Day 8

1. CSS Selectors: Selectors are patterns used to select the elements you want to style.

Basic Selectors:

Element Selector: Selects all elements of a given type.

css

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

Class Selector: Selects elements with a specific class attribute.

css

```
.my-class { font-size: 14px; }
```

ID Selector: Selects an element with a specific ID attribute.

css

```
#my-id { text-align: center; }
```

Attribute Selectors: Select elements based on an attribute or attribute value.

css

```
a[href] { color: green; } a[target="_blank"] { font-weight: bold; }
```

Combinator Selectors:

Descendant Selector: Selects all elements that are descendants of a specified element.

css

```
div p { color: red; }
```

Child Selector: Selects all elements that are direct children of a specified element.

css

```
div > p { font-size: 18px; }
```

Adjacent Sibling Selector: Selects an element that is the next sibling of a specified element.

css

```
h1 + p { margin-top: 20px; }
```

General Sibling Selector: Selects all siblings of a specified element.

css

```
h1 ~ p { color: gray; }
```

Pseudo-class Selectors: Apply styles to elements based on their state.

css

```
a:hover { color: red; } input:focus { border: 2px solid blue; }
```

Pseudo-element Selectors: Apply styles to a part of an element.

css

```
p::first-line { font-weight: bold; } p::before { content: "Note: "; font-weight: bold; }
```

2. CSS Properties: Properties define the styles applied to the selected elements.

Text Properties:

`color`: Sets the color of the text.

Css

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

`font-size`: Sets the size of the font.

css

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

`text-align`: Aligns the text inside an element.

css

```
h1 { text-align: center; }
```

Box Model Properties:

`width` and `height`: Set the width and height of an element.

css

```
div { width: 100px; height: 50px; }
```

`padding`: Adds space inside the element, around the content.

css

```
div { padding: 10px; }
```

`margin`: Adds space outside the element, around the border.

css

```
div { margin: 20px; }
```

`border`: Sets the border around an element.

```css
div { border: 1px solid black; }
```

Background Properties:

`background-color`: Sets the background color of an element.

```css
body { background-color: #f0f0f0; }
```

`background-image`: Sets a background image for an element.

```css
div { background-image: url('image.jpg'); }
```

Display and Positioning Properties:

`display`: Specifies the display behavior of an element.

```css
.hidden { display: none; }
```

`position`: Specifies the positioning method used for an element (static, relative, absolute, fixed, sticky).

```css
.absolute { position: absolute; top: 50px; left: 50px; }
```

Flexbox Properties:

`display: flex`: Defines a flex container and enables a flex context for all its direct children.

```css
.container { display: flex; }
```

`justify-content`: Aligns flex items along the main axis.

```css
.container { justify-content: center; }
```

`align-items`: Aligns flex items along the cross axis.

```css
.container { align-items: center; }
```

3. Practical Examples:

html

```
<!DOCTYPE html>
<html lang="en">
<head> <meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>CSS Selectors and Properties</title>
<style> /* Element Selector */ p { color: blue; font-size: 14px; }
 /* Class Selector */ .highlight { background-color: yellow; } /* ID Selector */ #unique { font-weight: bold; text-align: center; } /* Attribute Selector */ a[href^="https"] { color: green; } /* Descendant Selector *
/ div p { margin-left: 20px; } /* Child Selector */
ul > li { list-style-type: square; } /* Adjacent Sibling Selector */ h1 + p { font-style: italic; } /* General Sibling Selector */ h1 ~ p { color: gray; } /* Pseudo-class Selector */ a:hover { text-decoration: underline; }
 /* Pseudo-element Selector */ p::first-letter { font-size: 20px; color: red; } </style>
</head>
<body> <h1>This is a heading</h1>
<p>This is a paragraph.</p> <p class="highlight">This is a highlighted paragraph.</p>
<p id="unique">This is a unique paragraph.</p>
<a href="https://example.com">This is a link.</a>
<div> <p>This is a paragraph inside a div.</p> </div>
<ul> <li>List item 1</li> <li>List item 2</li>
</ul> <a href="https://example.com">Hover over this link.</a>
</body>
</html>
```

Reflection: Today's session on CSS selectors and properties was extremely insightful. Understanding how to select and style elements effectively using various selectors allows for more precise and efficient CSS coding. The properties we learned about will help me control the layout, appearance, and behavior of elements on the web page. I feel more confident in my ability to style web pages and look forward to practicing these new skills.

