Module 3 – Frontend – CSS and 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 HTML elements. It tells the browser which HTML elements you want to apply styles to.

Types of CSS Selectors with Examples:

1. Element Selector

- Selects HTML elements by tag name.

Example:

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

- **Meaning:** All <p> (paragraph) elements will have blue text.

2. Class Selector

- Selects elements with a specific class attribute.
- Syntax: . followed by class name.

Example:

CSS

```
.box {
  background-color: yellow;
}
```

HTML:

```
<div class="box">Hello</div>
```

- **Meaning:** All elements with class="box" will have a yellow background

3.ID Selector

- Selects an element with a specific ID.
- Syntax: # followed by ID name.

Example:

CSS:

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

HTML:

```
<h1 id="header">Welcome</h1>
```

- **Meaning:** The element with id="header" will have a font size of 24px.

4. Group Selector

- Use When: You want to apply the same style to multiple elements.

Example:

```
h1, h2, p {
  margin-bottom: 10px;
}
```

- All selected elements will get 10px bottom margin.

5. Descendant Selector

- Use When: You want to style elements inside another element.

  Example:

  ```
  nav a {
    text-decoration: none;
  }
  ```

- All <a> tags **inside <nav>** will have no underline.

6. Child Selector

- Use When: You want to style only direct children.

  Example:

```
ul > li {
  list-style: square;
}
```

- Only immediate <li> children of <ul> are selected.

7. Universal Selector

- Use When: You want to reset styles or apply styles to all elements.

  Example:

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
```

- Often used in CSS resets or base styles.

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

What is CSS Specificity?

- CSS specificity decides which style wins when more than one rule applies to the same HTML element.

Think of it like a score system.

The browser gives each CSS rule a specificity score — higher scores win.

- **Specificity Levels (From Lowest to Highest)**

Rule Type	Example	Importance
Tag Name	p, div	Low
Class	.box, .title	Medium
ID	#header	High
Inline Style	style="color:red"	Very High
!important	color:red !important	Always wins

Example:

Hello

p { color: red; } /* tag → low */

.text { color: green; } /* class → medium */

#main { color: orange; } /* ID → high */

Which color shows? → orange (because #main has the highest specificity).

What if there's !important?

Example:

.text { color: purple !important; }

Now purple wins, no matter what. !important overrides all other rules.

- **Summary**

- More specific rule = higher priority
- If two rules are equal, the last one written wins
- Use !important only when absolutely needed

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

- There are 3 main ways to apply CSS to HTML. Let's break them down simply:

---

1. Inline CSS

What is it?

CSS written directly inside an HTML tag using the style attribute.

Example:

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

Advantages:

- Quick and easy for small changes
- Useful for testing/debugging

Disadvantages:

- Hard to maintain
- Not reusable
- Clutters HTML
- Lower performance on big sites

---

2. Internal CSS

What is it?

CSS written inside the <style> tag in the <head> section of an HTML file.

Example:

```
<head>
  <style>
    p {
      color: green;
    }
  </style>
</head>
```

Advantages:

- Good for single-page styling
- Keeps CSS in one place (for that page)

Disadvantages:

- Not reusable for other pages
- Slows down multi-page websites

---

3. External CSS

- **What is it?**

CSS written in a separate file (e.g., style.css) and linked in the HTML.

