Module 3 – Frontend – CSS and CSS3

Theory Assignment

• 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 element(s) on the page to apply CSS rules to.

**Types of Selectors:**

1. **Element Selector** – Targets HTML elements directly.
   * **Syntax:** element { property: value; }
   * **Example:**

css

CopyEdit

p {

color: blue;

}

*(Applies blue text color to all <p> tags.)*

1. **Class Selector** – Targets elements with a specific class attribute.
   * **Syntax:** .className { property: value; }
   * **Example:**

css

CopyEdit

.highlight {

background-color: yellow;

}

*(Applies yellow background to elements with class="highlight".)*

1. **ID Selector** – Targets a single element with a specific ID.
   * **Syntax:** #idName { property: value; }
   * **Example:**

css

CopyEdit

#main-header {

font-size: 24px;

}

*(Applies 24px font size to the element with id="main-header".)*

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

**CSS specificity** determines which rule is applied when multiple CSS rules target the same element.

**Specificity Rules (from low to high):**

1. **Element selectors** (e.g., div, p) – low specificity
2. **Class selectors** (e.g., .box, .active)
3. **ID selectors** (e.g., #header)
4. **Inline styles** (e.g., style="color:red;") – highest specificity
5. **!important** overrides all (used cautiously)

**Example:**

html

CopyEdit

<p id="text" class="highlight" style="color: red;">Hello World</p>

css

CopyEdit

p { color: blue; } /\* specificity: 0,0,0,1 \*/

.highlight { color: green; } /\* specificity: 0,0,1,0 \*/

#text { color: orange; } /\* specificity: 0,1,0,0 \*/

* Final color will be **red** because inline style has the highest specificity.

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

| **Type** | **Description** | **Syntax** | **Advantages** | **Disadvantages** |
| --- | --- | --- | --- | --- |
| **Inline CSS** | CSS within an HTML element using style attribute. | <h1 style="color:red;"> | - Quick for single changes - Overrides other styles | - Not reusable - Poor readability - Hard to maintain |
| **Internal CSS** | CSS placed inside <style> tag in the <head>. | <style>h1 { color: red; }</style> | - Good for small pages - Keeps style in one file | - Not reusable across pages - Slower loading |
| **External CSS** | CSS written in separate .css file and linked via <link>. | <link rel="stylesheet" href="style.css"> | - Reusable across multiple pages - Cleaner HTML - Faster page loads with caching | - Extra HTTP request - Doesn’t work offline unless cached |

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.

**✅ 1. HTML Contact Form (Example)**

Make sure your HTML file has class and id attributes for styling:

html

CopyEdit

<!-- contact.html -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>Contact Form</title>

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

</head>

<body>

<form id="contact-form">

<h2>Contact Us</h2>

<label for="name">Name:</label>

<input type="text" id="name" class="form-input" />

<label for="email">Email:</label>

<input type="email" id="email" class="form-input" />

<label for="message">Message:</label>

<textarea id="message" class="form-input"></textarea>

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

</form>

</body>

</html>

**✅ 2. External CSS Styling (styles.css)**

css

CopyEdit

/\* Change the background color of the form \*/

#contact-form {

background-color: #f0f8ff;

width: 400px;

margin: 50px auto;

padding: 20px;

border-radius: 10px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

/\* Common styles for inputs, textarea \*/

.form-input {

display: block;

width: 100%;

margin-bottom: 15px;

padding: 10px;

border: 1px solid #ccc;

border-radius: 5px;

}

/\* Style the submit button \*/

#submit-btn {

background-color: #007bff;

color: white;

border: none;

padding: 12px;

border-radius: 5px;

cursor: pointer;

width: 100%;

font-weight: bold;

}

/\* Hover effect for submit button \*/

#submit-btn:hover {

background-color: #0056b3;

}

**✅ Summary of Requirements Met**

| **Requirement** | **Done** |
| --- | --- |
| Background color of the form | ✅ #contact-form |
| Padding/margins on form fields | ✅ .form-input |
| Submit button styling + hover | ✅ #submit-btn |
| Used class selectors for common | ✅ .form-input |
| Used ID selectors for unique | ✅ #contact-form, #submit-btn |

Let me know if you want a **live preview, PDF**, or **PPT of this styled form**.