Daily Diary - Web Development Training

Day 9

CSS Box Model and Fluid Layouts

Summary of the Day: On the tenth day of our web development training, we explored two important concepts in CSS: the CSS Box Model and fluid layouts. Understanding these topics is essential for creating well-structured and responsive web pages. The session covered the components of the CSS Box Model and techniques for designing fluid, flexible layouts.

Detailed Notes:

1. CSS Box Model: The CSS Box Model is a fundamental concept that describes how elements are structured and spaced on a web page.

Components of the Box Model:

Content: The actual content of the element, such as text or an image.

Padding: The space between the content and the border. It increases the size of the element without affecting its external dimensions.

Border: A line surrounding the padding (if any) and content.

Margin: The space outside the border, separating the element from other elements on the page.

Visual Representation:

css

```
element { width: 100px; height: 100px; padding: 10px; border: 5px solid black; margin: 15px; }
```

This would result in:

Content: 100px x 100px

Padding: 10px on all sides (total size becomes 120px x 120px)

Border: 5px on all sides (total size becomes 130px x 130px)

Margin: 15px on all sides (total space occupied becomes 160px x 160px)

Example:

Html

```
<style> .box { width: 100px; height: 100px; padding: 10px; border: 5px solid black; margin: 15px; background-color: lightblue; } </style> <div class="box">Box Model Example</div>
```

2. Fluid Layouts: Fluid layouts, also known as liquid layouts, adapt to the size of the user's viewport, making web pages more responsive.

Percentage-Based Widths: Using percentages allows elements to resize relative to their parent container.

css

```
.container { width: 80%; /* 80% of the parent container's width */ margin: 0 auto; /* Center the container */ }
```

Viewport Units: Viewport units (`vw` and `vh`) are relative to the size of the viewport.

`1vw` is 1% of the viewport width.

`1vh` is 1% of the viewport height.

css

```
.responsive-box { width: 50vw; /* 50% of the viewport width */ height: 50vh; /* 50% of the viewport height */ background-color: lightgreen; }
```

Flexbox: Flexbox is a powerful layout module that allows for the creation of flexible and responsive layouts.

css

```
.flex-container { display: flex; flex-wrap: wrap; justify-content: space-around; } .flex-item { flex: 1 1 auto; margin: 10px; background-color: lightcoral; }
```

Example:

html

```
<style> .flex-container { display: flex; flex-wrap: wrap; justify-content: space-around; } .flex-item { flex: 1 1 auto; margin: 10px; background-color: lightcoral; padding: 20px; text-align: center; } </style> <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>
```

3. Practical Examples:

Example with Box Model:

html

```
<!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0"> <title>Box Model Example</title> <style> .box { width: 200px; padding: 20px; border: 5px solid black; margin: 15px; background-color: lightblue; } </style> </head> <body> <div class="box">This is an example of the box model.</div> </body> </html>
```

Example with Fluid Layout:

html

```
<!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0"> <title>Fluid Layout Example</title> <style> .container { width: 80%; margin: 0 auto; background-color: lightgray; padding: 20px; } .responsive-box { width: 50vw; height: 50vh; background-color: lightgreen; margin: 20px 0; } .flex-container { display: flex; flex-wrap: wrap; justify-content: space-around; } .flex-item { flex: 1 1 200px; margin: 10px; background-color: lightcoral; padding: 20px; text-align: center; } </style> </head> <body> <div class="container"> <h1>Fluid Layout Example</h1> <div class="responsive-box">Responsive
```

```
Box</div> <div class="flex-container"> <div class="flex-item">Flex Item 1</div> <div class="flex-item">Flex Item 2</div> <div class="flex-item">Flex Item 3</div> </div> </div> </body> </html>
```

