**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.

* A **CSS selector** is a pattern used to select and style HTML elements. It tells the browser which HTML element(s) the CSS rules should apply to.
* In **CSS**, there are **five main types** of selectors, each used to target HTML elements in different ways:
* Basic Selectors
* Combinator Selectors
* Pseudo-class Selectors
* Pseudo-element Selectors
* Attribute Selectors
* Grouping Selector
* The basic selectors are categorized in following types with example :
* **Element selector :**
* Selects all elements of a specific HTML tag.
* It is also known as Type selector/Tag name selector.
* Example:
  + p{

color:red;

}

* **Class Selector :**
* Selects elements with a specific class attribute. Preceded by a period (.).
* Example :
  + .btn {

background: blue;

}

* **ID Selector:**
* Selects a *single* element with a specific id attribute. Preceded by a hash sign (#).
* ID should be unique within an HTML document.
* Example :
  + #main {

padding: 20px;

}

Question 2: Explain the concept of CSS specificity. How do conflicts

between multiple stylesget resolved?

* CSS specificity is a set of rules that the browser uses to determine which CSS rule is applied to an element when multiple rules could apply.
* It helps resolve conflicts between different CSS rules targeting the same element.
  + **Specificity Hierarchy (from lowest to highest):**

| **Selector Type** | **Specificity Value** |
| --- | --- |
| Universal selector \* | 0-0-0-0 |
| Element selectors div, p | 0-0-0-1 |
| Class selectors .class | 0-0-1-0 |
| Attribute selectors [type] | 0-0-1-0 |
| Pseudo-classes :hover | 0-0-1-0 |
| ID selectors #id | 0-1-0-0 |
| Inline styles | 1-0-0-0 |
| !important | Overrides all  (but avoid!) |

* Example:

/\* Specificity: 0-0-0-1 \*/

p {

color: blue;

}

/\* Specificity: 0-0-1-0 \*/

.text {

color: green;

}

/\* Specificity: 0-1-0-0 \*/

#intro {

color: red;

}

/\* Specificity: 1-0-0-0 \*/

<p id="intro" class="text" style="color: orange;">Hello</p>

* The final color applied will be **orange**, because inline styles have the highest specificity.

Question 3: What is the difference between internal, external, and inline

CSS? Discuss the advantages and disadvantages of each

approach.

* CSS can be added to HTML documents in three main ways:
  + 1.Inline css
  + 2.Internal css
  + 3.External css
* **Inline css:**
* CSS is written directly inside the HTML element using the “style” attribute.
* Example:
  + <p style="color: blue; font-size: 16px;">

This is a paragraph with inline styles.

</p>

* ***Advantages:***
* Quick for small changes.
* Useful for testing/debugging.
* Overrides other styles (high specificity).

* ***Disadvantages:***
* Not reusable.
* Makes HTML messy and hard to maintain.
* Poor separation of content and style.
* **Internal css:**
* Internal CSS is defined within the <style> tags in the <head> section of an HTML document. These styles apply only to the specific HTML page they are embedded in.
* Example:

<!DOCTYPE html>

<html>

<head>

<title>Internal CSS Example</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f0f0f0;

}

h1 {

color: green;

text-align: center;

}

p {

line-height: 1.5;

}

</style>

</head>

<body>

<h1>Welcome to My Page</h1>

<p>This paragraph has internal styles applied to it.</p>

<p>So does this one!</p>

</body>

</html>

* ***Advantages:***
* Styles apply to the whole page.
* Easier to manage than inline CSS.
* Good for single-page applications.
* ***Disadvantages:***
* Not reusable across multiple pages.
* Increases page load time if styles are large.
* Still mixes HTML and CSS (less modular).
* **External css:**
* External CSS involves writing CSS rules in a separate .css file and linking it to the HTML document using the <link> tag in the <head> section.
* This is the most common and recommended method for styling websites.
* Example:
  + styles.css (separate file):

body {

font-family: Verdana, sans-serif;

margin: 20px;

}

h1 {

color: purple;

border-bottom: 2px solid purple;

}

.container {

width: 80%;

margin: 0 auto;

padding: 15px;

background-color: #ffffff;

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

}

* + index.html:

<!DOCTYPE html>

<html>

<head>

<title>External CSS Example</title>

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

</head>

<body>

<div class="container">

<h1>My Website Title</h1>

<p>This content is styled by an external stylesheet.</p>

</div>

</body>

</html>

* ***Advantages:***
* Fully separates content (HTML) and style (CSS).
* Reusable across multiple pages.
* Easier to maintain, scalable, and reduces page clutter.
* Browser caching improves load times.
* ***Disadvantages:***
* Extra HTTP request (may affect first load time)
* No styling if the CSS file fails to load.

