**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 and target HTML elements on a web page to apply styles to them. They allow developers to specify which elements or groups of elements should receive certain CSS rules.

Here are examples of common CSS selectors:

* **Element Selector:** This selector targets all instances of a specific HTML element type.
* **Code:**

p {

color: blue;

}

This rule will apply a blue color to the text of all <p> (paragraph) elements on the page.

* **Class Selector:** This selector targets all elements that have a specific class attribute assigned to them. Class names are prefixed with a period (.).
* **Code:**

.highlight {

background-color: yellow;

}

This rule will apply a yellow background color to any HTML element with class="highlight", such as <span class="highlight">Important!</span> or <div class="highlight">Section</div>.

* **ID Selector:** This selector targets a single, unique HTML element based on its id attribute. IDs are unique within a document and are prefixed with a hash symbol (#).
* **Code:**

#main-header {

font-size: 3em;

}

This rule will apply a font size of 3em to the single HTML element with id="main-header", for example, <h1 id="main-header">Welcome</h1>

Question 2:

Explain the concept of CSS specificity. How do conflicts between multiple styles get resolved?

Ans:

CSS specificity is a set of rules used by browsers to determine which CSS style declarations apply to an HTML element when multiple conflicting rules target the same element. It acts as a weighting system, where more specific selectors have a higher "score" and thus take precedence over less specific ones.

How Specificity is Calculated:

Specificity is calculated based on a hierarchy of selector types, often visualized as a four-column value (a, b, c, d):

* **Inline Styles (a):**

Styles applied directly to an HTML element using the style attribute have the highest specificity. They are assigned a value of 1 in the 'a' column (e.g., 1,0,0,0).

* **IDs (b):**

Selectors targeting an element by its id attribute are next in specificity. Each ID selector contributes 1 to the 'b' column (e.g., #myID would be 0,1,0,0).

* **Classes, Attributes, and Pseudo-classes (c):**

Selectors using classes (.myClass), attribute selectors ([type="text"]), or pseudo-classes (:hover, :first-child) contribute 1 to the 'c' column for each instance (e.g., .myClass:hover would be 0,0,2,0).

* **Elements and Pseudo-elements (d):**

Basic element selectors (div, p) and pseudo-elements (::before, ::after) have the lowest specificity. Each element or pseudo-element selector contributes 1 to the 'd' column (e.g., div p::before would be 0,0,0,3).

Question 3:

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

Ans:

CSS can be implemented in three main ways: inline, internal, and external. Each method has distinct characteristics and is suitable for different scenarios.

1. Inline CSS:

**Definition:** Styles are applied directly to individual HTML elements using the s"tyle attribute.

* **Code:**

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

* **Advantages:**
* Quick and easy for applying styles to a single element.
* Useful for testing or making quick, isolated changes.
* Overrides all other CSS types due to its high specificity.
* **Disadvantages:**
* Poor separation of concerns (mixing HTML and CSS).
* Difficult to maintain and update styles across multiple elements or pages.
* Increases HTML file size, potentially impacting performance.

2. Internal CSS:

**Definition**: Styles are defined within a <style> tag in the <head> section of an HTML document.

* **Code**:

<head>

<style>

h1 {

color: green;

}

p {

font-family: Arial, sans-serif;

}

</style>

</head>

* **Advantages:**
* Styles apply to the entire page, providing centralized control for a single document.
* Better separation of concerns than inline CSS.
* **Disadvantages:**
* Not reusable across multiple pages, leading to code duplication if styles are needed elsewhere.
* Increases the size of individual HTML files.

