



**SCHOOL
OF
COMPUTER SCIENCE
AND
ENGINEERING**

**WEB TECHNOLOGY LAB
B22EF0606/ B22EH0606**

Fourth Semester
AY-2024-25
(Prepared in Jan-2025)

INDEX

SL. No	Contents	Page. no
1	Lab Objectives	2
2	Lab Outcomes	2
3	Lab Requirements	3
4	Guidelines to Students	3
5	List of Lab Exercises	4
6	Lab Exercises' Solutions:	6
LAB EXERCISES		
	Session – 1: Lab Exercise	7
	Session – 2: Lab Exercise	11
	Session – 3: Lab Exercise	14
	Session – 4: Lab Exercise	17
	Session – 5: Lab Exercise	20
	Session – 6: Lab Exercise	25
	Session – 7: Lab Exercise	28
	Session – 8: Lab Exercise	33
	Session – 9: Lab Exercise	37
	Session – 10: Lab Exercise	40
7	Learning Resources	

1. Lab Objectives:

The objectives of this course are to

- Build dynamic web pages with the help of various HTML tags and perform validation using Java Script objects by applying different event handling mechanisms.
- Comprehend the importance of CSS in designing a creative and dynamic website and embedding Java Script code in HTML.
- Understand and be able to develop JavaScript code to access the DOM structure of web document and object properties.
- Develop dynamic web pages with usage of server-side scripting.

2. Lab Outcomes:

On successful completion of this course; student shall be able to:

CO#	Course Outcomes	POs	PSOs
------------	------------------------	------------	-------------

CO1	Make use of HTML tags and CSS to build web pages for various applications.	1- 3,5 ,9-12	1,2,3
CO2	Demonstrate the usage of form data to validate the correctness of given input.	1-3,9-12	1,2
CO3	Apply the variety of presentation effects in HTML documents, including explicit positioning of elements using CSS.	1-5,10-12	1,2,3
CO4	Prepare a HTML document for Interactive webpage using JavaScript.	1-5,10-12	1,2,3
CO5	Illustrate the concept of functions in Javascript	1-5,10-12	1,2,3
CO6	Analyze the concepts of server side technologies for dynamic web applications.	1-5,10-12	1,2,3

3. Lab Requirements:

The following are the required hardware and software for this lab.

Hardware Requirements: A standard personal computer or laptop with the following minimum specifications:

1. **Processor:** Any modern multi-core processor (e.g., Intel Core i5 or AMD Ryzen series).
2. **RAM:** At least 4 GB of RAM for basic programs; 8 GB or more is recommended for larger data structures and complex algorithms.
3. **Storage:** A few gigabytes of free disk space for the development environment and program files.
4. **Display:** A monitor with a resolution of 1024x768 or higher is recommended for comfortable coding.
5. **Input Devices:** Keyboard and mouse (or other input devices) for coding and interacting with the development environment.

Software Requirements:

1. **Operating System:** You can develop and execute HTML programs for Web Technology Lab on various operating systems, including Windows, macOS, and Linux.
2. **Text Editor or Integrated Development Environment (IDE):**
 - **Text Editor:** You can use a simple text editor like Notepad (Windows), Nano (Linux/macOS), or any code-oriented text editor like Visual Studio Code, Sublime Text, or Atom.
 - **IDE:** Consider using a C/C++ integrated development environment like Code::Blocks, C++, or Eclipse CDT for a more feature-rich coding experience.
3. **Tools:**
 - Web Browser (Chrome, Firefox, Edge, or Safari)
 - Code Editor (Visual Studio Code, Sublime Text, Atom, Notepad++)
 - Node.js and npm (for ReactJS and other server-side functionalities)
 - Local Server (XAMPP, MAMP, or WAMP – Optional for more advanced setups)

4. Guidelines to Students:

- Equipment in the lab for the use of the student community. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care. Any damage caused is punishable.

- Students are required to carry their observation / programs book with completed exercises while entering the lab.
- Students are supposed to occupy the machines allotted to them and are not supposed to talk or make noise in the lab. The allocation is put up on the lab notice board.
- The lab can be used in free time / lunch hours by the students who need to use the systems should get prior permission from the lab in-charge.
- Lab records need to be submitted on or before the date of submission.
- Students are not supposed to use flash drives.

Practice

Here are some key activities involved in the Web Technology Lab:

1. Understand the Task: Read the instructions carefully to understand the problem before you start coding. Break the task into smaller parts.
2. Set Up Your Environment: Use a web browser (Chrome, Firefox) to run the code and a code editor (VS Code, Sublime Text) to write the code.
3. Organize Your Files: Keep your project files organized in folders, e.g., js/, css/, images/, and name files meaningfully (e.g., form-validation.js).
4. Write Small Code Segments: Write code in small parts and test them immediately in the browser to see if they work as expected.
5. Test Frequently: Test your code regularly after writing each function or feature, checking for errors in the browser's console.
6. Use Developer Tools: Use browser developer tools (F12 or Ctrl + Shift + I) to debug, inspect elements, and view console errors.
7. Comment Your Code: Add comments to explain what your code does, especially for complex sections, to make it easier to understand later.
8. Validate and Debug: Use console.log() for debugging. Test for edge cases (e.g., invalid inputs) to ensure everything works correctly.
9. Use Version Control: If possible, use Git to save versions of your code and avoid losing progress.
10. Collaborate and Ask for Help: Work with peers and ask questions when you're stuck. Use online resources like StackOverflow and MDN for solutions.

These activities are iterative and may involve revisiting earlier stages as the design and analysis process progresses. The goal is to create efficient and reliable algorithms that can effectively solve specific problems.

5. List of Lab Exercises and Mini-Projects:

Sl No.	Title of the Exercise
	Part A
1	Write JavaScript to validate the following fields of the Registration page. 1. First Name (Name should contains alphabets and the length should not be less than 6 characters).

	2. Password (Password should not be less than 6 characters length). 3. E-mail id (should not contain any invalid and must follow the standard pattern name@domain.com) 4. Mobile Number (Phone number should contain 10 digits only). 5. Last Name and Address (should not be Empty).
2	Write a JavaScript code that displays text “ TEXT-GROWING ” with increasing font size in the interval of 100ms in RED COLOR, when the font size reaches 50pt it displays “ TEXT-SHRINKING ” in BLUE color. Then the font size decreases to 5pt.
3	Write an HTML page that contains a selection box with a list of 5 countries. When the user selects a country, its capital should be printed next to the list. Add CSS to customize the properties of the font of the capital (color, bold and font size).
4	Write an HTML page that has one input, which can take multi line text and a submit button. Once the user clicks the submit button, it should show the number of characters, words and lines in the text entered using an alert message. Words are separated with white space and lines are separated with new line character.
5	Develop and demonstrate JavaScript with POP-UP boxes and functions for the following problems: a) Input: Click on Display Date button using onclick() function Output: Display date in the textbox b) Input: A number n obtained using prompt Output: Factorial of n number using alert c) Input: A number n obtained using prompt Output: A multiplication table of numbers from 1 to 10 of n using alert d) Input: A number n obtained using prompt and add Output: display sum in document another number using confirm entire n numbers using alert
6	Develop and demonstrate the usage of inline, internal and external style sheet using CSS
7	a) Develop and demonstrate, using JavaScript script, a XHTML document that contains three short paragraphs of text, stacked on top of each other, with only enough of each showing so that the mouse cursor can be placed over some part of them. When the cursor is placed over the exposed part of any paragraph, it should rise to the top to become completely visible. b) Modify the above document so that when a paragraph is moved from the top stacking position, it returns to its original position rather than to the bottom
8	Write a XHTML Program using Java script to create a web page with images where these images navigate between images using: <ul style="list-style-type: none"> • click/arrow button • automation • Text/Image
9	Write a XHTML Program using Java script to create multiple line of content. Use various buttons to <ul style="list-style-type: none"> • Adding new content • Replacing the existing content • Deleting the Content
10	Demonstrate how to create components in ReactJS 1.React Functional Components 2. React Class based Components