CSS Flexbox

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

**🔹 What is CSS Flexbox?**

**CSS Flexbox** (Flexible Box Layout) is a layout model in CSS used to design one-dimensional layouts. It provides a more efficient way to align, distribute, and space elements within a container—even when their size is unknown or dynamic.

Flexbox is especially useful for:

* Centering elements easily (both vertically and horizontally)
* Building responsive layouts
* Managing spacing and alignment in rows or columns

**🔹 Key Terms:**

* **Flex Container**  
  The parent element that uses display: flex or display: inline-flex. It becomes a *flex container* and all its direct children become *flex items*.  
  **Example:**

css

CopyEdit

.container {

display: flex;

}

* **Flex Item**  
  Any direct child of a flex container is a *flex item*.  
  **Example:**

html

CopyEdit

<div class="container">

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

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

</div>

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

**🔹 flex-direction**

Defines the **main axis** (direction) along which the flex items are placed.

| **Value** | **Description** |
| --- | --- |
| row (default) | Items are placed left to right |
| row-reverse | Items are placed right to left |
| column | Items are placed top to bottom |
| column-reverse | Items are placed bottom to top |

css

CopyEdit

.container {

display: flex;

flex-direction: row;

}

**🔹 justify-content**

Controls how items are **aligned along the main axis** (horizontal if row, vertical if column).

| **Value** | **Description** |
| --- | --- |
| flex-start | Align items to the start |
| flex-end | Align items to the end |
| center | Center items |
| space-between | Equal space **between** items |
| space-around | Equal space **around** items |
| space-evenly | Equal space **between and outside** |

css

CopyEdit

.container {

justify-content: center;

}

**🔹 align-items**

Aligns items along the **cross axis** (perpendicular to main axis).

| **Value** | **Description** |
| --- | --- |
| stretch (default) | Stretches items to fill container |
| flex-start | Align items to the top (or left) |
| flex-end | Align items to the bottom (or right) |
| center | Vertically (or horizontally) center |
| baseline | Align items based on text baseline |

css

CopyEdit

.container {

align-items: flex-start;

}

**✅ Summary Table**

| **Property** | **Axis Used** | **Controls** |
| --- | --- | --- |
| flex-direction | Main Axis | Layout direction (row/column) |
| justify-content | Main Axis | Spacing/alignment of items |
| align-items | Cross Axis | Alignment perpendicular to main |

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. Additional Requirements:

⇒ Use Flexbox to position and align the elements.

⇒ Apply different justify-content and align-items properties to observe theireffects.

⇒ Ensure the layout is responsive, adjusting for smaller screens.

**✅ 1. index.html**

html

CopyEdit

<!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>

<header class="header">Header</header>

<div class="container">

<aside class="sidebar">Sidebar</aside>

<main class="main-content">Main Content</main>

</div>

<footer class="footer">Footer</footer>

</body>

</html>

**✅ 2. style.css**

css

CopyEdit

\* {

box-sizing: border-box;

margin: 0;

padding: 0;

font-family: Arial, sans-serif;

}

body {

min-height: 100vh;

display: flex;

flex-direction: column;

}

/\* Header \*/

.header {

background-color: #333;

color: white;

padding: 20px;

text-align: center;

}

/\* Container with Sidebar + Main Content \*/

.container {

flex: 1;

display: flex;

flex-direction: row;

justify-content: space-between; /\* Try center, flex-start, etc. \*/

align-items: stretch; /\* Try center, flex-start \*/

}

/\* Sidebar \*/

.sidebar {

background-color: #f4f4f4;

width: 25%;

padding: 20px;

}

/\* Main Content \*/

.main-content {

background-color: #e2e2e2;

flex: 1;

padding: 20px;

