**CSS 3**

**Cascading Style Sheets**. It is used to control the style and layout of web pages. While HTML provides the structure (like headings, paragraphs, buttons), CSS determines **how these elements look**—their colors, size, spacing, and position.

A styling language introduced on **17th Dec 1996** by **W3C (www consorhum)** and spearheaded by **Tim Berners-Lee**.

**Frameworks: SASS, SCSS, Tailwind**

SYNTAX:

Selector {

property: value;

property: value;

}

EXAMPLE:

h1 {

color: red;

font-size: 24px;

}

Shorthand: font: italic bold 20px/1.5 Arial, sans-serif;

**Types of CSS:**

1. Inline CSS

* Written inside an HTML tag using the style attribute.
* **Priority**: Inline CSS has the **highest precedence**.

<h1 style="color: blue; font-size: 24px;">Hello, World!</h1>

1. Internal CSS

* Written within the <style> tag inside the <head> section of an HTML document.

<style>

p {

color: green;

font-size: 18px;

}

</style>

1. External CSS(prefereed most)

* Written in a separate .css file and linked using <link> in the <head> section.

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

**Types of selectors:**

Method to select an html element to apply css

1. **Element selector/Name/Tag**: p { color: green; } 🡪 Targets HTML tags directly.
2. **Class selector**: .highlight { background-color: yellow; } 🡪 Targets elements with a specific class (. for class). 🡪 duplicate is allowed here
3. **Id selector**: #header { text-align: center; } 🡪 Targets elements with a specific ID (# for ID).--> it should be unique in whole project and id is used by js
4. **Grouping Selectors**: h1, h2, p { font-family: Arial; }🡪Apply the same style to multiple selectors.
5. **Universal Selector**: \* { margin: 0; padding: 0; } 🡪 Targets all elements.
6. **Attribute Selector/type**: input[type="text"] {border: 1px solid blue;} 🡪Targets specific attribute

**CSS BOX MODEL**

(margin and border difference)

The **box model** defines how elements are structured on a webpage. Each element consists of:

1. **Content**: The actual text or image inside the box.
2. **Padding**: Space between the content and the border.
3. **Border**: The outline around the padding.
4. **Margin**: Space outside the border.

Top

Right

left

Content

bottom

Padding

Content

Border

Margin

div {

width: 300px;

height: 200px;

padding: 20px;

margin: 10px;

border: 2px solid black;

}

/\*shorthend padding:100px(top) 200px(right)  100px(bottom) 200px(left) ; \*/

            /\* 2values: 1st value is top and bottom and 2nd value is left and right\*/

            /\* 1 value: all side  \*/

**BACKGROUND**

(color pallets)

The background property in CSS allows you to style the background of elements like <div>, <body>, or any other HTML element. It can include colors, images, gradients, and other visual effects.

1. **Basic background color**

Use: names, Hexadecimal values, RGB or HSL

Ex: body

{

background-color: lightblue; /\* Using a color name \*/

}

* Colour name directly or go to color pallets(copy and paste)
* **Hexadecimal**: #RRGGBB (e.g., #ff5733)
* **RGB**: rgb(red, green, blue) (e.g., rgb(255, 87, 51))
* **HSL**: hsl(hue, saturation, lightness) (e.g., hsl(9, 100%, 60%))

1. **Background Image**

set an image as the background

Ex: body

{

background-image: url('background.jpg'); (local / web)

}

* **Background image properties**

1. **background-repeat: no-repeat;**

Possible values**: repeat, no-repeat, repeat-x, repeat-y**

1. **background-position: center center;**

You can use values like **top, bottom, left, right, or center**

1. **background-size:cover;**

elements: **cover, contain**

1. **background-attachment**: Controls whether the background image scrolls with the page.

**Values**:**scroll,fixed,local**

1. **Background Gradients**: allow to create a smooth transition between two or more colors.

**background: linear-gradient (to right, red, yellow);**

**background: radial-gradient(circle, red, yellow); /\*** Circular gradient \*/

Shorthand: Background

body {

background: #ff5733 url('background.jpg') no-repeat center center fixed;

}

**Flexbox layout**

It provides an efficient way to arrange items inside a container, even when their sizes are dynamic or unknown.

Parent class/ element : display:flex 🡪the parent element that holds all the flex items

Child elements: Flex items 🡪 The child elements inside the flex container

**MAIN PROPERTY OF FLEX CONTAINER**

Key Terminology:

1. Flex container: The parent element where Flexbox is applied.
2. Flex items: The child elements inside the flex container.

