Module 3 – Mernstack – CSS and CSS3

Name: Shivam Bavda

⇨ Theory Assignment

=>**CSS Selectors & Styling**

Q.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 HTML elements. It tells the browser which HTML elements should be targeted with specific styles defined in a CSS rule.

1. Element Selector

- Targets all HTML elements of a specific type.

Example:

```
p {
  color: blue;
}
```

This sets the text color of all <p> (paragraph) elements to blue.

---

2. Class Selector

- Targets elements that have a specific class attribute.
- Prefixed with a dot (.) in CSS.

HTML Example:

```
<div class="One">Hello</div>
<p class="One">World</p>
```

CSS Example:

```
.One {
  background-color: yellow;
```

}

This applies a yellow background to all elements with the highlight class.

---

3. ID Selector

- Targets a single element with a specific id attribute.
- Prefixed with a hash (#) in CSS.
- IDs should be unique within a page.

HTML Example:

<h1 id="class">Welcome</h1>

CSS Example:

#class {

font-size: 32px;

}

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

Ans: Every CSS selector has a **specificity value**, which is a weight calculated based on the presence of different types of selectors:

1. **Inline styles** (added directly to an element using the style attribute) have the highest specificity.
2. **ID selectors** are more specific than class selectors or element selectors.
3. **Class selectors**, attribute selectors, and pseudo-classes (like :hover) have medium specificity.
4. **Element selectors** and pseudo-elements (like ::before) have the lowest specificity.

5. The **universal selector** (*), combinators (+, >, ~, space), and grouping selectors (,) do **not** add to specificity.

Q.3 What is the difference between internal, external, and inline CSS? Discuss the advantages and disadvantages of each approach.

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

---

1. Inline CSS

Definition:
CSS is applied directly to an HTML element using the style attribute.

Example:

This is inline styled text.

Advantages:

- Quick and easy to apply styles to a single element.
- Useful for testing or small adjustments.

Disadvantages:

- Poor maintainability; styles are scattered in the HTML.
- Difficult to reuse styles.
- Violates the separation of content and design.
- Increases HTML file size.

---

2. Internal CSS

Definition:
CSS is written inside a <style> tag within the <head> section of an HTML document.

Example:

```
<head>
  <style>
    p {
      color: blue;
      font-size: 18px;
    }
  </style>
</head>
```

Advantages:

- Better organization than inline CSS.
- Useful for styling a single page.
- Keeps styles in one place for that specific page.

Disadvantages:

- Styles apply only to that page.
- Not reusable across multiple pages.
- Can increase page load time if used repeatedly in large projects.

---

3. External CSS

Definition:
CSS is written in a separate .css file and linked to the HTML document using the <link> tag.

Example:

```
<head>
  <link rel="stylesheet" href="styles.css">
</head>
```

styles.css:

```
p {
  color: green;
  font-size: 20px;
}
```

Advantages:

- Styles are reusable across multiple pages (DRY principle).
- Keeps HTML clean and improves maintainability.
- Faster loading after the first page, due to browser caching.

Disadvantages:

- Requires an extra HTTP request to load the CSS file.
- Slight delay in styling if the external file is large or not cached.

=>**CSS Box Model**

Q.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 sizes are calculated. Every HTML element is represented as a rectangular box, made up of the following components:

🔷 Components of the Box Model

1. **Content**
    - The actual content of the element (e.g., text, image).
    - Its size is defined by width and height properties.
2. **Padding**
    - The space between the content and the border.
    - It adds space **inside** the element, increasing its total size.

- Background color extends into the padding area.

3. **Border**
    - The line surrounding the padding and content.
    - Can have a width, style, and color.
    - Adds to the total size of the element.
4. **Margin**
    - The space **outside** the border, separating the element from others.
    - Margins are transparent and do **not** affect the element's background.

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

Ans: The box-sizing property in CSS determines how the **total size** of an element is calculated—specifically, how **width** and **height** are applied with respect to **padding** and **border**.

🟩 **1. content-box (✅ Default Value)**

- The width and height apply **only to the content**.
- **Padding and border are added outside** the specified width/height.
- This increases the total size of the element.

Example:

box-sizing: content-box; /* default */

width: 200px;

padding: 20px;

border: 5px solid black;

🟦 **2. border-box**

- The width and height include **content + padding + border**.
- The total size of the element remains as defined.
- Makes layout easier and more predictable.

