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

**CSS Selectors & Styling**

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

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

**Examples:**

1. Element selector → selects all elements of a given type.
2. p {
3. color: blue;
4. }

(This will make all <p> elements’ text blue.)

1. Class selector → selects elements with a specific class (denoted by .).
2. .highlight {
3. background-color: yellow;

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

CSS specificity is a set of rules the browser uses to decide which CSS rule is applied when multiple rules target the same element.

Order of specificity (lowest → highest):

1. Element selectors (e.g., p, h1) → lowest priority.
2. Class selectors (e.g., .btn, .highlight).
3. ID selectors (e.g., #header).
4. Inline styles (e.g., <p style="color:red;">) → highest priority.
5. If two rules have the same specificity, the one written last in the CSS file is applied.

Example:

<p id="special" class="highlight">Hello</p>

p { color: blue; } /\* element selector \*/

.highlight { color: green; } /\* class selector \*/

#special { color: red; } /\* ID selector \*/

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

1. **Inline CSS**
   * Written inside the style attribute of an HTML element.
2. **<p style="color: red;">Hello</p>**

**Advantages:**

* + Useful for testing or overriding small styles.  
    Disadvantages:
  + Breaks separation of content and style.

1. **Internal CSS**
   * Written inside <style> tags in the <head> section of the HTML document**.**
2. **<style>**
3. **p { color: blue; }**
4. **</style>**

**Advantages:**

* + Good for single-page styling.
  + Keeps HTML and CSS somewhat organized.  
     **Disadvantages:**

1. **External CSS**
   * Written in a separate .css file and linked with <link> tag.

**<link rel="stylesheet" href="styles.css">**

**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?**

The CSS box model describes how every HTML element is represented as a rectangular box that consists of four layers:

1. **Content**
   * The actual content of the element (text, image, etc.).
   * Controlled by properties like width, height, font-size.
2. **Padding**
   * The space between the content and the border.
   * Increases the element’s size, since padding is added inside the box.
3. **padding: 20px;**
4. **Border**
   * The line surrounding the padding and content.
   * Thickness adds to the overall element size.
5. **border: 2px solid black;**
6. **Margin**
   * The space outside the border, creating distance between this element and others.
   * Margins do not increase the element’s actual box size, but they affect spacing around it.

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

The box-sizing property controls how the total size of an element is calculated.

1. **content-box (default)**
   * width and height apply only to the content box.
   * Padding and border are added outside the specified width/height.
2. **box-sizing: content-box; /\* default \*/**

**Example:**

div {

width: 200px;

padding: 20px;

border: 10px solid;

}

1. **border-box**
   * width and height include content + padding + border.
   * The specified width is the final size of the element.
2. **box-sizing: border-box;**

**Example:**

div {

width: 200px;

padding: 20px;

border: 10px solid;

}

**CSS Flexbox**

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

**CSS Flexbox (Flexible Box Layout)** is a layout module in CSS that makes it easier to **align, distribute, and arrange space** among items in a container, even when their size is unknown or dynamic.

**Why it’s useful:**

* Automatically adjusts elements to fit different screen sizes (responsive design).
* Makes vertical and horizontal alignment easy.
* Provides control over spacing and order without complex CSS hacks.

**Key Terms:**

1. **Flex Container**
   * The parent element where display: flex; is applied.
   * Defines the flex context for its children.
2. .container {
3. display: flex;
4. }
5. **Flex Items**
   * The direct child elements inside a flex container.
   * Their behavior (size, alignment, spacing) is controlled by flex properties.
6. <div class="container">
7. <div class="item">1</div>
8. <div class="item">2</div>
9. <div class="item">3</div>
10. </div>

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

1. **justify-content** (controls alignment **along the main axis**)
   * Determines how flex items are spaced/distributed horizontally (row) or vertically (column).
2. justify-content: flex-start; /\* default: items start from left \*/
3. justify-content: flex-end; /\* items go to right \*/
4. justify-content: center; /\* items in center \*/
5. justify-content: space-between;/\* space between items \*/
6. justify-content: space-around; /\* equal space around items \*/
7. justify-content: space-evenly; /\* equal spacing between and around items \*/

1. **align-items** (controls alignment **along the cross axis**)
   * Defines how items are aligned vertically (when flex-direction = row).
2. align-items: flex-start; /\* items align at top \*/
3. align-items: flex-end; /\* items align at bottom \*/
4. align-items: center; /\* items center vertically \*/
5. align-items: stretch; /\* items stretch to fill container \*/
6. align-items: baseline; /\* align based on text baseline \*/

1. **flex-direction** (sets the **main axis** of flex items)
   * Defines the direction in which items are placed.
2. flex-direction: row; /\* default: left → right \*/
3. flex-direction: row-reverse; /\* right → left \*/
4. flex-direction: column; /\* top → bottom \*/
5. flex-direction: column-reverse;/\* bottom → top \*/

**CSS Grid**

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

**CSS Grid** is a powerful 2D layout system in CSS that allows you to arrange elements in **rows and columns**. It’s ideal for creating complex, grid-based layouts like webpages, dashboards, or galleries.

