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

* **CSS Selectors & Styling**

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

**✨ Types of CSS Selectors with Examples:**

1. 🔵 **Element Selector (tag name)**  
   ➤ Aa selector tag name na adhar par element ne select kare chhe.  
   **Example:**

{

color: red;

}

🔸 Aa code badha <h1> elements no color red banave chhe.

1. 🟢 **Class Selector (.classname)**  
   ➤ Aa selector class name na adhar par element ne select kare chhe. Dot (.) thi sharu thay chhe.  
   **Example:**

.highlight {

background-color: yellow;

}

🔸 Aa code badha elements ne je class "highlight" chhe, ema yellow background aape chhe.

1. 🔴 **ID Selector (#idname)**  
   ➤ Aa selector ek khas ID vala element ne target kare chhe. Hash (#) thi sharu thay chhe.  
   **Example:**

#main {

font-size: 20px;

}

🔸 Aa rule id="main" vala element no font size 20px kare chhe.

🔚 **Note:**

* .classname → Class Selector (multiple elements)
* #idname → ID Selector (unique element)
* tagname → Element Selector (all matching tags)

**2) Explain the concept of CSS specificity. How do conflicts between multiple styles get resolved?**

**CSS specificity** is a set of rules that determines **which style rule is applied** to an element when multiple rules could apply. The browser uses specificity to **resolve conflicts** between different CSS rules.

**🧠 How Specificity Works:**

Each selector has a **specificity value**. The higher the value, the more importance it has. The browser compares the specificity to decide which rule wins.

Specificity is calculated based on this format:  
**inline styles > ID selectors > class selectors > element selectors**

**🔢 Specificity Calculation (from highest to lowest):**

| **Selector Type** | **Specificity Points** |
| --- | --- |
| Inline styles | 1000 |
| ID selectors (#id) | 100 |
| Class/attribute/pseudo-class (.class, [type], :hover) | 10 |
| Element and pseudo-elements (div, p, ::before) | 1 |

**🔍 Example:**

<p id="title" class="blue">Hello World</p>

p { color: red; } /\* specificity = 1 \*/

.blue { color: green; } /\* specificity = 10 \*/

#title { color: blue; } /\* specificity = 100 \*/

✅ In this case, #title { color: blue; } wins because it has the highest specificity.

**🔄 Conflict Resolution:**

* If multiple rules apply to the same element, the rule with **higher specificity** is applied.
* If specificity is the same, the rule that comes **later in the CSS** wins.
* !important overrides everything but should be avoided for cleaner code.

🔚 **Conclusion:**  
CSS specificity helps decide **which style is stronger** when multiple rules target the same element. Understanding it is key to writing predictable and conflict-free CSS.

1. **What is the difference between internal, external, and inline CSS? Discuss the advantages and disadvantages of each approach.**

CSS can be applied in three main ways: **inline**, **internal**, and **external**. Each has its own use case, benefits, and limitations.

**1️⃣ Inline CSS**

* CSS is written directly inside the HTML element using the style attribute.  
  **Example:**

<p style="color: red;">Hello</p>

**Advantages:**  
✔ Easy to apply quickly to a single element  
✔ Useful for testing or small changes

**Disadvantages:**  
✘ Not reusable  
✘ Difficult to maintain for large websites  
✘ Breaks separation of content and style

**2️⃣ Internal CSS**

* CSS is written inside a <style> tag within the <head> section of an HTML file.  
  **Example:**

<head>

<style>

p { color: blue; }

</style>

</head>

**Advantages:**  
✔ Good for single-page styling  
✔ Styles are all in one place in the file

**Disadvantages:**  
✘ Not reusable across multiple pages  
✘ Increases page size if styles are large

**3️⃣ External CSS**

* CSS is written in a separate .css file and linked using the <link> tag.  
  **Example:**

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

**Advantages:**  
✔ Reusable across multiple pages  
✔ Clean and easy to maintain  
✔ Faster page load due to caching

**Disadvantages:**  
✘ Requires internet or path access to the CSS file  
✘ Extra HTTP request for loading the file

🔚 **Conclusion:**  
Each CSS method has a specific use.