Example:

box-sizing: border-box;

width: 200px;

padding: 20px;

border: 5px solid black;

- **content-box** is the default value of box-sizing.

=>**CSS Flexbox**

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

Ans: **CSS Flexbox** (Flexible Box Layout) is a layout model in CSS designed to provide a more efficient way to arrange elements within a container, even when their size is unknown or dynamic.

Flexbox makes it easier to:

- Align elements vertically and horizontally
- Distribute space dynamically
- Create responsive designs with less code

Why is Flexbox Useful?

- Simplifies complex layouts (like centering elements)
- Adapts to different screen sizes (responsive design)
- Handles spacing and alignment without needing floats or positioning
- Allows easy reordering of elements

Key Terms

🔷 **Flex Container**

- The **parent** element that holds and controls the layout of its children.
- Created by applying display: flex; or display: inline-flex;.

Example:

```
.container {
  display: flex;
}
```

```
html
<div class="container">
  <div>Item 1</div>
  <div>Item 2</div>
</div>
```

Flex Container:

- flex-direction: row, column, row-reverse, column-reverse
- justify-content: align items horizontally (e.g., center, space-between)
- align-items: align items vertically (e.g., stretch, center, flex-start)

- flex-wrap: controls whether items wrap to the next line

Flex Items:

- flex: shorthand for flex-grow, flex-shrink, and flex-basis
- align-self: overrides align-items for a specific item
- order: controls item order

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

Ans: In CSS Flexbox, layout and alignment are controlled through specific properties applied to the **flex container**. The three key properties are:

---

1. flex-direction

- **Purpose:** Defines the **main axis** along which flex items are laid out.
- **Effect:** Determines whether items are arranged in a row or a column, and in which direction.

Common Values:

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

---

2. justify-content

- **Purpose:** Aligns flex items along the **main axis** (which depends on flex-direction).
- **Effect:** Controls the distribution of space **between** and **around** items.

Common Values:

- flex-start: Items align at the **start** of the main axis.

- flex-end: Items align at the **end** of the main axis.
- center: Items are **centered** along the main axis.
- space-between: Items have **equal space between** them.
- space-around: Items have **equal space around** them (half space at edges).
- space-evenly: Items have **equal space between and around** them.

---

3. align-items

- **Purpose:** Aligns flex items along the **cross axis** (perpendicular to the main axis).
- **Effect:** Controls how items are positioned vertically in a row or horizontally in a column.

=>**CSS Grid**

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

Ans: **CSS Grid** is a powerful two-dimensional layout system in CSS that allows you to design web pages using rows and columns. It gives you fine control over the layout of items within a container and is ideal for more complex, grid-based designs.

Key Features:

- **Two-dimensional layout**: You can control both rows and columns.
- **Explicit placement**: You can place items exactly where you want using grid lines, names, or areas.
- **Grid template**: Define rows and columns using grid-template-rows, grid-template-columns, and grid-template-areas.

Flexbox (Flexible Box Layout) is a one-dimensional layout model in CSS that distributes space along a single axis—either a row or a column. It’s designed for aligning items within a container and is great for smaller-scale layouts.

Key Features:

- **One-dimensional layout**: Works on either a row **or** column at a time.
- **Content-based sizing**: Automatically adjusts items based on content size.
- **Alignment**: Strong support for aligning and justifying items.
- **Grid vs Flexbox: Key Differences**

Feature	CSS Grid	Flexbox
Layout Direction	Two-dimensional (rows/cols)	One-dimensional (row *or* col)
Best For	Entire page or major layout	Components and UI elements
Placement	Can explicitly place items	Relies on flow/order
Alignment	Complex, supports grid areas	Simple, axis-based alignment
Overlapping Items	Supported via grid layers	Not easily supported

When to Use Grid Over Flexbox:

Use **CSS Grid** when:

- You need a two-dimensional layout.
- You're designing overall page structure (e.g., header, sidebar, content, footer).
- You want precise placement of elements in a grid (e.g., dashboards, image galleries).

Use **Flexbox** when:

- You’re aligning items in a single row or column (e.g., navbar, form elements).
- You need to distribute space dynamically among items.
- You’re building components inside a grid layout.

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

Ans: **grid-template-columns**

Defines the **number and size** of columns in a grid layout.