```
<head>
  <link rel="stylesheet" href="style.css" />
</head>
```

Advantages:

- Reusable across many pages
- Cleaner HTML
- Faster page loads (browser can cache CSS file)
- Best practice for large projects

Disadvantages:

- Extra HTTP request (but minimal with caching)
- Doesn't work if the CSS file is missing or not linked properly

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 how every HTML element is structured and sized in a web page. It defines how space is calculated around elements.

1. Content

- The actual text, image, or other content inside the element.
- Width/height is applied to this area.

width: 200px;

height: 100px;

---

2. Padding

- Space between the content and the border.
- Adds extra space inside the box, increasing total size.

padding: 20px;

---

3. Border

- The line surrounding the padding and content.
- You can set width, color, and style.

border: 2px solid black;

---

4. Margin

- Space outside the border, separating elements.
- Does not affect the element's box size, but affects layout.

margin: 15px;

How Box Size is Calculated (by default)

By default, the total size of an element is:

Total Width = content + padding (left + right) + border (left + right)

Total Height = content + padding (top + bottom) + border (top + bottom)

Margins are not included in the box size but add space around the box.

Tip: Use box-sizing: border-box

box-sizing: border-box;

- This includes padding and border inside the width/height.
- Makes layouts easier to manage.

Example:

```
div {
  width: 200px;
  padding: 10px;
  border: 2px solid black;
  margin: 20px;
}
```

- Content Width: 200px
- Total Width: 200 + 10*2 + 2*2 = 224px
- Margin: Adds extra 20px space outside

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

- box-sizing in CSS controls how the total size of an element is calculated — whether borders and padding are included in the element's width/height.

---

1. content-box (Default)

- Only the content area is included in the width and height.
- Padding and border are added outside of that size.

Example:

CSS

box-sizing: content-box;

width: 200px;

padding: 20px;

border: 5px solid;

Total width = 200 + 20*2 + 5*2 = 250px

---

2. border-box

- The padding and border are included inside the width and height.
- The content shrinks to make space for padding and border.

Example:

CSS

box-sizing: border-box;

width: 200px;

padding: 20px;

border: 5px solid;

Total width = 200px exactly (content shrinks inside)

Default:

CSS

box-sizing: content-box;

- This is the default value in CSS.

Common Practice:

Developers usually reset it to border-box like this:

CSS

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

- Because it’s easier to manage layout without unexpected overflow.

CSS Flexbox

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

- Flexbox is a CSS layout method used to arrange items in a row or column. It makes it easier to space items evenly, center them, and make them responsive.

Two Main Parts:

1. Flex Container

- This is the parent element. You make it a flex container using:

display: flex;

2. Flex Items

- These are the children inside the container. They follow the flex rules.

Why Use Flexbox?

It helps you:

- Align items easily (center, left, right)
- Make items flexible in size
- Build responsive layouts without floats or positioning

Example:

HTML:

```
<div class="box">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
</div>
```

CSS:

```
.box {
  display: flex;
}
```

This will put "One", "Two", and "Three" in a row.

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

1. flex-direction

➤ **What it does:**

- It sets how the flex items are placed inside the flex container — in a row or column.

Think of it like:

Are the boxes arranged left-to-right or top-to-bottom?

Syntax & Meaning:

```
flex-direction: row;         /* default: horizontal left to right */
flex-direction: row-reverse;  /* horizontal right to left */
flex-direction: column;       /* vertical top to bottom */
flex-direction: column-reverse; /* vertical bottom to top */
```

Example:

```
.flex-container {
  display: flex;
  flex-direction: column;
}
```

- This stacks the items vertically.

---

2. justify-content

➤ **What it does:**

- It controls how items are aligned along the main axis (the direction set by flex-direction).

Works on:

- Horizontal axis if flex-direction: row
- Vertical axis if flex-direction: column

Values:

Value	Description
flex-start	Items are placed at the start (left/top)
flex-end	Items are placed at the end (right/bottom)
Center	Items are centered along the main axis
space-between	Equal space between items
space-around	Equal space around items
space-evenly	Equal space between and around items

Example:

```
.flex-container {
  display: flex;
  justify-content: space-between;
}
```

- This spreads items with equal space between them.

---

3. align-items

➤ What it does:

- It aligns items along the cross axis (perpendicular to flex-direction).

Works on:

- Vertical axis if flex-direction: row
- Horizontal axis if flex-direction: column

Values:

Value	Description

stretch	(Default) Items stretch to fill container
flex-start	Items align at the start of cross axis
flex-end	Items align at the end of cross axis
center	Items align at center of cross axis
baseline	Items align by their text baseline

Example:

```
.flex-container {
  display: flex;
  align-items: center;
}
```

- This centers the items vertically (when using row direction).

Summary Table:

Property	Controls alignment on	Works along axis
flex-direction	Item layout direction	N/A
justify-content	Space between items	Main axis
align-items	Position along cross axis	Cross axis

CSS Grid

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 design web pages using rows and columns.

It creates a two-dimensional layout, which means you can control both:

- the rows (vertical) and
- the columns (horizontal) at the same time.

Example:

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}
```

- This creates a 3-column layout where each column takes equal space.

How is Grid different from Flexbox?

Feature	Flexbox	Grid
Layout Direction	One-dimensional (row OR column)	Two-dimensional (row AND column)
Control	Better for content flow	Better for layout structure
Item alignment	In a line (like buttons or nav)	In a full layout (like a web page)
Placement	Items go one after another	Items can be placed anywhere (even in gaps)

When to use Grid?

Use CSS Grid when:

- You’re building the overall page layout (like header, sidebar, main, footer).
- You want to place items in rows and columns.
- You need precise control over where items appear.

Example Use Cases:

- Full website structure
- Product listing grid
- Gallery layout

When to use Flexbox?

Use Flexbox when:

- You're laying out items in a single row or column.
- You want items to adjust based on content size.
- You need simple alignment like center, space-between, etc.

Example Use Cases:

- Navbars
- Toolbars
- Buttons in a row
- Cards in one row

Summary:

Question	**Answer**
What is CSS Grid?	A layout system using rows & columns
How is it different from Flexbox?	Grid = 2D, Flexbox = 1D
When to use Grid?	For full page or grid-based layouts
When to use Flexbox?	For rows/columns of items with flexible sizing

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

1. grid-template-columns

- This property defines how many columns you want in your grid and how wide each column should be.

Syntax:

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

What it means:

- 1st column: 200px wide (fixed)
- 2nd column: 1fr → takes 1 part of remaining space
- 3rd column: 2fr → takes 2 parts (twice as much as 2nd column)

Example:

```
.container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
}
```

- This creates 3 columns, each 100px wide.

---

2. grid-template-rows

- This defines the height of rows in the grid.

Syntax:

```
.container {
  display: grid;
  grid-template-rows: 100px auto 1fr;
}
```

What it means:

- 1st row: 100px high
- 2nd row: auto → adjusts to content
- 3rd row: 1fr → takes remaining space

---

3. grid-gap (or gap)

- Used to add space between rows and columns.

Note: In modern CSS, we use gap instead of grid-gap, but both work.

Syntax:

```
.container {
  display: grid;
  grid-gap: 20px;
}
```

- Adds 20px gap both between rows and columns.

You can also define row and column gaps separately:

```
.container {
  display: grid;
  row-gap: 10px;
  column-gap: 20px;
}
```

Full Example:

```
<div class="container">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>
```

css

```
.container {
```

```
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;
  grid-template-rows: 100px 150px;
  gap: 15px;
}
```

This creates:

- 3 columns with proportional widths
- 2 rows with set heights
- 15px spacing between all grid items

Summary:

Property	**Use**
grid-template-columns	Defines the number and width of columns
grid-template-rows	Defines the number and height of rows
grid-gap / gap	Adds space between rows and columns

Responsive Web Design with Media Queries

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

- Media queries in CSS are a feature that allows you to apply different styles to a webpage based on the characteristics of the user's device, such as screen width, height, resolution, orientation, and more.
- They are written using the @media rule and typically target breakpoints to adjust the layout and content for different screen sizes (e.g., mobile, tablet, desktop).