**(***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.

* **Index.html:**-

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Contact Us</title>

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

</head>

<body>

<div class="container">

<form action="#" method="POST" id="contactForm">

<h2>Contact Us</h2>

<div class="form-group">

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

<input type="text" id="name" name="user\_name" required>

</div>

<div class="form-group">

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

<input type="email" id="email" name="user\_email" required>

</div>

<div class="form-group">

<label for="phone">Phone (Optional):</label>

<input type="tel" id="phone" name="user\_phone">

</div>

<div class="form-group">

<label for="subject">Subject:</label>

<input type="text" id="subject" name="user\_subject" required>

</div>

<div class="form-group">

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

<textarea id="message" name="user\_message" rows="5" required></textarea>

</div>

<div class="form-group">

<label>Preferred Contact Method:</label>

<input type="radio" id="contactEmail" name="contact\_method" value="email" checked>

<label for="contactEmail" class="radio-label">Email</label>

<input type="radio" id="contactPhone" name="contact\_method" value="phone">

<label for="contactPhone" class="radio-label">Phone</label>

</div>

<button type="submit" id="submitButton">Send Message</button>

</form>

</div>

</body>

</html>

* **style.css:-**

body {

font-family: Arial, sans-serif;

background-color: #f4f7f6;

display: flex;

justify-content: center;

align-items: center;

min-height: 100vh;

margin: 0;

padding: 20px;

box-sizing: border-box;

}

.container {

max-width: 600px;

width: 100%;

background-color: #ffffff;

padding: 40px;

border-radius: 8px;

box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);

}

/\* Styling for the form itself - using an ID selector for unique element \*/

#contactForm {

background-color: #e0f2f7;

padding: 30px;

border-radius: 10px;

box-shadow: inset 0 2px 5px rgba(0, 0, 0, 0.05);

}

#contactForm h2 {

text-align: center;

color: #333;

margin-bottom: 30px;

font-size: 2em;

border-bottom: 2px solid #a7d9ed;

padding-bottom: 10px;

}

/\* Styling for common form groups (divs holding label and input) \*/

.form-group {

margin-bottom: 20px;

}

label {

display: block;

margin-bottom: 8px;

font-weight: bold;

color: #555;

}

input[type="text"],

input[type="email"],

input[type="tel"],

textarea {

width: calc(100% - 20px);

padding: 12px;

margin: 0;

border: 1px solid #ccc;

border-radius: 5px;

box-sizing: border-box;

font-size: 1em;

transition: border-color 0.3s ease;

}

input[type="text"]:focus,

input[type="email"]:focus,

input[type="tel"]:focus,

textarea:focus {

border-color: #007bff;

outline: none;

box-shadow: 0 0 5px rgba(0, 123, 255, 0.2);

}

/\* Specific styling for the textarea \*/

textarea {

resize: vertical;

min-height: 100px;

}

/\* Styling for radio button labels (common class) \*/

.radio-label{

display: inline-block;

margin-right: 15px;

font-weight: normal;

cursor: pointer;

}

#submitButton {

display: block;

width: 100%;

padding: 15px;

background-color: #28a745;

color: white;

border: none;

border-radius: 5px;

font-size: 1.1em;

cursor: pointer;

transition: background-color 0.3s ease, transform 0.2s ease;

}

#submitButton:hover {

background-color: whitesmoke;

color:black;

transform: translateY(-2px);

box-shadow: 0 4px 10px rgba(0, 0, 0, 0.2);

}

**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?

* The CSS Box Model is a fundamental concept in web development that describes how HTML elements are rendered on a web page. Every HTML element can be thought of as a rectangular box, and the box model defines how the different parts of this box (content, padding, border, and margin) are structured and how they contribute to the element's total size.
* Understanding the box model is crucial for controlling the layout, spacing, and appearance of elements on your website.
* Here are the components of the CSS Box Model, from innermost to outermost:

**Content Box:**

* **Description:** This is the innermost part of the box, where the actual content of the HTML element resides. For text, it's where the text flows. For an image, it's the image itself.
* **Properties:** Its size is determined by the width and height CSS properties.
* **Effect on Size:** The width and height properties directly control the dimensions of the content area.

