Module 3 – mernstack – CSS and CSS3

CSS Selectors & Styling

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 an HTML page so that you can apply styles to them. It helps target specific HTML elements and apply CSS rules to them.

Here are examples of different types of CSS selectors:

1. Element Selector: This selector targets HTML elements directly. It selects all instances of that element on the page.

Example:

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

This will change the text color of all <p> (paragraph) elements to blue.

2. Class Selector: A class selector targets elements that have a specific class attribute. It starts with a dot (.) followed by the class name.

Example:

```
.myClass {
  font-size: 18px;
}
```

This will change the font size of all elements with the class myClass to 18px.

3. ID Selector: An ID selector targets an element with a specific id attribute. It starts with a hash (#) followed by the ID name.

Example:

```
#myId {
  background-color: yellow;
}
```

This will change the background color of the element with the ID myId to yellow.

Each selector helps you apply styles to specific elements, classes, or IDs, giving you full control over the page's design and layout.

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

ANS:- CSS Specificity is a mechanism that determines which CSS rule is applied to an element when there are conflicting styles. It is essentially a set of rules that the browser uses to decide which CSS declaration to apply when multiple rules target the same element.

CSS specificity is based on the matching rules in the selector. It can be thought of as a "weight" or "priority" system for different types of selectors. Here's how the specificity is calculated:

Specificity Hierarchy:

1. Inline styles (highest specificity):
   - If styles are applied directly on an element using the style attribute, they take the highest priority.

Example:

```
<div style="color: red;">This text is red.</div>
```

Specificity: 1000 (highest priority)

2. ID selectors:
   - ID selectors are more specific than class selectors or element selectors.

Example:

```
#header {
  color: blue;
}
```

Specificity: 100

3. Class selectors, attribute selectors, and pseudo-classes:
   - These have medium specificity.

Example:

```
.button {
  color: green;
}
```

Specificity: 10

4. Element selectors and pseudo-elements:
   - These have the lowest specificity.

Example:

```
p {
   color: purple;
}
```

Specificity: 1

How conflicts are resolved:

When multiple rules target the same element, the browser uses the specificity of the selectors to determine which rule to apply. The general rule is: the more specific rule wins. Here's how conflicts are resolved:

1. Inline styles always win, as they have the highest specificity.
2. If two selectors have the same specificity, the last defined rule in the CSS is applied (i.e., the one that appears last in the stylesheet or CSS block).
3. If there is a conflict between an ID selector, class selector, or element selector, the rule with the highest specificity value is applied.

Example:

```
/* Rule 1 */
#header {
  color: blue;
}

/* Rule 2 */
.button {
  color: green;
}

/* Rule 3 */
p {
```

color: red;

}

If you have this HTML:

<div id="header" class="button">Text</div>

Here's the specificity breakdown:

- Rule 1 (#header) has a specificity of 100 (ID selector).
- Rule 2 (.button) has a specificity of 10 (class selector).
- Rule 3 (p) has a specificity of 1 (element selector).

In this case, Rule 1 (#header { color: blue; }) will be applied because it has the highest specificity (100).

Conclusion:

CSS specificity ensures that more specific selectors override more general ones, helping avoid conflicts. In case of ties, the later rule takes precedence.

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

ANS:- Definition: Internal CSS is written within the <style> tag inside the <head> section of the HTML document.

- **Example**:

html

CopyEdit

```
<head>
  <style>
    body {
      background-color: lightblue;
    }
  </style>
</head>
```

- **Advantages**:
  - Allows you to style a single HTML document without affecting others.

  - Useful for smaller projects or when styles are specific to one page.

- **Disadvantages**:
  - Increases the size of the HTML document.
  - Styles are not reusable across multiple pages, leading to code duplication.

2. External CSS:

- **Definition**: External CSS is written in a separate .css file and linked to the HTML document using the <link> tag in the <head> section.
- **Example**:

html

CopyEdit

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

- **Advantages**:
  - Keeps HTML files clean and organized by separating structure and styling.
  - The same CSS file can be used across multiple pages, promoting code reusability.
  - Improves page load times by caching the CSS file.
- **Disadvantages**:
  - Requires an additional HTTP request to load the external CSS file.
  - If the link to the CSS file is broken or missing, styling will not be applied.