Example:

```
/* Default styles for mobile */
body {
  font-size: 16px;
}

/* Styles for devices wider than 768px (like tablets and desktops) */
@media (min-width: 768px) {
  body {
    font-size: 20px;
  }
}
```

Why Media Queries are Important for Responsive Design:

1. Device Adaptability: They ensure your website looks good and works well on all devices—mobile, tablet, laptop, desktop.
2. Improved User Experience: Adjusting layout and font sizes for different screen sizes makes content easier to read and navigate.
3. Performance Optimization: You can hide or show content depending on the device, improving performance on smaller or slower devices.

4. Maintainability: With breakpoints, you can manage style changes cleanly instead of writing completely different stylesheets for every device.

Summary:

- Media queries are essential for responsive design, which is the practice of creating web pages that adapt gracefully to various screen sizes and resolutions. They help make websites flexible, user-friendly, and future-ready.

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

- Here's a simple media query that decreases the font size for screens narrower than 600 pixels (like smartphones):

```
/* Default font size for larger screens */
body {
  font-size: 18px;
}

/* Media query for screens smaller than 600px */
@media (max-width: 600px) {
  body {
    font-size: 14px;
  }
}
```

Explanation:

- @media (max-width: 600px) means: “Apply the following styles only if the screen width is 600px or less.”
- This helps make text more readable on smaller devices by adjusting the font size appropriately.
- This is a common technique used in responsive web design to ensure usability across different screen sizes.

Typography and Web Fonts

Question 1: Explain the difference between web-safe fonts and custom web fonts. Whymight you use a web-safe font over a custom font?

- **Web-Safe Fonts vs. Custom Web Fonts**

Feature	Web-Safe Fonts	Custom Web Fonts
Definition	Fonts that are commonly pre-installed on most operating systems.	Fonts that are not pre-installed and must be loaded from the web.
Examples	Arial, Times New Roman, Courier New, Georgia, Verdana	Google Fonts like Roboto, Open Sans, Lato, or custom fonts via @font-face
Loading	No need to download – they load instantly	Must be downloaded from a server (e.g., Google Fonts or self-hosted)
Performance	Very fast – no delay in loading	Can slow down page load time if not optimized
Design Flexibility	Limited variety	Wide range of styles and weights for better branding
Browser Support	Supported everywhere by default	Require fallback and may not look consistent across all browsers/devices

---

Why Use a Web-Safe Font Over a Custom Font?

You might choose a web-safe font when:

- Performance is critical (e.g., low-bandwidth or mobile-first designs)
- Device compatibility must be guaranteed
- You want to avoid font loading issues or delays
- Privacy or security concerns prevent using external font services

---

Summary:

- Web-safe fonts = fast, reliable, but limited in style.
- Custom web fonts = more design control, but can affect performance.

➢ Use web-safe fonts for speed and reliability.
Use custom fonts when branding and visual appeal are a priority.

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

What is the font-family Property?

➢ The font-family property in CSS is used to specify the typeface (font) for text elements on a webpage.

You can list multiple fonts as a fallback system:

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

- The browser will use the first available font in the list.
- Generic font families (like sans-serif, serif, or monospace) act as fallbacks**.**

How to Apply a Custom Google Font to a Webpage

Step 1: Link the Google Font in HTML <head>

Example using the Google Font Roboto:

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

Step 2: Apply the Font Using font-family in CSS

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

Full Example:

```
<!DOCTYPE html>
<html>
<head>
  <link href="https://fonts.googleapis.com/css2?family=Roboto&display=swap"
rel="stylesheet">
  <style>
    body {
      font-family: 'Roboto', sans-serif;
    }
  </style>
</head>
<body>
  <h1>Hello, Google Fonts!</h1>
</body>
</html>
```

Summary:

- The font-family property defines which font to use.
- To use a Google Font:
    1. Import it via a <link> tag in HTML.
    2. Apply it with font-family in your CSS.
- Always include a fallback font for better compatibility.