* Use **inline** for very small, quick changes.
* Use **internal** for single-page projects.
* Use **external** for larger websites with multiple pages and reusable styles.

**• CSS Box Model**

**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 CSS that describes how every HTML element is structured and spaced on a web page.

It consists of **four main components**, from the inside out:

**1️⃣ Content**

* This is the actual content of the element (like text or image).
* It takes up space based on width and height values.

📌 **Effect on size:** This is the base size of the element.

**2️⃣ Padding**

* The space between the content and the border.
* It adds space **inside** the element, around the content.

📌 **Effect on size:** Increases the total size of the element by expanding inward space.

**3️⃣ Border**

* A line that surrounds the padding and content.
* You can style its width, color, and type.

📌 **Effect on size:** Adds to the total size of the element.

**4️⃣ Margin**

* The space **outside** the border, separating the element from others.
* It is transparent and used to create distance between elements.

📌 **Effect on size:** Does **not** affect the size of the element itself, but creates space around it.

**📐 Total Size Formula:**

By default (with box-sizing: content-box):  
**Total Width = content + padding + border**  
**Total Height = content + padding + border**

Margins are not included in total element size but affect spacing between elements.

**🔚 Conclusion:**  
The box model controls how an element’s size and spacing are calculated.  
Understanding it is essential for layout, spacing, and responsive design.

**2) What is the difference between border-box and content-box box-sizing in CSS? Which is the default?**

**1️⃣ content-box (Default value)**

* Only the **content** is counted in the width and height.
* **Padding and border** are **added outside** the specified size.

**Example:**

div {

box-sizing: content-box;

width: 200px;

padding: 20px;

border: 5px solid black;

}

🔸 **Total width = 200 + 20*2 + 5*2 = 250px**  
🔸 **Total height increases the same way**

**2️⃣ border-box**

* The **content, padding, and border** are all included **within** the specified width and height.
* The size stays fixed, and padding/border reduce content space.

**Example:**

div {

box-sizing: border-box;

width: 200px;

padding: 20px;

border: 5px solid black;

}

🔸 **Total width = 200px (no extra size)**  
🔸 Content area shrinks to fit padding and border inside.

**✅ Default value:**

content-box is the default box-sizing in CSS.

**🔚 Conclusion:**

* Use content-box when you want the width/height to apply to **content only**.
* Use border-box when you want the width/height to include **content + padding + border**, making layout easier to manage.

**• CSS Flexbox**

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

**CSS Flexbox** (Flexible Box Layout) is a one-dimensional layout model used in CSS. It helps in designing **responsive and flexible layouts** by distributing space efficiently among items, even when their size is unknown or dynamic.

**🌟 Why Flexbox is Useful:**

* Automatically adjusts items to different screen sizes
* Aligns items both horizontally and vertically
* Helps create equal spacing between elements
* Makes layout design easier with fewer media queries

**🔲 Flex Container**

* The parent element that holds flex items.
* You define it using display: flex or display: inline-flex.

**Example:**

.container {

display: flex;

}

📌 This makes .container a **flex container**, enabling all its children to behave as **flex items**.

**🔳 Flex Item**

* The **direct child** elements of a flex container.
* These items are laid out using flex properties like flex-grow, flex-shrink, and flex-basis.

**Example:**

<div class="container">

<div class="box">Item 1</div>

<div class="box">Item 2</div>

</div>

Here, each .box is a **flex item** inside the flex container .container.

**🔚 Conclusion:**  
Flexbox is a powerful CSS tool for creating responsive layouts with easy alignment and spacing.

* flex-container controls the layout
* flex-item responds to the rules set by the container

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

These properties are used in Flexbox to control the positioning, spacing, and direction of items inside a flex container.

