Module 3 – Mernstack – CSS and CSS3

Theory Assignment:

CSS Selectors & Styling

Question 1: What is a CSS selector? Provide examples of element, class, and ID selectors.
Ans:1 A **CSS selector** is essentially a pattern used to select and style HTML elements. It tells the browser *which* elements on the page to apply certain styles to.

1. Element Selector

This targets all instances of a specific HTML tag.

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

2. Class Selector

It targets elements that have a specific `class` attribute. Classes can be reused across multiple elements.

```
.card {
  border: 1px solid black;
  padding: 10px;
}
```

3. ID Selector

Targets a single, unique element by its `id` attribute. An ID should only be used once per page.

```
#header {
  background-color: lightgray;
}
```

Module 3 – Mernstack – CSS and CSS3

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

Ans:2 CSS specificity is a system that determines which rule takes precedence when more than one rule could apply to the same element. It's like a game of who's louder in a room of style declarations.

When multiple CSS styles target the same element, the browser resolves conflicts using **specificity**, which gives weight to different selectors: inline styles are strongest, followed by ID selectors, then class/attribute/pseudo-class selectors, and finally element selectors. If two rules have the same specificity, the one that appears later in the stylesheet wins.

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

Ans:3 CSS can be applied in three main ways—**inline**, **internal**, and **external**—each with its own use cases and trade-offs.

Inline CSS involves adding styles directly to HTML elements using the `style` attribute. It's useful for quick fixes or styling individual elements but clutters the HTML and reduces reusability.

Internal CSS is written within a `<style>` tag in the HTML document's `<head>`. It's handy when you want to apply styles to a single page, keeping styling and structure together. However, it doesn't scale well for large sites since styles must be repeated across multiple pages.

External CSS keeps all styles in a separate `.css` file and links it using the `<link>` tag. It promotes clean code, allows reusing styles across multiple pages, and makes maintenance easier. The downside is that it adds one more HTTP request, which might slightly impact page load time (though browsers usually cache it). For larger, multi-page websites, external CSS is generally the best practice because of its scalability and separation of concerns. Let me know if you'd like to see an example of all three in action!

Module 3 – Mernstack – CSS and CSS3

CSS Box Model

Question 1: Explain the CSS box model and its components (content, padding, border, margin). How does each affect the size of an element?
Ans:1 the **CSS box model** is essential for understanding how elements are visually laid out on a web page. Every HTML element is treated like a rectangular box, and the box model defines what's inside and around that box.

Content: This is the actual text or image inside the element — it forms the core of the box.
Padding: This creates space *between the content and the border*. It increases the element's total size but stays inside the border.
Border: It wraps around the padding (if any) and content. Its width contributes directly to the element's overall dimensions.
Margin: This is the outermost layer, creating space *between the element and its neighbors*. Margins don't affect the element's internal size but do affect layout spacing.

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

Ans:2 In CSS, the `box-sizing` property controls how the total width and height of an element are calculated. With `content-box` (the default value), only the content area is counted as the element's width and height—any padding and border are added on top, increasing the overall size. In contrast, `border-box` includes padding and borders within the specified width and height, meaning the content area shrinks to make room for them. This makes `border-box` often easier for layout consistency, especially when dealing with precise element sizing.

Module 3 – Mernstack – CSS and CSS3

CSS Flexbox

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

Ans:1 CSS **Flexbox** (short for *Flexible Box Layout*) is a layout model designed to make it easier to arrange elements in a one-dimensional space—either horizontally (row) or vertically (column). It allows elements to automatically adjust their size, spacing, and alignment to best fit different screen sizes and content needs, making it perfect for responsive design.

Flex Container: This is the parent element that has `display: flex` or `display: inline-flex` applied. It defines the flex context for all its child elements. The container controls the direction, alignment, wrapping, and spacing of the items inside it.

```
.container {

  display: flex;

}
```

Flex Items: These are the direct children of a flex container. Each item can grow, shrink, or stay at a specific size depending on the container's rules and available space.

```
<div class="container">

  <div class="item">One</div>

  <div class="item">Two</div>

  <div class="item">Three</div>

</div>
```

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

Ans:2 `flex-direction`

This sets the **main axis** along which the flex items are laid out:

- `row` (default): items flow left to right
- `row-reverse`: items flow right to left
- `column`: items stack top to bottom
- `column-reverse`: items stack bottom to top

`justify-content`

This aligns items along the **main axis** (set by `flex-direction`), controlling horizontal alignment in `row` or vertical in `column`:

- `flex-start`: items align at the start
- `center`: items centered
- `space-between`: equal space between items
- `space-around`: space on both sides of items
- `space-evenly`: equal space around all items

`align-items`

This aligns items along the **cross axis** (perpendicular to the main axis):

- `stretch` (default): items stretch to fill the container
- `flex-start`: items align at the start
- `center`: items centered
- `flex-end`: items align at the end
- `baseline`: aligns text baselines

Module 3 – Mernstack – CSS and CSS3

CSS Grid

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

Ans:1 CSS **Grid** is a powerful layout system that allows you to design web pages using a **two-dimensional grid**—meaning it handles layout in both **rows and columns**.You define a grid container using `display: grid`, set up rows and columns with `grid-template-rows` and `grid-template-columns`, and place child elements (called *grid items*) precisely where you want them within that grid structure.

Grid vs. Flexbox:

- **Grid** is for **two-dimensional layouts**—think of a photo gallery or a dashboard where you care about both row and column placement.
- **Flexbox** is best for **one-dimensional layouts**—like aligning items in a single row or column (e.g. navigation bars or list items).
- Grid gives you fine control over spacing, alignment, and overlap across both axes, while Flexbox automatically flows items in one direction and wraps them if needed.

When to Use Grid Over Flexbox:

- Use **Grid** when your design involves complex layouts with **both rows and columns** defined up front.
- Choose **Flexbox** when you're aligning items **in a single direction** or letting items adapt dynamically without needing a strict structure.

For example, a product card layout with image, title, and price in a defined grid works beautifully with Grid, while a responsive nav bar or button group is a Flexbox favorite.

Module 3 – Mernstack – CSS and CSS3

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

Ans:2 In CSS Grid, these three properties are your toolkit for defining the structure and spacing of the grid:

1. `grid-template-columns`

This property defines the **number and width of columns** in the grid

```
.container {
 display: grid;
  grid-template-columns: 200px 1fr 2fr;
}
```

2. `grid-template-rows`

Similar to columns, this property defines the **height of the rows**.

```
.container {
  display: grid;
  grid-template-rows: 100px auto 50px;
}
```

3. `grid-gap` (or `gap`)

This adds **spacing between grid rows and columns**.

```
.container {
  display: grid;
  gap: 20px;
}
```

Module 3 – Mernstack – CSS and CSS3

Responsive Web Design with Media Queries

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

Ans:1 Media queries in CSS allow you to apply different styles depending on the device's characteristics, like screen size, resolution, orientation, or color preference. They're crucial for responsive design because they let websites adapt gracefully to various screen sizes—ensuring content looks good and functions well on phones, tablets, laptops, or large monitors. For example, you might use a media query to stack columns vertically on smaller screens while keeping them side by side on larger ones, enhancing usability across all devices.

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

Ans:1 CSS media query that changes the font size when the screen width is less than 600 pixels:

```
@media (max-width: 600px) {

  body {

    font-size: 14px;

  }

}
```

This rule tells the browser: *"If the viewport is 600px wide or less, make the body text 14 pixels."* It helps make content more readable on smaller devices like phones.

Module 3 – Mernstack – CSS and CSS3

Typography and Web Fonts

Question 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:1 The difference between **web-safe fonts** and **custom web fonts** comes down to availability and consistency across devices. **Web-safe fonts** are pre-installed on most operating systems—like Arial, Times New Roman, or Verdana—so when you use them in your CSS, they're guaranteed to display the same on nearly all browsers without needing to load anything extra. **Custom web fonts**, on the other hand, are fonts that aren't typically available by default and must be loaded from a server using `@font-face` or services like Google Fonts.

You might choose a web-safe font over a custom font for **performance reasons**—because they require no additional loading time—and for **maximum compatibility**, especially on slow networks or older devices. It's a good fallback option when speed, simplicity, or guaranteed rendering takes priority over uniqueness or branding.

Question 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 specifies the typeface that should be used for text content. You can list multiple fonts as a fallback system—if the browser can't load the first font, it tries the next one. For example:

```
body {

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

}
```

Here, the browser tries to use **Roboto** first, then **Arial**, and finally a generic **sans-serif** font if the others aren't available.

To use a **custom Google Font**, follow these steps:

1. **Choose a font** at Google Fonts.
2. **Copy the** `<link>` **tag** provided (usually looks like this

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

Place it inside the `<head>` of your HTML document.

3. Apply the font in your CSS using `font-family`:

```
body {

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

}
```

This setup lets you bring in modern, stylish fonts while keeping fallbacks in case the custom font fails to load.