LAB EXERCISES

6. Lab Exercises' Solutions:

	Solutions
1.	<p>Write JavaScript to validate the following fields of the Registration page.</p> <ol style="list-style-type: none"> First Name (Name should contains alphabets and the length should not be less than 6 characters). Password (Password should not be less than 6 characters length). E-mail id (should not contain any invalid and must follow the standard pattern name@domain.com) Mobile Number (Phone number should contain 10 digits only). Last Name and Address (should not be Empty). <p>Problem Statement:</p> <p>This task is to design and implement a web-based registration form that validates user inputs for key fields like First Name, Last Name, Password, Email, Mobile Number, and Address. The validation rules include ensuring fields are not empty, meet length requirements, and follow standard patterns (e.g., email format, mobile number length). The validation should occur client-side using JavaScript, providing immediate feedback to users about errors before submission.</p> <p>Solution Overview:</p> <p>The solution involves developing a webpage with the following key components:</p> <p>HTML Structure:</p> <p>Create a form with fields for First Name, Last Name, Password, Email, Mobile Number, and Address. Include labels for each input field to guide the user. Use the id attribute for easy JavaScript access.</p> <p>Validation Logic in JavaScript:</p> <p>Add an event listener to the form's submit event to intercept submission. Use JavaScript to retrieve user inputs and validate them: First Name: Must contain only alphabets and be at least 6 characters long. Password: Must be at least 6 characters. Email: Must follow the format name@domain.com. Mobile Number: Must be exactly 10 digits. Last Name and Address: Must not be empty. Use regular expressions (Regex) for pattern-based validations. Provide error messages for invalid fields using alert().</p> <p>Feedback:</p> <p>If all validations pass, show a success message. If any validations fail, display a detailed list of errors.</p> <p>Intuition:</p> <p>The purpose of this program is to ensure user input integrity and correctness in a registration form. Without proper validation, incorrect or incomplete data can lead to issues in backend processing. By validating inputs on the client side, we provide an efficient way to guide users in entering the correct information, saving server resources and improving user experience.</p> <p>Code Implementation:</p> <pre><!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0"> <title>Registration Form</title> <script src="validation.js" defer></script></pre>

```
</head>
<body>
  <form id="registrationForm">
    <label for="firstName">First Name:</label>
    <input type="text" id="firstName" name="firstName"><br><br>

    <label for="lastName">Last Name:</label>
    <input type="text" id="lastName" name="lastName"><br><br>

    <label for="password">Password:</label>
    <input type="password" id="password" name="password"><br><br>

    <label for="email">E-mail:</label>
    <input type="text" id="email" name="email"><br><br>

    <label for="mobile">Mobile Number:</label>
    <input type="text" id="mobile" name="mobile"><br><br>

    <label for="address">Address:</label>
    <textarea id="address" name="address"></textarea><br><br>

    <button type="submit">Register</button>
  </form>
</body>
</html>
document.getElementById("registrationForm").addEventListener("submit", function(event) {
  // Prevent the form from submitting
  event.preventDefault();

  // Get form field values
  const firstName = document.getElementById("firstName").value.trim();
  const lastName = document.getElementById("lastName").value.trim();
  const password = document.getElementById("password").value.trim();
  const email = document.getElementById("email").value.trim();
  const mobile = document.getElementById("mobile").value.trim();
  const address = document.getElementById("address").value.trim();

  // Validate fields
  let isValid = true;
  let errorMessage = "";

  // 1. Validate First Name
  if (!/^[a-zA-Z]{6,}$/.test(firstName)) {
    isValid = false;
    errorMessage += "First Name must contain only alphabets and be at least 6 characters long.\n";
  }

  // 2. Validate Password
  if (password.length < 6) {
    isValid = false;
    errorMessage += "Password must be at least 6 characters long.\n";
  }

  // 3. Validate Email
  if (!/^[^@\\s@]+@[^\\s@]+\\.([^\\s@]+)$/.test(email)) {
    isValid = false;
    errorMessage += "E-mail must follow the standard format (name@domain.com).\n";
  }

  // 4. Validate Mobile Number
```



```
if (!/^d{10}$/.test(mobile)) {
    isValid = false;
    errorMessage += "Mobile Number must contain exactly 10 digits.\n";
}

// 5. Validate Last Name and Address
if (lastName === "") {
    isValid = false;
    errorMessage += "Last Name must not be empty.\n";
}
if (address === "") {
    isValid = false;
    errorMessage += "Address must not be empty.\n";
}

// Display validation result
if (isValid) {
    alert("Registration successful!");
} else {
    alert("Validation Error:\n" + errorMessage);
}
});
```

Sample Output:

First Name:

Last Name:

Password:

E-mail:

Mobile Number:

Address:

First Name:

Last Name:

Password:

E-mail:

Mobile Number:

Address:

This page says

Validation Error:
First Name must contain only alphabets and be at least 6 characters long.

Outcome of the Exercise:

After implementing the solution:

Validation Success: Users who input valid data will see a success message (Registration successful!).

Error Handling: Invalid inputs (e.g., a 5-character first name or an incorrect email format) will prompt error messages detailing the issues.

Prevention of Invalid Data Submission: The form cannot be submitted until all input fields meet the validation criteria.

By completing this program, Users gain confidence in the registration process, and developers can ensure data integrity before further processing.

Viva/Interview Questions:

1. Why is client-side validation important?

Answer: Client-side validation ensures that users input valid data before the form is submitted to the server. It reduces server load, improves user experience by providing immediate feedback, and minimizes the risk of invalid data corrupting the database.

2. What is the significance of `event.preventDefault()` in form validation?

Answer: The `event.preventDefault()` method prevents the default form submission behavior. It allows the program to validate the input fields first, ensuring no invalid data is submitted to the server.

3. How do regular expressions help in input validation?

Answer: Regular expressions provide a concise and efficient way to define patterns for matching specific input formats. For example, validating an email format (name@domain.com) or ensuring a mobile number contains exactly 10 digits.

4. What happens if a user enters an invalid value in one of the fields?

Answer: If any field contains invalid input, the program stops the form submission process, sets `isValid` to false, and displays an alert with specific error messages explaining what needs to be corrected.

5. How is the length of the password validated in this program?

Answer: The password is validated using a simple check:

```
if (password.length < 6) {  
    isValid = false;  
    errorMessage += "Password must be at least 6 characters long.\n";  
}
```

If the password length is less than 6 characters, an error message is added, and the form submission is halted.

6. Why do we use `.trim()` on user inputs?

Answer: The `.trim()` method removes unnecessary leading and trailing spaces from user input. This ensures that blank spaces don't interfere with validation rules, such as checking for empty fields or character counts.

7. What will happen if JavaScript is disabled in the browser?

Answer: If JavaScript is disabled, client-side validation will not work. The form will be submitted without checking the input, which could lead to invalid or harmful data being sent to the server. Server-side validation should always be implemented as a fallback for such scenarios.

8. How does this program handle multiple validation errors at the same time?

Answer: The program accumulates error messages in the `errorMessage` variable during the validation process. If multiple fields are invalid, their respective error messages are concatenated and displayed in a single alert box after validation.

9. What are the key differences between client-side and server-side validation?

Answer: Client-Side Validation: Performed in the browser before the form is submitted. It provides immediate feedback to the user and reduces server load.

Server-Side Validation: Performed on the server after form submission. It ensures security and handles scenarios where JavaScript is disabled or malicious inputs are sent.

	<p>10. How can you test this program for robustness? Answer: You can test the program by: Entering valid and invalid inputs for each field. Testing boundary cases (e.g., 6 characters for first name and password). Trying inputs with special characters, spaces, and mixed cases. Checking how it behaves with empty or partially filled fields</p>
2	<p>Write a JavaScript code that displays text “TEXT-GROWING” with increasing font size in the interval of 100ms in RED COLOR, when the font size reaches 50pt it displays “TEXT-SHRINKING” in BLUE color. Then the font size decreases to 5pt.</p> <p>Problem Statement:</p> <p>Create a JavaScript code to accomplish the described behavior. This code creates a dynamic text element that grows and shrinks its font size, toggling between "TEXT-GROWING" and "TEXT-SHRINKING" at the respective thresholds.</p> <p>Solution Overview:</p> <p>The JavaScript code dynamically animates text to grow and shrink in size while changing its color and content based on the font size. Here's a detailed breakdown:</p> <ul style="list-style-type: none"> • Text Content: The text starts as "TEXT-GROWING" in red. When the font size reaches 50pt, it switches to "TEXT-SHRINKING" in blue. • Font Size Behavior: The font size begins at 5pt and increases in intervals of 100 milliseconds. Upon reaching a maximum size of 50pt, the text switches to "TEXT-SHRINKING" and the font size decreases at the same interval until it reaches 5pt. Once it reaches 5pt, the text switches back to "TEXT-GROWING," repeating the cycle. • Color Changes: During the growing phase, the text is displayed in red. During the shrinking phase, the text is displayed in blue. • Animation Implementation: The setInterval function is used to execute the animation logic every 100 milliseconds. The growing variable tracks whether the text is currently increasing or decreasing in size. CSS properties like fontSize and color are dynamically updated to reflect the current state of the text. • User Interaction: No user interaction is required; the animation starts immediately when the page loads and runs in an infinite loop. This animation creates a visually appealing effect, suitable for showcasing dynamic text behavior on a webpage. <p>Intuition:</p> <p>The task requires a dynamic animation where text alternates between growing and shrinking states with changes in size, color, and displayed content. Here's the thought process:</p> <ul style="list-style-type: none"> • Starting Point: Initialize the text with a small font size (e.g., 5pt), red color, and the content "TEXT-GROWING." • Growth Phase: Continuously increase the font size at fixed intervals (100ms). Stop increasing when the font size reaches the maximum value (50pt) and switch to the shrinking phase. • Shrinking Phase: Change the text content to "TEXT-SHRINKING" and set the color to blue. Gradually decrease the font size at the same interval until it reaches the minimum value (5pt). • Looping Behavior: Alternate between the growing and shrinking phases to create a looping animation. • Implementation with JavaScript: Use a setInterval function to perform actions at a regular interval. Track the current font size using a variable (fontSize) and determine the direction of change (growing or shrinking) using a boolean flag (growing).

