CSS Selectors & Styling

Theory Assignment

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

Definition:- A CSS selector is a pattern used to select and style HTML elements. It defines which HTML elements the CSS rules will apply to.

Types of CSS Selectors

1. Element Selector

Targets all elements of a specific type (e.g., all <p> or <h1> elements).

Example:-

```
p {
    color: blue;
    font-size: 16px;
}
```

2. Class Selector (.)

Targets elements with a specific class attribute. Classes can be applied to multiple elements.

Example:-

```
<style>
.highlight {
    background-color: yellow;
    font-weight: bold;
}
```

</style>

This paragraph is highlighted.

3. ID Selector (#)

unique element using an `id` attribute. IDs should be unique per page.

Example:-

<style>

#main-title {

color: red;

text-align: center;

}

</style>

<h1 id="main-title">Welcome to My Website</h1>

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

CSS specificity is a ranking system that determines which CSS rule takes precedence when multiple rules apply to the same element. When there is a conflict, the rule with the highest specificity wins.

1.Inline styles (highest priority) → `1000`

2.ID selectors **(`#id`)** → `100`

3.Class, attribute, and pseudo-class selectors (.class, [attr], :hover) → `10`

4.Element and pseudo-element selectors (h1, p, ::before) → 1

5.Universal selector (`*`) and inherited styles → 0 (lowest priority)

Example : Competing Rules

<style>

p { color: blue; }

.text { color: red; }

#special { color: green; }

</style>

Hello World!

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

1.inline css

Inline CSS applies styles directly to an HTML element using the `style` attribute.

Example:-

This is inline CSS

Advantages:-

- Quick and easy to apply to a single element.
- Overrides other CSS rules (high specificity)

Disadvantages:-

- Not reusable (styles must be repeated for each element).
- Hard to maintain in large projects.
- Decreases readability and makes HTML cluttered

2.Internal css:-

Internal CSS is written inside the <style> tag within the <head> section of an HTML document

Example:-

```
<head>
   <style>
      p {
         color: blue;
         font-size: 18px;
      }
   </style>
</head>
<body>
   <p>This is internal CSS</p>
</body>
```

Advantages:-

- Keeps styles separate from HTML elements (cleaner than inline CSS).
- Useful for single-page websites where external CSS may be unnecessary.

Disadvantages:-

- Not reusable across multiple pages (styles apply only to one HTML file).
- Increases page load time if styles are long

3.External css:-

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

Example:-

```
<head>
<style>
p {
   color: green;
```

```
    font-size: 20px;
}
</style>
 <link rel="stylesheet" href="styles.css">
</head>
<body>
    <p>This is external CSS</p>
</body>
```

Advantages:

- Best for large projects (keeps styles separate from HTML).
- Reusable across multiple pages (single `.css` file for the whole website).
- Improves page loading speed (CSS files are cached by browsers).

Disadvantages:-

- Requires an extra HTTP request (browser must load an external file).
- No immediate effect (changes require reloading the page)

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 describes how every HTML element is structured and how it takes up space on a webpage. It consists of four components that affect an element's total size:

1.Content : The actual content inside the element (text, images, etc.).

2.Padding :The space between the content and the border.

3.Border : A visible or invisible outline surrounding the padding and content.

4.Margin → The space outside the element, creating distance between elements.

Example:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CSS Box Model</title>
    <style>
        .box {
            width: 200px;
            height: 100px;
            background-color: lightblue;
            padding: 20px;
            border: 5px solid black;
            margin: 30px;
        }
    </style>
</head>
<body>

