

# WAS CAE1

1. Write a HTML code to display form with text fields, check boxes and radio buttons, password field and submit button

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>HTML Form</title>
7  </head>
8  <body>
9      <h2>Registration Form</h2>
10     <form action="#" method="post">
11         <label for="name">Name:</label>
12         <input type="text" id="name" name="name" required><br><br>
13
14         <label for="email">Email:</label>
15         <input type="email" id="email" name="email" required><br><br>
16
17         <label for="password">Password:</label>
18         <input type="password" id="password" name="password" required><br><br>
19
20         <p>Gender:</p>
21         <input type="radio" id="male" name="gender" value="male">
22         <label for="male">Male</label>
23         <input type="radio" id="female" name="gender" value="female">
24         <label for="female">Female</label>
25         <input type="radio" id="other" name="gender" value="other">
26         <label for="other">Other</label><br><br>
27
28         <p>Interests:</p>
29         <input type="checkbox" id="sports" name="interests" value="sports">
30         <label for="sports">Sports</label>
31         <input type="checkbox" id="music" name="interests" value="music">
32         <label for="music">Music</label>
33         <input type="checkbox" id="reading" name="interests" value="reading">
34         <label for="reading">Reading</label><br><br>
35
36         <input type="submit" value="Submit">
37     </form>
38 </body>
39 </html>
40
```

This form includes:

- A text field for name
- An email field
- A password field
- Radio buttons for gender selection
- Checkboxes for interests selection
- A submit button

## 2. Write and explain table tags in HTML

### a) Cell Padding and Spacing

### b) Borders

### c) Formatting content in Table cells

### d) Nested Tables

### a) Cell Padding and Spacing

1. **Cell Padding** (`cellpadding`) adds space **inside** a cell, between content and border.
2. **Cell Spacing** (`cellspacing`) adds space **between** table cells.
3. Using `cellpadding="10"` makes cells more readable by increasing inner spacing.
4. `cellspacing="5"` creates a gap between adjacent cells, improving layout clarity.

```
<table border="1" cellpadding="10" cellspacing="5">
  <tr>
    <td>Cell 1</td>
    <td>Cell 2</td>
  </tr>
</table>
```

### b) Borders

1. The **border** attribute defines the width of table and cell borders.
2. **CSS styles** like `border: solid 2px black;` allow custom border designs.
3. The `border-collapse: collapse;` property merges adjacent borders for a clean look.
4. Different border styles (`dotted`, `dashed`, `double`) can enhance table presentation.

```
<table style="border: 2px solid black; border-collapse: collapse;">
  <tr>
    <td style="border: 1px solid black;">Cell 1</td>
    <td style="border: 1px solid black;">Cell 2</td>
  </tr>
</table>
```

### c) Formatting Content in Table Cells

1. **Text alignment** (`text-align: left/center/right;`) adjusts cell text placement.
2. **Font styling** (`font-weight: bold; font-style: italic;`) customizes text appearance.
3. **Background color** (`background-color: lightblue;`) highlights specific cells.
4. **Padding & margins** improve spacing within table cells for better readability.

```
<table border="1">
  <tr>
    <td style="text-align: center; font-weight: bold;">Bold Centered Text</td>
    <td style="background-color: lightgray;">Gray Background</td>
  </tr>
</table>
```

#### d) Nested Tables

1. A **nested table** is a table placed **inside another table's cell**.
2. Used for **complex layouts**, such as forms, invoices, or dashboards.
3. Improves **structuring of data** by grouping related content visually.
4. Can be styled separately using **CSS** to differentiate nested content.

