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.

What is CSS?

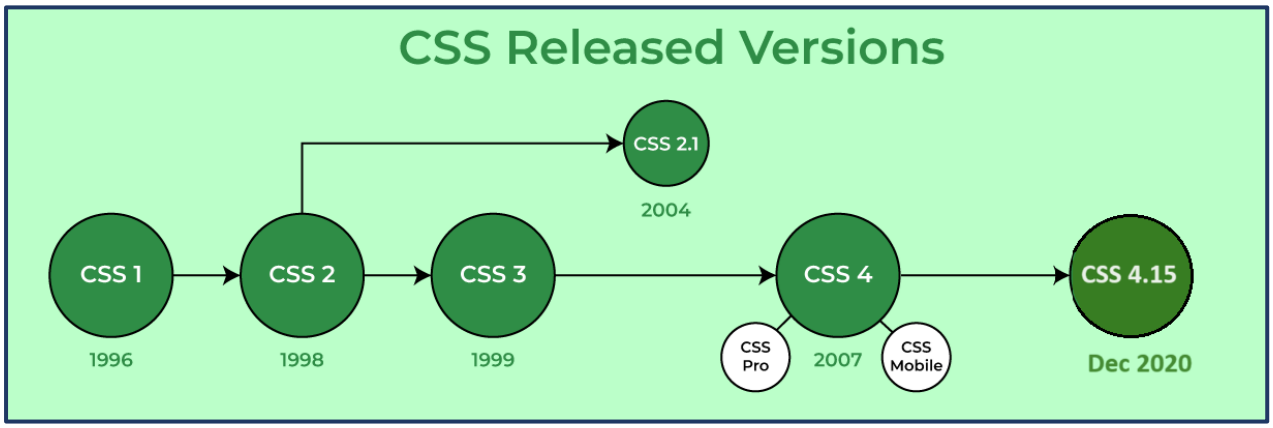
CSS, or **Cascading Style Sheets**, is a language used to style and enhance websites. It controls how **HTML elements**—such as *text*, *images*, and *buttons*—are displayed on a webpage. **With CSS**, you can adjust font sizes and colors, add backgrounds, and manage the layout, transforming a basic 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>
```

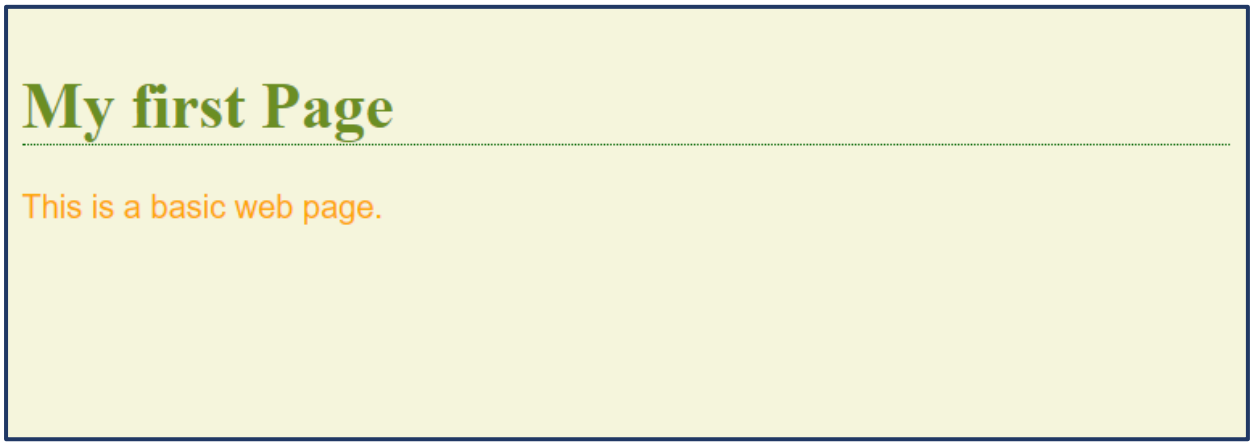
OUTPUT:

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.

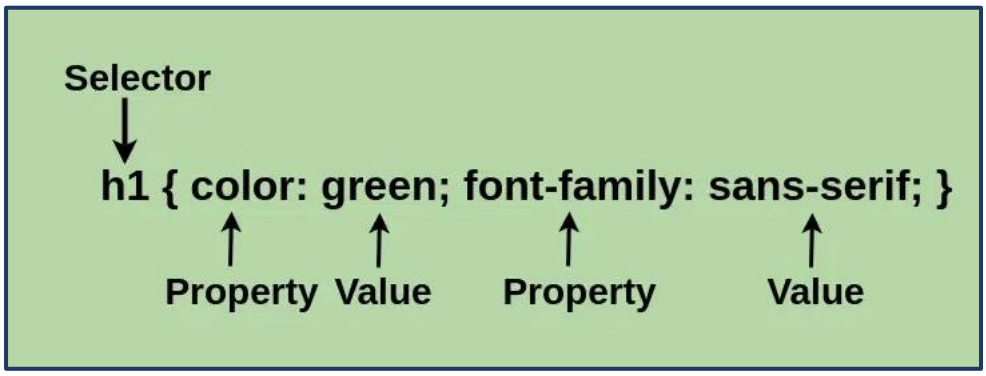
CSS Syntax

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:**

a. 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.

b. Why is CSS important?

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

c. 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.*

d. What is the syntax of CSS?

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

e. What are CSS selectors?

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

f. 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

1) Basic/Simple Selectors:

a. Universal Selector (*) : Selects all elements present on HTML page and applies the style to all elements.

b. 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.

c. 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.

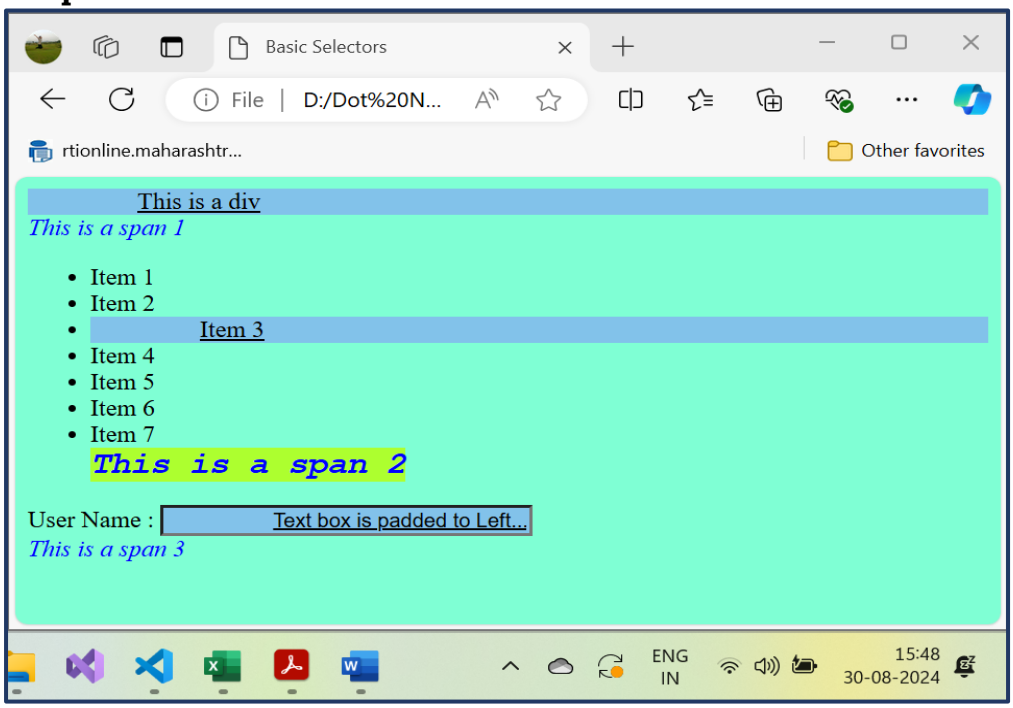
d. ID Selector (#ID) : Create a group of style properties and assign to only one element in a HTML. We cannot use ID selector more than one.

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*/
```

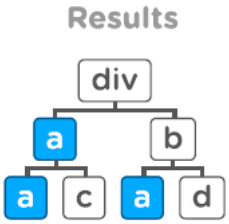
```
        #MyID {
            background: greenyellow;
            text-decoration: dotted;
            font-family: 'Courier New', Courier, monospace;
            font-weight: bolder;
            font-size: 21px;
        }
    </style>
</head>
<body>
    <div class="MyClassStyle">This is a div</div>
    <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:



2) **Combination:** A CSS selector can contain more than one Basic/Simple selector. Between the Basic/Simple selectors, we can include a combinator.

   a. **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.

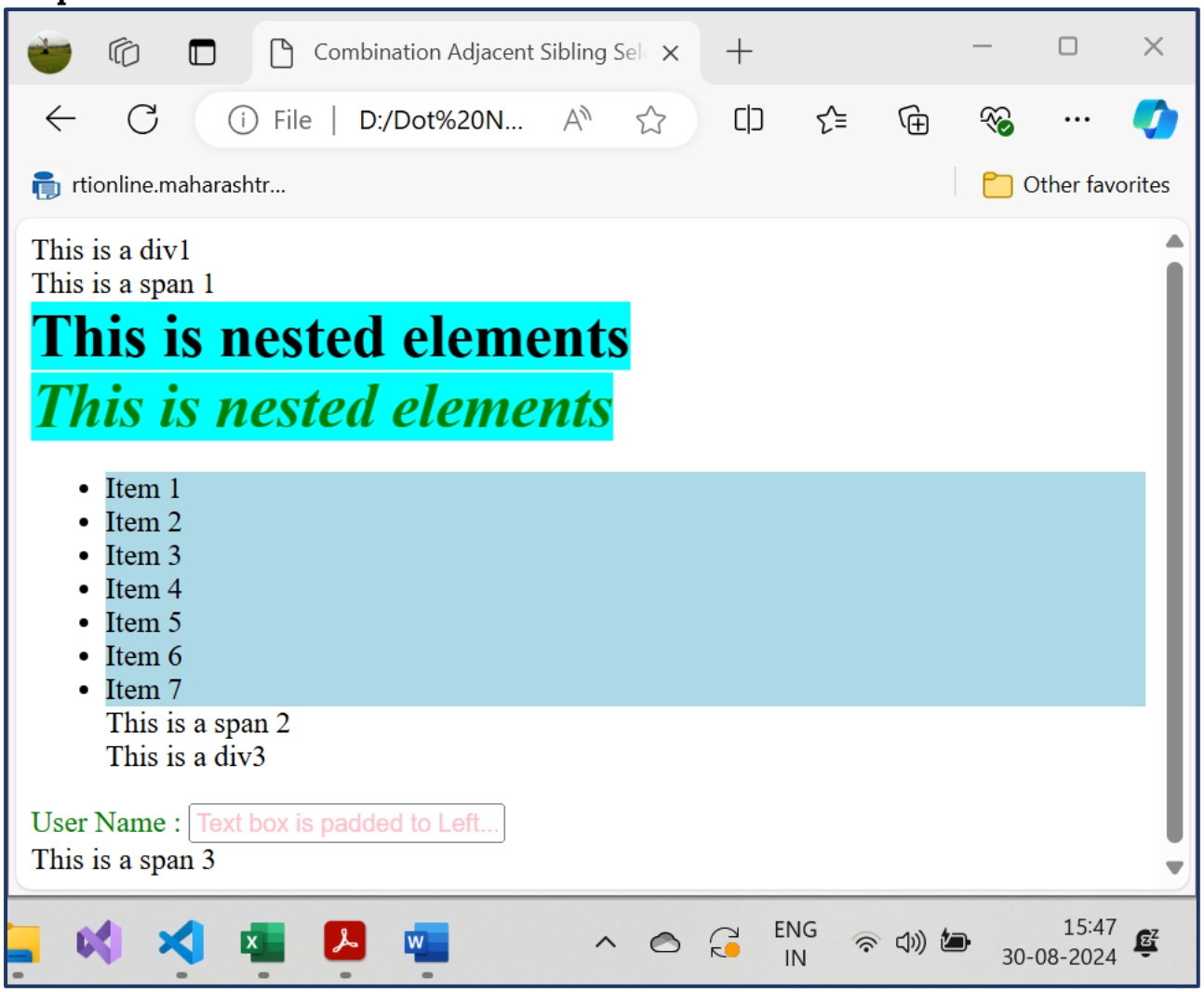


Example:

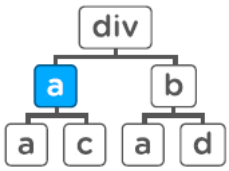
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Combination Adjacent Sibling Selectors</title>
    <style>
    /*Adjacent Sibling Selector */
    ul li{
        background: lightblue;
    }
    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:



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

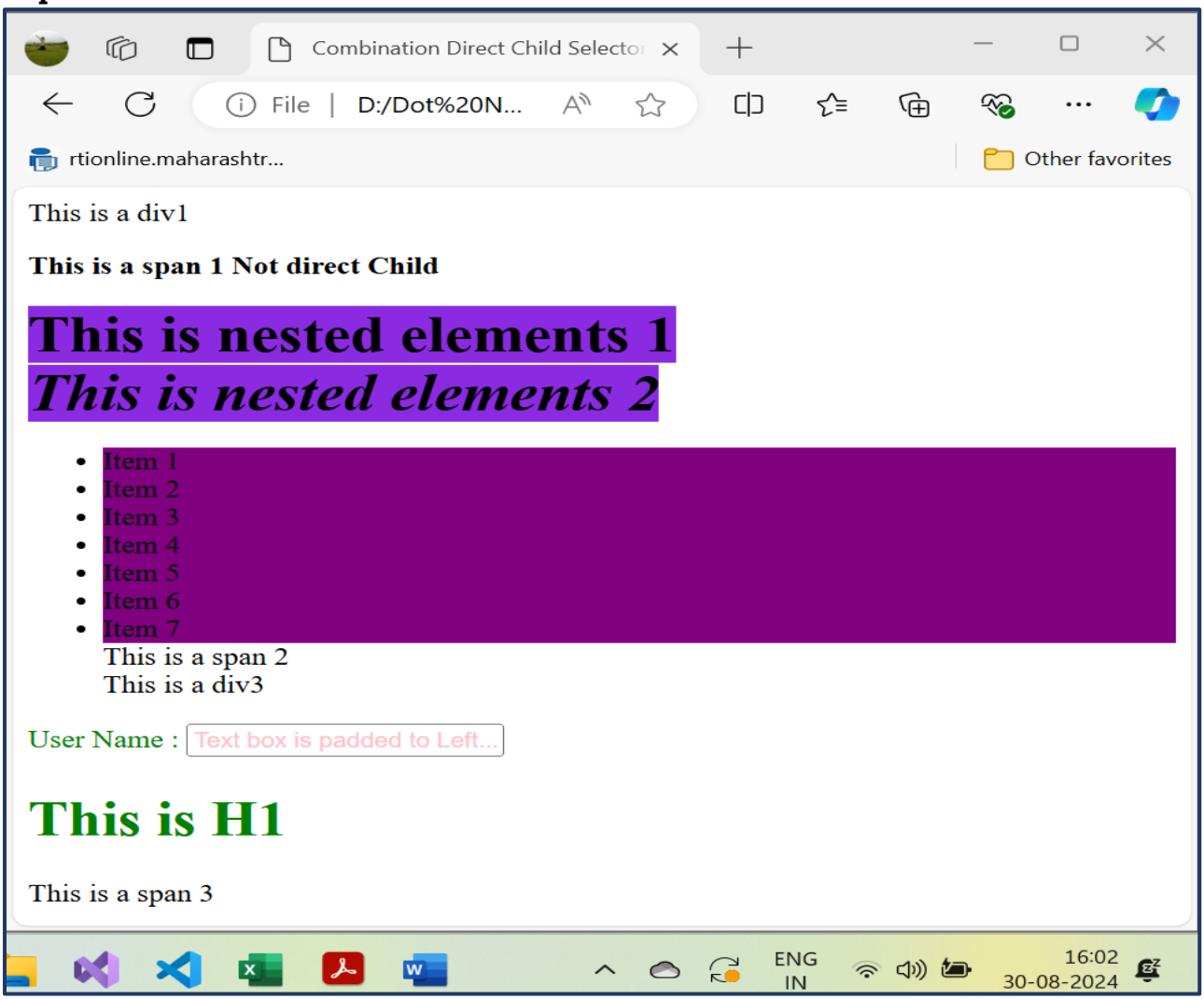


Example:

```
<!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:



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

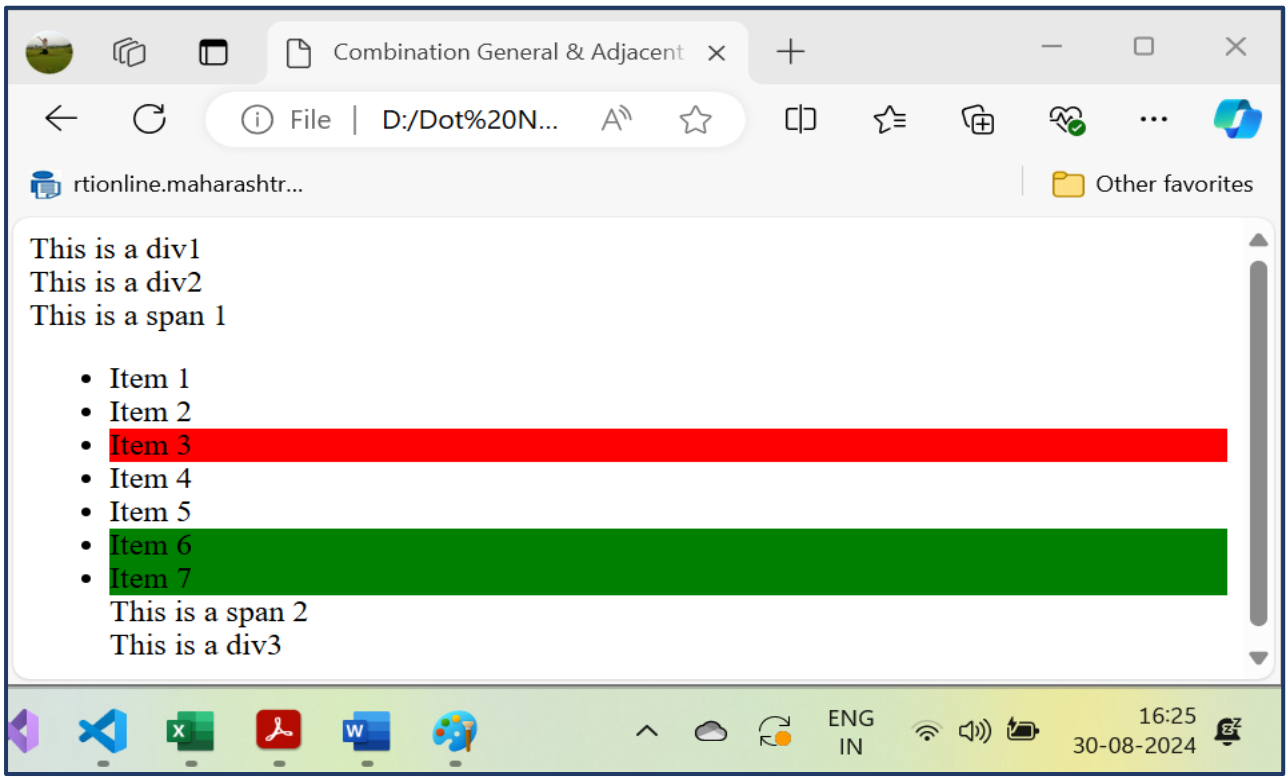
Example:
Example combined with Adjacent Siblings

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

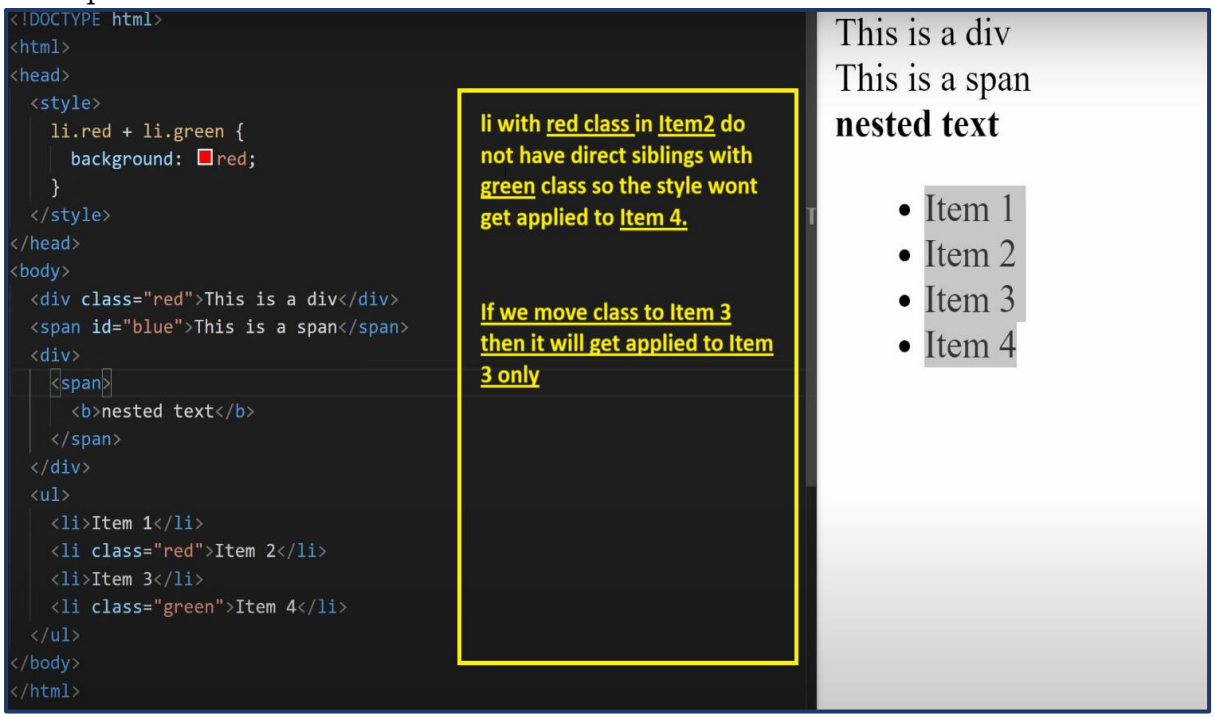
Example 1:

```
<!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:



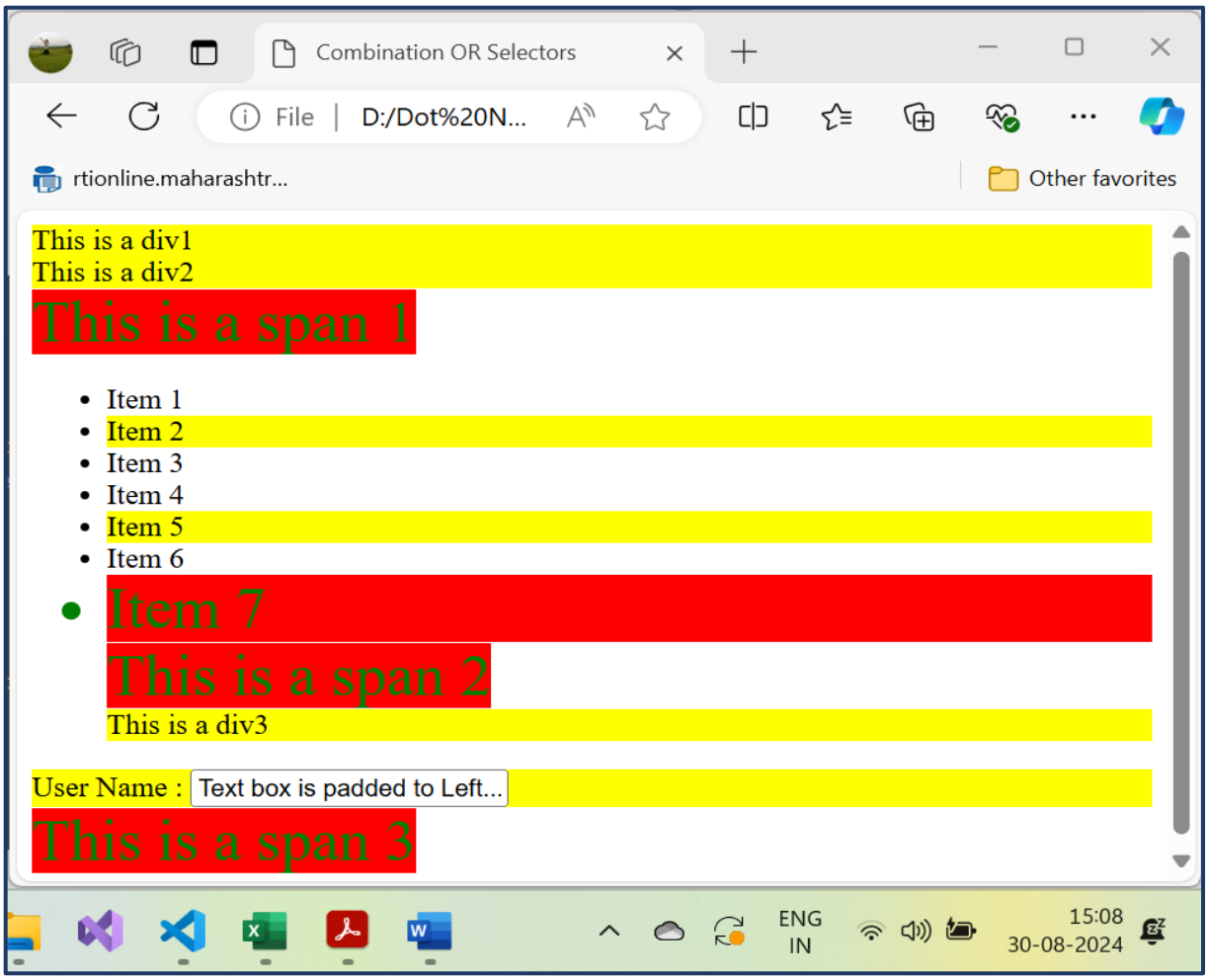
e. Or Selector (div, a)

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:



f. And Selector(div.c)

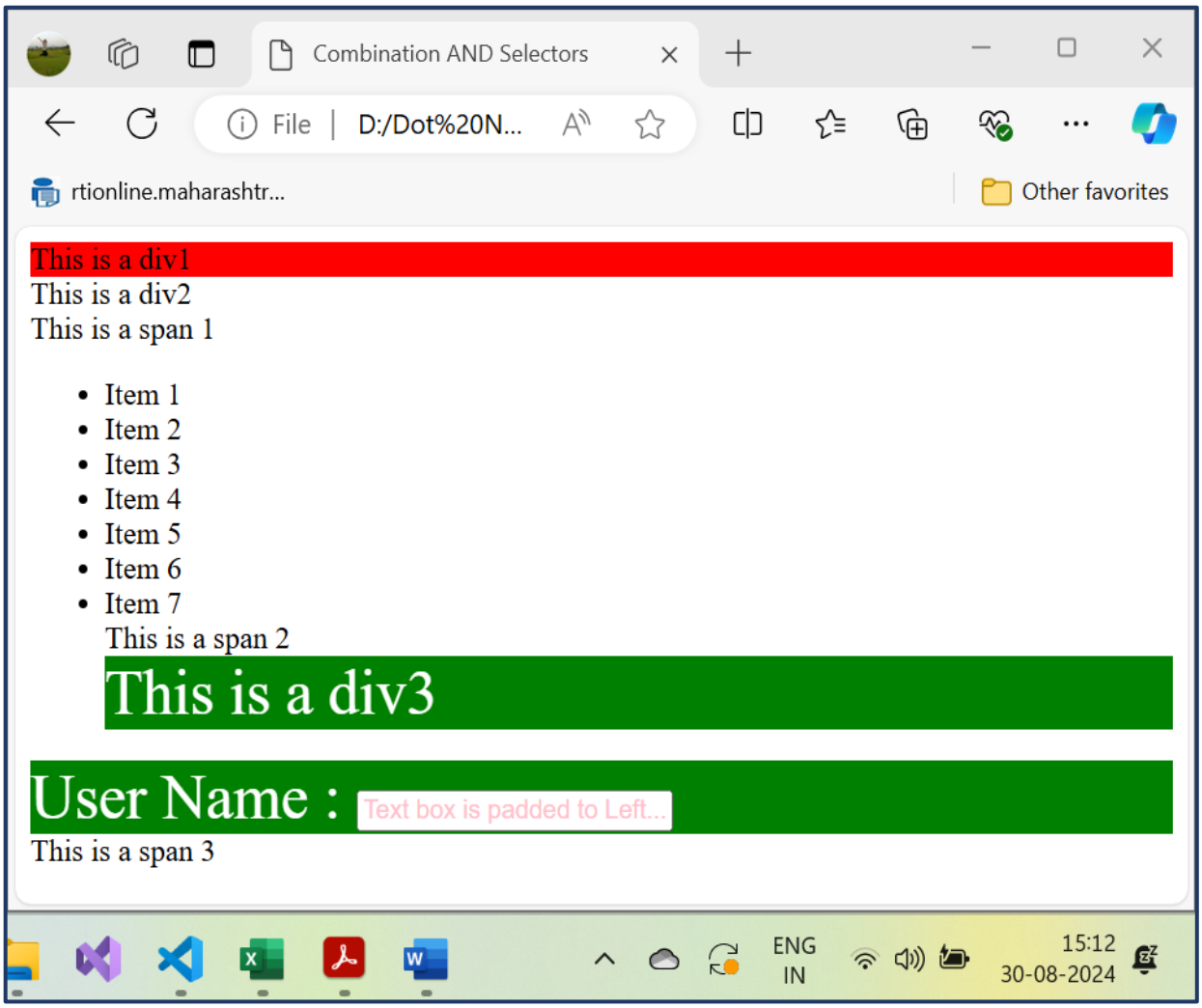
.a | div.c | .c | div

Example:

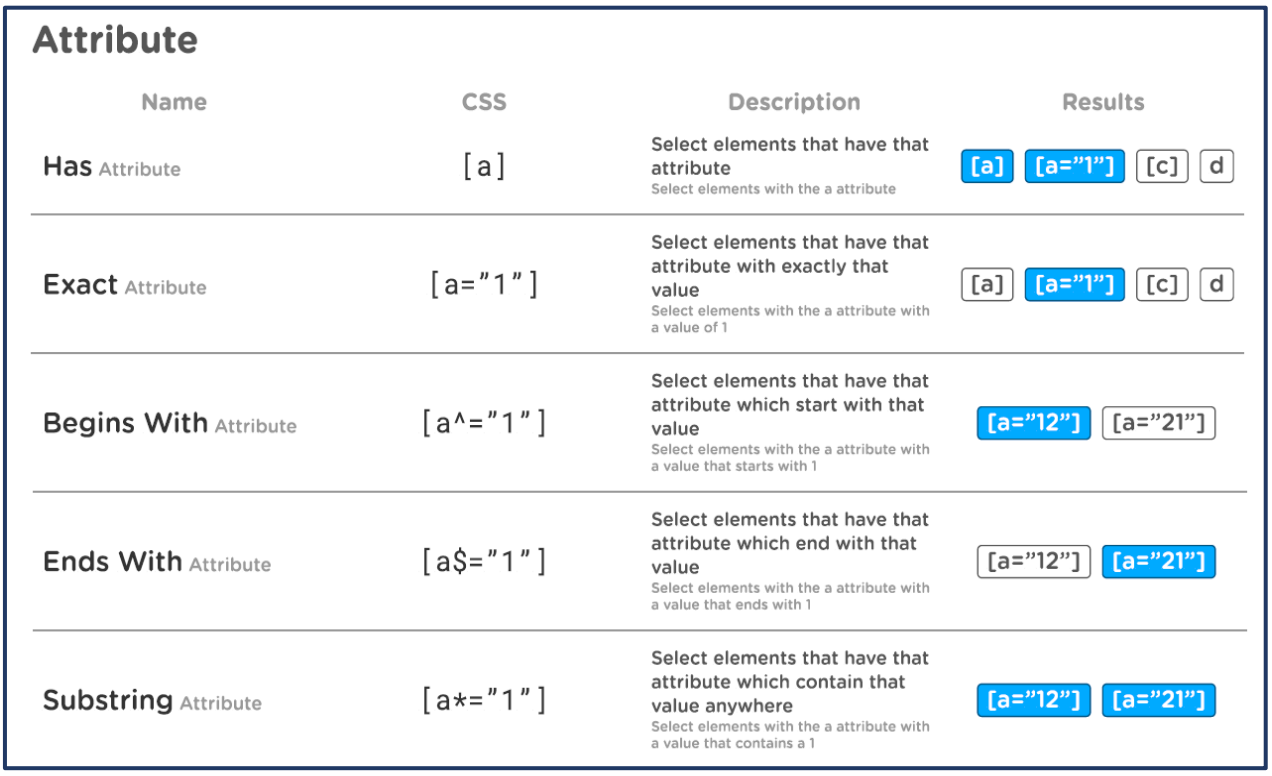
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Combination AND Selectors</title>
    <style>
/* And Selector */
div.red{
    background: red;
}
div.red.green-text, span.red.green-text{
    background: 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:


Combination AND Selectors
File | D:/Dot%20N...
rtionline.maharashtr...
Other favorites
This is a div1
This is a div2
This is a span 1
Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7
This is a span 2
This is a div3
User Name :
Text box is padded to Left...
This is a span 3
ENG
IN
15:12
30-08-2024

3) Attribute:



Attribute

Name	CSS	Description	Results
Has Attribute	[a]	Select elements that have that attribute Select elements with the a attribute	[a] [a="1"] [c] d
Exact Attribute	[a="1"]	Select elements that have that attribute with exactly that value Select elements with the a attribute with a value of 1	[a] [a="1"] [c] d
Begins With Attribute	[a^="1"]	Select elements that have that attribute which start with that value Select elements with the a attribute with a value that starts with 1	[a="12"] [a="21"]
Ends With Attribute	[a$="1"]	Select elements that have that attribute which end with that value Select elements with the a attribute with a value that ends with 1	[a="12"] [a="21"]
Substring Attribute	[a*="1"]	Select elements that have that attribute which contain that value anywhere Select elements with the a attribute with a value that contains a 1	[a="12"] [a="21"]

4) **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:

a. ::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.

b. ::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)

c. ::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.

d. ::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.
```