Dynamically update the font size, text content, and color based on the current state.

- **Styling:**

Use CSS to center the text and create a smooth transition effect for the font size changes.

This approach ensures a continuous and visually appealing animation that transitions smoothly between growing and shrinking states.

Code Implementation:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Text Growing and Shrinking</title>
  </head>
  <body>
    <center>
      <p id="demo">TEXT</p>
    </center>
    <script>
      // Define variables at the document level
      var size = 5; // Initial font size
      var growing = true; // Flag to track growth or shrinkage
      var ids = document.getElementById("demo"); // Reference to the text element
      // Function to handle the animation logic
      function animateText() {
        if (growing) {
          // Growing phase
          size += 5;
          ids.innerHTML = "TEXT GROWING";
          ids.setAttribute("style", "font-size:" + size + "px; color:red");
          // Switch to shrinking when size reaches 50px
          if (size >= 50) {
            growing = false;
          }
        } else {
          // Shrinking phase
          size -= 5;
          ids.innerHTML = "TEXT SHRINKING";
          ids.setAttribute("style", "font-size:" + size + "px; color:blue");
          // Switch back to growing when size reaches 5px
          if (size <= 5) {
            growing = true;
          }
        }
      }
      // Start the animation when the page loads
      var intervalId = setInterval(animateText, 100); // Run every 100 milliseconds
    </script>
  </body>
</html>
```

Sample Output:

TEXT SHRINKING

TEXT GROWING

Viva/Interview Questions:

1. What is the purpose of the setInterval() function in this code?

Answer: The setInterval() function is used to execute a function repeatedly at specified intervals (in this case, every 100 milliseconds). In the code, it is used to change the font size of the text periodically, causing the text to grow and shrink.

2. Why do we use clearInterval()? What would happen if it was not used?

Answer: clearInterval() is used to stop the repeated execution of a function that was started by setInterval(). Without it, the function would continue running indefinitely, causing the text to shrink below 5pt and possibly crash the browser due to continuous execution.

3. Explain how the fontSize is being updated in the code.

Answer: The fontSize is initially set to 10pt. In the changeTextSize() function, the fontSize is increased by 2pt every time the function is called, as long as the text is still growing. Once the font size reaches 50pt, the fontSize starts decreasing by 2pt, and the process continues until it reaches 5pt.

4. What is the role of the growing variable in this program?

Answer: The growing variable is a flag used to determine whether the text is growing or shrinking. When growing is true, the font size increases. Once the font size reaches 50pt, the growing variable is set to false, causing the font size to decrease instead of increase.

5. How does the textElement.style.fontSize property work?

Answer: The textElement.style.fontSize property directly manipulates the inline CSS of the HTML element. By setting it to a new value (e.g., '50pt'), it updates the font size of the element, causing the text to resize. In this code, the fontSize is dynamically updated in each interval.

6. Why does the text color change from red to blue in the code?

Answer: The text color changes from red to blue to indicate the transition from the growing phase to the shrinking phase. When the font size reaches 50pt, the text switches to "TEXT-SHRINKING" and the color is updated to blue to visually signify this change.

7. What would happen if the interval time (100ms) was changed to a smaller value?

Answer: If the interval time was reduced (e.g., to 50ms or 10ms), the font size would change more rapidly, and the text would grow and shrink faster. This would make the effect visually quicker and more dynamic.

8. How would you modify the code to make the text grow faster (e.g., by increasing font size more rapidly)?

Answer: To make the text grow faster, you can modify the increment value of the fontSize variable inside the changeTextSize() function. Instead of increasing by 2pt, you could increase it by a larger value, such as 5pt.

For example:

```
javascript
fontSize += 5; // Increase font size by 5pt
This would make the font size grow faster.
```

9. Can the transition property in CSS affect the text's growth and shrink effect? How?

Answer: Yes, the transition property in CSS can smooth the change in font size. The transition: font-size 0.1s ease; rule in the CSS ensures that the font size changes smoothly over 0.1 seconds whenever it is modified. Without this, the font size would change instantly rather than gradually.

10. How would you modify the code to make the text grow indefinitely rather than shrinking after reaching 50pt?

Answer: To make the text grow indefinitely, you would need to remove the shrinking logic in the code. Specifically, you could eliminate the section that switches the text to "TEXT-SHRINKING" and changes the color to blue. You would also need to remove the part of the code that decreases the font size. Here's an example:

```
javascript
if (fontSize >= 50) {
    // Stop shrinking and keep growing indefinitely
    // No shrinking logic needed
}
```

This would make the text continue growing indefinitely without ever shrinking back.

- 3 Write an HTML page that contains a selection box with a list of 5 countries. When the user selects a country, its capital should be printed next to the list. Add CSS to customize the properties of the font of the capital (color, bold and font size).

Problem Statement:

Create a web page that allows users to interactively select a country from a dropdown list and instantly view its capital city. The page should have the following features:

- **Country Selection:** A dropdown menu containing a list of five countries.
- **Capital Display:** Upon selecting a country, its corresponding capital should appear dynamically next to the dropdown.
- **Custom Styling:** The displayed capital city should have customized font properties using CSS:
The text should be bold.
The font color should be visually distinct.
The font size should be larger than the default size for easy readability.

Solution Overview:

The solution involves creating a simple and interactive web page using HTML, CSS, and JavaScript. Here's an overview of how the solution will be implemented:

- **HTML Structure:**

A <select> element will be used to create a dropdown menu listing five countries.

A or <div> will display the capital city dynamically based on the selected country.

- **CSS Styling:**

Custom CSS will be applied to style the displayed capital city:

The font color will be distinct, e.g., dark blue or green.

The text will be bold for emphasis.

The font size will be increased for readability.

Additional styling can be applied to enhance the page's overall aesthetics.

- **JavaScript Interactivity:**

A JavaScript function will handle the onchange event of the dropdown menu.

Based on the selected country, the corresponding capital will be displayed dynamically by updating the inner text of the designated element.

- **Functionality Flow:**

The user selects a country from the dropdown menu.

JavaScript detects the selected option and maps it to the corresponding capital.

The page updates to display the selected capital city with the specified styles applied.

Intuition:

The idea behind this solution is to make the interaction simple and intuitive for the user. By using a dropdown menu, users can quickly choose a country without any complexity. The design follows these principles:

- **User-Centric Design:**

Dropdown menus are a familiar and user-friendly way to present options. By dynamically displaying the capital city next to the selected country, the user gets instant feedback without needing to reload the page or perform additional actions.

- **Dynamic Interactivity with JavaScript:**

JavaScript enables real-time updates by responding to the onchange event of the dropdown menu.

This eliminates the need for manual updates or page refreshes, keeping the interaction seamless and responsive.

- **Enhanced Readability with CSS Styling:**

Customizing the capital's font properties ensures that it stands out and is easy to read. The bold text, distinct color, and larger font size highlight the information clearly, which is especially important for accessibility.

- **Minimalism and Clarity:**

The solution keeps the interface clean and focused on the main task: selecting a country and viewing its capital. This prevents information overload and ensures the user can interact with the page effortlessly.

Code Implementation:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Country Capital Selector</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
    }
    label {
      font-size: 18px;
      margin-right: 10px;
    }
    select {
      font-size: 16px;
      padding: 5px;
    }
    .capital {
      font-size: 20px;
      font-weight: bold;
      color: blue;
      margin-left: 10px;
    }
  </style>
</head>
<body>
  <h1>Country and Capital Selector</h1>
  <label for="country">Select a country:</label>
  <select id="country" onchange="showCapital()">
    <option value="">--Select--</option>
    <option value="Ottawa">Canada</option>
    <option value="New Delhi">India</option>
    <option value="Tokyo">Japan</option>
    <option value="London">United Kingdom</option>
    <option value="Washington, D.C.">United States</option>
  </select>
  <span id="capital" class="capital"></span>

  <script>
  function showCapital() {
    const countrySelect = document.getElementById("country");
    const capitalDisplay = document.getElementById("capital");
    const selectedCapital = countrySelect.value;

    // Display the capital or clear if no selection
    capitalDisplay.innerHTML = selectedCapital ? `Capital: ${selectedCapital}` : "";
    // capitalDisplay.textContent = selectedCapital ? `Capital: ${selectedCapital}` : "";
  }
  </script>
</body>
</html>
```

Sample Output:

Country and Capital Selector

Select a country:

Country and Capital Selector

Select a country: **Capital: New Delhi**

Outcome of the Exercise:

1. Functional Interactive Web Page:

Users can select a country from the dropdown menu, and the capital city is dynamically displayed next to the selection in bold and styled text.

