Module 3 – Mernstack – CSS and CSS3

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

A **CSS selector** is a pattern used to select and style HTML elements. It tells the browser **which HTML element(s)** the CSS rule should apply to.

**Element Selector**

* **Selects HTML tags directly.**
* **Syntax:** element { property: value; }

P{

Color :red;

Font-size:15px

}

**Class Selector**

* **Selects elements with a specific class attribute.**
* **Class selector start with “ .” symbol**
* **Syntax:** .classname { property: value; }

.card {

border: 1px solid black;

}

**ID Selector**

* **Selects a specific element with a unique ID.**
* **Id selector start with # symbol**
* **Syntax:** #idname { property: value; }

#header {

background-color: grey;}

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Question 3: What is the difference between internal, external, and inline CSS? Discuss theadvantages and disadvantages of each approach  1.inline css:  CSS is written directly within an HTML element’s style attribute.  <p style="color: red;">Hello</p>  Advantages:   * Quick and easy for single-use or testing. * Styles are applied immediately.   Disadvantages:   * Hard to maintain with many elements. * Not reusable – must repeat styles for each element. * Poor separation of HTML and CSS (not clean code). * Lower performance for larger projects.   2. Internal CSS  CSS is placed within a <style> tag inside the HTML file’s <head> section.  <head>  <style>  p {  color: green;  }  </style>  </head>  Advantages:   * Useful for styling a single HTML document. * Keeps styles in one place within the page. * Better structure than inline.   Disadvantages:   * Not reusable across multiple pages. * Increases HTML file size. * Slower load time for large websites with lots of internal styles.   External CSS  CSS is written in a separate .css file and linked to the HTML using <link> tag.  <head>  <link rel="stylesheet" href="style.css">  </head>  style.css:  css  CopyEdit  p {  color: blue;  }  Advantages:   * Best practice for large websites. * Keeps code clean and modular (separation of content and style). * Styles can be reused across multiple HTML pages. * Faster loading after first visit (browser caches CSS file).   Disadvantages:   * Requires extra HTTP request to fetch the CSS file. * Not ideal for small, one-page documents.   Task: 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.  Answer:  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <title>Contact Form</title>  <link rel="stylesheet" href="style.css">  </head>  <body>  <form id="contact-form">  <h2>Contact Us</h2>  <label for="name">Name:</label><br>  <input type="text" id="name" class="input-field"><br><br>  <label for="email">Email:</label><br>  <input type="email" id="email" class="input-field"><br><br>  <label for="message">Message:</label><br>  <textarea id="message" class="input-field"></textarea><br><br>  <button type="submit" id="submit-btn">Send Message</button>  </form>  </body>  </html>  External css file:  #contact-form {  background-color: #f2f2f2;  padding: 30px;  width: 400px;  margin: 50px auto;  border-radius: 8px;  box-shadow: 0 0 10px rgba(0,0,0,0.1);  }  .input-field {  width: 100%;  padding: 10px;  margin-top: 5px;  margin-bottom: 15px;  border: 1px solid #ccc;  border-radius: 5px;  box-sizing: border-box;  font-size: 16px;  }  #submit-btn {  background-color: #007BFF;  color: white;  padding: 12px 20px;  border: none;  border-radius: 6px;  cursor: pointer;  font-size: 16px;  transition: background-color 0.3s ease;  #submit-btn:hover {  background-color: #0056b3;  }  • Question 1: Explain the CSS box model and its components (content, padding, border,margin). How does each affect the size of an element?  The CSS Box Model is a fundamental concept that describes how every HTML element is structured and how its size is calculated in a web page layout.  **Padding**   * The **space between the content and the border**. * It increases the **space inside the element** without affecting other elements.   Example  padding: 20px;  **Border**   * A **line** surrounding the padding (and content). * Can be styled with thickness, color, and type.   Example:  border: 2px solid black;  **Margin**   * Margin create the space from outside of box or between two box, i.e., the space between this element and other elements.   Example:  margin: 30px;  **box-sizing: border-box**   * Padding and border are **included** inside the width and height. * Easier for layout control.   Example:  box-sizing: border-box;  What is the difference between border-boxand content-boxbox-sizing inCSS? Which is the default?  **content-box (Default)**   * **Only the content** is included in the width and height you set. * **Padding and border** are **added outside** the content box.   **border-box**   * **Content, padding, and border** are all **included** within the width and height. * The content area shrinks to fit padding and border.   