WEB PROGRAMMING –

Semantic and Non-Semantic Tags

Semantic tags are HTML tags that have meaning and clearly define the purpose of the content inside them. They describe the purpose of the content , making it easier for developers , browsers etc. to understand and interact with webpages.
semantic tags enhances the SEO score of our website.

They have specific attributes for their structure.
Semantic tags: header , nav, main, footer, aside , article , section , figure , mark etc.

Non-semantic tags do not provide any meaning about the content inside them; they only serve as generic containers. Typically used with CSS and JavaScript for layout and design. These provide flexibility for styling
Non-Semantic Tags: `<div>`, `<span>` (These can contain any type of content.)

HTML form tag –

The form tag in HTML are used to create interactive forms that allows the users to input user data and submit data to server. It is commonly used for login, registration , contact forms etc. Form tags include , form , input , label, text area (multiline) , select, options, button etc. drop-down-menu , profile upload, checkboxes, radio buttons.

Attribute	Description
`action`	Specifies the URL where form data should be sent.
`method`	Defines how data is sent (`GET` or `POST or Put`).
`enctype`	Specifies encoding type when submitting files (used with `POST`). For e.g. application/x-www-form-urlencoded , multipart/form-data (for file upload) , plain/text
`target`	Defines where to display the response (`_self`, `_blank`, etc.).
`autocomplete`	Enables or disables autocomplete (`on` or `off`).
`novalidate`	Disables form validation before submission.

METHODS IN FORM TAG –

POST – it sends the form data in the HTTP request body, it's not visible in the URL. It is secure for sensitive data.
GET – it appends the data to the URL as query parameters. Used in searching bars , filters, data retrieval .
But the main disadvantage is it is not secure for sensitive data . because your data is visible in the URL.

These are the other HTTP methods – but we cant use it directly in form . we need JavaScript Like fetch or axios . or backend framework like express.
PUT – it basically updates the existing data.
DELETE – deletes the existing data
PATCH – partially updates the existing data.

Tables in HTML –

The purpose of tables in html is used to create tables for organizing data in structured way while displaying data to users. It makes easier to read and compare data.
`<table>` – Creates a table.
`<tr>` – Table row.
`<td>` – Table data (cell).
`<th>` – Table header cell.
`<thead>` – Table header section.
`<tbody>` – Table body section.
`<tfoot>` – Table footer section.
`<caption>` – Table caption.

For eg.
```
<table>
    <thead>
      <tr>
        <th>Roll No</th>
        <th>Name</th>
        <th>Grade</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>101</td>
        <td>John Doe</td>
        <td>A</td>
      </tr>
      <tr>
        <td>102</td>
        <td>Jane Smith</td>
        <td>B</td>
      </tr>
      <tr>
        <td>103</td>
        <td>Emily Brown</td>
        <td>A</td>
      </tr>
    </tbody>
  </table>
```

CSS -

1. Inline CSS

- Applied directly to individual HTML elements using the `style` attribute.
- Useful for quick styling but not recommended for large-scale projects due to maintainability issues.

Example:

```
<p style="color: blue; font-size: 16px;">This is a paragraph.</p>
```

2. Internal CSS (Embedded CSS)

- Defined within the `<style>` tag inside the `<head>` section of an HTML document.
- Useful for applying styles to a single page without affecting other pages.

Example:

```
<head>
 <style>
  p {
   color: blue;
   font-size: 16px;
  }
 </style>
</head>
<body>
 <p>This is a paragraph.</p>
</body>
```

3. External CSS

- Styles are placed in a separate `.css` file and linked to the HTML document using the `<link>` tag.
- Recommended for large projects as it helps in reusability and maintainability.

Example:

CSS File (`styles.css`)

```
p {
 color: blue;
 font-size: 16px;
}
```

HTML File:

```
<head>
 <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
 <p>This is a paragraph.</p>
</body>
```

For best practice, **external CSS** is preferred for scalability and maintainability.