e. **::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.
```

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

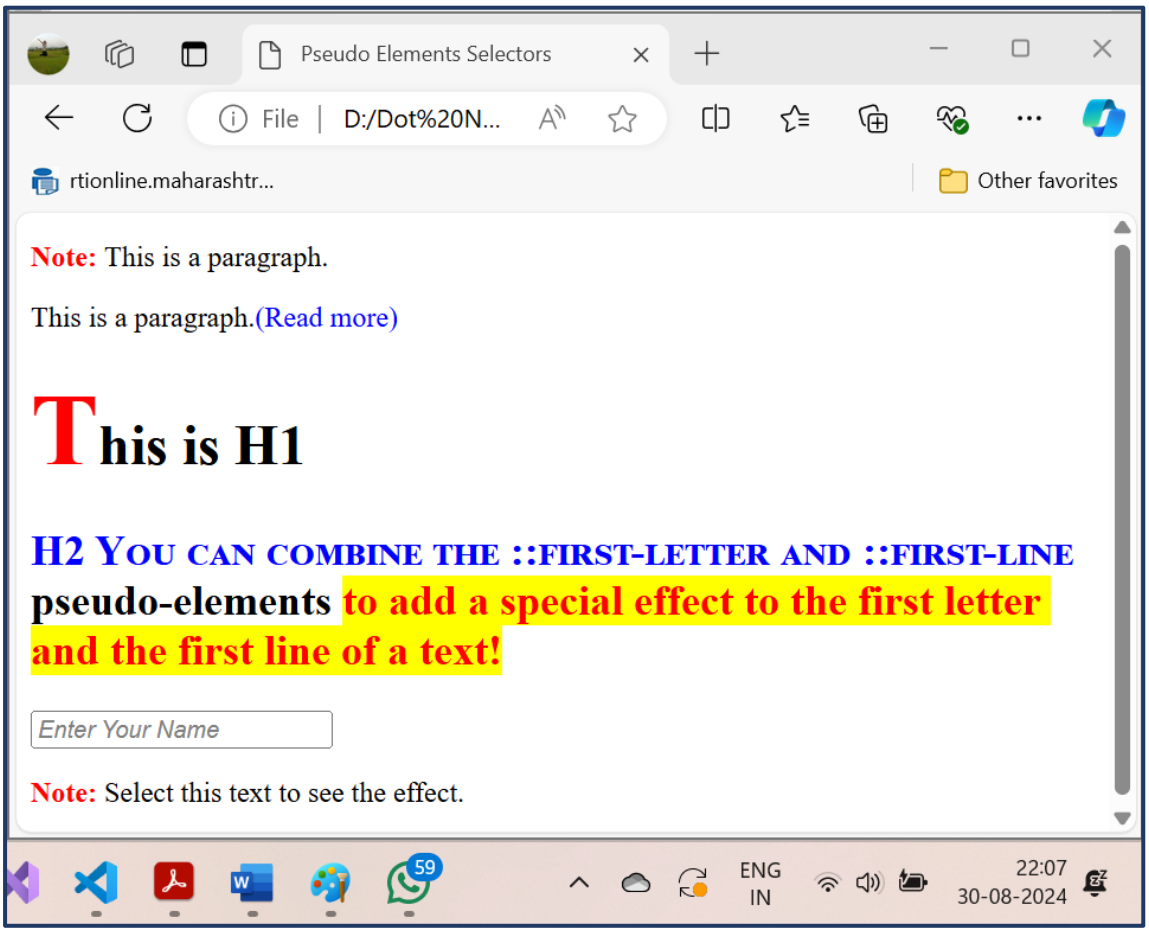
```
::selection {
    background-color: yellow;
    color: black;
}
<p>Select this text to see the effect.</p>

