**Module 3 – Mernstack – CSS and CSS3:**

CSS Selectors & Styling

Theory Assignment:

**Question 1: What is a CSS selector? Provide examples of element, class, and ID selectors.**

**Ans:**

A **CSS selector** is a pattern used to select and style specific HTML elements on a webpage. It tells the browser which element(s) the CSS rules should apply to.

**Types of CSS Selectors with Examples:**

**1. Basic Selectors:**

**A. Element Selector**

* **Definition:** Targets all elements of a specific type.
* **Example:**

p {

color: blue;

}

**B. Class Selector**

* **Definition:** Targets all elements with a specific class attribute.
* **Syntax:** Use a dot . followed by the class name.
* **Example:**

.highlight {

background-color: yellow;

}

**C. ID Selector**

* **Definition:** Targets a single element with a specific ID.
* **Syntax:** Use a hash # followed by the ID name.
* **Example:**

<h1 id="main-title">Welcome</h1>

#main-title {

font-size: 24px;

}

2. **Grouping Selectors:**

Style multiple elements at once:

h1, h2, p {

font-family: Arial;

}

There are many more css selectors…

**Question 2: Explain the concept of CSS specificity. How do conflicts between multiple stylesget resolved?**

**Ans:**

**CSS specificity** is a set of rules that determine **which CSS rule is applied** when multiple rules target the same HTML element.

When there is a conflict between styles, the rule with **higher specificity** wins.

**Specificity is calculated based on the types of selectors used. Each selector type has a point value:**

| **Selector Type** | **Specificity Value** |
| --- | --- |
| Inline styles | 1000 |
| ID selector (#id) | 0100 |
| Class selector (.class), attribute [type="text"], pseudo-class (:hover) | 0010 |
| Element selector (div, p), pseudo-element (::before) | 0001 |
| Universal selector (\*), combinators (+, >, ~), and group selectors (,) | 0000 (no value) |

**Example of Specificity in Action:**

<p id="message" class="alert">Hello!</p>

/\* Specificity: 0001 \*/

p {

color: black;

}

/\* Specificity: 0010 \*/

.alert {

color: green;

}

/\* Specificity: 0100 \*/

#message {

color: red;

}

**Final Color: red  
Because #message has the highest specificity.**

**Question 3: What is the difference between internal, external, and inline CSS? Discuss theadvantages and disadvantages of each approach.**

**Ans:**

CSS (Cascading Style Sheets) can be applied to HTML documents in three main ways: internal, external, and inline. Each method has its own advantages and disadvantages.

**1. Internal CSS:**

**Definition**: Internal CSS is defined within the **<style>** tag in the **<head>** section of an HTML document.

**Example:**

<!DOCTYPE html>

<html>

<head>

<style>

body {

background-color: lightblue;

}

h1 {

color: navy;

margin-left: 20px;

}

</style>

</head>

<body>

<h1>This is a heading</h1>

</body>

</html>

**Advantages**:

* **Convenience**: Useful for single-page websites where styles are not reused.
* **Easy to manage**: All styles are in one place, making it easier to edit.

**Disadvantages**:

* **Limited reusability**: Styles cannot be reused across multiple pages.
* **Increased page load time**: Each page with internal CSS must load its own styles, which can slow down loading times.

**2. External CSS:**

**Definition**: External CSS is defined in a separate **.css** file, which is linked to the HTML document using the **<link>** tag.

**Example:**

**<!-- index.html -->**

<!DOCTYPE html>

<html>

<head>

<link rel="stylesheet" type="text/css" href="styles.css">

</head>

<body>

<h1>This is a heading</h1>

</body>

</html>

**/\* styles.css \*/**

body {

background-color: lightblue;

}

h1 {

color: navy;

margin-left: 20px;

}

**Advantages**:

* **Reusability**: The same CSS file can be linked to multiple HTML pages, promoting consistency.
* **Separation of concerns**: Keeps HTML and CSS separate, making both easier to manage and maintain.
* **Reduced page load time**: Browsers cache external CSS files, which can speed up loading times for subsequent page visits.

**Disadvantages**:

* **Additional HTTP request**: Requires an extra request to load the CSS file, which can slow down the initial load time.
* **Dependency on file**: If the CSS file is missing or incorrectly linked, styles will not be applied.