2. Real-Time Updates:

The capital updates instantly upon selection, providing immediate feedback without requiring a page reload.

3. Enhanced User Experience:

Clear font styling (color, boldness, size) ensures the displayed capital is easy to read.

The --Select-- placeholder guides users on how to interact with the dropdown.

4. Responsive and Accessible Design:

By adding aria-live="polite" (if implemented), the capital display would also be screen-reader-friendly, improving accessibility.

The simple, responsive styling ensures a clean look on both desktop and mobile devices.

5. Reusable Code:

The JavaScript logic is simple and modular, making it easy to add more countries or modify the functionality in the future.

Viva/Interview Questions:

1. What is the primary functionality of your program?

The program allows users to select a country from a dropdown menu, and it dynamically displays the corresponding capital city next to the selection. The capital city is styled using CSS for better readability.

2. Which HTML elements are used to create the dropdown and display the capital?

The <select> element is used to create the dropdown menu, and the element with the id="capital" is used to display the capital dynamically.

3. How does the program update the capital dynamically?

The program uses JavaScript to handle the onchange event of the dropdown. When a user selects a country, the showCapital function retrieves the selected value and updates the inner text of the element.

4. Why did you use innerHTML (or textContent) in your JavaScript function?

innerHTML is used to dynamically update the content of the element with the selected capital. Alternatively, textContent could also be used, which is more secure when dealing with plain text, as it avoids rendering potential HTML tags.

5. How is the capital text styled in your program?

The capital text is styled using a CSS class .capital, which includes:

Font size: 20px
 Font weight: bold
 Font color: blue This ensures the text is distinct and easy to read.

6. What happens if no country is selected from the dropdown?

If no country is selected or the --Select-- placeholder is chosen, the showCapital function clears the capital display by setting the inner text of the element to an empty string.

7. How can you improve the accessibility of your program?

To improve accessibility:

Add aria-live="polite" to the displaying the capital, so screen readers announce updates.

Use a label properly associated with the <select> using the for attribute, which is already implemented in the program.

8. Why did you use the onchange event in the dropdown?

The onchange event is triggered whenever a user selects a new option from the dropdown. It allows the program to detect the selection and update the displayed capital accordingly.

9. What are the advantages of separating CSS from HTML?

Separating CSS from HTML:

Improves readability and maintainability of the code.

Makes it easier to update the styles without affecting the HTML structure.

Promotes reusability of styles across multiple elements or pages.

10. How can this program be extended in the future?

The program can be extended by:

Adding more countries and their capitals to the dropdown.

Providing additional details about the selected country, such as population or currency.

Using an external API to fetch live data about countries and capitals.

Adding animations to enhance the user experience when displaying the capital

- 4 **Write an HTML page that has one input, which can take multi line text and a submit button. Once the user clicks the submit button, it should show the number of characters, words and lines in the text entered using an alert message. Words are separated with white space and lines are separated with new line character.**

Problem Statement:

Create an HTML page with a multi-line text input and a submit button. When the user submits the form, display the following details in an alert message:

- The number of characters in the input text (excluding spaces).
- The number of words, where words are separated by whitespace.
- The number of lines, where lines are separated by newline characters.

Solution Overview:

- **HTML Structure:**

A textarea element allows users to input multi-line text.

A button element submits the form.

- **JavaScript Logic:**

Attach a function analyzeText(event) to the form's onsubmit event.

Retrieve the content of the textarea using document.getElementById.

- **Calculate:**

Character Count: Remove all spaces using .replace(/s+/g, "") and find the length of the resulting string.

Word Count: Split the trimmed text using .split(/s+/) and count the elements, handling empty input cases.

Line Count: Split the text using \n and count the resulting lines.

Display the results in an alert.

- **Event Handling:**

Prevent the default form submission using event.preventDefault().

Intuition:

This program analyzes text by using simple string manipulation techniques:

- Characters: Spaces are removed to count meaningful characters.
- Words: Words are identified by splitting text based on whitespace.
- Lines: Lines are identified by splitting the text based on newline characters.

The logic is efficient and leverages built-in JavaScript methods for string operations. This provides a quick and interactive way for users to analyze their text input.

Code Implementation:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Text Analysis</title>
</head>
<body>
  <h1>Text Analysis Tool</h1>
  <form id="textForm" onsubmit="analyzeText(event)">
    <label for="textInput">Enter your text below:</label><br><br>
    <textarea id="textInput" rows="10" cols="50"
      placeholder="Type or paste your text here..."></textarea><br><br>
    <button type="submit">Submit</button>
  </form>

  <script>
    function analyzeText(event) {
      const text = document.getElementById("textInput").value;

      // Count characters including spaces and newline
      // const charCount = text.length;

      // Count characters excluding spaces
      const charCount = text.replace(/\s+/g, "").length;

      // Count words (split by whitespace and filter out empty strings)
      const wordCount = text.trim() ? text.trim().split(/\s+/).length : 0;

      // Count lines (split by newline and filter out empty strings)
      const lineCount = text ? text.split(/\n/).length : 0;

      // Display the results in an alert
      alert(`Characters: ${charCount}\nWords: ${wordCount}\nLines: ${lineCount}`);
    }
  </script>
</body>
</html>
```

Sample Output:

Text Analysis Tool

Enter your text below:

Type or paste your text here...

Submit

Text Analysis Tool

Enter your text below:

Good Morning
This is word count program

Submit

This page says

Characters: 33

Words: 7

Lines: 2

OK

Outcome of the Exercise:

When the user interacts with the program, the following steps occur:

- **User Input:**

The user types or pastes text into the textarea input field on the web page.

- **Text Submission:**

The user clicks the **Submit** button.

- **JavaScript Analysis:**

The analyzeText function is triggered.

It calculates:

- **Number of characters (excluding spaces):** Removes all spaces and newline characters before counting.
- **Number of words:** Splits the text based on whitespace and counts the resulting words.
- **Number of lines:** Splits the text using the newline character \n.

Viva/Interview Questions:

1. What is the purpose of the replace method in JavaScript?

	<p>The replace() method in JavaScript is used to find and replace a substring or a pattern in a string with another string or value. string.replace(searchValue, newValue) searchValue: The value or regular expression to search for. newValue: The replacement string or function.</p> <p>2. How does the split method work in the script? The split() method splits a string into an array based on a specified delimiter. In this script: text.trim().split(/\s+/) splits the trimmed text into an array of words using one or more whitespace characters as the delimiter.</p> <p>3. What is the trim() method, and why is it used here? The trim() method removes leading and trailing whitespace (spaces, tabs, and newlines) from a string. It is used in this script to ensure accurate word counts by eliminating unnecessary spaces.</p> <p>4. How are the results displayed to the user? The results are displayed using a JavaScript alert box.</p> <p>5. How are words counted in the program? Words are counted by splitting the trimmed text using the regular expression \s+, which matches one or more whitespace characters. The length of the resulting array gives the word count.</p> <p>6. How does the program handle empty lines? Lines are counted by splitting the text using the newline character \n. even empty lines are counted as separate lines.</p> <p>7. How does the program handle cases where the input is only whitespace? If the input is only whitespace, the program trims the text, resulting in an empty string. It correctly calculates 0 for both the word and character counts.</p> <p>8. What are the default rows and columns of the <textarea>? The default size of a <textarea> is 2 rows and 20 columns. In this program, it is explicitly set to 10 rows and 50 columns.</p> <p>9. Can this program count special characters like punctuation? Yes, special characters like punctuation marks are included in the character count but are not treated as separate words.</p> <p>10. How can the program be modified to include spaces in the character count? To include spaces, replace const charCount = text.replace(/\s+/g, "").length; with const charCount = text.length;.</p>
5	<p>Develop and demonstrate JavaScript with POP-UP boxes and functions for the following problems:</p> <p>a) Input: Click on Display Date button using onclick() function Output: Display date in the textbox</p> <p>b) Input: A number n obtained using prompt Output: Factorial of n number using alert</p> <p>c) Input: A number n obtained using prompt Output: A multiplication table of numbers from 1 to 10 of n using alert</p> <p>d) Input: A number n obtained using prompt and add Output: display sum in document</p> <p>Problem Statement:</p> <p>a) Display Date: Implement a button that, when clicked, will display the current date in a textbox.</p>

- b) Factorial Calculation:** Take a number n as input using a prompt() and calculate the factorial of n, displaying the result in an alert() box.
- c) Multiplication Table:** Take a number n as input via prompt(), then display the multiplication table of n (from 1 to 10) in an alert() box.
- d) Sum Calculation:** Take a number n using prompt(), add a constant value (e.g., 5), and display the result directly in the document. .

Solution Overview:

For the Display Date function, we use the Date() function to get the current date and set it in a textbox when the button is clicked. For the Factorial function, a loop or recursion is used to calculate the factorial, and the result is shown using alert(). In the Multiplication Table task, a loop runs from 1 to 10 and calculates the table for a given number, displaying each result in an alert(). For the Sum Calculation, we add a predefined number to the input and display the sum in the document using document.write().

Intuition:

The core idea behind this is to practice basic DOM manipulation and user interaction via JavaScript's prompt(), alert(), and document.write() methods. For the date display, JavaScript's Date() is used to access the system's current date. The factorial and multiplication tasks use loops to handle simple arithmetic operations. The sum is calculated by directly manipulating the input and displaying the result.

Code Implementation:

The code provides a simple interactive webpage where users can perform different tasks with JavaScript functions. The Display Date button uses the Date() function to fetch the current date and display it in a textbox. The Factorial function calculates the factorial of a number n using a for loop and shows the result in an alert(). The Multiplication Table generates the multiplication table of the input number n from 1 to 10 using a loop and displays it in an alert, while the Sum Calculation adds a fixed value (e.g., 5) to the user's input and outputs the result in the document using document.write().

5a) Input: Click on Display Date button using onclick() function**Output: Display date in the textbox**

