**Module 3 – Frontend – CSS and CSS3**

**CSS Selectors & Styling**

**Theory Assignment:**

• **Question 1:**

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

🡪A CSS selector is a pattern used to select the HTML elements on a web page that you want to style. It allows you to target specific elements or groups of elements to apply CSS rules to them.

* **Element Selector:** This selector targets all instances of a specific HTML element type.

**EX:**

p {  
 color: blue; */\* Styles all <p> elements with blue text \*/*  
 }

* **Class Selector:** This selector targets all HTML elements that have a specific class attribute. It is denoted by a period (.) followed by the class name.

**EX:**

. first {  
 background-color: yellow; */\* Styles all elements with class="First" with a yellow background \*/*  
 }

* **ID Selector:** This selector targets a single, unique HTML element that has a specific ID attribute. It is denoted by a hash symbol (#) followed by the ID name. An ID should be unique within an HTML document.

**Ex:**

#main-header {  
 font-size: 24px; */\* Styles the element with id="main-header" with a font size of 24px \*/*  
 }

**• 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 rules are applied to an element when multiple rules target the same element and define conflicting styles. It is a set of rules that browsers use to resolve style conflicts.

**Conflict Resolution:**

When multiple styles conflict for a single element, the browser resolves them based on the following:

* **Specificity:**

The rule with the highest specificity value wins.

* **Order of Appearance (Cascade):**

If two or more rules have the exact same specificity, the rule that appears later in the stylesheet or the document (if inline styles are involved) takes precedence. This is known as the "cascade" principle.

* **Importance (! important):**

The! important declaration can override all other specificity rules. However, its use is generally discouraged as it can make CSS harder to maintain and debug.

* **Inheritance:**

Some CSS properties are inherited by child elements from their parent elements. If a style is not explicitly defined for a child element, it may inherit the value from its parent.

• **Question 3:**

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Inline CSS** | **Internal CSS** | **External CSS** |
| **Location** | It is used within HTML tag using the style attribute. | It is used within <head> section of HTML document. | It is used in a separate .css file. |
| **Selector Scope** | Affects a single element or a group of elements. | Affects multiple elements within the same HTML element. | Affects multiple HTML documents or an entire website. |
| **Reusability** | Not reusable. Styles need to be repeated for each element. | Can be reused on multiple elements within the same HTML document. | Can be reused on multiple HTML documents or an entire website. |
| **Priority** | Highest priority. Overrides internal and external styles. | Medium priority. Overrides external styles but can be overridden by inline styles. | Lowest priority. Can be overridden by both inline and internal styles. |
| **File Size** | Inline styles increase the HTML file size, which can affect the page load time. | Internal styles are part of the HTML file, which increases the file size. | External styles are in a separate file, which reduces the HTML file size and can be cached for faster page loads. |
| **Maintainability** | Not easy to maintain. Changes need to be made manually to each element. | Relatively easy to maintain. Changes need to be made in one place in the <head> section. | Easiest to maintain. Changes need to be made in one place in the external .css file. |

🡪

| **Approach** | **Where Defined** | **Best For** | **Advantages** | **Disadvantages** |
| --- | --- | --- | --- | --- |
| **Inline** | In element (style) | Quick fixes, testing | Simple, immediate | Not reusable, messy |
| **Internal** | <style> in <head> | Small/single pages | Centralized per page | Not reusable across pages |
| **External** | Separate .css file | Large/multi-page projects | Clean, reusable, cached | Needs extra file load |
|  |  |  |  |  |

**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?

* **Content:**
  + This is the innermost area where the actual content of the element resides, such as text, images, or other media.
  + **Effect on size:** The width and height CSS properties directly control the dimensions of the content area.
* **Padding:**
  + This is a transparent area that surrounds the content, creating space between the content and the element's border.
  + **Effect on size:** Padding adds to the element's total visible size. For example, if an element has a width of 100px and padding-left and padding-right of 10px each, the total visible width (including padding but excluding border and margin) will be 120px (100 + 10 + 10).
* **Border:**
  + This is a line that surrounds the padding and content areas, defining the visual boundary of the element.
  + **Effect on size:** Similar to padding, the border also adds to the element's total visible size. A border-width of 2px on all sides will add 4px to both the total width and height of the element (2px on each side).
* **Margin:**
  + This is a transparent area outside the border, creating space between the element and other adjacent elements on the page.
  + **Effect on size:** Margin creates space around the element, affecting its position relative to other elements, but it does not add to the element's own visible dimensions. It controls the external spacing.

