Name: Aayushi Goria

Module 16) Wd - Css And Css 3

Q. 1: What is a CSS selector?

A **CSS selector** is a pattern used to select and style HTML elements. It determines which elements a set of CSS rules will apply to.

Common Types of CSS Selectors:

1. **Universal Selector (*)** – Selects all elements.

```
* {
    margin: 0;
    padding: 0;
}
```

2. **Element Selector** – Selects all elements of a specific type.css

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

3. **Class Selector (`.`)** – Selects elements with a specific class.

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

4. **ID Selector (`#`)** – Selects an element with a specific ID.

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

5. **Group Selector (`,`)** – Selects multiple elements at once.

```
h1, h2, h3 {
    font-family: Arial, sans-serif;
}
```

6. **Child Selector (`>`)** – Selects direct children of an element.

```
div > p {
    color: green;
}
```

7. **Descendant Selector (space)** – Selects all nested elements inside another.

```
div p {
    font-size: 18px;
}
```

8. **Adjacent Sibling Selector (`+`)** – Selects the next immediate sibling.

```
h1 + p {
    color: gray;
}
```

9. **General Sibling Selector (`~`)** – Selects all siblings after an element.

```
h1 ~ p {
    font-style: italic;
}
```

10. **Attribute Selector** – Selects elements based on attributes.

```
input[type="text"] {
    border: 1px solid black;
}
```

Q 2: Explain the concept of CSS specificity.

CSS **specificity** determines which style rule is applied when multiple rules target the same element. It follows a ranking system based on the type of selector used.

Specificity Calculation

CSS assigns a "weight" to different types of selectors. The general rule is:

1. Inline styles (1000) → Highest priority
2. IDs (100)
3. Classes, attributes, and pseudo-classes (10)
4. Elements and pseudo-elements (1)
5. Universal selector (*) and inherited styles (0) → Lowest priority

Q 3: Difference Between Internal, External, and Inline CSS

CSS Type	Location	Usage Example	Pros	Cons
Inline CSS	Inside an HTML tag	Text	Quick & overrides other styles	Hard to maintain, not reusable
Internal CSS	Inside <style> in <head>	<style> p { color: blue; } </style>	Works for single pages, no extra file needed	Increases HTML size, not reusable
External CSS	In a separate .css file	<link rel="stylesheet" href="style.css">	Best for large projects, reusable, clean HTML	Extra HTTP request, won’t work if the file is missing

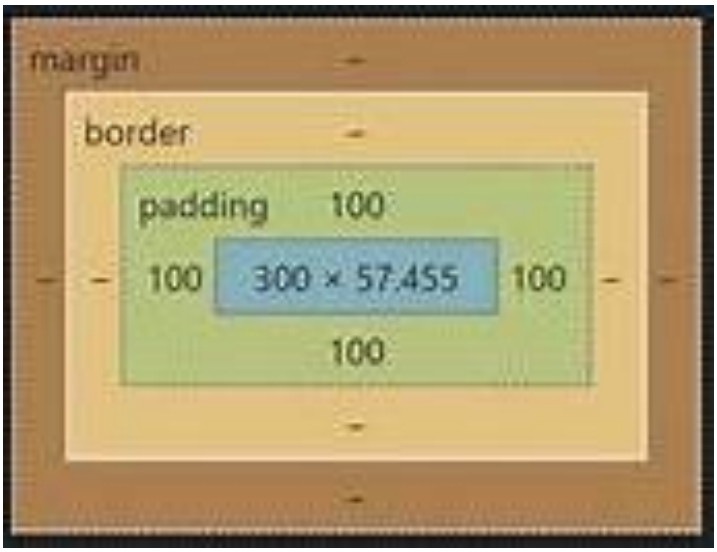
CSS Box Model

Q 1: CSS Box Model and Its Components

The CSS Box Model describes how elements are structured and sized in a webpage. Every HTML element is treated as a rectangular box consisting of the following layers:

1. Content – The actual content inside the element (text, images, etc.).
2. Padding – The space between the content and the border.
3. Border – The boundary that wraps around the padding and content.
4. Margin – The space outside the border that separates the element from others.

