**CASCADING STYLE SHEETS**

CSS, or Cascading Style Sheets, is the language used to style and enhance HTML documents. It defines the presentation of HTML elements on a web page, enabling changes to fonts, colors, sizes, spacing, column layouts, and animations.

**Table of Content**

1. [Basic CSS Example](https://www.geeksforgeeks.org/css-tutorial/?ref=dhm#basic-css-example)
2. [CSS Tutorial](https://www.geeksforgeeks.org/css-tutorial/?ref=dhm#css-tutorial)
3. [CSS Fundamentals](https://www.geeksforgeeks.org/css-tutorial/?ref=dhm#css-fundamentals)
4. [CSS Styling Techniques](https://www.geeksforgeeks.org/css-tutorial/?ref=dhm#css-styling-techniques)
5. [CSS Responsive Design and Media Queries](https://www.geeksforgeeks.org/css-tutorial/?ref=dhm#css-responsive-design-and-media-queries)
6. [Advanced CSS Topics](https://www.geeksforgeeks.org/css-tutorial/?ref=dhm#advanced-css-topics)
7. [CSS Online Quizzes](https://www.geeksforgeeks.org/css-tutorial/?ref=dhm#css-online-quizzes)
8. [CSS Practical Projects](https://www.geeksforgeeks.org/css-tutorial/?ref=dhm#css-practical-projects)
9. [CSS Interview Preparation](https://www.geeksforgeeks.org/css-tutorial/?ref=dhm#css-interview-preparation)
10. [CSS Preprocessors](https://www.geeksforgeeks.org/css-tutorial/?ref=dhm#css-preprocessors)
11. [CSS Frameworks](https://www.geeksforgeeks.org/css-tutorial/?ref=dhm#css-frameworks)

**What is CSS?**

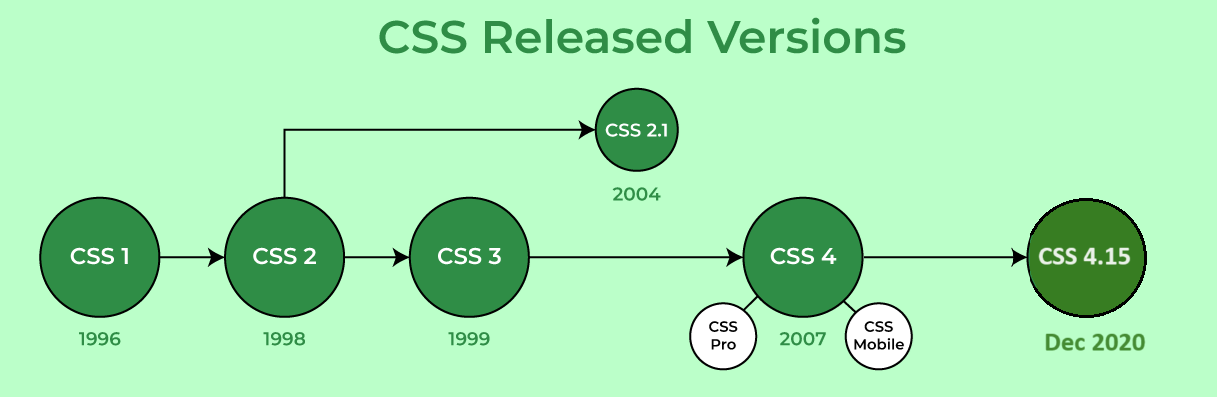
CSS, or **Cascading Style Sheets**, is a language used to style and enhance websites. It controls how **HTML elements**—such as *text*, *images*, and *buttons*—are displayed on a webpage. **With CSS**, you can adjust font sizes and colors, add backgrounds, and manage the layout, transforming a basic webpage into a visually appealing and user-friendly experience. CSS also simplifies **layout management** across multiple web pages by **using external stylesheets** stored in CSS files.

CSS (Cascading Style Sheets) is a language designed to simplify the process of making web pages presentable. It allows you to apply styles to HTML documents, describing how a webpage should look by prescribing colors, fonts, spacing, and positioning. CSS provides developers and designers with powerful control over the presentation of HTML elements.

HTML uses tags and CSS uses rulesets. CSS styles are applied to the HTML element using selectors. CSS is easy to learn and understand, but it provides powerful control over the presentation of an HTML document.

**Why CSS?**

* **Saves Time:**Write CSS once and reuse it across multiple HTML pages.
* **Easy Maintenance:** Change the style globally with a single modification.
* **Search Engine Friendly:** Clean coding technique that improves readability for search engines.
* **Superior Styles:** Offers a wider array of attributes compared to HTML.
* **Offline Browsing:**CSS can store web applications locally using offline cache, allowing offline viewing.



**CSS Syntax**

CSS consists of style rules that are interpreted by the browser and applied to the corresponding elements. A style rule set includes a selector and a declaration block.

* **Selector:** Targets specific HTML elements to apply styles.
* **Declaration:** Combination of a property and its corresponding value.

// HTML Element

<h1>GeeksforGeeks</h2>

// CSS Style

h1 { color: blue; font-size: 12px; }

**Where -**

Selector - h1

Declaration - { color: blue; font-size: 12px; }

* The **selector points** to the HTML element that you want to style.
* The **declaration block** contains one or more declarations separated by semicolons.
* Each **declaration** includes a CSS property name and a value, separated by a colon.

p {

color: blue;

text-align: center;

}

CSS **declaration always ends with a semicolon**, and declaration blocks are surrounded by **curly braces**. In above example, all paragraph element (<p> tag) will be centre-aligned, with a blue text color.

**Web Page with & without CSS**

**Without CSS:** In this example, we have not added any CSS style.

<!DOCTYPE html>

<html>

<head>

<title>Simple Web Page</title>

</head>

<body>

<main>

<h1>HTML Page</h1>

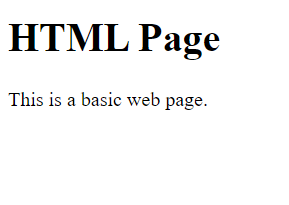
<p>This is a basic web page.</p>

</main>

</body>

</html>

**OUTPUT:**



**Using CSS:** In this example, we will add some CSS styles inside the HTML document to show how CSS makes a HTML page attractive and user-friendly.

<!DOCTYPE html>

<html>

<head>

<title>Simple web page</title>

<style>

main {

width: 600px;

height: 200px;

padding: 10px;

background: beige;

}

h1 {

color: olivedrab;

border-bottom: 1px dotted darkgreen;

}

p {

font-family: sans-serif;

color: orange;

}

</style>

</head>

<body>

<main>

<h1>My first Page</h1>

<p>This is a basic web page.</p>

</main>

</body>

</html>

**OUTPUT:**



CSS is essential for creating visually appealing and maintainable web pages. It enhances the website look and feel and user experience by allowing precise control over the presentation of HTML elements.

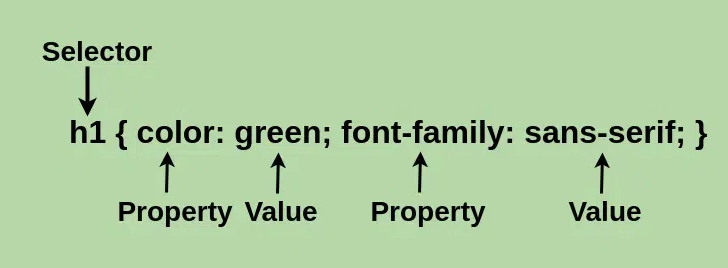
**CSS Syntax**

A **CSS Syntax** rule consists of a selector, property, and its value. The selector points to the HTML element where the CSS style is to be applied. The CSS property is separated by semicolons. It is a combination of the selector name followed by the **property: value** pair that is defined for the specific selector. let us see the syntax and how we can use the CSS to modernize the website.

**Syntax:**

selector { Property: value; }

For instance, we have declared a heading tag (h1) along with having assigned some ***property: value*** pair that is used to style the heading tag. Here, ***h1***is the selector,***{ color: green; font-family: sans-serif; }***is a declaration block and it can contain one or more declarations separated by semicolons,***color: green;***is a *property: value* pair that is applied to the HTML element to style them.



Every declaration has a CSS property name and a value, separated by a **colon(:)** and is surrounded by **curly braces({ })**. For declaring the multiple CSS properties, it can be separated by the **semicolon(;)**.

Let’s define each of these:

* **Declaration:**A combination of a property and its corresponding value.
* **Selector:**Used to target and select specific HTML elements to apply styles to.
* **Property:** Defines the specific aspect or characteristic of an element that you want to modify.
* **Value:** Assigned setting or parameter for a given property, determining how the selected element should appear or behave.

**Different Ways to Use CSS**

CSS has three ways to style the HTML:

* **Inline**: Add styles directly to HTML elements using the style attribute (limited use).
* **Internal**: Place styles within a *<style>* tag inside the HTML file, usually within the *<head>* section
* **External**: Create a separate CSS file with a *.css* extension and link it to your HTML file using the*<link>* tag.

**Advantages of CSS:**

* CSS you simply got to specify a repeated style for element once & use it multiple times as because CSS will automatically apply the required styles.
* CSS style is applied consistently across variety of sites. One instruction can control several areas which is advantageous.
* Web designers need to use few lines of programming for every page improving site speed.
* CSS simplifies both website development and maintenance as a change of one line of code affects the whole web site and maintenance time.
* It is less complex therefore the effort is significantly reduced.
* It helps to form spontaneous and consistent changes.
* CSS changes are device friendly. With people employing a batch of various range of smart devices to access websites over the web, there’s a requirement for responsive web design.
* It has the power for re-positioning. It helps us to determine the changes within the position of web elements who are there on the page.
* These bandwidth savings are substantial figures of insignificant tags that are indistinct from a mess of pages.
* Easy for the user to customize the online page
* It reduces the file transfer size.

**Disadvantages of CSS:**

* CSS, CSS 1 up to CSS3, result in creating of confusion among web browsers.
* With CSS, what works with one browser might not always work with another. The web developers need to test for compatibility, running the program across multiple browsers.
* There exists a scarcity of security.
* After making the changes we need to confirm the compatibility if they appear. The similar change effects on all the browsers.
* Browser compatibility (some styles sheet are supported and some are not).
* CSS works differently on different browsers. IE and Opera supports CSS as different logic.
* There might be cross-browser issues while using CSS.
* There are multiple levels which creates confusion for non-developers and beginners.

**Basic CSS Example**

Below example shows inline, internal, and external style sheet with different properties.

<!-- File name: **index.html** -->

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

**<!-- Importing External CSS -->**

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

**<!-- Using Internal CSS -->**

**<style>**

**h2 {**

**color: green;**

**background-color: orange;**

**}**

**</style>**

    <title>Inline Internal and External CSS Demo</title>

</head>

<body>

        <!-- Using Inline CSS -->

        <h2 style="text-align: center;">Welcome To CSS</h2>

        <p>CSS Tutorial</p>

</body>

</html>

/\* External CSS \*/

/\* File name: **style.css** \*/

p {

    text-align: center;

    color: chartreuse;

    font-size: 33px;

  }

**OUTPUT:**



* **FAQs:**

1. **What is CSS?**

*CSS (Cascading Style Sheets) is a language for styling HTML or XML documents, controlling layout, colors, fonts, and overall appearance to enhance user experience.*

1. **Why is CSS important?**

*CSS separates content from design, improves accessibility, enhances user experience, and provides responsive designs for different devices and screen sizes.*

1. **How do you add CSS to a web page?**

* *Inline CSS: Using style attribute within HTML.*
* *Internal CSS: Using <style> block in <head>.*
* *External CSS: Linking a CSS file with <link> tag.*

1. **What is the syntax of CSS?**

*CSS syntax: selector { property: value; }. Selectors target elements; properties define style attributes; values specify styling details.*

1. **What are CSS selectors?**

*CSS selectors target elements for styling. Common types: element, class, ID, and attribute selectors.*

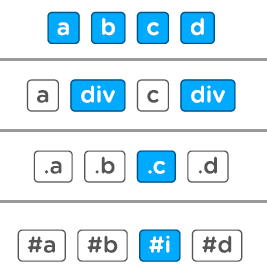
1. **What are CSS properties?**

*CSS properties define styles like color, font-size, margin, and padding. Example: p { color: blue; }.*

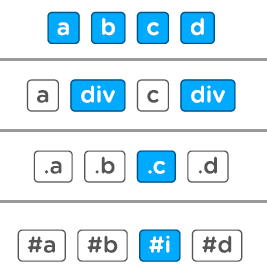
**SELECTORS IN CSS:**

**CSS selectors** are used *to select the content you want to style*. Selectors are the part of CSS rule set. CSS selectors select HTML elements are categorised in to following

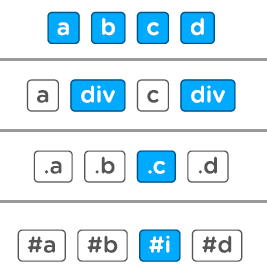
1. Basic/Simple Selectors
2. Combination
3. Attribute
4. Pseudo Element
5. Pseudo Class State
6. Pseudo Class position / other
7. Basic/Simple Selectors:
   1. **Universal Selector (\*)** : Selects all elements present on HTML page and applies the style to all elements.



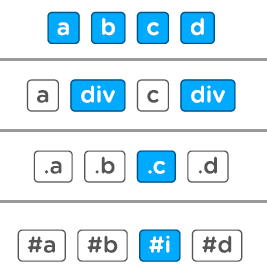
* 1. **Type Selector (Html\_Tag)** : Selects specific html tag and applies the style to only that tag i.e. whenever this tag appears the style gets applied to it.



* 1. **Class Selector(.Class\_Name)** : Creates a group of style properties and we use that class name as style to one or more elements in HTML.



* 1. **ID Selector (#ID)** : Create a group of style properties and assign to only one element in a HTML. We cannot use ID selector more than one.



**Example:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Document</title>

    <style>

        /\*Universal Selector\*/

        \* {

            background-color: aquamarine;

        }

        /\*Tyle Selector or Element Selector\*/

        span {

            font-style: italic;

            color: blue;

        }

        /\*Class Selector\*/

        .MyClassStyle {

            padding-left: 70px;

            text-decoration: underline;

            background: rgba(133,133,350,0.5);

        }

        /\*ID Selector\*/

        #MyID {

            background: greenyellow;

            text-decoration: dotted;

            font-family: 'Courier New', Courier, monospace;

            font-weight: bolder;

            font-size: 21px;

        }

    </style>

</head>

<body>

    <div class="MyClassStyle">This is a div</div>

    <span>This is a span 1</span>

    <ul>

        <li>Item 1</li>

        <li>Item 2</li>

        <li class="MyClassStyle">Item 3</li>

        <li>Item 4</li>

        <li>Item 5</li>

        <li>Item 6</li>

        <li>Item 7</li>

        <span id="MyID">This is a span 2</span></span>

    </ul>

    <div>

        <label for="username">User Name :</label>

        <input class="MyClassStyle" type="text" name="username" id="username" value="Text box is padded to Left....">

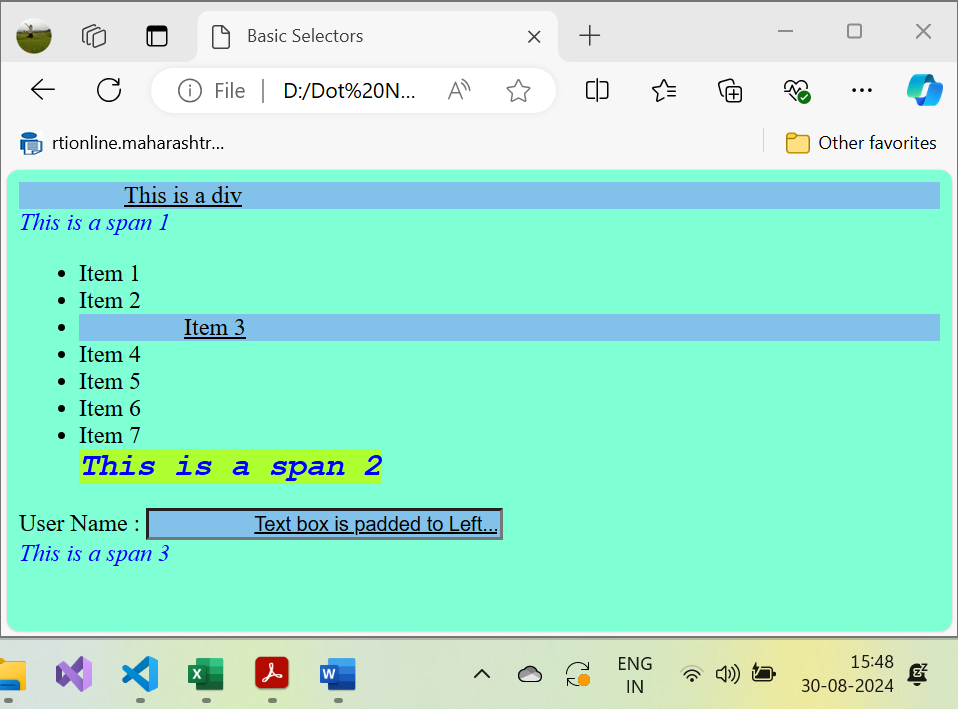
    </div>

    <span>This is a span 3</span></span>

</body>

</html>

Output:



1. **Combination:** A CSS selector can contain more than one Basic/Simple selector. Between the Basic/Simple selectors, we can include a combinator.
   1. **Descendant Selector (div a):** The descendant selector matches all elements that are descendants of a specified first element. Space between two elements indicates anything comes under first element with direct child or child of child of ….child second element, style will get applied to it.