CSS Specificity & Priority

CSS applies rules based on specificity. The order of priority is:

1. **Inline Styles** → `style="color: red;"`
2. **ID Selectors** → `#id`
3. **Class Selectors, Attribute Selectors, Pseudo-classes** → `.class`, `[type="text"]`, `:hover`
4. **Element Selectors** → `h1, p, div`
5. **Universal Selector (*) & Inherited Style**

ADVANCE CSS –

1.TRANSITIONS – allows CSS properties to change smoothly over time rather than directly or abruptly. This may enhance user experience.

For eg. Transition : property duration timing-functions
Property: background-color , border-radius , width , opacity etc.
Duration : 0.5s 1s 2s
Timing-function: ease-in, ease-in-out , linear

Example: Changing background color smoothly

```
button {
  background-color: blue;
  color: white;
  padding: 10px 20px
  border: none;
  cursor: pointer;
  transition: background-color 0.5s ease-in-out;
}
button:hover {
  background-color: red;
}
```

2.TRANSFORMS -

Transforms are used to rotate, scale, translate, or skew elements.

Example: Rotating and Scaling on Hover

```
.box {
  width: 100px;
  height: 100px;
  background-color: green;
  transition: transform 0.5s;
}
```

```
.box:hover {
  transform: rotate(45deg) scale(1.2);
}
```

Explanation:

- **When hovered, the `.box` rotates by 45 degrees and scales up by 1.2 times.**

Common Transform Functions

Function	Description
`rotate(45deg)`	Rotates the element 45 degrees
`scale(1.5)`	Enlarges the element by 1.5x
`translateX(50px)`	Moves the element 50px right
`translateY(-20px)`	Moves the element 20px up
`skew(30deg, 10deg)`	Skews the element

`translateX`

- Moves an element along the X-axis (left or right).
- Positive values move the element to the right, and negative values move it to the left.

`translateY`

- Moves an element along the Y-axis (up or down).
- Positive values move the element down, and negative values move it up.

3.KEYFRAMES-

In CSS, `@keyframes` is used to create animations. **Keyframes** are used to define the stages of an animation.so You can specify different styles for each stage and browser can smoothly transition from one set of styles to another over time.

```css
.loader {
      display: flex;
      justify-content: center;
      align-items: flex-end;
      height: 100vh;
      gap: 10px;
    }
```

```css
.circle {
  width: 20px;
  height: 20px;
  background-color: #3498db;
  border-radius: 50%;
  animation: bounce 0.6s infinite ease-in-out;
}

.circle:nth-child(2) {
  animation-delay: 0.2s;
}

.circle:nth-child(3) {
  animation-delay: 0.4s;
}

@keyframes bounce {
  0%, 100% {
    transform: translateY(0);
  }
  50% {
    transform: translateY(-20px);
  }
}
```

🔷 **Explanation:**

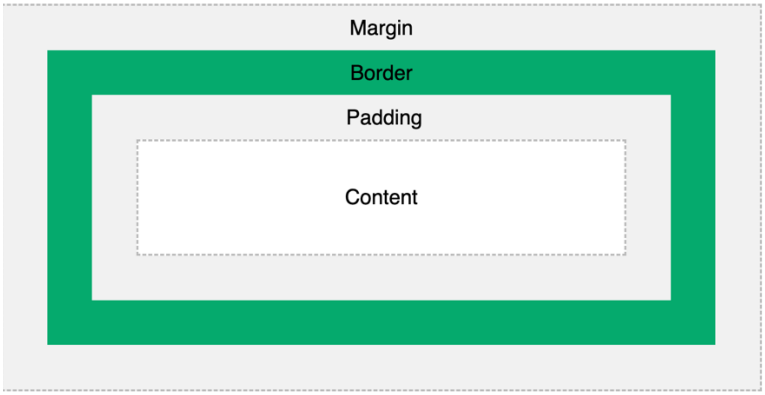
- **The `.ball` moves up (-50px) at 50%, then comes back down.**
- **The `animation` property specifies the name (bounce), duration (1s), and infinite looping.**

Property	Description
`animation-name`	The name of the `@keyframes` animation fade-in ,fadeout, slide, bounce, shake
`animation-duration`	How long the animation runs (e.g., 2s)
`animation-timing-function`	Eases movement (e.g., `ease-in-out`, `linear`)
`animation-delay`	Wait time before animation starts

`animation-iteration-count`	Number of times animation runs (`infinite` for looping)
`animation-direction`	`normal`, `reverse`, `alternate` (back & forth)

BOX MODEL IN CSS –

Margin and padding are both part of the CSS Box Model and are used to create spacing around elements, but they serve different purposes.



