

M.Sc. (Computer Applications)

Title of the Course: Web Technology Laboratory

Paper Code: CAMAEP-511

1. HTML Basics

- 1. Create a simple HTML webpage with headings, paragraphs, and links.**
Introduction to basic HTML elements.
- 2. Develop a webpage with a form that captures user input and displays the submitted data.**
Learning about forms and user input in HTML.
- 3. Implement client-side form validation using HTML5 form input types and attributes.**
Adding basic validation using HTML5.
- 4. Create a webpage with a table that dynamically populates data using JavaScript.**
Combining HTML tables with JavaScript for dynamic content.
- 5. Build a responsive image gallery using HTML, CSS, and JavaScript.**
Integrating HTML with CSS and JavaScript for responsive design.

2. CSS Basics

- 6. Design a webpage layout using CSS float and positioning techniques.**
Introduction to CSS positioning.
- 7. Implement CSS transitions to create smooth hover effects on webpage elements.**
Learning basic CSS transitions for interactive elements.
- 8. Implement CSS animations for visual effects.**
Advanced CSS animations.
- 9. Design a responsive website using CSS media queries to adapt to different screen sizes.**
Making websites responsive with media queries.
- 10. Design a webpage layout using CSS grid or flexbox for responsive design.**
Modern CSS layout techniques.
- 11. Build a website that incorporates CSS animations for visual effects.**
Applying CSS animations in a real-world scenario.

3. JavaScript Basics

- 12. Build a JavaScript calculator that performs basic arithmetic operations.**
Introduction to JavaScript and basic operations.
- 13. Create a JavaScript program that generates a random password based on user-specified criteria.**
Building a more complex JavaScript application.
- 14. Develop a JavaScript program that manipulates the Document Object Model (DOM) to dynamically update webpage content.**
Learning DOM manipulation.

15. **Develop a JavaScript slideshow that displays images with automatic transitions.**
Implementing JavaScript for dynamic content.
16. **Create a JavaScript quiz that presents multiple-choice questions and tracks the user's score.**
Interactive JavaScript applications.
17. **Build a JavaScript program that performs basic string manipulation tasks.**
Advanced JavaScript string handling.
18. **Create a JavaScript program that interacts with the browser's local storage to store and retrieve data.**
Using local storage for persistent data.
19. **Build a website that retrieves data from a web API using JavaScript and displays it dynamically.**
Working with APIs and dynamic data.

4. Advanced Concepts

20. **Develop a webpage with a contact form that sends user input to a server using AJAX.**
AJAX for asynchronous data handling.
21. **Develop a website that utilizes cookies for user session management.**
Managing user sessions with cookies.
22. **Build a website that displays images and uses image maps for client-server interaction.**
Combining images with interactive areas.
23. **Design a webpage layout using Bootstrap framework.**
Using a CSS framework for layout.
24. **Design a webpage layout using CSS framework such as Foundation or Bulma.**
Exploring different CSS frameworks.
25. **Develop a website that incorporates parallax scrolling using CSS and JavaScript.**
Creating advanced scrolling effects.
26. **Implement XML transformation using XSLT to convert an XML document into a different format.**
Advanced XML handling.
27. **Create an XML document that represents a simple data structure and validate it against a DTD.**
XML document creation and validation.
28. **Design a webpage layout using CSS float and positioning techniques.**
Applying various layout techniques.
29. **Develop a JavaScript program that manipulates the Document Object Model (DOM) to dynamically update webpage content.**
Advanced DOM manipulation.
30. **Implement CSS transforms to create visually appealing 3D effects on webpage elements.**
Using CSS for advanced visual effects.

1.Create a simple HTML webpage with headings, paragraphs, and links.

HTML Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Simple HTML Page</title>
</head>
<body>
  <header>
    <h1>Welcome to My Simple Webpage</h1>
  </header>

  <section>
    <h2>About This Page</h2>
    <p>This is a basic HTML page to demonstrate how to use headings, paragraphs, and
links.</p>
  </section>

  <section>
    <h2>Useful Links</h2>
    <p>Here are some useful links:</p>
    <ul>
      <li><a href="https://www.example.com" target="_blank">Example Website</a></li>
      <li><a href="https://www.w3schools.com" target="_blank">W3Schools</a></li>
      <li><a href="https://www.wikipedia.org" target="_blank">Wikipedia</a></li>
    </ul>
  </section>

  <footer>
    <p>Created by Me. &copy; 2024</p>
  </footer>
</body>
</html>
```

Output:

Welcome to My Simple Webpage

About This Page

This is a basic HTML page to demonstrate how to use headings, paragraphs, and links.

Useful Links

Here are some useful links:

- [Example Website](#)
- [W3Schools](#)
- [Wikipedia](#)

Created by Me. © 2024

Explanation:

1. **<!DOCTYPE html>:**
 - This declaration defines the document type and version of HTML. It tells the browser that this is an HTML5 document.
2. **<html lang="en">:**
 - The <html> element is the root of the HTML document. The lang="en" attribute specifies that the document is in English.
3. **<head>:**
 - Contains metadata about the document. This includes:
 - **<meta charset="UTF-8">:** Sets the character encoding to UTF-8, which supports many characters from various languages.
 - **<meta name="viewport" content="width=device-width, initial-scale=1.0">:** Ensures the page is responsive and scales correctly on different devices.
 - **<title>:** Sets the title of the webpage, which appears in the browser tab.
4. **<body>:**
 - Contains the content of the HTML document. It is divided into different sections:
 - **<header>:**
 - Typically contains introductory content or navigational links. Here, it includes a main heading <h1>, which is the largest and most important heading.
 - **<section>:**
 - Represents a section of content. Each section has its own heading (<h2>) and content:
 - **<h2>:** A secondary heading, less important than <h1>.
 - **<p>:** A paragraph element for text content.

- ****: An unordered list, which includes a list of links **<a>**. Each link uses the href attribute to specify the URL and target="_blank" to open the link in a new tab.
- **<footer>**:
 - Typically contains footer content like copyright information or contact details.

2. Develop a webpage with a form that captures user input and displays the submitted data.

HTML Code

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Form Submission Example</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      padding: 20px;
      max-width: 600px;
      margin: auto;
    }
    form {
      margin-bottom: 20px;
    }
    .result {
      border: 1px solid #ddd;
      padding: 10px;
      background-color: #f9f9f9;
    }
  </style>
</head>
<body>
  <h1>Form Submission Example</h1>

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

    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required><br><br>

    <label for="message">Message:</label><br>
    <textarea id="message" name="message" rows="4" cols="50"
required></textarea><br><br>

    <button type="submit">Submit</button>
  </form>

  <div id="result" class="result" style="display:none;">
    <h2>Submitted Data:</h2>
    <p><strong>Name:</strong> <span id="resultName"></span></p>
```

```

    <p><strong>Email:</strong> <span id="resultEmail"></span></p>
    <p><strong>Message:</strong> <span id="resultMessage"></span></p>
</div>

<script>
    document.getElementById('userForm').addEventListener('submit', function(event) {
        // Prevent the default form submission behavior
        event.preventDefault();

        // Get form values
        const name = document.getElementById('name').value;
        const email = document.getElementById('email').value;
        const message = document.getElementById('message').value;

        // Display the results
        document.getElementById('resultName').textContent = name;
        document.getElementById('resultEmail').textContent = email;
        document.getElementById('resultMessage').textContent = message;

        // Show the result div
        document.getElementById('result').style.display = 'block';
    });
</script>
</body>
</html>

```

Output:

Form Submission Example

Name:

Email:

Message:

Submitted Data:

Name: abc
Email: abc@gmail.com
Message: hello!

Explanation:

1. **<!DOCTYPE html>**:
 - Declares the document type and HTML version (HTML5).
2. **<html lang="en">**:
 - The root element of the HTML document. The lang="en" attribute specifies that the document is in English.
3. **<head>**:
 - Contains metadata and styles for the document:
 - **<meta charset="UTF-8">**: Sets the character encoding to UTF-8.
 - **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Ensures proper scaling on different devices.
 - **<title>**: Sets the title of the page, shown in the browser tab.
 - **<style>**: Includes internal CSS to style the page, such as font, margins, and styling for the result div.
4. **<body>**:
 - Contains the content of the webpage:
 - **<h1>**: Main heading for the page.
 - **<form id="userForm">**: Form element with an ID for JavaScript reference.
 - **<label>**: Provides labels for the form fields.
 - **<input>**: Input fields for name and email.
 - **<textarea>**: A multi-line text input for the message.
 - **<button>**: Submit button for the form.
 - **<div id="result" class="result" style="display:none;">**: Container for displaying the submitted data. Initially hidden using inline CSS.
 - **<script>**: Contains JavaScript to handle form submission:
 - **event.preventDefault()**: Prevents the form from submitting and refreshing the page.
 - **document.getElementById('name').value**: Retrieves the values entered in the form fields.
 - **document.getElementById('resultName').textContent**: Updates the content of the result div with the submitted data.
 - **document.getElementById('result').style.display = 'block';**: Shows the result div after submission.

3.Implement client-side form validation using HTML5 form input types and attributes

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>HTML5 Form Validation</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      padding: 20px;
      max-width: 600px;
      margin: auto;
    }
    form {
      margin-bottom: 20px;
    }
    .error {
      color: red;
      font-size: 0.9em;
    }
    .form-group {
      margin-bottom: 15px;
    }
  </style>
</head>
<body>
  <h1>HTML5 Form Validation Example</h1>
```

```
<form id="registrationForm">

  <div class="form-group">

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

    <input type="text" id="name" name="name" required minlength="3" maxlength="50"
placeholder="Enter your name">

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

  </div>


  <div class="form-group">

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

    <input type="email" id="email" name="email" required placeholder="Enter your
email">

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

  </div>


  <div class="form-group">

    <label for="password">Password:</label>

    <input type="password" id="password" name="password" required minlength="6"
placeholder="Enter your password">

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

  </div>


  <div class="form-group">

    <label for="age">Age:</label>

    <input type="number" id="age" name="age" required min="18" max="100"
placeholder="Enter your age">

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

  </div>


  <div class="form-group">

    <label for="website">Website:</label>
```

```
    <input type="url" id="website" name="website" placeholder="Enter your website  
(optional)">
```

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

```
</div>
```

```
<button type="submit">Submit</button>
```

```
</form>
```

```
<script>
```

```
    document.getElementById('registrationForm').addEventListener('submit',  
function(event) {
```

```
        let valid = true;
```

```
        // Validate name
```

```
        const name = document.getElementById('name');
```

```
        const nameError = document.getElementById('nameError');
```

```
        if (name.validity.valueMissing) {
```

```
            nameError.textContent = 'Name is required.';
```

```
            valid = false;
```

```
        } else if (name.validity.tooShort) {
```

```
            nameError.textContent = 'Name must be at least 3 characters long.';
```

```
            valid = false;
```

```
        } else {
```

```
            nameError.textContent = "";
```

```
        }
```

```
        // Validate email
```

```
        const email = document.getElementById('email');
```

```
        const emailError = document.getElementById('emailError');
```

```
        if (email.validity.valueMissing) {
```

```
            emailError.textContent = 'Email is required.';
```

```
    valid = false;
} else if (email.validity.typeMismatch) {
    emailError.textContent = 'Please enter a valid email address.';
    valid = false;
} else {
    emailError.textContent = "";
}

// Validate password
const password = document.getElementById('password');
const passwordError = document.getElementById('passwordError');
if (password.validity.valueMissing) {
    passwordError.textContent = 'Password is required.';
    valid = false;
} else if (password.validity.tooShort) {
    passwordError.textContent = 'Password must be at least 6 characters long.';
    valid = false;
} else {
    passwordError.textContent = "";
}

// Validate age
const age = document.getElementById('age');
const ageError = document.getElementById('ageError');
if (age.validity.valueMissing) {
    ageError.textContent = 'Age is required.';
    valid = false;
} else if (age.validity.rangeUnderflow) {
    ageError.textContent = 'Age must be at least 18.';
    valid = false;
}
```

```
    } else if (age.validity.rangeOverflow) {  
        ageError.textContent = 'Age must be less than or equal to 100.';  
        valid = false;  
    } else {  
        ageError.textContent = "";  
    }  
  
    // Validate website  
    const website = document.getElementById('website');  
    const websiteError = document.getElementById('websiteError');  
    if (website.validity.typeMismatch) {  
        websiteError.textContent = 'Please enter a valid URL.';  
        valid = false;  
    } else {  
        websiteError.textContent = "";  
    }  
  
    // If form is invalid, prevent submission  
    if (!valid) {  
        event.preventDefault();  
    }  
});  
</script>  
</body>  
</html>
```

Output:

HTML5 Form Validation Example

Name:



Please lengthen this text to 3 characters or more (you are currently using 1 character).

Password:


Age:

Website:

HTML5 Form Validation Example

Name:

Email:

 Please include an '@' in the email address. 'Abc' is missing an '@'.

Age:

Website:

HTML5 Form Validation Example

Name:

Email:

Password:

 Please lengthen this text to 6 characters or more (you are currently using 5 characters).

Website:


HTML5 Form Validation Example

Name:

Email:

Password:

Age:

 Please fill out this field. te (optional)

Explanation:

1. **<!DOCTYPE html>**:
 - Declares the document type and HTML version.
2. **<html lang="en">**:
 - Root element with language set to English.
3. **<head>**:
 - Contains metadata and styles:
 - **<meta charset="UTF-8">**: Character encoding.
 - **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Responsive design.
 - **<title>**: Page title.
 - **<style>**: Internal CSS for styling the form and error messages.
4. **<body>**:
 - Contains the main content:
 - **<form id="registrationForm">**: Form element with various input fields.
 - **<input type="text" id="name" name="name" required minlength="3" maxlength="50" placeholder="Enter your name">**: Text input with validation attributes.
 - **<input type="email" id="email" name="email" required placeholder="Enter your email">**: Email input with required validation.
 - **<input type="password" id="password" name="password" required minlength="6" placeholder="Enter your password">**: Password input with minimum length.
 - **<input type="number" id="age" name="age" required min="18" max="100" placeholder="Enter your age">**: Number input with range validation.
 - **<input type="url" id="website" name="website" placeholder="Enter your website (optional)">**: URL input for optional website.
 - ****: Error message placeholders for each field.
 - **<button type="submit">Submit</button>**: Submit button.
5. **<script>**:

- Contains JavaScript for custom form validation:
 - **event.preventDefault():** Prevents default form submission if validation fails.
 - **document.getElementById('name').validity:** Checks validity states for each field.
 - **textContent:** Updates error messages based on the validation result.

Key HTML5 Validation Attributes:

- **required:** Makes the field mandatory.
- **minlength:** Specifies the minimum length for text fields.
- **maxlength:** Specifies the maximum length for text fields.
- **type="email":** Validates that the input is a properly formatted email address.
- **type="password":** Used for password fields with additional validation like minlength.
- **type="number":** Validates that the input is a number and allows range constraints with min and max.
- **type="url":** Ensures that the input is a valid URL.

4. Create a webpage with a table that dynamically populates data using JavaScript.

1. HTML File (index.html)

This file will include the structure of the webpage and link to the external JavaScript file.

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Dynamic Table Example</title>

  <style>
    body {
      font-family: Arial, sans-serif;
      padding: 20px;
      max-width: 800px;
      margin: auto;
    }
    table {
      width: 100%;
      border-collapse: collapse;
    }
    table, th, td {
      border: 1px solid #ddd;
    }
    th, td {
      padding: 8px;
      text-align: left;
    }
    th {
      background-color: #f4f4f4;
    }
  </style>
</head>

<body>
  <table>
    <tr>
      <th>ID</th>
      <th>Name</th>
      <th>Age</th>
      <th>Gender</th>
      <th>Occupation</th>
    </tr>
    <tr>
      <td>1</td>
      <td>John</td>
      <td>25</td>
      <td>Male</td>
      <td>Software Engineer</td>
    </tr>
    <tr>
      <td>2</td>
      <td>Jane</td>
      <td>30</td>
      <td>Female</td>
      <td>Marketing Executive</td>
    </tr>
    <tr>
      <td>3</td>
      <td>Mike</td>
      <td>35</td>
      <td>Male</td>
      <td>Data Analyst</td>
    </tr>
    <tr>
      <td>4</td>
      <td>Emily</td>
      <td>28</td>
      <td>Female</td>
      <td>Product Manager</td>
    </tr>
    <tr>
      <td>5</td>
      <td>David</td>
      <td>40</td>
      <td>Male</td>
      <td>Business Development</td>
    </tr>
  </table>
</body>
</html>
```

```
        button {
            padding: 10px 15px;
            font-size: 16px;
            margin-bottom: 20px;
        }
    </style>
</head>
<body>
    <h1>Dynamic Table Example</h1>
    <button id="populateTable">Populate Table</button>

    <table id="dataTable">
        <thead>
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>Email</th>
                <th>Age</th>
            </tr>
        </thead>
        <tbody>
            <!-- Table rows will be added dynamically here -->
        </tbody>
    </table>

    <!-- Link to the external JavaScript file -->
    <script src="script.js"></script>
</body>
</html>
```