Example:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Combination Adjacent Sibling Selectors</title>

    <style>

    /\*Adjacent Sibling Selector \*/

    ul li{

        background: lightblue;

    }

    div b{

        background-color: aqua;

        font-size: 33px;

    }

    div .green{

        color: green;

    }

    </style>

</head>

<body>

    <div>This is a div1</div>

    <span>This is a span 1</span>

    <div>

        <span>

            <b>This is nested elements</b>

        </span>

    </div>

    <div>

        <span>

            <b><i class="green">This is nested elements</i></b>

        </span>

    </div>

    <ul>

        <li>Item 1</li>

        <li>Item 2</li>

        <li>Item 3</li>

        <li>Item 4</li>

        <li>Item 5</li>

        <li>Item 6</li>

        <li>Item 7</li>

        <span>This is a span 2</span></span>

        <div>This is a div3</div>

    </ul>

    <div>

        <label class="green" for="username">User Name :</label>

        <input style="color:pink;" type="text" name="username" id="username" value="Text box is padded to Left....">

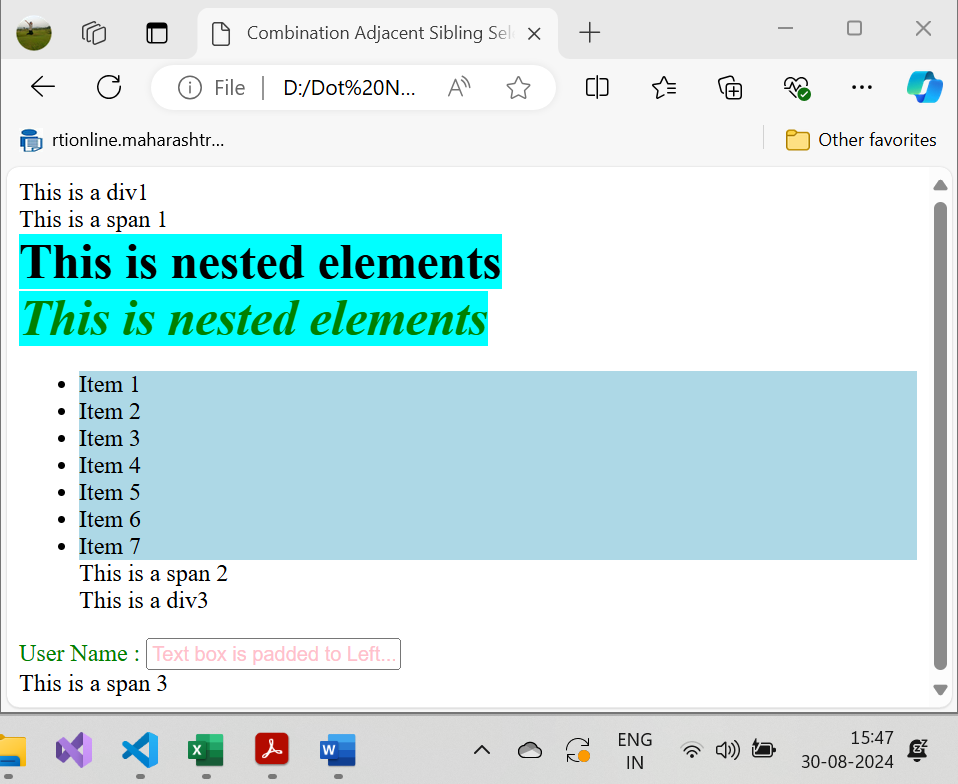
    </div>

    <span>This is a span 3</span></span>

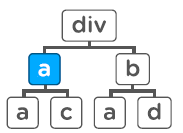
</body>

</html>

Output:



* 1. **Direct Child Selector (div > a):** The child selector selects all elements that are the **Direct** child of a specified element.



**Example:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Combination Direct Child Selectors</title>

    <style>

    /\*Direct Child Selector \*/

    ul > li{

        background: purple;

    }

    span > b{

        background-color: blueviolet;

        font-size: 33px;

    }

    div > .green{

        color: green;

    }

    </style>

</head>

<body>

    <div>This is a div1</div>

    <span>

        <p>

            <b>This is a span 1 Not direct Child</b>

        </p>

    </span>

    <div>

        <span>

            <b>This is nested elements 1</b>

        </span>

    </div>

    <div>

        <span>

            <b><i class="green">This is nested elements 2</i></b>

        </span>

    </div>

    <ul>

        <li>Item 1</li>

        <li>Item 2</li>

        <li>Item 3</li>

        <li>Item 4</li>

        <li>Item 5</li>

        <li>Item 6</li>

        <li>Item 7</li>

        <span>This is a span 2</span></span>

        <div>This is a div3</div>

    </ul>

    <div>

        <label class="green" for="username">User Name :</label>

        <input style="color:pink;" type="text" name="username" id="username" value="Text box is padded to Left....">

    </div>

    <div>

        <H1 class="green">This is H1</H1>

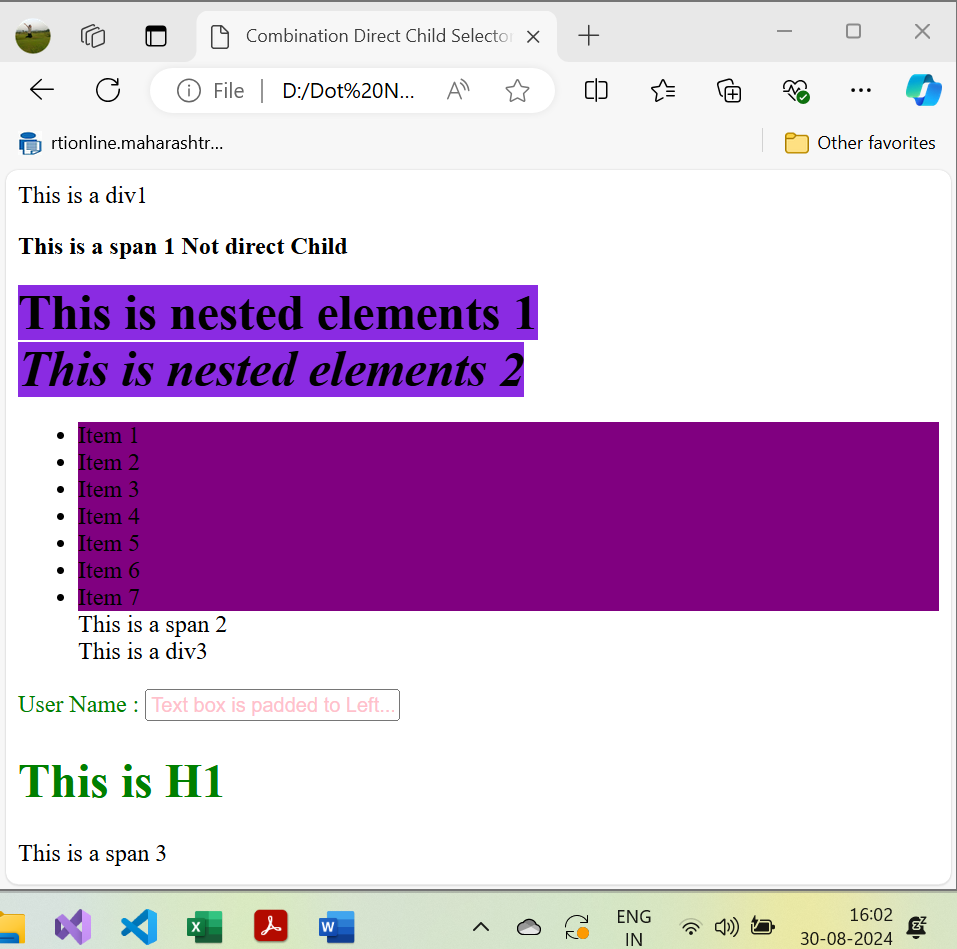
    </div>

    <span>This is a span 3</span></span>

</body>

</html>

**Output:**



* 1. **General Sibling Selector (div ~ a):** Style gets applied to only siblings mentioned after first element. It only selects elements after (i.e to siblings).



Example:

Example combined with Adjacent Siblings

* 1. **Adjacent Sibling Selector (div + a):** Style gets applied to very next sibling only



Example 1:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Combination General & Adjacent Sibling Selectors</title>

    <style>

    /\* General Sibling Selector \*/

    li.green ~li{

        background: green;

    }

    /\* Adjacent Sibling Selector \*/

    li.red + li{

        background: red;

    }

    </style>

</head>

<body>

    <div class="green">This is a div1</div>

    <div>This is a div2</div>

    <span>This is a span 1</span>

    <ul>

        <li>Item 1</li>

        <li class="red">Item 2</li>

        <li>Item 3</li>

        <li>Item 4</li>

        <li class="green">Item 5</li>

        <li>Item 6</li>

        <li>Item 7</li>

        <span>This is a span 2</span></span>

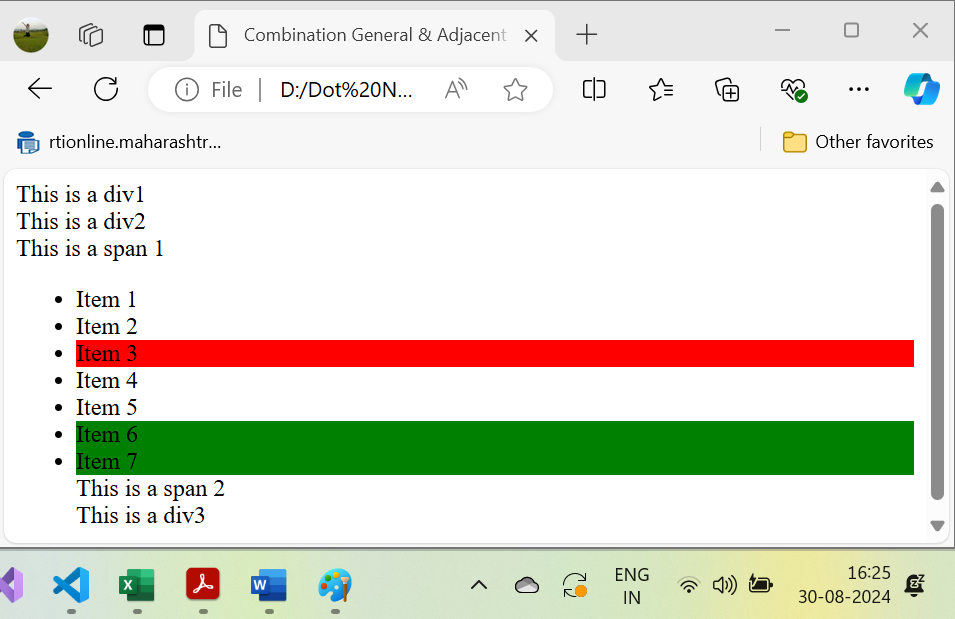
        <div class="green">This is a div3</div>

    </ul>

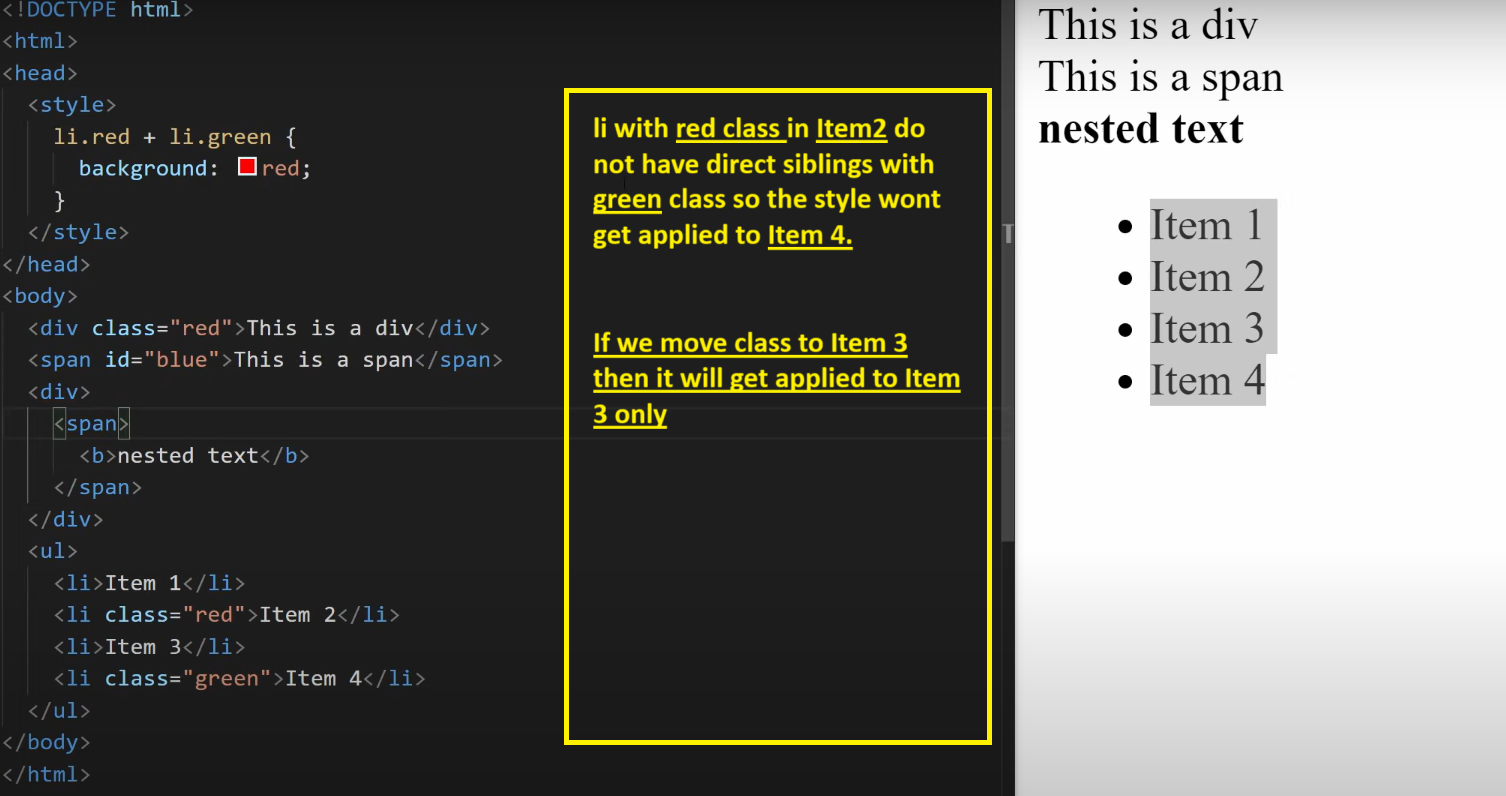
</body>

</html>

Output:



Example 2:



* 1. **Or Selector (div, a)**



Example:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Combination OR Selectors</title>

    <style>

/\* Or Selector \*/

div, .yellow{

    background: yellow;

}

span, li.red.green-text{

    background: red;

    font-size: 33px;

    color: green;

}

    </style>

</head>

<body>

    <div class="red">This is a div1</div>

    <div>This is a div2</div>

    <span>This is a span 1</span>

    <ul>

        <li>Item 1</li>

        <li class="yellow">Item 2</li>

        <li>Item 3</li>

        <li>Item 4</li>

        <li class="yellow green-text">Item 5</li>

        <li>Item 6</li>

        <li class="red green-text">Item 7</li>

        <span>This is a span 2</span></span>

        <div class="yellow">This is a div3</div>

    </ul>

    <div class="yellow">

        <label for="username">User Name :</label>

        <input type="text" name="username" id="username" value="Text box is padded to Left....">