3. Inline CSS:

- **Definition**: Inline CSS is applied directly to individual HTML elements using the style attribute.
- **Example**:

html

CopyEdit

```
<p style="color: red;">This is a red paragraph.</p>
```

- **Advantages**:

    - Quick and easy for styling individual elements without affecting others.
    - No need for external files or extra sections in the HTML document.

- **Disadvantages**:
    - Not efficient for large projects, as styles need to be repeated for each element.
    - Makes the HTML code cluttered and harder to maintain.
    - Does not support CSS inheritance or cascading as effectively as other methods.

Lab Assignment

Style the contact form (created in the HTML Forms lab) using external CSS. Thefollowing should be implemented:

⇒ Change the background color of the form.

⇒ Add padding and margins to form fields.

⇒ Style the submit button with a hover effect.

⇒ Use class selectors for styling common elements and ID selectors for uniqueelements.

ANS:-

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Contact Form</title>
   <link rel="stylesheet" href="style.css">

</head>
<body>
```

```
    <div class="form-container">
        <h2>Contact Us</h2>
        <form id="contact-form">
            <label for="name">Full Name:</label>
            <input type="text" id="name" class="form-field" placeholder="Enter your name" required>

            <label for="email">Email Address:</label>
            <input type="email" id="email" class="form-field" placeholder="Enter your email" required>

            <label for="message">Message:</label>
            <textarea id="message" class="form-field" placeholder="Your message" required></textarea>

            <button type="submit" id="submit-btn">Submit</button>
        </form>
    </div>

</body>
</html>
```

CSS FILE

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

```
.form-container {
  width: 50%;
  margin: 0 auto;
  padding: 20px;
  background-color: #f9f9f9;
  border-radius: 8px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}

#contact-form {
  display: flex;
  flex-direction: column;
}

.form-field {
  margin-bottom: 15px;
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
  font-size: 16px;
}

.form-field:focus {
  border-color: #007BFF;
  outline: none;
}
```

```
#submit-btn {
   padding: 12px 20px;
   border: none;
   background-color: #28a745;
   color: white;
   font-size: 16px;
   border-radius: 4px;
   cursor: pointer;
   transition: background-color 0.3s;
}

#submit-btn:hover {
   background-color: #218838;
}

label {
   font-size: 14px;
   margin-bottom: 5px;
}

h2 {
   text-align: center;
   margin-bottom: 20px;
}
```

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?

ANS:- The CSS box model defines how elements are structured on a webpage, consisting of four components: content, padding, border, and margin. The content is the actual area where text or images are displayed, and its size is determined by width and height. Padding adds space between the content and the border, increasing the overall size. Border surrounds the padding and content, also contributing to the total size. Margin creates space around the element, affecting its position but not its size. Together, padding, border, and margin increase the total size of an element.

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

ANS:- In CSS, the **box-sizing** property defines how the total width and height of an element are calculated, including or excluding padding and borders.

1. **content-box** (default):
   - **Definition**: The default value for box-sizing. The width and height apply only to the content area. Padding and borders are added outside the content area, increasing the overall size of the element.
   - **Effect**: The total size of the element is:
     Total width=Content width+Padding+Border\text{Total width} = \text{Content width} + \text{Padding} + \text{Border} This can lead to unexpected layout issues when you set specific width or height values.
2. **border-box**:
   - **Definition**: The width and height include the content, padding, and border. The total size of the element remains the same, even if padding and borders are added.
   - **Effect**: The total size of the element is:
     Total width=Width (content + padding + border)\text{Total width} = \text{Width (content + padding + border)} This makes it easier to control the element's dimensions, as padding and borders do not affect the overall size.

Default:

The default value is **content-box**.

Task

Create a profile card layout using the box model. The profile card shouldinclude:

⇒ A profile picture.

⇒ The user’s name and bio.

⇒ A button to "Follow" the user.

ANS:- <!DOCTYPE html>

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Profile Card</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>

    <div class="profile-card">
        <img src="WhatsApp Image 2024-12-19 at 1.35.16 PM.jpeg" alt="Profile Picture" 
class="profile-img">
        <h2 class="username">Sagar Lakhnotra</h2>
        <p class="bio">Frontend skil and BCA degree</p>
        <button class="follow-btn">Follow</button>
    </div>

</body>
</html>
```

CSS file

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

body {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    background-color: #f4f4f4;
}