2. JavaScript File (script.js)

This file will contain the JavaScript code responsible for populating the table.

```
document.getElementById('populateTable').addEventListener('click', function() {  
    // Sample data to populate the table  
    const data = [  
        { id: 1, name: 'John Doe', email: 'john.doe@example.com', age: 28 },  
        { id: 2, name: 'Jane Smith', email: 'jane.smith@example.com', age: 34 },  
        { id: 3, name: 'Michael Johnson', email: 'michael.johnson@example.com', age: 45 },  
        { id: 4, name: 'Emily Davis', email: 'emily.davis@example.com', age: 29 }  
    ];  
  
    // Reference to the table body  
    const tbody = document.getElementById('dataTable').getElementsByTagName('tbody')[0];  
  
    // Clear existing table rows (if any)  
    tbody.innerHTML = "";  
  
    // Populate table with data  
    data.forEach(item => {  
        // Create a new row  
        const row = document.createElement('tr');  
  
        // Create and append cells to the row  
        Object.values(item).forEach(value => {  
            const cell = document.createElement('td');  
            cell.textContent = value;  
            row.appendChild(cell);  
        });  
  
        // Append the row to the table body
```

```
tbody.appendChild(row);  
  
});  
  
});
```

Output:

Dynamic Table Example

Populate Table

ID	Name	Email	Age
----	------	-------	-----

Dynamic Table Example

Populate Table

ID	Name	Email	Age
1	John Doe	john.doe@example.com	28
2	Jane Smith	jane.smith@example.com	34
3	Michael Johnson	michael.johnson@example.com	45
4	Emily Davis	emily.davis@example.com	29

Explanation:

1. HTML Structure:

- **<!DOCTYPE html>**: Declares the document type and HTML version.
- **<html lang="en">**: Root element with language set to English.
- **<head>**:
 - **<meta charset="UTF-8">**: Character encoding.
 - **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Responsive design.
 - **<title>**: Page title.
 - **<style>**: Internal CSS to style the table and button.

- **<body>**:
 - **<h1>**: Main heading.
 - **<button id="populateTable">Populate Table</button>**: Button to trigger data population.
 - **<table id="dataTable">**: Table structure with thead for headers and tbody where rows will be dynamically inserted.

2. JavaScript:

- **document.getElementById('populateTable').addEventListener('click', function() {...});**
 - Adds an event listener to the button. When clicked, it triggers the function to populate the table.
- **const data = [...]**: Defines a sample array of objects, each representing a row of data.
- **const tbody = document.getElementById('dataTable').getElementsByTagName('tbody')[0];**: Gets a reference to the <tbody> element.
- **tbody.innerHTML = ''**: Clears any existing rows in the table body.
- **data.forEach(item => {...})**: Iterates over the data array to create and append new rows.
 - **const row = document.createElement('tr');**: Creates a new table row.
 - **Object.values(item).forEach(value => {...})**: Iterates over the values of each item and creates cells for each value.
 - **tbody.appendChild(row)**: Appends the newly created row to the table body.

5. Build a responsive image gallery using HTML, CSS, and JavaScript.

1. HTML Structure

Create an index.html file for the gallery structure:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Responsive Image Gallery</title>

  <link rel="stylesheet" href="styles.css">

</head>

<body>

  <h1>Responsive Image Gallery</h1>

  <div class="gallery">

    <div class="gallery-item">

      <div class="caption">C</div>

    </div>

    <div class="gallery-item">

      <div class="caption">JAVA</div>

    </div>

    <div class="gallery-item">

      <div class="caption">Python And SQL</div>

    </div>

    <div class="gallery-item">
```

```


<div class="caption">Python</div>

</div>

<!-- Add more images as needed -->

</div>

<script src="scripts.js"></script>

</body>

</html>
```

2. CSS Styling

Create a styles.css file to handle the gallery's appearance and responsiveness:

```
body {
    font-family: Arial, sans-serif;
    padding: 20px;
    margin: 0;
    background-color: #f5f5f5;
}
```

```
h1 {
    text-align: center;
    margin-bottom: 20px;
}
```

```
.gallery {
    display: flex;
    flex-wrap: wrap;
    gap: 15px;
    justify-content: center;
}
```



```
.gallery-item {  
    position: relative;  
    width: 100%;  
    max-width: 300px;  
    overflow: hidden;  
    border-radius: 8px;  
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);  
}
```

```
.gallery-item img {  
    width: 100%;  
    height: auto;  
    display: block;  
}
```

```
.caption {  
    position: absolute;  
    bottom: 0;  
    left: 0;  
    width: 100%;  
    padding: 10px;  
    background: rgba(0, 0, 0, 0.5);  
    color: white;  
    text-align: center;  
    font-size: 16px;  
    font-weight: bold;  
    opacity: 0;  
    transition: opacity 0.3s ease;  
}
```

```

.gallery-item:hover .caption {
  opacity: 1;
}

@media (max-width: 768px) {
  .gallery {
    flex-direction: column;
    align-items: center;
  }

  .gallery-item {
    max-width: 100%;
  }
}

```

3. JavaScript for Interactive Features

Create a scripts.js file for additional interactivity (optional):

```

// This script can be used for additional interactive features or gallery effects
// For example, you might want to add a lightbox feature for images

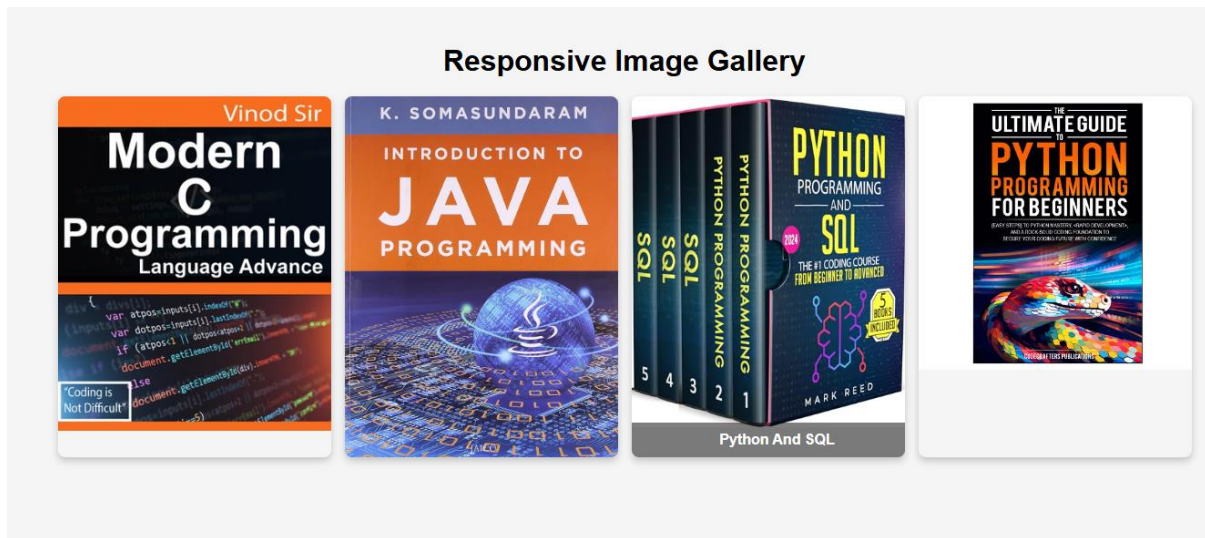
```

```

document.querySelectorAll('.gallery-item').forEach(item => {
  item.addEventListener('click', () => {
    // Example of an action, e.g., opening a modal or lightbox
    alert(`You clicked on ${item.querySelector('.caption').textContent}`);
  });
});

```

Output:



Explanation:

1. HTML (index.html):

- Contains the structure for the gallery with a div for each image and its caption.
- Links to the external CSS and JavaScript files for styling and functionality.

2. CSS (styles.css):

- **General Styling:** Sets up the font, padding, and background color.
- **Gallery Layout:** Uses Flexbox for a responsive layout that wraps items and centers them.
- **Gallery Item:** Styles each image and its caption. The position: relative on the gallery item and position: absolute on the caption allows the caption to overlay the image.
- **Hover Effect:** Caption opacity transitions on hover.
- **Responsive Design:** Uses media queries to adjust the layout on smaller screens.

3. JavaScript (scripts.js):

- Contains a simple script that demonstrates how to add interactivity, like displaying an alert when an image is clicked. This can be extended for more advanced features like modals or lightboxes.

6. Design a webpage layout using CSS float and positioning techniques.

1. HTML Structure (index.html)

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>CSS Float and Positioning Layout</title>

  <link rel="stylesheet" href="styles.css">

</head>

<body>

  <header>

    <h1>My Website</h1>

  </header>

  <div class="container">

    <aside class="sidebar">

      <h2>Sidebar</h2>

      <p>This is the sidebar content.</p>

    </aside>

    <main class="content">

      <h2>Main Content</h2>

      <p>This is the main content area. Here you can put your main content.</p>

    </main>

  </div>

  <footer>
```

```
<p>&copy; 2024 My Website</p>
```

```
</footer>
```

```
</body>
```

```
</html>
```

2. CSS Styling (styles.css)

```
/* General Styles */
```

```
body {
```

```
    font-family: Arial, sans-serif;
```

```
    margin: 0;
```

```
    padding: 0;
```

```
    box-sizing: border-box;
```

```
}
```

```
header, footer {
```

```
    background-color: #333;
```

```
    color: white;
```

```
    text-align: center;
```

```
    padding: 10px 0;
```

```
}
```

```
footer {
```

```
    font-size: 0.9em;
```

```
}
```

```
/* Container with Float Layout */
```

```
.container {  
    overflow: auto; /* Clears floats */  
    padding: 10px;  
}
```

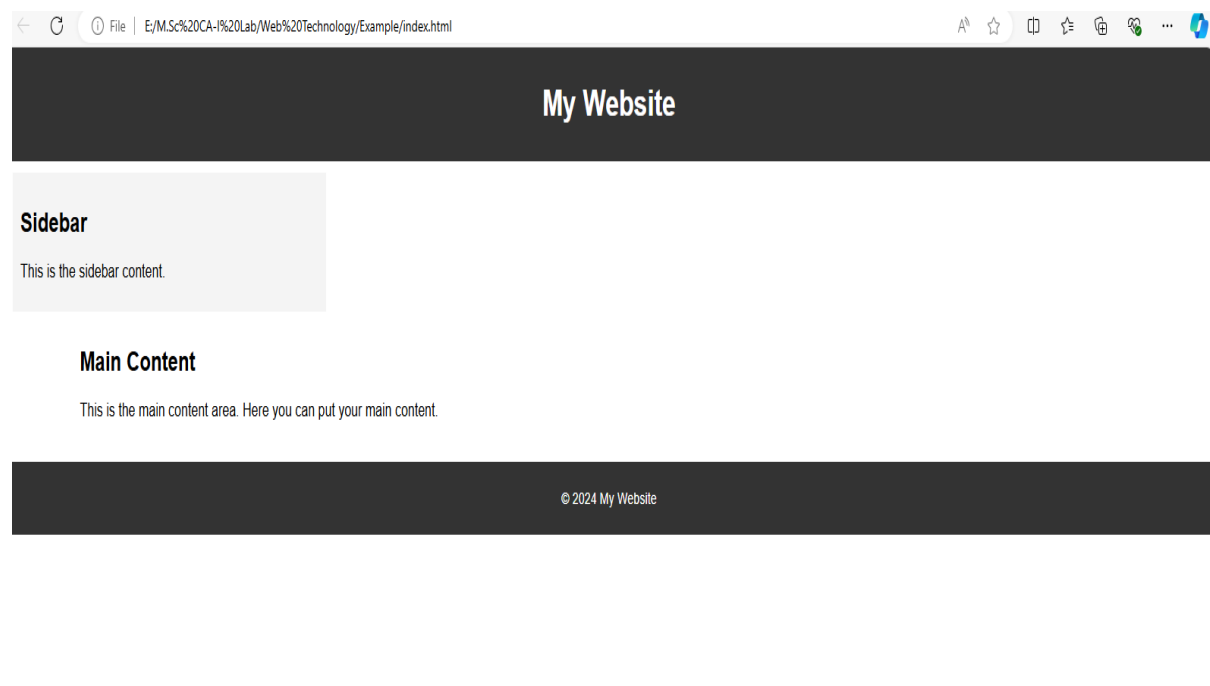
```
.sidebar {  
    float: left;  
    width: 25%;  
    background-color: #f4f4f4;  
    padding: 10px;  
}
```

```
.content {  
    float: left;  
    width: 70%;  
    background-color: #fff;  
    padding: 10px;  
    margin-left: 5%;  
}
```

```
@media (max-width: 768px) {  
    .sidebar, .content {  
        float: none;  
        width: 100%;  
        margin: 0;
```

```
}  
  
}  
  
/* Clearfix for Floating Elements */  
  
.clearfix::after {  
    content: "";  
    display: table;  
    clear: both;  
}
```

Output:



Explanation

1. **HTML (index.html):**
 - Defines a simple page structure with a header, a container that holds a sidebar and main content, and a footer.
2. **CSS (styles.css):**

- **General Styles:**
 - Sets up basic styles for the body, header, and footer, including padding and text alignment.
- **Container and Float Layout:**
 - .container uses overflow: auto to clear floats and prevent layout issues.
 - .sidebar and .content use float: left to position them side by side. width is specified to control the layout width.
 - .sidebar takes up 25% of the width, while .content takes up 70% with a margin-left to create space between the sidebar and content.
- **Responsive Design:**
 - Uses a media query to stack .sidebar and .content vertically on smaller screens (max-width: 768px).
 - When the screen is smaller than 768px, floats are removed, and the width is set to 100% to make the layout responsive.
- **Clearfix:**
 - The .clearfix class (though not used in this example) can be applied to parent elements of floated children to ensure that the parent properly wraps around the floated children.

CSS Positioning Techniques

1. **Static Positioning (Default):**
 - Elements are positioned according to the normal document flow.
 - position: static; (default value).
2. **Relative Positioning:**
 - The element is positioned relative to its normal position.
 - position: relative; allows you to use top, right, bottom, and left properties to adjust the element's position.
3. **Absolute Positioning:**
 - The element is positioned relative to the nearest positioned ancestor (position: relative, position: absolute).
 - position: absolute; removes the element from the normal document flow.
4. **Fixed Positioning:**
 - The element is positioned relative to the viewport and remains in place when scrolling.
 - position: fixed; keeps the element at a fixed position relative to the viewport.
5. **Sticky Positioning:**
 - The element is positioned based on the user's scroll position. It toggles between relative and fixed depending on the scroll position.
 - position: sticky; is used with top, bottom, left, or right to specify the sticky position.