1. display: flex; 🡪This property enables Flexbox on the container.

Example:

<style>

.container {

display: flex;

border: 2px solid black;

}

.item {

background-color: lightblue;

padding: 20px;

margin: 10px;

}

<div class="container">

<div class="item">Item 1</div>

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

<div class="item">Item 3</div>

</div>

Output:  
Items are aligned in a row by default.

2. flex-direction 🡪Defines the direction of the flex items.

| Value: | Description |
| --- | --- |
| Row: | Default; items in a row (left to right). |
| row-reverse: | Items in a row (right to left). |
| Column: | Items in a column (top to bottom). |
| column-reverse: | Items in a column (bottom to top). |

<style>

.container {

display: flex;

flex-direction: column;

border: 2px solid black;

}

</style>

3. justify-content 🡪 Aligns items along the main axis (horizontal by default).

|  |  |
| --- | --- |
| flex-start: | Default; items at the start. |
| flex-end: | Items at the end. |
| Center: | Items in the center. |
| space-between: | Equal space between items. |
| space-around: | Equal space around items. |
| space-evenly: | Equal space between and around items. |

<style>

.container {

display: flex;

justify-content: space-around;

}

</style>

4. align-items 🡪Aligns items along the cross axis (vertical by default).

|  |  |
| --- | --- |
| stretch | Default; items stretch to fill. |
| flex-start | Items align at the top. |
| flex-end | Items align at the bottom. |
| center | Items align in the center. |
| baseline | Items align by their text baseline. |

<style>

.container {

display: flex;

align-items: center;

height: 200px; /\* To see vertical alignment \*/

}

</style>

5. align-content 🡪Aligns multiple lines of items when the flex container has extra space.

|  |  |
| --- | --- |
| stretch | Default; lines stretch to fill space. |
| flex-start | Lines align at the start. |
| flex-end | Lines align at the end. |
| center | Lines align in the center. |
| space-between | Equal space between lines. |
| space-around | Equal space around lines. |

<style>

.container {

display: flex;

flex-wrap: wrap;

align-content: space-around;

height: 400px;

}

</style>

6. flex-wrap 🡪Specifies whether items should wrap when they overflow the container.

|  |  |
| --- | --- |
| nowrap | Default; no wrapping. |
| wrap | Items wrap onto new lines. |
| wrap-reverse | Items wrap in reverse order. |

<style>

.container {

display: flex;

flex-wrap: wrap;

}

</style>

7. flex (on items) 🡪Defines how much space an item takes relative to others.

|  |  |
| --- | --- |
| flex-grow | Proportion of extra space the item gets. |
| flex-shrink | Proportion of shrinking when space is tight. |
| flex-basis | Initial size before remaining space is distributed. |

<style>

.item1 {

flex: 2; /\* Takes double space \*/

}

.item2 {

flex: 1; /\* Takes normal space \*/

}

</style>

8. order (on items) 🡪 Controls the order of items in the container (default is 0).

<style>

.item1 {

order: 2; /\* Appears last \*/

}

.item2 {

order: 1; /\* Appears first \*/

}

</style>

**Pseudo-classes**

A pseudo-class defines the special state of an element. It starts with a colon (:)

| **Pseudo-class** | **Description** |
| --- | --- |

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | :hover | : Applies styles when the mouse hovers over an element. | |
| |  |  | | --- | --- | | :focus | : Applies styles when an element gains focus (like an input field). | |
| |  |  | | --- | --- | | :active | : Applies styles when an element is being clicked. | |
| |  |  | | --- | --- | | :first-child | : Selects the first child of a parent. | |
| |  |  | | --- | --- | | :last-child | : Selects the last child of a parent. | |
| |  |  | | --- | --- | | :nth-child(n) | : Selects the nth child (e.g., nth-child(2) selects the second child). | |
| |  |  | | --- | --- | | :nth-child(odd) | : Selects odd children. | |
| |  |  | | --- | --- | | :nth-child(even) | : Selects even children. | |
| |  |  | | --- | --- | | :not(selector) | : Selects all elements that do not match the given selector. | |
| |  |  | | --- | --- | | :checked | : Targets checked <input> elements like checkboxes and radio buttons. | |
| |  |  | | --- | --- | | :disabled | : Targets disabled form elements. | |
| |  |  | | --- | --- | | :enabled | : Targets enabled form elements. | |

**2. Pseudo-elements**

A pseudo-element styles specific parts of an element. It starts with a double colon (::).