```
<table border="1">
  <tr>
    <td>Main Table Cell</td>
    <td>
      <table border="1">
        <tr>
          <td>Nested Cell 1</td>
          <td>Nested Cell 2</td>
        </tr>
      </table>
    </td>
  </tr>
</table>
```

## 4.Types of CSS and Their Explanation with Examples

CSS (Cascading Style Sheets) is used to style HTML elements. There are **three main types of CSS**:

---

## 1. Inline CSS

### Explanation:

- CSS is applied directly to an HTML element using the `style` attribute.
- It affects only that specific element.
- Useful for quick styling but **not recommended** for large projects due to poor maintainability.

### Example:

```
html
CopyEdit
<!DOCTYPE html>

<html>

<head>

<title>Inline CSS Example</title>

</head>

<body>

<p style="color: blue; font-size: 18px;">

This is an example of Inline CSS.

</p>

</body>

</html>
```

### ✅ Pros:

- Quick to implement.
- Overrides other CSS styles easily.

### ❌ Cons:

- Not reusable.
- Difficult to manage in large projects.

---

## 2. Internal CSS (Embedded CSS)

### Explanation:

- CSS is written inside the `<style>` tag within the `<head>` section of the HTML file.
- It applies styles **only** to that specific HTML page.
- Useful when styling a **single** webpage without affecting other pages.

### Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Internal CSS Example</title>
  <style>
    p {
      color: green;
      font-size: 20px;
      font-weight: bold;
    }
  </style>
</head>
<body>
  <p>This is an example of Internal CSS.</p>
</body>
</html>
```

### ✅ Pros:

- Affects multiple elements within the same page.

- More structured than inline CSS.

#### ✗ Cons:

- Not reusable across multiple pages.
  - Can make the HTML file bulky if too many styles are included.
- 

## 3. External CSS

### Explanation:

- CSS is written in a separate `.css` file and linked using the `<link>` tag in the HTML `<head>`.
- The **best approach** for large websites as styles are applied consistently across multiple pages.
- Improves maintainability and **faster page loading** due to browser caching.

### Example:

#### CSS File (styles.css)

```
p {  
  color: red;  
  font-size: 22px;  
  text-align: center;  
}
```

#### HTML File (index.html)

```
<!DOCTYPE html>  
  
<html>  
  
  <head>  
  
    <title>External CSS Example</title>  
  
    <link rel="stylesheet" type="text/css" href="styles.css">  
  
  </head>  
  
  <body>  
  
    <p>This is an example of External CSS.</p>
```

```
</body>

</html>
```

✔ **Pros:**

- Can be applied to **multiple pages**, making it reusable.
- Keeps HTML code **clean and organized**.

✖ **Cons:**

- Requires an extra file.
- A website might **not load properly** if the CSS file is missing.

## Comparison Table of CSS Types

Type	Placement	Scope	Usage	Best For
Inline CSS	Inside HTML tag using <code>style</code>	Affects only one element	Quick fixes	Small changes
Internal CSS	Inside <code>&lt;style&gt;</code> in <code>&lt;head&gt;</code>	Affects only the current page	Page-specific styles	Single-page websites
External CSS	In a separate <code>.css</code> file	Can be applied to multiple pages	Large-scale projects	Websites with multiple pages

**Conclusion:**

- Use **Inline CSS** for quick fixes but **avoid it** in large projects.
- Use **Internal CSS** for a single webpage when styles don’t need to be shared.
- Use **External CSS** for **best performance and maintainability** in large website

**5.Explain WWW Vs Internet**

Feature	WWW (World Wide Web)	Internet
1. Definition	The World Wide Web is a collection of websites and web pages accessed via the Internet.	The Internet is a global network of interconnected computers that enables communication and data transfer.
2. Function	Provides access to information using web browsers.	Connects devices and allows data transfer across networks.
3. Components	Consists of web pages, websites, hyperlinks, and multimedia content.	Includes servers, routers, data centers, satellites, and other infrastructure.
4. Protocols Used	Uses HTTP/HTTPS for communication.	Uses multiple protocols like TCP/IP, FTP, SMTP, and more.
5. Dependency	WWW depends on the Internet to function.	The Internet can function without the WWW (e.g., emails, file transfers, VoIP).
6. Accessibility	Accessed via web browsers like Chrome, Firefox, and Edge.	Accessed through various applications like browsers, email clients, and messaging apps.
7. Example Services	Websites like Google, Facebook, Wikipedia.	Services like email (Gmail), file sharing, video streaming, and messaging apps.
8. Inventor	Created by Tim Berners-Lee in 1989.	Originated from ARPANET, developed by the U.S. Department of Defense in the 1960s.