**Padding Box:**

* **Description:** Padding is the transparent space *between* the content and the border of an element. It acts as a buffer zone, creating internal spacing within the box.
* **Properties:**
  + padding-top, padding-right, padding-bottom, padding-left: For individual sides.
  + padding: Shorthand for all four sides (e.g., padding: 10px; for all sides, padding: 10px 20px; for top/bottom and left/right, etc.).
* **Effect on Size:** Padding *adds* to the total size of the element. If an element has a width of 100px and padding-left: 10px; and padding-right: 10px;, its total width (including padding) will be 120px. The padding pushes the border outwards from the content.

**Border Box:**

* **Description:** The border is a line that surrounds the padding and content. It's visible and can have various styles, widths, and colors.
* **Properties:**
  + border-width: Thickness of the border.
  + border-style: Type of border (e.g., solid, dashed, dotted, none).
  + border-color: Color of the border.
  + border: Shorthand property (e.g., border: 1px solid black;).
  + Individual side properties: border-top, border-right, border-bottom, border-left (and their respective width, style, color sub-properties).
* **Effect on Size:** The border *adds* to the total size of the element. If an element has width: 100px, padding: 10px, and border: 2px solid black;, its total width will be 100px (content) + 2\*10px (padding) + 2\*2px (border) = 124px. The border pushes the margin outwards from the padding.

**Margin Box:**

* **Description:** Margin is the transparent space *outside* the border of an element. It creates space between the element and other adjacent elements. Unlike padding and borders, margins do not have a background color.
* **Properties:**
  + margin-top, margin-right, margin-bottom, margin-left: For individual sides.
  + margin: Shorthand for all four sides (similar to padding).
* **Effect on Size:** Margin *does not affect the actual size* of the element itself (its content, padding, or border). Instead, it affects the total space that the element occupies on the page, pushing other elements away from it. If an element has margin: 20px;, it will create a 20px empty space around its border, pushing other elements away by that amount.
* **Margin Collapse:** A unique behavior of vertical margins is called "margin collapse." When two vertical margins meet (e.g., the bottom margin of one paragraph and the top margin of the next), they collapse into a single margin equal to the larger of the two, rather than summing up. Horizontal margins do not collapse.
* **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-box and content- box

box-sizing inCSS? Which is the default?

* The box-sizing property in CSS determines **how the total width and height** of an element are calculated.
* There are two types:

### **1. content-box (Default):**

* **Only the content** size is defined by width and height.
* **Padding and border are added outside** the content box.
  + .box {

width: 200px;

padding: 20px;

border: 5px solid black;

box-sizing: content-box;

}

### **2. border-box**

* width and height include **content + padding + border**.
* Makes it easier to manage layouts.

.box {

width: 200px;

padding: 20px;

border: 5px solid black;

box-sizing: border-box;

}

* **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

should include:

• 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-sizing property to demonstrate both content-box and

border-box on different elements.

**code:-**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Profile Card</title>

<style>

\* {

box-sizing: border-box;

}

body {

margin: 0;

padding: 0;

font-family: Arial, sans-serif;

background-color: #f4f4f4;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

}

.profile-card {

width: 300px;

background-color: #fff;

border: 2px solid #ddd;

padding: 20px;

border-radius: 10px;

text-align: center;

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

margin: 20px;

}

.profile-img {

width: 100px;

height: 100px;

border-radius: 50%;

border: 4px solid #4caf50;

margin-bottom: 15px;

object-fit: cover;

box-sizing: content-box;

}

.profile-name {

font-size: 1.5em;

margin: 10px 0;

box-sizing: border-box;

}

.profile-bio {

font-size: 0.95em;

color: #555;

padding: 10px;

border: 1px dashed #ccc;

margin-bottom: 20px;

box-sizing: content-box;

