Module 3 – Frontend – css and css3

css selectors & styling

1)What is a css selector? provide examples of element, class, and id selectors.

- **types of css selectors**

1. element selector

targets html elements by their tag name.

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

2**. class selector**

- targets elements with a specific class attribute. classes are reusable.
- syntax: use a period (.) followed by the class name.

```
.highlight {
  background-color: yellow;
}
<div class="highlight">important note</div>
```

3. id selector

- targets a single, unique element by its id attribute.
- syntax: use a hash (#) followed by the id name.

```
#header {
  font-size: 24px;
}
<h1 id="header">welcome</h1>
```

2) explain the concept of css specificity. how do conflicts between multiple stylesget resolved?

selector type	specificity value
universal selector (*)	0,0,0,0
element selector (p, h1)	0,0,0,1
class selector (.box)	0,0,1,0

selector type	specificity value
attribute selector ([type="text"])	0,0,1,0
pseudo-classes (:hover, :first-child)	0,0,1,0
id selector (#header)	0,1,0,0
inline styles (style="...")	1,0,0,0
!important	overrides all (but should be avoided unless necessary)

- how specificity is calculated:

```
h1 {
  color: blue;       /* specificity: 0,0,0,1 */
}

.title {
  color: red;        /* specificity: 0,0,1,0 */
}

#main-title {
  color: green;      /* specificity: 0,1,0,0 */
}
<h1 id="main-title" class="title">hello</h1>
```

- what if specificities match?

  if two rules have equal specificity**,** the one that appears last in the css will win (this is called the cascade**).**

```
.title {
  color: red !important;
}
```

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 (Cascading Style Sheets) are **inline**, **internal**, and **external**. Each method has its specific use cases, advantages, and disadvantages:

1. Inline CSS

CSS is applied **directly inside the HTML element** using the style attribute.

```
<p style="color: red; font-size: 18px;">This is an inline styled paragraph.</p>
```

Advantages:

- Easy and quick for **testing** or **small changes**.
- Overrides both internal and external styles due to higher **specificity**.
- Does not require a separate CSS file.

Disadvantages:

- **Hard to maintain**: Style is mixed with HTML, violating separation of concerns.
- Not reusable: You have to repeat the same style in multiple elements.
- Increases **HTML size** and decreases readability.
- Not suitable for larger or responsive designs.

---

2. Internal CSS

CSS is written inside a <style> tag in the <head> section of the HTML document.

```
<head>
 <style>
  p {
   color: blue;
   font-size: 20px;
  }
 </style>
</head>
```

Advantages:

- Keeps CSS in one place within the document.
- Good for **single-page websites** or **small projects**.
- Allows the use of selectors, pseudo-classes, etc.

Disadvantages:

- Styles are **not reusable** across multiple HTML files.
- Increases the size of the HTML file.
- Less efficient for websites with many pages.

3. External CSS

CSS is written in a **separate .css file**, which is then linked to the HTML using <link>.

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

Advantages:

- **Best for large projects** or multi-page websites.
- Promotes separation of content and style (cleaner code).
- Styles can be reused across multiple pages.
- Improves **load time** through browser caching.

Disadvantages:

- Requires an extra HTTP request to fetch the .css file (can affect load speed on first visit).
- No styles will show if the CSS file is **missing or not loaded**.
- More complex setup compared to inline or internal.

CSS Box Model

(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 in web design and layout. It describes how every HTML element is structured and how its dimensions are calculated.

Each element is essentially a box that consists of the following four layers (from inside out):

1. Content

- This is the **innermost part** of the box.
- It contains **text, images, or other elements**.
- The size is controlled by properties like width and height.

Example:-

Css

width: 200px;

height: 100px;

2. Padding

- The **space between the content and the border**.
- It adds spacing **inside the element**, but around the content.
- It is **transparent**.

Example:-

Css

padding: 10px;

Adds 10px of space **inside the border**, around the content.

3. Border

- The **edge** around the padding and content.
- It has **thickness, style, and color**.

Example:

Css

border: 2px solid black;

Adds a 2px black border around the padding.

4. Margin

- The **outermost layer**.
- It adds **space between the element and other elements**.
- Also **transparent**.

Example:

margin: 20px;

Adds 20px of space **outside the border**, separating it from other elements.