.profile-card {
    width: 300px;
    background-color: #fff;
    border: 1px solid #ddd;
    border-radius: 10px;
    padding: 20px;
    text-align: center;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    margin: 20px;
}

.profile-img {
    width: 100px;
    height: 100px;
    border-radius: 50%;
    margin-bottom: 15px;

    object-fit: cover;

}


.username {

    font-size: 24px;

    font-weight: bold;

    margin-bottom: 10px;

}


.bio {

    font-size: 14px;

    color: #555;

    margin-bottom: 20px;

}


.follow-btn {

    padding: 10px 20px;

    background-color: #007BFF;

    color: white;

    border: none;

    border-radius: 5px;

    cursor: pointer;

    font-size: 16px;

    transition: background-color 0.3s ease;

}


.follow-btn:hover {
  background-color: #0056b3;
}
```

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 is a layout model that allows easy alignment and distribution of items within a container, even when their size is unknown or dynamic. A flex-container is the parent element that holds flex items and controls their layout with properties like display: flex. Flex-items are the direct children of the flex container, which are arranged based on flexbox properties. Flexbox is useful for creating responsive designs, aligning elements both horizontally and vertically, and simplifying complex layouts without the need for floats or positioning**.**

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

ANS:- In Flexbox, the following properties are used to control the alignment and arrangement of flex items within a flex container:

1. justify-content:
   - Purpose: Aligns flex items along the main axis (horizontal by default).
   - Values:
     - flex-start: Items are aligned at the start of the container.
     - flex-end: Items are aligned at the end of the container.
     - center: Items are centered in the container.
     - space-between: Items are spaced evenly, with the first item at the start and the last at the end.
     - space-around: Items are spaced evenly, with equal space around them.
     - space-evenly: Items are spaced evenly, with equal space between and around them.
2. align-items:
   - Purpose: Aligns flex items along the cross axis (vertical by default).
   - Values:

        - stretch: Items stretch to fill the container (default).
        - flex-start: Items are aligned at the start of the cross axis.
        - flex-end: Items are aligned at the end of the cross axis.
        - center: Items are aligned at the center of the cross axis.
        - baseline: Items are aligned to their baseline.

3. flex-direction:
    - Purpose: Defines the direction in which the flex items are laid out inside the flex container.
    - Values:
        - row: Items are placed in a horizontal row (default).
        - row-reverse: Items are placed in a horizontal row, but in reverse order.
        - column: Items are placed in a vertical column.
        - column-reverse: Items are placed in a vertical column, but in reverse order.

These properties allow for flexible and responsive layouts that can adapt to different screen sizes and orientations.

Task

Create a simple webpage layout using Flexbox. The layout should include:

⇒ A header.

⇒ A sidebar on the left.

⇒ A main content area in the center.

⇒ A footer.

ANS:-

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flexbox Layout</title>
```

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

    <div class="container">
        <header class="header">Header</header>
        <div class="main-content">
            <aside class="sidebar">Sidebar</aside>
            <div class="content">Main Content Area</div>
        </div>
        <footer class="footer">Footer</footer>
    </div>

</body>
</html>
```

CSS FILE

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

.container {
    display: flex;
    flex-direction: column;
    height: 100vh
```

```
/* Header styling */
.header {
    background-color: #4CAF50;
    color: white;
    text-align: center;
    padding: 20px;
    font-size: 24px;
}

/* Main content area */
.main-content {
    display: flex;
    flex: 1
    justify-content: space-between;
    align-items: flex-start;
    padding: 20px;
}

.sidebar {
    width: 200px;
    background-color: #f4f4f4;
    padding: 20px;
    border-right: 2px solid #ddd;
}

.content {
    flex: 1
    background-color: #f9f9f9;
    padding: 20px;
    margin-left: 20px;
}

.footer {
    background-color: #333;
    color: white;
    text-align: center;
    padding: 20px;
    font-size: 18px;
}

@media (max-width: 768px) {
    .container {
        flex-direction: column;
    }

    .main-content {
        flex-direction: column;
    }

    .sidebar {
        width: 100%;
        margin-bottom: 20px;
    }

    .content {
        margin-left: 0;
```

```
   }
}
```

CSS Grid

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

ANS:- CSS Grid is a two-dimensional layout system that allows you to design layouts with both rows and columns, offering complete control over the placement of items in both axes. It's ideal for creating complex, grid-based designs, like entire webpage layouts or card grids. Flexbox, on the other hand, is a one-dimensional layout system focused on arranging items in a row or column.

