THEORY ASSIGNMENT.css

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

Answer: A CSS selector is a pattern used to select and style HTML elements. It tells the browser which HTML elements to apply specific CSS rules to.

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

1. **Element Selector**

- **Description:** Targets all elements of a specific type.
- **Syntax**

```
element {
  /* CSS rules */
}
```

- **Example;**

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

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

---

3. ID Selector

- **Description:** Targets a single element with a specific ID.
- **Syntax:**

```
#idname {
  /* CSS rules */
}
```

- **Example;**

```
#header {
  font-size: 24px;
}
```

This applies a font size of 24px to the element with id="header"

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

Answer: CSS specificity is a set of rules browsers use to determine which CSS rule to apply when multiple rules target the same element.

When styles conflict, the one with higher specificity takes precedence.

- **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 style=""	1-0-0-0
! important rule	Overrides all

- **How Conflicts Are Resolved**

When multiple rules apply to the same element:

1. **Compare specificity**: Higher specificity wins.
2. **If specificity is equal**, the **later** rule in the CSS (or the one loaded last) wins.
3. ! important overrides everything (but can be overridden by another! important with higher specificity).

Example:

Html

```
<p id="intro" class="highlight">Hello! </p>
```

Css

```
p {
  color: blue;          /* Specificity: 0-0-0-1 */
}

. highlight {
  color: green;         /* Specificity: 0-0-1-0 */
}

#intro {
  color: red;           /* Specificity: 0-1-0-0 */
}
```

The text will be **red**, because #intro has the highest specificity.

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

Answer; Here's a clear explanation of the **difference between internal, external, and inline CSS**, along with their **advantages and disadvantages**:

♦ Inline CSS

- CSS is written directly within the HTML tag using the style attribute.

Example:

Html

```
<p style="color: red; font-size: 16px;">This is inline CSS. </p>
```

➢ Advantages:

- Quick and easy for small changes.
- Useful for testing or overriding other styles.
- Does not require a separate stylesheet.

➢ Disadvantages:

- Not reusable — must repeat the same styles for each element.
- Makes HTML cluttered and harder to maintain.
- Poor separation of content and presentation.
- Lower priority in larger projects.

2. Internal CSS

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

Example:

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

✅ Advantages:

- Better organization than inline CSS.
- Good for styling a single page.
- Keeps styles in one place within the HTML file.

❌ Disadvantages:

- Still mixes style with content.
- Styles can’t be reused across multiple pages.
- Increases HTML file size.

🔷 3. External CSS

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

Example:

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

styles.css:

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

✅ Advantages:

- Best practice for larger websites.
- Styles are reusable across multiple pages.
- Clean separation of content (HTML) and presentation (CSS).
- Faster loading after the first page (CSS is cached).

❌ **Disadvantages:**

- Requires multiple files to manage.
- Doesn't work if the CSS file fails to load (e.g., offline or broken link).
- Slightly more complex to set up for beginners.

❖ **Summary Table**

Type	Location	Reusable	Maintains Separation	Best For
Inline	In HTML tag (style)	✗	✗	Quick tests, one-time use
Internal	<style> in <head>	✗	⚠ Partially	Single-page styling
External	Separate .css file	✅	✅	Multi-page or large sites

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

Answer: The **CSS box model** is a fundamental concept that describes how elements on a webpage are structured and how their sizes and spacing are calculated. Each HTML element is essentially a box, and the box model defines the space that this element occupies.

✅ **Components of the CSS Box Model:**

1. **Content**
    - This is the **innermost part** of the box.
    - It contains the text, images, or other content of the element.
    - The **width** and **height** properties usually refer to the size of this area.
2. **Padding**
    - Padding is the space **between the content and the border**.
    - It creates **internal spacing** within the element, pushing the border outward.
    - It is **transparent** and increases the total size of the element.
3. **Border**
    - The border wraps around the padding and content.
    - You can style it (width, color, style), and it also **adds to the element's size**.

4. **Margin**
    - This is the **outermost space**, outside the border.
    - It creates **space between the element and other elements**.
    - It is **transparent** and does **not affect the element's internal size**, but does affect spacing between elements.

📏 How Each Component Affects the Size

By default (with box-sizing: content-box), the **total size** of an element is calculated like this:

Total Width = content width + padding (left + right) + border (left + right)

Total Height = content height + padding (top + bottom) + border (top + bottom)

The **margin** is added outside this total and affects the spacing around elements, not their internal size.

🛠️ Alternative: box-sizing: border-box

When using box-sizing: border-box, the **padding and border are included** in the declared width and height. This makes layout calculation simpler:

Total Width = declared width (includes content + padding + border)

Total Height = declared height (includes content + padding + border)

This model is often preferred because it avoids unexpected size increases.

