**CSS Selectors & Styling**

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

**ANS:** A **CSS selector** is a pattern used in CSS (Cascading Style Sheets) to select and style specific HTML elements. It tells the browser which HTML elements should be affected by the CSS rules.

**Examples:**

1. **Element Selector:**  
   Targets all elements of a specific type.

p {

color: blue;

}

This styles all <p> (paragraph) elements with blue text.

1. **Class Selector**

Targets elements with a specific class attribute.

.highlight {

background-color: yellow;

}

This styles all elements with class="highlight" to have a yellow background.

1. **ID Selector**

Targets an element with a specific ID attribute.

#main-header {

font-size: 24px;

}

This 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?

**ANS: CSS Specificity** is a set of rules browsers use to determine which style rule applies to an element when multiple rules could apply. It's essentially a way to resolve conflicts when multiple CSS selectors target the same element.

### **How Specificity Works:**

Each CSS selector has a specificity value, which is calculated based on the types of selectors used. The browser compares these values to decide which rule takes precedence.

|  |  |  |  |
| --- | --- | --- | --- |
| Category | Description | Example | Points |
| A | Inline styles | style="color: red;" | 1000 |
| B | ID selectors | #header | 100 |
| C | Class, attribute, and pseudo-classes | .menu, [type="text"] | 10 |
| D | Element and pseudo-elements | div, h1, ::before | 1 |

### **Conflict Resolution:**

When multiple rules apply to the same element:

1. **Compare Specificity**: The rule with the highest specificity wins.
2. **If Specificity is Equal**: The rule that appears later in the CSS (or imported styles) overrides earlier ones.
3. **Important Rule**: If a rule uses! important, it overrides any other rules, unless another! important rule has higher specificity.

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

**ANS:** Here's a breakdown of the **differences between internal, external, and inline CSS,** along with their advantages and disadvantages:

1. **Inline CSS:**

**Definition**: CSS styles are written directly inside an HTML element using the style attribute.

**Example**: <p style="color: red; font-size: 16px;">Hello World</p>

#### **Advantages:**

* Quick and easy for small changes.
* Useful for testing or overriding other styles quickly.

#### **Disadvantages:**

* Hard to maintain (styles are scattered throughout HTML).
* No separation of content and design.
* Difficult to reuse and apply consistent styles across multiple elements.

### **Internal CSS:**

**Definition**: CSS is placed within a <style> tag inside the <head> section of the HTML document.

**Example**: <head>

<style>

p {

color: blue;

}

</style>

</head>

#### **Advantages:**

* Useful for single-page websites or documents.
* Keeps styles centralized within the HTML file.

#### **Disadvantages:**

* Not reusable across multiple pages.
* Can clutter the HTML file if styles are lengthy.

### **External CSS:**

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

**Example**: <head>

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

</head>

#### **Advantages:**

* Best practice for large websites.
* Promotes reusability and clean separation of content and design.
* Styles can be cached by browsers, improving performance.

#### **Disadvantages:**

* Requires an extra HTTP request (though often negligible).
* No styles will show if the CSS file fails to load (e.g., due to a bad link).

**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?

**ANS:** The **CSS Box Model** is a fundamental concept in web design that describes how every HTML element is structured and how its size is calculated. It defines how elements are rendered in terms of **boxes**, consisting of the following components (from innermost to outermost):

1. **Content**

* This is the actual content of the element (text, images, etc.).
* The width and height you set using width and height apply to this area by default.

1. **Padding**

* The space **between the content and the border.**
* Increases the size of the element **without adding space outside.**
* Makes the content look less cramped.

1. **Border**

* The outline around the padding and content.
* Can be styled using border-width, border-style, and border-color.
* Adds to the total size of the element.

1. **Margin**

* The space **outside the border,** used to create space **between elements**.
* Does **not** increase the element’s size, but pushes it away from surrounding elements.

### **Total Size Calculation (Default Box Model)**

### By default, (box-sizing: content-box)

Total Width = content width + padding-left + padding-right + border-left + border-right

Total Height = content height + padding-top + padding-bottom + border-top + border-bottom

### **Alternate Model: box-sizing: border-box**

With this setting, the width and height include **content + padding + border**, making layout easier to manage.