**1️⃣ justify-content**

* Aligns flex items **horizontally** along the main axis
* Used to control space distribution between items

**Common values:**  
• flex-start – items start from the left  
• flex-end – items align to the right  
• center – items are centered  
• space-between – equal space **between** items  
• space-around – equal space **around** each item  
• space-evenly – equal space **between and around** items

**2️⃣ align-items**

* Aligns items **vertically** along the cross axis
* Used to control the vertical position of items inside the container

**Common values:**  
• stretch – items stretch to fill height (default)  
• flex-start – items align to the top  
• flex-end – items align to the bottom  
• center – items align to the vertical center  
• baseline – items align based on text baseline

**3️⃣ flex-direction**

* Sets the **direction** of flex items inside the container
* Determines whether items are placed in a row or column

**Common values:**  
• row – items are placed from left to right (default)  
• row-reverse – items go from right to left  
• column – items are placed from top to bottom  
• column-reverse – items go from bottom to top

**Conclusion:**  
These Flexbox properties help in building flexible and responsive layouts by managing the alignment and flow of elements inside a container.

**• CSS Grid**

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

CSS Grid is a layout model used to design **two-dimensional layouts** (both rows and columns). It gives better control for building complex web page structures.

🎯 **Grid**

* Allows layout in **both rows and columns**
* Items are placed into defined grid cells
* Uses properties like display: grid, grid-template-columns, grid-template-rows, gap, etc.

🧩 **Example:**

.container {

display: grid;

grid-template-columns: 1fr 1fr;

gap: 10px;

}

📏 **Flexbox**

* Works in **one direction at a time** (either row or column)
* Great for aligning items in a single line
* Uses properties like display: flex, flex-direction, justify-content, align-items

🔍 **Difference between Grid and Flexbox**

* 🟦 Grid → for **2D layouts** (row and column)
* 🟨 Flexbox → for **1D layouts** (row *or* column)
* 🟩 Grid gives more control over exact placement
* 🟧 Flexbox is better for distributing items along one axis

💡 **When to use Grid instead of Flexbox**

* 🖼️ Use **Grid** for full-page layouts, image grids, dashboards
* 📌 Use **Flexbox** for navbars, buttons, single-line layouts
* 🛠️ Combine both for responsive and complex layouts

✅ **Conclusion:**  
Both CSS Grid and Flexbox are powerful layout tools.  
Use **Grid** for structured 2D layouts, and **Flexbox** for flowing 1D layouts.  
They can also work together in modern responsive designs.

**2) Describe the grid-template-columns, grid-template-rows, and grid- gap properties. Provide examples of how to use them.**

These properties are used in **CSS Grid** to define the structure and spacing of rows and columns in a grid container.

🧱 **grid-template-columns**

* Defines the **number and width of columns** in the grid
* You can set fixed widths (e.g., 100px), percentages, or flexible units (fr)

**Example:**

.container {

display: grid;

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

}

👉 This creates **3 columns**: first and third take equal space, middle column is double wide.

📏 **grid-template-rows**

* Defines the **height of each row** in the grid
* Like columns, you can use px, %, fr, or auto

**Example:**

.container {

display: grid;

grid-template-rows: 100px 200px auto;

}

👉 This creates **3 rows**: 100px high, 200px high, and one auto-sized.

📐 **grid-gap** *(also written as gap)*

* Adds **space between grid rows and columns**
* You can define a single value or two values (row-gap column-gap)

**Example:**

.container {

display: grid;

grid-gap: 20px;

}

👉 Adds **20px gap** between all rows and columns.

Or:

grid-gap: 10px 30px;

👉 Adds **10px between rows** and **30px between columns**.

✅ **Conclusion:**

* grid-template-columns → sets **column structure**
* grid-template-rows → sets **row structure**
* grid-gap → adds **spacing** between grid items

**• Responsive Web Design with Media Queries**

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

**Media queries** are CSS rules that apply styles **based on the device's screen size, resolution, or other features**. They are essential for building **responsive websites** that look good on all devices—mobiles, tablets, and desktops.