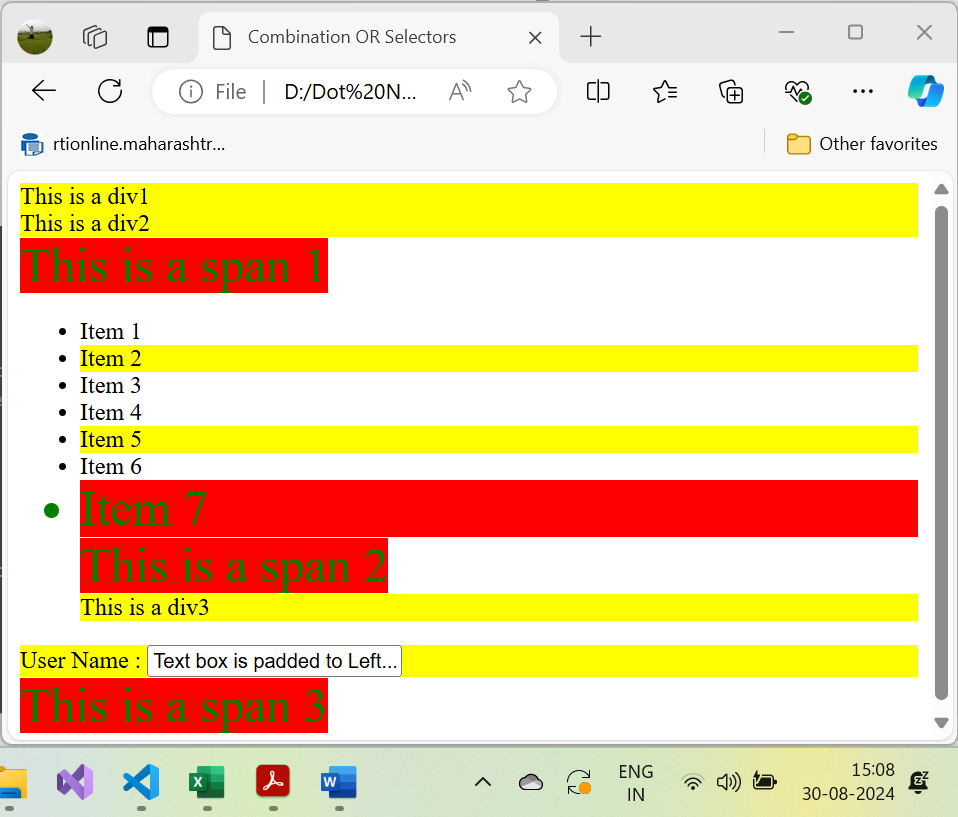
    </div>

    <span>This is a span 3</span></span>

</body>

</html>

Output:



* 1. **And Selector(div.c)**



Example:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Combination AND Selectors</title>

    <style>

/\* And Selector \*/

div.red{

    background: red;

}

div.red.green-text, span.red.green-text{

    background: green;

    font-size: 33px;

    color: white;

}

    </style>

</head>

<body>

    <div class="red">This is a div1</div>

    <div>This is a div2</div>

    <span>This is a span 1</span>

    <ul>

        <li>Item 1</li>

        <li>Item 2</li>

        <li>Item 3</li>

        <li>Item 4</li>

        <li>Item 5</li>

        <li>Item 6</li>

        <li>Item 7</li>

        <span>This is a span 2</span></span>

        <div class="red green-text">This is a div3</div>

    </ul>

    <div class="red green-text">

        <label for="username">User Name :</label>

        <input style="color:pink;" type="text" name="username" id="username" value="Text box is padded to Left....">

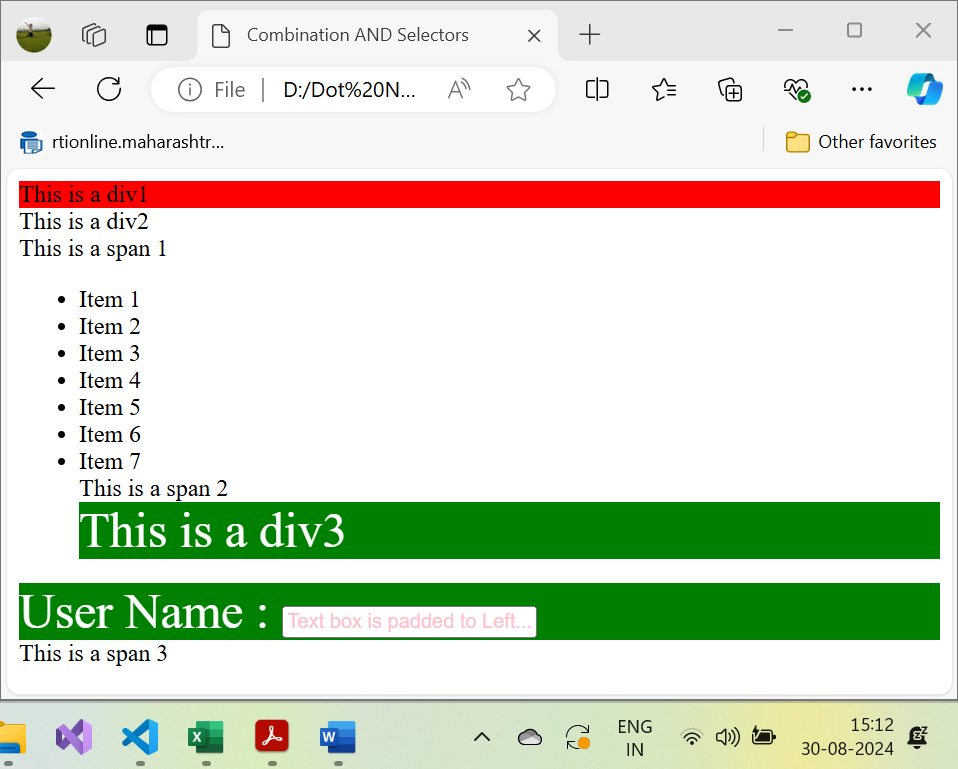
    </div>

    <span>This is a span 3</span></span>

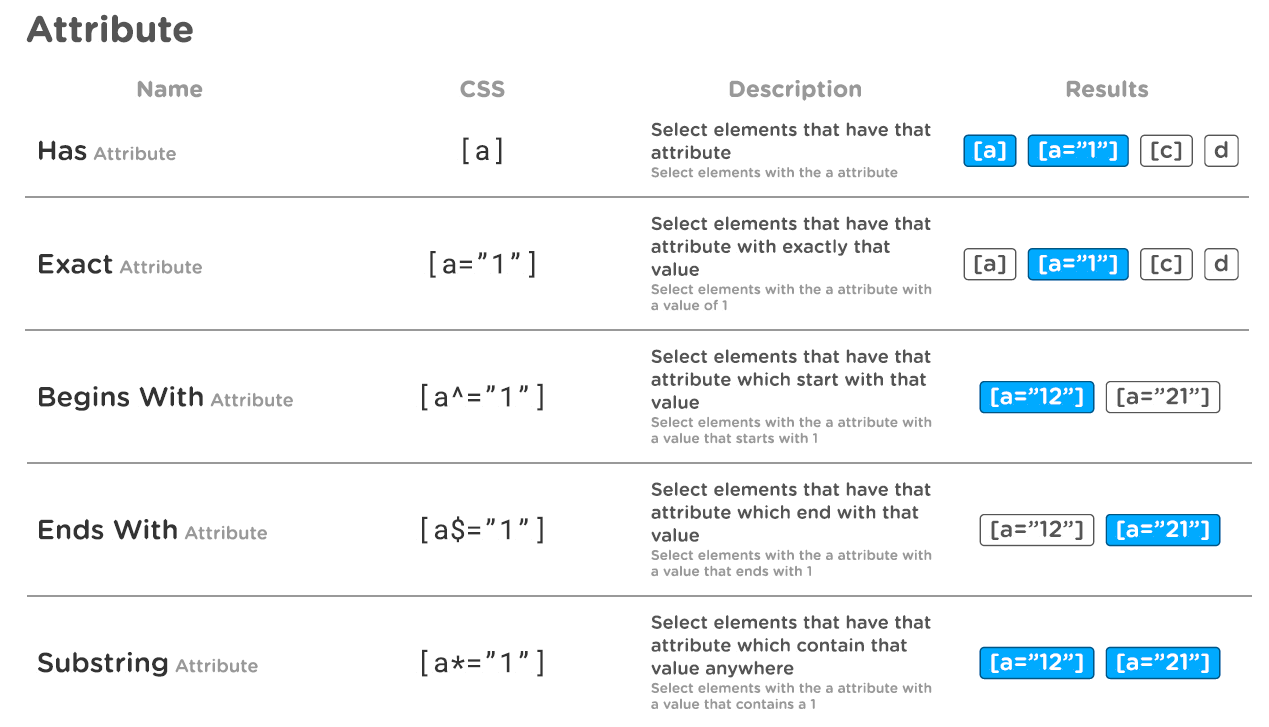
</body>

</html>

Output:



1. **Attribute:**

****

1. **Pseudo Element:** In CSS, pseudo-elements allow you to style specific parts of an element. They are denoted by double colons (::) followed by the pseudo-element name. Here are some common pseudo-elements along with examples:
   1. **::before** : This inserts content before an element.

**p::before {**

**content: "Note: ";**

font-weight: bold;

color: red;

}

<p>This is a paragraph.</p>

\*\*Result:\*\*

**Note:** This is a paragraph.

* 1. **::after :** This inserts content after an element.

**p::after {**

**content: " (Read more)";**

color: blue;

}

<p>This is a paragraph.</p>

\*\*Result:\*\*

This is a paragraph. (Read more)

* 1. **::first-letter** : This styles the first letter of an element.

p::first-letter {

font-size: 2em;

color: green;

}

<p>This is a paragraph.</p>

\*\*Result:\*\*

This is a paragraph.

* 1. **::first-line :** This styles the first line of an element.

p::first-line {

font-weight: bold;

color: purple;

}

<p>This is a long paragraph that might span multiple lines.</p>

\*\*Result:\*\*

\*\*This is a long paragraph\*\* that might span multiple lines.

* 1. **::placeholder :** This styles the placeholder text of an input element and as user start typing input value it gets disappear.

input::placeholder {

color: gray;

font-style: italic;

}

<input type="text" placeholder="Enter your name">

\*\*Result:\*\*

The placeholder text "Enter your name" will appear in gray and italic.

* 1. **::selection :** This styles the portion of text that is selected by the user.

**::selection {**

**background-color: yellow;**

**color: black;**

**}**

<p>Select this text to see the effect.</p>

\*\*Result:\*\*

When the text is selected, it will have a yellow background with black text.

**Usage Note:**

Pseudo-elements allow for more fine-grained control over specific parts of an element's content, making them powerful tools for web design.

**Example:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Pseudo Elements Selectors</title>

    <style>

        p::before{

            content: "Note: ";

            font-weight: bold;

            color: red;

        }

        div::after{

            content: "(Read more)";

            color: blue;

        }

        h1::first-letter {

            color: #ff0000;

            font-size: 57px;

        }

        h2::first-line {

            color: #0000ff;

            font-variant: small-caps;

        }

        input::placeholder {

            color: gray;

            font-style: italic;

        }

        ::selection {

            background-color: yellow;

            color: red;

        }

    </style>

</head>

<body>

    <p>This is a paragraph.</p>

    <div>This is a paragraph.</div>

    <h1>This is H1</h1>

    <h2> H2 You can combine the ::first-letter

        and ::first-line pseudo-elements to add a special

        effect to the first letter and the first line of

        a text!

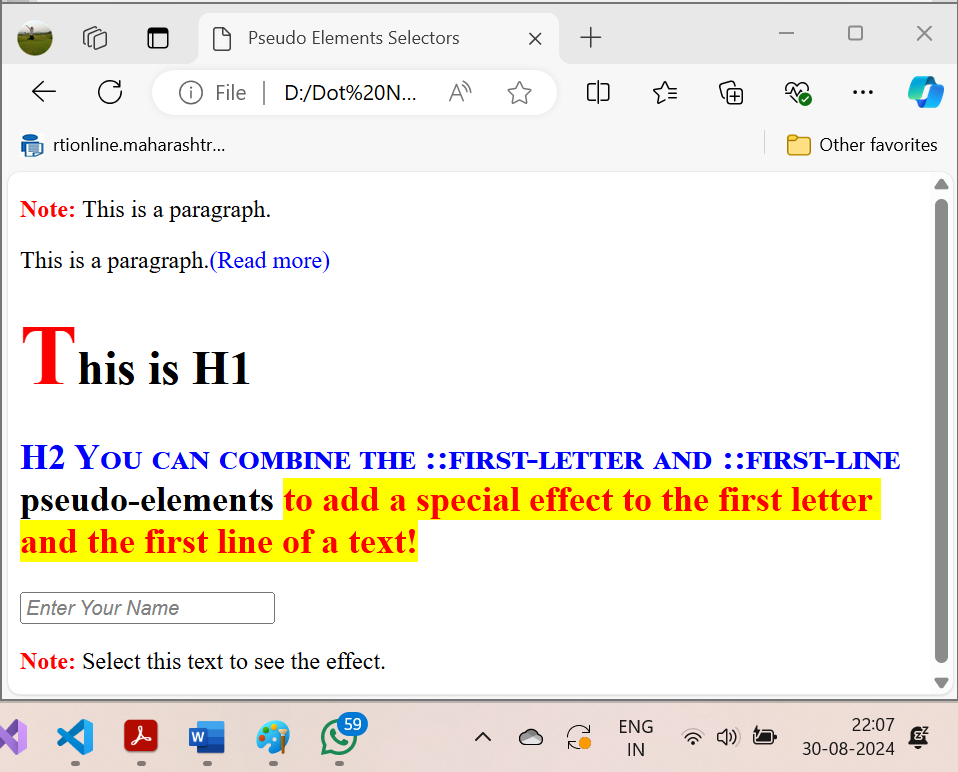
    </h2>

    <input type="text" placeholder="Enter Your Name">

    <p>Select this text to see the effect.</p>

</body>

</html>



1. Pseudo Class State

CSS Pseudo-classes are powerful and allow developers to style elements based on their specific states.

Pseudo-classes in CSS are used to define the special state of an element. They can be combined with a CSS selector to add an effect to existing elements based on their states. For instance, you can change the style of an element when the user hovers over it, or when a link is visited. All of these can be achieved using Pseudo Classes in CSS.

**Note**that pseudo-class names are not case-sensitive.

Most commonly used Pseudo classes states are as below:

* 1. **:hover** : Select elements that are hovered by the mouse.
  2. **:focus** : Select element that is focused. It is either by clicking or by tabbing to an element.
  3. **:required** : Select the element with required attribute.
  4. **:checked** : Select checkboxes / radio buttons that are with checked attribute.
  5. **:disabled** : Select the inputs that are with disabled attribute.
* [:active](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#active)
* [::after/:after](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#after-after)
* [::backdrop (experimental)](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#backdrop-experimental)
* [::before/:before](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#before-before)
* [:checked](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#checked)
* [:default](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#default)
* [:dir (experimental)](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#dir-experimental)
* [:disabled](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#disabled)
* [:empty](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#empty)
* [:enabled](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#enabled)
* [:first-child](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#first-child)
* [::first-letter/:first-letter](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#first-letter-first-letter)
* [::first-line/:first-line](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#first-line-first-line)
* [:first-of-type](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#first-of-type)
* [:focus](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#focus)
* [:fullscreen (experimental)](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#fullscreen-experimental)
* [:hover](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#hover)
* [:in-range](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#span-class-headeranchor-span-in-range)
* [:indeterminate](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#indeterminate)
* [:invalid](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#invalid)
* [:lang](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#lang)
* [:last-child](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#last-child)
* [:last-of-type](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#last-of-type)
* [:link](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#link)
* [:not](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#not)
* [:nth-child](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#nth-child)
* [:nth-last-child](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#nth-last-child)
* [:nth-last-of-type](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#nth-last-of-type)
* [:nth-of-type](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#nth-of-type)
* [:only-child](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#only-child)
* [:only-of-type](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#only-of-type)
* [:optional](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#optional)
* [:out-of-range](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#out-of-range)
* [::placeholder (experimental)](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#placeholder-experimental)
* [:read-only](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#read-only)
* [:read-write](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#read-write)
* [:required](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#required)
* [:root](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#root)
* [::selection](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#selection)
* [:scope (experimental)](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#scope-experimental)
* [:target](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#target)
* [:valid](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#valid)
* [:visited](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#visited)
* [Bonus content: A Sass mixin for links](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#bonus-content-a-sass-mixin-for-links)

Example:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Selectors Pseudo Class State</title>

    <style>

        /\*Selectors Pseudo Class State\*/

        li:hover{

            background: blueviolet;

        }

        a.highlight:hover {

        color: #ff0000;

        font-size: 22px;

        }

        input:focus{

            background: blue;

        }

        input:required{

            background: yellow;

        }

        input:checked{

            margin: 50px;

        }

        input:disabled{

            margin: 50px;

        }

    </style>

</head>

<body>

<ul>

    <li>Item 1</li>

    <li>Item 2</li>

    <li>Item 3</li>

    <li>Item 4</li>

    <li>Item 5</li>

    <li>Item 6</li>

    <li>Item 7</li>

</ul>

<div>

    <label for="txtUsername">User Name : </label>

    <input type="text" id="txtUsername"/>

</div><br>

<div>

    <label for="txtFirstname">First Name : </label>

    <input required type="text" id="txtFirstname"/>

</div>

<div>

    <input checked type="checkbox" id="txtFirstname"/>

</div>

<div>

    <input disabled type="checkbox" id="txtFirstname"/>

</div>

<p>When you hover over the first link below, it will change color and font size:</p>