7.Implement CSS transitions to create smooth hover effects on webpage elements.

1. HTML Structure (index.html)

Create an HTML file with some interactive elements (e.g., buttons and images) to apply hover effects.

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>CSS Transitions Example</title>

  <link rel="stylesheet" href="styles.css">

</head>

<body>

  <h1>CSS Transitions Demo</h1>

  <div class="container">

    <button class="transition-button">Hover over me!</button>

    <div class="transition-box"></div>

  </div>

</body>

</html>
```

2. CSS Styling (styles.css)

Apply styles and transitions to the interactive elements.

```
/* General Styles */
```

```
body {  
  
    font-family: Arial, sans-serif;  
  
    margin: 0;  
  
    padding: 20px;  
  
    background-color: #f0f0f0;  
  
    text-align: center;  
  
}
```

```
h1 {  
  
    margin-bottom: 20px;  
  
}
```

```
/* Container */
```

```
.container {  
  
    display: flex;  
  
    justify-content: center;  
  
    gap: 20px;  
  
}
```

```
/* Button with Transition */
```

```
.transition-button {  
  
    padding: 15px 30px;  
  
    font-size: 16px;  
  
    color: white;  
  
    background-color: #007bff;
```

```
border: none;

border-radius: 5px;

cursor: pointer;

transition: background-color 0.3s ease, transform 0.3s ease;
}


.transition-button:hover {

background-color: #0056b3;

transform: scale(1.1);
}


/* Box with Transition */

.transition-box {

width: 150px;

height: 150px;

background-color: #28a745;

border-radius: 8px;

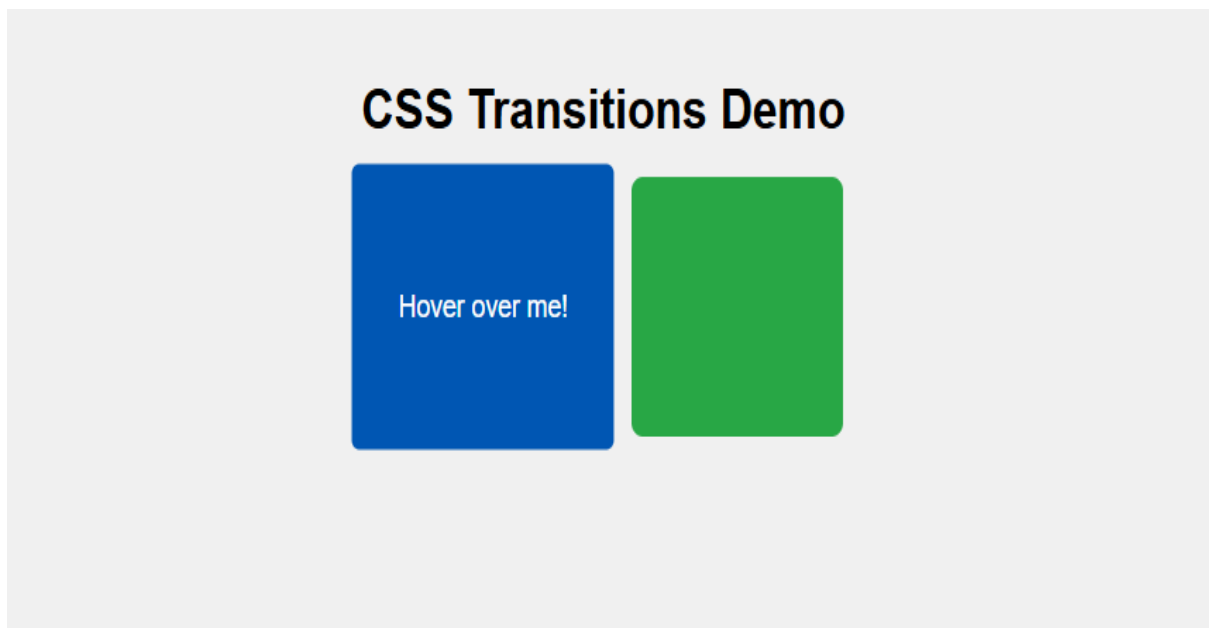
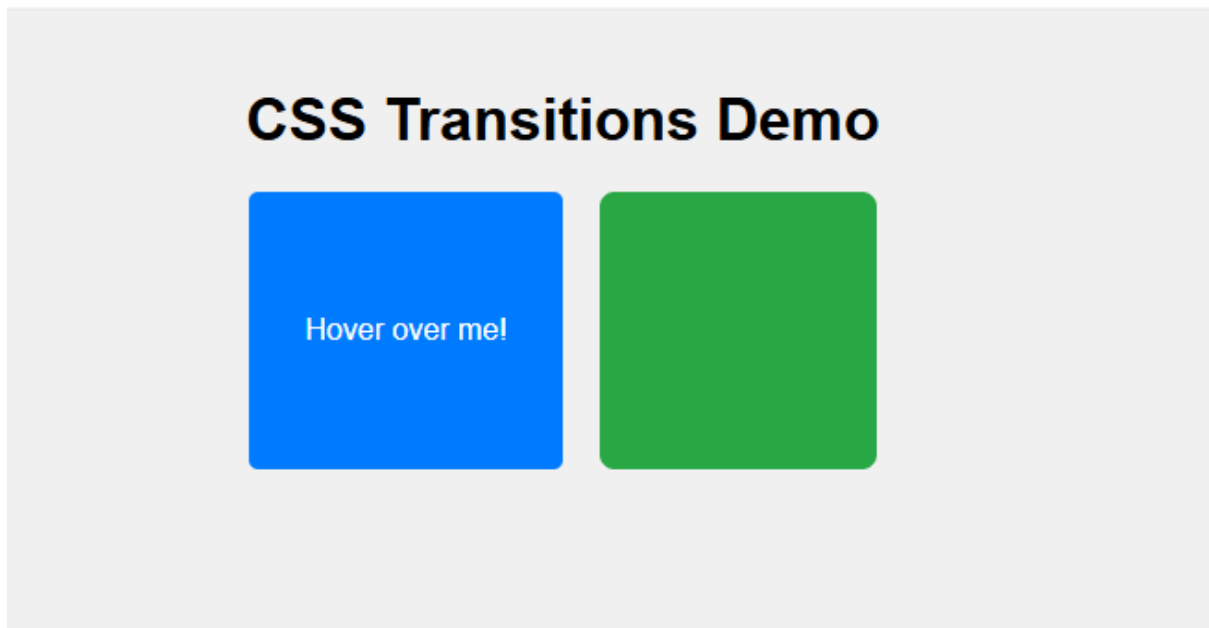
transition: background-color 0.3s ease, box-shadow 0.3s ease;
}


.transition-box:hover {

background-color: #218838;

box-shadow: 0 4px 15px rgba(0, 0, 0, 0.2);
}
```

Output:



Explanation

1. **HTML (index.html):**
 - Includes a heading and a container with a button and a div to demonstrate hover effects.
 - The class attributes are used to apply specific CSS rules to these elements.
2. **CSS (styles.css):**
 - **General Styles:**
 - Sets up basic styling for the body and heading.
 - **Container:**

- Uses Flexbox to center the button and box with some spacing.
- **Button with Transition:**
 - **.transition-button:**
 - Sets up the button's appearance (padding, font-size, color, background-color).
 - **transition: background-color 0.3s ease, transform 0.3s ease;;**
 - Specifies which properties to animate (background-color and transform).
 - 0.3s duration and ease timing function for smooth transitions.
 - **.transition-button:hover:**
 - Changes background color and scales the button when hovered.
- **Box with Transition:**
 - **.transition-box:**
 - Sets up the box's appearance (size, color, border-radius).
 - **transition: background-color 0.3s ease, box-shadow 0.3s ease;;**
 - Specifies transitions for background-color and box-shadow.
 - **.transition-box:hover:**
 - Changes the background color and adds a box-shadow on hover.

Key Points

- **transition Property:**
 - **transition: property duration timing-function delay;;**
 - property: The CSS property to transition (e.g., background-color, transform).
 - duration: How long the transition takes (e.g., 0.3s).
 - timing-function: The speed curve of the transition (e.g., ease, linear).
 - delay: Optional delay before the transition starts.
- **Hover Effects:**
 - Apply CSS transitions to create smooth, visually appealing hover effects that improve interactivity and user experience.

8.Implement CSS animations for visual effects.

1. HTML Structure (index.html)

Create an HTML file to include animated elements:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>CSS Animations Example</title>

  <link rel="stylesheet" href="styles.css">

</head>

<body>

  <h1>CSS Animations Demo</h1>

  <div class="animation-container">

    <div class="animated-box"></div>

    <div class="animated-circle"></div>

  </div>

</body>

</html>
```

2. CSS Styling (styles.css)

Apply animations and styles to the elements.

```
/* General Styles */
```

```
body {
```

```
    font-family: Arial, sans-serif;

    margin: 0;

    padding: 20px;

    background-color: #f0f0f0;

    text-align: center;

}
```

```
h1 {

    margin-bottom: 20px;

}
```

```
/* Animation Container */
```

```
.animation-container {

    display: flex;

    justify-content: center;

    align-items: center;

    gap: 20px;

    height: 400px;

}
```

```
/* Animated Box */
```

```
.animated-box {

    width: 100px;

    height: 100px;

    background-color: #007bff;
```

```
    animation: boxAnimation 3s infinite;
}
```

```
/* Animated Circle */
```

```
.animated-circle {
    width: 100px;
    height: 100px;
    background-color: #28a745;
    border-radius: 50%;
    animation: circleAnimation 3s infinite;
}
```

```
/* Keyframes for Box Animation */
```

```
@keyframes boxAnimation {
    0% {
        transform: translateX(0) rotate(0);
        background-color: #007bff;
    }
    50% {
        transform: translateX(200px) rotate(180deg);
        background-color: #0056b3;
    }
    100% {
        transform: translateX(0) rotate(360deg);
        background-color: #007bff;
    }
}
```



```

    }
}

/* Keyframes for Circle Animation */

@keyframes circleAnimation {

    0% {

        transform: scale(1) rotate(0);

        background-color: #28a745;

    }

    50% {

        transform: scale(1.5) rotate(180deg);

        background-color: #1e7e34;

    }

    100% {

        transform: scale(1) rotate(360deg);

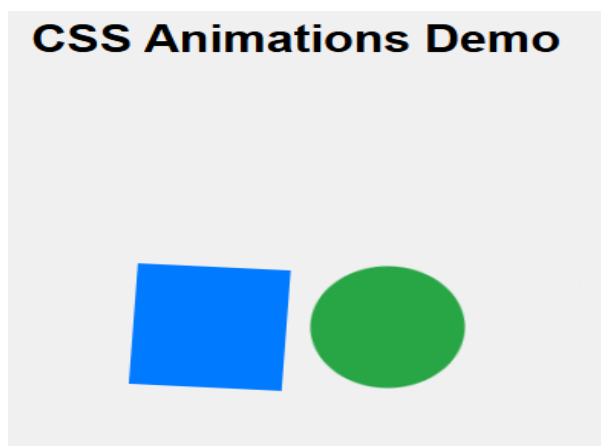
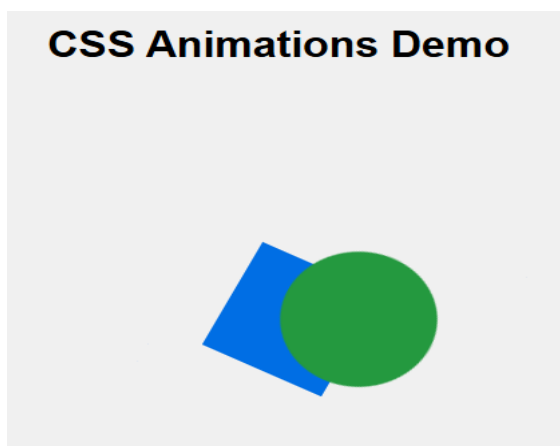
        background-color: #28a745;

    }

}

```

Output:



Explanation

1. **HTML (index.html):**
 - Contains two animated elements: a box and a circle, each with unique animation effects.
2. **CSS (styles.css):**
 - **General Styles:**
 - Sets up basic styling for the body and heading.
 - **Animation Container:**
 - Uses Flexbox to center the animated elements and provide space between them.
 - **Animated Box and Circle:**
 - **.animated-box:**
 - A simple square that animates using the boxAnimation keyframes.
 - **.animated-circle:**
 - A circular element that animates using the circleAnimation keyframes.
 - **Keyframes:**
 - **@keyframes boxAnimation:**
 - Defines the animation for the box, including transformation and color change at different stages (0%, 50%, 100%).
 - transform property is used to move (translateX) and rotate the box.
 - background-color changes through different stages.
 - **@keyframes circleAnimation:**
 - Defines the animation for the circle, including scaling and rotating effects.
 - transform property is used to scale and rotate the circle.
 - background-color changes through different stages.

Key Concepts

- **@keyframes:**
 - Defines the sequence of frames for an animation. Each keyframe represents a stage in the animation.
- **animation Property:**
 - **animation: name duration timing-function delay iteration-count direction;;**
 - name: The name of the @keyframes animation.
 - duration: The duration of one cycle of the animation (e.g., 3s).
 - timing-function: The speed curve of the animation (e.g., ease, linear).
 - delay: Optional delay before the animation starts.
 - iteration-count: How many times the animation should repeat (infinite for continuous).
 - direction: The direction of the animation (normal, reverse, alternate, alternate-reverse).
- **Transformations:**
 - **transform** property is used to apply various transformations like translate, scale, rotate, and skew.

- **Color Transitions:**
 - Changing properties like background-color during animations adds dynamic visual effects.

9.Design a responsive website using CSS media queries to adapt to different screen sizes.