How It Affects Element Size

By default, the **total size** of an element is:

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

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

Example Calculation:-

width: 200px;

padding: 10px;

border: 2px solid;

margin: 20px;

- **Total width** = 200 (content) + 20 (padding) + 4 (border) + 40 (margin) = **264px**

- **Optional: box-sizing Property**

To make sizing easier, use:

box-sizing: border-box;

This makes the width and height **include padding and border**, so total size won't increase unexpectedly.

(2)What is the difference between border-box and content-box box-sizing inCSS? Which is the default?

The box-sizing property in CSS controls how the total width and height of an element is calculated — i.e., whether padding and border are included in the width/height or not.

1. content-box (🔧 Default)

- Only the **content** is included in the width and height.
- Padding and border are **added** outside the specified width/height.

- **Example:**
- **Css:-**

box-sizing: content-box; /* default */

width: 200px;

padding: 10px;

border: 2px solid;

- ⇨ **Total width** = 200 (content) + 10+10 (padding) + 2+2 (border) = **224px**
- ⇨ **Total height** = height + padding + border

2. border-box

- The **content + padding + border** are all included **within** the specified width and height.
- Makes it easier to size elements predictably.

- **Example:**
- **Css:**

box-sizing: border-box;

width: 200px;

padding: 10px;

border: 2px solid;

- ⇨ Total width = 200px only, including content, padding, and border
- ⇨ So the actual content area is reduced to fit everything inside 200px.

CSS Flexbox

(1)CSS Flexbox (Flexible Box Layout) is a modern CSS layout model that provides an efficient way to arrange elements in a container, even when their size is unknown or dynamic.

It allows you to:

- Align elements horizontally or vertically
- Distribute space evenly or proportionally
- Handle responsive design without complex float or positioning rules.

Key Benefits of Flexbox:

- Easier alignment and spacing of items
- Supports dynamic and responsive layouts
- Works well for one-dimensional layouts (row or column)
- Avoids the need for floats or manual width calculations.

Important Terms:

1. Flex Container

- The **parent element** that holds flex items.
- Declared using:

Example:-

Css:-

display: flex;

It **enables Flexbox behavior** for its children.

HTML

```
<div class="flex-container">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
</div>
```

CSS:-

```
<style>
.flex-container {
  display: flex;
```

}

</style>

2. Flex Items

- The **child elements** inside a flex container.
- They become **flexible** and respond to Flexbox rules like:
    - flex-grow
    - flex-shrink
    - flex-basis
    - order
    - align-self

Example continued:

```
.item {
  flex: 1;
  padding: 10px;
  background-color: lightblue;
}
```

How Flexbox Works:

- You can set **direction** using flex-direction (row, column, etc.)
- Align items with:
    - justify-content → main axis (e.g., left, right, center)
    - align-items → cross axis (e.g., top, center, stretch)
    - align-content → multiple rows/columns

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

Below is a clear explanation of the **Flexbox properties**: justify-content, align-items, and flex-direction — these control layout and alignment of items in a flex container.

1. flex-direction

This property defines the **main axis** — the direction in which the flex items are placed.

Syntax:

CSS:-

flex-direction: row | row-reverse | column | column-reverse;

Value	Description	Visual
row	(default) Items go left → right (horizontal)	🟦 🟦
row-reverse	Items go right → left	🟦 🟦
column	Items go top → bottom (vertical)	⬇⬇⬇
column-reverse	Items go bottom → top	⬆⬆⬆

2. justify-content

Controls **alignment of items along the main axis** (horizontal if flex-direction: row, vertical if column).

Syntax:

CSS:-

justify-content: flex-start | flex-end | Center | space-between | space-around | space-evenly;

Value	Description
flex-start	Items aligned to the start of main axis (default)
flex-end	Items aligned to the end of main axis
center	Items centered along main axis
space-between	Items spread out; first & last at edges
space-around	Equal space around each item
space-evenly	Equal space between all items

3. align-items

Controls alignment along the cross axis (opposite of main axis).
So, if flex-direction is row, it aligns items vertically.