Content – it is the actual content area where text , images and other media appears
Padding – it is the space between the content and the border. Increases the elements visual size. The main goal is to make content more visible.
Margin – space outside the border , it is used to create the space between the element and surrounding.
Border- a line that is surrounded by content and padding.

DOM MANIPULATION

The DOM stands for the Document Object Model (DOM), which allows us to interact with the document and change its structure, style, and content. We can use the DOM to change the content and style of an HTML element by changing its properties.

1. Change the Content of an Element

You can change the content inside an HTML element using JavaScript. The two most common properties for this are innerHTML and textContent:

- **innerHTML:** Allows you to get or set the HTML content inside an element.
- **textContent:** Allows you to get or set the text content inside an element, ignoring any HTML tags.

2. Manipulate the Class Attribute

You can add, remove, or toggle classes on an element using JavaScript. This is helpful for styling or applying animations.

- **classList.add():** Adds a class to an element.
- ClassList.remove(): removes a class from element.

Example:

```
<div  id="box"  class="bg-blue"></div>

const box = document.getElementById("box");
box.classList.add("animate-bounce");
```

3. Set CSS Styles Using JavaScript

You can directly manipulate the CSS styles of an element using the style property. This allows you to dynamically change how elements appear on the page.

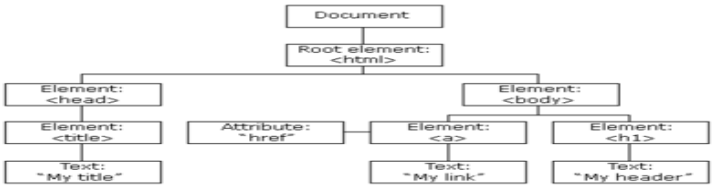
4. Create, Add, and Remove Elements

Sometimes, you need to create new elements, add them to the DOM, or remove existing ones. You can do this easily with the following methods

- **document.createElement():** Creates a new element.
- **appendChild():** Adds a new element to a parent element.
- **removeChild():** Removes a child element from a parent.

5.Document Selectors:

getElementById , by Classname , querySelector , querySelectorAll



NODE JS –

Node js is an opensource , cross platform JavaScript runtime environment that allows developers to run JavaScript code outside a web browser , typically on the server side.
NodeJS lets you to write the JavaScript to build backend services like APIs, web browsers , real-time apps, and more.

The primary features of NodeJs are –

1.Event Driven and NonBlocking I/O model :-
means multiple client requests simultaneously without waiting for each request to complete. Means file reads, DB calls , Api calls , Data fetching etc. don't block the execution.
Benefit: This model significantly improves the performance and scalability of applications, especially for I/O bound and data intensive tasks.

2.Single Threaded with Event Loop –
Even If nodejs is a single threaded , still it can handle thousands of concurrent requests efficiently . by offloading operations to the system kernel whenever possible. Since most modern kernels are multi-threaded, they can handle multiple operations executing in the background.
Example: When a request is made to a Node.js server, the event loop handles the request and delegates the actual I/O operations to the underlying system. Once the I/O operation is complete, the event loop resumes processing the request.

3.NPM (Node Package Manager)-
NPM is the default package manager for Node.js and is used to manage dependencies for Node.js projects. It provides a vast repository of reusable code and libraries that can be easily integrated into Node.js applications.
Example: Developers can use NPM to install packages like Express.js for building web servers, Mongoose for MongoDB integration, and many others.

4.JavaScript Everywhere –
Node.js allows developers to use JavaScript for both serverside and clientside development. This means that the same language can be used for the entire stack, making it easier for developers to write and maintain code.
This reduces the context switching overhead and simplifies the development process.
Benefit: This feature simplifies the development process and reduces the learning curve for developers who are already familiar with JavaScript.