1. HTML Structure (index.html)

Create a basic HTML structure for the website:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Responsive Design Example</title>

  <link rel="stylesheet" href="styles.css">

</head>

<body>

  <header>

    <h1>Responsive Website</h1>

    <nav>

      <ul>

        <li><a href="#">Home</a></li>

        <li><a href="#">About</a></li>

        <li><a href="#">Services</a></li>

        <li><a href="#">Contact</a></li>

      </ul>

    </nav>

  </header>

  <main>

    <section class="intro">

      <h2>Welcome to Our Website</h2>

      <p> Translation: "Pain itself is very important, for we are more aware of the pleasure it brings. We live for the sake of living, to the fullest extent.</p>
```

```
</section>

<section class="content">
  <article>
    <h3>Article 1</h3>
    <p>This is the first article. It has some interesting content.</p>
  </article>
  <article>
    <h3>Article 2</h3>
    <p>This is the second article. It also has some engaging information.</p>
  </article>
  <article>
    <h3>Article 3</h3>
    <p>This is the third article, which completes the trio of articles.</p>
  </article>
</section>
</main>

<footer>
  <p>&copy; 2024 Responsive Website. All rights reserved.</p>
</footer>
</body>
</html>
```

2. CSS Styling (styles.css)

Add styles and media queries to ensure responsiveness:

```
/* General Styles */
```

```
body {
  font-family: Arial, sans-serif;
  margin: 0;
```

```
padding: 0;
box-sizing: border-box;
}
```

```
header {
  background-color: #333;
  color: white;
  padding: 10px 0;
  text-align: center;
}
```

```
header h1 {
  margin: 0;
}
```

```
nav ul {
  list-style: none;
  padding: 0;
}
```

```
nav ul li {
  display: inline;
  margin-right: 15px;
}
```

```
nav ul li a {
  color: white;
  text-decoration: none;
}
```

```
/* Main Section */

main {
    padding: 20px;
}

.intro {
    text-align: center;
    margin-bottom: 20px;
}

.content {
    display: flex;
    flex-wrap: wrap;
    gap: 20px;
}

article {
    flex: 1 1 calc(33.333% - 40px); /* Adjusting width and spacing for articles */
    background-color: #f4f4f4;
    padding: 15px;
    border-radius: 5px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

/* Footer */

footer {
    background-color: #333;
    color: white;
    text-align: center;
    padding: 10px 0;
```

```
}
```

```
/* Media Queries */
```

```
@media (max-width: 768px) {
```

```
  nav ul li {
```

```
    display: block;
```

```
    margin-bottom: 10px;
```

```
  }
```

```
.content {
```

```
  flex-direction: column;
```

```
}
```

```
article {
```

```
  flex: 1 1 100%; /* Full width for articles on small screens */
```

```
}
```

```
}
```

```
@media (max-width: 480px) {
```

```
  header h1 {
```

```
    font-size: 1.5em; /* Smaller header text on very small screens */
```

```
}
```

```
nav ul li {
```

```
  text-align: center;
```

```
  margin-bottom: 5px;
```

```
}
```

```
.content {
```

```
  padding: 10px;
```

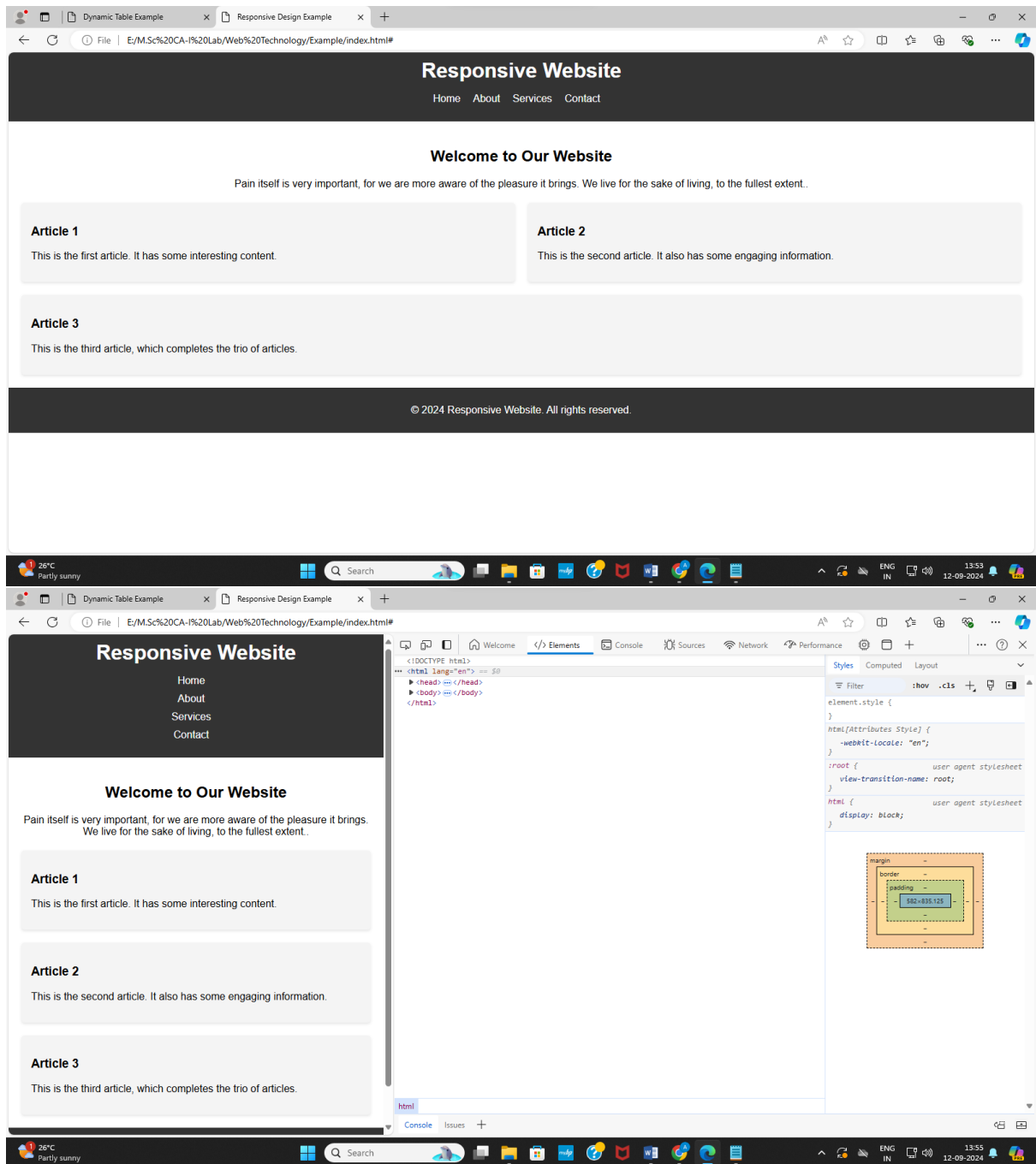


```
}

article {
  padding: 10px;
}

}
```

Output:



Explanation

1. HTML (index.html):

- Provides a simple structure with a header, main content area, and footer.
- Includes a navigation menu and multiple articles within the main section.

2. CSS (styles.css):

○ General Styles:

- Styles for body, header, navigation, and footer to ensure a clean layout.
- Flexbox is used in the .content section to layout articles in a row with gaps.

○ Media Queries:

▪ @media (max-width: 768px):

- Adjusts the navigation menu to stack items vertically on smaller screens.
- Changes the layout of the .content section to a column for better readability on medium-sized screens.
- Ensures that each article takes full width on smaller screens.

▪ @media (max-width: 480px):

- Further adjusts styles for very small screens (e.g., smartphones).
- Reduces font size for the header and adjusts padding and margin for a more compact layout.

Key Concepts

• Media Queries:

- Used to apply different styles depending on the viewport size. The syntax is @media (condition) { /* CSS rules */ }.
- Common conditions include max-width (for screens smaller than a certain width) and min-width (for screens larger than a certain width).

• Flexbox:

- Used to create flexible layouts. The flex property adjusts item sizes and spacing.
- flex-wrap: wrap; allows items to wrap onto multiple lines as needed.

• Responsive Design:

- Ensures that your layout adapts to different screen sizes and orientations, providing a better user experience across devices.

10.Design a webpage layout using CSS grid or flexbox for responsive design.

Designing a webpage layout using CSS Grid or Flexbox allows for modern, flexible, and responsive designs. Both techniques offer powerful tools for creating complex layouts, but they are used for different purposes:

- **CSS Grid** is excellent for creating two-dimensional layouts (rows and columns).
- **Flexbox** is ideal for one-dimensional layouts (either rows or columns).

Here, I'll provide examples of both techniques for a responsive webpage layout.

Example 1: Responsive Layout with CSS Grid

1. HTML Structure (index.html)

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>CSS Grid Layout</title>

  <link rel="stylesheet" href="styles.css">

</head>

<body>

  <header>

    <h1>CSS Grid Layout</h1>

    <nav>

      <ul>

        <li><a href="#">Home</a></li>

        <li><a href="#">About</a></li>

        <li><a href="#">Services</a></li>

        <li><a href="#">Contact</a></li>
```

```
        </ul>

    </nav>

</header>

<main>

    <section class="intro">

        <h2>Welcome</h2>

        <p>This is an introduction section.</p>

    </section>

    <section class="grid-container">

        <article class="grid-item">Item 1</article>

        <article class="grid-item">Item 2</article>

        <article class="grid-item">Item 3</article>

        <article class="grid-item">Item 4</article>

    </section>

</main>

<footer>

    <p>&copy; 2024 CSS Grid Layout. All rights reserved.</p>

</footer>

</body>

</html>
```

2. CSS Styling (styles.css)

```
/* General Styles */
```

```
body {  
  
    font-family: Arial, sans-serif;  
  
    margin: 0;  
  
    padding: 0;  
  
    box-sizing: border-box;  
  
}
```

```
header {  
  
    background-color: #333;  
  
    color: white;  
  
    padding: 15px;  
  
    text-align: center;  
  
}
```

```
header h1 {  
  
    margin: 0;  
  
}
```

```
nav ul {  
  
    list-style: none;  
  
    padding: 0;  
  
}
```

```
nav ul li {  
  
    display: inline;
```

```
    margin-right: 10px;
}
```

```
nav ul li a {
    color: white;
    text-decoration: none;
}
```

```
/* Main Section */
```

```
main {
    padding: 20px;
}
```

```
/* Grid Container */
```

```
.grid-container {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
    gap: 20px;
}
```

```
.grid-item {
    background-color: #f4f4f4;
    padding: 20px;
    border-radius: 5px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}
```

```
}
```

```
/* Footer */
```

```
footer {
```

```
background-color: #333;
```

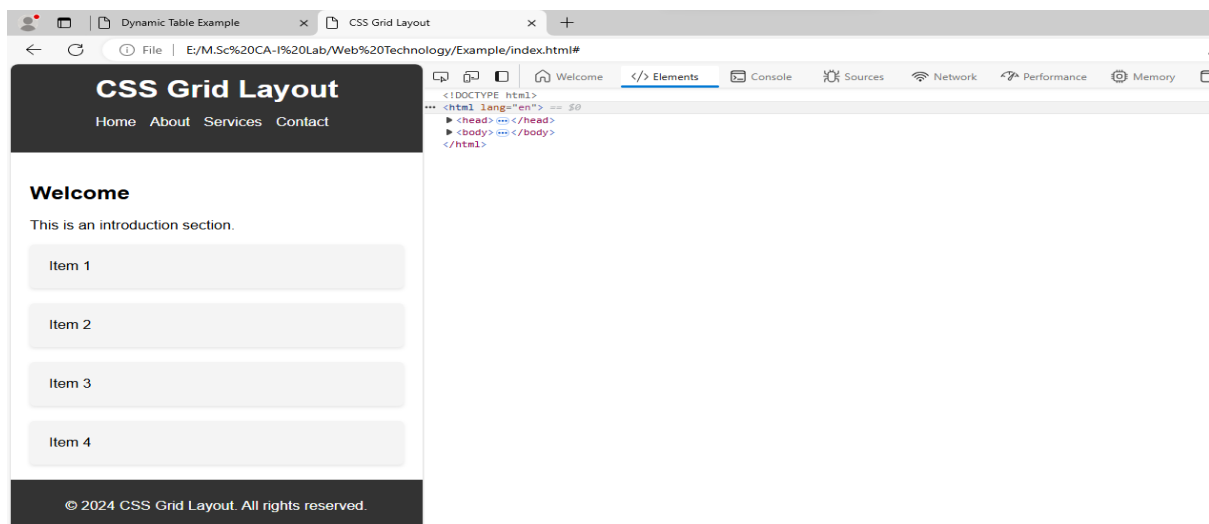
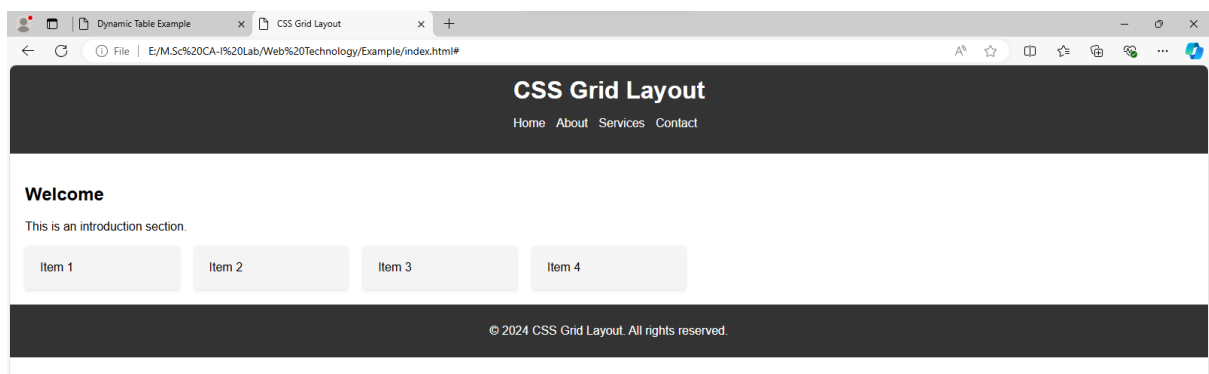
```
color: white;
```

```
text-align: center;
```

```
padding: 10px;
```

```
}
```

Output:



Explanation

- **CSS Grid Layout:**
 - .grid-container uses display: grid to set up a grid layout.

- grid-template-columns: repeat(auto-fill, minmax(200px, 1fr)) creates a responsive grid where each item has a minimum width of 200px and grows to fill available space. The auto-fill keyword automatically adjusts the number of columns based on the available space.
- gap: 20px adds space between grid items.

Example 2: Responsive Layout with Flexbox

1. HTML Structure (index.html)

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Flexbox Layout</title>

  <link rel="stylesheet" href="styles.css">

</head>

<body>

  <header>

    <h1>Flexbox Layout</h1>

    <nav>

      <ul>

        <li><a href="#">Home</a></li>

        <li><a href="#">About</a></li>

        <li><a href="#">Services</a></li>

        <li><a href="#">Contact</a></li>

      </ul>

    </nav>

  </header>
```



```
<main>

  <section class="intro">

    <h2>Welcome</h2>

    <p>This is an introduction section.</p>

  </section>


  <section class="flex-container">

    <div class="flex-item">Item 1</div>

    <div class="flex-item">Item 2</div>

    <div class="flex-item">Item 3</div>

  </section>

</main>


<footer>

  <p>&copy; 2024 Flexbox Layout. All rights reserved.</p>

</footer>

</body>

</html>
```

2. CSS Styling (styles.css)

```
/* General Styles */

body {

  font-family: Arial, sans-serif;

  margin: 0;

  padding: 0;
```

```
    box-sizing: border-box;
}
```

```
header {
    background-color: #333;
    color: white;
    padding: 15px;
    text-align: center;
}
```

```
header h1 {
    margin: 0;
}
```

```
nav ul {
    list-style: none;
    padding: 0;
}
```

```
nav ul li {
    display: inline;
    margin-right: 10px;
}
```

```
nav ul li a {
```

```
    color: white;

    text-decoration: none;
}
```

```
/* Main Section */
```

```
main {

    padding: 20px;
}
```

```
/* Flex Container */
```

```
.flex-container {

    display: flex;

    flex-wrap: wrap;

    gap: 20px;

    justify-content: center;
}
```

```
.flex-item {

    background-color: #f4f4f4;

    padding: 20px;

    border-radius: 5px;