}

.follow-btn {

padding: 10px 20px;

border: none;

background-color: #4caf50;

color: white;

font-size: 1em;

border-radius: 5px;

cursor: pointer;

box-sizing: border-box;

transition: background-color 0.3s ease;

}

.follow-btn:hover {

background-color: gray;

color:black;

transform: scale(1.2,1.2);

transition: all 2s ease;

}

</style>

</head>

<body>

<div class="profile-card">

<img

src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSiJzlJkfB\_k\_pnoRe\_pPS\_U8jYfMzSHCo2-w&s" alt="Profile Picture" class="profile-img">

<h2 class="profile-name">John Doe</h2>

<p class="profile-bio">Web Developer | Tech Enthusiast | Blogger</p>

<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.

* CSS **Flexbox** (short for *Flexible Box Layout*) is a layout module in CSS that provides a more efficient and responsive way to arrange and align elements within a container. It helps distribute space dynamically and align content even when the size of the elements is unknown or dynamic.
* Flexbox is particularly useful for creating **one-dimensional layouts**, meaning it lays out items in either a **row (horizontal)** or a **column (vertical)**.
* 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:
  + **Flex Container**
    - The **flex container** is the parent element where display: flex (or display: inline-flex) is applied.
    - This element becomes the flex context, meaning its child elements will follow the Flexbox rules.
    - Ex:
      * **Css** :

.container {

display: flex;

}

* + - * **html:**

<div class="container">

<div>Item 1</div>

<div>Item 2</div>

</div>

* + **Flex Items**
    - The flex items are the direct children of the flex container. These items are laid out and controlled using Flexbox properties like:
      * Flex-grow
      * Flex-shrink
      * Flex-basis
      * align-self
      * Order
    - Ex:
      * **Css:**

.container div {

flex: 1;

}

### **Why is Flexbox Useful?**

* Makes layout building easier and cleaner.
* Supports dynamic resizing and flexible item alignment.
* Works well for responsive designs.
* Simplifies vertical and horizontal centering.
* Reduces the need for floating elements or complex positioning.

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

* These three properties are core to how Flexbox controls the **alignment and direction** of flex items within a flex container.
  + **Flex-direction**
    - **Defines the main axis** of the flex container, i.e., whether items are laid out in rows or columns.

**Values:**

* row (default): Items placed **horizontally** from left to right.
* row-reverse: Items placed **horizontally** from right to left.
* column: Items placed **vertically** from top to bottom.
* column-reverse: Items placed **vertically** from bottom to top.

**Example:**

.container {

display: flex;

flex-direction: row;

}

* + **Justify-content**
    - **Aligns flex items along the main axis** (horizontal if flex-direction: row; vertical if column).

**Values:**

* flex-start (default): Items start at the beginning of the main axis.
* flex-end: Items at the end of the main axis.
* center: Items centered along the main axis.
* space-between: Equal space **between** items.
* space-around: Equal space **around** items.
* space-evenly: Equal space **between and around** items.

**Example:**

.container {

display: flex;

justify-content: space-between;

}

* + **align-items**
    - **Aligns flex items along the cross axis** (perpendicular to the main axis).

**Values:**

* stretch (default): Items stretch to fill the container.
* flex-start: Items align at the start of the cross axis.
* flex-end: Items align at the end of the cross axis.
* center: Items centered along the cross axis.
* baseline: Items align based on text baseline.

**Example:**

.container {

display: flex;

align-items: center;

}

**(***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 their

effects.

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

Code:-

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Flexbox Layout Example</title>

<style>

\* {

box-sizing: border-box;

margin: 0;

padding: 0;

}

body {

font-family: Arial, sans-serif;

min-height: 100vh;

display: flex;

flex-direction: column;

}

header {

background-color: #4CAF50;

color: white;

padding: 1rem;

text-align: center;

}

.content {

flex: 1;

display: flex;

flex-direction: row;

justify-content: space-between;

align-items: stretch;

padding: 1rem;

gap: 1rem;

}

aside {

background-color: #f4f4f4;

padding: 1rem;

width: 25%;

min-width: 200px;

}

main {

background-color: #eaeaea;

padding: 1rem;

flex: 1;

}

footer {

background-color: #333;

color: white;

text-align: center;

padding: 1rem;

}

/\* Responsive design \*/

@media (max-width: 768px) {

.content {

flex-direction: column;

justify-content: center;

align-items: stretch;

}

aside {

width: 100%;

}

}

</style>

</head>

<body>

<header>

<h1>My Flexbox Webpage</h1>

</header>

<div class="content">

<aside>

<h2>Sidebar</h2>

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

</aside>

<main>

<h2>Main Content</h2>

<p>This is the main area of the page. Resize the browser to see Flexbox in action!</p>

</main>

</div>

<footer>

<p>&copy; 2025 My Website</p>

</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?

