**1)ii --- Create a responsive page with Flexbox**

**Aim**

To create a responsive web page layout using CSS Flexbox that adapts to various screen sizes and provides a flexible, modern user interface structure.

**Algorithm**

1. Start the HTML structure with the necessary boilerplate (<!DOCTYPE html> etc.).
2. Create a main container (div) that will act as a Flexbox container.
3. Inside the container, add multiple child elements (e.g., header, sidebar, content, footer).
4. In the CSS:
   * Use display: flex; on the container.
   * Use flex-direction, justify-content, and align-items to control layout.
   * Use media queries to adjust flex properties for responsiveness.
5. Add styles (widths, colors, padding, etc.) to differentiate the elements.
6. Test responsiveness by resizing the browser or using developer tools.
7. End

**Program:**

**index.html**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

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

  <title>Responsive Flexbox Layout</title>

  <link rel="stylesheet" href="index.css" />

</head>

<body>

  <header>

    <h1>My Flexbox Page</h1>

    <nav>

      <a href="#">Home</a>

      <a href="#">About</a>

      <a href="#">Projects</a>

      <a href="#">Contact</a>

    </nav>

  </header>

  <div class="container">

    <main>

      <h2>Main Content</h2>

      <p>This is the main content area. It takes up more space than the sidebar.</p>

    </main>

    <aside>

      <h2>Sidebar</h2>

      <p>This is the sidebar. It contains extra information or links.</p>

    </aside>

  </div>

  <footer>

    <p>© 2025 My Flexbox Layout. All rights reserved.</p>

  </footer>

</body>

</html>

**index.css**

\* {

  margin: 0;

  padding: 0;

  box-sizing: *border-box*;

}

body {

  font-family: 'Segoe UI', *sans-serif*;

  background: #f0f0f0;

  color: #333;

}

header {

  background-color: #333;

  color: *white*;

  padding: 20px;

  text-align: *center*;

}

nav {

  margin-top: 10px;

}

nav a {

  color: *white*;

  margin: 0 15px;

  text-decoration: *none*;

}

nav a:hover {

  text-decoration: *underline*;

}

.container {

  display: *flex*;

  padding: 20px;

  gap: 20px;

}

main {

  flex: 3;

  background: *white*;

  padding: 20px;

  box-shadow: 0 2px 5px rgba(0,0,0,0.1);

}

aside {

  flex: 1;

  background: #fff8dc;

  padding: 20px;

  box-shadow: 0 2px 5px rgba(0,0,0,0.1);

}

footer {

  background: #333;

  color: *white*;

  text-align: *center*;

  padding: 15px;

  margin-top: 20px;

}

@media (max-width: 768px) {

  .container {

    flex-direction: *column*;

  }

  nav a {

    display: *inline-block*;

    margin: 5px;

  }

}

**Result**

A responsive web page layout is successfully created using CSS Flexbox.  
The layout adapts to screen size changes by stacking the sidebar and content vertically on smaller screens.

**1)iii----- Create a web page using multi-column blog layout**

**Aim**

To design a multi-column blog layout using HTML and CSS that displays blog posts in multiple columns, providing a clean and readable structure.

**Algorithm**

1. Start with a basic HTML5 structure.
2. Create a main container <div class="blog-container">.
3. Inside the container, add multiple blog post elements (<div class="post">).
4. Apply CSS Multi-column layout using the column-count or flexbox/grid layout.
5. Style each blog post for spacing, padding, and readability.
6. Use media queries to make the layout responsive for smaller screens.
7. Test the layout by resizing the browser.
8. End

**Program**

**index.html**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

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

  <title>Multi-Column Blog Layout</title>

  <link rel="stylesheet" href="index.css" />

</head>

<body>

  <header>

    <h1>My Blog</h1>

  </header>

  <section class="blog-container">

    <article class="blog-post">

      <h2>Post One</h2>

      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque euismod, nisi vel consectetur.</p>

      <a href="#">Read More</a>

    </article>

    <article class="blog-post">

      <h2>Post Two</h2>

      <p>Aliquam erat volutpat. Duis eget tincidunt augue, in laoreet libero. Nullam at blandit urna.</p>