5.CROSS PLATFORM –

Node.js is cross platform, meaning it can run on multiple operating systems, including Windows, macOS, and Linux.

6.Builtin Support for Asynchronous Programming

Explanation: Node.js provides builtin support for asynchronous programming through its APIs. This allows developers to write nonblocking code that can handle multiple tasks concurrently.

7.Lightweight and Efficient

Explanation: Node.js is lightweight and efficient, making it suitable for building scalable and high-performance applications. Its event driven architecture and nonblocking I/O model ensure that it can handle a large number of concurrent connections with minimal resource usage.

SYNCHRONOUS AND ASYNCHRONOUS -

In **synchronous programming**, tasks are executed **one after another**, in a sequential manner. Each operation **waits** for the previous one to complete before moving forward.

Key Characteristics:

- Blocking in nature
- Simple to understand and debug step by step
- Slows down performance for I/O operations like file reads or database queries

Example:

```
const fs = require('fs');
const data = fs.readFileSync('file.txt', 'utf8');
console.log(data);
console.log('Finished reading file');
```

In this example, the second console.log line will not run until the file is completely read.

Asynchronous programming allows tasks to **run in the background** without blocking the main execution thread. It uses **callbacks**, **promises**, or **async/await** to handle operations that take time, such as API calls, file systems, or databases.

Key Characteristics:

- Non-blocking
- More efficient for I/O-heavy applications
- Essential for high scalability in modern web apps

Example using async/await:

```
const fs = require('fs').promises;
async function readFile() {
  const data = await fs.readFile('file.txt', 'utf8');
  console.log(data);
}
readFile();
console.log('Reading file in the background');
```

In this example, the program continues executing without waiting for the file read operation to finish.

ROUTING IN EXPRESS JS –

Express.js is a **minimal and flexible web application framework** for **Node.js**. It is used to build **web applications** and **RESTful APIs** quickly and easily.
It simplifies the process of building **server-side logic**, managing routes, handling HTTP requests/responses, middleware, and more. making backend development faster and more maintainable.

Key Features of ExpressJS –

1. Minimal & Lightweight : doesn't need much overhead ,it's a lightweight framework .
2. Middleware Support: we can easilty integrate custom or third party middleware functions for tasks like authentication , error handling , cookies etc.
3. Routing: Built in routing system , its baciscally the process of determining how the application responds to the client request. For a particular end point.
4. Integration with Databases : works well with MONGODB , MYSQL , POSTGRESQL etc. authentication libraries (JWT,Password,bcrypt) etc.
5. Static File serving: can serve static files like HTML,CSS,JS with built in middlewares.

Example Use Case

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Welcome to Express.js');
});

app.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

Concept of Routing in Express

Routing in Express refers to the process of determining how an application responds to a client request for a particular endpoint. An endpoint is a combination of a URI (or path) and a specific HTTP request method (GET, POST, PUT, DELETE, etc.). Express uses middleware functions to handle these requests and responses.

In Express, you define routes using the `app` object, which is an instance of the Express application. You can handle different HTTP methods (GET, POST, PUT, DELETE) by using the corresponding methods on the `app` object.

1. GET Requests

GET requests are used to retrieve data from the server. You define a GET route using the `app.get()` method.

```
app.get('/users', (req, res) => {
  res.send('GET request to the /users endpoint');
});
```

2. POST Requests

POST requests are used to create new resources on the server. You define a

```
app.post('/users', (req, res) => {
  res.send('POST request to the /users endpoint');
});
```

3. PUT Requests

PUT requests are used to update existing resources on the server. You define a PUT route using the `app.put()` method.

```
app.put('/users/:id', (req, res) => {
  res.send(`PUT request to update user with ID: ${req.params.id}`);
});
```

4. DELETE Requests

DELETE requests are used to delete resources on the server. You define a

```
app.delete('/users/:id', (req, res) => {
  res.send(`DELETE request to remove user with ID: ${req.params.id}`);
});
```

ROLE OF MIDDLEWARE IN EXPRESS JS

Middleware is a fundamental concept in Express.js that acts as a bridge between incoming requests and outgoing responses. These functions execute sequentially during the request-response cycle, allowing you to modify requests, responses, or control the flow of execution.