}

/\* Footer \*/

.footer {

background-color: #333;

color: white;

padding: 15px;

text-align: center;

}

/\* Responsive Layout \*/

@media (max-width: 768px) {

.container {

flex-direction: column;

}

.sidebar {

width: 100%;

}

}

**✅ Features Implemented**

| **Feature** | **Implemented** |
| --- | --- |
| Header | ✅ .header |
| Sidebar | ✅ .sidebar |
| Main Content | ✅ .main-content |
| Footer | ✅ .footer |
| Flexbox layout | ✅ display: flex, justify-content, align-items |
| Responsive design | ✅ with @media for smaller screens |

CSS Grid

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

**🔹 What is CSS Grid?**

**CSS Grid Layout** is a powerful 2-dimensional layout system in CSS. It allows you to design web pages by placing elements into rows and columns.

With Grid, you can:

* Create complex layouts with both rows **and** columns
* Control layout areas precisely
* Easily build page templates

**🔹 Difference Between Grid and Flexbox**

| **Feature** | **CSS Grid** | **Flexbox** |
| --- | --- | --- |
| Layout direction | 2D (rows and columns) | 1D (either row **or** column) |
| Best use | Whole page or large sections | Components within a section |
| Alignment | Works in both directions | Works along one axis |
| Item positioning | Items can be placed anywhere | Items flow in order |

**🔹 When to Use Grid Over Flexbox?**

Use **CSS Grid** when:

* You need both **rows and columns**
* You’re building **complex page layouts**
* You want to place items in **specific grid cells**

Use **Flexbox** when:

* You need a **simple row or column**
* You want content to **flow dynamically**
* You’re building **components or toolbars**

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

**🔹 grid-template-columns**

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

**Example:**

css

CopyEdit

.container {

display: grid;

grid-template-columns: 200px 1fr 100px;

}

This creates 3 columns:

* First: 200px
* Second: auto (1 fraction of remaining space)
* Third: 100px

**🔹 grid-template-rows**

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

**Example:**

css

CopyEdit

.container {

display: grid;

grid-template-rows: 100px 2fr 1fr;

}

This creates 3 rows:

* First: 100px
* Second: twice the height of the third row

**🔹 grid-gap (or gap)**

Defines the spacing **between grid items** (both rows and columns).

**Example:**

css

CopyEdit

.container {

display: grid;

grid-template-columns: 1fr 1fr 1fr;

gap: 20px;

}

Creates 3 equal columns with **20px gap** between rows and columns.

You can also use:

css

CopyEdit

row-gap: 10px;

column-gap: 15px;

**✅ Summary Table**

| **Property** | **Purpose** | **Example** |
| --- | --- | --- |
| grid-template-columns | Set number/width of columns | 200px 1fr 100px |
| grid-template-rows | Set number/height of rows | 100px 2fr 1fr |
| grid-gap / gap | Space between grid cells | gap: 20px; |

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

: ⇒ A product image.

⇒ A product title.

⇒ A price. Additional Requirements:

⇒ Use grid-template-columns to create the grid layout.

⇒ Use grid-gap to add spacing between the grid items.