```
<!DOCTYPE html>
<html>
<head>
  <title>Display Date Example</title>
  <script>
    // Function to display the current date in the textbox
    function displayDate() {
      // Get the current date and time
      const currentDate = new Date().toLocaleString();

      // Display it in the textbox
      document.getElementById("dateTextbox").value = currentDate;
    }
  </script>
</head>
<body>
  <h2>Display Current Date</h2>
  <!-- Button to trigger the onclick function -->
  <button onclick="displayDate()">Display Date</button>
  <!-- Textbox to display the date -->
  <br><br>
  <input type="text" id="dateTextbox" placeholder="Date will be displayed here" readonly>
</body>
</html>
```

Display Current Date Display Current Date

5b) Input: A number n obtained using prompt

Output: Factorial of n number using alert

5b_factorial.html

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="UTF-8">
```

```
<title>Run JavaScript</title>
```

```
</head>
```

```
<body>
```

```
<h1>JavaScript Execution Example</h1>
```

```
<script src="5b_factorial.js"></script>
```

```
</body>
```

```
</html>
```

5b_factorial.js

```
function factorial(num) {
```

```
  if (num < 0) {
```

```
    return "Factorial not defined for negative numbers.";
```

```
  }
```

```
  let result = 1;
```

```
  for (let i = 1; i <= num; i++) {
```

```
    result *= i;
```

```
  }
```

```
  return result;
```

```
}
```

```
n = parseInt(prompt("Enter a number to find its factorial:"));
```

```
factorialResult = factorial(n);
```

```
alert("The factorial of " + n + " is: " + factorialResult);
```

Enter a number to find its factorial:

The factorial of 5 is: 120

5c) Input: A number n obtained using prompt

Output: A multiplication table of numbers from 1 to 10 of n using alert

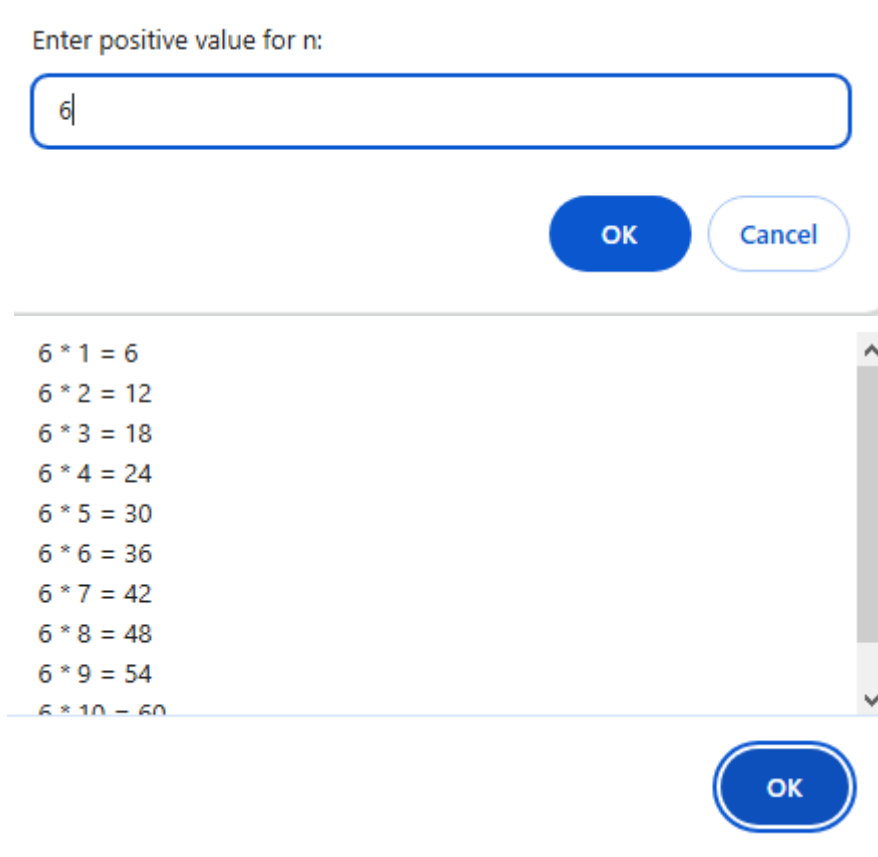
Program :

```
<html>
```

```
<head><title> Multiplication Table </title></head>
```

```
<body>
<script type="text/javascript">
var n=prompt("Enter positive value for n: "," ");
if(!isNaN(n))
{
    var table="";
    var number="";
    for(i=1;i<=10;i++) {
        number = n * i;
        table += n + " * " + i + " = " + number + "\n";
    }
    alert(table);
}
else {
    alert("Enter positive value");
    n=prompt("Enter positive value for n: "," ");
}
document.write(n+" table values displayed using alert ..<br />");
/* the * operator for example implicitly performs this coercion, so you wouldn't need the parseInt */
</script>
</body>
</html>
```

Output:



Enter positive value for n:

6

OK Cancel

6 * 1 = 6
6 * 2 = 12
6 * 3 = 18
6 * 4 = 24
6 * 5 = 30
6 * 6 = 36
6 * 7 = 42
6 * 8 = 48
6 * 9 = 54
6 * 10 = 60

OK

5d) Input: A number n obtained using prompt and add
Output: display sum in document

```
<html>
<head><title>sum of n numbers using popup boxes</title>
```

```
<script type="text/javascript">
function addsum()
{
    alert("Enter the numbers to be added");
    var keepgoing = true;
    var sumofnums = 0;
    while (keepgoing) {
        sumofnums = sumofnums + (parseInt(prompt("what's the next number to add?","")));

        keepgoing = confirm("add another number?") ;
    }
    alert("the sum of all your numbers is " + sumofnums) ;
}
</script> </head>
<body> <form name=frm>
<input type="button" value='sum of n numbers' onclick="addsum();">
</form> </body> </html>
```

sum of n numbers

Enter the numbers to be added

OK

what's the next number to add?

5|

OK

Cancel

add another number?

OK

Cancel

what's the next number to add?

6|

OK

Cancel

the sum of all your numbers is 11

OK

Outcome of the Exercise:

By completing this exercise, you'll learn how to use JavaScript to interact with the user through `prompt()` and `alert()`, manipulate the DOM with `document.write()`, and execute basic arithmetic operations like calculating a factorial and generating multiplication tables. You'll also practice event-driven programming through the `onclick()` function.

Interview Questions:**1. What does the `onclick()` function do in this program?**

Answer: The `onclick()` function triggers an event handler when the associated button is clicked. It runs the JavaScript function specified within the parentheses.

2. How does the `Date()` function work in the context of the "Display Date" problem?

Answer: The `Date()` function returns the current date and time of the system. It is used here to get the current date and display it in the textbox when the button is clicked.

3. What is the purpose of the `prompt()` function?

Answer: The `prompt()` function displays a dialog box that asks the user for input. It returns the value entered by the user as a string.