You would use CSS Grid over Flexbox when you need to create layouts that require control over both rows and columns, such as for page layouts with headers, sidebars, and content areas. Flexbox is better for simpler, linear layouts where items need to be aligned in one direction, like navigation bars or lists.

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

ANS:-

The **CSS Grid** properties **grid-template-columns**, **grid-template-rows**, and **grid-gap** control the structure and spacing of grid items. Here's an explanation of each property with examples:

1. **grid-template-columns**:
   - Defines the columns of the grid and their widths.
   - You can specify values in px, %, fr (fractional units), or auto.
   - Example:
   - .container {
   -     display: grid;
   -     grid-template-columns: 200px 1fr 2fr;
   - }
2. **grid-template-rows**:

    - Defines the rows of the grid and their heights.
    - You can use fixed values like px, em, fr, or auto.
    - Example:
    - .container {
    -     display: grid;
    -     grid-template-rows: 100px auto 200px;
    - }

3. **grid-gap** (or gap):
    - Specifies the space between rows and columns.
    - It accepts values in units like px, em, or rem.
    - Example:
    - .container {
    -     display: grid;
    -     grid-template-columns: 1fr 1fr;
    -     grid-template-rows: auto auto;
    -     grid-gap: 20px;
    - }

Complete Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Grid Example</title>
   <style>
      .container {
         display: grid;
         grid-template-columns: 200px 1fr 1fr;
```

```
            grid-template-rows: 150px auto;
            grid-gap: 10px;
        }

        .item {
            background-color: #4CAF50;
            color: white;
            padding: 20px;
            text-align: center;
        }
    </style>
</head>
<body>

    <div class="container">
        <div class="item">Item 1</div>
        <div class="item">Item 2</div>
        <div class="item">Item 3</div>
        <div class="item">Item 4</div>
    </div>

</body>
</html>
```

Explanation:

- **grid-template-columns** defines three columns: one fixed at 200px and the other two share the remaining space with 1fr each.
- **grid-template-rows** defines two rows: the first row is 150px tall, and the second row adjusts automatically to fit content.
- **grid-gap** creates a 10px space between both rows and columns.

This creates a flexible, responsive grid with a clear layout for managing columns and rows.

Task

Create a 3x3 grid of product cards using CSS Grid. Each card should contain:

⇒ A product image.

⇒ A product title.

⇒ A price.

ANS:-

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Product Grid</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>

  <div class="product-grid">
    <div class="product-card">
      <img src="https://via.placeholder.com/150" alt="Product 1" class="product-image">
      <h3 class="product-title">Product 1</h3>
      <p class="product-price">$20.00</p>
    </div>
    <div class="product-card">
      <img src="https://via.placeholder.com/150" alt="Product 2" class="product-image">
      <h3 class="product-title">Product 2</h3>
      <p class="product-price">$30.00</p>
```

```
</div>
<div class="product-card">
    <img src="https://via.placeholder.com/150" alt="Product 3" class="product-image">
    <h3 class="product-title">Product 3</h3>
    <p class="product-price">$40.00</p>
</div>
<div class="product-card">
    <img src="https://via.placeholder.com/150" alt="Product 4" class="product-image">
    <h3 class="product-title">Product 4</h3>
    <p class="product-price">$50.00</p>
</div>
<div class="product-card">
    <img src="https://via.placeholder.com/150" alt="Product 5" class="product-image">
    <h3 class="product-title">Product 5</h3>
    <p class="product-price">$60.00</p>
</div>
<div class="product-card">
    <img src="https://via.placeholder.com/150" alt="Product 6" class="product-image">
    <h3 class="product-title">Product 6</h3>
    <p class="product-price">$70.00</p>
</div>
<div class="product-card">
    <img src="https://via.placeholder.com/150" alt="Product 7" class="product-image">
    <h3 class="product-title">Product 7</h3>
    <p class="product-price">$80.00</p>
</div>
<div class="product-card">
    <img src="https://via.placeholder.com/150" alt="Product 8" class="product-image">
```

```
            <h3 class="product-title">Product 8</h3>
            <p class="product-price">$90.00</p>
        </div>
        <div class="product-card">
            <img src="https://via.placeholder.com/150" alt="Product 9" class="product-image">
            <h3 class="product-title">Product 9</h3>
            <p class="product-price">$100.00</p>
        </div>
    </div>

</body>
</html>
```

