**1. What is a CSS Selector?**

A CSS selector is a way to target and style specific HTML elements.

* **Element Selector**: Styles all elements of a specific type.

css

p {

color: blue;

}

*This makes all <p> (paragraph) elements blue.*

* **Class Selector**: Styles elements with a specific class.

css

.myClass {

font-size: 18px;

}

*Any element with class="myClass" will have font size 18px.*

* **ID Selector**: Styles a unique element with a specific ID.

css

#myId {

background-color: yellow;

}

*The element with id="myId" will have a yellow background.*

**2. What is CSS Specificity?**

CSS specificity decides which style will be applied when multiple rules target the same element.

**Rules of specificity:**

1. **Inline styles** (written inside an HTML tag) have the highest priority.
2. **ID selectors** are more powerful than class selectors.
3. **Class selectors** are stronger than element selectors.
4. If two rules have the same specificity, the last one written in the CSS file is applied.

Example:

css

p {

color: red; /\* Least specific \*/

}

.myClass {

color: green; /\* More specific \*/

}

#myId {

color: blue; /\* Most specific \*/

}

If an element has id="myId" and class="myClass", it will be **blue** because the ID selector is the strongest.

**3. Difference Between Internal, External, and Inline CSS**

| **CSS Type** | **How It Works** | **Pros** | **Cons** |
| --- | --- | --- | --- |
| **Inline CSS** | Written inside an HTML tag using the style attribute. | Quick to apply, good for one-time changes. | Hard to maintain, not reusable. |
| **Internal CSS** | Written inside a <style> tag in the <head> section. | Styles the whole page, no extra files needed. | Not reusable for multiple pages. |
| **External CSS** | Stored in a separate .css file and linked using <link>. | Best for large websites, reusable across pages. | Requires an extra file, needs internet if using online stylesheets. |

Example of **inline CSS**:

html

<p style="color: red;">This is red text.</p>

Example of **internal CSS**:

html

CopyEdit

<style>

p {

color: red;

}

</style>

Example of **external CSS**:

css

p {

color: red;

}

Linked in HTML like this:

html

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

**Conclusion:**

* Use **inline CSS** for quick changes.
* Use **internal CSS** for single-page designs.
* Use **external CSS** for large projects and better organization.

**CSS Box Model**

**1. What is the CSS Box Model?**

The **CSS box model** is a way to understand how elements are sized and spaced on a webpage. Every HTML element is like a box, and this box has four parts:

1. **Content** – The actual text or image inside the box.
2. **Padding** – Space inside the box, between the content and the border.
3. **Border** – The outline around the element.
4. **Margin** – Space outside the border, separating elements.

**How Each Part Affects Size**

* If you increase **padding**, the box gets bigger because the space inside grows.
* If you add a **border**, the box also gets bigger.
* If you increase **margin**, the space between elements increases, but the box itself stays the same size.

Example:

css

.box {

width: 200px;

padding: 20px;

border: 5px solid black;

margin: 10px;

}

**2. Border-Box vs. Content-Box Box-Sizing**

**content-box (Default)**

* Only the **content** has the specified width/height.
* Padding and border **increase** the total size.

css

.box {

width: 200px;

box-sizing: content-box;

padding: 20px;

border: 5px solid black;

}

**border-box**

* The specified width/height **includes** padding and border.
* The content area shrinks to fit inside.

css

.box {

width: 200px;

box-sizing: border-box;

padding: 20px;

border: 5px solid black;

}

✅ **Which is the default?**

* content-box is the default in CSS.
* border-box is often preferred for easier layout control.

css

\* {

box-sizing: border-box;

}

**CSS Flexbox**

**1. What is CSS Flexbox?**

CSS **Flexbox** (Flexible Box) is a layout model that helps arrange elements efficiently, especially when designing responsive websites. It makes it easy to align items, distribute space, and adjust layouts dynamically.

✅ **Why is it useful?**

* It automatically adjusts elements to fit different screen sizes.
* No need for complex float or positioning techniques.
* Elements inside a flex container can easily align and distribute space.