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 module in CSS designed to **efficiently arrange items** within a container — even when their **size is unknown or dynamic**.  It allows you to create **flexible and responsive layouts** without using floats or complicated positioning.  **Why Flexbox is Useful:**   * Automatically adjusts spacing and alignment. * Makes it easier to design **one-dimensional layouts** (row OR column). * Align items **horizontally or vertically**. * Supports **wrapping**, **reordering**, and **spacing** between items easily. * Works great for both **small components (like navbars)** and **larger page sections**.   **Flex Container**  The **parent element** that has display: flex or display: inline-flex.  **Example:**  .container {  display: flex;  }  **Flex Items**  The **direct children** of the flex container. They are automatically laid out according to the flexbox rules.  **In the example above:**  <div class="item">Box 1</div> <!-- Flex Item -->  <div class="item">Box 2</div> <!-- Flex Item -->  Question 2: Describe the properties justify-content, align-items, and flex-direction used in Flexbox.  **flex-direction:** Sets the **direction of the main axis**, which determines how flex items are placed inside the container.   | **Value** | **Description** | | --- | --- | | row *(default)* | Items are placed **horizontally (left to right)** | | row-reverse | Items are placed **right to left** | | column | Items are placed **vertically (top to bottom)** | | column-reverse | Items are placed **bottom to top** |   **justify-content**  Aligns items **along the main axis** (horizontal in row, vertical in column).   | **Value** | **Description** | | --- | --- | | flex-start *(default)* | Items align at the start of the main axis | | flex-end | Items align at the end of the main axis | | center | Items are centered along the main axis | | space-between | Equal space **between** items | | space-around | Equal space **around** each item | | space-evenly | Equal space **between and around** items | | **align-items**  Aligns items **along the cross axis** (perpendicular to the main axis).   * If flex-direction: row → cross axis is **vertical** * If flex-direction: column → cross axis is **horizontal**  | **Value** | **Description** | | --- | --- | | stretch *(default)* | Items stretch to fill container on cross axis | | flex-start | Items align at the **start** of cross axis | | flex-end | Items align at the **end** of cross axis | | center | Items are **centered** along the cross axis | | baseline | Items align based on text baseline | |  |   **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**  **<!DOCTYPE html>**  **<html lang="en">**  **<head>**  **<meta charset="UTF-8">**  **<title>Flexbox Layout</title>**  **<meta name="viewport" content="width=device-width, initial-scale=1.0">**  **<link rel="stylesheet" href="style.css">**  **</head>**  **<body>**  **<header class="header">Header</header>**  **<div class="container">**  **<aside class="sidebar">Sidebar</aside>**  **<main class="main">Main Content</main>**  **</div>**  **<footer class="footer">Footer</footer>**  **</body>**  **</html>**  **/\* Reset basic styles \*/**  **\* {**  **box-sizing: border-box;**  **margin: 0;**  **padding: 0;**  **}**  **body {**  **font-family: Arial, sans-serif;**  **min-height: 100vh;**  **display: flex;**  **flex-direction: column;**  **}**  **/\* Header and Footer \*/**  **.header,**  **.footer {**  **background-color: #007BFF;**  **color: white;**  **padding: 15px;**  **text-align: center;**  **}**  **/\* Flex container for main layout \*/**  **.container {**  **display: flex;**  **flex: 1;**  **justify-content: space-between; /\* Spread sidebar and main \*/**  **align-items: stretch; /\* Stretch height of children \*/**  **flex-wrap: wrap;**  **}**  **/\* Sidebar \*/**  **.sidebar {**  **background-color: #f4f4f4;**  **padding: 20px;**  **flex: 1 1 200px; /\* Grow/shrink, minimum width \*/**  **min-height: 300px;**  **}**  **/\* Main Content \*/**  **.main {**  **background-color: #e9ecef;**  **padding: 20px;**  **flex: 3 1 400px;**  **min-height: 300px;**  **}**  **/\* Footer \*/**  **.footer {**  **background-color: #343a40;**  **color: white;**  **padding: 15px;**  **text-align: center;**  **}**  **/\* Responsive Layout \*/**  **@media (max-width: 768px) {**  **.container {**  **flex-direction: column;**  **justify-content: center;**  **align-items: stretch;**  **}**  **}**  • Question 1: Explain CSS Grid and how it differs from Flexbox. When would you use Grid overFlexbox?  **SS Grid** is a two-dimensional layout system in CSS that allows you to **design web layouts using rows and columns**. It gives you powerful tools to control both the **horizontal and vertical** positioning of elements within a container.  **CSS Grid vs. Flexbox**   | **Feature** | **Flexbox** | **CSS Grid** | | --- | --- | --- | | Layout Direction | One-dimensional (row **or** column) | Two-dimensional (row **and** column) | | Best for | Laying out **items in a line** | Creating **full page or grid layouts** | | Axis Control | Controls layout in **one axis** | Controls layout in **both axes** | | Content Flow | Items flow **based on content** | Items placed in **explicit grid areas** | | Item Placement | Auto-alignment, order-based | Precise **row/column placement** |   • Question 1: What are media queries in CSS, and why are they important for responsivedesign?  **Media queries** are CSS rules that apply styles **conditionally based on the device’s characteristics**, such as:   * Screen width * Screen height * Device type * Orientation * Resolution   body {  font-size: 18px;  background: white;  }  /\* Apply this style only for screens 768px or less \*/  @media (max-width: 768px) {  body {  font-size: 16px;  background: lightgray;  }  }  Question 2: Write a basic media query that adjusts the font size of a webpage for screenssmaller than 600px.  /\* Default font size for desktops and tablets \*/  body {  font-size: 18px;  }  /\* Media query for screens smaller than 600px \*/  @media (max-width: 600px) {  body {  font-size: 14px;  }  }  Question 1: Explain the difference between web-safe fonts and custom web fonts.Whymight you use a web-safe font over a custom font  **What are they?**  **Web-safe fonts** are a set of fonts that are **pre-installed on most operating systems** (Windows, macOS, Linux, etc.). This means they will display consistently across almost all browsers and devices **without requiring a download**.  **🔹 Common Web-Safe Fonts:**   | **Font Family** | **Examples** | | --- | --- | | Serif | Times New Roman, Georgia | | Sans-serif | Arial, Verdana, Tahoma | | Monospace | Courier New, Lucida Console |   **🟦 Custom Web Fonts**  **🔹 What are they?**  **Custom web fonts** are fonts that are **not guaranteed to exist** on a user's device. These fonts are **loaded over the web**, often using services like:   * **Google Fonts** * **Adobe Fonts (Typekit)** * Self-hosted .woff, .ttf, or .otf files   **🔹 Example (Google Font):**  html  CopyEdit  <link href="https://fonts.googleapis.com/css2?family=Roboto&display=swap" rel="stylesheet">  <style>  body {  font-family: 'Roboto', sans-serif;  }  </style>  **Key Differences**   | **Feature** | **Web-Safe Fonts** | **Custom Web Fonts** | | --- | --- | --- | | **Availability** | Built-in on all systems | Downloaded from the web | | **Performance** | Very fast (no loading required) | May slow page load slightly | | **Design Freedom** | Limited variety | Huge variety of styles & weights | | **Fallback Needed** | Rarely | Always recommended |   Question 2: What is the font-familyproperty in CSS? How do you apply a custom Google Font to a webpage?  **What is the font-family Property?**  The font-family property in CSS specifies the **typeface** (font) to be used for text on a webpage.  It lets you:   * Choose a **primary font** * List **fallback fonts** in case the first one is unavailable   **🔹 Syntax:**  selector {  font-family: "Font Name", fallback, generic-family;  body {  font-family: "Georgia", "Times New Roman", serif;  }  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.  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <title>My Blog Post</title>  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <!-- Google Fonts -->  <link href="https://fonts.googleapis.com/css2?family=Playfair+Display:wght@600&family=Open+Sans&display=swap" rel="stylesheet">  <!-- Link to CSS -->  <link rel="stylesheet" href="style.css">  </head>  <body>  <div class="blog-container">  <h1 class="blog-title">The Art of Simplicity</h1>  <h2 class="blog-subtitle">Why Less is Often More</h2>  <p class="blog-body">  In today's fast-paced digital world, simplicity has become a powerful design philosophy. Whether it's in architecture, writing, or web development, stripping away the unnecessary helps us focus on what truly matters.  </p>  <p class="blog-body">  Clean layouts, focused messaging, and deliberate use of space can enhance user experience and improve readability. As creators, embracing simplicity not only makes our work more effective but also more memorable.  </p>  </div>  </body>  </html>  /\* Base Styles \*/  \* {  margin: 0;  padding: 0;  box-sizing: border-box;  }  body {  background: #f5f5f5;  font-family: 'Open Sans', sans-serif;  padding: 20px;  }  /\* Blog Container \*/  .blog-container {  max-width: 800px;  margin: auto;  background: white;  padding: 40px;  border-radius: 8px;  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);  }  /\* Title \*/  .blog-title {  font-family: 'Playfair Display', serif;  font-size: 36px;  margin-bottom: 10px;  color: #222;  }  /\* Subtitle \*/  .blog-subtitle {  font-family: 'Playfair Display', serif;  font-size: 22px;  color: #555;  margin-bottom: 30px;  }  /\* Body \*/  .blog-body {  font-size: 18px;  line-height: 1.7;  color: #333;  margin-bottom: 20px;  }  /\* Responsive Text \*/  @media (max-width: 768px) {  .blog-title {  font-size: 28px;  }  .blog-subtitle {  font-size: 18px;  }  .blog-body {  font-size: 16px;  }  .blog-container {  padding: 20px;  }  } |  |