    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

    flex: 1 1 calc(33.333% - 40px); /* Adjusts width and spacing */
}
```

```
/* Footer */
```

```
footer {
```

```
    background-color: #333;
```

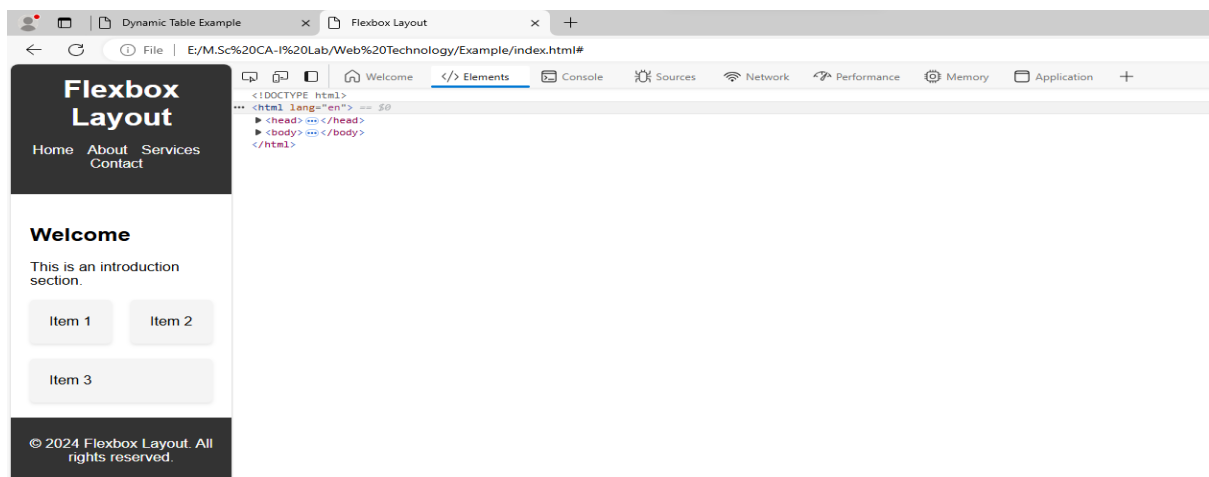
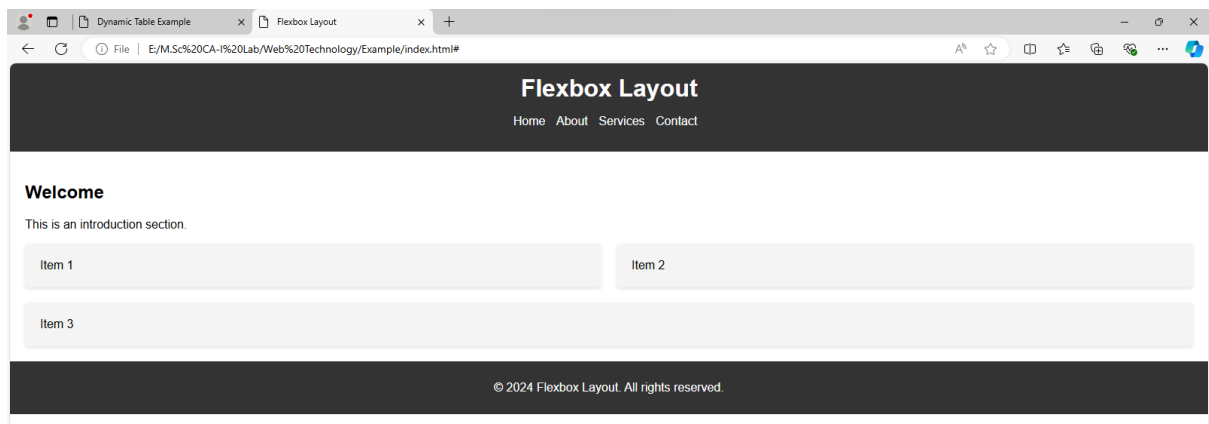
```
    color: white;
```

```
    text-align: center;
```

```
    padding: 10px;
```

```
}
```

Output:



Explanation

- **Flexbox Layout:**
 - .flex-container uses display: flex to create a flexible box layout.

- flex-wrap: wrap allows items to wrap onto multiple lines as needed.
- gap: 20px adds space between flex items.
- justify-content: center centers items horizontally.
- .flex-item has a flex property that allows items to grow and shrink based on available space. flex: 1 1 calc(33.333% - 40px) ensures three items per row with space for gaps.

Key Concepts

- **CSS Grid:**
 - Useful for creating complex grid layouts with both rows and columns.
 - Allows for precise control over grid areas and item placement.
- **Flexbox:**
 - Ideal for simpler one-dimensional layouts (either row or column).
 - Provides flexibility for aligning and distributing space among items in a container.

Both CSS Grid and Flexbox are essential for modern web design, and choosing between them often depends on the layout requirements and complexity of the design.

11. Build a website that incorporates CSS animations for visual effects.

1. HTML Structure (index.html)

Create an HTML file with elements that will be animated:

html

Copy code

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>CSS Animations Example</title>

  <link rel="stylesheet" href="styles.css">

</head>

<body>

  <header>

    <h1>CSS Animations</h1>

    <nav>

      <ul>

        <li><a href="#">Home</a></li>

        <li><a href="#">About</a></li>

        <li><a href="#">Services</a></li>

        <li><a href="#">Contact</a></li>

      </ul>

    </nav>

  </header>
```

```
<main>

  <section class="hero">

    <h2>Welcome to Our Site</h2>

    <p>Experience the magic of CSS animations!</p>

  </section>

  <section class="animated-section">

    <div class="box"></div>

    <div class="circle"></div>

  </section>

</main>

<footer>

  <p>&copy; 2024 CSS Animations. All rights reserved.</p>

</footer>

</body>

</html>
```

2. CSS Styling (styles.css)

Apply styles and animations to make the website engaging:

css

Copy code

```
/* General Styles */
```

```
body {
```

```
  font-family: Arial, sans-serif;
```

```
margin: 0;

padding: 0;

box-sizing: border-box;

background-color: #f9f9f9;

}
```

```
header {

background-color: #333;

color: white;

padding: 15px;

text-align: center;

}
```

```
header h1 {

margin: 0;

}
```

```
nav ul {

list-style: none;

padding: 0;

}
```

```
nav ul li {

display: inline;

margin-right: 15px;
```



```
}
```

```
nav ul li a {
```

```
    color: white;
```

```
    text-decoration: none;
```

```
}
```

```
/* Main Section */
```

```
main {
```

```
    padding: 20px;
```

```
    text-align: center;
```

```
}
```

```
/* Hero Section */
```

```
.hero {
```

```
    padding: 50px;
```

```
    background-color: #007bff;
```

```
    color: white;
```

```
    border-radius: 8px;
```

```
    animation: fadeIn 2s ease-in-out;
```

```
}
```

```
.hero h2 {
```

```
    margin: 0;
```

```
    font-size: 2em;
```

```
}
```

```
.hero p {  
    font-size: 1.2em;  
}
```

```
/* Animated Section */
```

```
.animated-section {  
    display: flex;  
    justify-content: center;  
    align-items: center;  
    gap: 20px;  
    padding: 20px;  
}
```

```
.box, .circle {  
    width: 100px;  
    height: 100px;  
    background-color: #28a745;  
    border-radius: 10px;  
    animation: bounce 2s infinite;  
}
```

```
.circle {  
    background-color: #ffc107;
```

```
border-radius: 50%;  
  
animation: rotate 3s linear infinite;  
  
}
```

```
/* Keyframes for Animations */
```

```
@keyframes fadeIn {  
  
  from {  
  
    opacity: 0;  
  
    transform: translateY(-20px);  
  
  }  
  
  to {  
  
    opacity: 1;  
  
    transform: translateY(0);  
  
  }  
  
}
```

```
@keyframes bounce {  
  
  0%, 20%, 50%, 80%, 100% {  
  
    transform: translateY(0);  
  
  }  
  
  40% {  
  
    transform: translateY(-30px);  
  
  }  
  
  60% {  
  
    transform: translateY(-15px);  
  
  }  
  
}
```

```

    }

}

@keyframes rotate {

    from {

        transform: rotate(0deg);

    }

    to {

        transform: rotate(360deg);

    }

}

/* Footer */

footer {

    background-color: #333;

    color: white;

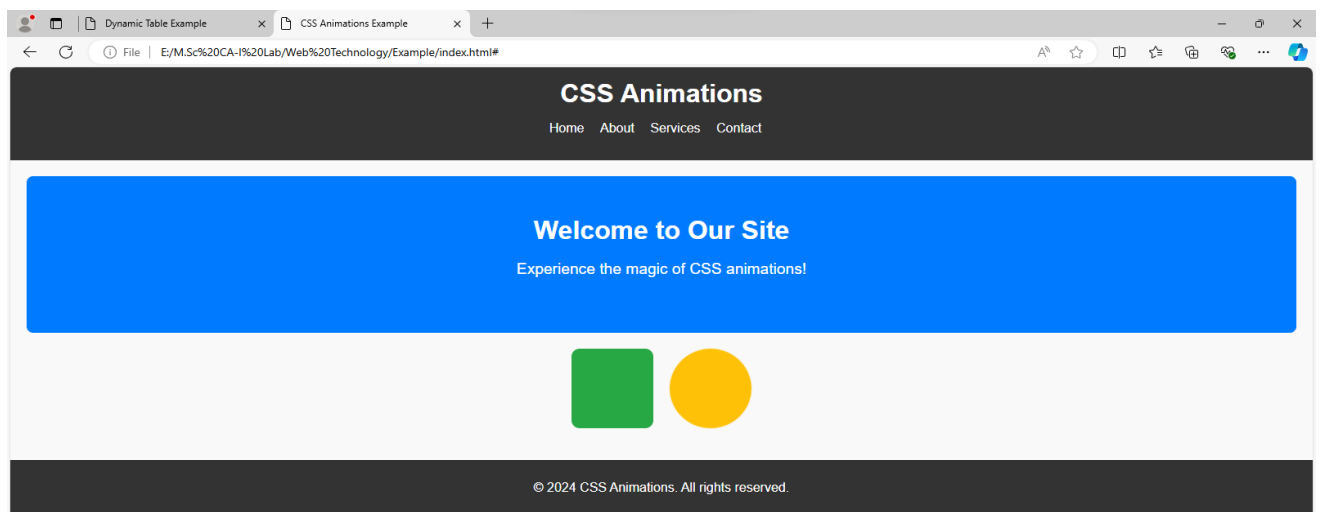
    text-align: center;

    padding: 10px;

}

```

Output:



Explanation

1. **HTML (index.html):**
 - **<header>**: Contains the site title and navigation.
 - **<main>**: Includes a hero section with a welcome message and an animated section with a box and a circle.
 - **<footer>**: Provides footer information.
2. **CSS (styles.css):**
 - **General Styles**: Basic styling for body, header, navigation, and footer.
 - **Hero Section**:
 - **.hero**: Uses the fadeIn animation to smoothly appear on the page.
 - **Animated Section**:
 - **.box and .circle**: Animated elements using bounce and rotate animations respectively.
 - **.box**: Applies a bounce animation to create a bouncing effect.
 - **.circle**: Applies a rotating animation for continuous spinning.
 - **Keyframes**:
 - **fadeIn**: Fades in the element from 0% to 100% opacity and moves it from above the starting position.
 - **bounce**: Creates a bouncing effect with multiple stages of movement.
 - **rotate**: Rotates the element continuously.

Key Concepts

- **CSS Animations:**
 - **@keyframes**: Define the animation's stages. Keyframes specify styles at various points during the animation.
 - **animation Property**: Controls the animation's name, duration, timing function, delay, iteration count, and direction.
- **@keyframes Syntax:**
 - **from and to**: Define the start and end points of the animation.
 - **Percentage (%)**: Define intermediate points in the animation.
- **Animation Timing Functions:**
 - **ease-in-out**: Starts and ends slowly.
 - **linear**: Constant speed throughout the animation.

12. Build a JavaScript calculator that performs basic arithmetic operations.

HTML (index.html)

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Basic Calculator</title>

  <style>

    .calculator {

      max-width: 250px;

      margin: 100px auto;

      padding: 20px;

      border: 1px solid #ccc;

      border-radius: 5px;

      box-shadow: 0 0 10px rgba(0,0,0,0.1);

    }

    .calculator input {

      width: 100%;

      margin-bottom: 10px;

      padding: 15px;

      box-sizing: border-box;

      font-size: 24px;

      text-align: right;

      border: 1px solid #ddd;
```

```
        border-radius: 5px;
    }

    .calculator button {
        width: 22%;
        padding: 15px;
        margin: 2px;
        box-sizing: border-box;
        font-size: 18px;
        border: 1px solid #ddd;
        border-radius: 5px;
        cursor: pointer;
        background-color: #f4f4f4;
    }

    .calculator button:hover {
        background-color: #e0e0e0;
    }

    .calculator button.operator {
        background-color: #f39c12;
        color: white;
    }

    .calculator button.operator:hover {
        background-color: #e67e22;
    }

    .calculator button.equal {
        background-color: #27ae60;
```

```
        color: white;

    }

    .calculator button.equal:hover {

        background-color: #2ecc71;

    }

    .calculator button.clear {

        background-color: #c0392b;

        color: white;

    }

    .calculator button.clear:hover {

        background-color: #e74c3c;

    }

</style>

</head>

<body>

    <div class="calculator">

        <input type="text" id="display" class="result" readonly>

        <button class="clear" onclick="clearDisplay()">C</button>

        <button class="operator" onclick="appendCharacter('/')">/</button>

        <button class="operator" onclick="appendCharacter('*')">*</button>

        <button class="operator" onclick="appendCharacter('-')">-</button>

        <button onclick="appendCharacter('7')">7</button>

        <button onclick="appendCharacter('8')">8</button>

        <button onclick="appendCharacter('9')">9</button>

        <button class="operator" onclick="appendCharacter('+')">+</button>
```



```

<button onclick="appendCharacter('4')">4</button>
<button onclick="appendCharacter('5')">5</button>
<button onclick="appendCharacter('6')">6</button>
<button class="equal" onclick="calculate()">=</button>
<button onclick="appendCharacter('1')">1</button>
<button onclick="appendCharacter('2')">2</button>
<button onclick="appendCharacter('3')">3</button>
<button onclick="appendCharacter('0')">0</button>
<button onclick="appendCharacter('.')">.</button>
</div>

<script src="calculator.js"></script>
</body>
</html>

```

JavaScript (calculator.js)

Make sure your calculator.js remains the same as provided earlier:

```

function appendCharacter(character) {
    const display = document.getElementById('display');

    // Prevent multiple operators in a row or invalid expression at the start
    if (display.value === " && isNaN(character)) return;

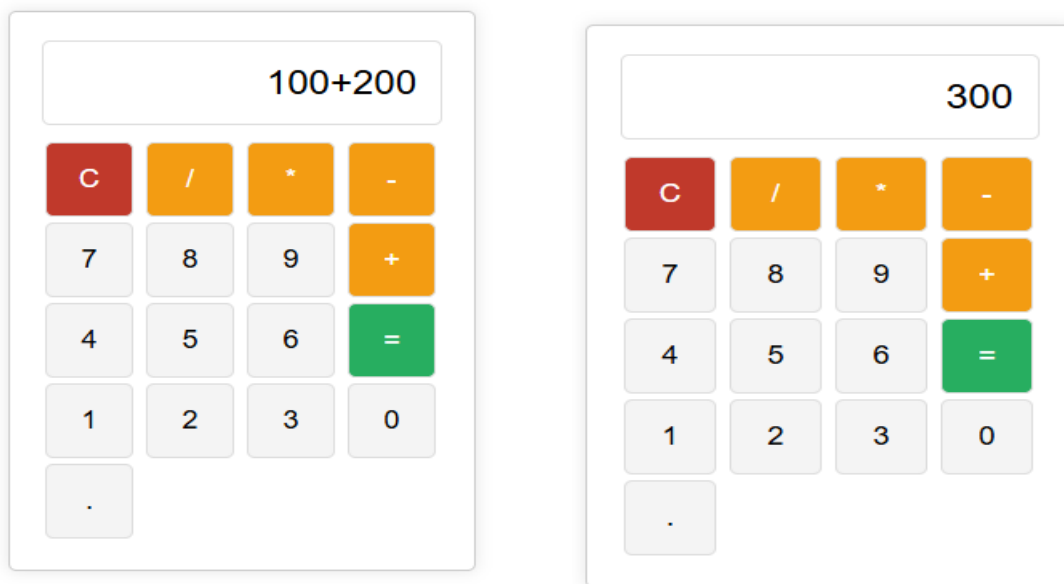
    if (display.value !== " && isNaN(character) && isNaN(display.value.slice(-1))) return;

    display.value += character;
}

```

```
function clearDisplay() {  
    document.getElementById('display').value = "";  
}  
  