Reflection: Today's session on the CSS Box Model and fluid layouts was enlightening. Understanding the box model helps in accurately controlling the spacing and sizing of elements, which is essential for creating well-structured web pages. Learning about fluid layouts, especially with techniques like percentage-based widths and Flexbox, has equipped me with the skills to design responsive, flexible web pages that adapt to different screen sizes. I look forward to applying these concepts in my future projects.

Daily Diary - Web Development Training

Day 11

CSS Layouts

Summary of the Day: On the eleventh day of our web development training, we focused on CSS layouts. Understanding different layout techniques in CSS is crucial for designing visually appealing and responsive web pages. The session covered various layout models, including block, inline, inline-block, and modern layout techniques such as Flexbox and CSS Grid.

Detailed Notes:

1. CSS Layout Basics:

Block Layout:

Block-level elements occupy the full width of their container and start on a new line.

Examples: `<div>`, `<p>`, `<h1>`, `<section>`

Properties:

```
css

div { display: block; width: 100%; }
```

Inline Layout:

Inline elements do not start on a new line and only occupy as much width as necessary.

Examples: `<span>`, `<a>`, `<strong>`

Properties:

```
css

a { display: inline; }
```

Inline-Block Layout:

Inline-block elements are similar to inline elements but can have width and height set.

Examples: `<img>`, `<button>`

Properties:

css

```
.inline-block { display: inline-block; width: 100px; height: 50px; }
```

2. Modern Layout Techniques:

Flexbox:

Flexbox is designed for one-dimensional layouts. It allows items to align and distribute space within a container.

Properties:

css

```
.flex-container { display: flex; justify-content: space-between; /* Align items horizontally */ align-items: center; /* Align items vertically */ } .flex-item { flex: 1; /* Grow items to fill available space */ margin: 10px; }
```

Example:

html

```
<style> .flex-container { display: flex; justify-content: space-between; align-items: center; background-color: lightgray; padding: 20px; } .flex-item { flex: 1; margin: 10px; background-color: lightcoral; text-align: center; padding: 20px; } </style> <div class="flex-container"> <div class="flex-item">Item 1</div> <div class="flex-item">Item 2</div> <div class="flex-item">Item 3</div> </div>
```

CSS Grid:

CSS Grid is a two-dimensional layout system that allows for both rows and columns.

Properties:

css

```
.grid-container { display: grid; grid-template-columns: repeat(3, 1fr); /* Three equal columns */ grid-gap: 10px; /* Gap between items */ } .grid-item { background-color: lightblue; text-align: center; padding: 20px; }
```

Example:

html

```
<style> .grid-container { display: grid; grid-template-columns: repeat(3, 1fr); grid-gap: 10px; } .grid-item { background-color: lightblue; text-align: center; padding: 20px; } </style> <div class="grid-container"> <div class="grid-item">Item 1</div> <div class="grid-item">Item 2</div> <div class="grid-item">Item 3</div> <div class="grid-item">Item 4</div> <div class="grid-item">Item 5</div> <div class="grid-item">Item 6</div> </div>
```

3. Positioning Techniques:

Static Positioning:

Default positioning of elements.

Example:

css

```
.static { position: static; }
```

Relative Positioning:

Positioned relative to its normal position.

Example:

css

```
.relative { position: relative; top: 10px; left: 20px; }
```

Absolute Positioning:

Positioned relative to its nearest positioned ancestor.

Example:

css

```
.absolute { position: absolute; top: 50px; left: 50px; }
```

Fixed Positioning:

Positioned relative to the browser window.

Example:

css

```
.fixed { position: fixed; bottom: 0; width: 100%; background-color: lightgray; }
```

Sticky Positioning:

Switches between relative and fixed positioning based on the user's scroll position.