**3 Inline Css:**

**Definition**: Inline CSS is applied directly to HTML elements using the **style** attribute.

**Example**:

<!DOCTYPE html>

<html>

<body>

<h1 style="color: navy; margin-left: 20px;">This is a heading</h1>

</body>

</html>

**Advantages**:

* **Quick styling**: Useful for quick, one-off styles that do not need to be reused.
* **Overrides other styles**: Inline styles have the highest specificity, allowing them to override external and internal styles.

**Disadvantages**:

* **Poor maintainability**: Styles are mixed with HTML, making it harder to manage and update.
* **Reusability issues**: Inline styles cannot be reused across multiple elements or pages.
* **Increased HTML size**: Can lead to larger HTML files, which may affect loading times.

**Lab Assignment:**

Task: Style the contact form (created in the HTML Forms lab) using external CSS.

The following should be implemented:

• Change the background color of the form.

• Add padding and margins to form fields.

• Style the submit button with a hover effect.

• Use class selectors for styling common elements and ID selectors for unique elements.

**Ans:**

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Contact Form</title>

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

</head>

<body>

<form id="contact-form">

<h2>Contact Us</h2>

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

<input type="text" id="name" class="input-field" placeholder="Your Name">

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

<input type="email" id="email" class="input-field" placeholder="Your Email">

<label for="message">Message:</label>

<textarea id="message" class="input-field" rows="5" placeholder="Your Message"></textarea>

<input type="submit" id="submit-btn" value="Send">

</form>

</body>

</html>

**Style.css:**

/\* Style the entire form \*/

#contact-form {

background-color: #f2f2f2;

width: 400px;

margin: 50px auto;

padding: 20px;

border-radius: 10px;

box-shadow: 0 0 10px #ccc;

font-family: Arial, sans-serif;

}

/\* Common style for inputs and textarea \*/

.input-field {

width: 100%;

padding: 10px;

margin: 10px 0 20px 0;

border: 1px solid #ccc;

border-radius: 5px;

box-sizing: border-box;

font-size: 16px;

}

/\* Submit button styling \*/

#submit-btn {

background-color: #4CAF50;

color: white;

padding: 12px 20px;

border: none;

border-radius: 5px;

cursor: pointer;

width: 100%;

font-size: 16px;

transition: background-color 0.3s ease;

}

/\* Hover effect on submit button \*/

#submit-btn:hover {

background-color: #45a049;

}

**CSS Box Model:  
Theory Assignment:**

**Question 1: Explain the CSS box model and its components (content, padding, border, margin). How does each affect the size of an element?**

**Ans:**

The CSS box model is a fundamental concept in web design that describes how elements are structured and how their dimensions are calculated. Every HTML element is represented as a rectangular box, and the box model consists of several components: content, padding, border, and margin. Understanding these components is crucial for effective layout and design.

**Components of the CSS Box Model**

1. **Content**:
   * **Definition**: This is the innermost part of the box model, where text, images, or other media are displayed.
   * **Size**: The size of the content area is defined by the **width** and **height** properties in CSS. The content area is the actual space that the content occupies.
2. **Padding**:
   * **Definition**: Padding is the space between the content and the border of the box. It creates an inner buffer around the content.
   * **Size**: Padding can be set using the **padding** property, which can take values for all four sides (top, right, bottom, left). Padding increases the overall size of the element because it adds space inside the box, pushing the border outward.
3. **Border**:
   * **Definition**: The border surrounds the padding (if any) and the content. It can be styled with different widths, colors, and styles (solid, dashed, etc.).
   * **Size**: The border size is defined using the **border** property. The width of the border adds to the overall size of the element, as it occupies space around the padding.
4. **Margin**:
   * **Definition**: Margin is the outermost space that separates the element from other elements. It creates space outside the border.
   * **Size**: Margins can be set using the **margin** property, which can also take values for all four sides. Margins do not affect the size of the element itself but do affect the distance between the element and other elements on the page.