    <div class="box">This is a box</div>
```

</body>

</html>

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

1.justify-content:-

This property defines how flex items are distributed along the main axis of the flex container. The main axis is determined by the flex-direction (more on that later), but by default, it's horizontal (left to right). justify-content controls the spacing and alignment of items in that direction.

1.flex-start: Items are packed toward the start of the main axis (e.g., left in a row).

2.flex-end: Items are packed toward the end of the main axis (e.g., right in a row).

3.center: Items are centered along the main axis, with equal space on either side.

4.space-between: Items are evenly distributed; the first item is at the start, the last at the end, and equal gaps exist between the rest.

5.space-around: Items are evenly distributed with equal space around each item (including half-sized spaces at the start and end).

6.space-evenly: Items are distributed with equal space between and around them, including full-sized spaces at the start and end.

2.align-items:-

This property controls the alignment of flex items along the cross axis, which is perpendicular to the main axis. By default (with a horizontal main axis), the cross axis is vertical. It's about how items are positioned "up and down" within the container's height.

1.stretch: Items stretch to fill the container along the cross axis (default, if no height is set on items).

2.flex-start: Items align to the start of the cross axis (e.g., top in a row).

3.flex-end: Items align to the end of the cross axis (e.g., bottom in a row).

4.center: Items are centered along the cross axis.

5.baseline: Items are aligned so their text baselines match up (useful for text-heavy layouts).

3.flex-direction:-

This property sets the direction of the main axis, essentially defining the flow of your flex items. It’s the foundation that justify-content and align-items build upon.

1.row: Items flow horizontally, left to right (default).

2.row-reverse: Items flow horizontally, right to left.

3.column: Items flow vertically, top to bottom.

4.column-reverse: Items flow vertically, bottom to top.

Example:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Flexbox Example</title>

  <style>

    .container {

      display: flex;

      border: 2px solid black;

      height: 300px;

      margin: 20px;

    }
```

```
    .item {
      background-color: lightblue;
      padding: 20px;
      font-size: 18px;
      text-align: center;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="item">Item 1</div>
    <div class="item">Item 2</div>
    <div class="item">Item 3</div>
  </div>
</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 and Flexbox are both powerful layout systems in CSS, but they serve slightly different purposes and excel in different scenarios.

CSS Grid is a two-dimensional layout system, meaning it allows you to control both rows *and* columns at the same time.

It's like a table, but way more flexible and without the semantic baggage. You define a grid container with rows and columns, then place items into specific grid cells or let them flow automatically.

Key features:

1.Explicit grid definition: You set the number and size of rows and columns using properties like grid-template-rows and grid-template-columns.

2.Precise placement: You can position items exactly where you want them using grid-row and grid-column (e.g., "put this item in row 2, column 3").

3.Gap control: Use gap (or row-gap and column-gap) to space out grid cells.

4.Alignment: Offers fine-tuned control over alignment in both dimensions with properties like justify-items, align-items, and their content equivalents.

Key Differences:-

1.Dimensions:
- Grid: Two-dimensional (rows *and* columns).
- Flexbox: One-dimensional (rows *or* columns).

2.Control:
- Grid: Offers precise control over the entire layout structure.
- Flexbox: Focuses on content flow and distribution within a single axis.

3.Complexity:
- Grid: More powerful but can be overkill for simple layouts.
- Flexbox: Simpler and more intuitive for linear arrangements.

You'd choose Grid over Flexbox when:

1.You need a two-dimensional layout: If your design involves both rows and columns—like a dashboard, a gallery with consistent spacing, or a complex page layout—Grid is the way to go. For example, a 3x3 photo gallery is much easier to manage with Grid.

2.Precise placement matters: If you want an item to span specific rows and columns (e.g., a sidebar spanning rows 1-3 next to a main content area), Grid's explicit placement shines.

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

1.grid-template-columns

Defination: Defines the number of columns in your grid and their sizes. Each value represents a column's width.

Values: Can use units like px, %, fr (fraction of available space), auto (content-based size), or keywords like minmax() and repeat().

used: On the grid container (element with display: grid).

- px (fixed):Sets column width in pixels (100px 200px 300px).
- fr (fractional):Sets column width relative to the container (50% 50%).
- fr (fractional):Distributes available space (1fr 2fr 1fr).
- auto:Column width adjusts based on content.
- repeat(n, value):Creates multiple columns of the same width.

2.grid-template-rows

Defination: Defines the number of rows in your grid and their heights. Each value represents a row's height.

Values: Same as grid-template-columns—px, %, fr, auto, minmax(), etc.

used: On the grid container.

- px (fixed):Sets row height in pixels (100px 200px).
- auto:Row height adjusts based on content.
- fr (fractional):Distributes available space between rows.
- repeat(n, value):Creates multiple rows of the same height.

3.grid-gap

Defination: Sets the space between grid cells (rows and columns). It's a shorthand for row-gap (vertical spacing) and column-gap (horizontal spacing).

Values: Any length unit (px, em, rem, %, etc.).

used: On the grid container.

- gap: 10px; : Adds 10px spacing between rows & columns.
- row-gap: 20px; column-gap: 30px; : Different spacing for rows & columns.

Example:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CSS Grid Example</title>
    <style>
        .container {
            display: grid;
            grid-template-columns: repeat(2, 3fr);
            grid-template-rows: repeat(2, 100px);
            gap: 15px;
            background-color: red;
            padding: 20px;
        }

        .item {
            background-color: green;
            color: white;
            display: flex;
            justify-content: center;
            align-items: center;
            font-size: 20px;
            border-radius: 5px;
        }
    </style>
</head>
<body>

    <div class="container">
        <div class="item">A</div>
        <div class="item">B</div>
        <div class="item">C</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 that allow you to apply styles conditionally based on characteristics of the user's device or viewport, such as its width, height, resolution, orientation, or even whether it's in dark mode.

They're written using the @media rule, followed by a condition, and then the styles that apply when that condition is met.

Example:-

```
@media (max-width: 600px) {
  body {
    font-size: 14px;
  }
}
```

Common Conditions:-

- max-width: Applies styles if the viewport is narrower than the specified value.
- min-width: Applies if the viewport is wider than the value.
- orientation: Targets portrait or landscape modes.
- resolution: For high-DPI screens (e.g., min-resolution: 2dppx).
- Multiple conditions can be combined with and, or (comma), or not.

Responsive design is all about making websites adapt seamlessly to different devices—phones, tablets, desktops, etc. Media queries are the cornerstone of this because they let you tailor the layout, typography, and other styles to suit the screen size or context.

Device Diversity:

- Screens range from tiny smartwatches to massive monitors. Media queries ensure your site looks good and works well everywhere by adjusting styles dynamically.

Improved User Experience:

- On a small phone screen, a multi-column layout might be unreadable, but a media query can stack it into a single column.

example:-