* **CSS Grid Layout** is a powerful 2-dimensional layout system in CSS used to design web pages by dividing the page into **rows and columns**. Unlike Flexbox (which works in a single dimension at a time), CSS Grid allows developers to control both the **horizontal (row)** and **vertical (column)** layout at the same time.
* **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 Grid Over Flexbox**
* **Use CSS Grid when:**

You are designing full page layouts or complex components with both rows and columns.

You want explicit control over the placement of items in a matrix.

Your layout requires overlapping elements, named grid areas, or asymmetric positioning.

* **Use Flexbox when:**

You need to align items in a single direction (like navbars, toolbars, or cards in a row).

You're building small UI components that don’t require a grid structure.  
 Your layout is content-driven (e.g., wrapping flexible items).

* **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-gap properties. Provide examples of how to use them.

* These properties are part of the **CSS Grid Layout** and are used to define the structure and spacing of the grid.

**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-gap to add spacing between the grid items.

• Apply hover effects to each card for better interactivity.

code:-

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

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

<title>Product Grid</title>

<style>

\* {

box-sizing: border-box;

margin: 0;

padding: 0;

}

body {

font-family: Arial, sans-serif;

padding: 20px;

background-color: #f5f5f5;

}

h1 {

text-align: center;

margin-bottom: 20px;

}

.grid-container {

display: grid;

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

gap: 20px; /\* space between cards \*/

}

.card {

background-color: white;

border-radius: 10px;

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

overflow: hidden;

transition: transform 0.3s ease, box-shadow 0.3s ease;

}

.card:hover {

transform: translateY(-5px);

box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);

}

.card img {

width: 100%;

height: 200px;

object-fit: cover;

}

.card-content {

padding: 15px;

text-align: center;

}

.card-content h3 {

font-size: 18px;

margin-bottom: 10px;

}

.card-content p {

font-size: 16px;

color: #333;

}

/\* Responsive for small screens \*/

@media (max-width: 768px) {

.grid-container {

grid-template-columns: repeat(2, 1fr);

}

}

@media (max-width: 500px) {

.grid-container {

grid-template-columns: 1fr;

}

}

</style>

</head>

<body>

<h1>Product Grid</h1>

<div class="grid-container">

<div class="card">

<img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQjtLebLKquRUXHlAKN6uqWKEzTqWWIsbPSHA&s" alt="Product 1">

<div class="card-content">

<h3>Product 1</h3>

<p>$19.99</p>

</div>

</div>

<div class="card">

<img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTqKbTokZGS0tVJFDguMJUwM9z78PviZXHVzg&s" alt="Product 2">

<div class="card-content">

<h3>Product 2</h3>

<p>$24.99</p>

</div>

</div>

<div class="card">

<img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSuGGcvjCINuayhaNI1cKue2OKPSxc1AqL1QA&s" alt="Product 3">

<div class="card-content">

<h3>Product 3</h3>

<p>$29.99</p>

</div>

</div>

<div class="card">

<img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTQZOfp4HQVilh6MSYS3d7ilwCVKB2PHV7c9Q&s" alt="Product 4">

<div class="card-content">

<h3>Product 4</h3>

<p>$15.99</p>

</div>

</div>

<div class="card">

<img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRQPtNDmUedy5jVCYBNrVe5Xnc8s8kCsBklew&s" alt="Product 5">

<div class="card-content">

<h3>Product 5</h3>

<p>$9.99</p>

</div>

</div>

<div class="card">

<img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcT9R9ssf51Sfg5JS\_mM\_G0CBsQ3sWfWImuuAg&s" alt="Product 6">

