# Wd - Css And Css 3

## CSS Selectors & Styling

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

Ans: A **CSS selector** is used to select HTML elements and apply styles to them.

**🔹 Examples:**

1. **Element Selector**  
   Selects all elements of a specific type.

p {

color: blue;

}

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

1. **Class Selector**  
   Targets elements with a specific class attribute (uses a dot .className).

.highlight {

background-color: yellow;

}

(Applies yellow background to any element with class="highlight")

1. **ID Selector**  
   Targets a specific element with an ID (uses a hash #idName).

#main-title {

font-size: 24px;

}

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

**CSS specificity** determines which style rule is applied when multiple rules target the same element.

**🔹 Specificity Ranking (Low to High):**

1. **Element selectors** – div, p, etc. (Lowest)
2. **Class selectors** – .class
3. **ID selectors** – #id (Highest)
4. **Inline styles** – style="..." in HTML (Override all)
5. **!important** – Overrides everything (but should be avoided if possible)

**🔹 Example:**

<p id="myText" class="highlight">Hello!</p>

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

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

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

**Question 3: What is the difference between internal, external, and inline CSS?**

| **Type** | **Description** | **Example** | **Advantages** | **Disadvantages** |
| --- | --- | --- | --- | --- |
| **Inline** | CSS is added directly inside an HTML tag using style="" | <p style="color:red;">Text</p> | Quick for small changes | Hard to maintain, low reusability |
| **Internal** | CSS is written inside a <style> tag in the HTML file's <head> | <style>p { color: red; }</style> | Good for single-page styling | Not reusable across multiple pages |
| **External** | CSS is in a separate .css file and linked with <link> | <link rel="stylesheet" href="style.css"> | Clean, reusable, easier to maintain | Requires extra HTTP request to load |

* 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 the size of every HTML element is calculated. It consists of 4 main parts, from innermost to outermost:**

**🔹 Box Model Components:**

**+--------------------------+**

**| Margin | ⬅ Outside space**

**| +--------------------+ |**

**| | Border | | ⬅ Element border**

**| | +--------------+ | |**

**| | | Padding | | | ⬅ Space inside border**

**| | |+------------+| | |**

**| | || Content || | | ⬅ Actual text/image/etc**

**| | |+------------+| | |**

**| | +--------------+ | |**

**| +--------------------+ |**

**+--------------------------+**

**🔹 Explanation of Each:**

1. **Content:**
   * **The actual text or image in the element.**
   * **Controlled by properties like width, height.**
2. **Padding:**
   * **Space inside the border, around the content.**
   * **Increases the size inside the box.**
3. **Border:**
   * **The line around the padding and content.**
   * **Increases total size of the element.**
4. **Margin:**
   * **Space outside the border, between this element and others.**
   * **Does not increase the element’s actual box size but pushes it away from other elements.**

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

**🔹 box-sizing controls how total element size is calculated when you set width and height.**

| **Property** | **Description** |
| --- | --- |
| **content-box (default)** | **Width/height apply only to content. Padding and border are added outside, increasing total size.** |
| **border-box** | **Width/height include content + padding + border. Total size remains as defined.** |

## 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 powerful layout model designed to **arrange items in a single dimension** (row or column). It helps create **responsive** and **space-efficient** layouts easily without using floats or positioning.

**🔹 Why Flexbox is Useful:**

* Align items horizontally or vertically
* Distribute space evenly between elements
* Easily rearrange elements
* Works well for both small and complex UI layouts

**🔹 Key Terms:**

* **Flex Container**:
  + The parent element where Flexbox is applied using:

display: flex;

= Example:

<div class="container"> <!-- This is the flex container -->

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

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

</div>

* **Flex Item**:
  + The **direct children** of the flex container.
  + They respond to the container's flex rules like spacing, alignment, etc.

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

These properties control layout behavior in Flexbox:

**🔹 flex-direction**

Defines the **main axis** (row or column) of the layout.

| **Value** | **Meaning** |
| --- | --- |
| row *(default)* | Items are laid out **horizontally** |
| column | Items are laid out **vertically** |
| row-reverse | Horizontal, but in reverse order |
| column-reverse | Vertical, but in reverse order |

.container {

display: flex;

flex-direction: row;

}

**🔹 justify-content**

Aligns items **along the main axis** (depends on flex-direction).

| **Value** | **Description** |
| --- | --- |
| flex-start (default) | Items start at the beginning |
| center | Items are centered |
| flex-end | Items end at the end |
| space-between | Equal space **between** items |
| space-around | Equal space **around** items |
| space-evenly | Equal space **between and around** |

css

CopyEdit

.container {

justify-content: center;

}

**🔹 align-items**

Aligns items **along the cross axis** (perpendicular to the main axis).

| **Value** | **Description** |
| --- | --- |
| stretch (default) | Items stretch to fill the container |
| flex-start | Aligned to the start |
| center | Aligned to the center |
| flex-end | Aligned to the end |
| baseline | Align based on text baseline |

.container {

align-items: flex-start;

}

**✅ Example:**

.container {

display: flex;

flex-direction: row;

justify-content: space-between;

align-items: center;

}

This makes items spread out horizontally with even gaps and centered vertically.

