❖ Frontend – CSS and CSS3

❖ CSS Selectors & Styling

➢ Theory Assignment :-

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

Ans:- A **CSS selector** is a pattern used to select elements on a web page to apply styles to them. CSS selectors target HTML elements based on attributes like tag name, class, ID, and more.

Types of CSS selectors:-

1. **Element Selector (Tag Selector)**:

- This selector targets HTML elements by their tag name (such as div, p, h1, etc.)

Example :-

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

2. **Class Selector**:

- This selector targets elements with a specific class attribute. The class name is preceded by a period (.).

Example :-

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

3. **ID Selector**:

- This selector targets an element with a specific id attribute. The ID is preceded by a hash (#).

Example :-

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

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

Ans :- **CSS Specificity**

CSS is a system that determines which CSS rule is applied when there are multiple conflicting rules targeting the same element. It helps the browser decide which style to use based on the specificity of the selectors.

Specificity is calculated based on the components of the CSS selector, and each type of selector contributes a certain amount to the specificity score. The more specific a selector is, the higher its specificity, meaning it will override styles with lower specificity.

How Specificity Works :-

Specificity is calculated using a four – part value , often represented as :

Inline styles (highest specificity)

IDs

Classes, attributes, and pseudo-classes

Elements and pseudo-elements (lowest specificity)

Example :-

```
/* Rule 1: element selector */
p {
  color: red;
}

/* Rule 2: class selector */
.text {
  color: blue;
}

/* Rule 3: ID selector */
#main {
  color: green;
}
```

Order of Resolution

If two selectors have the same specificity, the **last one in the CSS** (or the one defined later) will take precedence. This is because the browser processes styles in the order they appear in the stylesheet.

Example :-

```
/* Rule 1 */
button {
  background-color: red;
}

/* Rule 2 */
```

```
button.primary {
  background-color: blue;
}

/* Rule 3 */
button#submit {
  background-color: green;
}
```

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

Ans :- **1. Inline CSS**

Inline CSS is used when you apply styles directly to an individual HTML element using the style attribute.

Example :-

```
<p style="color: blue; font-size: 16px;">This is a blue paragraph.</p>
```

Advantage :-

- **Quick and easy** for applying styles to a specific element.
- **No need for external files**, so it's convenient for very small changes or testing.
- Can override other styles with higher specificity.

Disadvantage :-

- **Not reusable across multiple pages**: If you need the same styles on multiple pages, you'd have to duplicate the CSS in each page's <head> section.
- **Large file size**: As the CSS grows, embedding it within the HTML can increase the size of the file, especially for large projects.
- **Less efficient for multiple pages**: Internal CSS increases the load time because the styles are reloaded every time a new page is loaded.

2.Internal CSS:- Internal CSS is used when you include the CSS within the <style> tag in the <head>section of an html document.

```
<!DOCTYPE html>
<html>
<head>
  <style>
    p {
      color: blue;
```

```
    font-size: 16px;
  }
 </style>
</head>
<body>
 <p>This is a blue paragraph.</p>
</body>
</html>
```

Advantages:

- **Keeps HTML and CSS separate**: This method separates the structure of the document from its styling, which is better for maintainability than inline CSS.

No need for an external file: If you only have one HTML file, using internal CSS is simple and efficient.

More control over multiple elements: You can apply the same styles to multiple elements within the same page without repeating them.

Disadvantages:

- **Not reusable across multiple pages**: If you need the same styles on multiple pages, you'd have to duplicate the CSS in each page's <head> section.

Large file size: As the CSS grows, embedding it within the HTML can increase the size of the file, especially for large projects.

Less efficient for multiple pages: Internal CSS increases the load time because the styles are reloaded every time a new page is loaded.

External CSS

External CSS involve linking a separate CSS file to your html file using the <link>tag. The CSS file is typically stored in a .css file.