Example:

css

```
.sticky { position: -webkit-sticky; /* For Safari */ position: sticky; top: 0; background-color: yellow;
}
```

Reflection: Today's session on CSS layouts provided a comprehensive understanding of how to structure and position elements on a web page. The introduction to Flexbox and CSS Grid was particularly valuable, as these modern layout techniques offer powerful ways to create responsive and flexible designs. The positioning techniques further enhanced my

ability to control the placement of elements in various contexts. I look forward to applying these concepts to build more complex and visually appealing web pages.

Daily Diary - Web Development Training

Day 12

Flexbox

Summary of the Day: On the twelfth day of our web development training, we had an in-depth session on Flexbox. Flexbox, or the Flexible Box Layout, is a powerful CSS module that allows for the creation of responsive and flexible layouts. It simplifies the process of aligning and distributing space among items in a container, even when their size is unknown or dynamic.

Detailed Notes:

1. Introduction to Flexbox:

Flexbox is designed for one-dimensional layouts, either in a row or a column.

It consists of a flex container and flex items.

2. Flex Container Properties:

display: flex; Defines a flex container and enables flex context for all its direct children.

css

```
.flex-container { display: flex; }
```

flex-direction: Specifies the direction of the flex items.

css

```
.flex-container { flex-direction: row; /* Default */ } /* Other values: row-reverse, column, column-reverse */
```

flex-wrap: Determines whether flex items should wrap or not.

css

```
.flex-container { flex-wrap: nowrap; /* Default */ } /* Other values: wrap, wrap-reverse */
```

flex-flow: A shorthand for setting both `flex-direction` and `flex-wrap`.

css

```
.flex-container { flex-flow: row wrap; }
```

justify-content: Aligns flex items along the main axis.

css

```
.flex-container { justify-content: flex-start; /* Default */ } /* Other values: flex-end, center, space-between, space-around, space-evenly */
```

align-items: Aligns flex items along the cross axis.

css

```
.flex-container { align-items: stretch; /* Default */ } /* Other values: flex-start, flex-end, center, baseline */
```

align-content: Aligns flex lines when there is extra space in the cross axis.

css

```
.flex-container { align-content: stretch; /* Default */ } /* Other values: flex-start, flex-end, center, space-between, space-around */
```

3. Flex Item Properties:

order: Controls the order of the flex items.

css

```
.flex-item { order: 1; /* Default is 0 */ }
```

flex-grow: Specifies how much a flex item will grow relative to the rest.

css

```
.flex-item { flex-grow: 1; /* Default is 0 */ }
```

flex-shrink: Specifies how much a flex item will shrink relative to the rest.

css

```
.flex-item { flex-shrink: 1; /* Default */ }
```

flex-basis: Defines the initial size of a flex item.

css

```
.flex-item { flex-basis: 100px; /* Default is auto */ }
```

flex: A shorthand for `flex-grow`, `flex-shrink`, and `flex-basis`.

css

```
.flex-item { flex: 1 1 100px; }
```

align-self: Allows the default alignment (or the one specified by `align-items`) to be overridden for individual flex items.

css

```
.flex-item { align-self: auto; /* Default */ } /* Other values: flex-start, flex-end, center, baseline, stretch */
```

4. Practical Examples:

Example of Flex Container and Items:

html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Flexbox Example</title>
<style> .flex-container { display: flex; flex-direction: row; justify-content: space-around; align-items: center; background-color: lightgray; height: 200px; } .flex-item { background-color: lightcoral; padding: 20px; margin: 10px; text-align: center; flex: 1; } </style>
</head>
<body>
<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>
</body>
</html>
```

Example with Flex Properties:

html