Syntax:

grid-template-columns:

<track-size> <track-size> ...;

Common Units:

- px, em, % — fixed units
- fr — fraction of available space
- auto — size based on content

🔹 grid-template-rows

- Defines the **height** of each row in a grid layout.
- **Syntax:**

```
grid-template-rows:

<track-size> <track-size> ...;
```

🔹 grid-gap / gap

Specifies the **spacing between grid items.**

- gap is shorthand for setting both row and column gaps.
- row-gap and column-gap can also be used separately.

Example:

```
<div class="container">
```

```
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>

.container {
  display: grid;
  grid-template-columns: 100px 1fr 2fr;
  grid-template-rows: 50px 50px;
  gap: 10px;
}
```

=>**Responsive Web Design with Media Queries**

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

Ans: Media queries are **CSS rules** that let you apply styles based on specific conditions, such as the **width**, **height**, **device type**, or **screen resolution**.

They allow you to **ask the device or browser**:

Why Are Media Queries Important for Responsive Design?

Responsive design means making your website or app **look good and work well on all screen sizes** — from large desktop monitors to small mobile phones.

Media queries help achieve this because they let you:

✅ Adjust layouts for small vs. large screens
✅ Change font sizes or button sizes for touch devices
✅ Hide or show elements depending on the device
✅ Provide different styles for landscape vs. portrait orientation

Without media queries, you'd have a "one-size-fits-all" design, which often **doesn't work well on mobile** or **looks broken on large screens**. Media queries ensure your site **adapts smoothly**.

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

Ans: When a website is viewed on small devices like smartphones, the screen space is limited.

Without adjustments:

- Text might appear **too large**, taking up too much space.
- It can cause **horizontal scrolling**, breaking the layout.
- It may **reduce readability** because large fonts can crowd the screen.

By reducing the font size slightly, we make sure the content **fits comfortably** on smaller screens without sacrificing readability.

A **media query** is a CSS feature used to apply specific styles only when certain conditions are met, such as screen width, height, or device type.

To adjust the font size for screens smaller than 600 pixels, we use a **media query** with the condition max-width: 600px.

This means:

*"If the screen width is **600px or less**, apply the following CSS styles."*

The steps are:

1. **Set a default font size** (for all screens).
2. **Write a media query** that targets screens with a maximum width of 600px.
3. **Inside the media query**, define the new font size you want for smaller screens.

=>**Typography and Web Fonts**

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

Ans:

- **What are Web-Safe Fonts?**

- **Web-safe fonts** are fonts that are **pre-installed on most operating systems and devices** (like Windows, macOS, Android, iOS).
- Common examples:
  - ✅ Arial
  - ✅ Times New Roman
  - ✅ Verdana
  - ✅ Georgia
  - ✅ Courier New

These fonts are called *web-safe* because **you can be confident** they will display the same way across almost all browsers and devices **without needing to load any extra files**.

➢ **What are Custom Web Fonts?**

- **Custom web fonts** are fonts that are **not guaranteed to be installed** on the user's device.
- They are usually provided through **font files** (like .woff, .ttf) or loaded from online services such as:
  - ✅ Google Fonts
  - ✅ Adobe Fonts
  - ✅ Self-hosted font files

When you use custom fonts, the browser **downloads the font file** when loading the webpage so it can display the custom style.

Key Differences

Feature	Web-Safe Fonts	Custom Web Fonts
Availability	Pre-installed on most devices	Must be downloaded from the web or server
Load Time	No extra load time	Adds to page load time (downloads font file)

Feature	Web-Safe Fonts	Custom Web Fonts
Design Flexibility	Limited, only basic/common fonts	Huge variety of styles, weights, and unique designs
Compatibility	Works everywhere by default	Needs fallback fonts if the font fails to load

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

Ans: **What is the font-family Property in CSS?**

The font-family property in CSS defines **which font** should be used to display the text on a webpage.

✅ It allows you to **set the primary font** and also list **fallback fonts** in case the first one isn't available.

✅ You can specify:

- Named fonts (like 'Roboto' or 'Times New Roman')
- Generic font families (like serif, sans-serif, monospace)

Step 1: Import the Google Font

There are two main ways:

✅ **Using a <link> tag in HTML <head>:**

html

<head>

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

</head>

✅ **Or using @import in your CSS:**

```
css

@import
url('https://fonts.googleapis.com/css2?family=Roboto&display=swap');
```

Step 2: Apply the Font in CSS

In your stylesheet:

```
css

body {

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

}
```

This tells the browser to:

1. Use the **Roboto** font (downloaded from Google Fonts).
2. If Roboto fails, use a **generic sans-serif** font.