**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>
```



5) 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:

a. **:hover** : Select elements that are hovered by the mouse.
b. **:focus** : Select element that is focused. It is either by clicking or by tabbing to an element.
c. **:required** : Select the element with required attribute.
d. **:checked** : Select checkboxes / radio buttons that are with checked attribute.
e. **:disabled** : Select the inputs that are with disabled attribute.

- ✓ :active
- ✓ ::after/:after
- ✓ ::backdrop (experimental)
- ✓ ::before/:before
- ✓ :checked
- ✓ :default
- ✓ :dir (experimental)
- ✓ :disabled
- ✓ :empty

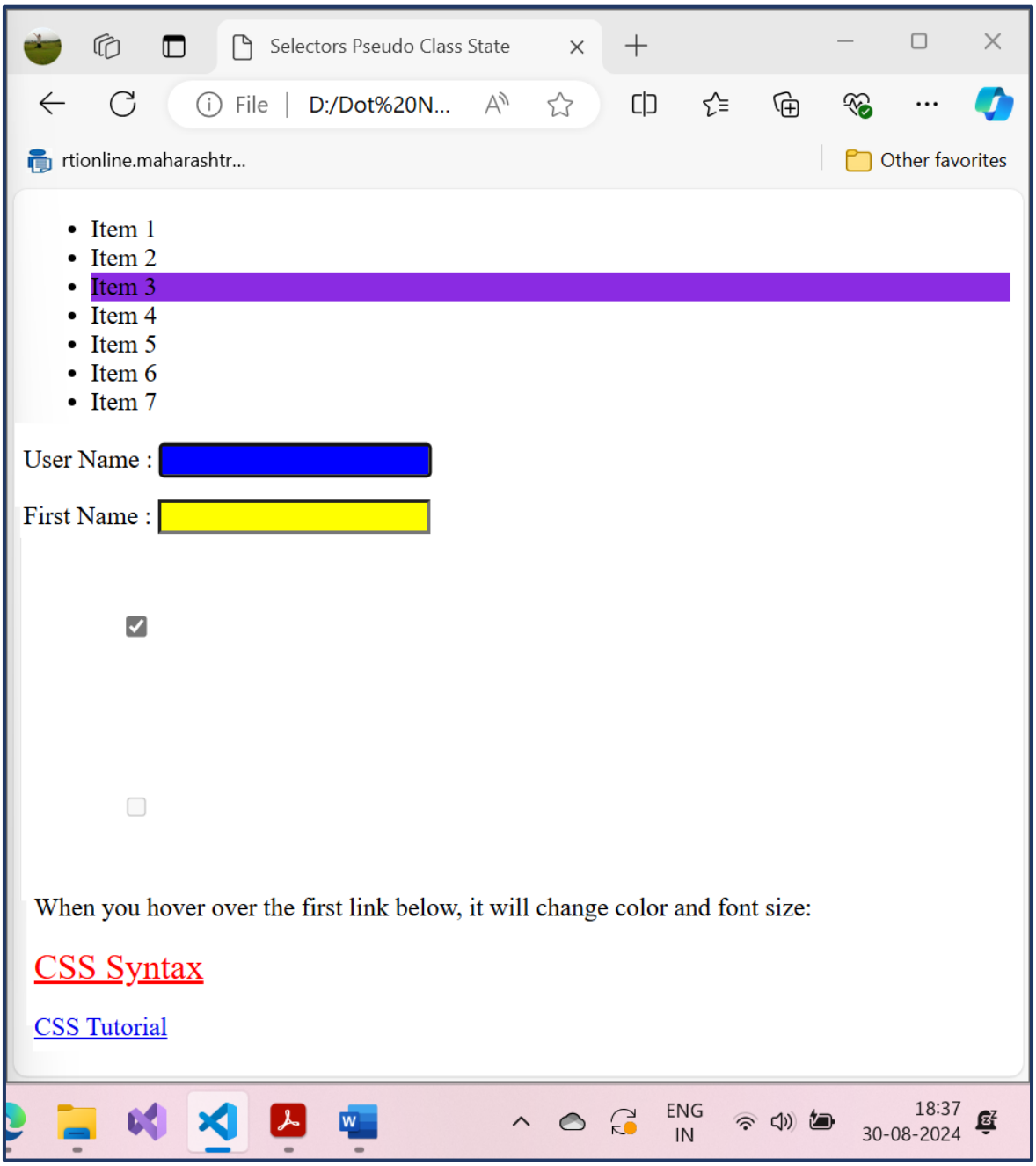
- ✓ :enabled
- ✓ :first-child
- ✓ ::first-letter/:first-letter
- ✓ ::first-line/:first-line
- ✓ :first-of-type
- ✓ :focus
- ✓ :fullscreen (experimental)
- ✓ :hover
- ✓ :in-range
- ✓ :indeterminate
- ✓ :invalid
- ✓ :lang
- ✓ :last-child
- ✓ :last-of-type
- ✓ :link
- ✓ :not
- ✓ :nth-child
- ✓ :nth-last-child
- ✓ :nth-last-of-type
- ✓ :nth-of-type
- ✓ :only-child
- ✓ :only-of-type
- ✓ :optional
- ✓ :out-of-range
- ✓ ::placeholder (experimental)
- ✓ :read-only
- ✓ :read-write
- ✓ :required
- ✓ :root
- ✓ ::selection
- ✓ :scope (experimental)
- ✓ :target
- ✓ :valid
- ✓ :visited
- ✓ 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:



6) Pseudo Class position / other:

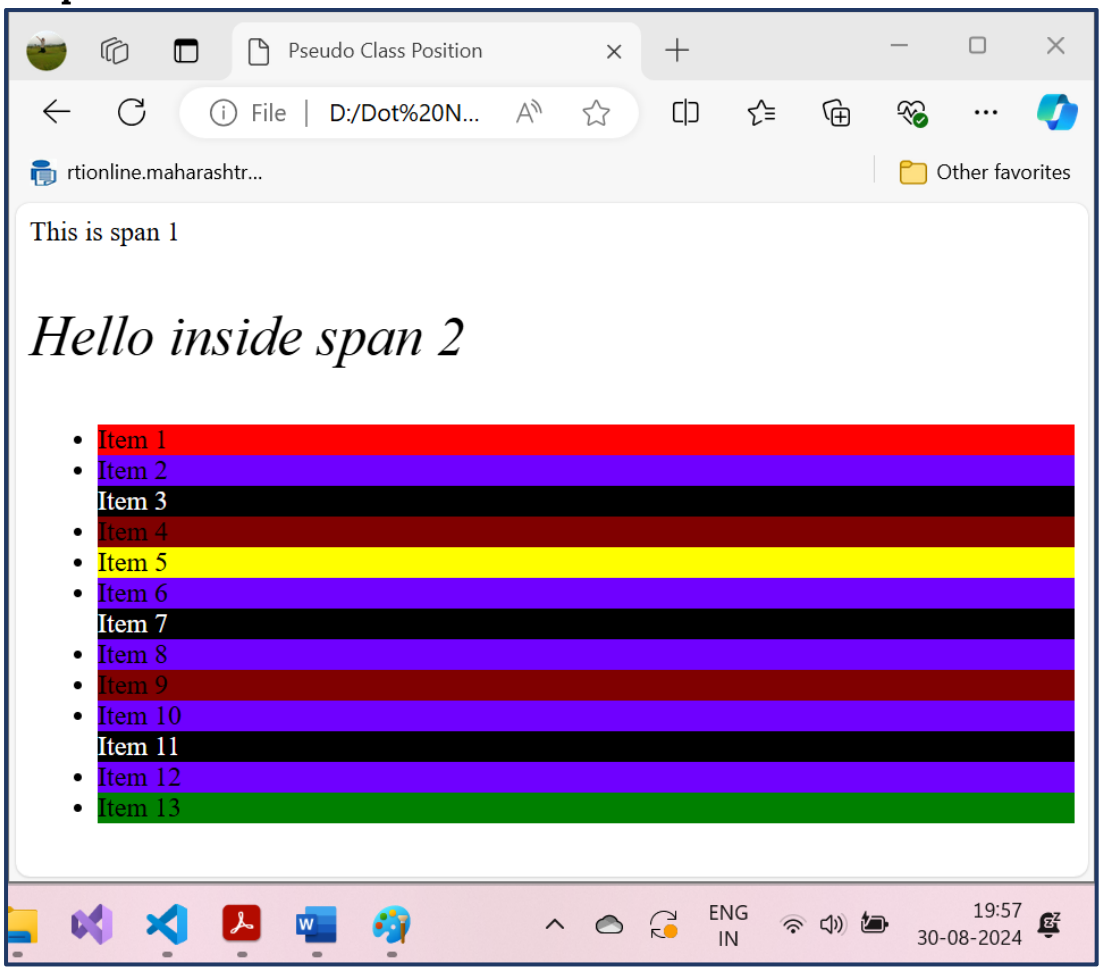
In CSS, pseudo-classes allow you to style elements based on their state, position in the document tree, user interaction, and other factors. They are written as :pseudo-class-name. Below is a breakdown of **positional pseudo-classes** and **other common pseudo-classes**.

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:

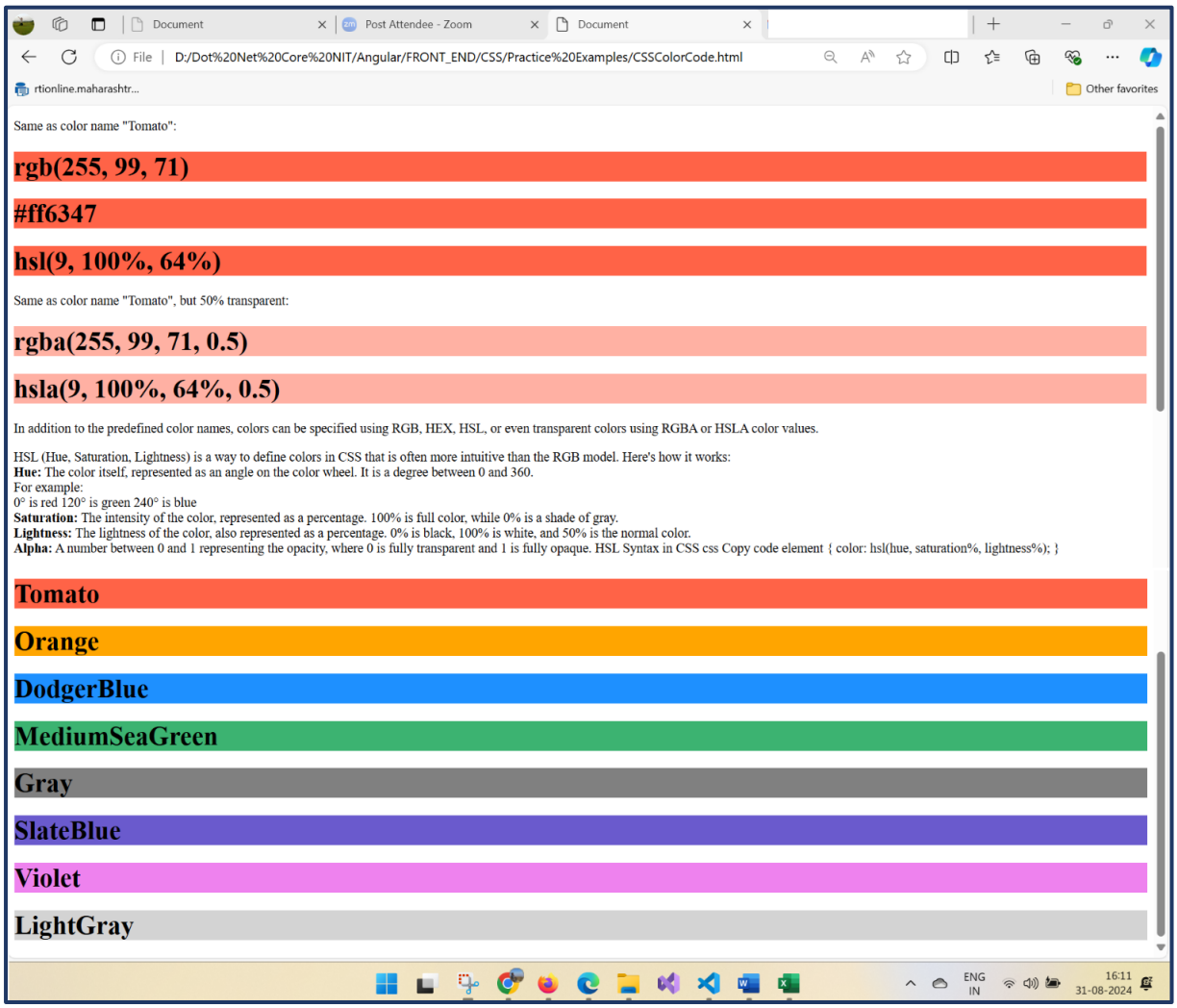
Each method has its own use case. **Hexadecimal and RGB/HSLA are great for precise control over color.** Named colors are convenient for simple, commonly used colors, while custom properties make it easier to maintain consistent color schemes across your styles.

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 is used to create space around the element, inside any defined border.
- **Border**: The border is used to cover the content & any padding, & also allows setting the style, color, and width of the border.
- **Margin**: Margin 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

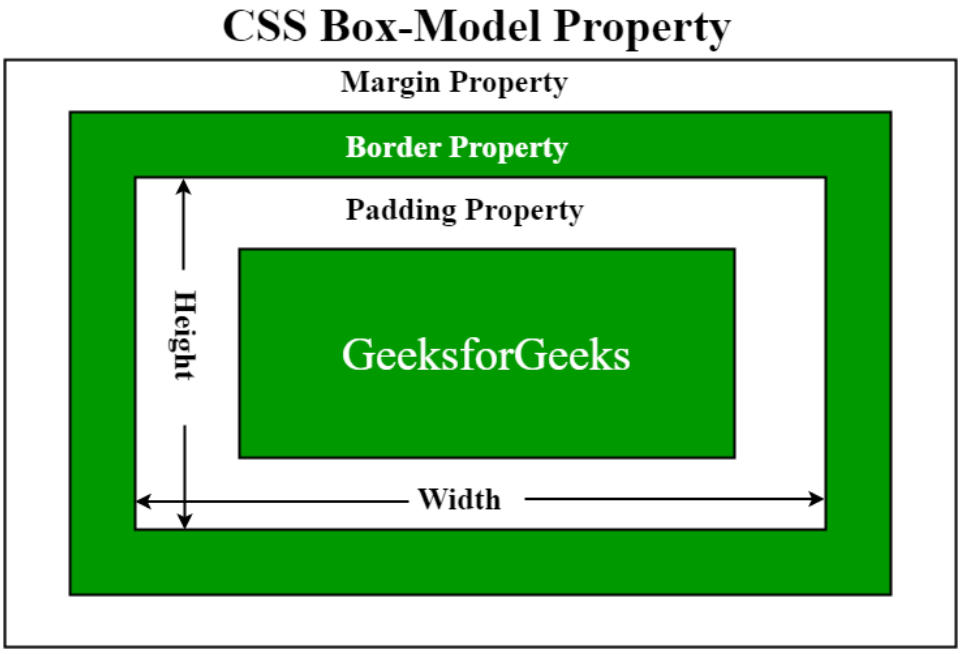
- Lies between the padding and margin.

- 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.

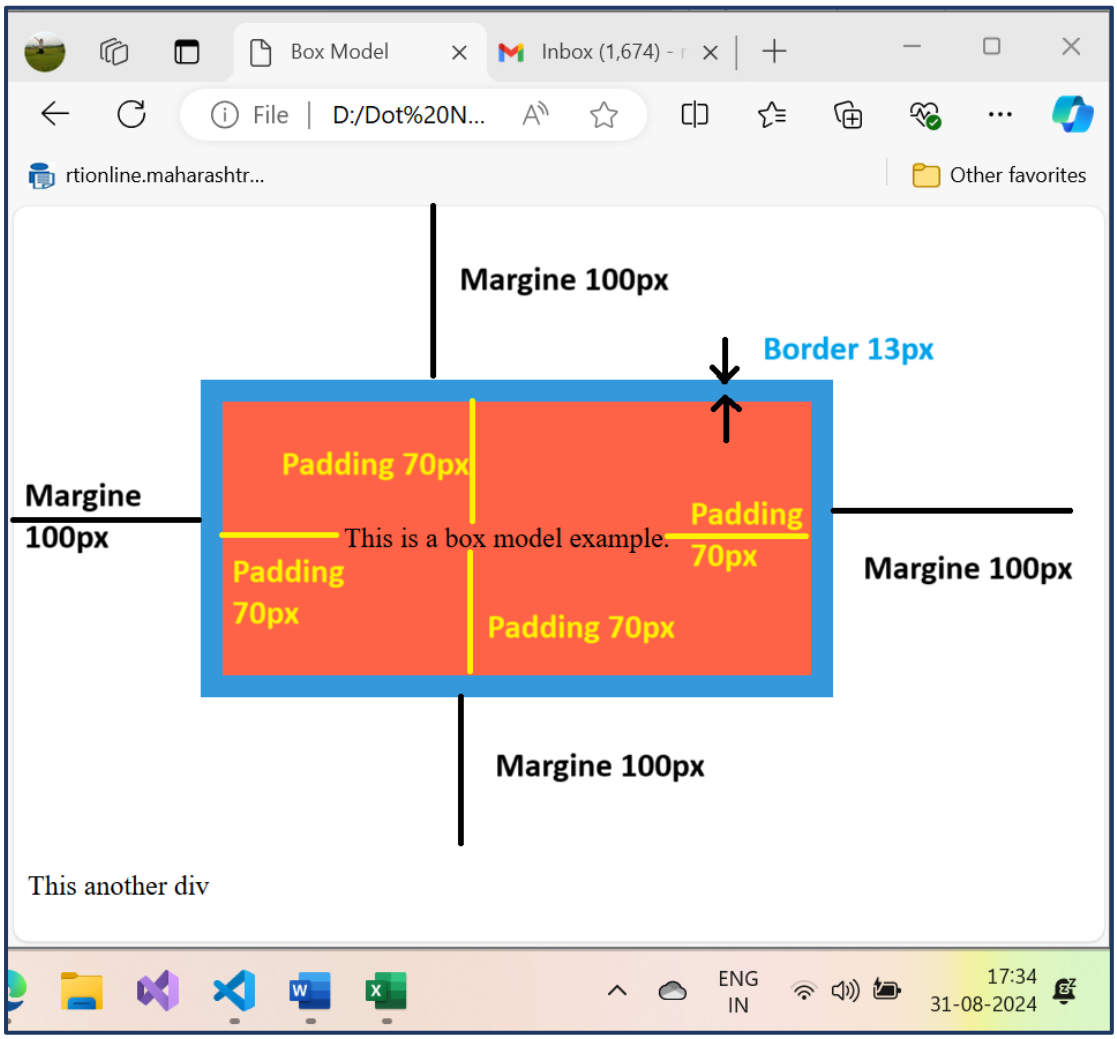
The following figure illustrates the **Box model** in CSS.



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.

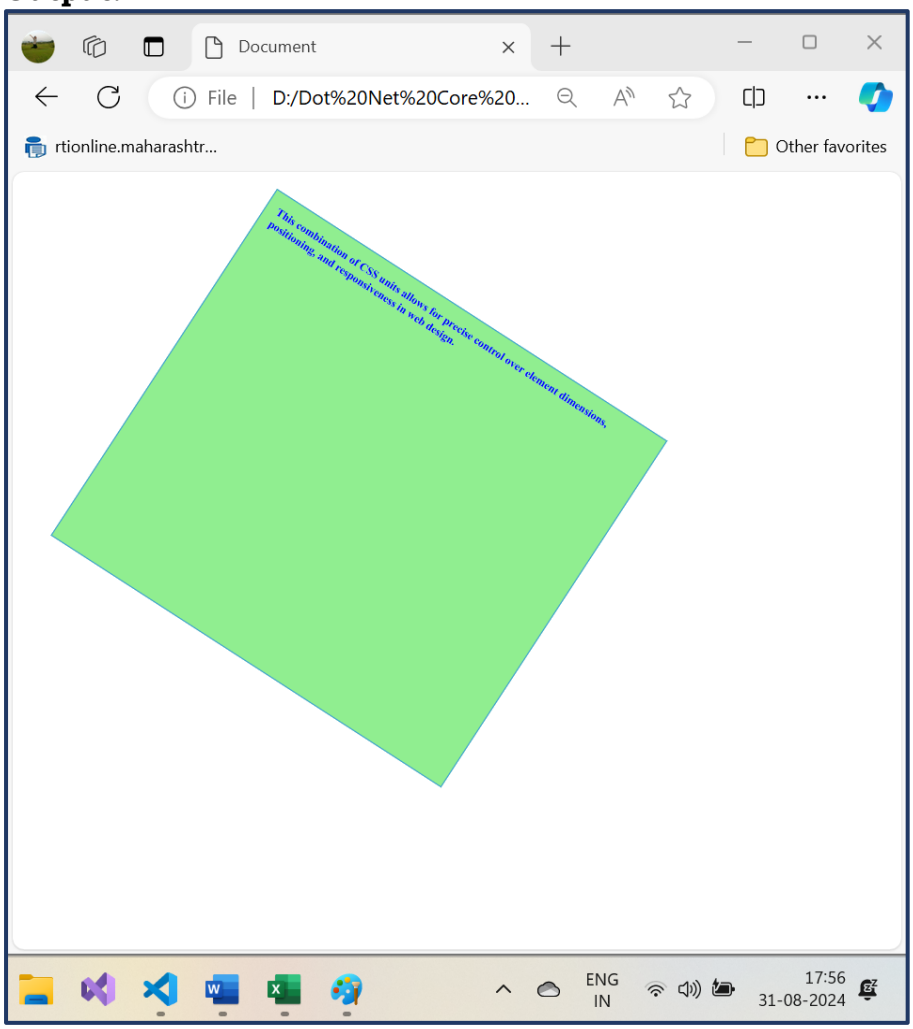
- **dpi (Dots Per Inch)**: Resolution in dots per inch. Example: min-resolution: 300dpi;
- **dpcm (Dots Per Centimeter)**: Resolution in dots per centimeter. Example: min-resolution: 118dpcm;
- **dppx (Dots Per Pixel)**: Resolution in dots per pixel (typically used for screens). Example: min-resolution: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

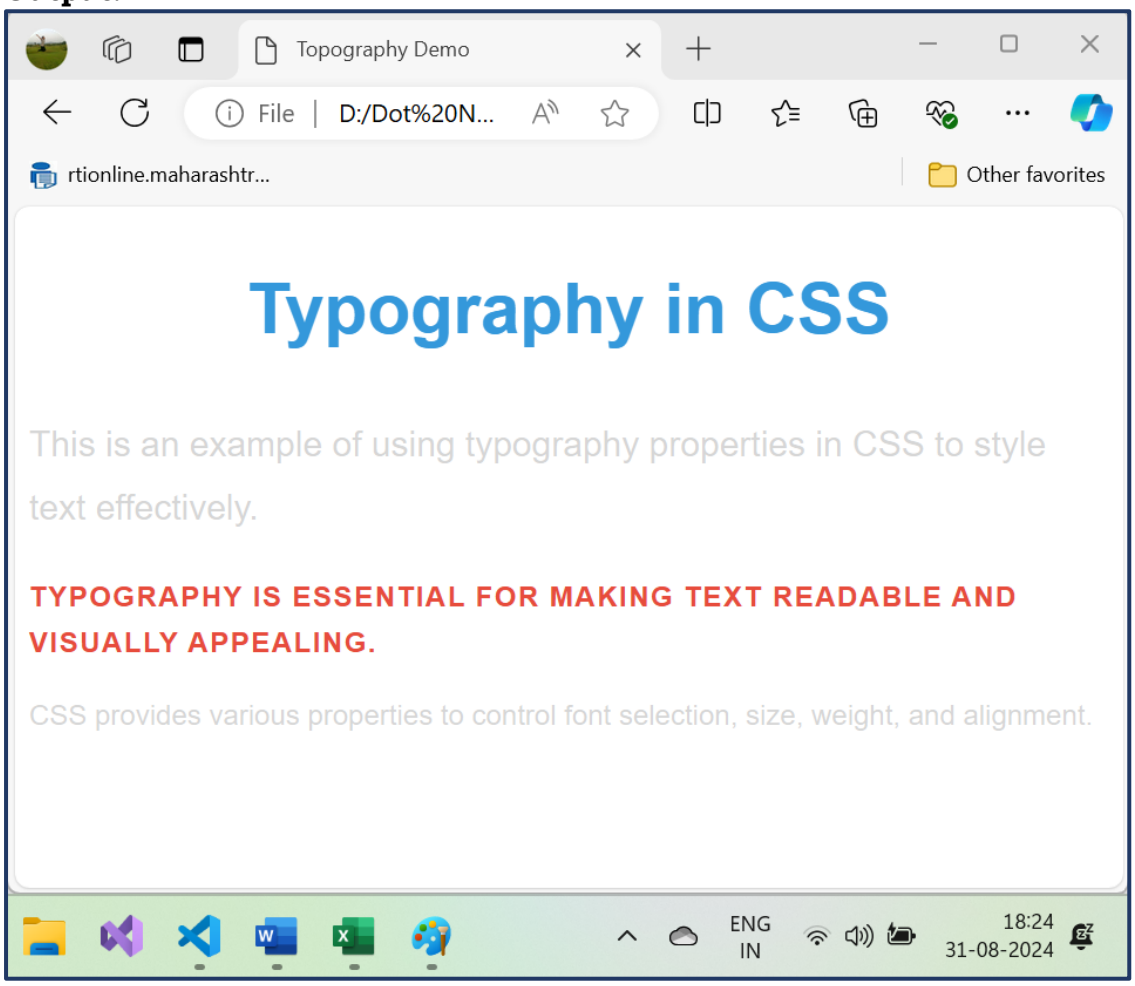
1. **font-family**: Specifies the font to be used for text.
2. **font-size**: Sets the size of the font.
3. **font-weight**: Defines the thickness of the font.
4. **font-style**: Specifies whether the font should be italic, oblique, or normal.
5. **line-height**: Sets the height of lines of text, impacting 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:

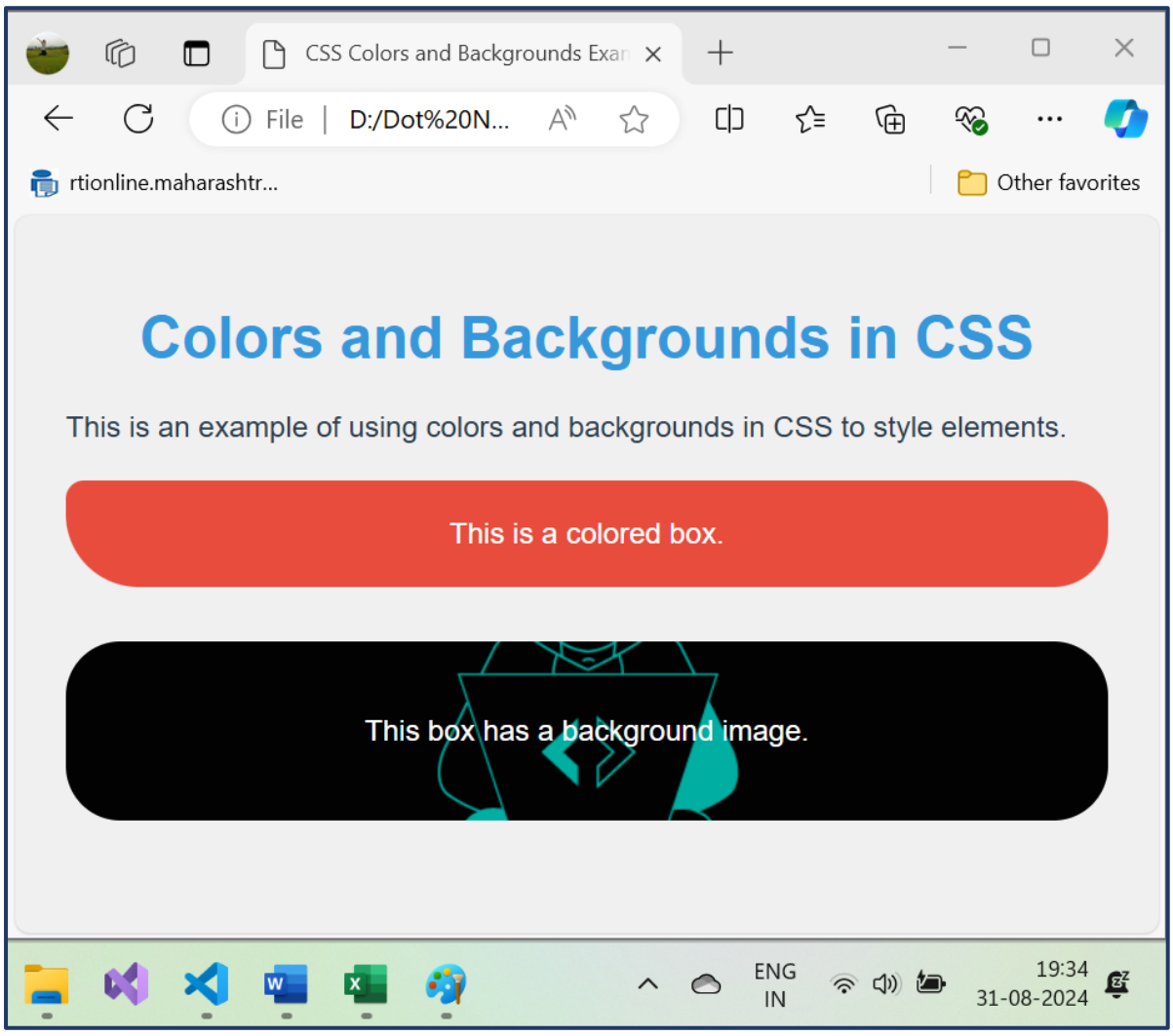
background: [background-color] [background-image] [background-repeat] [background-attachment] [background-position] / [background-size];

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:

```
.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:

<address>	<article>	<aside>	<blockquote>	<canvas>	<dd>
	<dl>	<dt>	<fieldset>	<figcaption>	<figure>
<footer>	<form>	<h1>-<h6>	<header>	<hr>	
<main>	<nav>	<noscript>		<p>	<pre>
<section>	<table>	<tfoot>		<video>	

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:

<a>	<abbr>	<acronym>	<b>	<bdo>	<big>
 	<button>	<cite>	<code>	<dfn>	
<i>	<img>	<input>	<kbd>	<label>	<map>
<object>	<output>	<q>	<samp>	<script>	<select>
<small>			<sub>	<sup>	<textarea>
<time>	<tt>	<var>			

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>
```

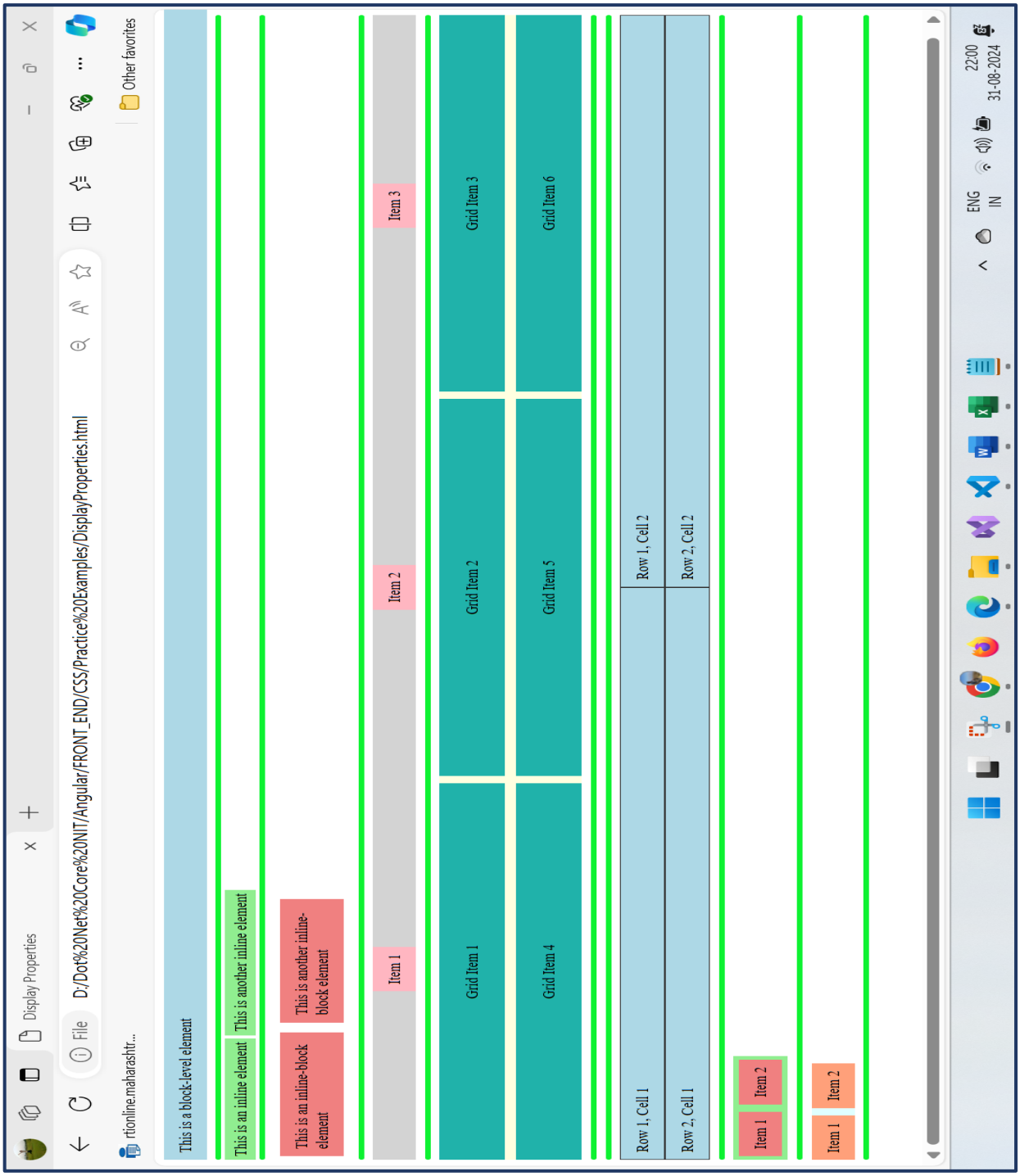
Example:

```
<!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:

```
/* 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>
```

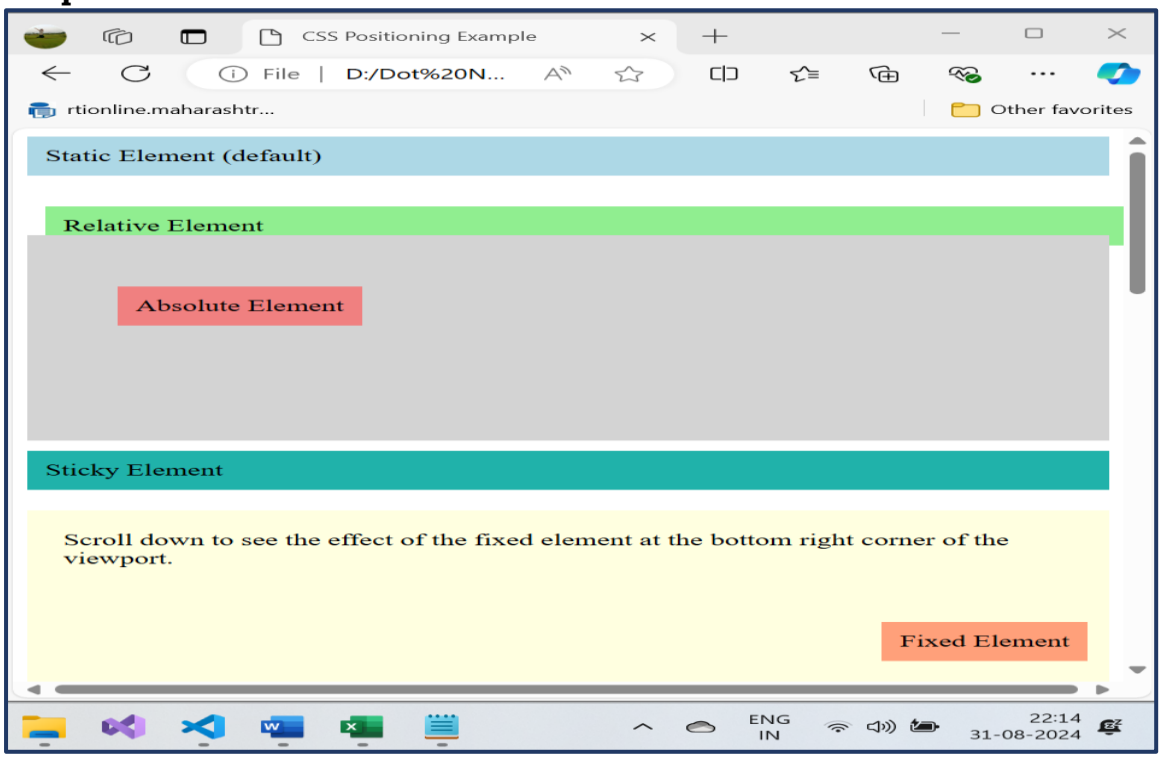
- Sticky elements are useful for creating fixed headers that scroll with the page until a certain point.

Example:

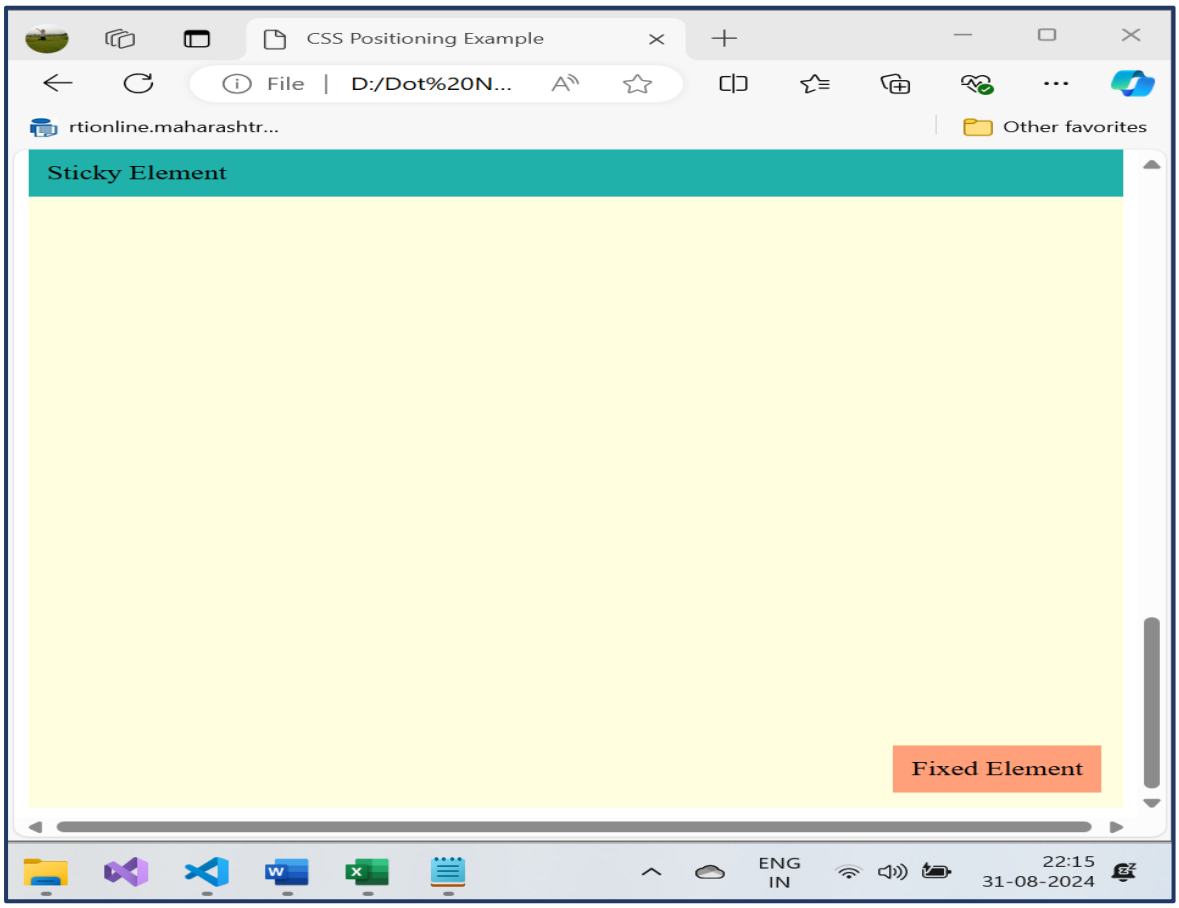
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CSS Positioning Example</title>
    <style>
        .static-element {
            position: static;
            background-color: lightblue;
            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:

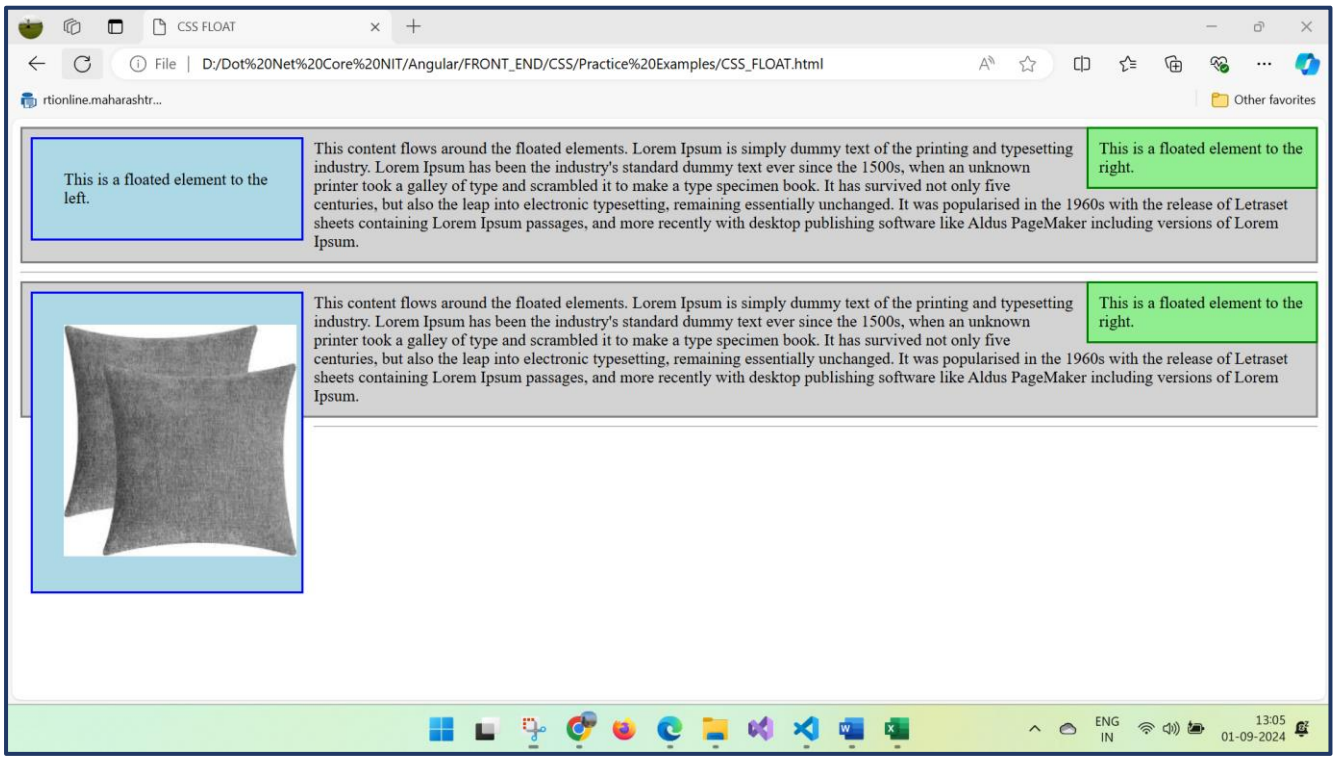
- left: The element floats to the left side of its container.
- right: The element floats to the right side of its container.
- none: The element does not float (this is the default).
- inherit: Inherits the float property from its parent.

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:

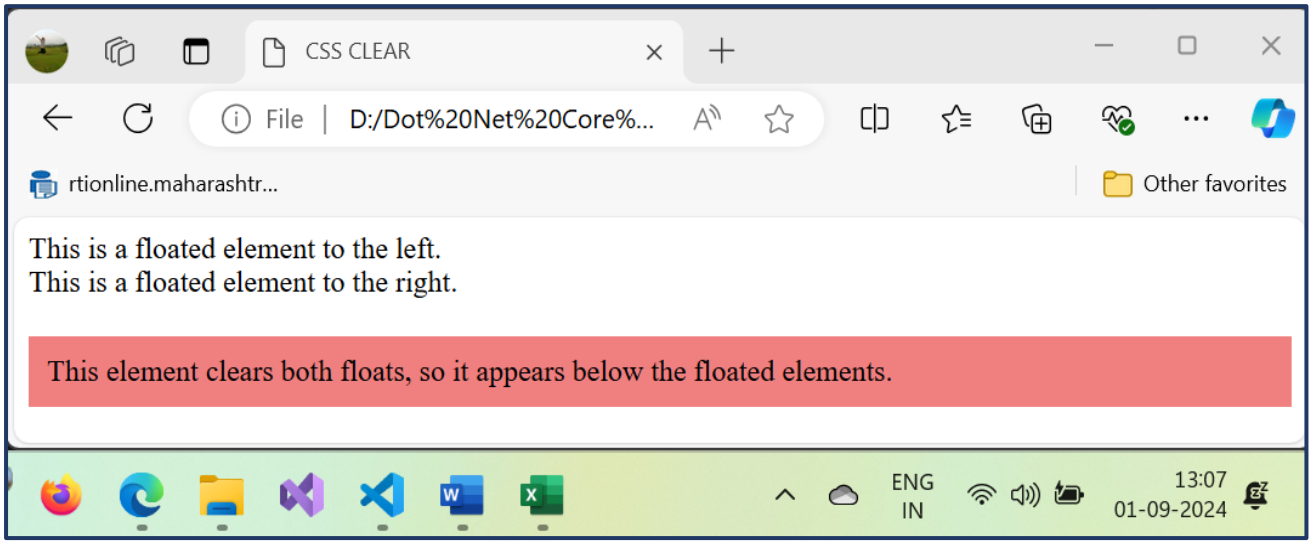
- left: The element is moved below left-floating elements.
- right: The element is moved below right-floating elements.
- both: The element is moved below both left and right floating elements.
- none: Default value; allows floating elements to affect the element.

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:

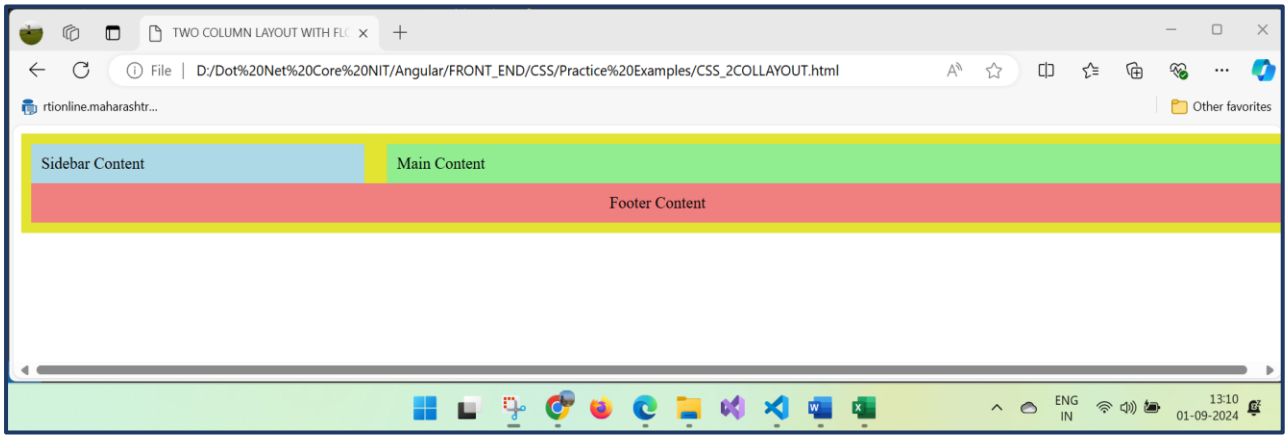
- .clear-both is used to ensure that this element appears below both the left and right floated elements.

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:

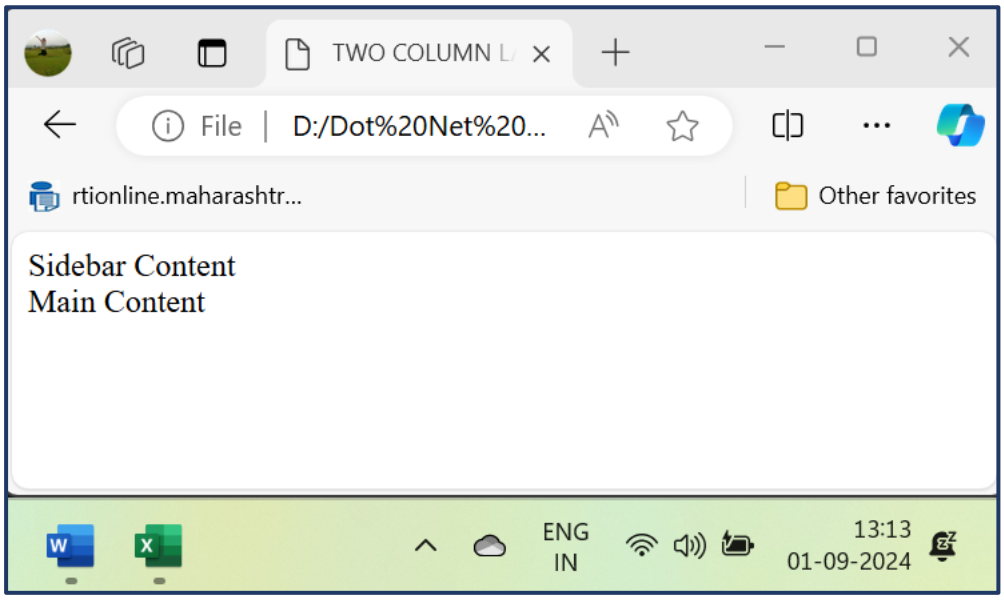
- **Overflow**: If the container only contains floated elements, it may collapse (height becomes zero) because floated elements are removed from the normal document flow.
- **Clearing Fix**: To prevent the collapse of containers with floated children, you can use the **clearfix** method.

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.

Margine and Outline:

In CSS, **borders** and **outlines** are used to style elements by creating a visual boundary around them. Though similar in appearance, they have distinct differences in behavior and how they affect the layout. Let's explore each with examples.

1. Borders in CSS

The **border** property defines the thickness, style, and color of the border surrounding an element. Borders are part of the element's box model, meaning they take up space and affect the element's size and layout.

Border Properties:

- **border-width**: Specifies the thickness of the border (e.g., 1px, 5px).
- **border-style**: Defines the style of the border (e.g., solid, dashed, dotted, double, none).

- **border-color**: Specifies the color of the border (e.g., #000, blue).

Example of Borders:

```
.border-example {
    border-width: 5px;
    border-style: solid;
    border-color: blue;
    padding: 20px;
    background-color: lightgray;
}

<div class="border-example">This element has a solid blue border.</div>
```

Explanation:

- The **border** is drawn inside the element's dimensions, surrounding the padding and content.
- Borders affect the element's layout, meaning they contribute to the element's total size.

Border Shorthand:

You can combine the border properties into a single line using shorthand.

```
.border-shorthand {
    border: 5px solid blue; /* Width, style, and color in one line */
    padding: 20px;
    background-color: lightgray;
}

<div class="border-shorthand">This element uses the border shorthand syntax.</div>
```

2. Outlines in CSS

The **outline** property is similar to the border, but it has key differences:

- **Outlines are drawn outside the border**, not inside the element's dimensions.
- **Outlines do not affect the element's size or layout**. They are purely decorative and do not occupy space in the document flow.
- **Outlines cannot be applied to individual sides** of an element (e.g., top, bottom).

Outline Properties:

- **outline-width**: Specifies the thickness of the outline (e.g., 2px, thin, thick).
- **outline-style**: Defines the style of the outline (e.g., solid, dashed, dotted).
- **outline-color**: Specifies the color of the outline (e.g., #000, red).
- **outline-offset**: Specifies the space between the outline and the element's border.

Example of Outline:

```
.outline-example {
    border: 2px solid black;
    outline-width: 5px;
    outline-style: dashed;
    outline-color: red;
    padding: 20px;
    background-color: lightyellow;
}

<div class="outline-example">This element has a dashed red outline outside its border.</div>
```

Explanation:

- The **outline** is drawn outside the border and does not affect the element's layout.

- Unlike borders, outlines do not take up space in the box model.

Outline Offset:

The outline-offset property controls the distance between the outline and the border of the element.

```
.outline-offset-example {
    border: 2px solid black;
    outline: 5px dashed red;
    outline-offset: 10px; /* Space between outline and border */
    padding: 20px;
    background-color: lightgreen;
}

<div class="outline-offset-example">This element has an outline with an offset.</div>
```

Explanation:

- The outline-offset creates a gap between the outline and the border, allowing for more visual separation.

Key Differences Between Border and Outline:

1. **Location**:
    - **Border**: Inside the element's box model, affecting the layout and occupying space.
    - **Outline**: Outside the border, not affecting the layout or taking up space.
2. **Box Model**:
    - **Border**: Part of the box model, contributing to the element's total size.
    - **Outline**: Not part of the box model, so it does not change the element's size.
3. **Individual Sides**:
    - **Border**: Can be applied to individual sides (top, right, bottom, left).
    - **Outline**: Applies to all sides uniformly.

Example Showing Both:

```
.both-example {
    border: 3px solid blue;
    outline: 5px dashed red;
    outline-offset: 10px;
    padding: 20px;
    background-color: lightblue;
}

<div class="both-example">This element has both a border and an outline.</div>
```

Explanation:

- The **border** is solid blue and inside the box model.
- The **outline** is dashed red, placed outside the border with a 10px offset, not affecting the element's layout.

Conclusion:

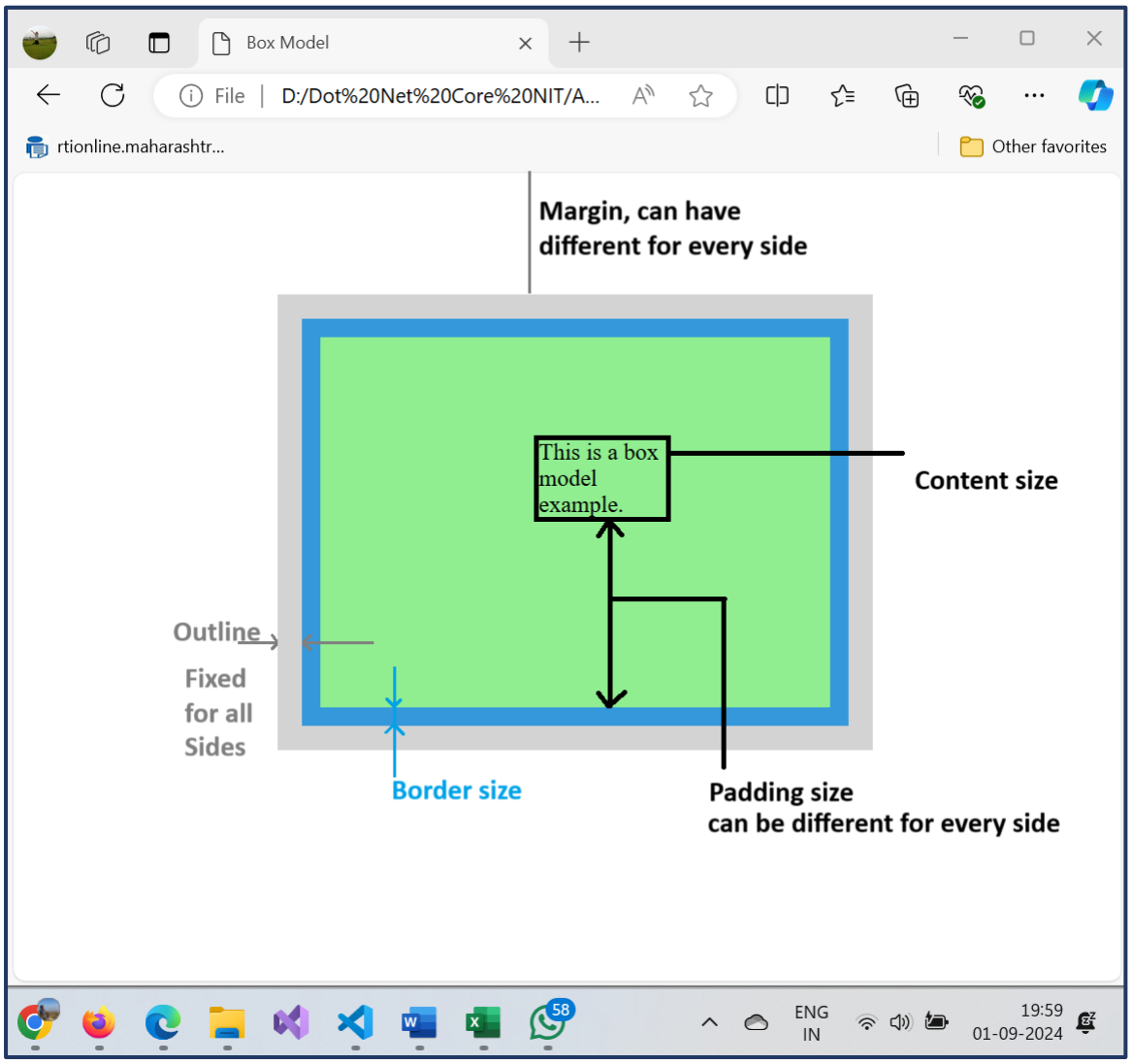
- Use **borders** when you need to affect the layout and style elements inside their box.
- Use **outlines** for visual emphasis or focus indicators without affecting the element's layout or size.
- **Borders** are more commonly used for consistent element spacing, while **outlines** are often used for highlighting elements (e.g., during keyboard focus).

