MODULE-3_CSS-CSS3

CSS Selectors & Styling

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

A **CSS selector** is a pattern used to select and style specific HTML elements. It tells the browser which elements the CSS rules should apply to.

Examples:

- **Element Selector**: Targets HTML elements by name.

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

Styles all `<p>` elements.

- **Class Selector**: Targets elements with a specific class attribute.

```
css
CopyEdit
.highlight {
  background-color: yellow;
}
```

Styles all elements with `class="highlight"`.

- **ID Selector**: Targets a single element with a specific ID.

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

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

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. It is calculated based on the type of selectors used.

Specificity hierarchy (from lowest to highest):

1. **Element selectors** (e.g., `div`, `p`)
2. **Class selectors**, **attribute selectors**, and **pseudo-classes** (e.g., `.class`, `[type="text"]`, `:hover`)
3. **ID selectors** (e.g., `#id`)
4. **Inline styles** (e.g., `style="color:red"`)
5. **!important** (overrides all other rules, but should be used sparingly)

Example:

```
html
CopyEdit
<p id="text" class="highlight">Hello World</p>
css
CopyEdit
p { color: black; }               /* Specificity: 0-0-1 */
.highlight { color: green; }      /* Specificity: 0-1-0 */
#text { color: red; }             /* Specificity: 1-0-0 */
```

Result: The text will be **red**, because the ID selector has the highest specificity.

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

The three main types of CSS are **inline**, **internal**, and **external**, each with its own use cases, advantages, and disadvantages.

Inline CSS is written directly inside an HTML tag using the `style` attribute. For example, `<h1 style="color:red;">Title</h1>`. It applies styles to a single element only. The main advantage of inline CSS is that it is quick and easy to apply, especially for small changes or testing. However, its biggest drawback is poor maintainability. Since styles are embedded within the HTML, it becomes difficult to manage and reuse, especially on larger websites. It also mixes content and presentation, which is not a good practice.

Internal CSS is written within a `<style>` tag inside the `<head>` section of an HTML document. This method is suitable when styling a single webpage. It keeps the style rules in one place, making it more organized than inline CSS. The advantage of internal CSS is that it does not require an external file and loads with the HTML page. However, it cannot be reused across multiple pages, and as the CSS grows, it can make the HTML file bulky and slower to load.

External CSS involves writing CSS rules in a separate `.css` file and linking it to the HTML file using a `<link>` tag. This is the most efficient and scalable way to manage styles, especially for websites with multiple pages. External CSS promotes code reusability, easier maintenance, and a clear separation between content and style. The main disadvantage is that it requires an extra HTTP request to load the external file, which may slightly affect page load time if the file is large or the network is slow.

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** is a fundamental concept that describes how the size of every HTML element is calculated and displayed on a web page. Each element is considered as a box made up of the following components:

1. **Content**:
   This is the innermost part of the box where text, images, or other elements appear. Its width and height are set using the `width` and `height` properties.
2. **Padding**:
   Padding is the space between the content and the border. It increases the space inside the box but does not affect the border. Padding adds to the total size of the element.
3. **Border**:
   The border wraps around the padding (if any) and content. Its thickness is added to the element's total dimensions.
4. **Margin**:
   Margin is the space outside the element's border. It separates the element from surrounding elements and does not affect the size of the box itself, but rather its position on the page.

Effect on size:
When calculating the total size of an element (under the default `box-sizing: content-box`), the total width is:

```
arduino
CopyEdit
Total width = content width + padding (left + right) + border (left + right)
Total height = content height + padding (top + bottom) + border (top +
bottom)
```

Margins are not included in the box size but affect spacing between elements.

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

The `box-sizing` property in CSS controls how the total width and height of an element are calculated.

1. **content-box** (default):
   In this model, the `width` and `height` apply **only to the content**. Padding and border are added **outside** the specified width and height.
   - Example: `width: 200px; padding: 10px; border: 5px;`
     ➤ Total width = 200 + 10*2* + *5*2 = **230px**
2. **border-box**:
   In this model, the `width` and `height` include **content, padding, and border**. The actual content area shrinks to accommodate padding and border within the defined width.
   - Example: `width: 200px; padding: 10px; border: 5px;`
     ➤ Content width = 200 - 10*2* - *5*2 = **170px**

Which is default?
By default, most browsers use `**content-box**`. However, many developers prefer `border-box` for easier layout control and use it by resetting styles like this:

```
css
CopyEdit
* {
  box-sizing: border-box;
}
```

CSS Grid

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

CSS Grid is a two-dimensional layout system in CSS that allows you to design web layouts using rows and columns. It enables precise control over both the horizontal and vertical placement of elements in a grid-based structure.

Difference between Grid and Flexbox:

Feature	CSS Grid	Flexbox
Layout Direction	Two-dimensional (rows & columns)	One-dimensional (row **or** column)
Use Case	Best for full page or large-scale layouts	Best for aligning content in a row or column
Item Placement	Can place items anywhere in the grid	Flow is based on content order
Gap Support	Has `grid-gap` for row & column gaps	Has `gap` (recently added support)

When to use Grid over Flexbox:

Use **CSS Grid** when:

- You need to design complex layouts with both rows and columns.
- You want to align items across multiple rows and columns.
- You're building full-page layouts or multi-section designs.