## CSS Grid

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

**🔹 What is CSS Grid?**

**CSS Grid** is a powerful layout system for **2D layouts** — it allows you to control rows **and** columns at the same time.

display: grid;

**🔹 Key Differences: Grid vs Flexbox**

| **Feature** | **CSS Flexbox** | **CSS Grid** |
| --- | --- | --- |
| Layout Direction | One-dimensional (row OR column) | Two-dimensional (rows AND columns) |
| Item Placement | Based on content flow | Based on defined grid structure |
| Alignment Control | Per line | Per row & column |
| Use Case | Navbars, lists, toolbars, etc. | Full-page layouts, image galleries, dashboards |

**✅ When to use CSS Grid:**

* When you need to design **complex layouts with both rows and columns**.
* For **grid-based page sections** (e.g., dashboard, gallery, templates).

**✅ When to use Flexbox:**

* For **simple linear layouts** (horizontal or vertical).
* For components inside a grid item (e.g., button groups, nav menus).

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

These are **core Grid properties** used to define structure and spacing in a grid layout.

**🔹 grid-template-columns**

Defines the number and size of columns.

.container {

display: grid;

grid-template-columns: 200px 1fr 100px;

}

* This creates **3 columns**: first is 200px, second takes remaining space (1fr), third is 100px.

**🔹 grid-template-rows**

Defines the number and size of rows.

.container {

grid-template-rows: 100px auto 50px;

}

* Creates 3 rows: fixed height, auto height (based on content), and small footer.

**🔹 grid-gap *(Now shorthand for row-gap and column-gap)***

Adds spacing **between rows and columns**.

.container {

grid-gap: 20px;

}

* Adds 20px space between grid rows and columns.

## Responsive Web Design with Media Queries

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

**🔹 What are Media Queries?**

**Media queries** are CSS rules that apply styles **only when certain conditions** about the device or screen size are true (like width, height, orientation, resolution, etc.).

**🔹 Why are they important?**

Media queries are essential for **responsive web design**, which means creating web pages that **look good on all screen sizes** — desktops, tablets, and mobile phones.

They allow you to:

* Adjust layouts for different screen sizes
* Resize fonts, images, or buttons
* Hide or show elements based on device type

**🔹 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**

Here’s a simple example:

@media (max-width: 600px) {

body {

font-size: 14px;

}

}

**🔹 Explanation:**

* max-width: 600px means the CSS inside will apply **only when the screen is 600px wide or smaller**.
* It changes the body’s font size to 14px on small devices like smartphones.

## 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?**

**🔹 Web-safe fonts:**

* These are **standard fonts** that are **pre-installed on most devices and operating systems**.
* Examples: Arial, Verdana, Times New Roman, Courier New, Georgia, Tahoma, Helvetica
* ✅ **Advantages**:
  + No extra loading time
  + Always available
  + Ensures consistent display across devices
* ❌ **Limitations**:
  + Limited design choice
  + May look generic or outdated

**🔹 Custom web fonts:**

* These are fonts **not installed on the user's device** but are loaded via **CSS (e.g., from Google Fonts or self-hosted)**.
* Example: Poppins, Roboto, Lato, Open Sans
* ✅ **Advantages**:
  + Great for unique branding and design
  + Modern, attractive styles
* ❌ **Limitations**:
  + Requires loading time (affects page speed)
  + Might not display if internet is slow or fails

**🔹 Why choose web-safe fonts?**

* Faster loading time
* Better fallback when performance is a priority
* No dependency on external servers

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

**🔹 What is font-family?**

The font-family property defines which font(s) should be used to display text.

body {

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

}

* You can list multiple fonts as **fallbacks** (browser uses the next if one isn’t available).

**✅ How to apply a Google Font:**

**🔹 Step 1: Copy the link from** [**Google Fonts**](https://fonts.google.com)

Example for **Roboto**:

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

Add this in the <head> of your HTML file.

**🔹 Step 2: Use it in your CSS**

body {

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

}