      <a href="#">Read More</a>

    </article>

    <article class="blog-post">

      <h2>Post Three</h2>

      <p>Curabitur lacinia, nulla ac malesuada posuere, lectus purus mattis ligula, ac porta purus enim.</p>

      <a href="#">Read More</a>

    </article>

    <article class="blog-post">

      <h2>Post Four</h2>

      <p>Donec dignissim diam at lorem convallis, ac rutrum arcu tincidunt. Sed porta sapien vitae ante.</p>

      <a href="#">Read More</a>

    </article>

    <article class="blog-post">

      <h2>Post Five</h2>

      <p>Sed nec ipsum ac felis tristique pharetra nec nec nulla. Suspendisse potenti.</p>

      <a href="#">Read More</a>

    </article>

    <article class="blog-post">

      <h2>Post Six</h2>

      <p>Phasellus tristique orci quis lacus pulvinar, eget rhoncus nulla porta. In nec tortor id.</p>

      <a href="#">Read More</a>

    </article>

  </section>

  <footer>

    <p>© 2025 My Blog. All rights reserved.</p>

  </footer>

</body>

</html>

**Index.css**

\* {

  margin: 0;

  padding: 0;

  box-sizing: *border-box*;

}

body {

  font-family: 'Segoe UI', *sans-serif*;

  background-color: #f9f9f9;

  color: #333;

}

header {

  background-color: #333;

  color: *white*;

  padding: 20px;

  text-align: *center*;

}

.blog-container {

  display: *grid*;

  grid-template-columns: repeat(*auto-fit*, minmax(300px, 1fr));

  gap: 20px;

  padding: 30px;

  max-width: 1200px;

  margin: *auto*;

}

.blog-post {

  background-color: *white*;

  padding: 20px;

  border-left: 5px *solid* #007BFF;

  box-shadow: 0 2px 6px rgba(0, 0, 0, 0.1);

}

.blog-post h2 {

  margin-bottom: 10px;

  color: #007BFF;

}

.blog-post p {

  margin-bottom: 10px;

}

.blog-post a {

  color: #007BFF;

  text-decoration: *none*;

  font-weight: *bold*;

}

.blog-post a:hover {

  text-decoration: *underline*;

}

footer {

  background-color: #333;

  color: *white*;

  text-align: *center*;

  padding: 15px;

  margin-top: 30px;

}

**Result**

A multi-column blog layout is successfully created. Blog posts are displayed in three columns on large screens, two on medium screens, and stacked on small screens, ensuring responsive readability.

**2)i 🡪 "Form Validation using JavaScript"**

**Aim**

To design a form validation system using JavaScript that checks user inputs for correctness before submission.

**Algorithm**

1. Start by creating an HTML form with input fields (e.g., Name, Email, Password).
2. Add a submit button with an onclick or onsubmit event.
3. In the JavaScript function:
   * Retrieve values from the input fields.
   * Check for empty fields.
   * Validate email format using regular expressions.
   * Validate password length or content.
4. If validation fails, prevent form submission and show an error message.
5. If validation passes, allow form submission or display success.
6. End

**Program**

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

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

  <title>Form Validation</title>

  <link rel="stylesheet" href="index.css" />

</head>

<body>

  <div class="form-container">

    <h2>Register</h2>

    <form id="registerForm">

      <label for="name">Name:</label>

      <input type="text" id="name" />

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

      <input type="email" id="email" />

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

      <input type="password" id="password" />

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

      <p id="errorMsg" class="error"></p>

      <p id="successMsg" class="success"></p>

    </form>

  </div>

  <script src="index.js"></script>

</body>

</html>

**Index.css**

body {

  font-family: *Arial*, *sans-serif*;

  background: #f9f9f9;

  padding: 40px;

}

.form-container {

  max-width: 400px;

  margin: *auto*;

  background: *white*;

  padding: 30px;

  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);

}

input, label, button {

  display: *block*;

  width: 100%;

  margin-bottom: 15px;

  font-size: 16px;

}

input {

  padding: 8px;

}

button {

  background: #007BFF;

  color: *white*;

  padding: 10px;

  border: *none*;

  cursor: *pointer*;

}

button:hover {

  background: #0056b3;

}

.error {

  color: *red*;

  font-weight: *bold*;