Use **Flexbox** when:

- You're aligning items in a single row or column (e.g., navbars, toolbars).
- You need content-based layout adjustment.
- You're working with smaller UI components.

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

These properties define the structure and spacing of the grid layout.

---

1. `grid-template-columns`

Specifies the number and size of columns in the grid.

```
css
CopyEdit
.container {
  display: grid;
  grid-template-columns: 200px 1fr 1fr;
}
```

- Creates 3 columns: the first is 200px wide, the other two share remaining space equally.

---

2. `grid-template-rows`

Defines the number and size of rows in the grid.

```
css
CopyEdit
.container {
  display: grid;
  grid-template-rows: 100px auto 50px;
}
```

- First row: 100px, second row adjusts automatically, third row: 50px.

---

3. `grid-gap` (or `gap`)

Sets spacing between rows and columns.

```
css
CopyEdit
.container {
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-gap: 20px;
}
```

- Adds 20px space between both columns and rows.

Alternatively, you can use:

```
css
CopyEdit
gap: 20px 10px; /* 20px row gap, 10px column gap */
```

Example Layout:

```
css
CopyEdit
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr); /* 3 equal columns */
  grid-template-rows: 100px 200px;
  gap: 15px;
}
```

This creates a 3x2 grid with equal columns, two rows of different heights, and a 15px gap between grid cells.

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 in CSS that provides a more efficient and responsive way to arrange items within a container, especially when the size of the items is unknown or dynamic. It simplifies tasks like centering elements, spacing them evenly, and creating flexible layouts without using floats or positioning.

Flexbox is particularly useful for:

- Creating horizontal or vertical alignment.
- Distributing space between items.
- Adapting layouts to different screen sizes.

Key Terms:

- **Flex Container**:
  This is the parent element that holds flex items. To create a flex container, you set `display: flex` or `display: inline-flex` on the element.

```
css
CopyEdit
.container {
  display: flex;
}
```

- **Flex Items**:
  These are the direct child elements of a flex container. Flex properties applied to the container affect the layout of these items.

```
html
CopyEdit
<div class="container">
  <div class="item">1</div>
  <div class="item">2</div>
</div>
```

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

These properties are applied to the **flex container** and control the alignment and direction of flex items:

1. **`flex-direction`**:
   Defines the **main axis** (the direction in which flex items are placed).
   - `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.

```
css
CopyEdit
.container {
  flex-direction: row;
}
```

2. **`justify-content`**:
   Aligns items **along the main axis** (horizontal by default).
   - `flex-start`: items align to the left/top.
   - `flex-end`: items align to the right/bottom.
   - `center`: items are centered.
   - `space-between`: items spread out with space between.
   - `space-around`: equal space around items.
   - `space-evenly`: equal space between and around items.

```
css
CopyEdit
.container {
  justify-content: space-between;
}
```

3. **`align-items`**:
   Aligns items **along the cross axis** (perpendicular to the main axis).
   - `flex-start`: align to the top/left of the container.
   - `flex-end`: align to the bottom/right.
   - `center`: items centered.
   - `stretch` (default): items stretch to fill the container.
   - `baseline`: items align by their text baseline.

```
css
CopyEdit
.container {
  align-items: center;
}
```

Responsive Web Design with Media Queries

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

Media queries are a feature in CSS that allow you to apply styles based on the characteristics of the device or screen, such as its width, height, resolution, or orientation. They are essential for creating **responsive web designs**, which adapt to different screen sizes and devices — from mobile phones to tablets and desktops.

With media queries, developers can write conditional CSS rules that only apply when certain conditions are met. This ensures that a website remains readable, user-friendly, and visually appealing across all devices without requiring separate versions for each screen size. Media queries help enhance usability, improve accessibility, and ensure a consistent experience for users on any device.

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

```
css
CopyEdit
@media screen and (max-width: 600px) {
  body {
    font-size: 14px;
  }
}
```

Explanation:

- `@media screen and (max-width: 600px)`: Targets screens that are **600 pixels wide or smaller**.
- Inside the block, we reduce the `font-size` of the `body` to make the text more suitable for small devices.

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 a set of fonts that are pre-installed on most operating systems and devices, such as **Arial, Times New Roman, Courier New, Georgia, and Verdana**. Since these fonts are already available on users' systems, they do not need to be downloaded, which ensures fast loading and consistent appearance across different browsers and platforms.

Custom web fonts, on the other hand, are fonts that are not necessarily installed on the user's device and are typically loaded from the web, such as from **Google Fonts or Adobe Fonts**. These fonts offer greater variety and branding flexibility, allowing designers to use unique typography that matches the style of the website.

You might choose a **web-safe font** over a custom font to ensure:

- **Faster load times** (no need to download fonts),
- **Better performance** on slow networks,
- **Compatibility** across all browsers and devices, even with limited internet access,
- **Accessibility** for users with browser font restrictions or limited data usage.

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 specifies the font to be used for the text content of an element. It can include multiple fonts as a fallback list, where the browser will use the first available font from the list.

Example of `font-family`:

```
css
CopyEdit
body {
  font-family: "Arial", "Helvetica", sans-serif;
}
```

This will use **Arial**, and if it's not available, fall back to **Helvetica**, and finally to any available **sans-serif** font.

To apply a custom Google Font:

1. **Import the font in your HTML `<head>`:**

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

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

```
css
CopyEdit
body {
  font-family: 'Roboto', sans-serif;
}
```