3. External CSS:

* **Definition:** Styles are stored in a separate .css file and linked to the HTML document using the <link> tag in the <head> section.
* **Code:**

<head>

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

</head>

(And in styles.css):

body {

background-color: lightgray;

}

h2 {

color: purple;

}

* **Advantages:**
* Excellent separation of concerns, keeping HTML and CSS entirely separate.
* Highly reusable: a single stylesheet can control the appearance of multiple pages.
* Improved maintainability and scalability for large websites.
* Faster page loading after the initial download, as CSS files are cached by the browser.
* **Disadvantages:**
* Requires an additional HTTP request to fetch the CSS file, potentially causing a brief "flash of unstyled content" (FOUC) on the initial load.
* Requires managing separate files

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?

The CSS Box Model is a fundamental concept in CSS that describes how every HTML element is rendered as a rectangular box. This model defines the structure and dimensions of an element, including its content, padding, border, and margin.

Components of the CSS Box Model:

* **Content:**

This is the innermost area of the box, where the actual content of the element (text, images, videos, etc.) resides. The width and height CSS properties directly control the dimensions of the content area.

* **Padding:**

This is the transparent space between the content area and the element's border. Padding adds internal spacing within the box, pushing the content away from the border. It can be controlled using properties like padding-top, padding-right, padding-bottom, and padding-left, or the shorthand padding. Padding increases the overall visible size of the element.

* **Border:**

This is a line that surrounds the padding and content area, defining the visual boundary of the element. Borders can be styled with various widths, styles (e.g., solid, dashed), and colors using properties like border-width, border-style, and border-color, or the shorthand border. The border's thickness adds to the element's total size.

* **Margin:**

This is the transparent space outside the element's border, separating it from neighboring elements. Margins are used to control the spacing and layout between different elements on a page. Like padding, margins can be controlled individually for each side (e.g., margin-top, margin-left) or with the margin shorthand. Margins add to the total space an element occupies on the page, but they do not increase the actual size of the element itself (the box's dimensions stop at the border).

How Each Component Affects Element Size:

* **Content:**

Directly determines the base dimensions of the element's visible content area.

* Padding:

Adds space inside the element, increasing its overall visible size (content + padding).

* Border:

Adds a visual line around the padding and content, further increasing the element's visible size (content + padding + border).

* Margin:

Adds space outside the element, affecting the spacing between elements on the page but not the intrinsic size of the element itself. The total space an element occupies on the page includes its margins.

Calculation of Total Element Size (Default box-sizing: content-box):

Total Width:

width (of content) + left-padding + right-padding + left-border + right-border

Total Height:

height (of content) + top-padding + bottom-padding + top-border + bottom-border

Note that the margin is not included in these calculations for the element's actual size, but it does contribute to the overall space the element occupies within the document flow. When box-sizing: border-box is used, the width and height properties include padding and border within their specified values.

Question 2:

What is the difference between border-box and content-box box-sizing in CSS? Which is the default?

Ans:

The CSS box-sizing property determines how an element's total width and height are calculated, specifically in relation to its content, padding, and border.

**1. content-box:**

* **Default behavior:** This is the default value for most HTML elements.
* **Calculation:** When box-sizing: content-box is applied, the width and height properties you set for an element refer only to the content area. Any padding and border are then added on top of this specified width and height, increasing the element's total occupied space.
* **Example:** If you set width: 100px, padding: 10px, and border: 5px, the total width of the element will be 100px (content) + 2 \* 10px (padding) + 2 \* 5px (border) = 130px.

**2. border-box:**

* **Calculation:**

When box-sizing: border-box is applied, the width and height properties you set for an element refer to the entire box, including the content, padding, and border. The content area then shrinks to accommodate the padding and border within the specified width and height.

* **Example:**

If you set width: 100px, padding: 10px, and border: 5px, the total width of the element will remain 100px. The content area will be reduced to 100px - 2 \* 10px (padding) - 2 \* 5px (border) = 70px.

The default value for box-sizing is content-box. This means that unless you explicitly set box-sizing: border-box; or use a CSS reset that applies it universally, elements will behave according to the content-box model.

What is CSS Flexbox, and how is it useful for layout design? Explain the terms

flex-container and flex-item.