Visual Representation:



Q 2: Difference Between border-box and content-box Box Sizing

The box-sizing property determines how the total width and height of an element are calculated.

Box Sizing Type	How Width & Height Are Calculated	Example (width: 200px; padding: 10px; border: 5px;)	Total Width
content-box *(Default)*	Width & height apply only to content. Padding and border are added separately.	200px (content) + 20px (padding) + 10px (border)	230px
border-box	Width & height include content, padding, and border. The content shrinks to fit.	200px (total including padding & border)	200px

Example Code

```
.content-box {
    width: 200px;
    padding: 10px;
    border: 5px solid black;
    box-sizing: content-box;
}
.border-box {
    width: 200px;
    padding: 10px;
    border: 5px solid black;
    box-sizing: border-box;
}
```

CSS Flexbox

Q 1: What is CSS Flexbox, and How is it Useful for Layout Design?

CSS Flexbox (Flexible Box Layout) is a layout model designed to make it easier to align and distribute space among items in a container, even when their sizes are unknown or dynamic. It is particularly useful for creating responsive layouts, centering elements, and managing spacing efficiently.

Key Components of Flexbox:

1. Flex Container ○ The parent element that holds the flex items.
   - Defined using display: flex; or display: inline-flex;.
   - Controls how child elements are positioned.

Example: .container { display: flex;

background-color: lightgray;

}

2. Flex Items
   - The child elements inside the flex container.
   - These items respond to flexbox properties like flexgrow, flex-shrink, and flex-basis.

Example:

.item {

flex: 1; /* Makes items flexible */

padding: 10px;

background-color: lightblue; border: 1px solid

blue;

}

Q 2: Describe justify-content, align-items, and flexdirection in Flexbox

1. justify-content (Horizontal Alignment)

Controls how flex items are aligned along the main axis (left to right for row, top to bottom for column).

Common Values:

- flex-start → Items align at the start (default).
- flex-end → Items align at the end.
- center → Items are centered.
- space-between → Items are spaced with no gaps at the ends.
- space-around → Equal space around each item.
- space-evenly → Equal space between and around items.

Example:

.container { display: flex;

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

}

2. align-items (Vertical Alignment)

Controls how flex items align along the cross axis (top to bottom for row, left to right for column).

Common Values:

- stretch → Items stretch to fill the container height (default).
- flex-start → Items align at the top.
- flex-end → Items align at the bottom.
- center → Items are centered.
- baseline → Aligns items based on text baselines.

Example:

.container { display:

flex;

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

}

3. flex-direction (Main Axis Direction)

Defines whether items are arranged horizontally (row) or vertically (column).

Common Values:

- row → Items placed left to right (default).
- 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 */
}
```

Key Takeaways:

- justify-content → Controls horizontal alignment.
- align-items → Controls vertical alignment.
- flex-direction → Sets layout direction (row/column).

CSS Grid

Q 1: What is CSS Grid, and How Does It Differ from Flexbox?

CSS Grid is a two-dimensional layout system that allows for precise placement of elements along both rows and columns. Unlike Flexbox, which is a one-dimensional layout system (either row or column), Grid provides greater control over complex layouts.

Differences Between Grid and Flexbox:

Feature	CSS Grid	Flexbox
Layout Type	Two-dimensional (rows & columns)	One-dimensional (row or column)
Main Use	Complex page layouts (grids, dashboards)	Aligning and distributing items (navigation bars, buttons, etc.)
Feature	CSS Grid	Flexbox
Alignment Control	Precise control over rows & columns	Focuses on alignment along one axis
Example Use Cases	Web page layouts, image galleries, dashboards	Navbars, buttons, cards, simple components

Question 2: grid-template-columns, grid-template-rows, and grid-gap

CSS Grid Properties

Property	Description	Example
grid-template-columns	Defines the number & size of columns in a grid.	grid-template-columns: 100px 200px 1fr; → 3 columns (fixed & flexible)
grid-template-rows	Defines the number & size of rows in a grid.	grid-template-rows: 50px auto 100px; → 3 rows (fixed & flexible)
grid-gap *(shorthand for row-gap & column-gap)*	Defines the space between grid items.	grid-gap: 10px; → 10px space between rows & columns

Example CSS Grid