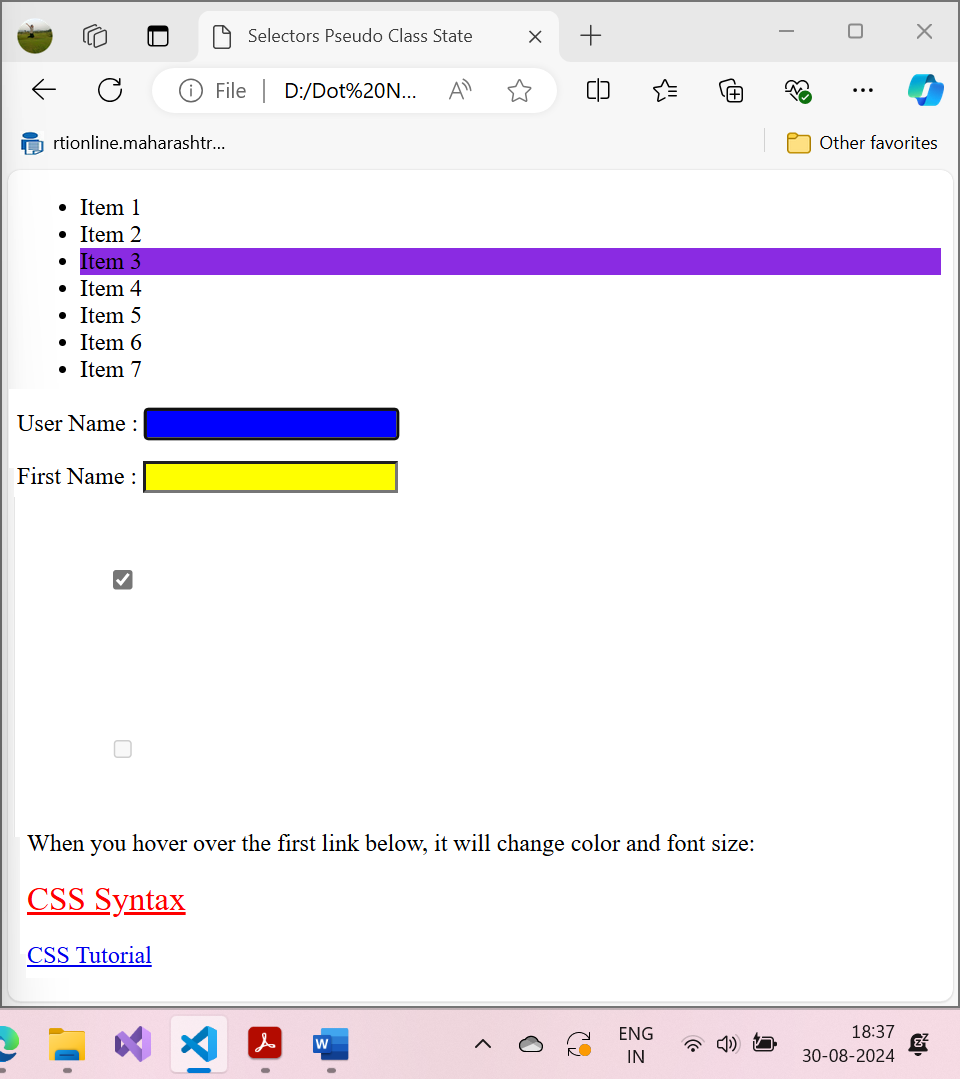
<p><a class="highlight" href="css\_syntax.asp">CSS Syntax</a></p>

<p><a href="default.asp">CSS Tutorial</a></p>

</body>

</html>

Output:



1. Pseudo Class position / other

Example:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Pseudo Class Position</title>

    <style>

        li:first-child{

            background: red;

        }

        li:last-child{

            background: green;

        }

        li:nth-child(5){

            background: yellow;

        }

        li:nth-child(2n){

            background: rgb(111, 0, 255);

        }

        li:nth-child(5n-1){

            background: maroon;

        }

        li:nth-last-child(4n-1){

            background: black;

            color: white;

        }

        span:only-child{

            font-size: 33px;

        }

        p:first-of-type{

            font-style: italic;

        }

    </style>

</head>

<body>

    <span>This is span 1</span>

    <div>

        <span>

            <p>Hello inside span 2</p>

        </span>

    </div>

    <ul>

        <li>Item 1</li>

        <li>Item 2</li>

        <li>Item 3</li>

        <li>Item 4</li>

        <li>Item 5</li>

        <li>Item 6</li>

        <li>Item 7</li>

        <li>Item 8</li>

        <li>Item 9</li>

        <li>Item 10</li>

        <li>Item 11</li>

        <li>Item 12</li>

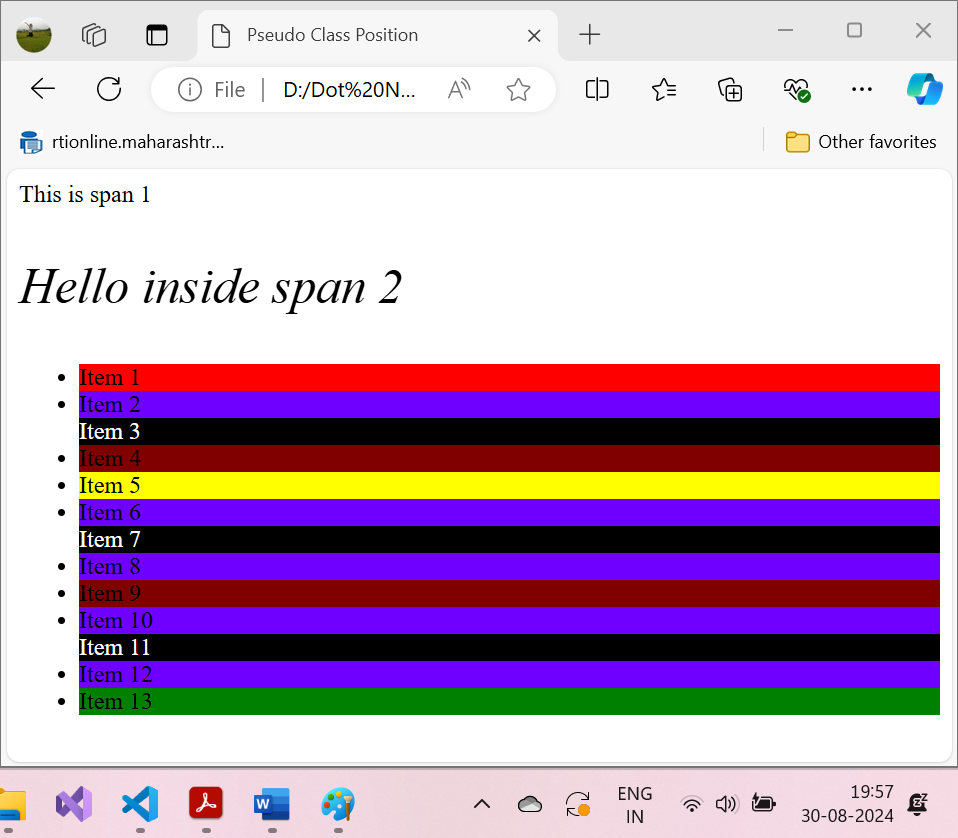
        <li>Item 13</li>

    </ul>

</body>

</html>

Output:



**COLOR Code:**

In CSS, there are several ways to assign colors to elements. Here are the most common methods:

**1. Hexadecimal Colors**

Hex color codes start with a # followed by six hexadecimal digits. It is one of the most widely used ways to define colors.

**element {**

**color: #FF5733; /\* Bright Orange \*/**

**}**

**Shortened Hex Code:** If the hex code consists of pairs of repeating digits, you can shorten it.

**element {**

**color: #F53; /\* Same as #FF5533 \*/**

**}**

**2. RGB Colors**

RGB stands for Red, Green, and Blue. You specify values between 0 and 255 for each color channel.

**element {**

**color: rgb(255, 87, 51); /\* Bright Orange \*/**

**}**

**3. RGBA Colors**

RGBA is an extension of RGB with an alpha (transparency) channel. The alpha value ranges from 0 (completely transparent) to 1 (completely opaque).

**element {**

**color: rgba(255, 87, 51, 0.7); /\* Bright Orange with 70% opacity \*/**

**}**

**4. HSL Colors**

HSL stands for Hue, Saturation, and Lightness. It is often more intuitive for adjusting the tone and vibrancy of colors.

**element {**

**color: hsl(14, 100%, 60%); /\* Bright Orange \*/**

**}**

**5. HSLA Colors**

HSLA is an extension of HSL with an alpha channel, allowing you to define transparency.

**element {**

**color: hsla(14, 100%, 60%, 0.7); /\* Bright Orange with 70% opacity \*/**

**}**

**6. Named Colors**

CSS supports a wide range of predefined color names. These are easy to use but provide limited options.

**element {**

**color: orange;**

**}**

**7. CSS Variables (Custom Properties)**

You can define custom properties (variables) for colors and reuse them throughout your styles.

**:root {**

**--main-color: #FF5733;**

**}**

**element {**

**color: var(--main-color);**

**}**

**8. CurrentColor Keyword**

You can use currentColor to apply the current text color to other properties (e.g., borders or background).

**element {**

**border: 2px solid currentColor;**

**}**

**9. System Colors (Deprecated)**

CSS supports system colors that reflect the user's system color scheme (deprecated in favor of custom properties and prefers-color-scheme).

**element {**

**color: ButtonText; /\* The color used for text on buttons in the system UI \*/**

**}**

**Usage Note:**

Each method has its own use case. **Hexadecimal and RGB/HSLA are great for precise control over color.** Named colors are convenient for simple, commonly used colors, while custom properties make it easier to maintain consistent color schemes across your styles.

**Example:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Document</title>

</head>

<body>

<p>Same as color name "Tomato":</p>

<h1 style="background-color:rgb(255, 99, 71);">rgb(255, 99, 71)</h1>

<h1 style="background-color:#ff6347;">#ff6347</h1>

<h1 style="background-color:hsl(9, 100%, 64%);">hsl(9, 100%, 64%)</h1>

<p>Same as color name "Tomato", but 50% transparent:</p>

<h1 style="background-color:rgba(255, 99, 71, 0.5);">rgba(255, 99, 71, 0.5)</h1>

<h1 style="background-color:hsla(9, 100%, 64%, 0.5);">hsla(9, 100%, 64%, 0.5)</h1>

<p>In addition to the predefined color names, colors can be specified using RGB, HEX, HSL, or even transparent colors using RGBA or HSLA color values.</p>

<p> HSL (Hue, Saturation, Lightness) is a way to define colors in CSS that is often more intuitive than the RGB model. Here's how it works:

<br>

<b>Hue:</b> The color itself, represented as an angle on the color wheel. It is a degree between 0 and 360.

<br>For example:

<br>

0° is red

120° is green

240° is blue<br>

<b>Saturation:</b> The intensity of the color, represented as a percentage. 100% is full color, while 0% is a shade of gray.

<br>

<b>Lightness:</b> The lightness of the color, also represented as a percentage. 0% is black, 100% is white, and 50% is the normal color.

<br>

<b>Alpha:</b> A number between 0 and 1 representing the opacity, where 0 is fully transparent and 1 is fully opaque.

HSL Syntax in CSS

css

Copy code

element {

    color: hsl(hue, saturation%, lightness%);

}</p>

<div>

    <h1 style="background-color:Tomato;">Tomato</h1>

    <h1 style="background-color:Orange;">Orange</h1>

    <h1 style="background-color:DodgerBlue;">DodgerBlue</h1>

    <h1 style="background-color:MediumSeaGreen;">MediumSeaGreen</h1>

    <h1 style="background-color:Gray;">Gray</h1>

    <h1 style="background-color:SlateBlue;">SlateBlue</h1>

    <h1 style="background-color:Violet;">Violet</h1>

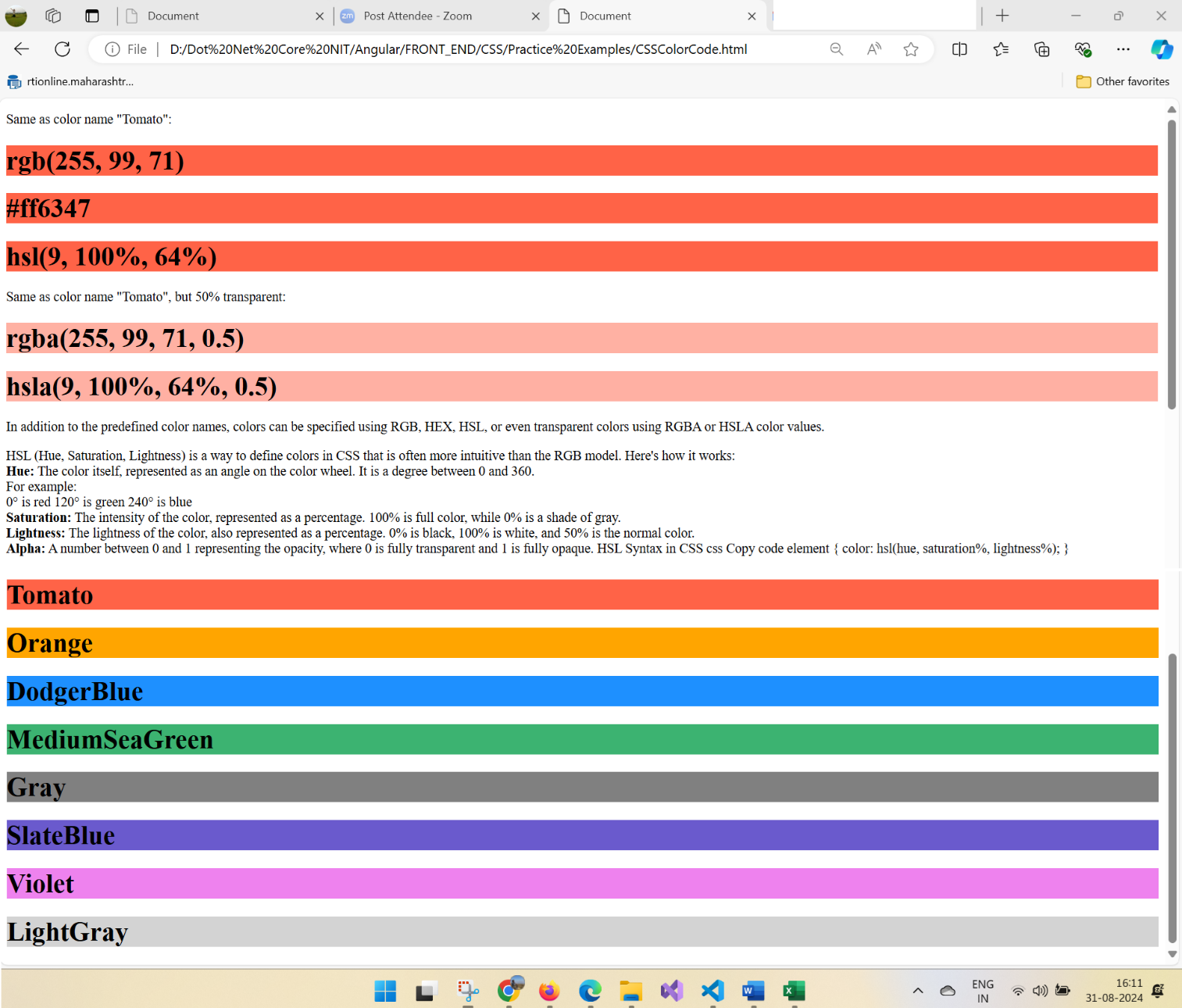
    <h1 style="background-color:LightGray;">LightGray</h1>

</div>

</body>

</html>

Output:



**CSS BOX MODEL:**

Use color wheel for best color combination:

<https://www.canva.com/colors/color-wheel/>

The CSS box model is essentially a box that wraps around every HTML element. It consists of: content, padding, borders and margins. The image below illustrates the box model:

**Let’s break down the key components:**

* **Content**: The actual data in text, images, or other media forms can be sized using the width and height property.
* **Padding**: [Padding](https://www.geeksforgeeks.org/css-margins-padding/#:~:text=CSS%20Padding%3A%20CSS%20paddings%20are%20used%20to%20create%20space%20around%20the%20element%2C%20inside%20any%20defined%20border.%20We%20can%20set%20different%20paddings%20for%20individual%20sides(top%2C%20right%2C%20bottom%2C%20left).%20It%20is%20important%20to%20add%20border%20properties%20to%20implement%20padding%20properties.) is used to create space around the element, inside any defined border.
* **Border**: The [border](https://www.geeksforgeeks.org/css-borders/) is used to cover the content & any padding, & also allows setting the style, color, and width of the border.
* **Margin**: [Margin](https://www.geeksforgeeks.org/css-margins-padding/#:~:text=through%20the%20example.-,CSS%20Margins,-%3A%20CSS%20margins) is used to create space around the element ie., around the border area.