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

**ANS:** The difference between border-box and content-box in CSS lies in **how the total width and height of an element are calculated.**

### **content-box (Default)**

* Only the **content** size is defined by width and height.
* **Padding and border are added on top** of those dimensions, increasing the total size.

#### **Example:** div {

#### width: 200px;

#### padding: 20px;

#### border: 5px solid black;

#### box-sizing: content-box;

#### }

#### Total width = 200 (content) + 40 (padding) + 10 (border) = 250px

### **2.** border-box

* The width and height include **content + padding + border.**
* Keeps the total size of the element fixed, which makes layout more predictable.

#### Example:

#### div {

width: 200px;

padding: 20px;

border: 5px solid black;

box-sizing: border-box;

}

In this case, content shrinks to fit inside the 200px total width:

* 200 = content + 40 (padding) + 10 (border)
* So content width = 150px

The default value for box-sizing is: content-box.

### **Why Use border-box?**

* Easier to manage layouts and align elements.
* Prevents unexpected overflow from padding or borders.

**CSS Flexbox**

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

**ANS: CSS Flexbox** is a powerful layout module in CSS that makes it easy to design flexible, responsive layouts without using floats or positioning. It allows items within a container to be automatically arranged depending on available space — great for aligning and distributing space among elements in a container.

**How Flexbox is Useful:**

* **Responsive Design:**

Flexbox easily adapts layouts to different screen sizes by allowing items to grow, shrink, and rearrange themselves within the container.

* **Simplified Layout:**

It reduces the complexity of creating layouts compared to using floats or absolute positioning.

* **Efficient Alignment and Distribution:**

Flexbox provides properties to easily align items within the container and distribute space between them, making it easy to achieve desired visual arrangements.

* **Control over Item Size:**

Flexbox allows for fine-grained control over how items grow and shrink, enabling you to create layouts that fit the available space effectively.

**Explanation of Terms:**

* **Flex-Container:**

The parent element to which display: flex or display: inline-flex is applied. This declaration transforms the element and its child elements into a flex container and flex items, respectively.

* **Flex-Item:**

The direct child elements within a flex container. Flexbox properties are applied to flex items to control their size, alignment, and behavior within the flex container.

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

### **ANS:**

### **Flex-direction:**

### **Purpose**: Defines the **main axis** along which the flex items are laid out.

**Values**:

* row (default): Items are placed left to right.
* row-reverse: Items are placed right to left.
* column: Items are placed top to bottom.
* column-reverse: Items are placed bottom to top.

### **justify-content:**

**Purpose**: Aligns items **along the main axis** (set by flex-direction).

**Values**:

* flex-start: Items align to the start of the main axis.
* flex-end: Items align to the end of the main axis.
* center: Items are centered along the main axis.
* space-between: Items are evenly spaced with no gap at the ends.
* space-around: Items have equal space around them.
* space-evenly: Items have equal space between and at the ends.

### **Align-items:**

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

**Values**:

* + flex-start: Items align to the start of the cross axis.
  + flex-end: Items align to the end of the cross axis.
  + center: Items are centered along the cross axis.
  + stretch (default): Items stretch to fill the container (if no height/width set).
  + baseline: Items align based on their text baseline.

**CSS Grid**

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

**ANS:**

**CSS Grid** is a two-dimensional layout system in CSS that allows you to create layouts by defining rows and columns in a grid container. It’s great for building more complex, structured layouts.

#### **Key Features of CSS Grid:**

* **Two-dimensional layout**: You can control both rows and columns.
* **Grid container & items**: You define a parent container as a grid using display: grid, and its children become grid items.
* **Grid tracks**: Use grid-template-rows and grid-template-columns to define the structure.
* **Grid areas**: You can label sections using grid-area for more semantic layout control.
* **Precise placement**: Items can be placed anywhere in the grid using line numbers or named areas.

| **Feature** | **Flexbox** | **CSS Grid** |
| --- | --- | --- |
| Layout Direction | One-dimensional (row **or** column) | Two-dimensional (rows **and** columns) |
| Best For | Aligning items in a single line or axis | Building complex, structured layouts |
| Item Placement | Based on content flow | Explicit row/column placement (precise) |
| Responsiveness | More natural for simple adaptive layouts | Powerful control over layout structure |

### **When to Use Grid Over Flexbox**

### **Use CSS Grid when:**

* You need to design a full page or section layout (headers, sidebars, footers, etc.).
* You want precise control over rows and columns.
* You are dealing with both horizontal **and** vertical alignment and positioning.
* You want to create overlapping content or use named grid areas.