**Flex-Container & Flex-Item**

* **Flex-Container**: The parent element that holds flex items.
* **Flex-Item**: The child elements inside the flex-container.

Example:

css

.container {

display: flex; /\* Makes it a flex container \*/

}

html

<div class="container">

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

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

</div>

**2. Important Flexbox Properties**

1️⃣ **justify-content** – Aligns items **horizontally** (main axis).

* flex-start (default) → Items align to the left.
* flex-end → Items align to the right.
* center → Items align in the center.
* space-between → Items spread out with space in between.
* space-around → Equal space around each item.

Example:

css

.container {

display: flex;

justify-content: center; /\* Centers items horizontally \*/

}

2️⃣ **align-items** – Aligns items **vertically** (cross axis).

* flex-start → Items align at the top.
* flex-end → Items align at the bottom.
* center → Items align in the middle.
* stretch (default) → Items stretch to fill the container.

Example:

css

.container {

display: flex;

align-items: center; /\* Centers items vertically \*/

}

3️⃣ **flex-direction** – Defines the **direction** of items.

* row (default) → Items placed left to right.
* 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: column; /\* Stacks items vertically \*/

}

**Summary**

| **Property** | **Purpose** | **Common Values** |
| --- | --- | --- |
| justify-content | Aligns items horizontally | center, flex-start, flex-end, space-between, space-around |
| align-items | Aligns items vertically | center, flex-start, flex-end, stretch |
| flex-direction | Sets item direction | row, row-reverse, column, column-reverse |

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

**CSS Grid** is a layout system in CSS that allows you to create complex, two-dimensional layouts. It works in both horizontal (rows) and vertical (columns) directions at the same time. With CSS Grid, you can define both rows and columns to create a grid-based structure, which gives you more control over how the items are placed inside the container.

**Flexbox**, on the other hand, is a one-dimensional layout system. It works in only one direction at a time (either horizontally or vertically), allowing you to align and distribute space among items in a container.

**Key Differences:**

* **Grid** is **two-dimensional** (works with both rows and columns).
* **Flexbox** is **one-dimensional** (works either with rows or columns).
* **Grid** is best for more **complex layouts** with multiple rows and columns.
* **Flexbox** is great for simpler, linear layouts or when you only need to align items in a single row or column.

**When to use Grid over Flexbox:**

* **Grid** is better when you have a **complex layout** with many rows and columns (e.g., website layouts, dashboards, etc.).
* **Flexbox** is better when you're aligning items in a **single direction** (like a navbar, a list of items, or buttons).

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

* **grid-template-columns**: This property defines how many columns you want in your grid and their sizes. You can set them using values like pixels (px), percentages (%), or the fr unit (fraction of available space).

**Example**:

.container {

display: grid;

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

}

This creates a grid with three columns: the first and third columns each take 1 fraction of space, and the second column takes 2 fractions.

* **grid-template-rows**: This property defines how many rows you want in your grid and their sizes. It works similarly to grid-template-columns.

**Example**:

.container {

display: grid;

grid-template-rows: 100px 200px;

}

This creates a grid with two rows: the first row is 100px tall, and the second row is 200px tall.

* **grid-gap**: This property adds space between the grid items. You can set both horizontal and vertical spacing between the items.

**Example**:

css

.container {

display: grid;

grid-template-columns: 1fr 1fr;

grid-template-rows: 1fr 1fr;

grid-gap: 20px; /\* 20px space between rows and columns \*/

}

This will create a grid with 2 rows and 2 columns, with 20px of space between each grid item.

In summary:

* **grid-template-columns** and **grid-template-rows** define the structure of your grid by specifying how many columns and rows you want and their sizes.
* **grid-gap** is used to add space between items within the grid

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

**Media queries** in CSS are used to apply different styles to a webpage depending on certain conditions, such as the screen size, resolution, or device orientation. They allow developers to create a responsive design, which means the layout of a webpage adjusts according to the device's screen size (like desktops, tablets, or mobile phones).