**How Each Component Affects the Size of an Element**

The total size of an element can be calculated by adding the sizes of the content, padding, border, and margin. The formula for the total width and height of an element is as follows:

* **Total Width**:

Total Width =Content Width + Padding Width +Padding Right+ Border left + Border Right +Margin left+ Margin Right

* **Total Height**: Content Height + padding Top+ Padding Bottom+ Border Top+ Border Bottom + Margin Top+ Margin Bottom

**Question 2: What is the difference between border-boxand content-boxbox-sizing in CSS? Which is the default?**

**Ans:**

In CSS, the **box-sizing** property determines how the width and height of an element are calculated, specifically how padding and borders are included in those dimensions. There are two main values for the **box-sizing** property: **content-box** and **border-box**.

**1. content-box**

* **Definition**: This is the default value for the **box-sizing** property. When **box-sizing** is set to **content-box**, the width and height of an element are calculated based only on the content area. Padding and borders are added to the specified width and height.
* **Calculation**: Total Width=Content Width+Padding Left+Padding Right+Border Left+Border Right Total Height=Content Height+Padding Top+Padding Bottom+Border Top+Border Bottom
* **Example**:

.box {

box-sizing: content-box; /\* Default \*/

width: 200px;

padding: 20px;

border: 5px solid black;

}

In this case, the total width of the box would be $200 + 20 + 20 + 5 + 5 = 250$ pixels.

**2. border-box**

* **Definition**: When **box-sizing** is set to **border-box**, the width and height of an element include the content, padding, and border. This means that the specified width and height are the total dimensions of the element, making layout calculations easier.
* **Calculation**: Total Width=Width(includes padding and border) Total Height=Height(includes padding and border)
* **Example**:

.box {

box-sizing: border-box;

width: 200px;

padding: 20px;

border: 5px solid black;

}

In this case, the total width of the box would still be 200 pixels, as the padding and border are included within that width.

**Default Value**

* The default value for the **box-sizing** property is **content-box**. This means that if you do not specify a value for **box-sizing**, the browser will use **content-box** by default.

**Lab Assignment:**

Task: Create a profile card layout using the box model. The profile card shouldinclude:

• A profile picture.

• The user’s name and bio.

• A button to "Follow" the user. Additional Requirements:

• Add padding and borders to the elements.

• Ensure the layout is clean and centered on the page using CSS margins.

• Use the box-sizingproperty to demonstrate both content-boxand border-boxon different elements.

**Ans:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Profile Card</title>

<style>

body {

background-color: #f0f0f0;

font-family: Arial, sans-serif;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

}

.profile-card {

background-color: #fff;

width: 300px;

padding: 20px;

border: 2px solid #ccc;

border-radius: 10px;

box-shadow: 0 4px 8px rgba(0,0,0,0.1);

text-align: center;

box-sizing: border-box;

}

.profile-picture {

width: 100px;

height: 100px;

border-radius: 50%;

border: 3px solid #4CAF50;

object-fit: cover;

margin-bottom: 15px;

box-sizing: content-box;

}

.username {

font-size: 22px;

font-weight: bold;

margin-bottom: 8px;

}

.bio {

font-size: 14px;

color: #555;

margin-bottom: 20px;

}

.follow-btn {

background-color: #4CAF50;

color: white;

border: none;

padding: 10px 20px;

border-radius: 5px;

font-size: 14px;

cursor: pointer;

transition: background-color 0.3s ease;

}

.follow-btn:hover {

background-color: #45a049;

}

</style>

</head>

<body>

<div class="profile-card">

<img src="https://randomuser.me/api/portraits/men/75.jpg" alt="Indian Boy" class="profile-picture">

<div class="username">Rohan Sharma</div>

<div class="bio">Student, tech lover, and aspiring full-stack developer from Mumbai.</div>

<button class="follow-btn">Follow</button>

</div>

</body>

</html>

**CSS FlexBox:**

**Theory Assignment:**

**Question 1: What is CSS Flexbox, and how is it useful for layout design? Explain the terms flex-container and flex-item.**

**Ans:**

CSS Flexbox, or the Flexible Box Layout, is a layout model in CSS that provides a more efficient way to design complex layouts and align items within a container. It allows for responsive design by enabling items to grow, shrink, and be distributed within a container, making it easier to create flexible and adaptive layouts.