📱 **What are media queries?**

* Special CSS conditions that apply styles only when certain rules are met
* Used to check screen **width, height, orientation, resolution**, etc.
* Start with @media followed by a condition

🔧 **Basic Syntax:**

@media (max-width: 600px) {

body {

background-color: lightblue;

}

}

👉 This will change the background color **only when the screen width is 600px or less**

🌐 **Why media queries are important:**

* 🖥️ Make your website **responsive** to different screen sizes
* 📲 Improve **user experience** across devices
* 📸 Help control layout, font size, image size, and element visibility
* ⚙️ Reduce the need for separate mobile and desktop sites

✅ **Conclusion:**  
Media queries are a key part of responsive design.  
They allow you to apply different styles based on device features, making your website look and work better on all screen sizes.

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

📄 **CSS Code:**

@media (max-width: 600px) {

body {

font-size: 14px;

background-color: #f2f2f2;

padding: 10px;

}

h1 {

font-size: 24px;

text-align: center;

}

p {

font-size: 13px;

line-height: 1.6;

}

.container {

width: 100%;

padding: 10px;

}

}

📝 **Explanation:**

* **@media (max-width: 600px)** → Targets small screens (mobile phones)
* **body** → Font is smaller, background soft, and padding added
* **h1** → Heading text shrinks and is centered
* **p** → Paragraph text is easier to read with more spacing
* **.container** → Makes layout full-width with padding for breathing space

✅ **Result:**  
This media query makes the webpage more readable and clean on **mobile screens**, ensuring better user experience.

**• Typography and Web Fonts**

**1) Explain the difference between web-safe fonts and custom web fonts. Why might you use a web-safe font over a custom font?**

Fonts used in web design can be categorized as **web-safe fonts** or **custom web fonts**, and each has its own purpose and use case.

🔤 **Web-safe fonts**

* These are **pre-installed fonts** found on almost all devices and operating systems
* They **do not need to be downloaded** by the browser
* Examples: Arial, Times New Roman, Verdana, Courier New, Georgia

✅ **Advantages:**

* Load faster (no extra download)
* High compatibility across devices and browsers
* Great for performance and fallback usage

🌐 **Custom web fonts**

* These are fonts that are **not installed by default** on devices
* They are loaded from the web using services like **Google Fonts**, **Adobe Fonts**, or hosted manually
* Example: Poppins, Lobster, Roboto, Open Sans

✅ **Advantages:**

* Offer unique design and branding
* More variety in font styles and weights
* Helps websites stand out visually

**Why use web-safe fonts instead of custom fonts?**

* 🔋 Better **performance** (no font loading delay)
* 📶 Useful when **internet speed is slow**
* 💻 Ensures consistent **rendering on older devices or browsers**
* 🧱 Good for **fallback options** when a custom font fails to load

✅ **Conclusion:**

* **Web-safe fonts** are fast and reliable, great for accessibility and performance
* **Custom fonts** are stylish and unique, great for design and branding  
  The best practice is to use custom fonts with a **web-safe fallback** to combine both beauty and reliability.

**2) What is the font-family property in CSS? How do you apply a custom Google Font to a webpage?**

The font-family property in CSS is used to **set the typeface** (font) for text on a webpage.

🔤 **What is font-family?**

* Specifies the **font style** for an element
* You can list **multiple fonts**, so if the first one isn't available, the next is used
* Syntax:

font-family: "Roboto", Arial, sans-serif;

👉 This means: Use **Roboto** first. If it’s not available, use **Arial**, then any **sans-serif** font.

🌐 **How to apply a custom Google Font to a webpage:**

✅ **Step 1: Import the font in HTML <head>**  
Go to [https://fonts.google.com](https://fonts.google.com/), choose a font (e.g., *Poppins*), and copy the <link> tag.

<link href="https://fonts.googleapis.com/css2?family=Poppins&display=swap" rel="stylesheet">

✅ **Step 2: Use the font in your CSS**

body {

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

}

👉 This sets the font of the whole page to **Poppins**, with **sans-serif** as fallback.

📝 **Why use Google Fonts?**

* Free and easy to use
* Huge collection of styles
* Works across all modern browsers
* Adds a professional, stylish look to websites

✅ **Conclusion:**  
The font-family property controls the appearance of text.  
To use a Google Font, just **import it in your HTML** and apply it in your CSS using font-family.