**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, or the Flexible Box Layout module, is a one-dimensional CSS layout model designed to provide an efficient way to lay out, align, and distribute space among items in a container, even when their size is unknown or dynamic. It is particularly useful for building flexible and responsive user interface components and sections, such as navigation bars, forms, and image galleries.

* **Responsive Design:**
* Flexbox makes it easy to create layouts that adapt to different screen sizes and orientations without complex media queries or JavaScript.
* **Alignment and Distribution:**
* It offers powerful properties for aligning items along both the main and cross axes, as well as distributing space between or around them.
* **Dynamic Content:**
* Flexbox handles situations where the size of content is unknown or changes, allowing items to adjust their size accordingly.
* **Simplified Ordering:**
* The order of items can be easily changed, even on smaller screens, using properties like order.
* Key Terms:
* **Flex-Container:** This is the parent element that holds the flex items. To make an element a flex-container, its display property must be set to flex or inline-flex. All direct children of the flex-container become flex-items and are subject to flexbox layout properties.
* **Code**:

.flex-container {

display: flex;

}

* **Flex-Item:** These are the direct children of a flex-container. They are the individual elements within the flexbox layout that are arranged, aligned, and sized according to the flexbox properties applied to their parent container or to themselves.
* **Code:**

<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>

Question 2:

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

Ans:

In Flexbox, justify-content, align-items, and flex-direction are key properties for controlling the layout of items within a flex container. justify-content aligns items along the main axis (horizontal by default), while align-items aligns them along the cross axis (vertical by default). flex-direction determines the direction of the main axis.

**1. justify-content:**

* **Purpose:**

Aligns flex items along the main axis (horizontal by default, or vertical if flex-direction is column).

* **Common Values:**
* flex-start: Items are packed toward the start of the container.
* flex-end: Items are packed toward the end of the container.
* center: Items are centered along the main axis.
* space-between: Items are evenly distributed in the line, with the first item at the start and the last item at the end.
* space-around: Items are evenly distributed in the line with equal space around them.
* space-evenly: Items are distributed so that the spacing between any two adjacent items, before the first item, and after the last item is the same.

**Example:**

If flex-direction is row (default), justify-content: center will horizontally center the items. If flex-direction is column, it will vertically center the items.

**2. align-items:**

* **Purpose:** Aligns flex items along the cross axis (perpendicular to the main axis).
* **Common Values:**
* stretch: (default) Items are stretched to fill the container along the cross axis.
* flex-start: Items are aligned to the start of the cross axis.
* flex-end: Items are aligned to the end of the cross axis.
* center: Items are centered along the cross axis.
* baseline: Items are aligned such that their baselines align.

**Example:** If flex-direction is row, align-items: center will vertically center the items. If flex-direction is column, it will horizontally center the items.

**3. flex-direction:**

* **Purpose:** Defines the direction of the main axis.
* **Values:**
* row (default): Items are placed horizontally from left to right.
* row-reverse: Items are placed horizontally from right to left.
* column: Items are placed vertically from top to bottom.
* column-reverse: Items are placed vertically from bottom to top.

**Example:** If flex-direction is column, the main axis becomes vertical, and justify-content will align items vertically, while align-items will align them horizontally.

**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:**
* CSS Grid is a **two-dimensional layout system**.
* It helps you design web layouts by aligning content in **rows and columns**.
* You define a grid **container** and place **grid items** inside it.
* Example use: Creating full-page layouts, image galleries, or dashboards.
* **Flexbox:**
* Flexbox is a **one-dimensional layout system**.
* It works along a **single axis**: **row or column**.
* Best for **aligning items in a line** or adjusting spacing.

Example use: Navigation bars, toolbars, cards in a row, etc.

