plain text, making it platform-independent.
- Language-independent: Although derived from JavaScript syntax, JSON is supported by almost all programming languages.
- Lightweight: Uses minimal syntax compared to XML, resulting in faster data transmission.
- Structured data representation: Organizes data in a clear key–value format.

JSON Data Structure

JSON represents data using two main structures:

<u>Objects</u>

- Unordered collections of key–value pairs
- Keys are strings; values can be strings, numbers, objects, arrays, booleans, or null

<u>Arrays</u>

- Ordered lists of values
- Values can be of any valid JSON data type

JSON Data Types

- String – Text data enclosed in double quotes
- Number – Integer or floating-point values
- Boolean – true or false
- Array – Ordered collection of values
- Object – Collection of key–value pairs
- Null – Represents an empty or unknown value

Why JSON is Used

- Data exchange between client and server (e.g., web applications, APIs)
- Configuration files and data storage
- Interoperability between different systems and platforms
- Efficient alternative to XML due to simpler syntax and smaller size

JSON vs XML

Feature	JSON	XML
Readability	Easy	Verbose
Data Size	Compact	Larger
Parsing Speed	Fast	Slower
Data Structure	Key–Value	Tag-based

JSON is a standard data representation format designed for efficient data exchange. Its simplicity, readability, and cross-platform support make it the backbone of modern web services, APIs, and client-server communication.

but No Support for comments, limited data types and Not suitable for complex document markup.

Standard Format for Data Exchange

- JSON is the default format for APIs
- Used in AJAX, Fetch API, REST APIs
- Almost all modern web applications use JSON

```
{
 "id": 101,
 "name": "Laptop",
 "price": 55000
}
```

Easy to Convert Between JSON and JavaScript Objects

JavaScript provides built-in methods:

JSON.parse() → JSON string ➝ JS object

JSON.stringify() → JS object ➝ JSON string

JSON vs JavaScript Object

JavaScript Object//JS Object → used for **programming logic**

```
let person = {
  name: “Amit",
  age: 35,
  department: "CSE"
};
```

JSON //JSON → used for **data transfer**

```
{
  "name": “Amit",
  "age": 35,
  "department": "CSE"
}
```

Converting JSON to JavaScript Object

```
let jsonData = '{"name":“Amit","age":35}';
let obj = JSON.parse(jsonData);
console.log(obj.name); // Amit
console.log(obj.age);  // 35
```

JSON.parse() is used to convert a JSON-formatted string into a JavaScript object, so that the data can be accessed, manipulated, and processed within a JavaScript program.

SON data received from external sources (server, API, file, localStorage) is always in string format.

JavaScript cannot directly work with JSON strings as objects—so we use JSON.parse().

Converting JavaScript Object to JSON

```
let student = {
  roll: 101,
  name: "Amit",
  branch: "CSE"
};
let jsonString = JSON.stringify(student);
console.log(jsonString);
```

//JSON.stringify() is used to convert a JavaScript object (or array) into a JSON-formatted string, so that it can be stored, transmitted, or sent over a network.

JSON with Array

```
{
  "students": [
    { "roll": 1, "name": "Amit" },
    { "roll": 2, "name": "Riya" },
    { "roll": 3, "name": "Neha" }
  ]
}
```

Accessing in JavaScript

```
let data = {
  students: [
    { roll: 1, name: "Amit" },
    { roll: 2, name: "Riya" },
    { roll: 3, name: "Neha" }
  ]
};
console.log(data.students[1].name); // Riya
```

JSON Example with HTML

```
<!DOCTYPE html><html><body>
<h3>Student Details</h3>
<p id="output"></p>
<script>
let jsonData = '{"name":“Amit","designation":"Professor","dept":"CSE"}';
let obj = JSON.parse(jsonData);
document.getElementById("output").innerHTML =
  "Name: " + obj.name + "<br>" +
  "Designation: " + obj.designation + "<br>" +
  "Department: " + obj.dept;
</script></body></html>
```

Practical Example: Student Registration + Save to JSON + Read back

```
!DOCTYPE html>
<html>
<head>
  <title>JSON Practical Example</title>
  <style>
    body { font-family: Arial; margin: 20px; }
    input, button { padding: 8px; margin: 5px; }
    table { border-collapse: collapse; width: 60%; margin-top: 10px; }
    th, td { border: 1px solid #999; padding: 8px; text-align: left; }
  </style>
</head>
<body>
```

```
<h2>Student Registration (JSON Practical)</h2>
<input id="name" placeholder="Enter Name">
<input id="roll" placeholder="Enter Roll No">
<input id="dept" placeholder="Enter Department">
<button onclick="saveStudent()">Save Student</button>
<button onclick="loadStudents()">Load Students</button>
<h3>Saved Students</h3>
<table>  <thead>
    <tr>  <th>Roll</th>
      <th>Name</th>
      <th>Department</th> </tr>
  </thead>
  <tbody id="tableBody"></tbody>
</table>
```

```
<script>
function saveStudent() {
  // 1) Read values from inputs
  let name = document.getElementById("name").value.trim();
  let roll = document.getElementById("roll").value.trim();
  let dept = document.getElementById("dept").value.trim();
  if (!name || !roll || !dept) {
    alert("Please fill all fields!");
    return;
  }
  // 2) Create a JavaScript object (NOT JSON yet)
  let studentObj = {
    roll: roll,
    name: name,
    dept: dept
  };
```

```
// 3) Get existing students from localStorage (it is JSON string)
 let oldData = localStorage.getItem("students"); // string OR null

 // 4) Convert JSON string to JS array
 let studentsArray = oldData ? JSON.parse(oldData) : [];

 // 5) Add new student object into array
 studentsArray.push(studentObj);

 // 6) Convert JS array -> JSON string (because storage needs string)
 localStorage.setItem("students", JSON.stringify(studentsArray));
 // clear input
  document.getElementById("name").value = "";
  document.getElementById("roll").value = "";
  document.getElementById("dept").value = "";

  alert("Student saved in JSON format!");
  loadStudents(); // refresh table
}
```

```
function loadStudents() {
  let data = localStorage.getItem("students"); // JSON string

  // If no data, show empty
  if (!data) {
    document.getElementById("tableBody").innerHTML = "<tr><td
colspan='3'>No students found</td></tr>";
    return;
  }
// Convert JSON string back into JS array of objects
 let studentsArray = JSON.parse(data);
 let rows = "";
 studentsArray.forEach(s => {
   rows += `
     <tr>
       <td>${s.roll}</td>
       <td>${s.name}</td>
       <td>${s.dept}</td>
     </tr>
   `;
 });
 document.getElementById("tableBody").innerHTML = rows;
}
```

```
// Auto-load table on page refresh
loadStudents();
</script>
</body>
</html>
```

In this program:

- We take user input and create a JavaScript object.
- We store data in localStorage, but localStorage can store only STRING.
- So we convert object/array into JSON string using JSON.stringify()
- When reading back, we convert JSON string into object using JSON.parse()