<div class="card-content">

<h3>Product 6</h3>

<p>$39.99</p>

</div>

</div>

<div class="card">

<img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQ48u0CzL7Zu737DQisB8PFAreKuMMhsHl8gA&s" alt="Product 7">

<div class="card-content">

<h3>Product 7</h3>

<p>$49.99</p>

</div>

</div>

<div class="card">

<img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcS0\_vvwQpwA9rWXFbbau5WJTJCk4SypR8xeHA&s" alt="Product 8">

<div class="card-content">

<h3>Product 8</h3>

<p>$59.99</p>

</div>

</div>

<div class="card">

<img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSj0D3\_szWBkKaEQmToVBN8mg7BGstXNHBirg&s" alt="Product 9">

<div class="card-content">

<h3>Product 9</h3>

<p>$12.99</p>

</div>

</div>

</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?

* Media queries are a feature of CSS used to apply different styles to different devices or screen sizes.
* They help create responsive web designs by allowing you to change the layout, font sizes, colors, or visibility of elements based on the device's characteristics such as:
* Width
* Height
* Resolution
* Orientation
* Device type
* Syntax:

@media (condition) {

/\* CSS rules here \*/

}

* Example:

@media (max-width: 768px) {

body {

background-color: lightgray;

}

}

This rule applies only when the screen width is 768px or smaller.

### **Why are Media Queries Important for Responsive Design?**

### Media queries are essential to make websites look and work well across a wide range of devices:

| **Importance** | **Explanation** |
| --- | --- |
| Device adaptability | Adjust layout for mobiles, tablets, desktops, etc. |
| Better user experience | Ensures content is readable and accessible on all screens |
| Avoids horizontal scrolling | Helps rearrange elements to fit smaller screens |
| Performance and accessibility | Allows showing/hiding content as needed |
| Supports mobile-first design | Build for small screens first, then scale up |

* **Common Media Query Use Cases:**

/\* Mobile \*/

@media (max-width: 600px) {

.menu {

display: none;

}

}

/\* Tablets \*/

@media (min-width: 601px) and (max-width: 1024px) {

.container {

padding: 20px;

}

}

/\* Desktops \*/

@media (min-width: 1025px) {

.container {

max-width: 1200px;

}

}

Question 2: Write a basic media query that adjusts the font size of a

webpage for screens smaller than 600px.

* 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.

code:-

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

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

<title>Responsive Webpage</title>

<style>

\* {

box-sizing: border-box;

margin: 0;

padding: 0;

}

body {

font-family: Arial, sans-serif;

line-height: 1.6;

background-color: #f5f5f5;

}

/\* Navigation Bar \*/

nav {

background-color: #333;

color: white;

padding: 1rem;

text-align: center;

}

nav h1 {

font-size: 24px;

}

/\* Main Content \*/

.container {

display: flex;

padding: 20px;

gap: 20px;

}

.left-column,

.right-column {

flex: 1;

background-color: white;

padding: 20px;

border-radius: 8px;

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

}

/\* Footer \*/

footer {

background-color: #333;

color: white;

text-align: center;

padding: 1rem;

margin-top: 20px;

}

/\* Responsive Design \*/

@media (max-width: 768px) {

.container {

flex-direction: column;

padding: 10px;

}

.left-column,

.right-column {

padding: 15px;

}

nav h1 {

font-size: 20px;

}

.left-column,

.right-column,

footer {

font-size: 16px;

}

}

@media (max-width: 480px) {

nav h1 {

font-size: 18px;

}

.left-column,

.right-column,

footer {

font-size: 14px;

padding: 12px;

}

}

</style>

</head>

<body>

<!-- Navigation -->

<nav>

<h1>My Responsive Site</h1>

</nav>

<!-- Content Section -->

<div class="container">

<div class="left-column">

<h2>Left Column</h2>

<p>

This is the left column. It contains text or other content.

On smaller screens, this column will stack vertically above the right column.

</p>

</div>

<div class="right-column">

<h2>Right Column</h2>

<p>

This is the right column. It complements the content on the left.

Resize the browser to see how the layout adjusts!

</p>

</div>

</div>

<!-- Footer -->

<footer>

<p>&copy; 2025 My Website. All rights reserved.</p>

</footer>

</body>

</html>