## 8.HTTP Request vs. HTTP Response (5 Points Each)

### 1. HTTP Request (Client → Server)

1. **Initiated by the client** (browser, API, or application) to request data from the server.
2. **Consists of a request line** (method, URL, HTTP version). Example: `GET /index.html HTTP/1.1`.
3. **Includes headers** (like `User-Agent`, `Accept`, `Content-Type`) to provide metadata about the request.
4. **May contain a request body** (in `POST`, `PUT` requests) to send form data or JSON payloads.
5. **Uses HTTP methods** such as `GET`, `POST`, `PUT`, `DELETE`, `PATCH`,

`GET /index.html HTTP/1.1`

`Host: www.example.com`

`User-Agent: Mozilla/5.0`

### 2. HTTP Response (Server → Client)

1. **Sent by the server** after processing the request.
2. **Includes a status code** (e.g., `200 OK`, `404 Not Found`, `500 Internal Server Error`) indicating the result.



3. **Contains headers** (like `Content-Type`, `Date`, `Server`) to describe the response.
4. **May have a response body** containing HTML, JSON, XML, or other data formats.
5. **Helps the browser render content** or APIs process the received data.

HTTP/1.1 200 OK

Content-Type: text/html

Date: Mon, 14 Feb 2025 12:00:00 GMT

<html>

<body>

<h1>Welcome to Example.com</h1>

</body>

</html>

## 13.Client-Server Architecture

### Introduction:

Client-Server Architecture is a network model where multiple clients request and receive services from a central server. It is widely used in web applications, database management, and network communication.

---

### 1. Components of Client-Server Architecture

Component	Description
<b>Client</b>	A device (computer, smartphone, etc.) or application that requests services from the server. Example: Web browsers, mobile apps.
<b>Server</b>	A powerful system that processes client requests and provides data or services. Example: Web servers, database servers.
<b>Network</b>	The medium (Wi-Fi, LAN, Internet) that connects clients and servers for communication.

---

### 2. Working of Client-Server Architecture

1. **Client Sends a Request** – The client (e.g., browser) requests a resource from the server.
2. **Server Processes the Request** – The server verifies, retrieves, and processes the requested data.

3. **Server Sends a Response** – The processed data is sent back to the client in a suitable format (HTML, JSON, etc.).
4. **Client Displays the Response** – The received data is rendered or used by the client application.

**Example:**

- A user visits `www.example.com`.
  - The **client (browser)** sends a `GET` request to the **server**.
  - The **server** fetches and returns the requested webpage.
  - The **client (browser)** displays the webpage.
- 

### 3. Types of Client-Server Architecture

Type	Description
1-Tier	Client and server are on the same system. Example: Standalone applications.
2-Tier	The client interacts directly with the server. Example: Simple database applications.
3-Tier	Uses an intermediate layer (e.g., Application Server) between client and server. Example: Web applications with a database.
N-Tier (Multi-Tier)	Uses multiple servers for better scalability. Example: Cloud computing, large-scale web apps.

---

### 4. Advantages of Client-Server Architecture

1. **Centralized Management** – The server manages data, ensuring security and control.
  2. **Scalability** – More clients can be added without affecting performance significantly.
  3. **Efficient Data Sharing** – Clients can access shared resources like databases and files.
  4. **Improved Security** – Servers can enforce authentication and access control policies.
  5. **Faster Processing** – Dedicated servers provide quick responses to multiple clients.
- 

### 5. Disadvantages of Client-Server Architecture

1. **Server Overload** – A high number of client requests can slow down the server.