4. How is the factorial calculated in the code?

Answer: The factorial is calculated by initializing a variable `result` to 1, then multiplying it by each integer from 1 to `n` using a `for` loop.

5. What will happen if a non-numeric value is entered in the prompt for factorial or multiplication?

Answer: If a non-numeric value is entered, it may result in incorrect behavior or an error. This can be mitigated by adding input validation.

6. Why do we use `alert()` to display the result in the multiplication table and factorial problems?

Answer: `alert()` is used to display the result in a pop-up dialog box to grab the user's attention immediately.

7. What does the `document.write()` function do in the "Sum Calculation" problem?

Answer: `document.write()` writes the output directly to the HTML document, displaying the sum on the webpage.

8. How does the program handle user input for the multiplication table?

Answer: The program takes a number input through `prompt()`, then uses a `for` loop to calculate the products from 1 to 10 and formats the result for display.

9. Can you explain the importance of using `parseInt()` in the factorial and multiplication table functions?

Answer: `parseInt()` converts the string input from `prompt()` into an integer, which is necessary for mathematical operations like multiplication or calculating factorial.

10. What happens if the user cancels the prompt or enters an invalid input?

Answer: If the user cancels the prompt or enters invalid input, the program may not work as expected. Input validation can be added to handle such cases gracefully.

6 Develop and demonstrate the usage of inline, internal and external style sheet using CSS**Problem Statement:**

The task is to develop and demonstrate the usage of inline, internal, and external stylesheets using CSS to style an HTML document. The implementation should include examples showcasing each method's application and highlight their differences. By illustrating how these methods can be used effectively, the goal is to provide a comprehensive understanding of CSS styling techniques.

Solution Overview:

The solution involves using inline styles directly within HTML elements, internal styles within a <style> tag in the <head> section, and external styles linked through a separate CSS file. This approach demonstrates how CSS can be used in different contexts to achieve consistent and effective styling of web pages.

Intuition:

Using inline CSS provides a quick way to style individual elements, internal CSS offers centralized styling within a single HTML document, and external CSS allows for reusability and easier maintenance by separating content from design. Together, these methods form a comprehensive understanding of CSS styling techniques.

Code Implementation:

This HTML document demonstrates the use of inline, internal, and external CSS styles to style elements on a webpage. Inline CSS is applied directly within an HTML element (e.g., the first paragraph), internal CSS is defined within the <style> tag in the <head> section (e.g., styles for the second paragraph and the body), and external CSS is linked via an external stylesheet (e.g., styling for the third paragraph). Together, these methods showcase various approaches to applying CSS.

6.html

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>CSS Style Demonstration</title>

  <!-- Internal CSS -->
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: lightblue;
    }

    h1 {
      color: darkblue;
    }

    .internal-style {
      color: green;
      font-size: 20px;
    }
  </style>

  <!-- External CSS link -->
  <link rel="stylesheet" href="6_style.css">
</head>
<body>

  <h1>Demonstration of CSS Styles</h1>
  <!-- Inline CSS -->
  <p style="color: red; font-size: 18px;">This is a paragraph with inline CSS styling.</p>

  <p class="internal-style">This paragraph uses internal CSS (from the style tag in head).</p>

  <p class="external-style">This paragraph uses external CSS (from the linked stylesheet).</p>

</body>
</html>
```

6_style.css

```
.external-style {  
    color: purple;  
    font-size: 22px;  
    font-weight: bold;  
}
```

Sample Output:

Demonstration of CSS Styles

This is a paragraph with inline CSS styling.

This paragraph uses internal CSS (from the style tag in head).

This paragraph uses external CSS (from the linked stylesheet).

Outcome of the Exercise:

The implementation and demonstration of inline, internal, and external CSS styles yielded valuable insights into their usage and effectiveness. The key observations and outcomes are summarized below:

- **Demonstration of CSS Techniques:**

The exercise successfully showcased the three main methods of applying CSS: inline, internal, and external styles. Each method was demonstrated with practical examples, highlighting their respective syntax and behavior.

- **Visual Differentiation:**

The effects of each CSS method were clearly visible on the webpage:

Inline CSS: Styled a specific paragraph directly within the HTML, enabling quick and localized styling.

Internal CSS: Provided centralized styles for multiple elements within the document, demonstrating structured and reusable styling.

External CSS: Allowed for consistent styling through a linked stylesheet, emphasizing reusability across multiple web pages.

- **Separation of Concerns:**

The exercise illustrated the importance of separating content (HTML) and presentation (CSS). The use of external CSS, in particular, reinforced the best practice of keeping styles modular and manageable.

- **Practical Application:**

The styled webpage demonstrated real-world scenarios where these CSS methods can be applied:

Inline CSS for quick adjustments.

Internal CSS for project-specific styles.

External CSS for shared styles in multi-page applications.

- **Learning and Understanding:**

By implementing and observing the effects of each CSS method, the exercise provided a hands-on learning experience. It highlighted the trade-offs between the simplicity of inline CSS, the organization of internal CSS, and the scalability of external CSS.

- **Customization and Flexibility:**

The flexibility to modify styles through different approaches allowed for experimentation and understanding of how these methods can be combined effectively to achieve desired designs.

Viva/Interview Questions:

1. What is the purpose of CSS in web development?

Answer: CSS (Cascading Style Sheets) is used to control the presentation, layout, and design of web pages, allowing developers to style HTML elements and create visually appealing websites.

2. What are the main differences between inline, internal, and external CSS?

Answer: Inline CSS is applied directly to an HTML element using the style attribute.

Internal CSS is defined within a <style> tag in the <head> section of the HTML document.

External CSS is stored in a separate file and linked to the HTML document using the <link> tag.

3. What are the advantages and disadvantages of inline CSS?

Answer: Advantages: Quick to apply and useful for small, localized styling.

Disadvantages: Difficult to maintain, not reusable, and can clutter HTML code.

4. Why is internal CSS less scalable compared to external CSS?

Answer: Internal CSS is embedded within a single HTML document, which limits its reusability. For multi-page websites, each page would require its own <style> section, making it harder to maintain and update.

5. What is the cascading order in CSS, and how does it affect the application of styles?

Answer: The cascading order determines how CSS rules are applied when there are conflicts. Priority is given in the following order:

- Inline styles (highest priority).
- Internal and external styles based on specificity.
- Browser default styles (lowest priority).

6. How does the separation of content and style benefit web development?

Answer: Separating content (HTML) and style (CSS) enhances code readability, makes it easier to maintain and update styles without affecting the structure, and allows for consistent design across multiple pages.

7. What is the role of a CSS class selector, and how is it used?

Answer: A CSS class selector is used to define styles for a group of elements. It is applied by prefixing a dot (.) to the class name in the CSS and adding the class attribute to HTML elements.

8. What are some scenarios where you might prefer inline CSS over other methods?

Answer: Inline CSS is useful for quick styling of single elements, debugging or testing specific styles, and overriding other styles in unique cases.

9. How can external CSS improve the performance of a website?

Answer: External CSS reduces redundancy by centralizing styles in one file, enabling browser caching, and allowing multiple pages to share the same stylesheet, leading to faster load times for subsequent pages.

10. What is the significance of specificity in CSS, and how is it calculated?

Answer: Specificity determines which CSS rule is applied when multiple rules match the same element. It is calculated based on the number of ID selectors (highest weight), class selectors, and element selectors (lowest weight) in the rule.

- 7** **Develop and demonstrate, using Javascript script, a XHTML document that contains three short paragraphs of text, stacked on top of each other, with only enough of each showing so that the mouse cursor can be placed over some part of them. When the cursor is placed over the exposed part of any paragraph, it should rise to the top to become completely visible.**

Problem Statement:

Create an XHTML document containing three short paragraphs of text stacked on top of each other, with only a portion of each visible. When the mouse cursor is placed over any exposed part of the paragraphs, that paragraph should rise to the top and become fully visible. This interaction will make use of JavaScript to dynamically change the visibility of the paragraphs based on mouse position.

Solution Overview:

The solution involves using HTML for structuring the content, CSS for positioning the paragraphs and hiding parts of them, and JavaScript for detecting mouse interaction. The paragraphs will be stacked using absolute or relative positioning with z-index to manage which one appears on top. Upon hovering over any visible portion of a paragraph, JavaScript will change its z-index to make it fully visible.

Intuition:

The core idea is to use mouse hover events to trigger dynamic visibility changes. Initially, parts of each paragraph are hidden using CSS overflow: hidden. By manipulating the z-index dynamically in JavaScript when hovering over the exposed portion of a paragraph, we ensure that the hovered paragraph moves to the top and becomes fully visible.

Code Implementation:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Interactive Paragraphs</title>
  <style>
    /* Basic styling for each paragraph */
    .para {
      position: absolute;
      width: 300px;
      height: 50px;
      overflow: hidden;
      padding: 10px;
      border: 2px solid #000;
      font-size: 18px;
      transition: all 0.3s ease;
    }
    #para1 {
      top: 100px;
      left: 200px;
      background-color: lightblue;
    }
    #para2 {
      top: 140px;
      left: 220px;
      background-color: lightgreen;
    }
    #para3 {
      top: 180px;
      left: 240px;
      background-color: lightcoral;
    }
  </style>
</head>
<body>

  <!-- Three stacked paragraphs -->
  <div class="para" id="para1" onmouseover="bringToFront('para1')">
    This is the first paragraph. It is partially hidden.
  </div>
  <div class="para" id="para2" onmouseover="bringToFront('para2')">
    This is the second paragraph, slightly hidden under the first one.
  </div>
  <div class="para" id="para3" onmouseover="bringToFront('para3')">
    This is the third paragraph, hidden behind the other two.
```