Syntax:

align-items: stretch | flex-start | flex-end | center | baseline;

Value	Description
stretch	(default) Items stretch to fill container height/width
flex-start	Items align to **top** (if row) or **left** (if column)
flex-end	Items align to **bottom** (if row) or **right** (if column)
center	Items are **centered** along cross axis
baseline	Align items by **text baseline**

⇨ Quick Example:

```
<div class="flex-container">
  <div>Box 1</div>
  <div>Box 2</div>
  <div>Box 3</div>
</div>

<style>
.flex-container {
  display: flex;
  flex-direction: row;        /* Items go left to right */
  justify-content: space-between; /* Space between items */
  align-items: center;        /* Vertically center items */
}
</style>
```

CSS Grid

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

⇨ What is CSS Grid?

CSS Grid is a layout system in CSS that allows you to create two-dimensional layouts — managing rows and columns at the same time.It's perfect for:

- Complex page layouts
- Grid-like designs (galleries, dashboards, etc.)
- Aligning items in both vertical and horizontal axes

Example:

CSS:-

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

What is Flexbox?

Flexbox (Flexible Box Layout) is a **one-dimensional** layout system — it aligns items in **either a row or a column**, but not both at the same time.

It's ideal for:

- Nav bars
- Toolbars
- Centering elements
- Layouts with content of **variable sizes.**

Key Differences: CSS Grid vs Flexbox

Feature	CSS Grid	Flexbox
Layout Type	Two-dimensional (rows + columns)	One-dimensional (row **or** column)
Use Case	Full page layouts, image galleries	Menus, buttons, single row/column
Item Placement	Precise row/column placement (grid-area)	Auto flow based on content order
Alignment	Controls **both axes** together	Controls **one axis** at a time
Responsiveness	Better for **fixed grid-based layouts**	Better for **fluid/responsive** items
Browser Support	Modern browsers (good support)	Very well supported

When to Use CSS Grid Over Flexbox

Situation	Use
You need to build a complex layout with multiple rows and columns	✅ Grid
You want to align items along one direction (row or column)	✅ Flexbox
You want equal spacing between items in a row	✅ Flexbox
You need precise control over grid areas (e.g., dashboard layout)	✅ Grid
You're creating a form, toolbar, or nav bar	✅ Flexbox
You want to overlap items or layer them	✅ Grid

Example Use Case:

- ⇨ **Grid Example:**

CSS:-

```
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
}
```

⇨ **Flexbox Example:**

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

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

These properties are used to define the structure and spacing of a grid layout in CSS. Let's explore each one with examples:

1. grid-template-columns

Defines the **number and width of columns** in the grid container.

Syntax:

grid-template-columns: value value ...;

Common values:

- **px, em, % — Fixed units**
- **fr — Fractional unit (portion of remaining space)**
- **auto — Size based on content**

⇨ **Example:**
⇨ **CSS:-**

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

This creates 3 columns:

- 1st = 200px

- 2nd = 1 portion
- 3rd = 2 portions of the remaining space

⇨ **grid-template-rows**

- Defines the **number and height of rows** in the grid container.

Syntax:

grid-template-rows: value value ...;

Example:

CSS:-

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

This creates 3 rows:

- 1st = 100px
- 2nd = size based on content
- 3rd = 50px

3. grid-gap (Also: row-gap, column-gap)

Defines the **space between grid items** (not around them).
This is shorthand for setting both row and column gaps.

Syntax:

CSS:-

grid-gap: rowGap columnGap;

Example:

CSS:-

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-template-rows: auto auto;
  grid-gap: 20px 10px;
}
```

Adds:

- 20px vertical space between rows
- 10px horizontal space between columns

You can also use:

```
row-gap: 20px;
column-gap: 10px;
```

⇨ Full Example:-

```
<div class="container">
  <div>Box 1</div>
  <div>Box 2</div>
  <div>Box 3</div>
  <div>Box 4</div>
</div>

<style>
.container {
  display: grid;
  grid-template-columns: 1fr 2fr;
```

```
  grid-template-rows: 100px 150px;

  grid-gap: 15px;

  background-color: #f2f2f2;

}

.container div {

  background-color: lightblue;

  padding: 10px;

  text-align: center;