2. **Single Point of Failure** – If the server fails, all clients lose access to services.
3. **Network Dependency** – Clients must have a stable connection to interact with the server.
4. **Higher Cost** – Setting up and maintaining servers requires significant resources.
5. **Security Risks** – Hackers can target centralized servers, leading to data breaches.

## Difference Between HTTP and HTTPS

Feature	HTTP (HyperText Transfer Protocol)	HTTPS (HyperText Transfer Protocol Secure)
1. Security	Not secure; data is transmitted in plain text.	Secure; data is encrypted using SSL/TLS.
2. Encryption	No encryption, making data vulnerable to hackers.	Uses encryption to protect sensitive data.
3. Protocol Used	Uses HTTP (port 80).	Uses HTTP over SSL/TLS (port 443).
4. Data Integrity	No protection against data tampering.	Ensures data integrity, preventing modifications.
5. Trust & Authentication	No authentication; attackers can perform man-in-the-middle attacks.	Uses SSL/TLS certificates for authentication, making it more trustworthy.
6. Usage	Suitable for non-sensitive data (e.g., blogs, public websites).	Used for secure transactions (e.g., banking, e-commerce, login pages).

## Why is HTTPS More Secure than HTTP?

1. **Data Encryption:** HTTPS encrypts the data using SSL/TLS, preventing hackers from intercepting sensitive information.
2. **Authentication:** HTTPS uses digital certificates (SSL/TLS) to verify the server's identity, preventing phishing attacks.
3. **Data Integrity:** HTTPS ensures that data sent between the client and server is not altered or tampered with.
4. **Protection Against Man-in-the-Middle Attacks:** Encryption prevents unauthorized interception and modification of data.
5. **Better SEO Ranking:** Search engines like Google prioritize HTTPS websites, improving visibility.
6. **User Trust & Browser Warnings:** Browsers mark HTTP sites as “Not Secure”, discouraging users from entering personal data.

## Role of TCP (Transmission Control Protocol) in Ensuring Reliable Data Transmission

TCP is a core protocol of the Internet that ensures reliable, ordered, and error-checked delivery of data across networks. It plays a crucial role in web communication, file transfers, and online applications.

---

## 1. Connection-Oriented Communication

- TCP establishes a reliable **connection** between sender and receiver before data transmission.
  - It follows a **three-way handshake** (SYN, SYN-ACK, ACK) to ensure both parties are ready for communication.
- 

## 2. Reliable Data Delivery

- TCP **guarantees** that all sent data reaches the destination **without loss or corruption**.
  - If data packets are lost, TCP **automatically retransmits** them.
- 

## 3. Data Segmentation and Reassembly

- Large data streams are broken into smaller **packets** for transmission.
  - TCP ensures that these packets are **reassembled in the correct order** at the destination.
- 

## 4. Error Detection and Correction

- TCP uses **checksums** to verify data integrity.
  - If errors are detected, the receiver **requests retransmission** of corrupted packets.
- 

## 5. Flow Control (Avoids Overloading Receiver)

- TCP prevents the sender from overwhelming the receiver with too much data.

- Uses a **sliding window mechanism** to regulate data flow based on the receiver's processing speed.
- 

## 6. Congestion Control (Avoids Network Overload)

- TCP adjusts the data transmission rate based on network conditions.
- Implements algorithms like **Slow Start, Congestion Avoidance, and Fast Retransmit** to manage traffic.

## 8. Multiplexing (Handling Multiple Connections)

- TCP allows multiple applications (web browsing, video streaming, email) to use the network simultaneously.
  - Uses **port numbers** to direct data to the correct application on the receiving device.
- 

## 9. Ensuring End-to-End Communication

- TCP enables direct communication between two devices **irrespective of network type**.
- It ensures that data is transmitted securely and completely between client and server.