**Key Features of CSS Flexbox**

1. **Direction Control**: Flexbox allows you to define the direction of the flex items (row or column) using the **flex-direction** property.
2. **Alignment**: You can easily align items along the main axis (horizontal or vertical) and the cross axis using properties like **justify-content**, **align-items**, and **align-content**.
3. **Responsive Design**: Flexbox makes it easier to create responsive layouts that adapt to different screen sizes without the need for complex calculations or media queries.
4. **Space Distribution**: It allows for the distribution of space between items, enabling you to control how much space is allocated to each item.

**Terms**

1. **Flex Container**:
   * **Definition**: The flex container is the parent element that holds the flex items. It is defined by setting the **display** property to **flex** or **inline-flex**.
   * **Example**:

.flex-container {

display: flex; /\* or display: inline-flex; \*/

}

2. **Flex Item**:

* **Definition**: Flex items are the direct children of the flex container. These items can be manipulated using various flex properties to control their size, order, and alignment within the flex container.
* **Example**: .flex-item {

flex: 1; /\* This allows the item to grow and fill available space \*/

}

**How Flexbox is Useful for Layout Design**

* **Simplified Layouts**: Flexbox simplifies the process of creating complex layouts by providing a straightforward way to align and distribute space among items.
* **Dynamic Resizing**: Flex items can grow or shrink to fit the available space, making it easier to create responsive designs that adapt to different screen sizes.
* **Alignment Control**: Flexbox provides powerful alignment options, allowing you to center items both horizontally and vertically with minimal effort.
* **Order Control**: You can change the visual order of flex items without altering the HTML structure using the **order** property.

**Example:**

<div class="flex-container">

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

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

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

</div>

.flex-container {

display: flex;

justify-content: space-between; /\* Distributes space between items \*/

align-items: center; /\* Aligns items vertically in the center \*/

}

.flex-item {

flex: 1; /\* Each item will grow equally to fill the container \*/

margin: 10px; /\* Adds space around each item \*/

padding: 20px;

background-color: lightblue;

text-align: center;

}

**Question 2: Describe the properties justify-content, align-items, and flex-direction used in Flexbox.**

**Ans:**

In CSS Flexbox, the properties **justify-content**, **align-items**, and **flex-direction** are essential for controlling the layout and alignment of flex items within a flex container. Here’s a detailed description of each property:

**1. flex-direction**

* **Definition**: The **flex-direction** property defines the direction in which the flex items are placed in the flex container. It determines the main axis along which the items are laid out.
* **Possible Values**:
  + **row**: (default) Items are placed in a row from left to right.
  + **row-reverse**: Items are placed in a row from right to left.
  + **column**: Items are placed in a column from top to bottom.
  + **column-reverse**: Items are placed in a column from bottom to top.
* **Example**:

.flex-container {

display: flex;

flex-direction: row; /\* Items are arranged in a row \*/

}

**2. justify-content**

* **Definition**: The **justify-content** property aligns flex items along the main axis (the direction defined by **flex-direction**). It controls the distribution of space between and around the items.
* **Possible Values**:
  + **flex-start**: Items are packed toward the start of the flex container (default).
  + **flex-end**: Items are packed toward the end of the flex container.
  + **center**: Items are centered along the main axis.
  + **space-between**: Items are evenly distributed, with the first item at the start and the last item at the end.
  + **space-around**: Items are evenly distributed with equal space around them.
  + **space-evenly**: Items are evenly distributed with equal space between them.
* **Example**:

.flex-container {

display: flex;

justify-content: space-between; /\* Distributes space between items \*/

}

**3. align-items**

* **Definition**: The **align-items** property aligns flex items along the cross axis (perpendicular to the main axis). It controls how items are aligned within the flex container.
* **Possible Values**:
  + **stretch**: (default) Items are stretched to fill the container along the cross axis.
  + **flex-start**: Items are aligned at the start of the cross axis.
  + **flex-end**: Items are aligned at the end of the cross axis.
  + **center**: Items are centered along the cross axis.
  + **baseline**: Items are aligned along their baseline.
* **Example**:

.flex-container {

display: flex;

align-items: center; /\* Centers items vertically in the container \*/

}

**Lab Assignment:**

Task: Create a simple webpage layout using Flexbox. The layout should include:

• A header.