```
@media (max-width: 500px) {
  .columns {
    flex-direction: column; /* Stacks Flexbox items */
  }
}
```

Efficiency Over Separate Sites:

- Before media queries (introduced in CSS3), developers often built separate mobile sites (e.g., m.example.com). Media queries let you handle all devices with one codebase, saving time and maintenance effort.

Control Over Breakpoints:

- You define “breakpoints” where the design shifts (e.g., 768px for tablets). This precision ensures layouts adapt logically rather than relying on rigid, one-size-fits-all rules.

example:-

```
.sidebar {
  width: 300px;
}
@media (max-width: 768px) {
  .sidebar {
    width: 100%; /* Full width on smaller screens */
  }
}
```

Accessibility and Context:

- Beyond size, media queries can respond to user preferences (e.g., @media (prefers-color-scheme: dark) for dark mode) or high-contrast settings, making your site more inclusive.

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

example:-

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Responsive Font Size</title>

    <style>

        body {

            font-size: 20px;

            padding: 20px;

            text-align: center;

            font-family: Arial;

        }

        @media (max-width: 600px) {

            body {

                font-size: 20
```

```
        }
    }
  </style>
</head>
<body>
  <h1>Responsive Font Size </h1>
  <p>hello world</p>
</body>
</html>
```

Typography and Web Fonts

Theory Assignment

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?

Key Differences:-

Availability:

- Web-safe: Pre-installed, always available.
- Custom: Must be downloaded, depends on network and hosting.

Variety:

- Web-safe: Small, generic set of fonts.
- Custom: Endless creative possibilities

Performance:

- Web-safe: No additional requests, faster page load.

- Custom: Extra HTTP request(s), potential delay (mitigated by formats like WOFF2 or font preloading).

Control

- Web-safe: Limited to system rendering, slight variations across platforms.
- Custom: Precise design control, consistent look everywhere (once loaded).

Fallbacks:

- Web-safe: Rarely need fallbacks since they're ubiquitous.
- Custom: Require fallbacks (e.g., sans-serif) for cases where the font fails to load

Definition:

- Web-Safe :Web-safe fonts are typefaces pre-installed on most operating systems (e.g., Windows, macOS, Linux) and devices, ensuring they render consistently without additional downloads. Examples include Arial, Times New Roman, Verdana, and Georgia.
- Custom Web:Custom web fonts are typefaces not typically installed on devices by default. They're sourced from external files (e.g., .woff, .ttf) hosted on your server or a service like Google Fonts, Adobe Fonts, or Font Awesome. Examples include Roboto, Open Sans, or a bespoke font you've licensed.

why you might choose a web-safe font over a custom web font:

Faster Performance: Web-safe fonts are pre-installed on devices, so they load instantly without extra HTTP requests. Custom fonts require downloading, which can delay rendering (e.g., 100-500ms), impacting page speed—critical for user retention and SEO.

Guaranteed Availability: Since they're built into most operating systems (e.g., Arial, Times New Roman), web-safe fonts always work, even if the network fails. Custom fonts risk not loading, leaving users with fallbacks that might disrupt your design.

Simplicity: No need for @font-face, font files, or CDNs. Web-safe fonts are plug-and-play, saving setup time—ideal for quick projects or minimal designs.

Compatibility: They're supported across all devices and browsers, even outdated ones, while custom fonts depend on modern format support (e.g., WOFF2) and proper hosting.

Reliability in Low Bandwidth: On slow or unreliable connections, custom fonts might not load, but web-safe fonts ensure text remains readable.

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 is used to specify the font of text elements on a webpage. It allows developers to define multiple font choices (fallback fonts) to ensure proper rendering across different devices and browsers.

Syntax:

```
selector {

    font-family: "Preferred Font", "Backup Font", generic-family;

}
```

Example:-

```
body {

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

}
```

How to Apply a Custom Google Font to a Webpage?

1.go to Google Fonts and pick a font (e.g., Poppins).

2.Add the Font to Your HTML (<link> Method)

insert the following <link> inside the <head> section of your HTML:

```
<head>

    <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;700&display=swap"
rel="stylesheet">

</head>
```

3. Apply the Font in CSS

```
body {
```

```
    font-family: "Poppins", sans-serif;
}
```

4.Alternative Method: Using @import in CSS

```
@import
url('https://fonts.googleapis.com/css2?family=Poppins:wght@400;700&display=swap');
body {
    font-family: "Poppins", sans-serif;
}
```

Example:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Google Fonts Example</title>
    <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;700&display=swap"
rel="stylesheet">
    <style>
        body {
            font-family: "Poppins", sans-serif;
            text-align: center;
            padding: 20px;
        }
```

```
        h1 {
            font-weight: 700;
        }

        p {
            font-weight: 400;
        }
    </style>
</head>
<body>

    <h1>Using Google Fonts in CSS</h1>
    <p>This text is displayed using the "Poppins" font from Google Fonts.</p>

</body>
```