Key Characteristics of Middleware

Middleware functions receive three parameters:

- `req` (request object) - contains information about the HTTP request
- `res` (response object) - used to send responses back to the client
- `next` - a function that passes control to the next middleware in the stack
- err – designed to catch and handle errors that occurs during request processing.

Types of Middleware

- **Application-level middleware** is bound to the entire app using `app.use()` or specific HTTP methods like `app.get()` , `app.post()` `etc`. These functions run for every request or specific routes depending on how they're configured.
- **Router-level middleware :** these are bounded to specific routes for e.g. placing order , user dashboard , locations, cart , etc. these middleware should only be accessed by the authenticated users.

```
const token = req.cookies?.accessToken ||
req.header("Authorization")?.replace("Bearer ","")
        //Authorization: Bearer <token>

        if(!token){
            throw new ApiError(401,"unauthorized request")
        }
        const decodedToken = jwt.verify(token,process.env.ACCESS_TOKEN_SECRET)
        const user = await User.findById(decodedToken?._id).select("-password -
refreshToken")
        if(!user){
            throw new ApiError(401,"Invalid access token")
        }
        req.user = user;
        next()
```

```
const router = Router();
router.route("/current-user").get(verifyJWT,getCurrentUser)
router.route("/update-account").patch(verifyJWT,updateAccountDetails)
router.route("/logout").post(verifyJWT,logoutUser)
router.route("/refresh-token").post(refreshAccessToken)
router.route("/change-password").post(verifyJWT,changeCurrentPassword)
```

```
app.use("/api/v1/users",userRouter)
```

- **Error-handling middleware** has a special signature with four parameters `(err, req, res, next)` and is designed to catch and handle errors that occur during request processing.
- **Built-in middleware** includes functions like `express.static()` for serving static files and `express.json()` for parsing JSON payloads. , express.urlencoded – parses URL-ncoded bodies .

```
app.use(express.json({limit:"16kb"}))
app.use(express.urlencoded({extended:true , limit:"16kb"}))
app.use(express.static("public"))
```

Third-party middleware extends Express functionality through packages like `cors` for handling cross-origin requests, `helmet` for security headers, or `morgan` for logging means it help developers to monitor incoming req , analysis traffic,response time etc. Multer for file uploading. Cookie parser: It's commonly used to handle session tracking, user authentication.

```
app.use(cors({
origin: process.env.CORS_ORIGIN,
credentials: true,
methods:[“GET”,”POST,”PUT”,”DELETE”],
allowHeaders:[‘content-type’,’Authorization’]
}));
app.use(cookie-parser());
- we can set the cookies res.cookie()
-we can apply maxAge , httpOnly etc.
-res.clearCookie()
app.use(morgan('dev'));
app.use(helmet());  // it enables all necessary http protections.
```

NO SQL (MONGO-DB SETUP) –

1.Install MongoDB locally in your computer or use MongoDB Atlas for cloud storing .
2.Install all the dependencies required like npm install express mongoose
3.Install all the other npm modules required like cors , cookie-parser , bcrypt , helmet

4.Connect To DB –

```
import ‘mongoose’ from mongoose;
const connectDB = async () => {
  try {
    await mongoose.connect('mongodb://localhost:27017/mydb'); // Replace with your URI
    console.log('MongoDB connected successfully');
  } catch (error) {
    console.error('MongoDB connection failed:', error.message);
    process.exit(1);
  }
};
Export default connectDB();
```

5.Models/User.js
Define collection and schema

```
import ‘mongoose’ from mongoose;
const userSchema = new mongoose.Schema({
  name: String,
  email: String,
  age: Number
}, { timestamps: true });
const User = mongoose.model(“User”,userSchema)
```

6.userController.js
CRUD operations

```
const registerUser = async (req, res) => {
  try {
    const {username,name,email,password} = req.body();
    const user = User.create({
     fullname,email,password
   })
    return res.status(201).json({message:user created successfully})
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
};
```

7.Routes setup

```
const router = Router();
Router.route('/registerUser',registerUser);
```