• A sidebar on the left.

• A main content area in the center.

• A footer. Additional Requirements:

• Use Flexbox to position and align the elements.

• Apply different justify-contentand align-itemsproperties to observe theireffects.

• Ensure the layout is responsive, adjusting for smaller screens.

**Ans:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Flexbox Layout</title>

<style>

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

font-family: Arial, sans-serif;

display: flex;

flex-direction: column;

min-height: 100vh;

}

header {

background-color: #4CAF50;

color: white;

text-align: center;

padding: 15px;

font-size: 24px;

}

.container {

display: flex;

flex: 1;

flex-direction: row;

justify-content: space-between;

align-items: stretch;

padding: 10px;

}

.sidebar {

background-color: #f0f0f0;

padding: 20px;

width: 200px;

}

.main-content {

background-color: #fff;

flex: 1;

padding: 20px;

margin-left: 10px;

}

footer {

background-color: #333;

color: white;

text-align: center;

padding: 10px;

}

/\* Responsive layout \*/

@media screen and (max-width: 768px) {

.container {

flex-direction: column;

justify-content: center;

align-items: stretch;

}

.sidebar {

width: 100%;

margin-bottom: 10px;

}

.main-content {

margin-left: 0;

}

}

</style>

</head>

<body>

<header>

My Flexbox Webpage

</header>

<div class="container">

<div class="sidebar">

<h3>Sidebar</h3>

<p>Links, categories, or menu items here.</p>

</div>

<div class="main-content">

<h2>Main Content</h2>

<p>This is the main content area. Flexbox makes layout design easier and more flexible.</p>

</div>

</div>

<footer>

&copy; 2025 Flex Layout. All rights reserved.

</footer>

</body>

</html>

**CSS Grid:**

**Theory Assignment:**

**Question 1: Explain CSS Grid and how it differs from Flexbox. When would you use Grid over Flexbox?**

**Ans:**

CSS Grid Layout is a powerful two-dimensional layout system in CSS that allows developers to create complex web layouts with rows and columns. It provides a way to design web pages using a grid-based approach, enabling precise control over the placement and alignment of elements within a grid.

**Key Features of CSS Grid**

1. **Two-Dimensional Layout**: Unlike Flexbox, which is primarily one-dimensional (either row or column), CSS Grid allows for both rows and columns to be defined simultaneously, making it suitable for more complex layouts.
2. **Grid Lines and Areas**: CSS Grid uses grid lines to define the structure of the grid, and developers can create grid areas to position items within the grid.
3. **Explicit and Implicit Grids**: You can define explicit grid layouts with specific rows and columns, or let the grid create implicit rows and columns as needed.
4. **Responsive Design**: CSS Grid makes it easy to create responsive layouts that adapt to different screen sizes using media queries and grid template areas.

**Differences Between CSS Grid and Flexbox:**

| **Feature** | **CSS Grid** | **Flexbox** |
| --- | --- | --- |
| Dimension | Two-dimensional (rows and columns) | One-dimensional (row or column) |
| Layout Control | More control over complex layouts | Best for simpler layouts |
| Alignment | Aligns items in both dimensions | Aligns items in a single direction |
| Use Cases | Complex layouts with overlapping items | Simple layouts, navigation bars, or single-dimensional arrangements |
| Grid Areas | Supports named grid areas | Does not support named areas |

**When to Use CSS Grid Over Flexbox**

1. **Complex Layouts**: If you need to create a complex layout with both rows and columns, such as a magazine-style layout or a dashboard, CSS Grid is the better choice.
2. **Overlapping Items**: When you want to position items in a way that they overlap or span multiple rows and columns, CSS Grid provides the necessary control.
3. **Two-Dimensional Layouts**: If your design requires precise control over both horizontal and vertical alignment, CSS Grid is more suitable.
4. **Grid Template Areas**: When you want to define specific areas of the layout using named grid areas, CSS Grid allows for a more intuitive and readable approach.