```
<!DOCTYPE html>
<html lang="en"> <head> <meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Flexbox Properties</title>
<style> .flex-container { display: flex; flex-direction: column; flex-wrap: wrap; justify-content: center; align-items: flex-start; align-content: space-between; height: 300px; background-color: lightblue; } .flex-item { background-color: lightgreen; margin: 10px; padding: 20px; text-align: center; order: 2; flex: 1 1 100px; align-self: center; } .flex-item:first-child { order: 1; flex: 2 1 150px; } </style>
</head>
<body>
<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 class="flex-item">Item 4</div> </div> </body> </html>
```

Reflection: Today's session on Flexbox was incredibly informative. The flexibility and simplicity that Flexbox offers for creating responsive layouts are impressive. Learning about the various properties and how they interact allowed me to understand the full potential of Flexbox in modern web design. I feel confident in using Flexbox to build responsive, efficient, and well-structured web layouts.

Daily Diary - Web Development Training

Day 13

CSS Grid

Summary of the Day: On the thirteenth day of our web development training, we delved into CSS Grid. CSS Grid is a powerful two-dimensional layout system that allows developers to create complex and responsive grid-based layouts with ease. The session focused on understanding the basics of CSS Grid, including container properties and grid item properties.

Detailed Notes:

1. Introduction to CSS Grid: CSS Grid is designed to handle both rows and columns, providing a more robust layout system compared to Flexbox, which is primarily one-dimensional.

2. CSS Grid Container Properties: The following properties are used to define a grid container and specify its behavior.

display: grid; Turns an element into a grid container.

css

```
.grid-container { display: grid; }
```

grid-template-columns and grid-template-rows: Define the number and size of the columns and rows in the grid.

css

```
.grid-container { grid-template-columns: 1fr 1fr 1fr; /* Three equal columns */ grid-template-rows: 100px 200px; /* Two rows with specific heights */ }
```

grid-template-areas: Names grid areas for easier reference and layout.

css

```
.grid-container { grid-template-areas: 'header header header' 'sidebar content content' 'footer footer footer'; }
```

grid-gap (or gap): Sets the spacing between rows and columns.

css

```
.grid-container { grid-gap: 10px; /* Space between rows and columns */ }
```

grid-auto-flow: Controls how auto-placed items are inserted into the grid.

css

```
.grid-container { grid-auto-flow: row; /* Default */ } /* Other values: column, dense, row dense, column dense */
```

3. CSS Grid Item Properties: The following properties are used to define the placement and behavior of grid items within a grid container.

grid-column-start, grid-column-end, grid-row-start, grid-row-end: Specify the start and end positions of a grid item.

css

```
.grid-item { grid-column-start: 1; grid-column-end: 3; /* Span across the first two columns */ grid-row-start: 1; grid-row-end: 2; /* Span across the first row */ }
```

grid-column and grid-row: Shorthand for setting both the start and end positions.

css

```
.grid-item { grid-column: 1 / 3; /* Span across the first two columns */ grid-row: 1 / 2; /* Span across the first row */ }
```

grid-area: Assigns an item to a named grid area.

css

```
.grid-item { grid-area: header; }
```

justify-self: Aligns the grid item within its column.

css

```
.grid-item { justify-self: center; /* Default is stretch */ } /* Other values: start, end, stretch */
```

align-self: Aligns the grid item within its row.

css

```
.grid-item { align-self: center; /* Default is stretch */ } /* Other values: start, end, stretch */
```

4. Practical Examples:

Basic Grid Layout:

html

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

```
<title>CSS Grid Example</title>

<style> .grid-container { display: grid; grid-template-columns: 1fr 1fr 1fr; grid-template-rows: 100px 100px; grid-gap: 10px; background-color: lightgray; } .grid-item { background-color: lightcoral; text-align: center; padding: 20px; } </style>

</head>

<body>

<div class="grid-container"> <div class="grid-item">Item 1</div> <div class="grid-item">Item 2</div> <div class="grid-item">Item 3</div> <div class="grid-item">Item 4</div> <div class="grid-item">Item 5</div> <div class="grid-item">Item 6</div> </div>

</body>


</html>
```

Grid Layout with Named Areas:

html

```
<!DOCTYPE html>

<html lang="en"> <head>

<meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>CSS Grid Named Areas</title>