function calculate() {  
    const display = document.getElementById('display');  
    try {  
        // Use eval carefully, and only for arithmetic operations  
        // Ensure the expression is safe and valid  
        let result = eval(display.value);  
        // Check if result is a number  
        if (isNaN(result) || !isFinite(result)) {  
            throw new Error('Invalid result');  
        }  
        display.value = result;  
    } catch (error) {  
        // In case of error, show 'Error'  
        display.value = 'Error';  
    }  
}
```

Output:



Explanations:

The HTML code sets up the structure of the calculator, defining its layout and how it should appear on the page. Here's a detailed explanation:

1. HTML Structure:

- **<!DOCTYPE html>**: Defines the document type and version of HTML being used (HTML5 here).
- **<html>**: The root element of the HTML document.
- **<head>**: Contains meta-information about the document, such as character set and viewport settings, along with the title and internal CSS styles.
- **<body>**: Contains the content of the document, which in this case is the calculator.

2. Calculator Layout:

- **<div class="calculator">**: A container for the calculator, styled and centered on the page.
- **<input type="text" id="display" class="result" readonly>**: An input field where the user sees the current calculation and result. It is set to readonly to prevent user edits.
- **<button>** elements: Each button represents a key on the calculator. The onclick attribute specifies a JavaScript function to call when the button is clicked.

3. Button Classes:

- **class="clear"**: Applied to the "C" button to clear the display.
- **class="operator"**: Applied to arithmetic operators (/, *, -, +) to differentiate their style.
- **class="equal"**: Applied to the "=" button to calculate the result.

CSS (Internal Style Block)

The CSS provides visual styling for the calculator. Here's a breakdown:

1. **.calculator:**

- **max-width: 250px;** Limits the width of the calculator to 250 pixels.
- **margin: 100px auto;** Centers the calculator horizontally and adds a margin from the top.
- **padding: 20px;** Adds padding inside the calculator container.
- **border: 1px solid #ccc;** Adds a light gray border around the calculator.
- **border-radius: 5px;** Rounds the corners of the border.
- **box-shadow: 0 0 10px rgba(0,0,0,0.1);** Adds a subtle shadow for a 3D effect.

2. **input:**

- **width: 100%;** Makes the input field take up the full width of its container.
- **margin-bottom: 10px;** Adds space below the input field.
- **padding: 15px;** Adds padding inside the input field for better readability.
- **font-size: 24px;** Increases the font size to make it easy to read.
- **text-align: right;** Aligns the text to the right, as you would expect for a calculator display.
- **border: 1px solid #ddd;** Adds a border around the input field.
- **border-radius: 5px;** Rounds the corners of the border.

3. **button:**

- **width: 22%;** Makes each button take up about 22% of the width of its container, allowing for a grid layout.
- **padding: 15px;** Adds padding inside the buttons.
- **margin: 2px;** Adds a small margin around each button.
- **font-size: 18px;** Sets the font size for button text.
- **border: 1px solid #ddd;** Adds a border around each button.
- **border-radius: 5px;** Rounds the corners of the buttons.

- **cursor: pointer;** Changes the cursor to a pointer when hovering over the buttons.

4. Button States:

- **.calculator button:hover:** Changes the background color of buttons when hovered over for better visual feedback.
- **.calculator button.operator:** Styles the arithmetic operator buttons with a distinct background color.
- **.calculator button.equal:** Styles the "=" button with a different background color.
- **.calculator button.clear:** Styles the "C" button with a red background color to signify its special function.

JavaScript (calculator.js)

The JavaScript code adds functionality to the calculator. Here's how it works:

1. **appendCharacter(character):**

- **const display = document.getElementById('display');** Gets the display input element.
- **if (display.value === '' && isNaN(character)) return;** Prevents adding an operator if the display is empty.
- **if (display.value !== '' && isNaN(character) && isNaN(display.value.slice(-1))) return;** Prevents adding multiple operators consecutively.
- **display.value += character;** Appends the clicked character to the display value.

2. **clearDisplay():**

- **document.getElementById('display').value = '';** Clears the display input field.

3. **calculate():**

- **let result = eval(display.value);** Evaluates the mathematical expression in the display using eval.
- **if (isNaN(result) || !isFinite(result)):** Checks if the result is a valid number and finite. If not, it throws an error.
- **display.value = result;** Displays the result of the calculation.
- **catch (error):** Catches any errors during evaluation (e.g., invalid expressions) and sets the display to 'Error'.

12.Create a JavaScript program that generates a random password based on user-specified criteria.

Creating a JavaScript program to generate a random password based on user-specified criteria involves a few steps:

1. **User Input:** Collect user preferences such as length of the password and whether to include uppercase letters, lowercase letters, numbers, and special characters.
2. **Password Generation:** Generate a password based on the specified criteria.
3. **Output the Password:** Display or return the generated password.

Here's a simple implementation of such a program:

HTML

First, create the HTML structure to gather user input and display the generated password.

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Password Generator</title>

  <style>

    body {

      font-family: Arial, sans-serif;

      display: flex;

      justify-content: center;

      align-items: center;

      height: 100vh;

      margin: 0;

      background-color: #f0f0f0;

    }

    .container {

      background: #fff;

      padding: 20px;

      border-radius: 10px;
```

```
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
    max-width: 400px;
    width: 100%;
}
.form-group {
    margin-bottom: 15px;
}
.form-group label {
    display: block;
    margin-bottom: 5px;
}
.form-group input[type="number"] {
    width: 100%;
    padding: 10px;
    box-sizing: border-box;
}
.form-group input[type="checkbox"] {
    margin-right: 10px;
}
.form-group button {
    padding: 10px 20px;
    font-size: 16px;
    background-color: #007bff;
    color: #fff;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}
.form-group button:hover {
    background-color: #0056b3;
```

```

    }
    #password {
        margin-top: 15px;
        font-size: 18px;
        word-wrap: break-word;
        background: #f7f7f7;
        padding: 10px;
        border: 1px solid #ddd;
        border-radius: 5px;
    }
</style>
</head>
<body>
    <div class="container">
        <h1>Password Generator</h1>
        <div class="form-group">
            <label for="length">Password Length:</label>
            <input type="number" id="length" min="1" value="12">
        </div>
        <div class="form-group">
            <label><input type="checkbox" id="includeUppercase"> Include Uppercase
Letters</label>
            <label><input type="checkbox" id="includeNumbers"> Include Numbers</label>
            <label><input type="checkbox" id="includeSpecial"> Include Special
Characters</label>
        </div>
        <div class="form-group">
            <button onclick="generatePassword()">Generate Password</button>
        </div>
        <div id="password">Your generated password will appear here.</div>
    </div>

```



```
<script src="password-generator.js"></script>
</body>
</html>
```

JavaScript

Now, create the JavaScript file password-generator.js to handle the password generation logic.

```
function generatePassword() {
    const length = parseInt(document.getElementById('length').value);
    const includeUppercase = document.getElementById('includeUppercase').checked;
    const includeNumbers = document.getElementById('includeNumbers').checked;
    const includeSpecial = document.getElementById('includeSpecial').checked;

    if (isNaN(length) || length <= 0) {
        alert('Please enter a valid password length.');
```



```
        return;
    }

    // Character sets
    const lowercaseChars = 'abcdefghijklmnopqrstuvwxyz';
    const uppercaseChars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
    const numberChars = '0123456789';
    const specialChars = '!@#$%^&*()_+-=[]{}|;:,.<>?';

    let charSet = lowercaseChars;

    if (includeUppercase) {
        charSet += uppercaseChars;
    }

    if (includeNumbers) {
```

```

        charSet += numberChars;
    }
    if (includeSpecial) {
        charSet += specialChars;
    }

    if (charSet.length === 0) {
        alert('Please select at least one character type.');
```

return;

```

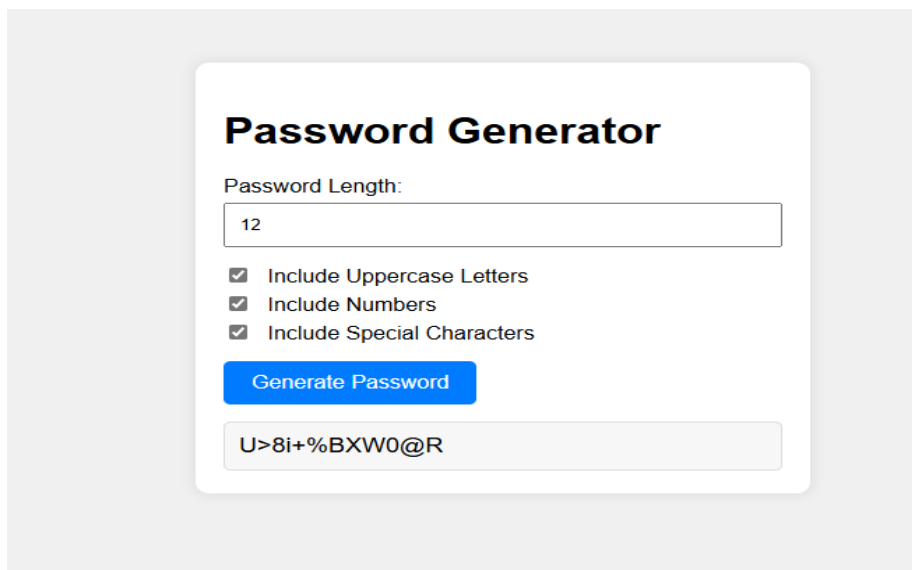
    }

    let password = "";
    for (let i = 0; i < length; i++) {
        const randomIndex = Math.floor(Math.random() * charSet.length);
        password += charSet[randomIndex];
    }

    document.getElementById('password').textContent = password;
}

```

Output:



The screenshot shows a web application titled "Password Generator". It features a form with the following elements:

- A label "Password Length:" followed by a text input field containing the value "12".
- Three checked checkboxes:
 - ☒ Include Uppercase Letters
 - ☒ Include Numbers
 - ☒ Include Special Characters
- A blue button labeled "Generate Password".
- A light gray output box displaying the generated password: "U>8i+%BXW0@R".

Explanation

HTML

Purpose: Defines the structure and content of the password generator interface.

1. Document Structure:

- **<!DOCTYPE html>**: Declares the document type as HTML5.
- **<html lang="en">**: The root element of the HTML document, with the language set to English.
- **<head>**: Contains metadata, including character set, viewport settings, the document title, and internal CSS.
- **<body>**: Contains the main content of the document, which in this case is the password generator interface.

2. Password Generator Layout:

- **<div class="container">**: A wrapper for the calculator, styled with a maximum width, centered alignment, padding, and shadow.
- **<h1>Password Generator</h1>**: The title of the password generator.
- **Input Fields:**
 - **<label for="length">Password Length:</label>**: Label for the password length input field.
 - **<input type="number" id="length" min="1" value="12">**: Allows the user to specify the desired password length. Defaults to 12.
- **Checkboxes:**
 - **<input type="checkbox" id="includeUppercase"> Include Uppercase Letters**: Option to include uppercase letters in the password.
 - **<input type="checkbox" id="includeNumbers"> Include Numbers**: Option to include numbers.
 - **<input type="checkbox" id="includeSpecial"> Include Special Characters**: Option to include special characters.
- **Button:**
 - **<button onclick="generatePassword()">Generate Password</button>**: Button that triggers the password generation when clicked.
- **Display Area:**
 - **<div id="password">Your generated password will appear here.</div>**: Displays the generated password.

CSS

Purpose: Provides styling for the password generator to make it visually appealing and user-friendly.

1. General Styles:

- **body:** Centers the content vertically and horizontally, applies a background color, and uses a clean, readable font.
- **.container:** Styles the main container with padding, a border, rounded corners, and a shadow for a subtle 3D effect.
- **.form-group:** Styles for form elements, including spacing and layout adjustments.

2. Form Elements:

- **input[type="number"]:** Styles the input field for password length to be full-width and user-friendly.
- **input[type="checkbox"]:** Styles for checkboxes and labels, ensuring they align well and are easy to use.
- **button:** Styles for the button with padding, a background color, border radius, and hover effects for better user interaction.
- **#password:** Styles for the password display area to make it stand out and handle long passwords gracefully.

JavaScript

Purpose: Implements the logic for generating a random password based on user input.

1. Function: generatePassword()

- **Input Reading:**
 - **const length = parseInt(document.getElementById('length').value);** Retrieves and parses the password length.
 - **const includeUppercase = document.getElementById('includeUppercase').checked;** Checks if the user wants uppercase letters.
 - **const includeNumbers = document.getElementById('includeNumbers').checked;** Checks if the user wants numbers.
 - **const includeSpecial = document.getElementById('includeSpecial').checked;** Checks if the user wants special characters.
- **Validation:**

- **Checks if the length is valid:** If not, it displays an alert.
- **Ensures at least one character type is selected:** If no types are selected, it alerts the user.
- **Character Sets:**
 - **const lowercaseChars:** Defines characters for lowercase letters.
 - **const uppercaseChars:** Defines characters for uppercase letters.
 - **const numberChars:** Defines characters for numbers.
 - **const specialChars:** Defines characters for special symbols.
- **Build Character Set:**
 - **Combines character sets** based on user selections.
- **Password Generation:**
 - **Loop:** Generates a password by selecting random characters from the combined set.
 - **document.getElementById('password').textContent = password;** Displays the generated password.

14. Develop a JavaScript program that manipulates the Document Object Model (DOM) to dynamically update webpage content.

Manipulating the Document Object Model (DOM) with JavaScript allows you to dynamically update webpage content based on user interaction or other conditions. Here's a step-by-step guide and example demonstrating how to manipulate the DOM to update webpage content dynamically.

Example Overview

Let's create a simple web page with interactive features that:

1. **Update Text Content:** Change text content when a button is clicked.
2. **Add New Elements:** Add new elements to the page dynamically.
3. **Remove Elements:** Remove elements from the page.
4. **Change Element Styles:** Modify styles of existing elements.

HTML Structure

First, create an HTML file that will serve as the base for your interactions.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>DOM Manipulation Example</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
      background-color: #f4f4f4;
    }
    .container {
```

```
background: #fff;
padding: 20px;
border-radius: 8px;
box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
max-width: 500px;
width: 100%;
text-align: center;
}
button {
padding: 10px 20px;
margin: 10px;
border: none;
border-radius: 5px;
cursor: pointer;
font-size: 16px;
}
#addButton {
background-color: #28a745;
color: white;
}
#removeButton {
background-color: #dc3545;
color: white;
}
#changeStyleButton {
background-color: #007bff;
color: white;
}
#dynamicContent {
margin-top: 20px;
```

```

    }
    .newElement {
        margin: 10px 0;
        padding: 10px;
        background-color: #e9ecef;
        border-radius: 5px;
    }
</style>
</head>
<body>
    <div class="container">
        <h1>DOM Manipulation Example</h1>
        <button id="updateTextButton">Update Text Content</button>
        <button id="addButton">Add New Element</button>
        <button id="removeButton">Remove Last Element</button>
        <button id="changeStyleButton">Change Style</button>
        <div id="dynamicContent">This is some static content.</div>
    </div>

    <script src="dom-manipulation.js"></script>
</body>
</html>

```

JavaScript (dom-manipulation.js)

Now, create a JavaScript file to handle the DOM manipulation.