## 20.Difference Between XML and HTML (8 Points)

Feature	XML (Extensible Markup Language)	HTML (HyperText Markup Language)
1. Purpose	Designed to store and transport data.	Designed to display data on web pages.
2. Structure	User-defined tags, making it flexible for data representation.	Predefined tags used for web page formatting.
3. Data Handling	Focuses on <b>storing, organizing, and transporting data</b> .	Focuses on <b>displaying data with formatting</b> .
4. Tag Usage	Custom tags can be created by users.	Uses predefined tags like <p>, <h1>, <table>.
5. Strictness	Case-sensitive and follows strict syntax rules.	Not case-sensitive and is more lenient with syntax errors.
6. Nesting Rules	Tags must be <b>properly nested and closed</b> .	Some tags, like   and <img>, do not need closing.
7. Data Storage	Used for data exchange between applications (e.g., Web APIs, configuration files).	Used for structuring web content in browsers.
8. Formatting	No predefined style; needs external processing to display.	Supports formatting with CSS for styling web pages.

---

## How is XML Used to Represent Data?

XML is widely used to store and transport data in a structured format. It provides a **human-readable** and **machine-readable** way to represent hierarchical data.

### Example of XML Representing Data

```
<students>

  <student>

    <name>John Doe</name>

    <age>22</age>

    <course>Computer Science</course>

  </student>

  <student>

    <name>Jane Smith</name>

    <age>21</age>

    <course>Information Technology</course>

  </student>

</students>
```

### Explanation:

- `<students>` is the **root element** that holds multiple `<student>` elements.
- Each `<student>` has sub-elements like `<name>`, `<age>`, and `<course>` for structured data representation.
- XML is **self-descriptive**, meaning the tags clearly define the data they hold.

### Where XML is Used?

1. **Web Services & APIs** – Used in **SOAP-based APIs** for data exchange.
2. **Configuration Files** – Many applications store settings in XML (e.g., Android `.xml` files).
3. **Data Storage & Transfer** – Used in databases and **RSS feeds**.
4. **Document Formatting** – Used in **SVG graphics** and **XHTML** for structured documents.

# Operators, Functions, and Arrays in JavaScript

---

## 1. Operators in JavaScript

Operators perform operations on variables and values. JavaScript supports different types of operators:

### A. Arithmetic Operators

Used for mathematical calculations.

Operator	Description	Example
+	Addition	5 + 3 // 8
-	Subtraction	10 - 4 // 6
*	Multiplication	6 * 2 // 12
/	Division	8 / 2 // 4
%	Modulus (Remainder)	10 % 3 // 1
**	Exponentiation	2 ** 3 // 8

### B. Comparison Operators

Used to compare values and return `true` or `false`.

Operator	Description	Example
==	Equal to	5 == '5' // true
===	Strict equal (checks type too)	5 === '5' // false
!=	Not equal	10 != 5 // true
>	Greater than	7 > 3 // true
<	Less than	5 < 8 // true
>=	Greater than or equal	6 >= 6 // true
<=	Less than or equal	4 <= 2 // false

### C. Logical Operators

Used to perform logical operations.

Operator	Description	Example
&&	Logical AND	(5 > 3 && 10 > 5) // true
^		^
!	Logical NOT	!(10 > 5) // false

### D. Assignment Operators

Used to assign values to variables.

Operator	Description	Example
=	Assign	let a = 10
+=	Add and assign	a += 5 // a = 15
-=	Subtract and assign	a -= 3 // a = 12
*=	Multiply and assign	a *= 2 // a = 24

**Example of Operators in JavaScript:**

```
let x = 10;

let y = 5;

console.log(x + y); // Output: 15

console.log(x > y); // Output: true

console.log(x === "10"); // Output: false

console.log(x > 5 && y < 10); // Output: true
```

---

## 2. Functions in JavaScript

A function is a reusable block of code designed to perform a specific task.