**Why are they important for responsive design?**

* **Adaptability:** Media queries make your website look good on all devices, from large desktop monitors to small mobile screens.
* **User Experience:** They ensure that the content and layout are easy to read and interact with, no matter the screen size.
* **Performance:** You can optimize your site’s appearance and functionality for smaller screens, making it faster and more usable on mobile devices.

By using media queries, you can change things like layout, font size, images, and other styles depending on the user's device or screen size.

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

Here's a basic example of a media query that changes the font size for devices with a screen width of **less than 600px** (like mobile phones):

css

@media (max-width: 600px) {

body {

font-size: 14px; /\* Adjusts font size for smaller screens \*/

}

}

**How it works:**

* @media (max-width: 600px) means that the styles inside the curly braces will only apply when the screen width is **600px or smaller**.
* Inside the media query, we set the font-size of the body to **14px**, which is smaller and more suitable for smaller screens.

So, this will make the text easier to read on devices like mobile phones, where smaller text sizes are often needed for better readability.

In summary:

* Media queries allow us to apply different styles depending on the device's characteristics.
* They are crucial for creating responsive websites that adjust smoothly for different screen sizes.

**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 available across all devices and operating systems. These fonts are guaranteed to be displayed correctly because they are pre-installed on almost every device. Examples of web-safe fonts include Arial, Times New Roman, and Courier New.

**Custom web fonts** are fonts that are not standard on devices and need to be loaded from external sources, like Google Fonts or Adobe Fonts. These fonts give more flexibility and creativity to a website's design, allowing you to choose unique typography that fits the brand or style of the website.

**Key Differences:**

* **Web-safe fonts**:
  + Already installed on most devices.
  + Limited options (like Arial, Times, Verdana, etc.).
  + Load faster since they don’t require external files.
* **Custom web fonts**:
  + Not pre-installed on devices.
  + Can be unique, stylish, and tailored to the website's design.
  + Need to be loaded from external sources, which may impact page loading time.

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

* **Faster loading times**: Web-safe fonts don’t require additional resources to load, so they can make your webpage load faster.
* **Reliability**: Since they are pre-installed on most systems, there's no risk of the font failing to load or displaying incorrectly.
* **Compatibility**: Web-safe fonts are more consistent across devices and browsers.

However, if you want a unique look and feel for your website, custom web fonts are often the best choice despite the loading time consideration.

**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 defines the font to be used for text on a webpage. You can specify one or more font names, separated by commas, in order of preference. If the first font isn't available on the user’s device, the next one in the list will be used, and so on.

**Example:**

body {

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

}

In this example:

* If **Arial** is available on the user’s device, it will be used.
* If not, it will try **Helvetica**.
* If neither is available, it will default to **sans-serif** (a generic font family).

**How to apply a custom Google Font to a webpage:**

To use a Google Font, you need to:

1. **Choose a Google Font** from the [Google Fonts website](https://fonts.google.com/).
2. **Link the font** in the <head> section of your HTML file.
3. **Apply the font using the font-family property** in your CSS.

**Steps to apply a Google Font:**

1. **Go to Google Fonts** and choose the font you want (e.g., **Roboto**).
2. **Copy the <link> tag** provided by Google Fonts and add it to the <head> section of your HTML document:

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

1. **Use the font in your CSS** by applying the font-family property:

body {

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

}

**Example:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

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

<title>Custom Google Font</title>

</head>

<body>

<h1>Welcome to My Website!</h1>

<p>This page uses the Roboto font from Google Fonts.</p>

</body>

</html>

In this example:

* The **Roboto** font is applied to the body of the webpage.
* If **Roboto** is unavailable, the browser will fall back to the **sans-serif** font.

In summary:

* **Web-safe fonts** are pre-installed on most devices and are reliable and fast, but they offer limited choices.
* **Custom web fonts** give you more design flexibility but require external resources.
* The **font-family** property in CSS defines the font style, and Google Fonts can be easily integrated by linking the font in HTML and using it in CSS