CSS FILE

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

.product-grid {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    grid-gap: 20px;
    padding: 20px;
}

.product-card {
    background-color: #fff;
    border: 1px solid #ddd;
    border-radius: 8px;
    overflow: hidden;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    text-align: center;
    transition: transform 0.3s, box-shadow 0.3s;
}

.product-image {
    width: 100%;
    height: auto;
    object-fit: cover;
}

.product-title {
    font-size: 18px;
    margin: 10px 0;
    color: #333;
}

.product-price {
    font-size: 16px;
    color: #888;
    margin-bottom: 15px;
}

.product-card:hover {
    transform: translateY(-5px);
    box-shadow: 0 8px 12px rgba(0, 0, 0, 0.2);
}

@media (max-width: 768px) {
    .product-grid {
      grid-template-columns: 1fr 1fr;
    }
}

@media (max-width: 480px) {
    .product-grid {
      grid-template-columns: 1fr;
    }
}
```

Responsive Web Design with Media Queries

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

ANS:- Media queries in CSS are used to apply different styles depending on the characteristics of the device or screen size. They enable web developers to create responsive designs that adapt to various screen sizes, resolutions, and device types, such as desktops, tablets, and mobile phones.

They are important for responsive design because they allow the layout and style of a webpage to change based on the user's device, ensuring that the content is readable and visually appealing on all screen sizes. Without media queries, websites might not display properly on devices with different screen resolutions or aspect ratios.

Example:

```
@media (max-width: 768px) {
  body {
    font-size: 14px;
  }
}
@media (max-width: 480px) {
  body {
    font-size: 12px;
  }
}
```

Media queries are key for building flexible, mobile-first websites that provide optimal user experiences across devices.

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

ANS:- Here's a basic media query that adjusts the font size for screens smaller than 600px:

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

This media query targets devices with a screen width of 600px or less and sets the font size of the body text to 14px for better readability on smaller screens.

Lab Assignment

(Task)

Build a responsive webpage that includes:

⇒ A navigation bar.

⇒ A content section with two columns.

⇒ A footer.

ANS:-

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Responsive Webpage</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>

  <header>
    <nav class="navbar">
      <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">About</a></li>
        <li><a href="#">Services</a></li>
        <li><a href="#">Contact</a></li>
      </ul>
    </nav>
  </header>

  <main>
    <section class="content">
      <div class="column left-column">
        <h2>Left Column</h2>
```

```
            <p>This is the left column content. It will stack below the right column on smaller screens.</p>
        </div>
        <div class="column right-column">
            <h2>Right Column</h2>
            <p>This is the right column content. It will also stack below the left column on smaller screens.</p>
```

CSS FILE

```
body {
    font-family: Arial, sans-serif;
    line-height: 1.6;
}

header {
    background-color: #333;
    padding: 10px 0;
}

.navbar ul {
    list-style: none;
    display: flex;
    justify-content: center;
    align-items: center;
}

.navbar ul li {
    margin: 0 15px;
}

.navbar ul li a {
    color: white;
    text-decoration: none;
    font-size: 18px;
}

.navbar ul li a:hover {
    text-decoration: underline;
}

main {
    padding: 20px;
}

.content {
    display: flex;
    justify-content: space-between;
    gap: 20px;
}

.column {
    background-color: #f4f4f4;
    padding: 20px;
    width: 48%;
    border-radius: 5px;
```

```
}

footer {
   background-color: #333;
   color: white;
   text-align: center;
   padding: 10px;
   position: relative;
   bottom: 0;
   width: 100%;
}

@media (max-width: 768px) {
   .content {
      flex-direction: column;
   }

   .column {
      width: 100%;
      margin-bottom: 20px;
   }

   .navbar ul {
      flex-direction: column;
      align-items: center;
   }

   .navbar ul li {
        margin: 10px 0;
    }

    body {
        font-size: 14px;
        padding: 10px;
    }

    .navbar ul li a {
        font-size: 16px;
    }
}
```

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?

ANS:- Web-safe fonts are a set of fonts that are pre-installed on most devices and operating systems, ensuring they are universally available without needing to load additional files. Examples include Arial, Times New Roman, and Courier New. They are fast to load and ensure compatibility across browsers and platforms.