**A. Function Declaration**

```
function greet(name) {

  return "Hello, " + name;}

console.log(greet("John")); // Output: Hello, John
```

**B. Function Expression**

```
const add = function(a, b){

return a + b;};

console.log(add(3, 4)); // Output: 7
```

**C. Arrow Function (ES6)**

```
const multiply = (x, y) => x * y;

console.log(multiply(5, 2)); // Output: 10
```



## D. Function with Default Parameters

```
function greetUser(name = "Guest") {  
  return "Hello, " + name;}  
  
console.log(greetUser()); // Output: Hello, Guest
```

---

# 3. Arrays in JavaScript

An array is a collection of elements stored in a single variable.

## A. Creating an Array

```
let fruits = ["Apple", "Banana", "Cherry"];  
  
console.log(fruits); // Output: ["Apple", "Banana", "Cherry"]
```

## B. Accessing Array Elements

```
javascript  
CopyEdit  
  
console.log(fruits[0]); // Output: Appleconsole.log(fruits[2]); // Output: Cherry
```

## C. Adding Elements to an Array

```
javascript  
CopyEdit  
  
fruits.push("Orange"); // Adds to the end  
  
console.log(fruits); // Output: ["Apple", "Banana", "Cherry", "Orange"]
```

## D. Removing Elements from an Array

```
javascript  
CopyEdit  
  
fruits.pop(); // Removes last element  
  
console.log(fruits); // Output: ["Apple", "Banana", "Cherry"]
```

## E. Looping Through an Array

javascript

CopyEdit

```
for (let i = 0; i < fruits.length; i++) {  
  console.log(fruits[i]); }  

```

## F. Array Methods

Method	Description	Example
push()	Adds an item to the end	fruits.push("Mango")
pop()	Removes the last item	fruits.pop()
shift()	Removes the first item	fruits.shift()
unshift()	Adds an item to the beginning	fruits.unshift("Grapes")
length	Returns array length	fruits.length
indexOf()	Finds index of an element	fruits.indexOf("Banana")

---

## Conclusion

- **Operators** perform mathematical and logical operations.
- **Functions** help in writing reusable blocks of code.
- **Arrays** store multiple values efficiently.

## Alert, Confirmation, and Prompt Box in JavaScript

JavaScript provides three types of popup dialog boxes to interact with users:

1. **Alert Box** – Displays a message with an "OK" button.
  2. **Confirmation Box** – Asks users to confirm an action with "OK" and "Cancel" buttons.
  3. **Prompt Box** – Allows users to enter input in a text field.
- 

### 1. Alert Box (`alert()`)

- Used to **display a simple message** to the user.
- It does **not return any value**.

- Typically used for **warnings, notifications, or important messages**.

## Syntax:


```
alert("This is an alert box!");
```

## Example:

```
javascript
CopyEdit
function showAlert() {
    alert("Warning! You are about to delete a file.");
}
showAlert();
```

## Output:

A popup appears with the message:

 "Warning! You are about to delete a file."

(With an "OK" button)

---

## 2. Confirmation Box (`confirm()`)

- Used to **confirm user actions**.
- Displays a message with **"OK"** and **"Cancel"** buttons.
- Returns **true** if the user clicks "OK" and **false** if "Cancel" is clicked.

## Syntax:

```
confirm("Are you sure?");
```

## Example:

```
function confirmDelete() {
    let userResponse = confirm("Are you sure you want to delete this?");
```

```
if (userResponse) {  
    alert("File deleted successfully.");  
} else {  
    alert("File deletion canceled.");  
}  
  
confirmDelete();
```

## Output:

A popup appears with:

🔴 "Are you sure you want to delete this?"

(With "OK" and "Cancel" buttons)

- If "OK" is clicked → Another alert appears: **"File deleted successfully."**
  - If "Cancel" is clicked → Another alert appears: **"File deletion canceled."**
- 

## 3. Prompt Box (`prompt()`)

- Used to **accept user input** through a **text field**.
- Returns the **user's input as a string** if "OK" is clicked, otherwise returns `null`.