**1. Content Area**

* Contains the actual data, such as text, images, or other media.
* Sized using the width and height properties.
* Bounded by the content edge.

**2. Padding Area**

* Surrounds the content area.
* Space within the border box.
* Dimensions are determined by the width and height of the padding box.

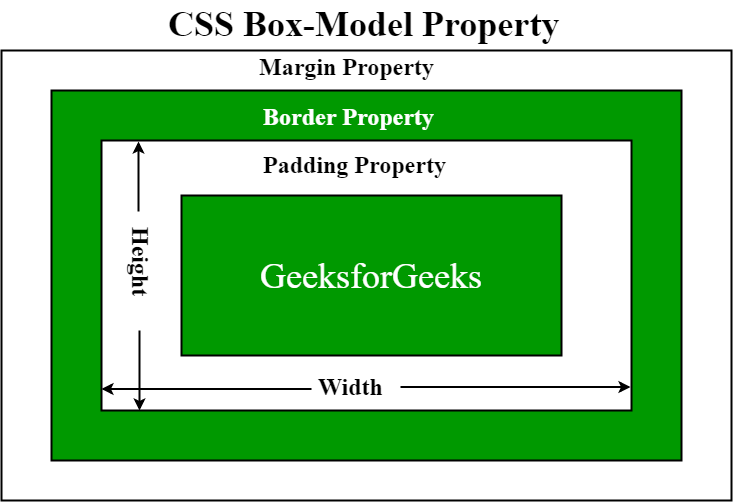
**3. Border Area**

* Lies between the padding and margin.
* Width and height are defined by the border.

**4. Margin Area**

* Separates the element from adjacent elements.
* Dimensions specified by the margin-box width and height.

The following figure illustrates the **Box model**in CSS.



**Example:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Box Model</title>

    <style>

    .box-model-example {

        width: 200px;               /\* Content width \*/

        padding: 70px;              /\* Space inside the box \*/

        border: 13px solid #3498db;  /\* Border thickness and color \*/

        margin: 100px;               /\* Space outside the box \*/

        background-color: tomato;  /\* Background color for visibility \*/

    }

    </style>

</head>

<body>

    <div class="box-model-example">

        This is a box model example.

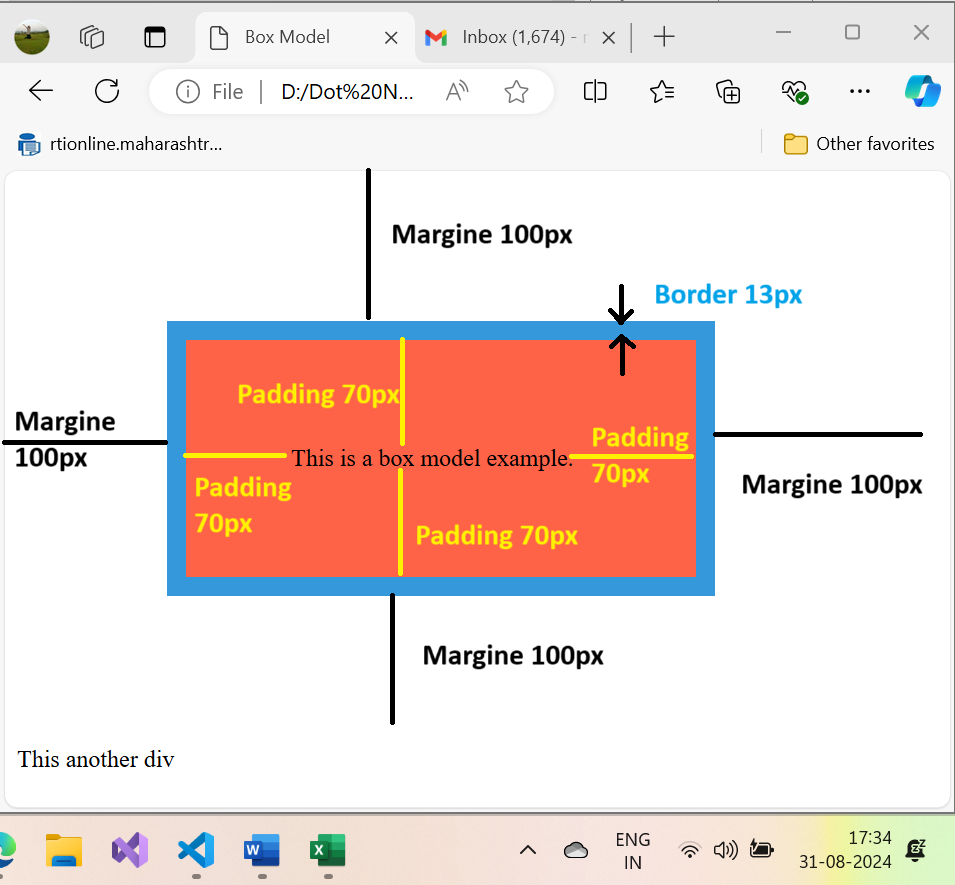
    </div>

    <div>This another div</div>

</body>

</html>

**Output:**



**CSS Units:**

CSS units are used to define the size, positioning, spacing, and other visual aspects of elements on a webpage. Understanding these units is crucial for creating flexible and responsive layouts. CSS provides a variety of units that can be broadly classified into **absolute units** and **relative units**.

**1. Absolute Units**

Absolute units are fixed units of measurement. They are not affected by other elements or the viewport size, making them less flexible for responsive design.

* **px (Pixels)**: The most commonly used unit for web design. Pixels are a fixed unit, meaning that 1px represents one pixel on the screen. Example: width: 100px;
* **pt (Points)**: Typically used for print, 1pt equals 1/72 of an inch. Example: font-size: 12pt;
* **cm (Centimeters)**: Measurement in centimeters, useful for print styles. Example: width: 2cm;
* **mm (Millimeters)**: Measurement in millimeters. Example: height: 10mm;
* **in (Inches)**: Measurement in inches. Example: width: 1in;
* **pc (Picas)**: 1pc equals 12 points. Typically used in print design. Example: font-size: 1pc;

**2. Relative Units**

Relative units are based on the size of other elements, such as the parent container or the viewport. These units are essential for creating responsive designs.

* **% (Percentage)**: A percentage of the parent element’s size. Example: width: 50%; (50% of the parent element's width).
* **em**: Relative to the font size of the element. For example, if the font size of an element is 16px, 1em will be equal to 16px. Example: padding: 2em; (which equals twice the font size).
* **rem (Root em)**: Relative to the font size of the root element (<html>). If the root element’s font size is 16px, 1rem equals 16px throughout the document. Example: margin: 1.5rem;
* **vw (Viewport Width)**: A percentage of the viewport's width. 1vw equals 1% of the viewport's width. Example: width: 50vw; (50% of the viewport width).
* **vh (Viewport Height)**: A percentage of the viewport's height. 1vh equals 1% of the viewport's height. Example: height: 50vh; (50% of the viewport height).
* **vmin**: The smaller of vw and vh. Example: font-size: 5vmin;
* **vmax**: The larger of vw and vh. Example: width: 10vmax;
* **ch**: Relative to the width of the character 0 in the current font. Example: width: 40ch; (Width equivalent to 40 characters).
* **ex**: Relative to the height of the lowercase letter x in the current font. Example: height: 2ex;
* **lh (Line Height)**: Relative to the line-height of the element. Example: margin-top: 2lh;

**3. Time Units**

Time units are used for animations and transitions.

* **s (Seconds)**: Defines time in seconds. Example: transition-duration: 2s;
* **ms (Milliseconds)**: Defines time in milliseconds. Example: animation-duration: 500ms;

**4. Angle Units**

Angle units are used for defining rotations and gradients.

* **deg (Degrees)**: A full circle is 360 degrees. Example: transform: rotate(45deg);
* **rad (Radians)**: A full circle is 2π radians. Example: transform: rotate(1rad);
* **grad (Gradians)**: A full circle is 400 gradians. Example: transform: rotate(100grad);
* **turn**: A full circle is 1 turn. Example: transform: rotate(0.5turn); (Half a circle or 180 degrees).

**5. Frequency Units**

Frequency units are used for audio-related properties.

* **Hz (Hertz)**: Frequency in hertz. Example: animation-timing-function: steps(10, jump-none);
* **kHz (Kilohertz)**: Frequency in kilohertz.

**6. Resolution Units**

Resolution units are used in media queries and image resolution settings.

* **dpi (Dots Per Inch)**: Resolution in dots per inch. Example: min-resolution: 300dpi;
* **dpcm (Dots Per Centimeter)**: Resolution in dots per centimeter. Example: min-resolution: 118dpcm;
* **dppx (Dots Per Pixel)**: Resolution in dots per pixel (typically used for screens). Example: min-resolution: 2dppx;

**Example:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Document</title>

    <style>

        .box {

            width: 50%;                 /\* 50% of the parent element's width \*/

            padding: 1em;               /\* Padding relative to the element's font size \*/

            margin: 233px;               /\* Margin in pixels \*/

            font-size: 1.5rem;          /\* Font size relative to the root element's font size \*/

            border: 2px solid #3498db;  /\* Border with fixed pixel size \*/

            height: 50vh;               /\* Height relative to 50% of the viewport height \*/

            transform: rotate(33deg);   /\* Rotation by 45 degrees \*/

            animation-duration: 2s;     /\* Animation duration in seconds \*/

            color: blue;

            background: lightgreen;

        }

    </style>

</head>

<body>

 <h3 class="box">This combination of CSS units allows for precise control over element dimensions, positioning, and responsiveness in web design.</h3>

</body>

</html>

**Output:**



**Topography:**

Typography in CSS refers to the styling of text on a webpage, which includes properties like font selection, font size, line height, letter spacing, and more. Typography plays a crucial role in web design, influencing the readability, aesthetics, and user experience of a website.

**Key Typography Properties in CSS**

1. **font-family**: Specifies the font to be used for text.
2. **font-size**: Sets the size of the font.
3. **font-weight**: Defines the thickness of the font.
4. **font-style**: Specifies whether the font should be italic, oblique, or normal.
5. **line-height**: Sets the height of lines of text, impacting readability.
6. **letter-spacing**: Controls the space between characters.
7. **word-spacing**: Controls the space between words.
8. **text-align**: Aligns text within its containing element (left, right, center, justify).
9. **text-decoration**: Adds decoration to text, such as underline, overline, or strikethrough.
10. **text-transform**: Controls capitalization (uppercase, lowercase, capitalize).
11. **color**: Sets the color of the text.

**Example of Typography in CSS**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Topography Demo</title>

    <style>

    body {

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

        color: #3333; /\* Global text color \*/

        line-height: 1.6; /\* Global line height for better readability \*/

    }

    .typography-example h1 {

        font-size: 2.5rem; /\* Large heading size \*/

        font-weight: bold; /\* Bold heading \*/

        text-align: center; /\* Center-align the heading \*/

        color: #3498db; /\* Heading color \*/

    }

    .typography-example .lead-text {

        font-size: 1.25rem; /\* Larger font size for emphasis \*/

        line-height: 1.8; /\* Increase line height for better spacing \*/

        margin-bottom: 20px; /\* Add space after the paragraph \*/

    }

    .typography-example .highlight {

        font-weight: 700; /\* Bold text for highlighting \*/

        color: #e74c3c; /\* Highlight color \*/

        text-transform: uppercase; /\* Transform text to uppercase \*/

        letter-spacing: 1px; /\* Increase spacing between letters \*/

    }

    .typography-example p {

        font-size: 1rem; /\* Default paragraph size \*/

        margin-bottom: 15px; /\* Space between paragraphs \*/

    }

    </style>

</head>

<body>

    <div class="typography-example">

        <h1>Typography in CSS</h1>

        <p class="lead-text">This is an example of using typography properties in CSS to style text effectively.</p>

        <p class="highlight">Typography is essential for making text readable and visually appealing.</p>

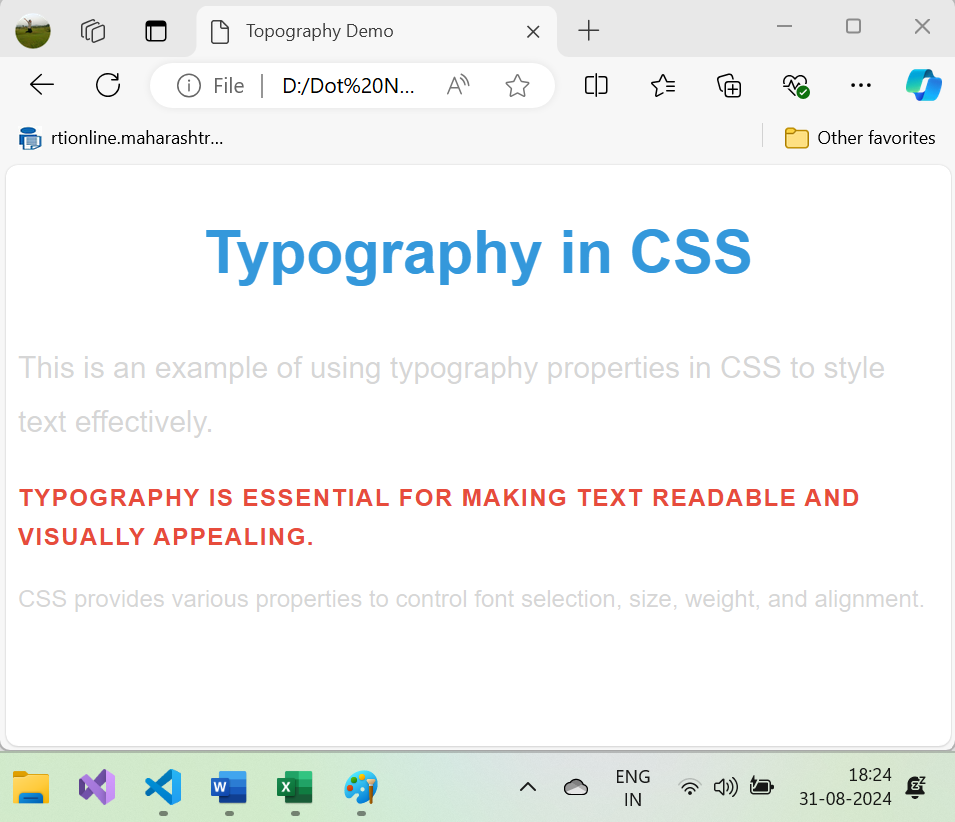
        <p>CSS provides various properties to control font selection, size, weight, and alignment.</p>

    </div>

</body>

</html>

**Output:**



**Explanation:**

* **Global Typography Settings**: The body sets a global font family (Arial, sans-serif) and text color (#333). A global line-height of 1.6 is applied for better readability across the page.
* **Headings**: The h1 heading has a larger font size (2.5rem), bold font weight, and is center-aligned. The color is set to a blue shade (#3498db).
* **Lead Text**: The class .lead-text increases the font size (1.25rem), adds extra line height (1.8), and creates spacing below the paragraph.
* **Highlight**: The .highlight class applies bold text, sets the color to red (#e74c3c), transforms the text to uppercase, and adds letter spacing (1px) for emphasis.
* **Paragraphs**: Regular paragraphs have a font size of 1rem and a margin at the bottom for spacing between paragraphs.

**Importance of Typography in Web Design:**

1. **Readability**: Good typography improves readability, making the content easy to scan and read.
2. **Aesthetics**: Typography contributes to the visual appeal of a website, creating a sense of hierarchy and structure.
3. **Brand Identity**: Typography can reinforce brand identity by using specific fonts and styles that resonate with the brand's personality.

By mastering typography in CSS, you can ensure that your text is not only functional but also visually compelling, enhancing the overall user experience of your website.

**COLORS AND BACKGROUND:**

In CSS, colors and backgrounds are essential for creating visually appealing designs. You can control the color of text, borders, backgrounds, and various other elements. CSS provides a wide range of options for setting colors and backgrounds, including solid colors, gradients, images, and more.

**Colors in CSS**

1. **Color Properties**:
   * **color**: Sets the text color.
   * **background-color**: Sets the background color of an element.
   * **border-color**: Sets the color of an element's border.
   * **outline-color**: Sets the color of an outline around an element.
2. **Color Values**:
   * **Named Colors**: CSS provides a list of predefined color names (e.g., red, blue, green).
   * **Hexadecimal Colors**: Represent colors using hex codes (e.g., #ff0000 for red).
   * **RGB Colors**: Specify colors using RGB values (e.g., rgb(255, 0, 0) for red).
   * **RGBA Colors**: Like RGB but with an additional alpha channel for transparency (e.g., rgba(255, 0, 0, 0.5) for semi-transparent red).
   * **HSL Colors**: Use hue, saturation, and lightness values to define colors (e.g., hsl(0, 100%, 50%) for red).
   * **HSLA Colors**: HSL with an alpha channel for transparency (e.g., hsla(0, 100%, 50%, 0.5)).