}

.success {

  color: *green*;

  font-weight: *bold*;

}

**Index.js**

document.getElementById("registerForm").addEventListener("submit", function(e) {

  e.preventDefault();

  const name = document.getElementById("name").value.trim();

  const email = document.getElementById("email").value.trim(); *// no validation*

  const password = document.getElementById("password").value.trim();

  const errorMsg = document.getElementById("errorMsg");

  const successMsg = document.getElementById("successMsg");

  errorMsg.textContent = "";

  successMsg.textContent = "";

  if (name === "" || email === "" || password === "") {

    errorMsg.textContent = "All fields are required.";

    return;

  }

  if (password.length < 8) {

    errorMsg.textContent = "Password must be at least 8 characters.";

    return;

  }

  successMsg.textContent = "Form submitted successfully!";

});

**Result**

A form with fields for Name, Email, and Password is successfully validated using JavaScript.

* Empty fields show an error.
* Incorrect email formats are rejected.
* Short passwords are not accepted.  
  The user gets real-time feedback before submitting the form.

**2)ii "To-do List Application using JavaScript"**

**Aim**

To develop a To-Do List Application using JavaScript that allows users to add, display, and delete tasks interactively.

**Algorithm**

1. Start with an HTML structure containing:
   * A text input field.
   * An "Add Task" button.
   * A section to display the task list.
2. On clicking the "Add" button:
   * Read the input value.
   * Validate if it's not empty.
   * Create a new list item and add it to the task list.
3. Each task has a "Delete" button to remove it from the list.
4. Use JavaScript DOM manipulation to:
   * Append tasks dynamically.
   * Remove tasks using removeChild() or element.remove().
5. Style the application for better user experience.
6. End

**Program**

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

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

  <title>Simple To-Do List</title>

  <link rel="stylesheet" href="index.css" />

</head>

<body>

  <h2>To-Do List</h2>

  <input type="text" id="taskInput" placeholder="Add a task">

  <button onclick="addTask()">Add</button>

  <ul id="taskList"></ul>

  <script src="index.js"></script>

</body>

</html>

**Index.css**

body {

  font-family: *sans-serif*;

  text-align: *center*;

  padding: 30px;

}

input, button {

  padding: 10px;

  margin: 5px;

}

ul {

  list-style: *none*;

  padding: 0;

}

li {

  padding: 10px;

  margin: 5px *auto*;

  max-width: 300px;

  border: 1px *solid* #ccc;

  display: *flex*;

  justify-content: *space-between*;

  cursor: *pointer*;

}

.done {

  text-decoration: *line-through*;

  color: *gray*;

}

.delete {

  background: *red*;

  color: *white*;

  border: *none*;

  cursor: *pointer*;

}

**Index.js**

function addTask() {

  const input = document.getElementById("taskInput");

  const list = document.getElementById("taskList");

  if (input.value.trim() === "") return;

  const li = document.createElement("li");

  li.innerHTML = `

    <span onclick="this.classList.toggle('done')">${input.value}</span>

    <button class="delete" onclick="this.parentElement.remove()">X</button>

  `;

  list.appendChild(li);

  input.value = "";

}

**Result**

The To-Do List Application allows users to:

* Add new tasks.
* See them displayed instantly.
* Remove tasks by clicking the "Delete" button.  
  The app uses JavaScript DOM manipulation for interactive behavior.

**2)iii "Image Slider using JavaScript"**

**Aim**

To design and implement an Image Slider using JavaScript that allows users to view images in a slideshow format with previous and next controls.

**Algorithm**

1. Start with a basic HTML structure to hold:
   * Image container.
   * Navigation buttons (Previous, Next).
2. Store multiple image paths in a JavaScript array.
3. Initialize an index to track the current image.
4. On clicking Next, increment the index and display the next image.
5. On clicking Previous, decrement the index and show the previous image.
6. Use setAttribute() or src to update the image dynamically.
7. Loop to the start or end when reaching the limits.
8. Style the slider for clean navigation.
9. End

**Program**

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>Simple Image Slider</title>