## Syntax:

```
javascript  
CopyEdit  
prompt("Enter your name:");
```

## Example:

```
function getUser_name() {  
    let name = prompt("Enter your name:", "Guest");  
    if (name !== null && name !== "") {  
        alert("Hello, " + name + "!");  
    }  
    else {
```

```
alert("No name entered.");

}}getUserName();
```

## Output:

A popup appears with:

👉 "Enter your name:" (Text field + "OK" & "Cancel" buttons)

- If user enters "John" and clicks "OK" → Another alert appears: **"Hello, John!"**
- If user clicks "Cancel" or leaves it blank → Another alert appears: **"No name entered."**

---

## Comparison Table: Alert vs Confirmation vs Prompt

Feature	Alert Box ( <code>alert()</code> )	Confirmation Box ( <code>confirm()</code> )	Prompt Box ( <code>prompt()</code> )
Purpose	Display message to user	Ask user to confirm an action	Take input from user
Buttons	Only "OK"	"OK" and "Cancel"	"OK" and "Cancel"
Returns	Nothing ( <code>undefined</code> )	<code>true</code> (OK), <code>false</code> (Cancel)	User input (string) or <code>null</code>
Example Use	Show warnings or alerts	Confirm deletion/logout	Get user's name or age

---

## Conclusion:

- `alert()` is for **displaying messages** to users.
- `confirm()` asks for **user confirmation** before proceeding.
- `prompt()` is used to **get input** from users.

**JavaScript code to validate Name, Email, and Mobile Number in a Registration Form using regular expressions.**

---

## Features of this Validation Script:

### 1. Name Validation

- Only alphabets and spaces are allowed.
- Minimum 3 characters required.

### 2. Email Validation

- Must follow standard email format (e.g., example@domain.com).

### 3. Mobile Number Validation

- Only 10-digit numbers are allowed.
- No alphabets or special characters allowed.

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Registration Form Validation</title>

  <style>

    body { font-family: Arial, sans-serif; }

    .error { color: red; }

  </style>

</head>

<body>

  <h2>Registration Form</h2>

  <form id="registrationForm" onsubmit="return validateForm()">

    <label for="name">Name:</label>

    <input type="text" id="name" name="name">

    <span class="error" id="nameError"></span>

    <br><br>

    <label for="email">Email:</label>

    <input type="text" id="email" name="email">

    <span class="error" id="emailError"></span>

    <br><br>

    <label for="mobile">Mobile Number:</label>

    <input type="text" id="mobile" name="mobile">

    <span class="error" id="mobileError"></span>
```

```
<br><br>
```

```
<input type="submit" value="Register">
```

```
</form>
```

```
<script>
```

```
function validateForm() {
```

```
    let isValid = true;
```

```
    // Get input values
```

```
    let name = document.getElementById("name").value;
```

```
    let email = document.getElementById("email").value;
```

```
    let mobile = document.getElementById("mobile").value;
```

```
    // Regular Expressions for validation
```

```
    let namePattern = /^[A-Za-z\s]{3,}$/;
```

```
    let emailPattern = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
```

```
    let mobilePattern = /^[0-9]{10}$/;
```

```
    // Name Validation
```

```
    if (!namePattern.test(name)) {
```

```
        document.getElementById("nameError").innerText = "Enter a valid name (Min 3 letters, only  
alphabets & spaces)";
```

```
        isValid = false;
```

```
    } else {
```

```
        document.getElementById("nameError").innerText = "";
```

```
    }
```

```
    // Email Validation
```

```
    if (!emailPattern.test(email)) {
```

```
        document.getElementById("emailError").innerText = "Enter a valid email (e.g.,  
example@domain.com)";
```

```
        isValid = false;
```

```

    } else {
        document.getElementById("emailError").innerText = "";
    }

    // Mobile Number Validation
    if (!mobilePattern.test(mobile)) {
        document.getElementById("mobileError").innerText = "Enter a valid 10-digit mobile number";
        isValid = false;
    } else {
        document.getElementById("mobileError").innerText = "";
    }

    return isValid; // Prevent form submission if invalid
}
</script>

</body>
</html>

```

## How to Include JavaScript in an HTML Document (Detailed Explanation)

JavaScript can be included in an HTML document using different methods. Each method serves a specific purpose depending on the complexity and reusability of the script.

---

### 1. Inline JavaScript (Using the `onclick` Attribute)

- JavaScript can be added directly inside **HTML elements** using event attributes like `onclick`, `onmouseover`, `onchange`, etc.
- Best for **small scripts** like button clicks.