**Commonly Used Pseudo-elements:**

| **Pseudo-element** | **Description** |
| --- | --- |
| ::before | Inserts content before the element. |
| ::after | Inserts content after the element. |
| ::first-letter | Styles the first letter of an element. |
| ::first-line | Styles the first line of an element. |
| ::placeholder | Styles the placeholder text in an input field. |
| ::selection | Styles the portion of text selected by the user. |

**DISPLAY**

The display property defines how an HTML element is displayed on a webpage. It is one of the most important CSS properties as it determines the layout and structure of elements.

**1. Block**

* A block-level element takes up the **entire width** of its parent container, starting on a new line.­
* Examples of block elements: <div>, <p>, <h1>, <section>.

**Properties:**

* Height and width can be applied.
* Always starts on a new line.

**2. None­**

* The display: none; property **hides** the element completely from the page.
* The element won't take up any space on the page.

**3. Inline**

* Inline elements **do not start on a new line**.
* They only take up as much width as necessary (content width).
* Examples: <span>, <a>, <b>, <strong>.

**Properties:**

* You **cannot** apply height and width directly to inline elements.

**4. Inline-Block**

* Behaves like an inline element but allows you to set **height** and **width** like a block element.

**5. Flex**

* The display: flex; property enables **Flexbox**, a powerful layout tool for aligning and distributing items in a container.

**6. Grid**

* The display: grid; property enables the Grid layout system for arranging items in rows and columns.

**POSITION**

The position property determines how an element is **placed** in the document. It works with properties like top, right, bottom, and left.

**1. Static (Default)**

* Elements are positioned according to the normal document flow.
* **Does not respond** to top, left, right, or bottom.

**2. Relative**

* The element is positioned relative to its **normal position**.
* You can use top, left, right, and bottom to **offset** the element from its original position.

**3. Absolute**

* The element is positioned **relative to its nearest positioned ancestor** (if none, it is positioned relative to the viewport).
* **Removes** the element from the normal document flow.

**4. Fixed**

* The element is **fixed to the viewport** and does not move when the page is scrolled.
* Commonly used for sticky headers or floating buttons.

**5. Sticky**

* The element **sticks to its parent container** as you scroll down the page.
* Combines relative and fixed behaviors.

**Comparison of Position Types**

| **Position** | **Description** |
| --- | --- |
| **Static** | Default positioning, follows normal flow. |
| **Relative** | Offset relative to its normal position. |
| **Absolute** | Positioned relative to the nearest ancestor or viewport. |
| **Fixed** | Fixed relative to the viewport and does not move with scrolling. |
| **Sticky** | Sticks to its parent and toggles between relative and fixed during scrolling. |
|  |  |

**ANIMATION**

Animation in CSS lets you **animate the styles** of an element (e.g., color, position, size, etc.) without using JavaScript.

**. Key Properties for Animation**

To use animations, you primarily need the following CSS properties:

1. **animation-name**: Defines the name of the @keyframes animation.
2. **animation-duration**: Specifies how long the animation will run (e.g., 2s, 5s).
3. **animation-timing-function**: Defines the speed curve of the animation (e.g., ease, linear, ease-in, ease-out).
4. **animation-delay**: Delays the start of the animation (e.g., 2s).
5. **animation-iteration-count**: Specifies how many times the animation will run (e.g., 1, infinite).
6. **animation-direction**: Defines the direction of the animation (e.g., normal, reverse, alternate).
7. **animation-fill-mode**: Determines the state of the element before and after the animation (e.g., forwards, backwards, both).
8. **animation-play-state**: Pauses or resumes the animation (running, paused).

**2. Keyframes**

@keyframes defines the stages of the animation. There are two types of keyframes:

1. **from and to**

**Type 1: Using from and to**

**from**: Represents the initial state of the animation.

**to**: Represents the final state of the animation.

1. **Percentage-based keyframes**

Specify multiple stages of the animation using percentages (0% to 100%).

**FULL ANIMATION SYNTAX:**

selector {

animation-name: keyframes\_name;

animation-duration: duration;

animation-timing-function: timing\_function;

animation-delay: delay;

animation-iteration-count: iteration\_count;

animation-direction: direction;

animation-fill-mode: fill\_mode;

animation-play-state: play\_state;

}

**Common Animation**: animation: colorChange 5s ease-in-out 2s infinite alternate forwards;

**ADDING MULTIPLE ANIMATION**: animation: slide 3s ease-in-out infinite, rotate 5s linear infinite;