}

</style>
```

Responsive Web Design with Media Queries

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

- ⇨ **What Are Media Queries in CSS?**

  Media queries are CSS techniques used to apply styles conditionally based on the device's characteristics such as:

- Screen width or height
- Device type (screen, print, etc.)
- Orientation (landscape/portrait)
- Resolution (retina or standard displays)

  They are essential for responsive web design, allowing your layout to adapt to different screen sizes — like mobile phones, tablets, and desktops.

CSS:-

```
@media (condition) {

  /* CSS rules that apply only if the condition is true */

}
```

Example: Basic Responsive Media Query

```
/* Base styles for all devices */
body {
  font-size: 16px;
  background: white;
}

/* Styles for screens smaller than 768px (like mobiles) */
@media (max-width: 768px) {
  body {
    font-size: 14px;
    background: lightgray;
  }
}
```

On screens **≤768px**, font-size will shrink, and background will change.

Condition	Example	Description
max-width	(max-width: 600px)	Applies styles when screen ≤ 600px
min-width	(min-width: 1024px)	Applies when screen ≥ 1024px
orientation	(orientation: landscape)	Applies to landscape devices
device-width	(device-width: 320px)	Exact device width

Why Are Media Queries Important for Responsive Design?

Device Adaptability
→ Ensures content looks good on mobile, tablet, laptop, or desktop.

Improved User Experience
→ Adapts font sizes, layouts, navigation, etc., for different screens.

Performance Optimization
→ Can hide or rearrange content to reduce clutter on small screens.

Future-Proofing
→ Prepares your layout for newer devices with varying screen sizes.

- **Advanced Example:**

```
/* For phones */
@media (max-width: 600px) {
  .nav {
    flex-direction: column;
  }
}

/* For tablets */
@media (min-width: 601px) and (max-width: 992px) {
  .nav {
    flex-direction: row;
  }
}

/* For desktops */
@media (min-width: 993px) {
  .nav {
    justify-content: space-between;
  }
}
```

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

Below Mentioned basic media query in CSS that adjusts the font size for screens smaller than 600px:

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

 body {

   font-size: 14px;

 }

}
```

Explanation:

- @media (max-width: 600px) targets devices/screens **600 pixels wide or less**.
- Inside the block, we define the font-size for the body (or any element you want to style).
- You can adjust 14px to any value suitable for your design.

Typography and Web Fonts

⇨ Difference Between Web-Safe Fonts and Custom Web Fonts:

Feature	Web-Safe Fonts	Custom Web Fonts
Definition	Fonts that are pre-installed on most devices.	Fonts loaded from the web or external files.
Examples	Arial, Times New Roman, Verdana, Georgia	Google Fonts (e.g., Roboto, Lato), Adobe Fonts
Availability	Readily available on all systems	Must be downloaded when the page loads
Performance	Very fast (no need to load externally)	Can slow page load due to extra HTTP requests
Design Flexibility	Limited (basic styles)	High flexibility with modern, unique typefaces
Compatibility	Excellent across all devices and browsers	May require fallbacks or additional configuration

Why You Might Use a Web-Safe Font Over a Custom Font:

1. Faster Load Times
   No additional HTTP requests or font file downloads — great for performance optimization.

2. Better Compatibility
   Works across all browsers and operating systems without issues.
3. Low Bandwidth Situations
   Ideal for users on slow or limited data connections.
4. No External Dependencies
   No risk of font CDN being down or blocked by firewalls.
5. Simple Projects or Emails
   Web-safe fonts are perfect for emails or quick prototypes where reliability matters more than style.

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

The font-family property in CSS specifies the typeface (font) used to display text on a webpage.

CSS:-

```
p {

  font-family: Arial, sans-serif;

}
```

- You can specify **multiple fonts** as a fallback list.
- The browser uses the **first available** font in the list.
- Generic font families like sans-serif, serif, or monospace should be used last as a fallback.

- ## How to Apply a Custom Google Font to a Webpage

Step 1: Include the Font from Google Fonts

- Go to https://fonts.google.com, choose a font (e.g., **Roboto**), and copy the `<link>` tag provided.
- **Example:**

```
<head>
```

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

    </head>
```

- Step 2: **Use the Font in CSS with font-family**

CSS:-

```
body {

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

}
```

- **Full Example:**

HTML:-

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>Google Font Example</title>

  <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>

  <p>This paragraph uses the Roboto font.</p>
```

```
</body>

</html>
```