**Wd-Css and Css3 Theory**

**CSS Selectors & Styling**

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

A CSS selector selects the HTML element(s) you want to style. CSS selectors are used to select the HTML elements you want to style.

**Element:**

h4{

color: gold;

background-color: silver;

font-size: 1.673em;

}

**Class:**

.box{

height: 1.75em;

width: 2.057em;

background-color: brown;

border-width: 1px;

border-color: purple;

border-style: solid;

}

**ID:**

#dxt{

color: blue;

background-color: red;

height: 3.45em;

width: 4.63em;

}

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

**CSS specificity** is the set of rules browsers use to decide which CSS rule applies when multiple rules target the same element. It’s like a score — the more specific the selector, the higher its priority.

**Specificity order (from lowest to highest):**

1. **Element selectors** (div, p)
2. **Class/attribute/pseudo-class selectors** (.class, [type], :hover)
3. **ID selectors** (#id)
4. **Inline styles** (style="...")
5. **!important** overrides all (but should be used sparingly)

**More specific rules override less specific ones**, even if they appear later in the stylesheet.

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

**1. Inline CSS**

CSS is applied directly to an element using the style attribute.

**Advantages:**

* Quick for testing or small changes.
* Overrides other styles due to high specificity.

**Disadvantages:**

* Hard to maintain for large projects.
* Poor separation of content and style.
* No reusability across elements or pages.
* Bloats HTML code.

**2. Internal CSS**

CSS is placed inside a <style> tag in the <head> section of an HTML file.

**Advantages:**

* Useful for styling a single HTML page.
* Keeps styles in one place (within the document).

**Disadvantages:**

* Not reusable across multiple pages.
* Increases page size if repeated in many files.
* Less efficient than external CSS for larger sites.

**3. External CSS**

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

**Advantages:**

* Best for large projects.
* Reusable across multiple pages.
* Keeps HTML clean and style logic separate.
* Better performance (cached by browsers).

**Disadvantages:**

* Requires an extra HTTP request (can affect loading slightly if not optimized).
* No styles are visible if the CSS file fails 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 and how it interacts with surrounding elements. Every element is essentially a **rectangular box** made up of the following layers:

**1. Content**

* **What it is**: The actual content inside the element — text, images, etc.
* **Affects size?** Yes, it’s the base size.
* **Set with**: width, height

**2. Padding**

* **What it is**: Space **inside** the element, between the content and the border.
* **Affects size?** Yes, adds space **inside** the box.
* **Set with**: padding, padding-top, etc.

**3. Border**

* **What it is**: The line around the padding and content.
* **Affects size?** Yes, adds thickness **outside padding**.
* **Set with**: border, border-width, etc.

**4. Margin**

* **What it is**: Space **outside** the element, separating it from other elements.
* **Affects size?** No, but affects layout/spacing.
* **Set with**: margin, margin-top, etc.

**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.

**content-box (Default)**

* **Default behavior in CSS**
* width and height apply **only to the content.**
* **Padding and border are added outside** the content size.

**border-box**

* width and height **include** content, padding, and border.
* Makes sizing and layout more predictable.

**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 model designed to **efficiently align and distribute space** among items in a container — even when their size is unknown or dynamic.

It makes building **responsive layouts** much easier than using floats or positioning.

**Flex Container**

* The parent element with display: flex or display: inline-flex.
* It controls the layout behavior of its direct children (flex items).

**Flex Item**

* Direct children of a flex container.
* These items follow the flex container's layout rules (direction, alignment, spacing).

**Why Flexbox is Useful**

* **One-dimensional** layout system (handles rows **or** columns)
* Easily aligns items **horizontally or vertically**
* Handles **spacing, alignment, and resizing** with minimal code
* Great for building **responsive designs**

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

**flex-direction**

Defines the main axis — i.e., the direction in which flex items are laid out.