**Example Of Css Grid:**

<div class="grid-container">

<div class="grid-item item1">Item 1</div>

<div class="grid-item item2">Item 2</div>

<div class="grid-item item3">Item 3</div>

<div class="grid-item item4">Item 4</div>

</div>

.grid-container {

display: grid;

grid-template-columns: repeat(3, 1fr); /\* 3 equal columns \*/

grid-template-rows: auto; /\* Rows will adjust based on content \*/

gap: 10px; /\* Space between grid items \*/

}

.grid-item {

background-color: lightblue;

padding: 20px;

text-align: center;

}

**Question 2: Describe the grid-template-columns, grid-template-rows, and grid-gapproperties. Provide examples of how to use them.**

**Ans:**

The **grid-template-columns**, **grid-template-rows**, and **grid-gap** properties are fundamental CSS Grid properties used to define the structure and spacing of grid layouts.

**1. grid-template-columns**

* **Definition**: Specifies the number and sizes of the columns in a grid container.
* **Usage**: Defines the column track sizes explicitly.
* **Values**:
  + Length units (e.g., **px**, **em**, **%**)
  + Fractional unit (**fr**) to take a fraction of the available space
  + Keywords like **auto**, **min-content**, **max-content**
  + Repeat function **repeat(n, size)** for repeated patterns

**Example**:

.grid-container {

display: grid;

grid-template-columns: 200px 1fr 2fr;}

This creates 3 columns where:

* First column is 200px wide
* Second column takes 1 fraction of remaining space
* Third column takes 2 fractions of remaining space

**2. grid-template-rows**

* **Definition**: Specifies the number and sizes of the rows in a grid container.
* **Usage**: Defines row track sizes explicitly.
* **Values**: Same as **grid-template-columns**.

**Example**:

.grid-container {

display: grid;

grid-template-rows: 100px auto 50px;

}

This creates 3 rows where:

* First row is 100px tall
* Second row height adjusts automatically based on content
* Third row is 50px tall

**3. grid-gap (or gap)**

* **Definition**: Controls the space (gutter) between rows and columns.
* **Usage**: Sets both row and column gaps or can be split into **row-gap** and **column-gap**.
* **Values**: Length values like **px**, **em**, **%**.

**Example:**

.grid-container {

display: grid;

grid-template-columns: repeat(3, 1fr);

grid-gap: 20px;

}

This adds 20px spacing between all rows and columns.

**Combined Example:**

<div class="grid-container">

<div>Item 1</div>

<div>Item 2</div>

<div>Item 3</div>

<div>Item 4</div>

</div>

.grid-container {

display: grid;

grid-template-columns: 1fr 2fr 1fr;

grid-template-rows: 100px 200px;

grid-gap: 15px 30px; /\* 15px row-gap, 30px column-gap \*/

background-color: #f0f0f0;

}

.grid-container > div {

background-color: #4caf50;

color: white;

display: flex;

align-items: center;

justify-content: center;

font-weight: bold;

border-radius: 8px;

}

* This grid will have 3 columns where the middle column is twice as wide as the others.
* There are 2 rows with fixed heights 100px and 200px.
* Row gaps of 15px and column gaps of 30px separate the items uniformly.

**Lab Assignment:**

Task: Create a 3x3 grid of product cards using CSS Grid. Each card should contain:

• A product image.

• A product title.

• A price. Additional Requirements:

• Use grid-template-columnsto create the grid layout.

• Use grid-gapto add spacing between the grid items.

• Apply hover effects to each card for better interactivity.

**Ans:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<title>3x3 Product Grid</title>

<style>

@import url('https://fonts.googleapis.com/css2?family=Inter:wght@400;700&display=swap');

body {

margin: 0;

font-family: 'Inter', sans-serif;

background-color: #f8fafc;

padding: 40px;

display: flex;

justify-content: center;

}

.product-grid {

display: grid;

grid-template-columns: repeat(3, 1fr);

grid-gap: 24px;

max-width: 1200px;

width: 100%;

}

.product-card {

background: white;

border-radius: 16px;

box-shadow: 0 4px 12px rgb(0 0 0 / 0.1);

display: flex;

flex-direction: column;

overflow: hidden;

cursor: pointer;

transition: transform 0.3s cubic-bezier(0.4, 0, 0.2, 1),

box-shadow 0.3s cubic-bezier(0.4, 0, 0.2, 1);