**• Question 2:**

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

🡪 content-box (Default):

* In this model, the width and height properties you set for an element apply only to the content area.
* Any padding and border values are added on top of this content width and height, increasing the overall dimensions of the element.
* For example, if you set width: 100px; and padding: 10px; border: 2px solid black; the total rendered width of the element will be 100px (content) + 20px (padding) + 4px (border) = 124px.

 🡪border-box:

* In this model, the width and height properties you set for an element include the content, padding, and border.
* The browser automatically adjusts the content area's size to accommodate the specified padding and border within the declared width and height.
* For example, if you set width: 100px; and padding: 10px; border: 2px solid black; the total rendered width of the element will be exactly 100px. The content area will shrink to 100px - 20px (padding) - 4px (border) = 76px.

Which is the default?

The default value for the box-sizing property in CSS is content-box.

**CSS Flexbox**

**Theory Assignment:**

**• Question 1:**

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

🡪 **CSS Flexbox**

**Flexbox (Flexible Box Layout)** is a CSS layout module designed to arrange elements in a one-dimensional (row or column) layout. It helps developers create responsive and flexible web layouts without using float, table, or complicated positioning.

It is especially useful for:

* Distributing space between items.
* Aligning items vertically and horizontally.
* Handling dynamic sizes of elements in a container.
* Making layouts responsive across different screen sizes.

**Key Terms**

1. **Flex-Container**
   * The parent element that uses display: flex; or display: inline-flex;
   * It defines a flex context for its direct children.
   * Properties applied to the flex-container (e.g., flex-direction, justify-content, align-items) control the positioning and alignment of its child elements.

**Example:**

.container {

display: flex;

flex-direction: row; /\* or column \*/

justify-content: center;

align-items: center;

}

**Flex-Item**

* The direct child elements of a flex-container.
* They automatically become flexible and can be controlled using properties like flex-grow, flex-shrink, and flex-basis.
* These properties determine how much space each item will take relative to others.

**Example:**

. item {

flex: 1; /\* Item grows and shares available space equally \*/}

**• Question 2:**

Describe the properties justify-content, align-items, and flexdirection used in Flexbox.

🡪**1. justify-content**

* **Definition:** Aligns **flex-items horizontally** (along the **main axis**).
* **Values:**
  + flex-start → Items align at the start of the container (default).
  + flex-end → Items align at the end.
  + center → Items align at the center.
  + space-between → Equal space between items, first and last items stick to edges.
  + space-around → Equal space around items, including sides.
  + space-evenly → Equal spacing between all items and edges.

**2. align-items**

* **Definition:** Aligns **flex-items vertically** (along the **cross axis**).
* **Values:**
  + flex-start → Items align at the top of the container.
  + flex-end → Items align at the bottom.
  + center → Items align at the center vertically.
  + stretch → Items stretch to fill the container (default, if no height set).
  + baseline → Items align by their text baseline.

**3. flex-direction**

* **Definition:** Defines the **main axis direction** (row or column).
* **Values:**
  + row → Items placed left to right (default).
  + row-reverse → Items placed right to left.
  + column → Items placed top to bottom.
  + column-reverse → Items placed bottom to top.

**Example:**

.container {

display: flex

flex-direction: row;

justify-content: center;

align-items: flex-start;

}

**CSS Grid**

**Theory Assignment:**

**• Question 1:**

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

🡪**CSS Grid**

* CSS Grid is a **two-dimensional layout system** in CSS.
* It allows you to arrange elements into **rows and columns** simultaneously.
* The parent element is called a **grid container**, and its children are **grid items**.
* You can define **explicit rows and columns** with properties like grid-template-rows, grid-template-columns, and place items precisely using grid-row and grid-column.

**Flexbox**

* Flexbox is a **one-dimensional layout system**.
* It helps distribute items along a **single axis** (row **or** column).
* The parent is a **flex container**, and children are **flex items**.
* It’s excellent for controlling spacing, alignment, and order of items along that axis.

**When to Use Grid over Flexbox**

✅ Use **CSS Grid** when:

* You need a **full page or section layout** (e.g., a webpage with header, sidebar, main content, and footer).
* You want **precise control** of rows and columns.
* The design is inherently **two-dimensional**.