**Backgrounds in CSS**

1. **Background Properties**:
   * **background-color**: Sets a background color for an element.
   * **background-image**: Sets a background image for an element.
   * **background-repeat**: Controls the repetition of background images (e.g., repeat, no-repeat, repeat-x, repeat-y).
   * **background-position**: Specifies the position of the background image (e.g., top, center, bottom).
   * **background-size**: Defines the size of the background image (e.g., cover, contain, auto).
   * **background-attachment**: Determines whether the background image scrolls with the page or is fixed (e.g., scroll, fixed).
   * **background-blend-mode**: Defines how a background image blends with the background color.
2. **Background Shorthand Property**: The background property allows you to set multiple background properties in a single declaration.

**Syntax:**

background: [background-color] [background-image] [background-repeat] [background-attachment] [background-position] / [background-size];

**Example: Colors and Backgrounds in CSS**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>CSS Colors and Backgrounds Example</title>

    <style>

    body {

        background-color: #f0f0f0; /\* Light grey background \*/

        color: #333; /\* Dark grey text color \*/

        font-family: Arial, sans-serif;

    }

    .container {

        padding: 20px;

    }

    h1 {

        color: #3498db; /\* Blue text color \*/

        text-align: center;

    }

    p {

        color: #2c3e50; /\* Darker grey text color \*/

    }

    .colored-box {

        background-color: #e74c3c; /\* Red background color \*/

        border-radius: 10px 20px 30px 40px;

        color: white; /\* White text color \*/

        padding: 20px;

        margin: 20px 0;

        text-align: center;

    }

    .background-box {

        background-image: url('Coder.png'); /\* Background image \*/

        background-repeat: no-repeat;

        background-size: cover; /\* Cover the entire box \*/

        background-position: center; /\* Center the background image \*/

        border-radius: 30px;

        color: white; /\* White text color \*/

        padding: 40px;

        margin: 30px 0;

        text-align: center;

    }

    </style>

</head>

<body>

    <div class="container">

        <h1>Colors and Backgrounds in CSS</h1>

        <p>This is an example of using colors and backgrounds in CSS to style elements.</p>

        <div class="colored-box">This is a colored box.</div>

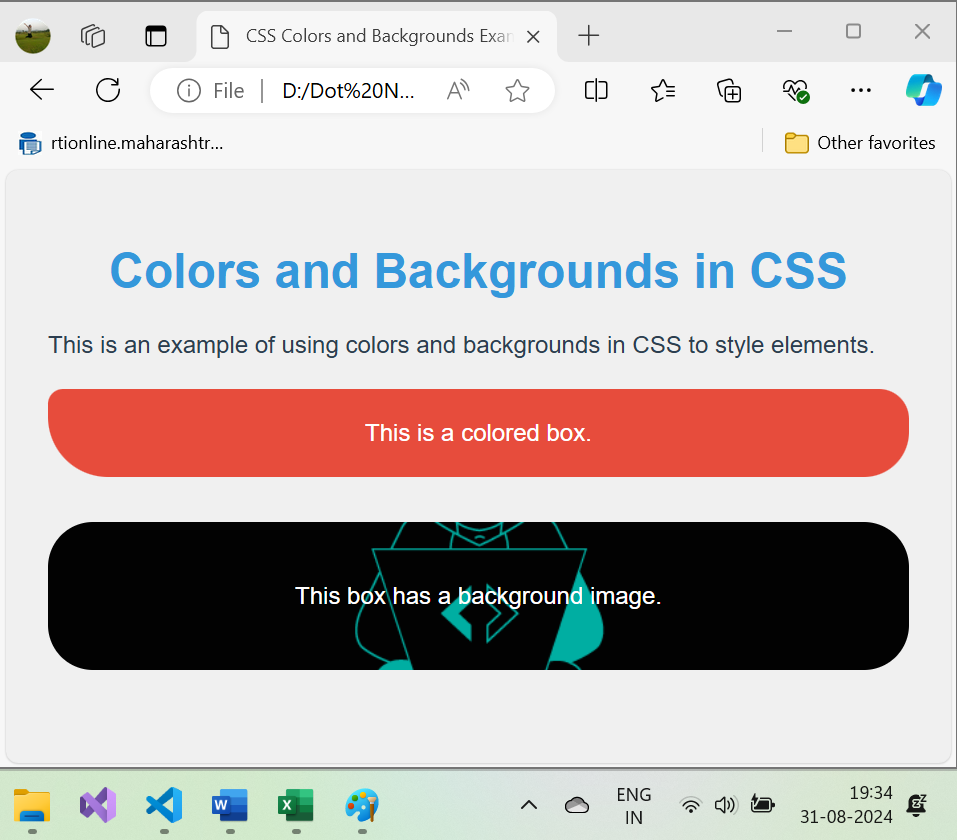
        <div class="background-box">This box has a background image.</div>

    </div>

</body>

</html>

**Output:**

****

**Explanation:**

1. **Global Background and Text Color**:
   * The body element has a light grey background color (#f0f0f0) and a dark grey text color (#333).
   * The font family is set to Arial, sans-serif.
2. **Text Color**:
   * The h1 heading has a blue color (#3498db).
   * The paragraph has a darker grey color (#2c3e50).
3. **Colored Box**:
   * The .colored-box class applies a red background color (#e74c3c).
   * The text color is set to white, and padding and margin are added for spacing.
4. **Background Image**:
   * The .background-box class uses a background image (background.jpg).
   * The image is set to not repeat (no-repeat) and cover the entire box (cover).
   * The background image is centered within the box (center), and the text color is set to white.

**Background Shorthand Example:**

You can combine background properties using the shorthand background property:

css

Copy code

.background-box {

background: url('background.jpg') no-repeat center / cover;

color: white;

padding: 40px;

margin: 20px 0;

text-align: center;

}

This shorthand property sets the background image, repeat behavior, position, and size in a single declaration.

**Conclusion:**

Colors and backgrounds in CSS are fundamental to creating visually engaging and aesthetically pleasing web pages. By using different color values and background properties, you can style your content in a way that enhances readability and user experience.

**Display Properties:**

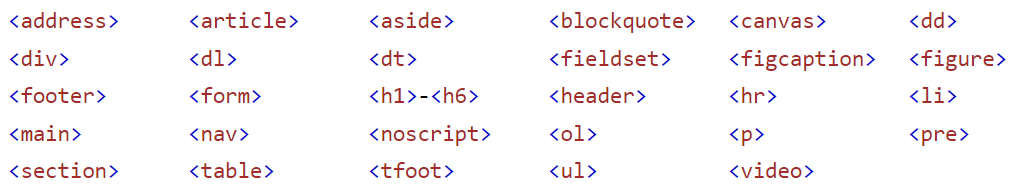
The **display** property in CSS determines how an element is rendered on the page. It controls the layout of elements and how they are arranged in relation to one another. Understanding the display property is essential for creating layouts in CSS.

**Common Display Values:**

1. **block**
2. **inline**
3. **inline-block**
4. **flex**
5. **grid**
6. **none**
7. **table**
8. **inline-flex**
9. **inline-grid**

**1. display: block**

* A block-level element takes up the full width available, starting on a new line.
* Examples of block elements:



**Example:**

.block-element {

display: block;

width: 100%;

padding: 10px;

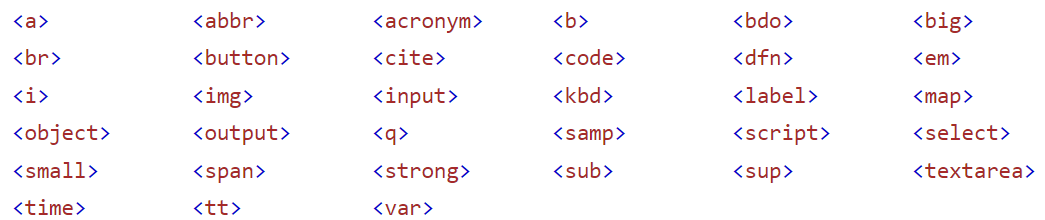
background-color: lightblue;

}

<div class="block-element">This is a block-level element</div>

**2. display: inline**

* An inline element takes up only as much width as necessary, and does not force a new line.
* Examples of inline elements:



**Example:**

.inline-element {

display: inline;

padding: 5px;

background-color: lightgreen;

}

<span class="inline-element">This is an inline element</span>

<span class="inline-element">This is another inline element</span>

**3. display: inline-block**

* Inline-block elements are similar to inline elements but they respect width and height properties. They do not start on a new line like block elements.

**Example:**

.inline-block-element {

display: inline-block;

width: 150px;

padding: 10px;

background-color: lightcoral;

margin: 5px;

}

<div class="inline-block-element">This is an inline-block element</div>

<div class="inline-block-element">This is another inline-block element</div>

**4. display: flex**

* Flexbox is a powerful layout mode designed for laying out items in one dimension (either row or column). It provides powerful alignment and distribution properties.

**Example:**

.flex-container {

display: flex;

justify-content: space-around;

background-color: lightgray;

}

.flex-item {

padding: 10px;

background-color: lightpink;

}

<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>

**5. display: grid**

* CSS Grid is another powerful layout system, providing a two-dimensional layout that allows you to arrange items into rows and columns.

**Example:**

.grid-container {

display: grid;

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

gap: 10px;

background-color: lightyellow;

}

.grid-item {

background-color: lightseagreen;

padding: 20px;

text-align: center;

}

<div class="grid-container">

<div class="grid-item">Grid Item 1</div>

<div class="grid-item">Grid Item 2</div>

<div class="grid-item">Grid Item 3</div>

<div class="grid-item">Grid Item 4</div>

<div class="grid-item">Grid Item 5</div>

<div class="grid-item">Grid Item 6</div>

</div>

**6. display: none**

* Elements with display: none are completely removed from the layout flow and are not visible on the page. This is often used to hide elements conditionally (e.g., through JavaScript).

**Example:**

.hidden-element {

display: none;

}

<div class="hidden-element">This element is hidden and won't be visible</div>

**7. display: table**

* This value makes an element behave like a table element (similar to HTML <table>).

**Example:**

.table {

display: table;

width: 100%;

background-color: lightblue;

}

.table-row {

display: table-row;

}

.table-cell {

display: table-cell;

padding: 10px;

border: 1px solid #333;

}

<div class="table">

<div class="table-row">

<div class="table-cell">Row 1, Cell 1</div>

<div class="table-cell">Row 1, Cell 2</div>

</div>

<div class="table-row">

<div class="table-cell">Row 2, Cell 1</div>

<div class="table-cell">Row 2, Cell 2</div>

</div>

</div>

**8. display: inline-flex**

* An inline-flex container behaves like an inline element, but its children are laid out using the flexbox model. It allows for more precise alignment and spacing in inline contexts.

**Example:**

.inline-flex-container {

display: inline-flex;

background-color: lightgreen;

}

.inline-flex-item {

padding: 10px;

margin: 5px;

background-color: lightcoral;

}

<div class="inline-flex-container">

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

<div class="inline-flex-item">Item 2</div>

</div>

**9. display: inline-grid**

* Similar to inline-flex, inline-grid behaves like an inline element but with grid layout capabilities.

**Example:**

.inline-grid-container {

display: inline-grid;

grid-template-columns: repeat(2, 1fr);

gap: 10px;

background-color: lightcyan;

}

.inline-grid-item {

background-color: lightsalmon;

padding: 10px;

text-align: center;

}

<div class="inline-grid-container">

<div class="inline-grid-item">Item 1</div>

<div class="inline-grid-item">Item 2</div>

</div>

**Example:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Display Properties</title>

    <style>

    .block-element {

        display: block;

        width: 100%;

        padding: 10px;

        background-color: lightblue;

    }

    .inline-element {

        display: inline;

        padding: 5px;

        background-color: lightgreen;

    }

    .inline-block-element {

        display: inline-block;

        width: 150px;

        padding: 10px;

        background-color: lightcoral;

        margin: 5px;

    }

    .flex-container {

        display: flex;

        justify-content: space-around;

        background-color: lightgray;

    }.flex-item {

        padding: 10px;

        background-color: lightpink;

    }

    .grid-container {

        display: grid;

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

        gap: 10px;

        background-color: lightyellow;

    }

    .grid-item {

        background-color: lightseagreen;

        padding: 20px;

        text-align: center;

    }

    .hidden-element {

        display: none;

    }

    .table {

        display: table;

        width: 100%;

        background-color: lightblue;

    }

    .table-row {

        display: table-row;

    }

    .table-cell {

        display: table-cell;

        padding: 10px;

        border: 1px solid #333;

    }

    .inline-flex-container {

        display: inline-flex;

        background-color: lightgreen;

    }

    .inline-flex-item {

        padding: 10px;

        margin: 5px;

        background-color: lightcoral;

    }

    .inline-grid-container {

        display: inline-grid;

        grid-template-columns: repeat(2, 1fr);

        gap: 10px;

        background-color: lightcyan;

    }

    .inline-grid-item {

        background-color: lightsalmon;

        padding: 10px;

        text-align: center;

    }

    hr {

        border: 3px solid #0df231;

        border-radius: 5px;;

    }

    </style>

</head>

<body>

    <div class="block-element">This is a block-level element</div>

    <hr>

    <span class="inline-element">This is an inline element</span>

    <span class="inline-element">This is another inline element</span>

    <hr>

    <div class="inline-block-element">This is an inline-block element</div>

    <div class="inline-block-element">This is another inline-block element</div>

    <hr>

    <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>

    <hr>

    <div class="grid-container">

        <div class="grid-item">Grid Item 1</div>

        <div class="grid-item">Grid Item 2</div>

        <div class="grid-item">Grid Item 3</div>

        <div class="grid-item">Grid Item 4</div>

        <div class="grid-item">Grid Item 5</div>

        <div class="grid-item">Grid Item 6</div>

    </div>

    <hr>

    <div class="hidden-element">This element is hidden and won't be visible</div>

    <hr>

    <div class="table">

        <div class="table-row">

            <div class="table-cell">Row 1, Cell 1</div>

            <div class="table-cell">Row 1, Cell 2</div>

        </div>

        <div class="table-row">

            <div class="table-cell">Row 2, Cell 1</div>

            <div class="table-cell">Row 2, Cell 2</div>

        </div>

    </div>

    <hr>

    <div class="inline-flex-container">

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

        <div class="inline-flex-item">Item 2</div>

    </div>

    <hr>

    <div class="inline-grid-container">

        <div class="inline-grid-item">Item 1</div>

        <div class="inline-grid-item">Item 2</div>

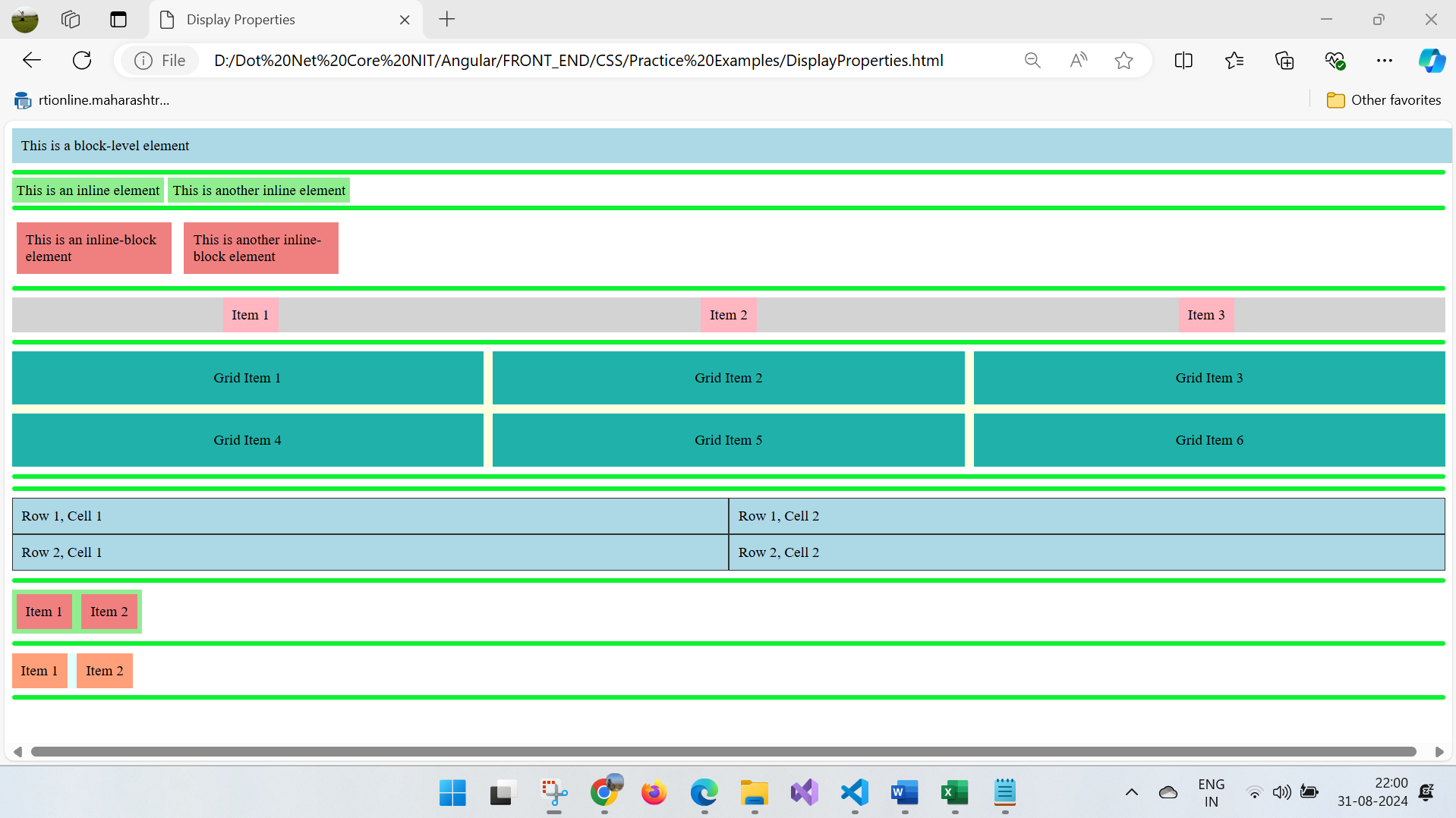
    </div>

    <hr>

</body>

</html>

**Output:**



**Example Summary:**

css

Copy code

/\* Styles for each display type \*/

.block-element {

display: block;

}

.inline-element {

display: inline;

}

.inline-block-element {

display: inline-block;

}

.flex-container {

display: flex;

}

.grid-container {

display: grid;

}

.hidden-element {

display: none;

}

.table {

display: table;

}

.inline-flex-container {

display: inline-flex;

}

.inline-grid-container {

display: inline-grid;

}

By understanding and using the display property in CSS, you can create flexible, efficient layouts for your web pages, adapting your design based on the specific needs of your content.