**Key Differences between Grid and Flexbox:**

| **Feature** | **Flexbox** | **Grid** |
| --- | --- | --- |
| Layout Type | 1D (one dimension: row **or** column at a time) | 2D (both rows **and** columns at the same time) |
| Best For | Aligning and distributing elements in a line (navigation bars, buttons, cards). | Full page layouts, galleries, complex grids. |
| Axis | Works with **main axis** and **cross axis** | Works with **rows and columns together** |
| Control | Content-based (items shrink/grow) | Layout-based (explicit placement of items in cells) |

**When to use Grid:**

* Complex page layouts (rows + columns together).
* You need precise placement of items (e.g., grid-area).

**When to use Flexbox:**

* Simpler, one-dimensional layouts.
* When you want items to flexibly distribute themselves in one direction.

**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 how many columns and their width.
2. .container {
3. display: grid;
4. grid-template-columns: 100px 200px auto;
5. }

1. **grid-template-rows**
   * Defines how many rows and their height.
2. .container {
3. display: grid;
4. grid-template-rows: 150px 150px auto;
5. }

First 2 rows = 150px, third row grows automatically.

1. **grid-gap (or gap in modern CSS)**
   * Adds space **between grid items** (both rows & columns).
2. .container {
3. display: grid;
4. grid-template-columns: repeat(3, 1fr);
5. grid-gap: 20px;
6. }

Creates 3 equal-width columns with 20px spacing between them.

You can also specify separately:

gap: 20px 40px; /\* 20px row-gap, 40px column-gap \*/

**Example Putting it All Together:**

<div class="container">

<div class="item">1</div>

<div class="item">2</div>

<div class="item">3</div>

<div class="item">4</div>

<div class="item">5</div>

<div class="item">6</div>

</div>

.container {

display: grid;

grid-template-columns: repeat(3, 1fr); /\* 3 equal columns \*/

grid-template-rows: 100px 100px; /\* 2 equal rows \*/

gap: 15px; /\* space between items \*/

}

.item {

background: lightblue;

text-align: center;

font-size: 20px;

padding: 20px;

}

**Responsive Web Design with Media Queries**

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

A **media query** is a CSS feature that allows you to apply styles **only when certain conditions are true**, such as screen size, device type, or orientation.

**Why they’re important:**

* They enable **responsive design**, so websites look good on desktops, tablets, and mobile devices.
* They help adapt layouts, font sizes, and images depending on screen width/height.
* They prevent the need for multiple separate versions of the same site.

**Basic Syntax:**

@media (condition) {

/\* CSS rules here \*/

}

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

/\* Default font size for larger screens \*/

body {

font-size: 18px;

}

/\* Media query for screens smaller than 600px \*/

@media (max-width: 600px) {

body {

font-size: 14px;

}

}

**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?**

1. **Web-safe fonts**
   * Fonts that are commonly installed on almost all operating systems (Windows, macOS, Linux, etc.).
   * Examples: **Arial, Times New Roman, Verdana, Georgia, Courier New, Tahoma**.
   * Since they’re pre-installed, the browser doesn’t need to download them → **fast loading**.
2. **Custom web fonts**
   * Fonts that are not pre-installed on devices. They must be loaded from a server (e.g., Google Fonts, Adobe Fonts).
   * Example: **Roboto, Open Sans, Lato, Poppins**.
   * They give designers more creative freedom.

**Why use a web-safe font over a custom font?**

* **Performance** → Web-safe fonts load instantly (no extra HTTP request).
* **Fallback** → Reliable if the custom font fails to load.
* **Compatibility** → Works on old browsers or restricted environments.

Best practice: Use **custom fonts for branding/design**, but provide **web-safe fallback fonts**.

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

1. **font-family property**
   * Defines which font to use for an element.
   * Accepts a **font stack** (preferred font + fallbacks).
2. p {
3. font-family: "Arial", "Helvetica", sans-serif;
4. }

If Arial isn’t available, it tries Helvetica, then defaults to any sans-serif font.

1. **Applying a Google Font**

**Step 1:** Import the font into your HTML <head> (via <link>):

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

**Step 2:** Apply it in CSS using font-family:

body {

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

}