* **Main Differences:**

|  |  |  |
| --- | --- | --- |
| Feature | **CSS Grid** | **Flexbox** |
| Layout direction | 2D (rows and columns) | 1D (row **or** column) |
| Alignment control | Precise control in both axes | One axis at a time |
| Content size | Works well with full layout | Better for small UI sections |
| Positioning | Items placed by grid coordinates | Items placed in order |

* **When to Use Grid:**
  + Complex layouts with **rows and columns**.
  + Designing whole web pages or sections.
  + Need for **precise control** over both directions.
* **When to Use Flexbox:**
  + Layouts in a **single row or column**.
  + Distributing space between items.
  + Simple alignments like nav bars, buttons, forms.

Question 2:

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

Ans:

The grid-template-columns, grid-template-rows, and grid-gap properties are fundamental to CSS Grid Layout, enabling control over the structure and spacing of grid items within a containe.

**1.grid-template-columns:**

This property defines the number and size of columns in a grid. Values can be specified using lengths (e.g., px, em, rem), percentages, the fr unit (fractional unit, distributing available space proportionally), or keywords like auto or minmax(). example.

**Code:**

.container {

display: grid;

grid-template-columns: 100px 1fr 20%; /\* First column 100px, second takes 1 fraction of remaining space, third 20% of container width \*/

}

**2. grid-template-rows:**

Similar to grid-template-columns, this property defines the number and size of rows in a grid. It accepts the same types of values. example.

**Code**:

.container {

display: grid;

grid-template-rows: auto 50px 1fr; /\* First row sized automatically, second 50px, third takes 1 fraction of remaining space \*/

}

**3. grid-gap (or gap):**

This property is a shorthand for row-gap and column-gap, defining the space between grid rows and columns. It accepts one or two values: if one value is provided, it applies to both row and column gaps; if two values are provided, the first sets the row-gap and the second sets the column-gap. example.

**Code:**

.container {

display: grid;

grid-gap: 20px; /\* 20px gap between both rows and columns \*/

}

.another-container {

display: 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:

Media queries in CSS are a feature that allows the application of specific CSS styles based on the characteristics of the device or viewport displaying a web page. These characteristics can include screen width, height, orientation (portrait or landscape), resolution, and even user preferences like preferred color scheme (light or dark mode).

The core of a media query is the @media rule, which contains a media type (e.g., screen, print) and one or more media features (e.g., min-width, orientation). Styles defined within a media query block are only applied if the specified conditions are met.

Example:

@media screen and (max-width: 768px) {

/\* Styles applied only when the screen width is 768px or less \*/

body {

font-size: 14px;

}

.navigation {

flex-direction: column;

}

}

Importance for Responsive Design:

Media queries are fundamental to responsive web design because they enable websites to adapt their layout and appearance dynamically to provide an optimal viewing experience across a wide range of devices and screen sizes. Without media queries, a website designed for a large desktop monitor might appear unreadable or unusable on a smaller mobile device, requiring users to zoom and scroll excessively. By using media queries, developers can:

* **Adjust layouts:**

Change the number of columns, element positioning, and overall page structure to suit different screen dimensions.

* **Optimize content display:**

Alter font sizes, image sizes, and spacing to ensure readability and visual appeal on various devices.

* **Enhance user experience:**

Provide tailored interactions, such as larger touch targets for mobile devices or simplified navigation menus for smaller screens.

* **Improve accessibility:**

Adapt styles based on user preferences, like high contrast modes or reduced motion settings.

In essence, media queries are the mechanism that allows a single codebase to serve multiple device types effectively, ensuring a consistent and user-friendly experience regardless of how a user accesses the website.

Question 2:

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

Ans:

A basic media query to adjust the font size of a webpage for screens smaller than 600px can be implemented in CSS.

**Code:**

body {

font-size: 16px; /\* Default font size for larger screens \*/

}

@media only screen and (max-width: 600px) {

body {

font-size: 14px; /\* Adjust font size for screens 600px or smaller \*/

}

}

In this example:

body { font-size: 16px; } sets a default font size for the body element, which will apply to screens wider than 600px.