```

document.addEventListener('DOMContentLoaded', () => {
    const updateTextButton = document.getElementById('updateTextButton');
    const addButton = document.getElementById('addButton');
    const removeButton = document.getElementById('removeButton');
    const changeStyleButton = document.getElementById('changeStyleButton');
    const dynamicContent = document.getElementById('dynamicContent');

```



```
// Update text content
```

```
updateTextButton.addEventListener('click', () => {  
    dynamicContent.textContent = 'The text content has been updated!';  
});
```

```
// Add new element
```

```
addButton.addEventListener('click', () => {  
    const newElement = document.createElement('div');  
    newElement.className = 'newElement';  
    newElement.textContent = 'This is a new dynamically added element.';  
    dynamicContent.appendChild(newElement);  
});
```

```
// Remove last element
```

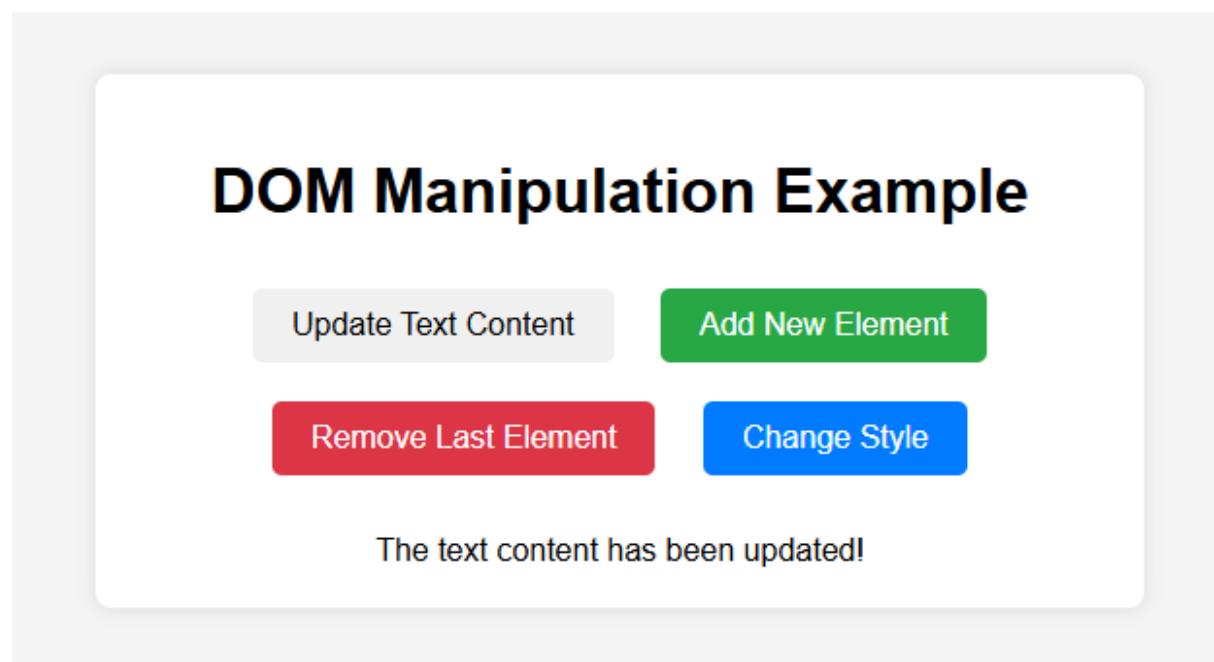
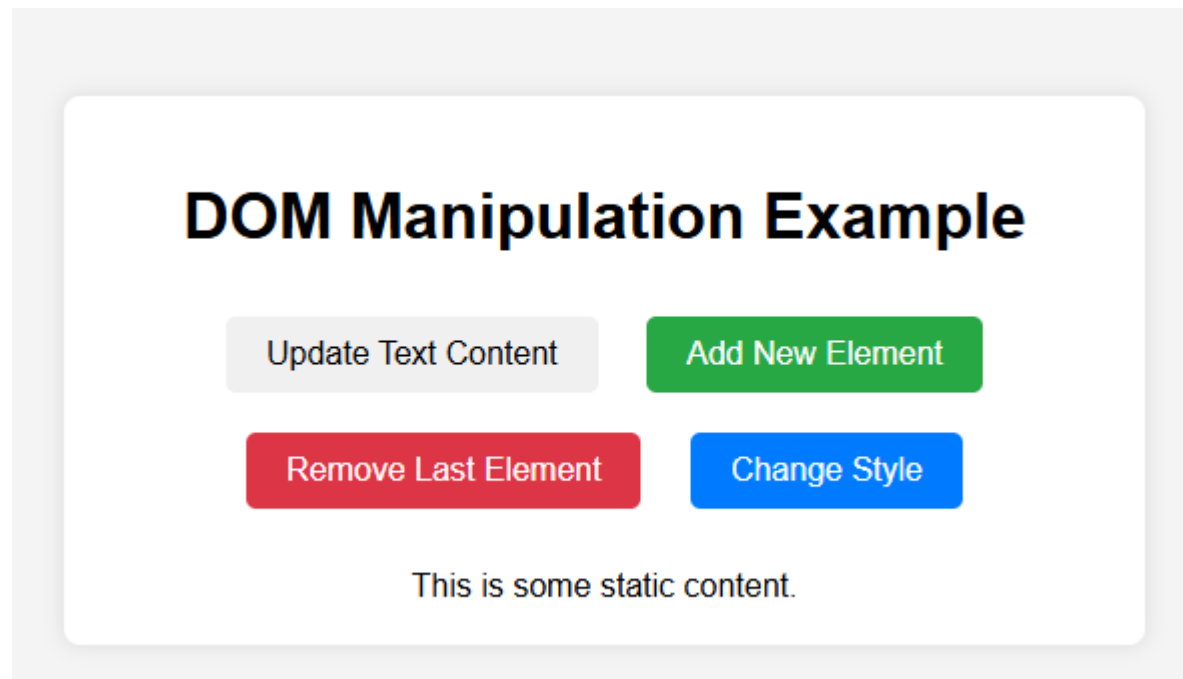
```
removeButton.addEventListener('click', () => {  
    const lastElement = dynamicContent.querySelector('.newElement:last-of-type');  
    if (lastElement) {  
        dynamicContent.removeChild(lastElement);  
    } else {  
        alert('No elements to remove.');    }  
});
```

```
// Change style
```

```
changeStyleButton.addEventListener('click', () => {  
    dynamicContent.style.backgroundColor = '#e0f7fa';  
    dynamicContent.style.color = '#00796b';  
    dynamicContent.style.fontSize = '20px';  
    dynamicContent.style.padding = '20px';  
});
```

```
dynamicContent.style.borderRadius = '10px';  
});  
});
```

Output:



DOM Manipulation Example

Update Text Content

Add New Element

Remove Last Element

Change Style

The text content has been updated!

This is a new dynamically added element.

DOM Manipulation Example

Update Text Content

Add New Element

Remove Last Element

Change Style

The text content has been updated!

This is a new dynamically added element.

Explanation

HTML

1. Container Structure:

- **<div class="container">**: Wraps the entire content with some styling for layout and design.
- **Buttons**: Buttons to trigger various DOM manipulations.
- **<div id="dynamicContent">**: A placeholder to demonstrate the dynamic changes.

CSS

1. Styling:

- Basic styles to make the page look neat and centered.
- Different button colors for different actions.
- Styles for newly added elements to differentiate them from static content.

JavaScript

1. Event Listeners:

- **DOMContentLoaded**: Ensures the DOM is fully loaded before running the JavaScript code.

2. Update Text Content:

- **updateTextButton.addEventListener('click', ...)**: Changes the text content of the #dynamicContent div when the button is clicked.

3. Add New Element:

- **addButton.addEventListener('click', ...)**: Creates a new <div> element, adds it to the #dynamicContent div, and styles it with the class newElement.

4. Remove Last Element:

- **removeButton.addEventListener('click', ...)**: Removes the last element with the class newElement from the #dynamicContent div.

5. Change Style:

- **changeStyleButton.addEventListener('click', ...)**: Changes the background color, text color, font size, padding, and border radius of the #dynamicContent div.

15. Develop a JavaScript slideshow that displays images with automatic transitions.

1. HTML Structure

Start with the HTML to set up your slideshow container and image elements.

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Slideshow Example</title>

  <link rel="stylesheet" href="styles.css">

</head>

<body>

  <div class="slideshow-container">

    <div class="slide fade">

    </div>

    <div class="slide fade">

    </div>

    <div class="slide fade">

    </div>

    <!-- Add more slides as needed -->

  </div>

  <script src="script.js"></script>

</body>

</html>
```

2. CSS Styling

Add some basic styling for the slideshow. Save this as styles.css.

```
/* Hide all images by default */
```

```
.slide {  
    display: none;  
}
```

```
/* Style the slideshow container */
```

```
.slideshow-container {  
    position: relative;  
    max-width: 100%;  
    margin: auto;  
    overflow: hidden;  
}
```

```
/* Fade effect */
```

```
.fade {  
    animation: fade 2s ease-in-out infinite;  
}
```

```
@keyframes fade {
```

```
    0% { opacity: 0; }  
    20% { opacity: 1; }  
    80% { opacity: 1; }  
    100% { opacity: 0; }
```

```
}
```

```
img {
```

```
    width: 100%;  
    height: auto;
```

```
}
```

3. JavaScript for Automatic Transitions

The JavaScript controls the slideshow transitions. Save this as script.js.

```
document.addEventListener('DOMContentLoaded', () => {  
  let slideIndex = 0;  
  
  function showSlides() {  
    const slides = document.getElementsByClassName('slide');  
  
    // Hide all slides  
    for (let i = 0; i < slides.length; i++) {  
      slides[i].style.display = 'none';  
    }  
  
    // Move to the next slide  
    slideIndex++;  
    if (slideIndex > slides.length) {  
      slideIndex = 1;  
    }  
  
    // Show the current slide  
    slides[slideIndex - 1].style.display = 'block';  
  
    // Change slide every 3 seconds  
    setTimeout(showSlides, 3000);  
  }  
  
  showSlides(); // Initial call to start the slideshow  
});
```

Output:



Explanation

1. HTML Structure:

- **<div class="slideshow-container">**: This is a container that holds all the slideshow images.
- **<div class="slide fade">**: Each slide is contained in a <div> with two classes: slide (for functionality) and fade (for CSS animation).

- ****: This is an image element inside the slide. You can add as many slides as you like.

2. CSS Styling:

- **.slide**: By setting `display: none;`, all slides are hidden initially.
- **.slideshow-container**: This container is styled to ensure that the images fit within it and are centered. `overflow: hidden` ensures that any overflowing content (such as parts of the image) is hidden.
- **.fade**: This class applies a CSS animation named `fade` to the slides.
- **@keyframes fade**: Defines the fade animation that gradually changes the opacity of the slides to create a fading effect. The animation takes 2 seconds (2s) and runs continuously (infinite).

3. JavaScript Functionality:

- `document.addEventListener('DOMContentLoaded', () => { ... });`: This ensures that the JavaScript runs only after the HTML document has fully loaded.
- `let slideIndex = 0;`: This variable keeps track of the current slide index.
- `function showSlides() { ... }`: This function manages the visibility and transition of slides.
- `const slides = document.getElementsByClassName('slide');`: Retrieves all elements with the class `slide`.
- `for (let i = 0; i < slides.length; i++) { slides[i].style.display = 'none'; }`: Hides all slides by setting their `display` property to `none`.
- `slideIndex++;`: Increments the slide index to move to the next slide.
- `if (slideIndex > slides.length) { slideIndex = 1; }`: Resets `slideIndex` to 1 if it exceeds the number of slides, creating a loop.
- `slides[slideIndex - 1].style.display = 'block';`: Displays the current slide by setting its `display` to `block`.
- `setTimeout(showSlides, 3000);`: Calls `showSlides` again after 3 seconds (3000 milliseconds) to continue the slideshow.

Customization

- **Transition Time**: Adjust the 2s in fade animation to change the duration of the fade effect.
- **Slide Interval**: Modify the 3000 milliseconds in `setTimeout` for a different interval between slides.
- **Additional Controls**: Add navigation arrows or dots for manual control if desired

12. Create a JavaScript quiz that presents multiple-choice questions and tracks the user's score..

1. HTML Structure

Start with the HTML to create the structure of your quiz. Save this as index.html.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript Quiz</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="quiz-container">
    <div id="quiz"></div>
    <button id="next">Next Question</button>
    <div id="result"></div>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

2. CSS Styling

Add some basic styling to make the quiz look nicer. Save this as styles.css.

```
body {
  font-family: Arial, sans-serif;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  margin: 0;
  background-color: #f4f4f4;
}

.quiz-container {
  background: #fff;
  border-radius: 8px;
  padding: 20px;
  box-shadow: 0 0 10px rgba(0,0,0,0.1);
  max-width: 600px;
  width: 100%;
}

.question {
  font-size: 1.2em;
```

```

    margin-bottom: 10px;
}

.options {
    list-style: none;
    padding: 0;
}

.options li {
    margin-bottom: 10px;
}

button {
    padding: 10px 20px;
    font-size: 1em;
    border: none;
    border-radius: 5px;
    background-color: #007bff;
    color: #fff;
    cursor: pointer;
}

button:hover {
    background-color: #0056b3;
}

#result {
    margin-top: 20px;
    font-size: 1.2em;
}

```

3. JavaScript Functionality

The JavaScript will handle the quiz logic, including displaying questions, tracking the score, and managing the quiz flow. Save this as script.js.

```

document.addEventListener('DOMContentLoaded', () => {
    const quizData = [
        {
            question: "What is the capital of France?",
            options: ["Berlin", "Madrid", "Paris", "Rome"],
            answer: "Paris"
        },
        {
            question: "Which planet is known as the Red Planet?",
            options: ["Earth", "Mars", "Jupiter", "Saturn"],
            answer: "Mars"
        },
        {
            question: "What is the largest ocean on Earth?",

```

```

        options: ["Atlantic Ocean", "Indian Ocean", "Arctic Ocean", "Pacific Ocean"],
        answer: "Pacific Ocean"
    }
];

let currentQuestionIndex = 0;
let score = 0;

function loadQuestion() {
    const quizContainer = document.getElementById('quiz');
    quizContainer.innerHTML = "";

    const questionData = quizData[currentQuestionIndex];
    const questionElement = document.createElement('div');
    questionElement.classList.add('question');
    questionElement.textContent = questionData.question;
    quizContainer.appendChild(questionElement);

    const optionsList = document.createElement('ul');
    optionsList.classList.add('options');
    questionData.options.forEach(option => {
        const optionElement = document.createElement('li');
        optionElement.innerHTML = `<input type="radio" name="option"
value="${option}"> ${option}`;
        optionsList.appendChild(optionElement);
    });
    quizContainer.appendChild(optionsList);
}

function checkAnswer() {
    const selectedOption = document.querySelector('input[name="option"]:checked');
    if (selectedOption) {
        const answer = selectedOption.value;
        if (answer === quizData[currentQuestionIndex].answer) {
            score++;
        }
        currentQuestionIndex++;
        if (currentQuestionIndex < quizData.length) {
            loadQuestion();
        } else {
            showResult();
        }
    } else {
        alert('Please select an option!');
    }
}

function showResult() {
    const resultContainer = document.getElementById('result');
    resultContainer.textContent = `Your score is ${score} out of ${quizData.length}`;
}

```

```
        document.getElementById('next').style.display = 'none'; // Hide the next button
    }

    document.getElementById('next').addEventListener('click', checkAnswer);

    loadQuestion(); // Load the first question initially
});
```

Output:

What is the capital of France?

☐ Berlin

☐ Madrid

☐ Paris

☐ Rome

Next Question

What is the largest ocean on Earth?

☐ Atlantic Ocean

☐ Indian Ocean

☐ Arctic Ocean

☒ Pacific Ocean

Your score is 2 out of 3

Explanation

HTML:

- **<div id="quiz">**: This is where the quiz questions and options will be dynamically inserted.
- **<button id="next">Next Question</button>**: Button to proceed to the next question.
- **<div id="result">**: This will display the user's final score.

CSS:

- Provides styling to make the quiz look clean and centered.
- Styles for buttons, questions, and options for better user experience.

JavaScript:

- **quizData**: An array of quiz questions, options, and the correct answer.
- **loadQuestion()**: Displays the current question and options.
- **checkAnswer()**: Checks if the selected answer is correct, updates the score, and moves to the next question.
- **showResult()**: Displays the final score after the quiz is complete.
- **Event Listener**: Listens for clicks on the "Next Question" button to proceed.

Flow:

1. **Initial Load**: The first question is loaded when the page is ready.
2. **Next Question**: On clicking the "Next Question" button, the selected answer is checked, and the score is updated.
3. **Completion**: When all questions are answered, the final score is displayed, and the "Next Question" button is hidden.

17. Build a JavaScript program that performs basic string manipulation tasks.

Building a JavaScript program that performs various basic string manipulation tasks can be a great way to understand string handling in JavaScript. In this example, we'll create a program that performs several common string operations, such as reversing a string, changing case, finding substrings, and replacing text.