Custom web fonts, on the other hand, are fonts that are not pre-installed and need to be downloaded from a web service or hosted server, like Google Fonts or Adobe Fonts. These fonts offer more design flexibility and creative control but may affect page load times.

You might choose a web-safe font over a custom font to prioritize faster loading times, improve performance, and ensure consistency across all devices without additional dependencies.

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

ANS:- The font-family property in CSS specifies the typeface for the text in an element. It allows you to define one or more font names, and the browser will use the first available font in the list.

Syntax:

```
font-family: "Font1", "Font2", "Font3", sans-serif;
```

In this example:

- "Font1" is the preferred font.
- "Font2" is the fallback font.
- "Font3" is the last fallback font.
- sans-serif is a generic font family in case none of the specified fonts are available.

Applying a Custom Google Font:

To apply a custom Google Font to a webpage, follow these steps:

1. Choose a Google Font: Go to Google Fonts and select the font you want to use.
2. Embed the Google Font:
   - Copy the <link> tag provided by Google Fonts.
   - Paste it inside the <head> section of your HTML document.

Example:

```
<link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap"
rel="stylesheet">
```

3. Apply the Font in CSS: In your CSS file, use the font-family property to apply the Google Font to your elements.

Example:

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

This will apply the "Roboto" font to the text on your webpage, with a fallback to sans-serif if the Google Font fails to load.

Task

Create a blog post layout with the following:

⇒ A title, subtitle, and body content.

⇒ Use at least two different fonts (one for headings, one for body content).

⇒ Style the text to be responsive and easy to read

ANS:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Blog Post</title>
    <!-- Link to Google Fonts -->
    <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&family=Playfair+Display:wght@400;700&display=swap" rel="stylesheet">
    <link rel="stylesheet" href="style.css">
</head>
<body>

    <header class="blog-header">
        <h1 class="blog-title">The Ultimate Guide to Web Development</h1>
        <h2 class="blog-subtitle">Everything you need to know to start your career in web development.</h2>
    </header>

    <main class="blog-content">
        <section class="blog-post">
            <p class="body-content">
        Web development is a broad field with various aspects that you can specialize in. Whether you're interested in front-end, back-end, or full-stack development, there's always something new to learn.

        This guide will walk you through the essential skills you'll need to get started, and offer insights into how to build your web development career.

      </p>

      <p class="body-content">

        From mastering HTML and CSS to learning programming languages like JavaScript and Python, web development requires continuous learning and hands-on practice. With the rise of new frameworks and tools, staying up to date is more important than ever.

      </p>

    </section>

  </main>


  <footer class="blog-footer">

    <p>&copy; 2025 My Blog. All rights reserved.</p>

  </footer>


</body>

</html>
```

CSS FILE

```
* {

  margin: 0;

  padding: 0;

  box-sizing: border-box;

}


body {
  font-family: 'Roboto', sans-serif;
  background-color: #f9f9f9;
  color: #333;
  line-height: 1.6;
  padding: 20px;
}

.blog-header {
  text-align: center;
  margin-bottom: 30px;
}

.blog-title {
  font-family: 'Playfair Display', serif;
  font-size: 2.5rem;
  font-weight: 700;
  margin-bottom: 10px;
}

.blog-subtitle {
  font-family: 'Roboto', sans-serif;
  font-size: 1.5rem;
  font-weight: 400;
  color: #555;
}

.blog-content {
  max-width: 800px;
  margin: 0 auto;
}

.blog-post {
  background-color: white;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}

.body-content {
  font-size: 1rem;
  line-height: 1.8;
  margin-bottom: 20px;
}

.blog-footer {
  text-align: center;
  margin-top: 40px;
  font-size: 0.875rem;
  color: #777;
}

@media (max-width: 768px) {
  .blog-title {
      font-size: 2rem;
  }
```

```
  .blog-subtitle {
    font-size: 1.25rem;
  }

  .body-content {
    font-size: 0.95rem;
  }

  .blog-post {
    padding: 15px;
  }
}

@media (max-width: 480px) {
  .blog-title {
    font-size: 1.75rem;
  }

  .blog-subtitle {
    font-size: 1.125rem;
  }

  .body-content {
    font-size: 0.9rem;
  }
}
```