```
.container {
   display: grid;
   grid-template-columns: 1fr 2fr 1fr;
   grid-template-rows: 100px 200px;
   grid-gap: 10px;
}
```

Responsive Web Design with Media Queries

Q 1: What Are Media Queries in CSS, and Why Are They Important for Responsive Design?

Media queries are a feature in CSS that allow styles to be applied conditionally based on a device's screen size, resolution, or other characteristics. They enable responsive design, ensuring websites look good on all devices (desktops, tablets, mobiles).

Why Media Queries Are Important:

Create flexible layouts that adapt to different screen sizes. Enhance user experience by optimizing design for different devices.

Reduce the need for separate mobile & desktop websites.

Improve accessibility by adjusting font sizes, spacing, and layouts dynamically.

Q 2: Basic Media Query for Screens Smaller Than 600px

The following media query reduces the font size when the screen width is 600px or smaller:

```
@media (max-width: 600px) {

body {

font-size: 14px;

}

}
```

Explanation:

- @media (max-width: 600px): Applies styles only when the screen width is 600px or less.
- body { font-size: 14px; }: Changes the default font size for better readability on small screens.

Typography and Web Fonts

Question 1: Difference Between Web-Safe Fonts and Custom Web Fonts

1. Web-Safe Fonts

Web-safe fonts are pre-installed on most operating systems (Windows, macOS, Linux), ensuring that they display consistently across different devices and browsers.

Examples of Web-Safe Fonts:

- Arial
- Times New Roman
- Verdana
- Georgia
- Courier New Advantages:
- Loads faster since no external files are required.
- Ensures consistent appearance across all devices.

Disadvantages:

- Limited choices, making designs less unique.

2. Custom Web Fonts

Custom fonts (e.g., Google Fonts, Adobe Fonts) are not preinstalled on devices and must be downloaded from an external source before rendering.

Examples of Custom Web Fonts:

- Roboto (Google Fonts)
- Open Sans (Google Fonts)
- Lora (Adobe Fonts)

Advantages:

- More design flexibility with unique typography.
- Brand consistency across platforms.

Disadvantages:

- Slightly slower loading times due to external requests.
- Fallback fonts needed in case the custom font fails to load.

When to Use Web-Safe Fonts Over Custom Fonts?

- When performance and fast loading times are critical.
- When designing for email templates (since custom fonts may not render in all email clients).
- When font consistency across all devices is a priority.

Question 2: What is the font-family Property in CSS?

The font-family property specifies which font should be used for text in an element. It allows specifying multiple fonts as fallbacks in case the preferred font is unavailable.

Example:

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

- If Arial is available, it will be used.
- If not, Helvetica will be used.
- If neither is available, a generic sans-serif font will be displayed.

To Apply a Custom Google Font to a Webpage

Step 1: Import the Font in the <head>

Add the following <link> tag inside the HTML <head> section: html

```
<link
href="https://fonts.googleapis.com/css2?family=Roboto:wgh
t@400;700&display=swap" rel="stylesheet"> Step 2: Use the Font in CSS css
```

CopyEdit body {

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

}

Alternative: Use @import in CSS

css

CopyEdit

@import
url('https://fonts.googleapis.com/css2?family=Roboto:wght@
400;700&display=swap');

body {

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

}