1. HTML Structure

Create a simple HTML interface to interact with the string manipulation functions. Save this as index.html.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>String Manipulation</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <h1>String Manipulation Program</h1>
    <textarea id="inputString" placeholder="Enter your string here..."></textarea>
    <button onclick="performStringManipulations()">Manipulate String</button>
    <div id="results">
      <p id="reversed">Reversed String: </p>
      <p id="uppercased">Uppercased String: </p>
      <p id="lowercased">Lowercased String: </p>
      <p id="findSubstring">Find Substring: </p>
      <p id="replaceText">Replace Text: </p>
    </div>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

2. CSS Styling

Add some basic styles to make the interface look good. Save this as styles.css.

```
body {
  font-family: Arial, sans-serif;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  margin: 0;
  background-color: #f4f4f4;
```

```

}

.container {
  background: #fff;
  border-radius: 8px;
  padding: 20px;
  box-shadow: 0 0 10px rgba(0,0,0,0.1);
  max-width: 600px;
  width: 100%;
}

textarea {
  width: 100%;
  height: 100px;
  margin-bottom: 10px;
  padding: 10px;
  border: 1px solid #ddd;
  border-radius: 4px;
}

button {
  padding: 10px 20px;
  font-size: 1em;
  border: none;
  border-radius: 5px;
  background-color: #007bff;
  color: #fff;
  cursor: pointer;
}

button:hover {
  background-color: #0056b3;
}

#results p {
  margin: 10px 0;
}

```

3. JavaScript Functionality

The JavaScript will handle the string manipulation logic. Save this as script.js.

```

function performStringManipulations() {
  const input = document.getElementById('inputString').value;

  // Reversing a string
  function reverseString(str) {
    return str.split('').reverse().join("");
  }
}

```



```

// Converting to uppercase
function toUpperCase(str) {
    return str.toUpperCase();
}

// Converting to lowercase
function toLowerCase(str) {
    return str.toLowerCase();
}

// Finding a substring
function findSubstring(str, substring) {
    return str.includes(substring) ? `Substring "${substring}" found!` : `Substring "${substring}" not found.`;
}

// Replacing text
function replaceText(str, oldText, newText) {
    return str.replace(new RegExp(oldText, 'g'), newText);
}

// Getting user input for substring and replacement
const substringToFind = prompt('Enter the substring to find:');
const textToReplace = prompt('Enter the text to replace:');
const replacementText = prompt('Enter the replacement text:');

// Perform manipulations
const reversed = reverseString(input);
const uppercased = toUpperCase(input);
const lowercased = toLowerCase(input);
const foundSubstring = findSubstring(input, substringToFind);
const replacedText = replaceText(input, textToReplace, replacementText);

// Display results
document.getElementById('reversed').textContent = `Reversed String: ${reversed}`;
document.getElementById('uppercased').textContent = `Uppercased String: ${uppercased}`;
document.getElementById('lowercased').textContent = `Lowercased String: ${lowercased}`;
document.getElementById('findSubstring').textContent = foundSubstring;
document.getElementById('replaceText').textContent = `Replaced Text: ${replacedText}`;
}

```

Output:

String Manipulation Program

Computer Applications

Manipulate String

Reversed String: snoitacilppA retupmoC

Uppercased String: COMPUTER APPLICATIONS

Lowercased String: computer applications

Substring "com" not found.

Replaced Text: Computer Applications

Explanation

HTML:

- `<textarea id="inputString">`: Allows users to input their string.
- `<button onclick="performStringManipulations()>`: Triggers the JavaScript function when clicked.
- `<div id="results">`: Displays the results of the string manipulations.

CSS:

- Provides basic styling for the textarea, button, and results to make the UI user-friendly.

JavaScript:

- **reverseString(str)**: Reverses the given string by splitting it into characters, reversing the array, and then joining it back together.
- **toUpperCase(str)**: Converts the string to uppercase.
- **toLowerCase(str)**: Converts the string to lowercase.
- **findSubstring(str, substring)**: Checks if a substring exists within the main string and returns a message accordingly.
- **replaceText(str, oldText, newText)**: Replaces all occurrences of oldText with newText using a regular expression.

- **performStringManipulations():** Retrieves user input, performs the string manipulations, and displays the results.

18. Create a JavaScript program that interacts with the browser's local storage to store and retrieve data.

Interacting with the browser's local storage allows you to store data persistently across page reloads and browser sessions. Local storage is part of the Web Storage API and provides a way to store key-value pairs in a web browser.

Let's create a simple JavaScript program that demonstrates how to use local storage. This program will include:

1. Saving data to local storage.
2. Retrieving data from local storage.
3. Displaying the stored data on the webpage.
4. Clearing data from local storage.

1. HTML Structure

We'll create a basic HTML interface that allows users to input data, save it, retrieve it, and clear it from local storage. Save this as index.html.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Local Storage Example</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <h1>Local Storage Example</h1>
    <input type="text" id="dataKey" placeholder="Enter key">
    <input type="text" id="dataValue" placeholder="Enter value">
    <button onclick="saveData()">Save Data</button>
    <button onclick="retrieveData()">Retrieve Data</button>
    <button onclick="clearData()">Clear Data</button>
    <div id="output"></div>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

2. CSS Styling

Add some basic styling to make the interface look better. Save this as styles.css.

```
body {
  font-family: Arial, sans-serif;
  display: flex;
  justify-content: center;
```

```

    align-items: center;
    height: 100vh;
    margin: 0;
    background-color: #f4f4f4;
}

.container {
    background: #fff;
    border-radius: 8px;
    padding: 20px;
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
    max-width: 500px;
    width: 100%;
}

input {
    width: calc(100% - 22px);
    padding: 10px;
    margin: 5px 0;
    border: 1px solid #ddd;
    border-radius: 4px;
}

button {
    padding: 10px 20px;
    font-size: 1em;
    border: none;
    border-radius: 5px;
    background-color: #007bff;
    color: #fff;
    cursor: pointer;
    margin: 5px;
}

button:hover {
    background-color: #0056b3;
}

#output {
    margin-top: 20px;
}

```

3. JavaScript Functionality

Now, let's implement the JavaScript functionality for interacting with local storage. Save this as script.js.

```

// Save data to local storage
function saveData() {
    const key = document.getElementById('dataKey').value;

```

```

const value = document.getElementById('dataValue').value;

if (key && value) {
    localStorage.setItem(key, value);
    document.getElementById('output').textContent = `Data saved: ${key} = ${value}`;
} else {
    document.getElementById('output').textContent = 'Please enter both key and value.';
}
}

// Retrieve data from local storage
function retrieveData() {
    const key = document.getElementById('dataKey').value;
    const value = localStorage.getItem(key);

    if (key && value) {
        document.getElementById('output').textContent = `Retrieved data: ${key} = ${value}`;
    } else {
        document.getElementById('output').textContent = key ? 'No data found for the provided key.' : 'Please enter a key.';
    }
}

// Clear data from local storage
function clearData() {
    const key = document.getElementById('dataKey').value;

    if (key) {
        localStorage.removeItem(key);
        document.getElementById('output').textContent = `Data for key "${key}" has been cleared.`;
    } else {
        document.getElementById('output').textContent = 'Please enter a key to clear.';
    }
}

```

Output:

Local Storage Example

roll no

25

Save Data

Retrieve Data

Clear Data

Data saved: roll no = 25

Local Storage Example

roll no

25

Save Data

Retrieve Data

Clear Data

Retrieved data: roll no = 25

Local Storage Example

roll no

25

Save Data

Retrieve Data

Clear Data

Data for key "roll no" has been cleared.

Explanation

HTML:

- **<input>**: Two text inputs for entering the key and value.
- **<button>**: Buttons to trigger saving, retrieving, and clearing data.
- **<div id="output">**: A div to display messages and results.

CSS:

- Provides styling for inputs, buttons, and the output area to make the interface user-friendly.

JavaScript:

- **saveData()**: Retrieves the key and value from the inputs, then stores them in local storage using `localStorage.setItem()`. Displays a confirmation message or an error if inputs are missing.
- **retrieveData()**: Retrieves the value associated with the key from local storage using `localStorage.getItem()`. Displays the result or an error message if the key is missing or no data is found.
- **clearData()**: Removes the data associated with the key from local storage using `localStorage.removeItem()`. Displays a confirmation message or an error if the key is missing.

19. Build a website that retrieves data from a web API using JavaScript and displays it dynamically.

To create a website that retrieves data from the Wikipedia API and displays it dynamically using JavaScript, follow these detailed steps:

1. Project Structure

Create a folder for your project and within it, create the following files:

- index.html
- styles.css
- script.js

2. Create index.html

This file structures the content of your webpage:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Wikipedia Search</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <h1>Wikipedia Search</h1>
  <input type="text" id="searchInput" placeholder="Search Wikipedia">
  <button id="searchButton">Search</button>
  <div id="resultsContainer"></div>

  <script src="script.js"></script>
</body>
</html>
```

Explanation:

- `<input type="text" id="searchInput" placeholder="Search Wikipedia">`: An input field for users to enter their search query.
- `<button id="searchButton">Search</button>`: A button to trigger the search.
- `<div id="resultsContainer"></div>`: A container to display the search results.

3. Create styles.css

This file provides basic styling:

```
body {
  font-family: Arial, sans-serif;
  margin: 20px;
```

```
    background-color: #f4f4f4;
    text-align: center;
}

h1 {
    color: #333;
}

input[type="text"] {
    padding: 10px;
    font-size: 16px;
    width: 300px;
    margin-right: 10px;
}

button {
    padding: 10px 20px;
    font-size: 16px;
    cursor: pointer;
    background-color: #007BFF;
    color: white;
    border: none;
    border-radius: 5px;
}

button:hover {
    background-color: #0056b3;
}

#resultsContainer {
    margin-top: 20px;
    text-align: left;
}

.result {
    background-color: white;
    border: 1px solid #ddd;
    border-radius: 5px;
    padding: 15px;
    margin: 10px auto;
    max-width: 800px;
}

.result h2 {
    margin-top: 0;
}

.result a {
    color: #007BFF;
    text-decoration: none;
}
```

```

}

.result a:hover {
  text-decoration: underline;
}

```

Explanation:

- Styles the page to make it visually appealing, with a clean layout and responsive design.

4. Create script.js

This file handles fetching data from the Wikipedia API and updating the webpage:

```

document.addEventListener('DOMContentLoaded', () => {
  const searchButton = document.getElementById('searchButton');
  const searchInput = document.getElementById('searchInput');
  const resultsContainer = document.getElementById('resultsContainer');

  searchButton.addEventListener('click', () => {
    const query = searchInput.value.trim();
    if (query === "") {
      resultsContainer.innerHTML = '<p>Please enter a search term.</p>';
      return;
    }

    const apiUrl = `https://en.wikipedia.org/w/api.php?action=query&format=json&prop=extracts&titles=${encodeURIComponent(query)}&exintro&origin=*`;

    fetch(apiUrl)
      .then(response => response.json())
      .then(data => {
        resultsContainer.innerHTML = ""; // Clear previous results

        const pages = data.query.pages;
        for (const pageId in pages) {
          const page = pages[pageId];
          const resultElement = document.createElement('div');
          resultElement.className = 'result';
          resultElement.innerHTML = `
            <h2><a href="https://en.wikipedia.org/?curid=${pageId}"
target="_blank">${page.title}</a></h2>
            <p>${page.extract}</p>
          `;
          resultsContainer.appendChild(resultElement);
        }
      })
      .catch(error => {
        resultsContainer.innerHTML = '<p>Error fetching data.</p>';
      })
  });
}

```

```

        console.error('Error:', error);
    });
});
});

```

Output:

Wikipedia Search

Maharashtra

Maharashtra (ISO: *Mahārāṣṭra*; Marathi: [məɦaːrɑːʂtɾə]) is a state in the western peninsular region of India occupying a substantial portion of the Deccan Plateau. It is bordered by the Arabian Sea to the west, the Indian states of Karnataka and Goa to the south, Telangana to the southeast and Chhattisgarh to the east, Gujarat and Madhya Pradesh to the north, and the Indian union territory of Dadra and Nagar Haveli and Daman and Diu to the northwest. Maharashtra is the second-most populous state in India.

The state is divided into 6 divisions and 36 districts, with the state capital being Mumbai, the most populous urban area in India, and Nagpur serving as the winter capital. The Godavari and Krishna are the two major rivers in the state and forests cover 16.47 per cent of the state's geographical area. The state is home to six UNESCO World Heritage Sites: Ajanta Caves, Ellora Caves, Elephanta Caves, Chhatrapati Shivaji Terminus (formerly Victoria Terminus), The Victorian Gothic and Art Deco Ensembles of Mumbai and The Western Ghats, a heritage site made up of 39 individual properties of which 4 are in Maharashtra. The State is the single largest contributor to India's economy with a share of 14 per cent in all-India nominal GDP. The economy of Maharashtra is the largest in India, with a gross state domestic product (GSDP) of ₹42.5 trillion (US\$510 billion) and GSDP per capita of ₹335,247 (US\$4,000). The service sector dominates the state's economy, accounting for 69.3 per cent of the value of the output of the country. Although agriculture accounts for 12 per cent of the state GDP, it employs nearly half the population of the state.

Maharashtra is one of the most industrialised states in India. The state's capital, Mumbai, is India's financial and commercial capital. India's largest stock exchange Bombay Stock Exchange, the oldest in Asia, is located in the city, as is National Stock Exchange, which is the second largest stock exchange in India and one of world's largest derivatives exchanges. The state has played a significant role in the country's social and political life and is widely considered a leader in terms of agricultural and industrial production, trade and transport, and education. Maharashtra is the ninth-highest ranking among Indian states in the human development index.

The region that encompasses the state has a history going back many millennia. Notable dynasties that ruled the region include the Asmakas, the Mauryas, the Satavahanas, the Western Satraps, the Abhiras, the Vakatakas, the Chalukyas, the Rashtrakutas, the Western Chalukyas, the Seuna Yadavas, the Khaljis, the Tughlaqs, the Bahamanis and the Mughals. In the early nineteenth century, the region was divided between the Dominions of the Peshwa in the Maratha Confederacy and the Nizamate of Hyderabad. After two wars and the proclamation of the Indian Empire, the region became a part of the Bombay Province, the Berar Province and the Central Provinces of India under direct British rule and the Deccan States Agency under Crown suzerainty. Between 1950 and 1956, the Bombay Province became the Bombay State in the Indian Union, and Berar, the Deccan states and the Gujarat states were merged into the Bombay State. On 1 May 1960, the State of Bombay was bifurcated into the State of Maharashtra and State of Gujarat after a long struggle for special state for Marathi language speaking people through Samyukta Maharashtra Movement (transl. United Maharashtra movement).

Explanation:

- `document.addEventListener('DOMContentLoaded', ...)`: Ensures the script runs after the page has fully loaded.
- `searchButton.addEventListener('click', ...)`: Adds an event listener to the search button to trigger the API request.
- `const query = searchInput.value.trim();`: Retrieves and trims the user's search query.
- `const apiUrl = ...`: Constructs the URL for the Wikipedia API request. The `&origin=*` parameter helps handle CORS (Cross-Origin Resource Sharing) issues.
- `fetch(apiUrl)`: Sends the API request.

- `.then(response => response.json())`: Processes the response and converts it to JSON.
- `const pages = data.query.pages;`: Extracts the pages data from the API response.
- `for (const pageId in pages) { ... }`: Iterates through the pages and creates HTML elements for each result.
- `resultElement.innerHTML = ...`: Populates the result elements with the page title and extract.
- `resultsContainer.appendChild(resultElement);`: Adds each result to the results container.
- `.catch(error => { ... })`: Handles any errors that occur during the fetch process.

How to Test

1. **Open index.html in your web browser** by double-clicking the file or using your code editor's preview feature.
2. **Type a search query** into the input field and click the "Search" button.
3. **View the results** displayed dynamically below the button.