}

.product-card:hover {

transform: translateY(-8px) scale(1.03);

box-shadow: 0 10px 24px rgb(0 0 0 / 0.15);

}

.product-image {

width: 100%;

height: 180px;

object-fit: cover;

border-bottom: 1px solid #e5e7eb;

background-color: #e0e7ff;

}

.product-info {

padding: 16px 20px;

flex-grow: 1;

display: flex;

flex-direction: column;

justify-content: space-between;

}

.product-title {

font-weight: 700;

font-size: 1.1rem;

color: #1e293b;

margin-bottom: 8px;

line-height: 1.3;

}

.product-price {

font-weight: 600;

font-size: 1rem;

color: #10b981; /\* Emerald green \*/

}

</style>

</head>

<body>

<div class="product-grid" role="list">

<article class="product-card" role="listitem" tabindex="0">

<img

src="https://placehold.co/400x300/2563eb/ffffff?text=Product+1"

alt="Blue wireless headphones"

class="product-image"

loading="lazy"

/>

<div class="product-info">

<h2 class="product-title">Blue Wireless Headphones</h2>

<p class="product-price">$59.99</p>

</div>

</article>

<article class="product-card" role="listitem" tabindex="0">

<img

src="https://placehold.co/400x300/f97316/ffffff?text=Product+2"

alt="Stylish orange sneaker"

class="product-image"

loading="lazy"

/>

<div class="product-info">

<h2 class="product-title">Stylish Orange Sneaker</h2>

<p class="product-price">$89.99</p>

</div>

</article>

<article class="product-card" role="listitem" tabindex="0">

<img

src="https://placehold.co/400x300/10b981/ffffff?text=Product+3"

alt="Green ergonomic chair"

class="product-image"

loading="lazy"

/>

<div class="product-info">

<h2 class="product-title">Green Ergonomic Chair</h2>

<p class="product-price">$149.99</p>

</div>

</article>

<article class="product-card" role="listitem" tabindex="0">

<img

src="https://placehold.co/400x300/8b5cf6/ffffff?text=Product+4"

alt="Violet smart smartwatch"

class="product-image"

loading="lazy"

/>

<div class="product-info">

<h2 class="product-title">Violet Smart Smartwatch</h2>

<p class="product-price">$199.00</p>

</div>

</article>

<article class="product-card" role="listitem" tabindex="0">

<img

src="https://placehold.co/400x300/f43f5e/ffffff?text=Product+5"

alt="Red coffee machine"

class="product-image"

loading="lazy"

/>

<div class="product-info">

<h2 class="product-title">Red Coffee Machine</h2>

<p class="product-price">$79.99</p>

</div>

</article>

<article class="product-card" role="listitem" tabindex="0">

<img

src="https://placehold.co/400x300/fbbf24/ffffff?text=Product+6"

alt="Yellow vintage camera"

class="product-image"

loading="lazy"

/>

<div class="product-info">

<h2 class="product-title">Yellow Vintage Camera</h2>

<p class="product-price">$249.50</p>

</div>

</article>

<article class="product-card" role="listitem" tabindex="0">

<img

src="https://placehold.co/400x300/3b82f6/ffffff?text=Product+7"

alt="Blue gaming mouse"

class="product-image"

loading="lazy"

/>

<div class="product-info">

<h2 class="product-title">Blue Gaming Mouse</h2>

<p class="product-price">$39.99</p>

</div>

</article>

<article class="product-card" role="listitem" tabindex="0">

<img

src="https://placehold.co/400x300/14b8a6/ffffff?text=Product+8"

alt="Turquoise wireless speaker"

class="product-image"

loading="lazy"

/>

<div class="product-info">

<h2 class="product-title">Turquoise Wireless Speaker</h2>

<p class="product-price">$89.95</p>

</div>

</article>

<article class="product-card" role="listitem" tabindex="0">

<img

src="https://placehold.co/400x300/64748b/ffffff?text=Product+9"

alt="Gray office desk lamp"

class="product-image"

loading="lazy"

/>

<div class="product-info">

<h2 class="product-title">Gray Office Desk Lamp</h2>

<p class="product-price">$45.00</p>

</div>

</article>

</div>

</body>

</html>

**Responsive Web Design with Media Queries:**

**Theory Assignment:**

**Question 1: What are media queries in CSS, and why are they important for responsive design?**

**Ans:**

Media queries in CSS are a powerful feature that allows developers to apply different styles to a web page based on the characteristics of the device or viewport displaying the content. They enable responsive design by allowing the layout and appearance of a website to adapt to various screen sizes, resolutions, and orientations.