@media only screen and (max-width: 600px) is the media query rule.

only screen specifies that these styles should apply only to screen media types.

(max-width: 600px) indicates that the enclosed styles will be applied when the viewport width is 600 pixels or less.

Inside the media query, body { font-size: 14px; } overrides the default font size for screens meeting the specified condition, making the text smaller on smaller screens.

**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:

### **Difference Between Web-Safe Fonts and Custom Web Fonts**

### **1. Web-Safe Fonts**

* **Definition**: Fonts that are pre-installed on most operating systems (Windows, macOS, Linux).
* **Examples**: Arial, Times New Roman, Verdana, Courier New, Georgia, Trebuchet MS.
* **How they work**: The browser uses the font already installed on the user’s device.
* **No extra loading time**: Since the font is already on the system, it loads instantly.

### **2. Custom Web Fonts**

* **Definition**: Fonts that are not installed on the user’s device, and are loaded from the web (often via a service like Google Fonts).
* **Examples**: Roboto, Open Sans, Lato, Montserrat (via Google Fonts or Adobe Fonts).
* **How they work**: The browser downloads the font file from a server when the page loads.
* **Requires internet**: Slower to load, especially on poor connections.

|  |  |
| --- | --- |
| Reason | Explanation |
| **Faster performance** | No download needed – improves page speed. |
| **Better compatibility** | Works on all browsers and devices without fallback. |
| **Offline support** | Works even without an internet connection. |
| **Simplicity** | No need to manage font hosting or links. |

### **Use Web-Safe Fonts When:**

* You want **fast-loading** pages.
* Your project is **simple or minimal**.
* You don’t want to worry about **extra font loading or licensing**.

### **Use Custom Fonts When:**

* You want **unique branding or design**.
* You need a specific **style not available in standard fonts**.
* You're okay with slight performance trade-offs.

Question 2:

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

Ans:

The font-family property in CSS is used to specify the font or a prioritized list of fonts (a "font stack") that a browser should try to use for displaying text within an HTML element.

Here's how it works:

1. Prioritized List (Font Stack): You provide a comma-separated list of font names, ordered by preference.
2. Browser Evaluation: The browser starts at the beginning of the list and attempts to find the first specified font on the user's system or through embedded font resources (like @font-face rules or imported web fonts).
3. Fallback Mechanism: If the browser cannot locate the first font, it moves to the next font in the list and continues this process until it finds a font it can use.
4. Generic Font Family: It's best practice to end your font-family list with a generic font family (like serif, sans-serif, monospace, cursive, or fantasy). This ensures that even if none of your specified fonts are available, the browser will still display the text using a similar generic font.
5. Quotation Marks: If a font name has spaces, it must be enclosed in quotation marks, such as "Times New Roman".

**Example:**

p {

font-family: "Source Sans Pro", Arial, sans-serif;

}

How to apply a custom Google Font to a webpage

Here's a step-by-step guide to applying a custom Google Font to your webpage:

1. Visit Google Fonts: Go to [fonts.google.com](https://fonts.google.com/).
2. Select your desired font: Browse or search for the font you want to use and click on it.
3. Choose styles and weights: On the font's page, select the specific styles (e.g., regular, italic) and weights (e.g., 400, 700) you want to use.
   * Note: Only select the styles and weights you truly need, as loading more variations can affect page load times, [says www.munnelly.com](https://www.munnelly.com/webdev/exercises/working-with-google-fonts).
4. Get the embed code: After selecting your styles, a "Embed" option will appear. Click it to reveal the code for embedding the font.
5. Copy the <link> tag: In the Embed tab, copy the provided <link> tag. It will look something like this:

html

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

6.Paste the link in your HTML: Paste this <link> tag into the <head> section of your HTML document.

7.Apply the font in CSS: In your CSS file, use the font-family property to apply the new Google Font to the desired element(s).

* + Example:

css

body {

font-family: 'Your Font Name', sans-serif;

}