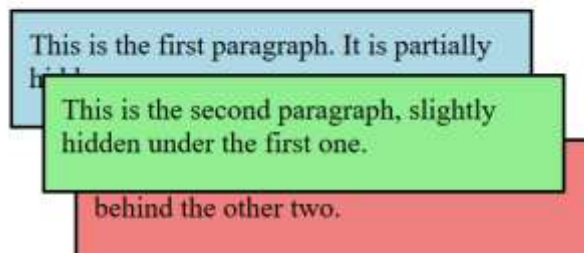
```
</div>

<script>
  // Variable to track the topmost paragraph
  var topLayer = "";

  // Function to bring the paragraph to the front
  function bringToFront(paraId) {
    // Reset the z-index of the previously topmost paragraph
    if (topLayer && topLayer !== paraId) {
      document.getElementById(topLayer).style.zIndex = "";
    }

    // Bring the hovered paragraph to the front
    document.getElementById(paraId).style.zIndex = "10";
    topLayer = paraId;
  }
</script>

</body>
</html>
```

Sample output:

b) Modify the above document so that when a paragraph is moved from the top stacking position, it returns to its original position rather than to the bottom.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Interactive Paragraphs</title>
  <style>
    /* Basic styling for each paragraph */
    .para {
      position: absolute;
      width: 300px;
```

```
height: 50px;
overflow: hidden;
padding: 10px;
border: 2px solid #000;
font-size: 18px;
transition: all 0.3s ease;
}
#para1 {
top: 100px;
left: 200px;
background-color: lightblue;
z-index: 1;
}
#para2 {
top: 140px;
left: 220px;
background-color: lightgreen;
z-index: 2;
}
#para3 {
top: 180px;
left: 240px;
background-color: lightcoral;
z-index: 3;
}
</style>
</head>
<body>

<!-- Three stacked paragraphs -->
<div class="para" id="para1" onmouseover="bringToFront('para1')" onmouseout="moveBack('para1')">
  This is the first paragraph. It is partially hidden.
</div>
<div class="para" id="para2" onmouseover="bringToFront('para2')" onmouseout="moveBack('para2')">
  This is the second paragraph, slightly hidden under the first one.
</div>
<div class="para" id="para3" onmouseover="bringToFront('para3')" onmouseout="moveBack('para3')">
  This is the third paragraph, hidden behind the other two.
</div>

<script>
  // Store the original z-index values of each paragraph
  const originalZIndexes = {
    para1: 1,
    para2: 2,
    para3: 3
  };

  // Track the topmost paragraph to manage its state
  var topLayer = "";

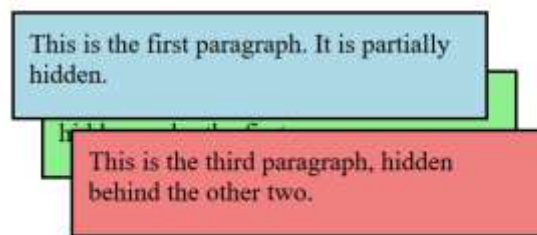
  // Function to bring the paragraph to the front
  function bringToFront(paraId) {
    // Only bring to front if it's not already on top
    if (topLayer !== paraId) {
      // Move the previously top paragraph back to its original z-index
      if (topLayer) {
        document.getElementById(topLayer).style.zIndex = originalZIndexes[topLayer];
      }
      // Bring the hovered paragraph to the top

```

```
document.getElementById(paraId).style.zIndex = 10;
topLayer = paraId;
}
}

// Function to move the paragraph back to its original position
function moveBack(paraId) {
    // Move the paragraph back to its original z-index
    document.getElementById(paraId).style.zIndex = originalZIndexes[paraId];
    // Reset the topLayer if the mouse has left the paragraph
    if (topLayer === paraId) {
        topLayer = "";
    }
}
}
</script>

</body>
</html>
```

Sample output:**Outcome of the Exercise:**

This exercise demonstrates how to dynamically manipulate HTML elements based on user interaction using JavaScript. The result is an interactive set of paragraphs that only show a part of each paragraph initially. When the mouse hovers over any part of the paragraph, it brings the full paragraph to the front, allowing users to read it fully. This technique can be applied to other interactive content layouts.

Interview Questions:**1. What is the purpose of the onmouseover event in this code?**

Answer: The onmouseover event triggers the bringToFront() function when the mouse cursor hovers over any visible portion of the paragraph. It is used to detect when the user interacts with a paragraph.

2. Why is position: relative; used in the CSS for the paragraphs?

Answer: position: relative; is used to allow the element's position to be controlled and manipulated via the z-index property, which helps in stacking the paragraphs.

3. What does the z-index property do in this context?

Answer: The z-index property controls the stacking order of the paragraphs. A higher z-index value ensures that the hovered paragraph is displayed on top of the others.

4. Why is overflow: hidden; used for the paragraphs?

Answer: overflow: hidden; is used to hide the parts of the paragraphs that are outside their container, making them appear as partially visible stacks.

5. What happens when the bringToFront() function is triggered?

Answer: When triggered, bringToFront() changes the z-index of the hovered paragraph to 10, bringing it to the front. After 500ms, the z-index is reset to its default value.

6. Why is a setTimeout() function used in bringToFront()?

Answer: setTimeout() is used to reset the z-index of the paragraph back to its original state after 500ms, ensuring that it doesn't remain on top after the hover is finished.

7. How does the interaction differ if the setTimeout() is removed?

Answer: Without setTimeout(), the paragraph will stay on top permanently after the mouse hover, rather than returning to its original position after a brief delay.

8. How can you modify the code to make the paragraphs fully visible when hovered over without using z-index?

Answer: Instead of using z-index, you can animate the height of the paragraphs to expand them when hovered over, making them fully visible without stacking them on top.

9. What is the role of the div element in this code?

Answer: The div elements are used to wrap the paragraphs and define areas that will be shown or hidden based on the user's interaction with them.

10. Can you explain the concept of stacking context in relation to z-index?

Answer: Stacking context refers to the order in which elements are rendered on the page. The z-index controls the stacking of elements within the same stacking context, meaning elements with higher z-index values will appear on top of elements with lower z-index values.

8 Write a XHTML Program using Java script to create a web page with images where these images navigate between images using:

- click/arrow button
- automation
- Text/Image

Problem Statement:

The task is to Create a web page using XHTML and JavaScript that displays a series of images in a slider format. The application should support image navigation, automation and styling. The interface should provide multiple mouse click events for navigation. The focus is on using JavaScript to demonstrate dynamic DOM manipulation while maintaining strict XHTML compliance.

Solution Overview:

The solution involves developing a webpage with the following key components:

Use XHTML for the structure of the web page.

Use JavaScript to handle the image navigation and automation.

Use CSS for styling the image slider and navigation buttons.

Intuition:

The goal is to create an interactive experience where paragraphs are stacked on top of each other and initially only partially visible. Using JavaScript, we can manipulate the z-index of each paragraph to bring the hovered one to the front, making it fully visible. By updating the z-index on hover, we ensure that the currently

hovered paragraph rises above the others while maintaining smooth transitions. This interaction helps users focus on one paragraph at a time, making the content more readable.

Code Implementation:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Image Slider</title>
  <style>
    #slider {
      position: relative;
      width: 600px;
      height: 400px;
      overflow: hidden;
    }
    #slider img {
      width: 100%;
      height: 100%;
      display: none;
    }
    #slider img.active {
      display: block;
    }
    .nav-button {
      position: absolute;
      top: 50%;
      transform: translateY(-50%);
      background-color: rgba(0, 0, 0, 0.5);
      color: white;
      border: none;
      padding: 10px;
      cursor: pointer;
    }
    #prev {
      left: 10px;
    }
    #next {
      right: 10px;
    }
  </style>
</head>
<body>
  <div id="slider">
    
    
    
    <button id="prev" class="nav-button">Prev</button>
    <button id="next" class="nav-button">Next</button>
  </div>
  <div>
    <button onclick="showImage(0)">Image 1</button>
    <button onclick="showImage(1)">Image 2</button>
    <button onclick="showImage(2)">Image 3</button>
  </div>
  <script>
    var currentIndex = 0;
    var images = document.querySelectorAll('#slider img');
    var totalImages = images.length;
```

```
function showImage(index) {  
    images[currentIndex].classList.remove('active');  
    currentIndex = index;  
    images[currentIndex].classList.add('active');  
}  
  
document.getElementById('prev').onclick = function() {  
    showImage((currentIndex - 1 + totalImages) % totalImages);  
};  
  
document.getElementById('next').onclick = function() {  
    showImage((currentIndex + 1) % totalImages);  
};  
  
function autoSlide() {  
    showImage((currentIndex + 1) % totalImages);  
}  
  
setInterval(autoSlide, 3000); // Change image every 3 seconds  
</script>  
</body>  
</html>
```