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: 100px;                /* Content width */
        padding: 70px 100px 130px 150px; /* Space inside the box */
        border: 13px solid #3498db;  /* Border thickness and color */
        outline: 17px solid lightgray;
        margin: 100px 130px 160px 190px; /* Space outside the box */
        background-color: lightgreen;/*Background color for visibility*/
    }
    </style>
</head>
<body>
    <div class="box-model-example">
        This is a box model example.
    </div>
</body>
</html>
```

OUTPUT:



Z-Index:

The **z-index** property in CSS controls the vertical stacking order of elements that overlap each other. It determines which element appears on top of others when they overlap. The higher the z-index value, the closer the element is to the viewer (i.e., the higher it appears in the stack).

The z-index property only works on elements that have a position value other than static. This means you need to use position: relative, position: absolute, position: fixed, or position: sticky for z-index to take effect.

How z-index Works:

- **Higher z-index values**: Elements with higher z-index values appear on top of elements with lower z-index values.
- **Default value**: The default value of z-index is auto, which means the stacking order is determined by the order in the HTML document.
- **Negative values**: Elements with negative z-index values can be placed behind other elements.

Example of z-index in CSS

Let's create an example where three elements overlap each other, and we'll control their stacking order using z-index.

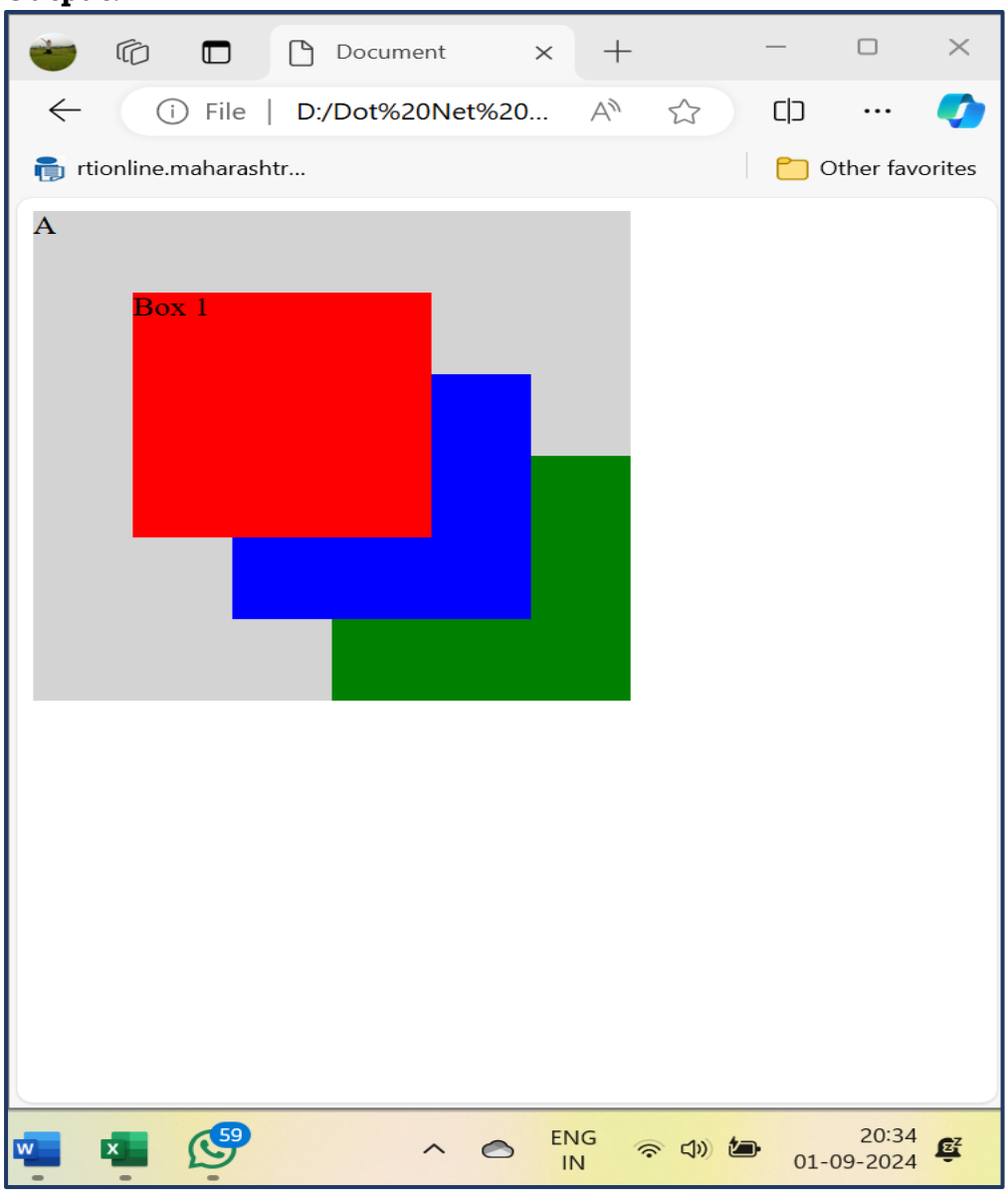
Example:

```
<!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>
.container {
    position: relative;
    width: 300px;
    height: 300px;
    background-color: lightgray;
}
.box1 {
    position: absolute;
    top: 50px;
    left: 50px;
    width: 150px;
    height: 150px;
    background-color: red;
    z-index: 2; /* This element will be on top */
}
.box2 {
    position: absolute;
    top: 100px;
    left: 100px;
    width: 150px;
    height: 150px;
    background-color: blue;
    z-index: 1; /* This element will be in the middle */
}
.box3 {
    position: absolute;
    top: 150px;
    left: 150px;
    width: 150px;
    height: 150px;
    background-color: green;
    z-index: 0; /* This element will be at the bottom */
}
.box4 {
    position: absolute;
    top: 150px;
    left: 150px;
    width: 150px;
    height: 150px;
    background-color: green;
    z-index: -1; /* This element will be behind everything */
}
    </style>
</head>
<body>
    <div class="container">A
        <div class="box1">Box 1</div>
```

```
        <div class="box2">Box 2</div>
        <div class="box3">Box 3</div>
    </div>
</body>
</html>
```

Output:



Explanation:

- **Box 1** (z-index: 2) is stacked on top because it has the highest z-index.
- **Box 2** (z-index: 1) is in the middle.
- **Box 3** (z-index: 0) is at the bottom of the stack.

Visualization:

- **Box 1 (red)** will appear on top of **Box 2 (blue)** and **Box 3 (green)**.
- **Box 2 (blue)** will appear on top of **Box 3 (green)** but below **Box 1 (red)**.
- **Box 3 (green)** will be at the bottom, covered by both **Box 1** and **Box 2**.

Important Notes:

1. **Stacking Context**: z-index works within the context of stacking. Certain elements, such as those with position: relative or position: absolute, create their own stacking context. The z-index values only apply within their stacking context. If a parent

element has a stacking context, the z-index of child elements is only relative to each other, not to elements outside the parent.

2. **Auto Value**: The default value for z-index is auto, meaning the element's order in the DOM determines its stacking order when no z-index is applied.

Example with Negative z-index:

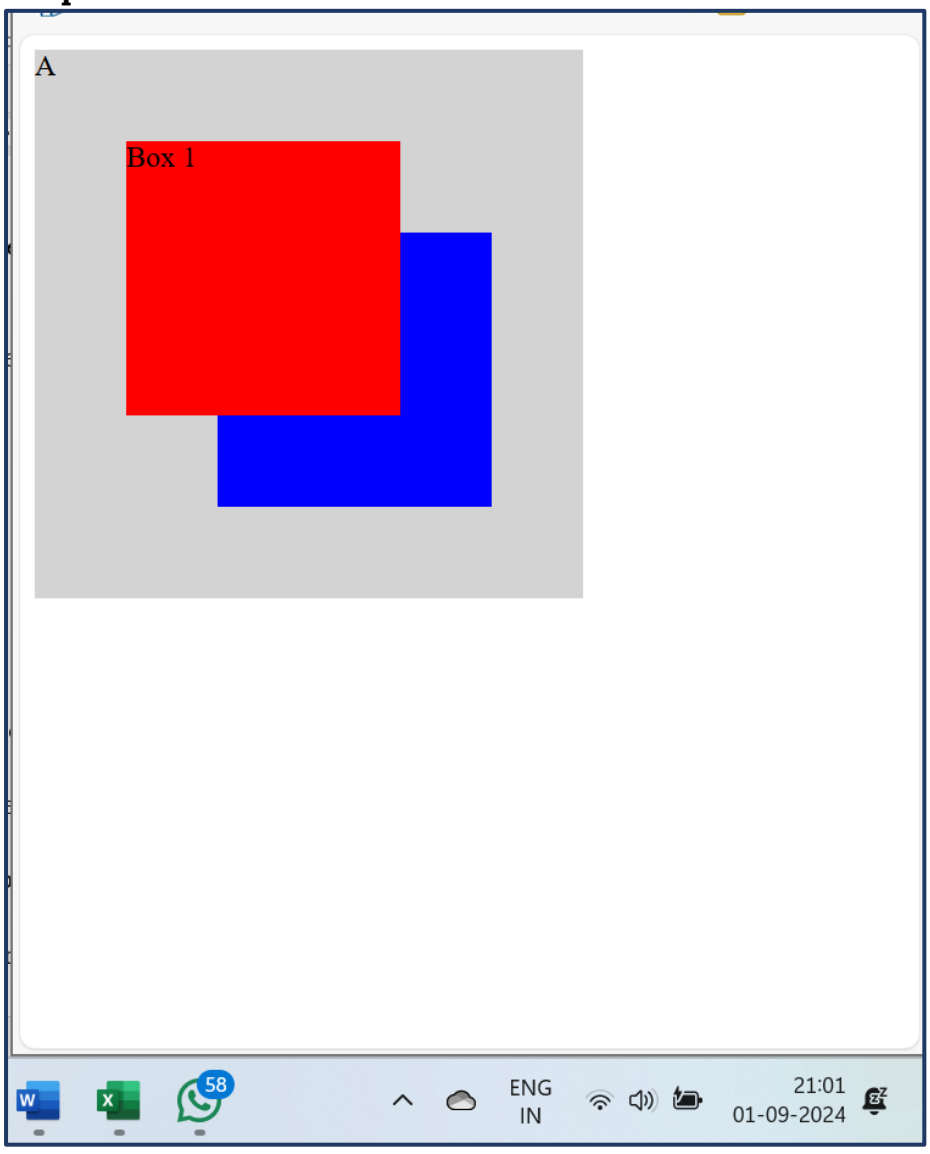
You can use a negative z-index value to push an element behind others.

```
.box3 {
    position: absolute;
    top: 150px;
    left: 150px;
    width: 150px;
    height: 150px;
    background-color: green;
    z-index: -1; /* This element will be behind everything */
}
```

In this case, **Box 3** will appear behind **Box 1** and **Box 2** due to the negative z-index.

Note : In above code Box 4 instead of Box 3

Output:



Conclusion:

- The z-index property is useful for controlling the stacking order of overlapping elements.
- It only works on positioned elements (those with position set to relative, absolute, fixed, or sticky).
- Higher z-index values bring elements to the front, and lower or negative values push elements behind others.

By understanding and using z-index, you can control the layering of elements in complex layouts where elements overlap.

Media Queries:

Media Queries in CSS are a powerful tool used to create responsive designs. They allow you to apply different styles depending on the device or screen size, making your website adaptable to various screen resolutions and device orientations.

What are Media Queries?

Media queries consist of a media type (e.g., screen, print) and one or more conditions that query certain characteristics of the user's device (e.g., screen width, height, orientation). Based on these conditions, different CSS rules can be applied.

Syntax of Media Queries

```
@media (condition) {
  /* CSS rules */
}
```

You can combine multiple conditions using logical operators like and, or, and not.

Example of a Simple Media Query:

```
/* Default styles for all devices */
body {
  background-color: white;
  font-size: 16px;
}

/* Styles for screens wider than 768px (e.g., tablets and desktops) */
@media (min-width: 768px) {
  body {
    background-color: lightblue;
    font-size: 18px;
  }
}

/* Styles for screens wider than 1024px (e.g., desktops and large tablets) */
@media (min-width: 1024px) {
  body {
    background-color: lightgreen;
    font-size: 20px;
  }
}
```

Explanation:

- **Default styles** are applied to all devices.
- **@media (min-width: 768px)** applies styles to devices with a screen width of 768 pixels or wider (e.g., tablets).
- **@media (min-width: 1024px)** applies styles to devices with a screen width of 1024 pixels or wider (e.g., desktops).

This approach enables you to create a responsive design that adjusts based on the screen size.

Breakpoints in Responsive Design

Breakpoints are the specific screen widths where your design adjusts to different layouts. These are typically based on common device screen sizes. You can define custom breakpoints based on the needs of your design, but some commonly used breakpoints are:

1. **Mobile Devices**: 0 - 767px
2. **Tablets**: 768px - 1023px
3. **Small Desktops**: 1024px - 1279px
4. **Large Desktops**: 1280px and above

Example of Media Queries with Common Breakpoints:

```
/* Base styles for mobile devices (0 - 767px) */
body {
  background-color: white;
  font-size: 16px;
}

/* Tablet styles (768px - 1023px) */
@media (min-width: 768px) and (max-width: 1023px) {
  body {
    background-color: lightblue;
    font-size: 18px;
  }
}

/* Small desktop styles (1024px - 1279px) */
@media (min-width: 1024px) and (max-width: 1279px) {
  body {
    background-color: lightgreen;
    font-size: 20px;
  }
}

/* Large desktop styles (1280px and above) */
@media (min-width: 1280px) {
  body {
    background-color: lightcoral;
    font-size: 22px;
  }
}
```

Explanation:

- **Mobile Devices (0 - 767px)**: These styles are the default and apply to all devices unless overridden by larger breakpoints.
- **Tablets (768px - 1023px)**: These styles apply to tablets, adjusting the background color and font size.
- **Small Desktops (1024px - 1279px)**: For small desktop screens, the styles adjust accordingly.
- **Large Desktops (1280px and above)**: For larger screens, the design changes to accommodate the larger viewport.

Responsive Images Example:

Images can also be made responsive using media queries.

```
/* Default image size for mobile */
img {
  width: 100%; /* The image takes up the full width of the screen */
  height: auto;
}

/* For larger screens, reduce the image size */
@media (min-width: 768px) {
  img {
    width: 80%;
  }
}

@media (min-width: 1024px) {
  img {
    width: 60%;
  }
}
```

Orientation Media Queries:
You can also target devices based on their orientation (landscape or portrait).