⇒ Apply hover effects to each card for better interactivity

**✅ 1. Folder Structure**

css

CopyEdit

product-grid/

├── index.html

└── style.css

**✅ 2. index.html**

html

CopyEdit

<!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>

<h1 class="title">Our Products</h1>

<div class="grid-container">

<!-- Repeat this card for each product -->

<div class="card">

<img src="https://via.placeholder.com/150" alt="Product" />

<h3>Product Title</h3>

<p>$19.99</p>

</div>

<!-- Copy the above .card block 8 more times to make a 3x3 grid -->

<div class="card">

<img src="https://via.placeholder.com/150" alt="Product" />

<h3>Product Title</h3>

<p>$29.99</p>

</div>

<div class="card">

<img src="https://via.placeholder.com/150" alt="Product" />

<h3>Product Title</h3>

<p>$39.99</p>

</div>

<div class="card">

<img src="https://via.placeholder.com/150" alt="Product" />

<h3>Product Title</h3>

<p>$24.99</p>

</div>

<div class="card">

<img src="https://via.placeholder.com/150" alt="Product" />

<h3>Product Title</h3>

<p>$14.99</p>

</div>

<div class="card">

<img src="https://via.placeholder.com/150" alt="Product" />

<h3>Product Title</h3>

<p>$49.99</p>

</div>

<div class="card">

<img src="https://via.placeholder.com/150" alt="Product" />

<h3>Product Title</h3>

<p>$59.99</p>

</div>

<div class="card">

<img src="https://via.placeholder.com/150" alt="Product" />

<h3>Product Title</h3>

<p>$12.99</p>

</div>

<div class="card">

<img src="https://via.placeholder.com/150" alt="Product" />

<h3>Product Title</h3>

<p>$34.99</p>

</div>

</div>

</body>

</html>

**✅ 3. style.css**

css

CopyEdit

body {

font-family: Arial, sans-serif;

padding: 20px;

background-color: #f8f9fa;

}

.title {

text-align: center;

margin-bottom: 30px;

}

.grid-container {

display: grid;

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

gap: 20px; /\* grid-gap is now just gap \*/

}

.card {

background-color: #fff;

border: 1px solid #ddd;

border-radius: 8px;

padding: 15px;

text-align: center;

box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);

transition: transform 0.3s ease, box-shadow 0.3s ease;

}

.card img {

width: 100%;

height: auto;

border-radius: 4px;

}

.card h3 {

margin: 10px 0 5px;

}

.card p {

color: green;

font-weight: bold;

}

/\* Hover effect \*/

.card:hover {

transform: translateY(-5px);

box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);

}

/\* Responsive Design \*/

@media (max-width: 768px) {

.grid-container {

grid-template-columns: 1fr 1fr;

}

}

@media (max-width: 500px) {

.grid-container {

grid-template-columns: 1fr;

}

}

**✅ Summary of Features Implemented**

| **Requirement** | **Implemented** |
| --- | --- |
| 3x3 grid using grid-template-columns | ✅ repeat(3, 1fr) |
| Spacing using gap (previously grid-gap) | ✅ gap: 20px |
| Product image, title, and price in each card | ✅ Yes |
| Hover effect for interactivity | ✅ transform + box-shadow |
| Responsive design | ✅ Media queries for smaller screens |

Responsive Web Design with Media Queries

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

**🔹 What are Media Queries?**

**Media queries** are a feature of CSS that allow you to apply styles based on the **characteristics of the user’s device**, such as:

* screen width and height
* orientation (landscape or portrait)
* resolution
* and more

**🔹 Why are they important for responsive design?**

Media queries are **essential** for making web pages look good on **all screen sizes and devices**.

**Benefits:**

1. ✅ **Responsive layout** – Adapts the design for mobile, tablet, laptop, or large screens.
2. ✅ **Better user experience** – Font sizes, spacing, and touch areas can be adjusted for usability.
3. ✅ **No need for multiple websites** – One website fits all screen sizes.
4. ✅ **Improved performance & accessibility** – You can hide large images or animations on small or slow devices.

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

/\* Base font size for large screens \*/

body {

font-size: 18px;

}

/\* Media query for small screens (less than 600px) \*/

@media (max-width: 600px) {

body {

font-size: 16px;

}

}

🔍 Explanation:

@media (max-width: 600px) means “if the screen width is 600 pixels or smaller.”

Inside this block, the font size is changed to 16px for better readability on mobile devices.

This overrides the default 18px size when the condition is met.

Build a responsive webpage that includes:

⇒ A navigation bar.

⇒ A content section with two columns.

⇒ A footer. Additional Requirements:

⇒ Use media queries to make the webpage responsive for mobile devices.

⇒ On smaller screens (below 768px), stack the columns vertically.

⇒ Adjust the font sizes and padding to improve readability on mobile.

