Question 1: What is a CSS selector? Provide examples of element, class, and ID selectors
answer= A **CSS selector** is a pattern used to target HTML elements so you can apply styles to them.

Examples:

1. Element Selector

Selects all HTML elements of a specific type.

```
p {
  color: blue;
}
```

(This styles all `<p>` tags.)

2. Class Selector

Selects elements with a specific class (uses a dot `.`).

```
.button {
  background-color: green;
}
```

(Styles any element with `class="button"`.)

3. ID Selector

Selects a single element with a specific ID (uses a hash `#`).

```
#header {
  font-size: 24px;
}
```

(Styles the element with `id="header"`.)

Let me know if you want a quick diagram or table summarizing them!

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

answer= CSS specificity is a set of rules used by the browser to determine which CSS style should be applied when multiple selectors target the same HTML element. Each type of selector has a different weight: inline styles have the highest priority, followed by ID selectors, then class and pseudo-class selectors, and finally element selectors, which have

the lowest specificity. When two or more rules conflict, the browser first compares their specificity values; the rule with the higher specificity is applied. If two rules have the same specificity, the browser applies the one that appears later in the stylesheet. Inline styles override all normal CSS rules, and declarations marked with `!important` take top priority, though they should be used sparingly. This system ensures consistent, predictable styling when multiple CSS rules interact.

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

answer= Internal, external, and inline CSS are three ways to apply styles to a webpage, each with its own strengths and limitations. **Inline CSS** is written directly inside an HTML element using the `style` attribute. It has the highest specificity and is useful for quick, single-element styling, but it clutters the HTML and makes large projects hard to maintain. **Internal CSS** is placed inside a `<style>` tag within the `<head>` section of an HTML document. It keeps styles separate from the content and is suitable for styling a single page, but it increases page size and cannot be reused across multiple pages. **External CSS** stores styles in a separate `.css` file linked to the HTML document. This method promotes clean code, easier maintenance, and reusability across many pages, making it ideal for larger websites. However, external stylesheets require additional HTTP requests, which may slightly slow initial page loading. Overall, external CSS is best for large projects, internal CSS works for single-page styling, and inline CSS should be used sparingly for simple, one-time adjustments

Question 1: Explain the CSS box model and its components (content, padding, border, margin). How does each affect the size of an element?

answer= The **CSS box model** describes how every HTML element is structured and how its total size is calculated. It is made of four main components.

(1) Content: This is the actual text or image inside the element, and its width and height form the core size of the box.

(2) Padding: The space between the content and the border; increasing padding makes the element larger because it adds extra space inside the box.

(3) Border: The line surrounding the padding and content; adding border thickness increases the total size of the element.

(4) Margin: The outer space around the element that separates it from other elements; margin does not increase the element's own size but increases the space it occupies on the page. Together, these components determine how much total space an element takes up in the layout.

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

answer= In contrast, **border-box** includes the content, padding, and border all **within the specified width and height**. This prevents the element from growing in size when padding or borders are applied, making layout control easier.

In short:

- **content-box (default):** width = content only → padding + border increase total size
- **border-box:** width = content + padding + border → total size stays fixed

Question 1: Explain the CSS box model and its components (content, padding, border, margin). How does each affect the size of an element?

answer= The **CSS box model** describes how every HTML element is built and how its total space on a webpage is calculated. It consists of **four layers**. The **content** is the innermost area where text or images appear; its width and height form the main size of the element. Surrounding the content is the **padding**, which adds space inside the element and increases its total size because it pushes the border outward. The next layer is the **border**, a visible line around the padding; adding border thickness also increases the overall width and height of the element. The outermost layer is the **margin**, which creates space outside the element, separating it from other elements. While margin does not change the element's internal size, it does affect the amount of space the element occupies on the page. Together, these four parts determine how big an element appears and how it fits within a layout.