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

</head>

<body>

  <h2>Simple Image Slider</h2>

  <div id="slider">

    <img src="assets/html.png" class="active" alt="1">

    <img src="assets/css.png" alt="2">

    <img src="assets/js.png" alt="3">

  </div>

  <button class="btn" onclick="prev()">❮ Prev</button>

  <button class="btn" onclick="next()">Next ❯</button>

  <script src="index.js"></script>

</body>

</html>

**Index.css**

body {

  text-align: *center*;

  font-family: *Arial*, *sans-serif*;

}

#slider {

  width: 400px;

  height: 250px;

  margin: 40px *auto*;

  position: *relative*;

}

#slider img {

  width: 100%;

  height: 100%;

  display: *none*;

}

#slider img.active {

  display: *block*;

}

.btn {

  margin: 10px;

  padding: 5px 15px;

  font-size: 16px;

}

**Index.js**

let slides = document.querySelectorAll("#slider img");

let index = 0;

function showSlide(i) {

  slides.forEach(img => img.classList.remove("active"));

  slides[i].classList.add("active");

}

function next() {

  index = (index + 1) % slides.*length*;

  showSlide(index);

}

function prev() {

  index = (index - 1 + slides.*length*) % slides.*length*;

  showSlide(index);

}

setInterval(next, 3000);

**Result**

An interactive image slider is created using JavaScript.

* Clicking Next cycles forward through the images.
* Clicking Previous cycles backward.
* Images loop back when the beginning or end is reached.

**2) iv "Calculator App using JavaScript"**

**Aim**

To create a Calculator Web Application using JavaScript that performs basic arithmetic operations like addition, subtraction, multiplication, and division.

**Algorithm**

1. Start with an HTML structure for:
   * A display screen.
   * Buttons for digits (0-9) and operations (+, –, ×, ÷).
2. Attach event listeners to each button.
3. On button click:
   * Display the number or operator on the screen.
   * Store the expression as a string.
4. On pressing =, evaluate the string using JavaScript’s eval() function.
5. Show the result in the display field.
6. Handle C or Clear button to reset the display.
7. End

**Program**

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>Simple Calculator</title>

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

</head>

<body>

  <div class="calculator">

    <input type="text" id="display" disabled>

    <div class="buttons">

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

      <button onclick="appendValue('/')">/</button>

      <button onclick="appendValue('\*')">\*</button>

      <button onclick="appendValue('-')">-</button>

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

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

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

      <button onclick="appendValue('+')">+</button>

      <button onclick="appendValue('4')">4</button>

      <button onclick="appendValue('5')">5</button>

      <button onclick="appendValue('6')">6</button>

      <button onclick="calculate()">=</button>

      <button onclick="appendValue('1')">1</button>

      <button onclick="appendValue('2')">2</button>

      <button onclick="appendValue('3')">3</button>

      <button onclick="appendValue('0')">0</button>

      <button onclick="appendValue('.')">.</button>

    </div>

  </div>

  <script src="index.js"></script>

</body>

</html>

**Index.css**

body {

  display: *flex*;

  justify-content: *center*;

  align-items: *center*;

  height: 100vh;

  background: #f5f5f5;

  font-family: *Arial*, *sans-serif*;

}

.calculator {

  border: 1px *solid* #ccc;

  padding: 20px;

  background: *white*;

  border-radius: 10px;

  box-shadow: 2px 2px 10px rgba(0,0,0,0.1);

}

#display {

  width: 100%;

  height: 40px;

  font-size: 20px;

  margin-bottom: 10px;

  text-align: *right*;

  padding: 5px;

}

.buttons {

  display: *grid*;

  grid-template-columns: repeat(4, 50px);

  gap: 10px;

  justify-content: *center*;

}

button {

  padding: 10px;

  font-size: 18px;

  cursor: *pointer*;

}

**Index.js**

function appendValue(value) {

  document.getElementById("display").value += value;

}

function clearDisplay() {

  document.getElementById("display").value = "";

}

function calculate() {

  try {

    const result = eval(document.getElementById("display").value);

    document.getElementById("display").value = result;

  } catch {

    document.getElementById("display").value = "Error";

  }

}

**Result**

A fully functional calculator application is created using JavaScript.  
It supports:

* Number input and basic arithmetic operations.
* Displaying real-time expression and result.
* Clearing/resetting input using the Clear (C) button.