Here's a complete example of a **responsive webpage** that includes:

* ✅ A **navigation bar**
* ✅ A **two-column content section**
* ✅ A **footer**
* ✅ Responsive design using **media queries**

**✅ 1. Folder Structure**

pgsql

CopyEdit

responsive-page/

├── index.html

└── style.css

**✅ 2. index.html**

html

CopyEdit

<!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>

<nav class="navbar">

<h1>My Website</h1>

<ul class="nav-links">

<li><a href="#">Home</a></li>

<li><a href="#">Services</a></li>

<li><a href="#">Contact</a></li>

</ul>

</nav>

<section class="content">

<div class="column left">

<h2>Left Column</h2>

<p>This is the left content column. It contains some sample text.</p>

</div>

<div class="column right">

<h2>Right Column</h2>

<p>This is the right content column. It also contains sample text.</p>

</div>

</section>

<footer class="footer">

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

</footer>

</body>

</html>

**✅ 3. style.css**

css

CopyEdit

/\* Reset default styling \*/

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

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

line-height: 1.6;

}

/\* Navbar \*/

.navbar {

background-color: #333;

color: #fff;

padding: 15px 20px;

display: flex;

justify-content: space-between;

align-items: center;

flex-wrap: wrap;

}

.navbar h1 {

font-size: 24px;

}

.nav-links {

list-style: none;

display: flex;

gap: 20px;

}

.nav-links li a {

color: white;

text-decoration: none;

font-size: 16px;

}

/\* Content Section \*/

.content {

display: flex;

padding: 30px;

gap: 20px;

}

.column {

flex: 1;

padding: 20px;

background-color: #f4f4f4;

border: 1px solid #ddd;

border-radius: 8px;

}

/\* Footer \*/

.footer {

background-color: #333;

color: white;

text-align: center;

padding: 15px;

margin-top: 30px;

}

/\* Responsive Design \*/

@media (max-width: 768px) {

.content {

flex-direction: column;

}

.navbar {

flex-direction: column;

align-items: flex-start;

}

.navbar h1 {

font-size: 20px;

margin-bottom: 10px;

}

.nav-links {

flex-direction: column;

gap: 10px;

}

body {

font-size: 16px;

}

.column {

padding: 15px;

}

}

**✅ Features Checklist**

| **Feature** | **Implemented** |
| --- | --- |
| Navigation Bar | ✅ Yes |
| Two-column content section | ✅ Using Flexbox |
| Footer | ✅ Styled and centered |
| Responsive layout for mobile | ✅ Using @media (max-width: 768px) |
| Columns stack vertically on small screens | ✅ Yes |
| Font sizes & padding adjusted for mobile | ✅ Yes |

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**

* These are fonts **pre-installed** on almost all operating systems (Windows, macOS, Linux).
* They **do not need to be downloaded** by the browser.
* Ensures **fast loading** and **consistent appearance** across all devices.

**Examples:**

* Arial
* Times New Roman
* Verdana
* Georgia
* Courier New

**🔹 Custom Web Fonts**

* These fonts are **not installed** by default on the user’s system.
* They are loaded from an **external source** such as Google Fonts or Adobe Fonts.
* Gives designers more **styling flexibility** and **modern typeface options**.

**Examples:**

* Roboto
* Poppins
* Open Sans
* Montserrat
* Lato

**🔹 Why choose web-safe fonts over custom fonts?**

| **Criteria** | **Web-safe Fonts ✅** | **Custom Web Fonts ❌** |
| --- | --- | --- |
| Speed | Loads faster | Slightly slower (download needed) |
| Compatibility | Works everywhere | May not load if connection fails |
| Best for | Emails, simple websites, performance-focused apps | Branding, modern designs |

✅ **Use web-safe fonts** when:

* Performance is a top priority
* You are designing for low-bandwidth users
* You need guaranteed cross-platform compatibility

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

**🔹 What is font-family in CSS?**

The font-family property in CSS specifies the **font** to be used for text content on a webpage.