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

answer= Any padding and border added to the element increase its total size, meaning the final displayed element becomes larger than the specified width. In contrast, the **border-box** model includes the content, padding, and border all inside the assigned width and height. This means the element's total size stays fixed, even when padding or borders are added, making layouts easier to manage. In summary, **content-box is the default**, and **border-box** helps maintain consistent sizing by keeping padding and borders within the set dimensions.

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

answer= **CSS Flexbox (Flexible Box Layout)** is a modern layout module in CSS that helps you create flexible, responsive, and easily manageable layouts. It allows elements inside a container to automatically adjust their size, space, and alignment without needing complex floats or positioning.

Why Flexbox is Useful

Flexbox makes layout design easier because it:

Distributes space efficiently between items

Aligns elements horizontally or vertically

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

answer= **flex-direction**

This property defines **the direction in which flex-items are placed inside the flex-container**.

Possible values:

row items placed horizontally (left to right) *(default)*

row-reverse items placed right to left

column items placed vertically (top to bottom)

column-reverse items bottom to top

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

answer= **CSS Grid** is a two-dimensional layout system in CSS that allows you to design web pages using rows and columns. Unlike Flexbox, which is primarily **one-dimensional** (handling either a row or a column at a time), Grid can control both horizontal and vertical placement simultaneously. This makes it ideal for creating complex layouts like dashboards, galleries, or page structures with multiple rows and columns.

Differences from Flexbox:

- **Flexbox:** One-dimensional (row or column), best for aligning items in a single direction.
- **Grid:** Two-dimensional (rows and columns), best for overall page layouts and complex

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

answer= **grid-template-columns**: This property specifies the **number and width of columns** in a grid. You can set fixed widths (e.g., `100px`), flexible widths using fractions (`1fr`), or percentages.

grid-template-rows: This defines the **number and height of rows** in the grid. Like columns, you can use fixed units, fractions, or percentages.

grid-gap (or gap): This property adds **spacing between rows and columns**. You can set a single value for both or separate values for rows and columns.

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?

answer= **Web-safe fonts** are a set of fonts that are widely available across most operating systems and browsers, such as Arial, Times New Roman, or Verdana. They are guaranteed to display correctly on nearly all devices without requiring additional downloads. **Custom web fonts**, on the other hand, are fonts that are not pre-installed on user devices and are loaded via services like Google Fonts, Adobe Fonts, or by hosting font files on a server.

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

answer= The **`font-family`** property in CSS is used to specify the typeface for text within an element. It allows developers to define a list of fonts as a fallback chain, so if the first font is unavailable on the user's device, the browser will use the next available font in the list. For example,

`font-family: Arial, Helvetica, sans-serif;` will use Arial if available, otherwise Helvetica, or any default sans-serif font. To apply a **custom Google Font** to a webpage, you first include a `<link>` tag in the HTML `<head>` section that points to the desired font, such as `<link href="https://fonts.googleapis.com/css2?family=Roboto&display=swap" rel="stylesheet">`.

Then, you use the `font-family` property in CSS to apply that font to specific elements, for instance, `body { font-family: 'Roboto', sans-serif; }`.

Including a fallback font like `sans-serif` ensures that the text remains readable even if the custom font fails to load.

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

answer= **Media queries** in CSS are a technique that allows you to apply different styles to a webpage based on the characteristics of the device or viewport, such as screen width, height, resolution, or orientation. They are written using the `@media` rule and enable developers to create **responsive designs** that adapt to various devices, from desktops and tablets to mobile phones.

Media queries are important because they ensure that a website remains **usable, readable, and visually appealing** on different screen sizes. For example, you can adjust font sizes, hide or show elements, or change layout structures depending on whether the user is on a large desktop screen or a small mobile device. Without media queries, a site designed for one screen size may look broken or be difficult to navigate on other devices, leading to poor user experience.

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

answer= `@media (max-width: 600px)` targets devices with a viewport width of **600px or less**.

Inside the block, you can override styles, here reducing the **font size** to make text more readable on small screens.

This helps make webpages **responsive**, ensuring they look good on mobile devices.