8.server.js

```
const app = express();
const PORT = process.env.PORT || 5000;

connectDB(); // MongoDB Connection
app.use(express.json()); // Middleware
app.use('/api/users', userRoutes); // Routes
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

SQL SETUP FOR WEB APPLICATION –

1.Setting Up SQL database –

Install Mysql workbench locally or can use the remote cloud services like Azure Mysql.

2.Start your DB server , then create the new database:

```
CREATE DATABASE myapp;
```

3.Creating Tables and Defining RelationShips :

```
CREATE TABLE Users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255),
  email VARCHAR(255)
);

CREATE TABLE Posts (
  id INT AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(255),
  content VARCHAR,
  userId INT,
  FOREIGN KEY (userId) REFERENCES Users(id)
);
```

4.Querying data using SQL :

```
-- Create
INSERT INTO Users (name, email) VALUES ('Anuj', 'anuj@example.com');

-- Read
SELECT * FROM Users;

-- Update
UPDATE Users SET email = 'new@example.com' WHERE id = 1;

-- Delete
DELETE FROM Users WHERE id = 1;
```

5.Integrating With Nodejs using Sequelize (ORM) , there are many ORMs (object relational mapper) like Prisma , Drizzle , sqllite3 etc.

Install dependencies like
Npm install sequelize mysql2

6.Connect To DB:

```
Import { Sequelize }from sequelize)
const sequelize = new Sequelize('myapp', 'root', 'password', {
  host: 'localhost',
  dialect: 'mysql',
});

Export const sequelize;
```

7.models
User.js

```
import { DataTypes } from 'sequelize';
import sequelize from '../config/db.js';
const User = sequelize.define('User', {
  name: DataTypes.STRING,
  email: DataTypes.STRING
});
export default User;
```

Posts.js

```
import { DataTypes } from 'sequelize';
import sequelize from '../config/db.js';
import User from './User.js';

const Post = sequelize.define('Post', {
  title: DataTypes.STRING,
  content: DataTypes.TEXT
});

Post.belongsTo(User);
User.hasMany(Post);

export default Post;
```

8.server.js

```
import express from 'express';
import sequelize from './config/db.js';
import User from './models/User.js';
import Post from './models/Post.js';

const app = express();
app.use(express.json());

sequelize.sync().then(() => {
  console.log('✅ Database synced');
});

// Create a new user
app.post('/users', async (req, res) => {
  try {
    const user = await User.create(req.body);
    res.json(user);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// Get all users with their posts
app.get('/users', async (req, res) => {
  try {
    const users = await User.findAll({ include: Post });
    res.json(users);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
const PORT = 5000;
app.listen(PORT, () => console.log(`🚀 Server running on
http://localhost:${PORT}`));
```

Node.js function to insert a new user into a users table and retrieve all users.

Example schema:

```
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100),
  email VARCHAR(100)
);

const mysql = require('mysql2/promise');

// Setup MySQL connection pool
const db = mysql.createPool({
  host: 'localhost',
  user: 'root',
  password: 'your_password',
  database: 'your_database'
});

// Insert a new user and fetch all users
async function insertAndFetchUsers(name, email) {
  try {
    await db.execute('INSERT INTO users (name, email) VALUES (?, ?)', [name,
email]);

    // Retrieve all users
    const [users] = await db.execute('SELECT * FROM users');

    console.log('All Users:', users);
    return users;

  } catch (err) {
    console.error('Database Error:', err.message);
    throw err;
  }
}