**Syntax:**

css

CopyEdit

element {

font-family: 'Font Name', fallback;

}

**Example:**

css

CopyEdit

body {

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

}

* 'Arial' is the primary font
* sans-serif is the fallback if Arial isn't available

**🔹 How to apply a custom Google Font to a webpage**

**✅ Step-by-step:**

1. **Visit** <https://fonts.google.com>
2. **Select a font**, e.g., *Roboto*
3. **Copy the link tag** and paste it into the <head> of your HTML:

html

CopyEdit

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

1. **Apply the font using CSS:**

css

CopyEdit

body {

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

}

**✅ Alternate method (using @import in CSS):**

css

CopyEdit

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

body {

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

}

**✅ Summary Table**

| **Term** | **Description** | **Example** |
| --- | --- | --- |
| Web-safe Font | Pre-installed, fast, reliable | Arial, Georgia |
| Custom Font | Loaded from web (Google Fonts) | Roboto, Poppins |
| font-family | CSS property to set font | font-family: 'Roboto', sans-serif; |
| Google Font Integration | Use <link> in <head> or @import in CSS | ✅ |

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. Additional Requirements:

⇒ Use a custom font from Google Fonts.

⇒ Adjust line-height, font-size, and spacing for improved readability.

**✅ Folder Structure**

pgsql

CopyEdit

blog-post/

├── index.html

└── style.css

**✅ 1. index.html**

html

CopyEdit

<!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>

<!-- Google Fonts -->

<link href="https://fonts.googleapis.com/css2?family=Playfair+Display:wght@700&family=Open+Sans&display=swap" rel="stylesheet" />

<!-- CSS -->

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

</head>

<body>

<div class="blog-container">

<h1 class="title">The Power of Simplicity in Web Design</h1>

<h2 class="subtitle">Why less is often more</h2>

<div class="content">

<p>

In the digital world, simplicity is not just a design trend — it's a powerful strategy. Clean, focused interfaces help users find what they need quickly and reduce cognitive load.

</p>

<p>

By using whitespace, readable fonts, and intuitive layouts, designers can enhance usability and accessibility. A minimalist approach not only improves the user experience but also makes websites faster and easier to maintain.

</p>

<p>

Always remember: simplicity isn't about removing features — it's about improving clarity and purpose.

</p>

</div>

</div>

</body>

</html>

**✅ 2. style.css**

css

CopyEdit

/\* Reset \*/

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

/\* Body styling \*/

body {

background-color: #f9f9f9;

font-family: 'Open Sans', sans-serif;

padding: 30px;

color: #333;

}

/\* Blog container \*/

.blog-container {

max-width: 800px;

margin: auto;

background: #fff;

padding: 40px;

border-radius: 10px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

/\* Title \*/

.title {

font-family: 'Playfair Display', serif;

font-size: 2.5rem;

margin-bottom: 15px;

color: #222;

}

/\* Subtitle \*/

.subtitle {

font-family: 'Open Sans', sans-serif;

font-size: 1.3rem;

color: #555;

margin-bottom: 25px;

}

/\* Content paragraph \*/

.content p {

font-size: 1.1rem;

line-height: 1.8;

margin-bottom: 20px;

}

/\* Responsive adjustments \*/

@media (max-width: 600px) {

.title {

font-size: 2rem;

}

.subtitle {

font-size: 1.1rem;

}

.content p {

font-size: 1rem;

line-height: 1.6;

}

.blog-container {

padding: 25px;

}

}

**✅ Requirements Checklist**

| **Requirement** | **✅ Completed** |
| --- | --- |
| Title, subtitle, body | ✅ Yes |
| Two different fonts | ✅ Playfair Display (headings) + Open Sans (body) |
| Responsive font sizes | ✅ Media queries for mobile |
| Custom Google Fonts | ✅ Loaded in <head> |
| Line-height, spacing | ✅ line-height: 1.8, margin-bottom: 20px |
| Easy readability | ✅ Clean layout, readable font sizes |