```

<!DOCTYPE html>

<html>

<head>

```



```
<title>Inline JavaScript Example</title>
</head>
<body>

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

</body>
</html>
```

#### ✅ Pros:

✓ Simple and easy for small functions.

#### ❌ Cons:

✗ Hard to maintain for large projects.

✗ Mixing HTML and JavaScript reduces readability.

---

## 2. Internal JavaScript (Inside `<script>` Tag)

- JavaScript can be written inside the `<script>` tag in the **same HTML file**, usually within the `<head>` or `<body>` section.
- Best for **small to medium-sized scripts**.

### Example (Internal JavaScript in `<script>` Tag)

```
<!DOCTYPE html>
<html>
<head>
  <title>Internal JavaScript Example</title>
  <script>
    function showMessage() {
      alert("Hello from Internal JavaScript!");
    }
  </script>
</head>
```

```
<body>
```

```
<button onclick="showMessage()">Click Me</button>
```

```
</body>
```

```
</html>
```

#### ✅ Pros:

✓ Keeps JavaScript separate from HTML elements.

#### ❌ Cons:

✗ If too much JavaScript is written inside HTML, it can make the file **bulky**.

---

## 3. External JavaScript (Using .js File)

- JavaScript is written in a **separate .js file** and linked using the `<script>` tag.
- Best for **large projects** as it improves **code organization and reusability**.

### Steps to Use External JavaScript:

#### Step 1: Create an External JavaScript File (`script.js`)

```
function showMessage() {  
    alert("Hello from External JavaScript!");  
}
```

#### Step 2 Link the JavaScript File in HTML (`index.html`)

```
<!DOCTYPE html>  
  
<html>  
  
<head>  
    <title>External JavaScript Example</title>  
    <script src="script.js"></script> <!-- Linking external JS -->  
</head>  
  
<body>  
  
    <button onclick="showMessage()">Click Me</button>
```

</body>

</html>

What is jQuery? Explain how to include jQuery in an HTML page and provide an example of its usage.

## What is jQuery?

jQuery is a **fast, small, and feature-rich JavaScript library** that simplifies **HTML document traversal, event handling, animations, and AJAX interactions** for web development.

### Key Features of jQuery:

- ✓ **Simplifies DOM Manipulation** (Easier to select and modify HTML elements)
  - ✓ **Cross-browser Compatibility** (Works across different browsers)
  - ✓ **Event Handling** (Handles user interactions like clicks, form submissions)
  - ✓ **AJAX Support** (Fetch data without reloading the page)
  - ✓ **Built-in Effects and Animations** (E.g., fade, slide, hide, show)
- 

## How to Include jQuery in an HTML Page

There are **two ways** to include jQuery:

1. **Using a CDN (Recommended)**
2. **Downloading and Hosting Locally**

### 1. Including jQuery via CDN (Recommended)

The easiest way to include jQuery is by using a **CDN (Content Delivery Network)** like Google or jQuery's official website.

Add the following line inside the `<head>` or before `</body>`:

```
<!-- jQuery from Google CDN -->
```

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
```

#### ♦ Advantage:

- ✓ Faster loading due to browser caching and CDN optimization.

---

## 2. Downloading and Using Local jQuery

1. **Download jQuery** from the official site: <https://jquery.com/download/>
2. Save the file (e.g., `jquery.min.js`) in your project folder.
3. Link it in your HTML file:

html

CopyEdit

```
<script src="js/jquery.min.js"></script>
```

### ◆ Advantage:

✓ Works **offline**, even if the internet is not available.

---