Sample Output:



Prev

Next

Image 1

Image 2

Image 3



Prev

Next

Image 1

Image 2

Image 3

Outcome of the Exercise:

	<p>The program provides a dynamic and interactive web interface where users can manipulate the content of a webpage in real-time. Users can:</p> <ul style="list-style-type: none"> • click/arrow button • automation • Text/Image <p>By completing this program, users gain practical experience with:</p> <ul style="list-style-type: none"> • JavaScript-based DOM manipulation techniques (innerHTML/innerText). • Event-driven programming to handle user interactions. • XHTML for creating well-structured, browser-compatible web pages. • This program demonstrates how to create responsive, real-time interfaces, forming the basis for developing more advanced web applications. <p>Viva/Interview Questions:</p> <ol style="list-style-type: none"> 1. What is Click/Arrow Button Navigation Answer: Use the left and right arrows to move between images. 2. What is Text/Image Navigation Text/Image Navigation Answer: Click the text buttons under the images to navigate directly to a specific image. 3. What is Automated Slideshow Answer: Images change automatically every 3 seconds 4. What is JavaScript? Answer: JavaScript is a high-level, interpreted programming language that is primarily used for creating and controlling dynamic web content. It can update and change HTML and CSS, and can calculate, manipulate, and validate data. 5. Is it possible to break JavaScript Code into several lines? Answer: Breaking within a string statement can be done by using a backslash, '\,' at the end of the first line. 6. What is a prompt box? Answer: A prompt box is a box that allows the user to enter input by providing a text box. 7. What would be the result of 3+2+"7"? Answer: Since 3 and 2 are integers, they will be added numerically. And since 7 is a string, its concatenation will be done. So the result would be 57. 8. What is the fullform of DOM Answer: Document Object Model (DOM) 9. How does the image navigation work using click buttons? Answer: The navigation works by changing the index variable when the "Next" or "Previous" button is clicked. 10. What would happen if the clearInterval function is not called? Answer: If clearInterval is not called, the automated navigation will continue to run indefinitely, switching images every 2 seconds, regardless of any manual interactions.
9	<p>Write a XHTML Program using Java script to create multiple line of content. Use various buttons to</p> <ul style="list-style-type: none"> • Adding new content • Replacing the existing content • Deleting the Content <p>Problem Statement:</p>

The task is to create a dynamic web application using XHTML and JavaScript that allows users to manipulate a content area interactively. The application should support adding new lines of content, replacing all existing content with new input, and clearing all content entirely. The interface should include buttons for each operation and an input field for entering the desired content. The focus is on using JavaScript to demonstrate dynamic DOM manipulation while maintaining strict XHTML compliance.

Solution Overview:

The solution involves developing a webpage with the following key components:

- **XHTML Structure:** A strict and well-structured XHTML document containing a content area, an input field, and buttons for adding, replacing, and deleting content.
- **JavaScript for Interactivity:** Functions are implemented to handle the three operations:
- **Add Content:** Appends the user input as a new line to the existing content.
- **Replace Content:** Replaces the entire content with the user input.
- **Delete Content:** Clears all the content in the content area.
- **Styling:** Basic CSS is used to enhance the usability and appearance of the application.
- **Edge Case Handling:** Input validation ensures users cannot perform actions like adding empty content. The application uses innerHTML or innerText for DOM manipulation to simplify the implementation, making the program efficient and lightweight.

Intuition:

Dynamic content manipulation enhances user interactivity by enabling real-time updates without reloading the page. Using innerHTML or innerText simplifies DOM manipulation, ensuring a responsive and intuitive interface for managing content efficiently.

Code Implementation:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Content Manipulation</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
    }
    #content {
      border: 1px solid #ccc;
      padding: 10px;
      margin-bottom: 10px;
      min-height: 100px;
      white-space: pre-line; /* To display new lines */
    }
    button {
      margin-right: 10px;
    }
  </style>
  <script type="text/javascript">
    // Function to add new content
    function addContent() {
      const contentDiv = document.getElementById("content");
      const newContent = document.getElementById("inputText").value;
      if (newContent.trim() === "") {
        alert("Please enter some content.");
        return;
      }
      contentDiv.innerHTML += newContent + "\n"; // Append new content with a newline
      document.getElementById("inputText").value = ""; // Clear input field
    }
  </script>
</html>
```

```
}

// Function to replace existing content
function replaceContent() {
    const contentDiv = document.getElementById("content");
    const newContent = document.getElementById("inputText").value;
    if (newContent.trim() === "") {
        alert("Please enter some content.");
        return;
    }
    contentDiv.innerHTML = newContent; // Replace entire content
    document.getElementById("inputText").value = ""; // Clear input field
}

// Function to delete all content
function deleteContent() {
    const contentDiv = document.getElementById("content");
    contentDiv.innerHTML = ""; // Clear all content
}
</script>
</head>
<body>
    <h1>Content Manipulation</h1>
    <div id="content"></div>
    <input type="text" id="inputText" placeholder="Enter your content here" />
    <button onclick="addContent()">Add Content</button>
    <button onclick="replaceContent()">Replace Content</button>
    <button onclick="deleteContent()">Delete Content</button>
</body>
</html>
```

Sample Output:

Content Manipulation

Hello

Enter your content here

Add Content

Replace Content

Delete Content

Content Manipulation

Good afternoon

Enter your content here

Add Content

Replace Content

Delete Content

Content Manipulation



Outcome of the Exercise:

The program provides a dynamic and interactive web interface where users can manipulate the content of a webpage in real-time. Users can:

1. Add Content: Append new lines of text to the existing content seamlessly.
2. Replace Content: Replace all current content with a single input, effectively overwriting it.
3. Delete Content: Clear all content in the designated area instantly.

By completing this program, users gain practical experience with:

- JavaScript-based DOM manipulation techniques (innerHTML/innerText).
- Event-driven programming to handle user interactions.
- XHTML for creating well-structured, browser-compatible web pages.
- This program demonstrates how to create responsive, real-time interfaces, forming the basis for developing more advanced web applications.

Viva/Interview Questions:

1. What is the difference between innerHTML and innerText in JavaScript

Answer: innerHTML sets or retrieves the HTML content of an element, including HTML tags.

innerText sets or retrieves only the text content, ignoring any HTML tags and preserving the text as plain content.

2. Why did we use XHTML instead of plain HTML in this program?

Answer: XHTML enforces stricter syntax rules, such as properly closing all tags and maintaining case sensitivity. This ensures better compatibility with XML-based parsers and consistent rendering across browsers.

3. How does JavaScript handle DOM manipulation in this application?

Answer: JavaScript interacts with the DOM using methods like innerHTML or innerText to update the content of an element dynamically. It also uses event listeners to trigger these updates when buttons are clicked.

4. How does the concept of event-driven programming apply to this application?

Answer: Event-driven programming involves responding to user actions, such as button clicks. In this application, event listeners are attached to buttons, triggering specific JavaScript functions to dynamically manipulate the content area based on user input.

1
0

7. Learning Resources:

Reference Books:

1. Web Programming, building internet applications, Chris Bates 2nd edition, Wiley Dremtech
2. Java Script, D.Flanagan, O'Reilly, SPD

JOURNALS/MAGAZINES:

1. <https://www.inderscience.com/jhome.php?jcode=IJWET>
2. <http://www.w3schools.com/>
3. <http://getbootstrap.com/>

SWAYAM/NPTEL/MOOCs:

1. <http://nptel.ac.in>
2. <https://www.udemy.com/course/angularjs-for-beginners-udemy/>
3. <https://www.coursera.org/learn/introduction-to-front-end-development>