**Definition of Media Queries**

A media query consists of a media type (such as **screen**, **print**, etc.) and one or more expressions that check for specific conditions, such as viewport width, height, resolution, and orientation. When the conditions of a media query are met, the styles defined within that media query are applied.

**Syntax of Media Queries**

The basic syntax of a media query is as follows:

@media media-type and (condition) {

/\* CSS rules here \*/

}

**Example:**

@media screen and (max-width: 600px) {

body {

background-color: lightblue;

}

}

In this example, the background color of the body will change to light blue when the viewport width is 600 pixels or less.

**Importance of Media Queries for Responsive Design**

1. **Adaptability**: Media queries allow web pages to adapt to different screen sizes and resolutions, ensuring that content is accessible and visually appealing on a wide range of devices, from mobile phones to large desktop monitors.
2. **Improved User Experience**: By tailoring the layout and design to fit the user's device, media queries enhance the overall user experience. This includes adjusting font sizes, images, navigation menus, and other elements to ensure they are easy to read and interact with.
3. **Performance Optimization**: Media queries can help optimize performance by loading only the necessary styles for a specific device. This can reduce the amount of CSS that needs to be processed, leading to faster load times.
4. **Separation of Concerns**: Media queries promote a clean separation of styles for different devices, making it easier to maintain and update the CSS code. Developers can organize styles based on device characteristics rather than creating separate stylesheets for each device.
5. **Future-Proofing**: As new devices with varying screen sizes and resolutions continue to emerge, media queries provide a flexible solution for ensuring that websites remain responsive and functional across all platforms.

**Question 2: Write a basic media query that adjusts the font size of a webpage for screens smaller than 600px**

**Ans:**

A basic media query that adjusts the font size of a webpage for screens smaller than 600px would look like this:

@media screen and (max-width: 600px) {

body {

font-size: 14px;

}

}

This CSS rule means that when the viewport width is 600 pixels or less, the font size of the entire page’s body text will be set to 14 pixels, providing better readability on smaller screens.

**LAB ASSIGNMENT:**

Task: Build a responsive webpage that includes:

• A navigation bar.

• A content section with two columns.

A footer. Additional Requirements:

• Use media queries to make the webpage responsive for mobile devices.

• On smaller screens (below 768px), stack the columns vertically.

• Adjust the font sizes and padding to improve readability on mobile

**Ans:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Responsive Webpage</title>

<style>

\* {

box-sizing: border-box;

margin: 0;

padding: 0;

}

body {

font-family: Arial, sans-serif;

line-height: 1.6;

}

/\* Navigation Bar \*/

nav {

background-color: #333;

color: #fff;

padding: 15px 20px;

text-align: center;

font-size: 20px;

}

/\* Content Section \*/

.content {

display: flex;

padding: 20px;

gap: 20px;

}

.column {

flex: 1;

padding: 20px;

background-color: #f4f4f4;

border-radius: 8px;

}

/\* Footer \*/

footer {

background-color: #222;

color: #fff;

text-align: center;

padding: 15px 10px;

font-size: 16px;

}

/\* Responsive Design \*/

@media (max-width: 768px) {

.content {

flex-direction: column;

}

nav {

font-size: 18px;

padding: 12px 10px;

}

.column {

padding: 15px;

font-size: 16px;

}

footer {

font-size: 14px;

padding: 10px;

}

}

</style>

</head>

<body>

<nav>

My Responsive Website

</nav>

<div class="content">

<div class="column">

<h2>Column 1</h2>

<p>This is the first column of content. It contains text that will stack on mobile devices.</p>

</div>

<div class="column">

<h2>Column 2</h2>

<p>This is the second column of content. Both columns are responsive using Flexbox and media queries.</p>

</div>

</div>

<footer>

© 2025 My Responsive Webpage. All rights reserved.

</footer>

</body>

</html>