**Values:**

* row (default) → items laid out **left to right**
* row-reverse → **right to left**
* column → **top to bottom**
* column-reverse → **bottom to top**

**justify-content**

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

**Common values:**

* flex-start → items align at the start
* flex-end → at the end
* center → centered
* space-between → equal space between items
* space-around → equal space around items
* space-evenly → equal space between and around items

**align-items**

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

**Common values:**

* stretch (default) → stretch to fill container
* flex-start → align to top (or left for flex-direction: column)
* flex-end → align to bottom (or right)
* center → center items vertically (if flex-direction: row)

**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 2-dimensional layout system that lets you design web pages using rows and columns. It allows you to place items precisely in a grid structure and control both horizontal and vertical layout at the same time.

**Difference between grid and flexbox:**

| **Feature** | **CSS Grid** | **Flexbox** |
| --- | --- | --- |
| **Layout type** | 2D (rows **and** columns) | 1D (row **or** column at a time) |
| **Best for** | Full-page or section layouts | Aligning items in a single line |
| **Item control** | Position items by grid lines or area | Flow-based; items placed in order |
| **Alignment** | Complex, supports grid-gap, grid-area | Simple, great for spacing items |
| **Responsiveness** | Great with media queries or auto-fit | Great with flex-wrap |

**When to Use CSS Grid:**

* You need a complex layout with rows and columns (like a full page or card layout).
* Items are not strictly linear and may span multiple rows/columns.
* You want precise control over placement, like defining template areas.

**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, size, and width of the columns in a grid container.
* You specify column sizes separated by spaces.

**Example:**

.container {

display: grid;

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

}

This creates a grid with 3 columns:

* First column 100px wide
* Second column 200px wide
* Third column 100px wide

**grid-template-rows**

* Defines the number, size, and height of the rows in a grid container.
* Similar syntax to columns.

**Example:**

.container {

display: grid;

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

}

Creates 3 rows:

* First row 50px tall
* Second row 100px tall
* Third row 50px tall

**grid-gap (or shorthand: gap)**

* Defines the space between rows and columns (gutters).
* Can specify one value (gap between rows and columns are the same) or two values (row gap and column gap).

**Examples:**

.container {

display: grid;

grid-gap: 10px;

}

Adds a 10px gap between all 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?**

Media queries in CSS are rules that allow web developers to apply styles based on specific conditions such as screen size, resolution, orientation, or device type. They are a fundamental part of responsive web design because they enable a website’s layout and appearance to adapt to different screen sizes and devices, ensuring a consistent and user-friendly experience across desktops, tablets, and mobile phones. By using media queries, developers can define different styles for various breakpoints—for example, adjusting font sizes, repositioning elements, or hiding certain content on smaller screens. This flexibility helps create websites that are visually appealing and functionally effective on any device, which is essential in today’s multi-device digital environment.

**Question 2: Write a basic media query that adjusts the font size of a webpage 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?**

**Web-safe fonts** are fonts that are commonly pre-installed on most operating systems and devices, such as Arial, Times New Roman, or Verdana. Because they're already available on users' systems, they don't need to be downloaded, which means they load quickly and consistently across different browsers and platforms.

**Custom web fonts**, on the other hand, are fonts that are not necessarily pre-installed on users' devices. These are typically loaded from external sources (like Google Fonts or Adobe Fonts) using CSS @font-face rules. Custom fonts offer more design flexibility and branding opportunities but can increase page load times and may require fallbacks if they fail to load.

You might choose a **web-safe font** over a custom font when performance, compatibility, or simplicity is a priority—such as for fast-loading sites, email templates, or when ensuring a consistent appearance across older devices or limited internet connections.

**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 typeface (font) of text on a webpage. It allows you to define a prioritized list of fonts so that if the first choice isn't available on the user's device, the browser will try the next one in the list.

body {

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

}

**How to Apply a Custom Google Font:**

To use a Google Font, follow these steps:

**1.Include the font link in the <head> of your HTML:**

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

**2.Apply the font using the font-family property in CSS:**

body {

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

}

This tells the browser to use the **Roboto** font from Google Fonts. If it fails to load, it will use a generic **sans-serif** font as a fallback.