**CSS POSITIONING:**

CSS positioning allows you to control the placement of elements on a webpage. Understanding the different position values helps you create layouts that can adapt to various screen sizes and requirements. There are five main CSS positioning types:

1. **static** (default)
2. **relative**
3. **absolute**
4. **fixed**
5. **sticky**

**1. Position: static**

* **Default position**: All elements are positioned statically by default. This means they follow the normal flow of the document.
* **No special positioning rules** are applied.

**Example:**

.static-element {

position: static;

background-color: lightblue;

padding: 10px;

}

<div class="static-element">This is a static element (default position)</div>

**2. Position: relative**

* A **relative element** is positioned relative to its normal position in the document flow. You can adjust its position using the top, right, bottom, or left properties, but the space it would have occupied remains in the document flow.

**Example:**

.relative-element {

position: relative;

top: 20px; /\* Moves the element 20px down from its normal position \*/

left: 10px; /\* Moves the element 10px to the right \*/

background-color: lightgreen;

padding: 10px;

}

<div class="relative-element">This is a relative element</div>

**3. Position: absolute**

* An **absolute element** is positioned relative to its nearest positioned ancestor (i.e., an ancestor with a position other than static). If no such ancestor exists, it is positioned relative to the initial containing block (usually the viewport).

**Example:**

.container {

position: relative; /\* Nearest positioned ancestor \*/

height: 200px;

background-color: lightgray;

}

.absolute-element {

position: absolute;

top: 50px;

left: 50px;

background-color: lightcoral;

padding: 10px;

}

<div class="container">

<div class="absolute-element">This is an absolute element</div>

</div>

* In this example, the .absolute-element is positioned relative to the .container, which has a position: relative;.

**4. Position: fixed**

* A **fixed element** is positioned relative to the viewport, meaning it stays in the same position even when the page is scrolled. It is taken out of the normal document flow and doesn’t affect other elements on the page.

**Example:**

.fixed-element {

position: fixed;

bottom: 20px;

right: 20px;

background-color: lightsalmon;

padding: 10px;

}

<div class="fixed-element">This is a fixed element</div>

* This example demonstrates a fixed element, often used for elements like floating action buttons, sticky navigation bars, or banners.

**5. Position: sticky**

* A **sticky element** toggles between relative and fixed positioning based on the user’s scroll position. It behaves like a relative element until it reaches a certain point on the page (specified by top, left, bottom, or right), at which point it becomes fixed.

**Example:**

.sticky-element {

position: sticky;

top: 0; /\* Becomes fixed when it reaches the top of the viewport \*/

background-color: lightseagreen;

padding: 10px;

}

<div class="sticky-element">This is a sticky element</div>

* Sticky elements are useful for creating fixed headers that scroll with the page until a certain point.

**Example:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>CSS Positioning Example</title>

    <style>

        .static-element {

            position: static;

            background-color: lightblue;

            padding: 10px;

            margin-bottom: 10px;

        }

        .relative-element {

            position: relative;

            top: 20px;

            left: 10px;

            background-color: lightgreen;

            padding: 10px;

            margin-bottom: 10px;

        }

        .container {

            position: relative;

            height: 200px;

            background-color: lightgray;

            margin-bottom: 10px;

        }

        .absolute-element {

            position: absolute;

            top: 50px;

            left: 50px;

            background-color: lightcoral;

            padding: 10px;

        }

        .fixed-element {

            position: fixed;

            bottom: 20px;

            right: 20px;

            background-color: lightsalmon;

            padding: 10px;

        }

        .sticky-element {

            position: sticky;

            top: 0;

            background-color: lightseagreen;

            padding: 10px;

            margin-bottom: 20px;

        }

        .content {

            height: 1500px; /\* Long content to demonstrate scroll effects \*/

            padding: 20px;

            background-color: lightyellow;

        }

    </style>

</head>

<body>

    <div class="static-element">Static Element (default)</div>

    <div class="relative-element">Relative Element</div>

    <div class="container">

        <div class="absolute-element">Absolute Element</div>

    </div>

    <div class="sticky-element">Sticky Element</div>

    <div class="content">

        Scroll down to see the effect of the fixed element at the bottom right corner of the viewport.

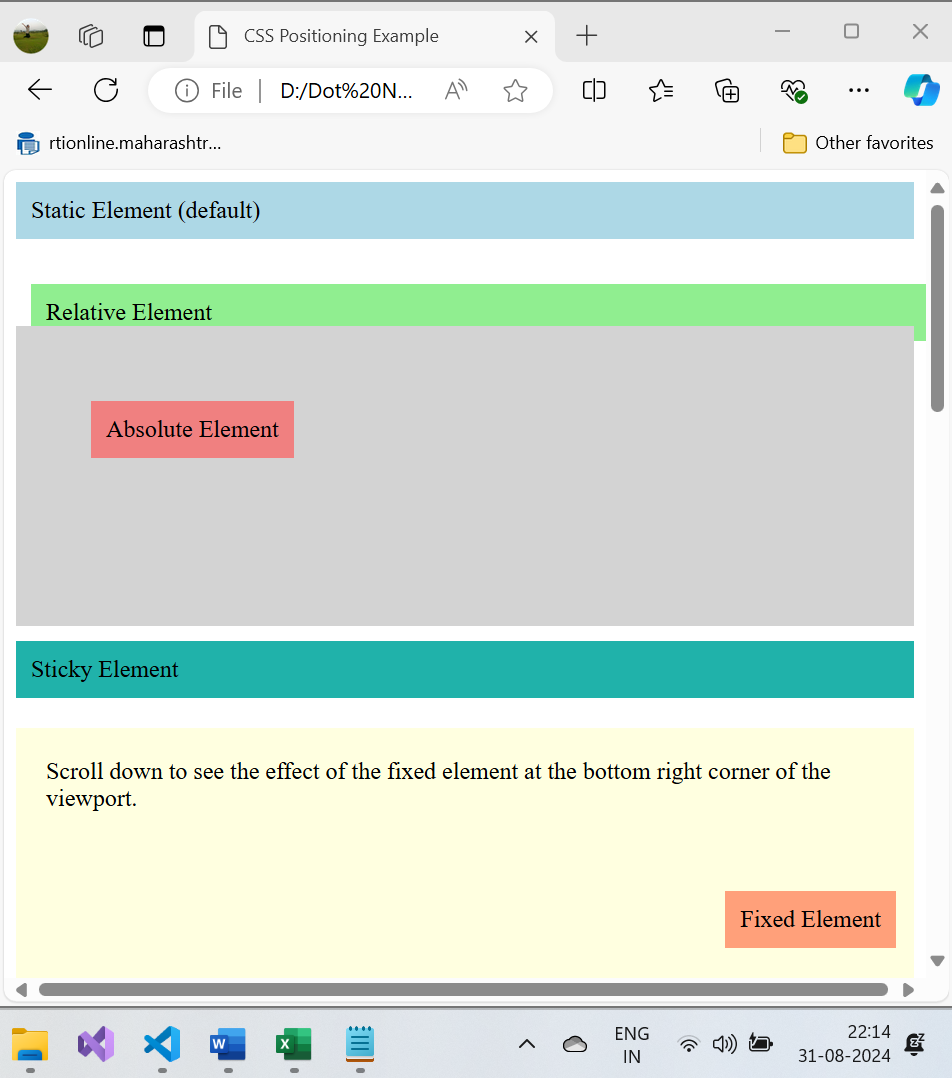
    </div>

    <div class="fixed-element">Fixed Element</div>

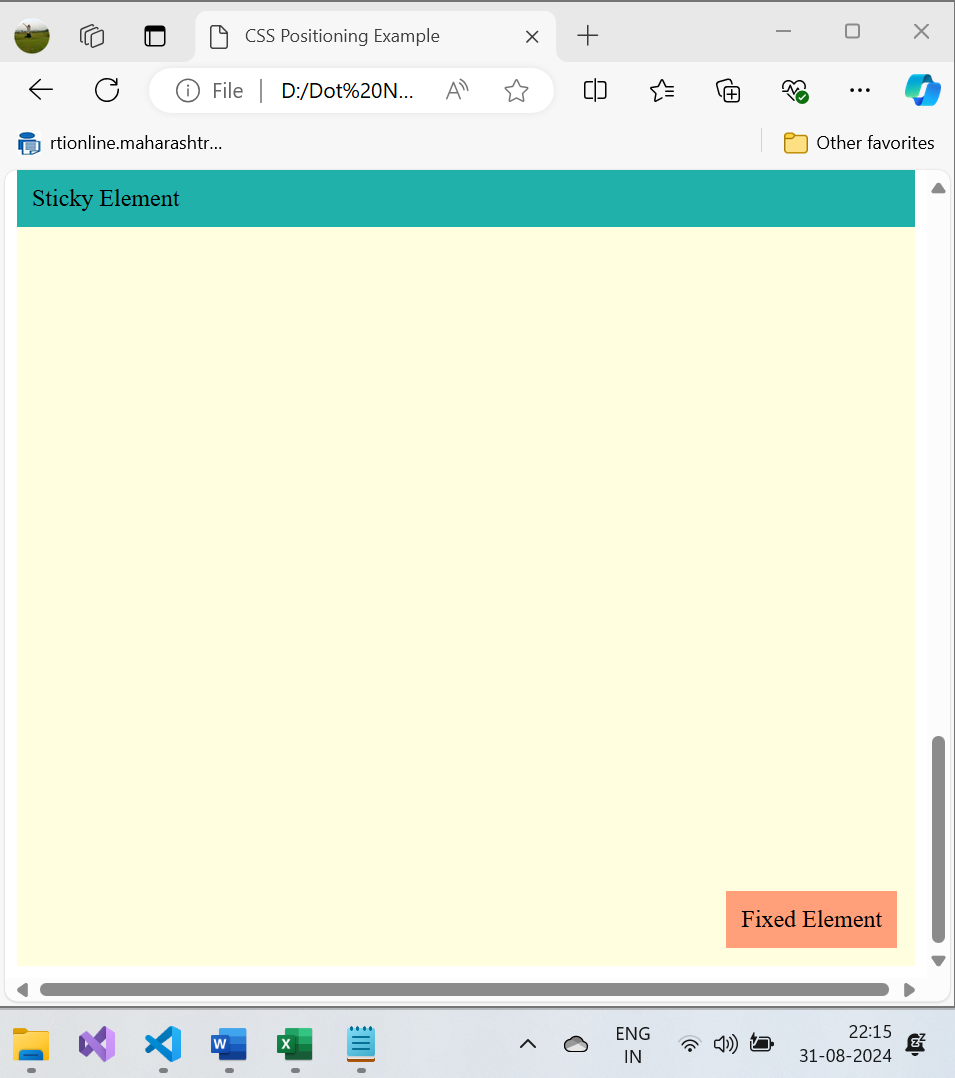
</body>

</html>

**Output:**



**After Scroll:**



**CSS FLOAT AND CLEAR:**

**CSS Floating** is a layout technique that allows you to place an element to the left or right of its container, and the content will flow around it. **Clearing** is used to control the behavior of elements that follow floating elements in the document flow.

**1. Floating in CSS**

The float property can be set to:

* left: The element floats to the left side of its container.
* right: The element floats to the right side of its container.
* none: The element does not float (this is the default).
* inherit: Inherits the float property from its parent.

**Example of Floating:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>CSS FLOAT</title>

    <style>

        /\* Floating \*/

    .float-left {

        float: left;

        width: 200px;

        margin: 10px;

        background-color: lightblue;

        border: 2px solid blue;

        padding: 30px;

    }

    .float-right {

        float: right;

        width: 200px;

        margin-left: 10px;

        background-color: lightgreen;

        border: 2px solid green;

        padding: 10px;

    }

    .content {

        background-color: lightgray;

        border: 2px solid gray;

        padding: 10px;

    }

    </style>

</head>

<body>

    <div class="float-left">This is a floated element to the left.</div>

    <div class="float-right">This is a floated element to the right.</div>

    <div class="content">This content flows around the floated elements.

        Lorem Ipsum is simply dummy text of the printing and typesetting industry.

        Lorem Ipsum has been the industry's standard dummy text ever since the 1500s,

        when an unknown printer took a galley of type and scrambled it to make a type specimen book.

        It has survived not only five centuries, but also the leap into

        electronic typesetting, remaining essentially unchanged. It was popularised

        in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages,

        and more recently with desktop publishing software like Aldus

        PageMaker including versions of Lorem Ipsum.</div>

        <hr>

    <div class="float-left"><img src="Pillow.png" alt="Coder"></div>

    <div class="float-right">This is a floated element to the right.</div>

    <div class="content">This content flows around the floated elements.

        Lorem Ipsum is simply dummy text of the printing and typesetting industry.

        Lorem Ipsum has been the industry's standard dummy text ever since the 1500s,

        when an unknown printer took a galley of type and scrambled it to make a type specimen book.