Example :-

```
<!-- HTML file -->
<link rel="stylesheet" href="styles.css">
<p>This is a blue paragraph.</p>
```

Example :-

```
/* styles.css */
```

```
p {
  color: blue;
  font-size: 16px;
}
```

Advantages:

- **Separation of concerns**: Keeps the HTML structure separate from the styling, making it easier to maintain, read, and manage.
- **Reusability**: The same external CSS file can be applied to multiple HTML pages, ensuring consistent design across a website and reducing redundancy.
- **Improved loading performance**: Browsers cache the external CSS file, so it only needs to be downloaded once, improving loading times for subsequent page visits.
- **Easier maintenance**: Updating the CSS in one place updates the styles across all pages using that CSS file.

Disadvantages:

- **Additional HTTP request**: The browser needs to request the external CSS file, which can increase load time, especially if the file is large and the server is slow.
- **Not ideal for very small projects**: For single-page projects or simple HTML documents, external CSS might seem unnecessary, and internal or inline CSS might be more practical.
- **Requires file management**: You need to manage multiple files (HTML and CSS), which can be cumbersome for small projects or quick prototypes.

❖ CSS BOX MODEL

THEORY ASSIGNMENT :-

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

Ans :- CSS box model is a fundamental concept that describe the rectangular concept that are generated for elements on a webpage is represented as a box and the box model consists of four main parts .

1.Content

what it is the actual content of the element such as text image or other nested element .

effect on size the size of the content place is controlled by the (width) and (height)properties of the element . This is the innermost part of the box and determines the core area where the content is placed .

2. Padding

What it is The space between the content and the border of the element . Padding create extra space around the content inside the element .

Effect on size Padding increases the total size of the element . for example if you have a width of 100px and 10px of padding on each side the overall width of the element would be 120px (100px width +10px padding on the left +10px padding on the right).

3.Border

What it is A can define surrounding the padding and content of the element . you can define its thickness , style , (solid , dashed , etc) and color .

Effect on size border adds extra space to the total size of the element . if a border is 2px thick it increases the elements width and height by 2px on all sides .

4. **Margin**

3. What it is the space outside the border creating distance between the element and other surrounding element itself but create space around it .

Effect on size Margin do not affect the element internal size (content + padding + border) but they control the spacing between the element and other elements . A larger margin means more space between the element and other content but it increase the element box size .

Example :-

```
div {
  width: 200px;
  height: 100px;
  padding: 10px;
  border: 2px solid black;
  margin: 20px;
}
```

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

Which is the default ?

Ans :- Difference Between border-box and content-box:

1. Content-box (default)
   how it work the width and height properties apply only to the content of the element . Padding and border are added outside the content box which increases the total size of the element .

   Example :-

```
div {
  width: 200px;
  height: 100px;
  padding: 10px;
```

```
  border: 2px solid black;
}
```

Formula :-

Total Width = content width + left padding + right padding + left border + right border

Total Height = content height + top padding + bottom padding + top border + bottom Border .

2. **border-box**
   How it works: The width and height properties include the **content**, **padding**, and **border**. In other words, the specified width and height are the **total size**, so the padding and border are *subtracted* from the content area to fit within the specified dimensions.

   Example :-

```
div {
  width: 200px;
  height: 100px;
  padding: 10px;
  border: 2px solid black;
  box-sizing: border-box;
}
```

   Formula :-

Total Width = width (content + padding + border)

Total Height = height (content + padding + border)

=> When to Use Which?

content-box (default):

- Useful when you want to control the exact size of the content, and you're okay with the padding and borders increasing the total size of the element.

border-box:

- It's often more practical for layouts because it makes it easier to manage the total element size. This is especially helpful when you want to maintain a fixed size for an element, including padding and border, without it growing unexpectedly.

CSS FLEXBOX

THEORY ASSIGNMENT :-

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

Ans :- **CSS Flexbox** (short for **Flexible Box Layout**) is a layout model that allows you to design complex layouts with a more efficient and predictable way of distributing space and aligning items within a container, even when the size of the items is unknown or dynamic. It makes it easy to align elements horizontally and vertically within a container, and it helps create flexible and responsive designs.