<style> .grid-container { display: grid; grid-template-areas: 'header header header' 'sidebar content content' 'footer footer footer'; grid-gap: 10px; background-color: lightgray; } .header { grid-area: header; background-color: lightblue; text-align: center; padding: 20px; } .sidebar { grid-area: sidebar; background-color: lightgreen; text-align: center; padding: 20px; } .content { grid-area: content; background-color: lightcoral; text-align: center; padding: 20px; } .footer { grid-area: footer; background-color: lightgoldenrodyellow; text-align: center; padding: 20px; } </style>

</head>

<body>

<div class="grid-container"> <div class="header">Header</div> <div class="sidebar">Sidebar</div> <div class="content">Content</div> <div class="footer">Footer</div> </div>

</body>

</html>
```

Reflection: Today's session on CSS Grid was enlightening. Learning about the container and item properties of CSS Grid has opened up new possibilities for creating complex, responsive web layouts. The ability to control both rows and columns simultaneously makes CSS Grid an incredibly powerful tool for modern web design. I am excited to apply these concepts in my future projects to build sophisticated and well-organized web pages.

Daily Diary - Web Development Training

Day 14

Applying External CSS

Summary of the Day: On the fourteenth day of our web development training, we were tasked with enhancing the visual appeal of our HTML webpages by applying external CSS. This task was designed to help us practice and reinforce our understanding of CSS, including selectors, properties, and different layout techniques we have learned so far.

Detailed Notes:

1. Setting Up External CSS:

Create a separate CSS file (e.g., `styles.css`).

Link the external CSS file to the HTML document using the `<link>` tag within the `<head>` section.

html

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

2. Structuring the HTML:

Ensure the HTML document is well-structured with semantic tags.

Example of a basic HTML structure:

html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>My Webpage</title>
<link rel="stylesheet" href="styles.css"> </head>
<body>
<header class="header">Header Content</header>
<nav class="navbar">Navigation Bar</nav> <main class="main-content"> <section class="content">Main Content</section>
<aside class="sidebar">Sidebar Content</aside> </main>
```

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

</body>

</html>
```

3. Writing External CSS:

Basic Styling:

css

```
/* styles.css */ body { font-family: Arial, sans-serif; margin: 0; padding: 0; background-color: #f0f0f0; } .header, .navbar, .footer { background-color: #333; color: white; text-align: center; padding: 10px 0; } .main-content { display: flex; justify-content: space-between; margin: 20px; } .content, .sidebar { background-color: #fff; padding: 20px; margin: 10px; box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); flex: 1; }
```

Advanced Styling with Flexbox and Grid:

css

```
/* Flexbox for Navigation Bar */ .navbar { display: flex; justify-content: space-around; align-items: center; } /* Grid for Main Content Layout */ .main-content { display: grid; grid-template-columns: 2fr 1fr; gap: 20px; } .content { grid-column: 1 / 2; } .sidebar { grid-column: 2 / 3; }
```

Adding Visual Enhancements:

css

```
/* Adding Hover Effects */ .header, .navbar, .footer { transition: background-color 0.3s; } .header:hover, .navbar:hover, .footer:hover { background-color: #555; } /* Styling Links */ a { color: #333; text-decoration: none; transition: color 0.3s; } a:hover { color: #007BFF; } /* Styling Buttons */ button { background-color: #007BFF; color: white; border: none; padding: 10px 20px; cursor: pointer; transition: background-color 0.3s; } button:hover { background-color: #0056b3; }
```

4. Reflection:

Enhancing our HTML pages with external CSS allowed me to see the immediate impact of styling on web design. It reinforced the importance of separating content (HTML) from presentation (CSS) for better maintainability and scalability.

Practicing with various CSS properties and layout techniques, such as Flexbox and Grid, provided a deeper understanding of how to create visually appealing and responsive designs.

The task also highlighted the importance of consistency in styling and the use of best practices in CSS.