🖼 Visual Summary

```
|<---- margin ---->|
  |<-- border -->|
    |<- padding ->|
      [ content ]
```

📝 Example

```
div {
  width: 200px;
  padding: 10px;
  border: 5px solid black;
  margin: 20px;
}
```

- If box-sizing: content-box:
  - Total width = 200 + 20 (padding) + 10 (border) = **230px**
  - Plus margin = **230px + 40px = 270px** of total horizontal space
- If box-sizing: border-box:
  - Total width remains **200px**, including padding and border

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

In **Flexbox** (Flexible Box Layout), the properties justify-content, align-items, and flex-direction are used to control the layout and alignment of flex items within a flex container. Here's a breakdown of each:

---

1. flex-direction

- **Purpose**: Defines the **main axis** along which flex items are placed in the flex container.
- **Values**:
    - row (default): Items are placed **horizontally** from left to right.
    - row-reverse: Items are placed **horizontally** from right to left.
    - column: Items are placed **vertically** from top to bottom.
    - column-reverse: Items are placed **vertically** from bottom to top.
- **Example**:

```
.container {
  display: flex;
  flex-direction: row;
}
```

2. justify-content

- **Purpose**: Aligns flex items **along the main axis** (which is set by flex-direction).
- **Values**:
  - flex-start: Items align to the **start** of the main axis.
  - flex-end: Items align to the **end** of the main axis.
  - center: Items are **centered** along the main axis.
  - space-between: Equal **space between** items; first and last item at edges.
  - space-around: Equal space **around** each item.
  - space-evenly: Equal space **between and around** all items.
- **Example**:

```
.container {
  display: flex;
  justify-content: space-between;
}
```

♦ **Summary Table:**

Property	Axis Affected	Purpose
flex-direction	Main Axis	Sets direction of items
justify-content	Main Axis	Aligns items along main axis
align-items	Cross Axis	Aligns items along cross axis

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

Answer:

CSS Grid:

CSS Grid is a powerful layout system designed for building **two-dimensional** layouts on the web. It allows you to control both **rows and columns** simultaneously, making it ideal for complex layouts.

Flexbox vs CSS Grid – Key Differences:

Feature	**CSS Grid**	**Flexbox**
Layout Type	**2D layout** (rows & columns)	**1D layout** (either row *or* column)
Axis Control	Main and cross axes at once	Only one axis at a time
Content vs Layout	Layout-first	Content-first
Item Placement	Items can be placed anywhere	Items flow in order
Best For	Grid-like, complex page structures	Aligning items in a row/column

- **When to Use Grid over Flexbox:**

♦ **Use Grid when**:

- You need to design a **full-page layout** or grid-based design (like a photo gallery or dashboard).
- You want **precise control** over rows and columns.
- Your layout requires **overlapping items** or **named areas**.

♦ **Use Flexbox when**:

- You're aligning **items in a single direction** (like a navbar or button group).
- You want simple alignment and distribution of items.

Question 2: Describe grid-template-columns, grid-template-rows, and grid-gap. Provide examples.

Answer: grid-template-columns

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

- **Syntax**:
- grid-template-columns: 100px 200px auto;

➤ Creates 3 columns: 100px, 200px, and auto-sized.

- **Fractional Units** (fr):
- grid-template-columns: 1fr 2fr 1fr;

➤ Columns share available space in a 1:2:1 ratio.

♦ grid-template-rows

Defines the **number and size of rows** in the grid.

- **Syntax**:
- grid-template-rows: 50px 100px;

➤ Two rows: 50px and 100px.

- You can also use auto, fr, %, etc.

♦ grid-gap (or gap)

Sets the **space between rows and columns**.

- **Syntax**:
- grid-gap: 20px;

➤ 20px gap between rows and columns.

- **Shorthand for row and column gap**:
- grid-gap: 10px 30px; /* 10px row gap, 30px column gap */

Note: Modern syntax prefers gap over grid-gap, but both are valid.

♦ Example: Basic Grid Layout

```
.container {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;
  grid-template-rows: 100px 100px;
  gap: 20px;
}
<div class="container">
```

```
  <div>Item 1</div>
  <div>Item 2</div>
  <div>Item 3</div>
  <div>Item 4</div>
  <div>Item 5</div>
  <div>Item 6</div>
</div>
```

♦ This creates:

- 3 columns (1:2:1 ratio)
- 2 rows (each 100px tall)
- 20px gap between items

8. Responsive Web Design with Media Queries

Theory Assignment