✅ Use **Flexbox** when:

* You’re arranging items in a **single row or column** (like a nav menu, button group, or form elements).
* The layout requires **content-based flexibility** rather than strict grid placement.

**• 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**

* Defines the **number and size of columns** in a grid container.
* You can set fixed sizes (e.g., 100px), relative sizes (fr unit), or even use auto for automatic sizing.

**Example:**

.container {

display: grid;

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

}

➡ Creates **3 columns**:

* 1st = 100px,
* 2nd = 200px,
* 3rd = takes remaining space.

**2. grid-template-rows**

* Defines the **number and size of rows** in the grid.
* Similar to columns, you can use fixed sizes, fractions (fr), or auto.

✅ Example:

.container {

display: grid;

grid-template-rows: 150px 100px auto;

}

➡ Creates **3 rows**:

* 1st = 150px,
* 2nd = 100px,
* 3rd = adjusts automatically.

**3. grid-gap (or gap in modern CSS)**

* Sets the **space between rows and columns** in the grid.
* You can set one value (applies to both rows and columns) or two values (row-gap column-gap).

✅ Example:

.container {

display: grid;

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

grid-template-rows: auto auto;

grid-gap: 20px;

}

**Responsive Web Design with Media Queries**

**Theory Assignment**

**• Question 1:**

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

🡪A media query checks for certain conditions (like screen width) and applies CSS rules only if those conditions are true.

🡪Media queries in **CSS** are rules that let you apply different styles depending on the **device’s characteristics**, such as screen size, resolution, or orientation.

**Why Media Queries are Important in Responsive Design:**

1. **Device Adaptability** – Ensure content looks good on all devices (desktop, tablet, mobile).
2. **Improved User Experience** – Text, buttons, and layouts adjust to be readable and usable without zooming or scrolling.
3. **Mobile-first Design** – Designers can build layouts for smaller screens first, then expand for larger ones.
4. **Better Performance** – Avoids loading unnecessary large layouts or assets on small devices.
5. **Accessibility** – Makes websites more usable for everyone, regardless of screen size or device.

**• Question 2:**

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

🡪A basic media query to adjust the font size of a webpage for screens smaller than 600px utilizes the @media rule with the max-width feature.

Code

*/\* Default font size for larger screens \*/*  
body {  
 font-size: 16px;  
}  
  
*/\* Media query for screens with a maximum width of 600px \*/*  
@media screen and (max-width: 600px) {  
 body {  
 font-size: 14px; */\* Adjust font size for smaller screens \*/*  
 }  
}

**Typography and Web Fonts**

**Theory Assignment**

**1. Web-Safe Fonts**

* Definition: Fonts that are pre-installed on most operating systems (Windows, macOS, Linux, etc.) and browsers.
* Examples: Arial, Times New Roman, Verdana, Georgia, Courier New, Trebuchet MS.
* How They Work: Since these fonts are already available on users’ devices, the browser just applies them directly without needing to download anything.

**2. Custom Web Fonts**

* Definition: Fonts that are not pre-installed on devices but are loaded via the web, usually through services like Google Fonts, Adobe Fonts, or self-hosted font files (e.g., .woff, .ttf).
* Examples: Open Sans, Roboto, Lato, Poppins, Montserrat.
* How They Work: The browser downloads the font file from a server and then renders the text with it.

**3. Why Use a Web-Safe Font Over a Custom Font?**

You might prefer web-safe fonts because:

* Performance: No extra download time—faster page load speed.
* Compatibility: Works consistently across different devices and browsers.
* Fallback Reliability: If custom fonts fail to load due to poor network, users still see the intended font.
* Accessibility: Lighter for low-bandwidth users and ensures readability.
* Simplicity: Easy to implement without needing external services.

• Question 2:

What is the font-family property in CSS? How do you apply a custom Google Fonts to a webpage?

🡪The font-family property in CSS specifies the font to be used for an HTML element. It can accept multiple font names as a "fallback" system, meaning if the browser cannot render the first specified font, it will attempt to use the next one in the list. This ensures that the text will always be displayed with a suitable font.

**Applying a Custom Google Font to a Webpage**

**Option A: Using <link> in HTML <head>**

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

**Option B: Using @import in CSS**

@import url('https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swa