#### **Use Flexbox when:**

* You’re aligning items **in a single direction** (e.g., a navbar, toolbar, or buttons).
* The content size is unknown or dynamic and you want it to adjust naturally.
* You need alignment tools like justify-content or align-items on one axis.

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

**ANS:**

### **grid-template-columns**

* **Purpose**: Defines the number and size of columns in a CSS Grid layout.
* **Syntax Example**:

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

Creates 3 columns: first is 100px, second is 200px, third takes remaining space.

* **Other common units**:
  + fr (fractional unit): 1fr 2fr divides space in a 1:2 ratio.
  + repeat() function: repeat(3, 1fr) creates 3 equal columns.

### **2. grid-template-rows**

* **Purpose**: Defines the number and size of rows in the grid layout.
* **Syntax Example**:

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

First row is 50px, second takes remaining space, third is 100px.

### **3. grid-gap (shorthand for *row-gap* and *column-gap*)**

* **Purpose**: Defines the spacing between grid rows and columns.
* **Syntax Example**:

grid-gap: 10px 20px;

Sets 10px between rows and 20px between columns.

Can also be written as:

row-gap: 10px;

column-gap: 20px;

### **Full Example**:

.grid-container {

display: grid;

grid-template-columns: 1fr 2fr 1fr;

grid-template-rows: 100px auto;

grid-gap: 20px;

}

This creates a grid with 3 columns (left and right are equal, middle is double), and 2 rows (first is 100px tall, second fills remaining height), with a 20px gap between all grid items.

**Responsive Web Design with Media Queries**

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

**ANS:**

**Media queries** are a feature in CSS that allow you to apply styles **based on the characteristics of the user's device**, such as screen width, height, resolution, orientation, or even device type.

### **Why Are They Important for Responsive Design?**

Responsive design aims to make websites look and function well across **all screen sizes and devices**—from mobile phones to large desktop monitors. Media queries help achieve this by:

* Adapting layouts, fonts, and elements based on screen size.
* Reducing the need for separate mobile and desktop sites.
* Ensuring a consistent user experience on all devices.

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

**ANS:** Here's a basic media query that adjusts the font size 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?**

### **ANS: Web-Safe Fonts**

* **Definition**: Fonts that are **pre-installed** on most operating systems and devices.
* **Examples**: Arial, Times New Roman, Georgia, Verdana, Courier New, Trebuchet MS.
* **Usage**: Specified directly in CSS (no additional setup needed).

### **Custom Web Fonts**

* **Definition**: Fonts that are **not installed by default** on devices and must be **loaded via the web**, typically using services like **Google Fonts** or by hosting font files.
* **Usage**: Requires @font-face or a <link> to load fonts from a provider.

### **Why Use Web-Safe Fonts Over Custom Fonts?**

* **Performance**: No font loading time = faster page loads.
* **Reliability**: Always available, reducing the risk of fallback fonts.
* **Accessibility**: Users with limited bandwidth or old browsers may not load custom fonts.
* **Fallback strategy**: Often used as a backup when a custom font fails to load.

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

**ANS:**

The font-family property in CSS specifies **which font(s)** should be used to display text content on a webpage.

#### **Syntax:**

selector {

font-family: "Font Name", fallback-font, generic-family;

}

* **Font Name**: The primary font you'd like to use.
* **Fallback Font**: An alternative if the primary font is unavailable.
* **Generic Family**: A general category (serif, sans-serif, monospace, etc.) used as a final fallback.

#### **Example:**

body {

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

}

### **How to Apply a Custom Google Font to a Webpage**

#### **Step 1: Import the Font from Google Fonts**

#### Add a <link> tag inside the <head> of your HTML file:

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

#### **Step 2: Apply the Font in CSS**

Use the font-family property with the font's name:

|  |  |  |  |
| --- | --- | --- | --- |