Terms:

1. Flex Container:

A **flex container** is the parent element that holds the flex items. It is the element to which **display: flex;** or **display: inline-flex;** is applied. When you apply display: flex;, the container becomes a flex container and its child elements (the flex items) will automatically follow the flex layout model.

How to create a flex container:

```
.flex-container {

  display: flex;

}
```

2. Flex Item:

Flex items are the children of the flex container. These are the individual elements inside the flex container that are laid out according to the flex model. By default, each child element within the flex container becomes a flex item.

How to make an item a flex item: Just place the item inside a flex container. No special styling is required unless you want to control their behavior further.

Example :-

```
<div class="flex-container">

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

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

  <div class="flex-item">Item 3</div>

</div>
```

CSS:

```
.flex-container {

  display: flex;

}



.flex-item {

  flex: 1; /* Distribute space evenly among items */

}
```

Key Flexbox Properties:

For the Flex Container:

1. **display: flex;**: Turns the element into a flex container.
2. **flex-direction**: Defines the direction of the main axis (default is row).
    - row: Items are laid out horizontally.
    - column: Items are laid out vertically.
    - row-reverse: Items are laid out in the opposite direction

column-reverse: Items are laid out in the opposite direction (bottom to top).

- **justify-content**: Aligns flex items along the **main axis** (horizontal by default).
    - Options: flex-start, flex-end, center, space-between, space-around, etc.
- **align-items**: Aligns flex items along the **cross axis** (vertical by default).
    - Options: flex-start, flex-end, center, stretch,

flex-wrap: Controls whether the flex items should wrap onto multiple lines if there's not enough space in the container.

- nowrap: No wrapping (default).
- wrap: Items will wrap onto multiple lines.
- wrap-reverse: Items wrap in reverse order.

align-content: Aligns multiple lines of flex items (when wrapping) along the **cross axis**.

- Similar to align-items, but for wrapped lines.

Why Flexbox is Useful for Layout Design:

Responsive Design: Flexbox makes it easy to create layouts that adapt to different screen sizes, like adjusting the number of columns in a grid.

Vertical Alignment: Aligning items vertically (which was traditionally tricky with other layout methods) is easy with properties like align-items and align-self.

Equal Height Columns: Flexbox can be used to create layouts where elements (such as columns) have equal height, regardless of the content inside them.

Simplified Layouts: Flexbox eliminates the need for using floats or positioning, which can be complex and difficult to maintain.

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

Ans :- 1 . flex-direction

This property define the direction of the main axis that is the direction in which the flex item are placed in the flex container .

Values:

- row (default): Items are placed **horizontally**, from left to right.
- row-reverse: Items are placed **horizontally**, but in reverse (right to left).
- column: Items are placed **vertically**, from top to bottom.
- column-reverse: Items are placed **vertically**, from bottom to top.

2. justify-content

This control how item are aligned along the main axis (horizontal if flex direction: row , vertical if column).

Common values:

- flex-start: Items align to the **start** of the main axis.
- flex-end: Items align to the **end**.
- center: Items are **centered**.
- space-between: Items are spaced **with equal space between** them.
- space-around: Items have equal space **around** them.
- space-evenly: Items have **equal space between and around** them.

3 . align-items

This controls how items are **aligned along the cross axis** (perpendicular to the main axis).

Common values:

- stretch (default): Items stretch to fill the container (if height/width isn't set).
- flex-start: Items align to the **start** of the cross axis.
- flex-end: Items align to the **end**.
- center: Items are **centered**.
- baseline: Items align based on their **text baseline**.

CSS GRID

Theory Assignment

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

Ans :- CSS grid is a powerful layout system in CSS used to create two-dimensional layout . It lets you control both rows and column at the same time , making it perfect for creating structured layouts like webpages , dashboards or galleries .