        It has survived not only five centuries, but also the leap into

        electronic typesetting, remaining essentially unchanged. It was popularised

        in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages,

        and more recently with desktop publishing software like Aldus

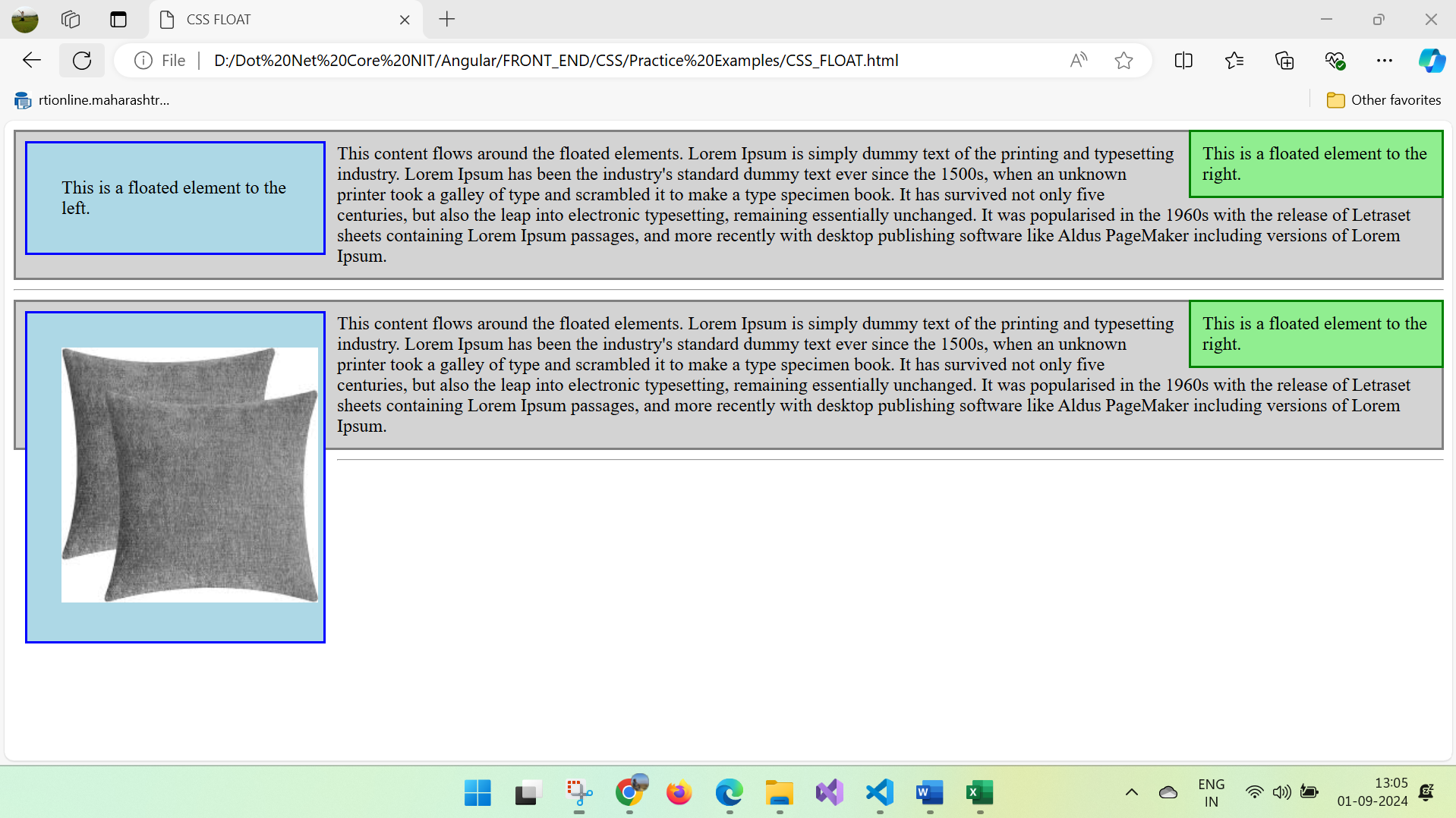
        PageMaker including versions of Lorem Ipsum.</div>

        <hr>

</body>

</html>

**OUTPUT:**



**Explanation:**

* .float-left is floated to the left, allowing the content to wrap around it on the right.
* .float-right is floated to the right, allowing the content to wrap around it on the left.
* The .content div flows around the floated elements.

**2. Clearing in CSS**

When using floats, sometimes the following elements might flow around the floated elements, creating layout issues. The clear property ensures that an element does not wrap around a floating element. The possible values for clear are:

* left: The element is moved below left-floating elements.
* right: The element is moved below right-floating elements.
* both: The element is moved below both left and right floating elements.
* none: Default value; allows floating elements to affect the element.

**Example of Clearing:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>CSS CLEAR</title>

    <style>

    /\*Clearing\*/

    .clear-both {

        clear: both;

        background-color: lightcoral;

        padding: 10px;

        margin-top: 20px;

    }

    </style>

</head>

<body>

        <div class="float-left">This is a floated element to the left.</div>

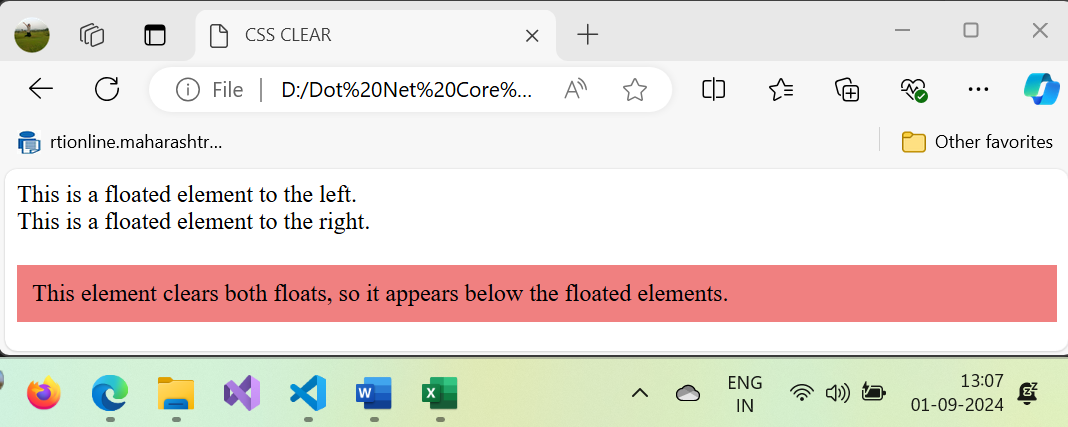
        <div class="float-right">This is a floated element to the right.</div>

        <div class="clear-both">This element clears both floats, so it appears below the floated elements.</div>

</body>

</html>

**OUTPUT:**

****

**Explanation:**

* .clear-both is used to ensure that this element appears below both the left and right floated elements.

**Practical Example: Two-column Layout with Floats**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>TWO COLUMN LAYOUT WITH FLOAT</title>

    <style>

    .container {

        width: 100%;

        background-color: rgb(227, 227, 49);

        padding: 10px;

    }

    .sidebar {

        float: left;

        width: 25%;

        background-color: lightblue;

        padding: 10px;

    }

    .main-content {

        float: right;

        width: 70%;

        background-color: lightgreen;

        padding: 10px;

    }

    .footer {

      clear: both; /\* Ensures that the footer appears below both floated elements \*/

      background-color: lightcoral;

      padding: 10px;

      text-align: center;

    }

    </style>

</head>

<body>

    <div class="container">

        <div class="sidebar">Sidebar Content</div>

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

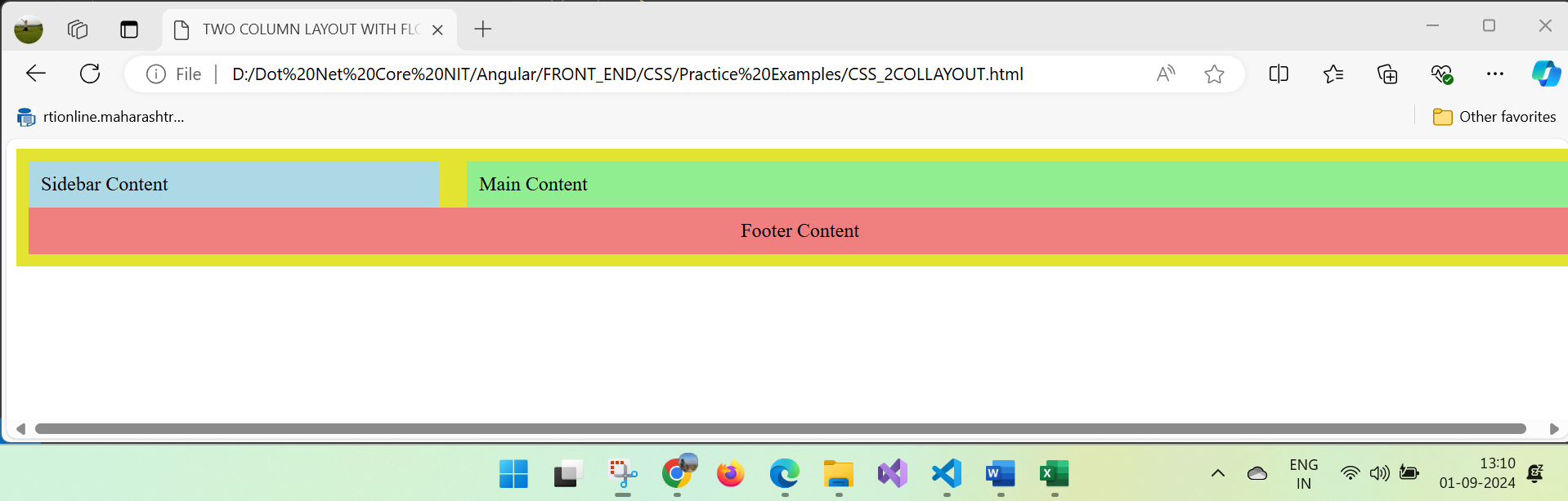
        <div class="footer">Footer Content</div>

    </div>

</body>

</html>

**OUTPUT:**



**Explanation:**

* The .sidebar is floated to the left, taking up 25% of the container's width.
* The .main-content is floated to the right, taking up 70% of the container's width.
* The .footer uses clear: both; to make sure it clears both floated elements and appears below them.

**Issues with Floating:**

* **Overflow**: If the container only contains floated elements, it may collapse (height becomes zero) because floated elements are removed from the normal document flow.
* **Clearing Fix**: To prevent the collapse of containers with floated children, you can use the **clearfix** method.

**Clearfix Example:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>TWO COLUMN LAYOUT WITH FLOAT</title>

    <style>

        .clearfix::after {

            content: "";

            display: table;

            clear: both;

        }

</style>

</head>

<body>

    <div class="container clearfix">

        <div class="sidebar">Sidebar Content</div>

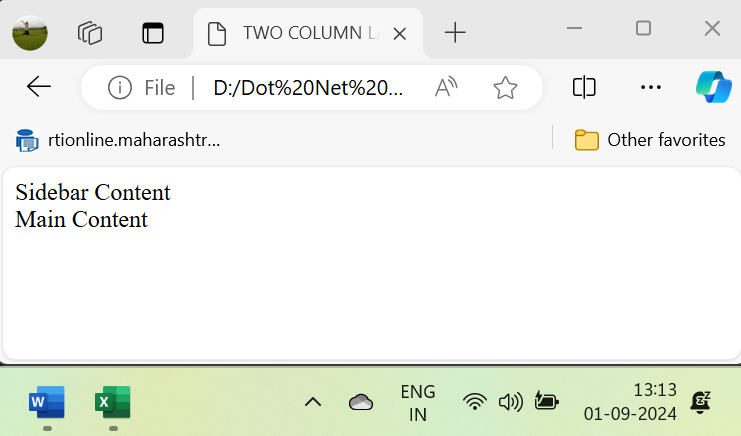
        <div class="main-content">Main Content</div>

    </div>

</body>

</html>

**OUTPUT:**

****

In this example, the clearfix technique adds an invisible element that clears the floats, allowing the container to wrap around its floated children.

**Conclusion:**

* **Floating** allows you to create simple layouts, such as columns or image-wrapped text.
* **Clearing** ensures that elements following floated elements are properly positioned in the layout.
* While floats are still useful, modern layout techniques like **flexbox** and **grid** provide more powerful and flexible ways to build responsive layouts.

**Other Topics:**

**Q. Priorities of styles that gets applied to element is hight for inline, then internal and least is external, however we can calculate the priority by using below calculations :**

[**https://www.w3schools.com/css/css\_specificity.asp**](https://www.w3schools.com/css/css_specificity.asp)

**How to Calculate Specificity?**

Memorize how to calculate specificity!

Start at 0, add 100 for each ID value, add 10 for each class value (or pseudo-class or attribute selector), add 1 for each element selector or pseudo-element.

Note: Inline style gets a specificity value of 1000, and is always given the highest priority!

Note 2: There is one exception to this rule: if you use the [!important](https://www.w3schools.com/css/css_important.asp) rule, it will even override inline styles!

The table below shows some examples on how to calculate specificity values:

|  |  |  |
| --- | --- | --- |
| **Selector** | **Specificity Value** | **Calculation** |
| P | 1 | 1 |
| p.test | 11 | 1 + 10 |
| p#demo | 101 | 1 + 100 |
| <p style="color: pink;"> | 1000 | 1000 |
| #demo | 100 | 100 |
| .test | 10 | 10 |
| p.test1.test2 | 21 | 1 + 10 + 10 |
| #navbar p#demo | 201 | 100 + 1 + 100 |
| \* | 0 | 0 (the universal selector is ignored) |

**The selector with the highest specificity value will win and take effect!**

**Consider these three code fragments:**

**Example**

A: h1  
B: h1#content  
C: <h1 id="content" style="color: pink;">Heading</h1>

The specificity of A is 1 (one element selector)  
The specificity of B is 101 (one ID reference + one element selector)  
The specificity of C is 1000 (inline styling)

Since the third rule (C) has the highest specificity value (1000), this style declaration will be applied.

**Q. Single Or Double Colon For Pseudo-Elements?**[**#**](https://www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-and-pseudo-elements/#single-or-double-colon-for-pseudo-elements)

The short answer is, in most cases, either.

The double colon (::) was introduced in [CSS3](https://www.smashingmagazine.com/learning-css3-useful-reference-guide/) to differentiate pseudo-elements such as ::before and ::after from pseudo-classes such as :hover and :active. All browsers support double colons for pseudo-elements except Internet Explorer (IE) 8 and below.

Some pseudo-elements, such as ::backdrop, accept only a double colon, though.

Personally, I use single-colon notation so that my CSS is backwards-compatible with legacy browsers. I use double-colon notation on those pseudo-elements that require it, of course.

Nesting of CSS code

@layer Directive

@container

The fundamental CSS topics are essential for building a solid foundation in web development. These core concepts cover the basics that every front-end developer should understand:

**~~1. CSS Syntax and Structure~~**

* **Selectors**: Targeting HTML elements (e.g., element, class, ID selectors).
* **Properties and Values**: Defining styles (e.g., color: blue;).
* **CSS Ruleset**: The combination of a selector and a declaration block (e.g., p { color: red; }).

**~~2. Selectors~~**

* **Basic Selectors**: Element, class, and ID selectors (e.g., h1, .class, #id).
* **Combinators**: Descendant, child, adjacent sibling, and general sibling selectors.
* **Attribute Selectors**: Target elements based on attributes (e.g., [type="text"]).

**~~3. Box Model~~**

* **Content**: The actual content of the box (text, images, etc.).
* **Padding**: Space between the content and the border.
* **Border**: A border that surrounds the padding and content.
* **Margin**: Space outside the border, creating distance from other elements.

**~~4. CSS Units~~**

* **Absolute Units**: Fixed size units (e.g., px, pt, cm).
* **Relative Units**: Scalable units relative to other elements (e.g., em, rem, %, vw, vh).

**~~5. Typography~~**

* **Font Properties**: Setting fonts, font size, font style, and font weight.
* **Line Height**: Controlling space between lines of text.
* **Text Alignment and Decoration**: Aligning text and adding decorations like underlines.

**~~6. Colors and Backgrounds~~**

* **Color**: Applying color to text, borders, and backgrounds.
* **Backgrounds**: Background color, images, position, size, and repeat properties.

**~~7. Display Property~~**

* **Block and Inline**: Understanding block (div, p) vs inline elements (span, a).
* **Inline-Block**: Combining block and inline properties.
* **None**: Hiding elements from display.

**~~8. Positioning~~**

* **Static Positioning**: The default positioning for elements.
* **Relative Positioning**: Position relative to the element's normal position.
* **Absolute Positioning**: Positioned relative to the nearest positioned ancestor.
* **Fixed Positioning**: Positioned relative to the viewport.
* **Sticky Positioning**: Combination of relative and fixed positioning.

**9. Floating and Clearing**

* **Float Property**: Floating elements to the left or right.
* **Clear Property**: Preventing elements from wrapping around floated elements.
* **Clearfix**: Technique to contain floated elements within a container.

**10. Margins and Padding**

* **Margin**: Space outside the element's border.
* **Padding**: Space inside the element's border, around the content.
* **Shorthand Properties**: Using shorthand for margin and padding (margin: 10px 20px;).

**11. Borders and Outline**

* **Borders**: Styling the element's border (width, style, color).
* **Border Radius**: Creating rounded corners.
* **Outline**: Similar to borders, but does not affect element size.

**12. Z-Index**

* **Stacking Context**: Controlling the vertical stacking order of positioned elements.
* **Z-Index Property**: Higher values place elements above lower values.

**13. Media Queries**

* **Responsive Design**: Adapting styles based on device screen size.
* **Breakpoints**: Defining specific points at which the layout should change.

**14. CSS Cascading and Inheritance**

* **Cascading**: How CSS rules are applied in the order of specificity and importance.
* **Inheritance**: How certain properties are inherited from parent elements to children (e.g., color).

**~~15. Specificity~~**

* **Specificity Calculation**: Understanding how to calculate specificity for conflicting CSS rules.
* **Importance**: Using !important to override specificity (sparingly).

**~~16. Pseudo-Classes and Pseudo-Elements~~**

* **Pseudo-Classes**: Applying styles based on state (e.g., :hover, :focus, :nth-child).
* **Pseudo-Elements**: Styling parts of an element (e.g., ::before, ::after).

**17. Shorthand Properties**

* **Background**: Combining multiple background properties.
* **Margin and Padding**: Specifying margins and padding in one line.
* **Font and Border Shorthands**: Combining font and border properties.

**18. CSS Variables (Custom Properties)**

* **Defining Variables**: Creating reusable values (e.g., --primary-color: #3498db;).
* **Using Variables**: Applying variables to style rules (color: var(--primary-color);).

These fundamental topics will give you a strong understanding of how to apply CSS to style web pages effectively. Mastering these concepts is essential for building more advanced layouts and responsive designs.