insertAndFetchUsers('John Doe', 'john@example.com');
```

Q.7 Write a JavaScript function to dynamically add multiple list items to an unordered list and another function to remove all list items.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Dynamic List with User Input</title>
  <style>
    body { font-family: Arial, sans-serif; padding: 20px; }
    input, button { margin: 5px 0; padding: 8px 12px; }
    ul { margin-top: 15px; }
    li { padding: 4px 0; }
  </style>
</head>
<body>
  <h2>Enter items (comma-separated):</h2>

  <input type="text" id="itemInput" placeholder="e.g. Apple, Banana, Cherry">
  <br>
  <button onclick="addListItems()">Add Items</button>
  <button onclick="clearList()">Clear List</button>

  <ul id="myList"></ul>

  <script>
    // Add list items from user input
    function addListItems() {
      const input = document.getElementById('itemInput').value;
      const items = input.split(',').map(item => item.trim()).filter(item => item
!== '');

      const ul = document.getElementById('myList');
      items.forEach(item => {
        const li = document.createElement('li');
        li.textContent = item;
        ul.appendChild(li);
      });
      // Clear input field after adding
      document.getElementById('itemInput').value = '';
    }
    // Remove all list items
    function clearList() {
      const ul = document.getElementById('myList');
      ul.innerHTML = '';
    }
  </script>
</body>
</html>
```

Q . FEEDBACK FORM

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Feedback Form</title>
  <style></style>
  </style>
</head>
<body>

  <h2>Feedback Form</h2>

  <form id="feedbackForm">
    <label for="name">Your Name:</label>
    <input type="text" id="name" required>

    <label>Overall Experience:</label>
    <div class="radio-group">
      <input type="radio" name="experience" value="Excellent" required> Excellent
      <input type="radio" name="experience" value="Good"> Good
      <input type="radio" name="experience" value="Average"> Average
      <input type="radio" name="experience" value="Poor"> Poor
    </div>

    <label>What features did you use?</label>
    <div class="checkbox-group">
      <input type="checkbox" name="features" value="Search"> Search
      <input type="checkbox" name="features" value="Filter"> Filter
      <input type="checkbox" name="features" value="Profile"> Profile
      <input type="checkbox" name="features" value="Chat"> Chat
    </div>

    <label for="comments">Additional Comments:</label>
    <textarea id="comments" rows="4"></textarea>

    <button type="button" onclick="processForm()">Submit Feedback</button>
  </form>

  <div id="result"></div>

  <script>
    function processForm() {
      const name = document.getElementById('name').value;
      const experience =
document.querySelector('input[name="experience"]:checked')?.value || 'N/A';

      const featureElems =
document.querySelectorAll('input[name="features"]:checked');
```

```
      const features = Array.from(featureElems).map(el => el.value).join(', ') ||
'None';

      const comments = document.getElementById('comments').value || 'No
comments';

      const output = `
        <h3>Feedback Summary</h3>
        <p><strong>Name:</strong> ${name}</p>
        <p><strong>Experience:</strong> ${experience}</p>
        <p><strong>Features Used:</strong> ${features}</p>
        <p><strong>Comments:</strong> ${comments}</p>
      `;

      document.getElementById('result').innerHTML = output;
    }
  </script>

</body>
</html>
```

Q. VALIDATION FORM

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Registration</title>
</head>
<body>
  <form onsubmit="return handleSubmit(event)">
    <input type="text" id="username" placeholder="Username" required />
    <input type="email" id="email" placeholder="Email" required />
    <input type="tel" id="phone" placeholder="Phone (10 digits)" required />
    <input type="password" id="password" placeholder="Password" required />
    <input type="password" id="confirm" placeholder="Confirm Password" required
/>
    <button type="submit">Register</button>
    <div class="message" id="msgBox"></div>
  </form>

  <script>
    function handleSubmit(e) {
      e.preventDefault();

      const username = document.getElementById("username").value.trim();
      const email = document.getElementById("email").value.trim();
      const phone = document.getElementById("phone").value.trim();
      const password = document.getElementById("password").value;
      const confirm = document.getElementById("confirm").value;
      const msg = document.getElementById("msgBox");

      msg.style.color = "red";

      if (!username || !email || !phone || !password || !confirm) {
        msg.textContent = "Fill all fields.";
        return false;
      }

      if (!/^\d{10}$/.test(phone)) {
        msg.textContent = "Phone must be 10 digits.";
        return false;
      }

      if (!/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email)) {
        msg.textContent = "Invalid email.";
        return false;
      }

      if (password !== confirm) {
```

```
        msg.textContent = "Passwords do not match.";
        return false;
      }

      msg.style.color = "green";
      msg.textContent = "Registration successful!";
      return true;
    }
  </script>
</body>
</html>
```