Feature	CSS Grid	Flexbox
Axis	**Two-dimensional** (rows and columns)	**One-dimensional** (either row **or** column)
Layout	Best for full page or section layout	Best for aligning items within a single row or column
Placement	Items can be placed anywhere in the grid using line numbers or area names	Items are placed in order along the main axis
Overlap	Supports item **overlap**	No overlap (items flow linearly)
Control	Full control over both axes	Control over one axis at a time

When to use Grid over Flexbox?

- You're designing a **full layout** or complex component with rows **and** columns.
- You need to **align items in both directions** (e.g., a card layout, image gallery).
- You want to **precisely place items** in specific grid cells.

Use Flexbox **when:**

- You're building a **navigation bar**, button group, or simple linear layout.
- You want to align and distribute **items in one direction** (either row or column).
- You need content to **wrap or space evenly** in a line.

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

Ans :- grid-template-columns

This property defines the number and size of the columns in a grid .

Syntax example :-

CSS

```
.grid-container {
Display : grid ;
Grid-template-column : 200px 100px auto;
}
```

CSS

grid-template-columns: repeat(3, 1fr);

3. **grid-template-rows**

```
.grid-container {
  display: grid;
  grid-template-rows: 100px 150px auto;
}
```

CSS

```
grid-template-rows: repeat(2, 200px);
```

4. **grid-gap** (also known as gap)

CSS

```
.grid-container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(2, 150px);
  grid-gap: 20px;
}
```

CSS

```
grid-gap: 10px 30px; /* 10px row gap, 30px column gap */
```

Responsive Web Design with Media Queries

Theory assignment :-

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

Ans :- **What are Media Queries in CSS?**

Media queries are a feature in CSS that let you apply styles **based on the device's characteristics**, such as:

- Screen **width** and **height**
- **Orientation** (portrait or landscape)
- **Resolution**
- And more...

Why Are Media Queries Important for Responsive Design?

- Create **mobile-friendly** designs.
- **Avoid content overflow** or unreadable layouts on smaller screens.
- Deliver a better **user experience** across devices.
- Reduce the need for separate mobile sites or apps.

Example :-

```
/* Default styling for desktop */
body {
  font-size: 18px;
  background-color: white;
}

/* For screens 768px wide or smaller (e.g., tablets or phones) */
@media (max-width: 768px) {
  body {
    font-size: 16px;
    background-color: lightgray;
  }
}

/* For very small screens (like phones) */
@media (max-width: 480px) {
  body {
    font-size: 14px;
    background-color: lightblue;
  }
}
```

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

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

/* Media query for screens smaller than 600px */
@media (max-width: 600px) {
  body {
    font-size: 16px;
  }
```

}

TYPOGRAPHY AND WEB FONTS

Theory assignment :-

• 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?

Ans :- **1. Web-Safe Fonts**

Web-safe fonts are a set of fonts that are **pre-installed on most operating systems and devices**, such as Windows, macOS, iOS, and Android.

Examples:

- Arial
- Times New Roman
- Verdana
- Georgia
- Courier New

2. Custom Web Fonts

Custom web fonts are fonts that are **not pre-installed** on devices. They are loaded from the web (often using services like **Google Fonts**, **Adobe Fonts**, or by self-hosting).

Examples:

- Roboto
- Lato
- Open Sans
- Poppins
- Montserrat

To use one :

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


<style>
  body {
    font-family: 'Roboto', sans-serif;
  }
</style>
```

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

You might choose a **web-safe font** when:

- You want **faster loading** (especially on slower connections).
- You're building a site where **performance is a priority**.
- You want to ensure **compatibility across all devices**—no font fallback issues.
- You're creating something simple or professional where standard fonts are just fine (e.g., emails, admin dashboards).

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

Ans :- font family :-

It is used to specify the typeface (font) for text content on a webpage .

Example :-

CSS

```
Body {
 Font-family: “Arial”, “Helvetica”, sans-serif;
  }
```

Step 1 add link to the font in your html <head>

Html

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

CSS

```
 Body {
 Font-family: ‘Poppins’ , sans-serif ;
  }
```