```
/* Styles for devices in landscape mode */
@media (orientation: landscape) {
  body {
    background-color: lightyellow;
  }
}

/* Styles for devices in portrait mode */
@media (orientation: portrait) {
  body {
    background-color: lightpink;
  }
}
```

Best Practices for Responsive Design:

1. **Mobile-First Approach**: Start designing for the smallest screen (mobile) and gradually add styles for larger screens using media queries.
2. **Use Relative Units**: Use em, rem, %, and vh/vw instead of fixed pixel values to make your design more fluid.
3. **Breakpoints Based on Content**: Set breakpoints based on your content's layout needs rather than specific device sizes.

Conclusion:
Media queries are essential for creating responsive designs that work well on various screen sizes and devices. By strategically setting breakpoints and adjusting styles, you can ensure a great user experience across different devices, from mobile phones to large desktop monitors.

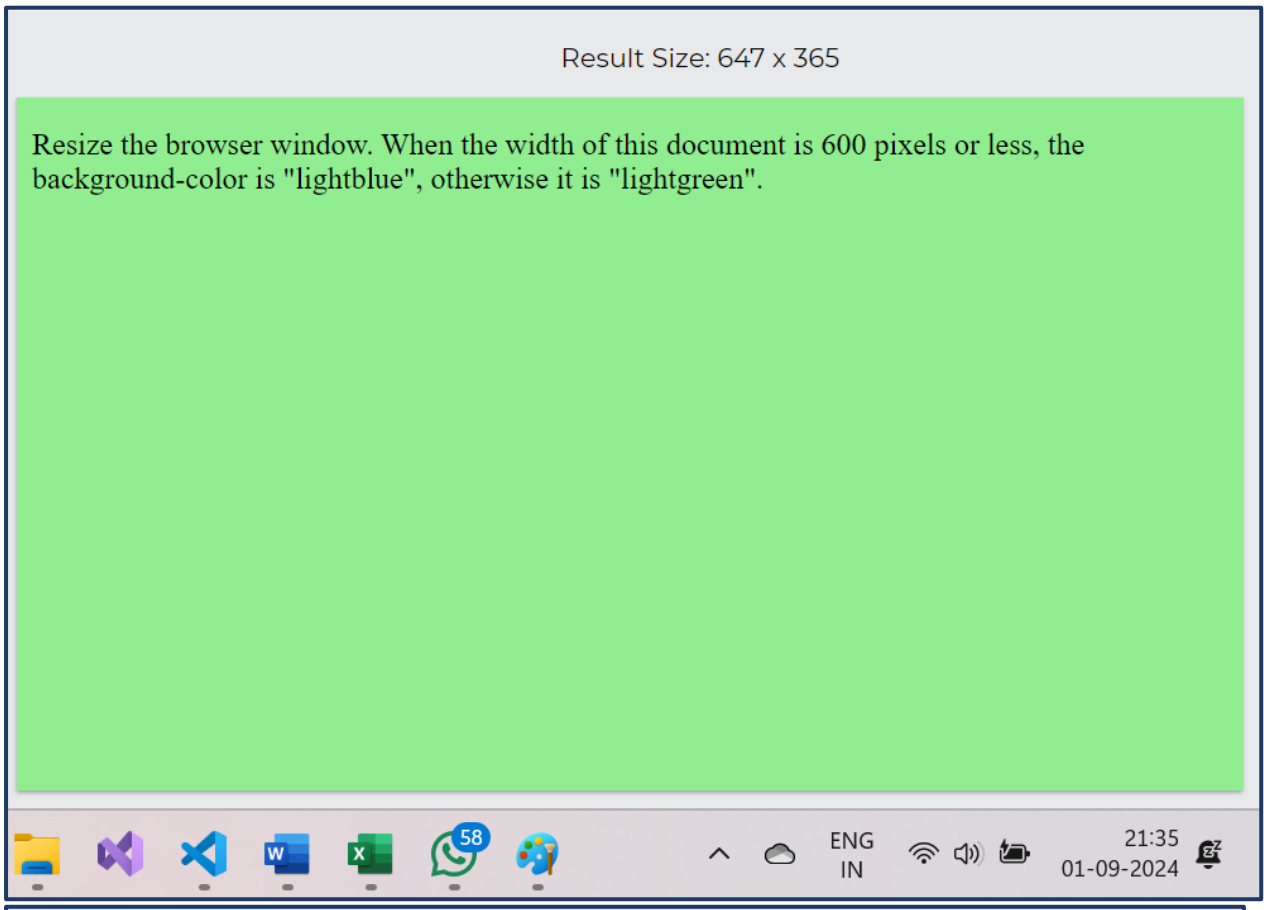
Responsive Web Design - Media Queries:
What is a Media Query?
Media query is a CSS technique introduced in CSS3.

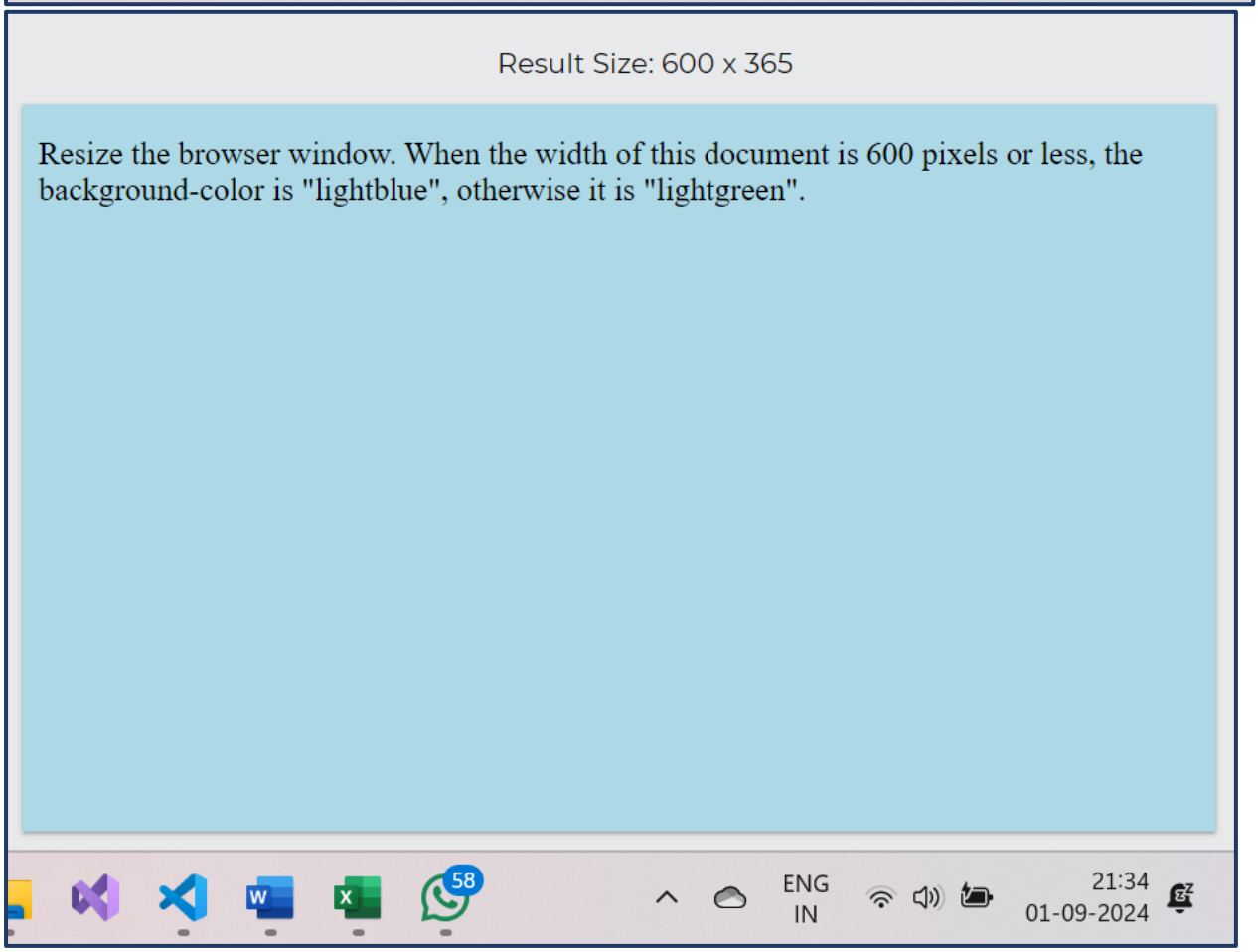
It uses the @media rule to include a block of CSS properties only if a certain condition is true.

Example 1:

If the browser window is 600px or smaller, the background color will be lightblue:

```
@media only screen and (max-width: 600px) {
  body {
    background-color: lightblue;
  }
}
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
body {
  background-color: lightgreen;
}

@media only screen and (max-width: 600px) {
  body {
    background-color: lightblue;
  }
}
</style>
</head>
<body>

<p>Resize the browser window. When the width of this document is 600 pixels or less, the background-color is "lightblue", otherwise it is "lightgreen".</p>

</body>
</html>
```


Result Size: 647 x 365
Resize the browser window. When the width of this document is 600 pixels or less, the background-color is "lightblue", otherwise it is "lightgreen".
58
ENG
IN
21:35
01-09-2024


Result Size: 600 x 365
Resize the browser window. When the width of this document is 600 pixels or less, the background-color is "lightblue", otherwise it is "lightgreen".
58
ENG
IN
21:34
01-09-2024

Example 2:

Add a Breakpoint

Earlier in this tutorial we made a web page with rows and columns, and it was responsive, but it did not look good on a small screen.

Media queries can help with that. We can add a breakpoint where certain parts of the design will behave differently on each side of the breakpoint.

Desktop

Phone

Use a media query to add a breakpoint at 768px:

Example

When the screen (browser window) gets smaller than 768px, each column should have a width of 100%:

```
/* For desktop: */
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}

@media only screen and (max-width: 768px) {
  /* For mobile phones: */
  [class*="col-"] {
    width: 100%;
  }
}
```

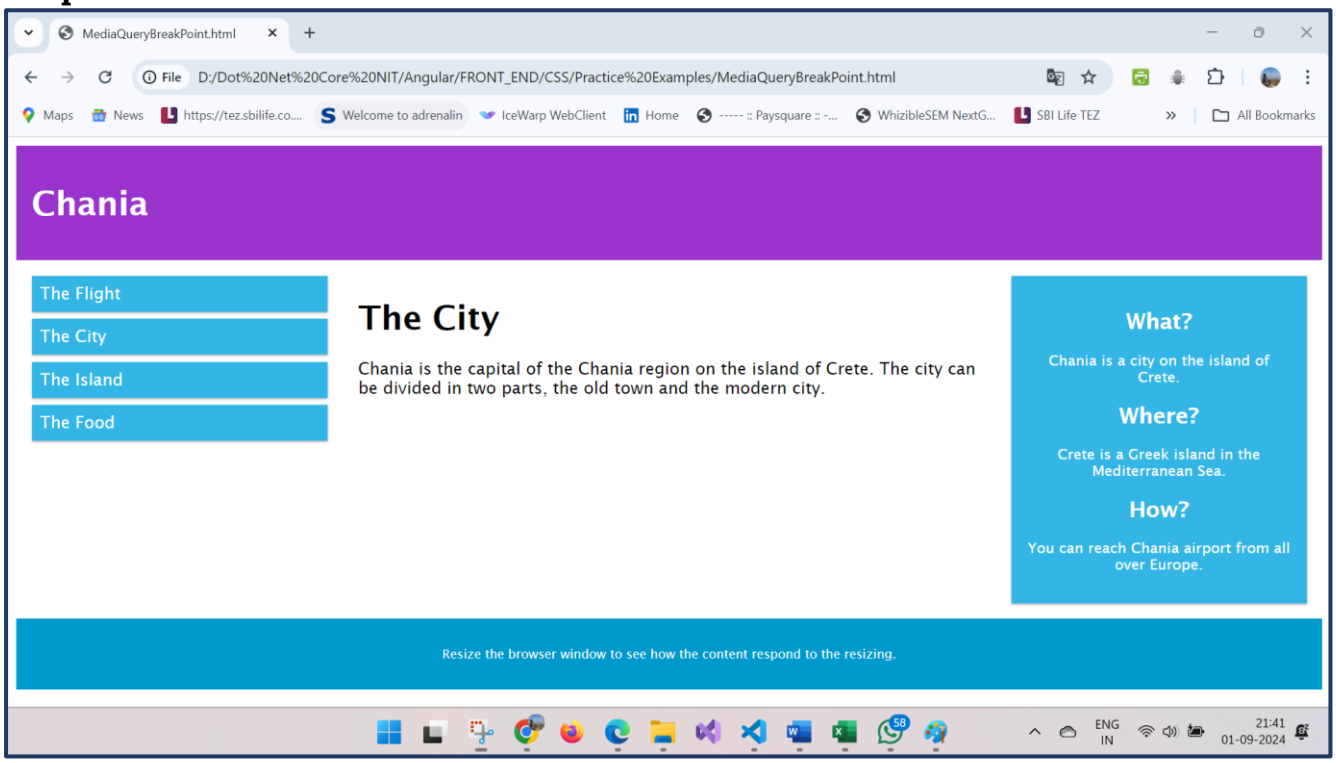
Complete Code:

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
* {
  box-sizing: border-box;
}

.row::after {
  content: "";
  clear: both;
  display: block;
}

[class*="col-"] {
  float: left;
  padding: 15px;
}

html {
  font-family: "Lucida Sans", sans-serif;
}

.header {
  background-color: #9933cc;
  color: #ffffff;
  padding: 15px;
}

.menu ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}

.menu li {
  padding: 8px;
  margin-bottom: 7px;
  background-color: #33b5e5;
  color: #ffffff;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}

.menu li:hover {
  background-color: #0099cc;
}

.aside {
  background-color: #33b5e5;
```

```
  padding: 15px;
  color: #ffffff;
  text-align: center;
  font-size: 14px;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}

.footer {
  background-color: #0099cc;
  color: #ffffff;
  text-align: center;
  font-size: 12px;
  padding: 15px;
}

/* For desktop: */
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}

@media only screen and (max-width: 768px) {
  /* For mobile phones: */
  [class*="col-"] {
    width: 100%;
  }
}
</style>
</head>
<body>

<div class="header">
  <h1>Chania</h1>
</div>

<div class="row">
  <div class="col-3 menu">
    <ul>
    <li>The Flight</li>
    <li>The City</li>
    <li>The Island</li>
    <li>The Food</li>
    </ul>
```

```
  </div>

  <div class="col-6">
    <h1>The City</h1>
    <p>Chania is the capital of the Chania region on the island of Crete. The city
can be divided in two parts, the old town and the modern city.</p>
  </div>

  <div class="col-3 right">
    <div class="aside">
      <h2>What?</h2>
      <p>Chania is a city on the island of Crete.</p>
      <h2>Where?</h2>
      <p>Crete is a Greek island in the Mediterranean Sea.</p>
      <h2>How?</h2>
      <p>You can reach Chania airport from all over Europe.</p>
    </div>
  </div>
</div>

<div class="footer">
  <p>Resize the browser window to see how the content respond to the resizing.</p>
</div>

</body>
</html>
```

Output 1:



Output 2:

Always Design for Mobile First

Mobile First means designing for mobile before designing for desktop or any other device (This will make the page display faster on smaller devices).

This means that we must make some changes in our CSS.

Instead of changing styles when the width gets *smaller* than 768px, we should change the design when the width gets *larger* than 768px. This will make our design Mobile First:

Example

```
/* For mobile phones: */
[class*="col-"] {
  width: 100%;
```

```
}

@media only screen and (min-width: 768px) {
  /* For desktop: */
  .col-1 {width: 8.33%;}
  .col-2 {width: 16.66%;}
  .col-3 {width: 25%;}
  .col-4 {width: 33.33%;}
  .col-5 {width: 41.66%;}
  .col-6 {width: 50%;}
  .col-7 {width: 58.33%;}
  .col-8 {width: 66.66%;}
  .col-9 {width: 75%;}
  .col-10 {width: 83.33%;}
  .col-11 {width: 91.66%;}
  .col-12 {width: 100%;}
}
```

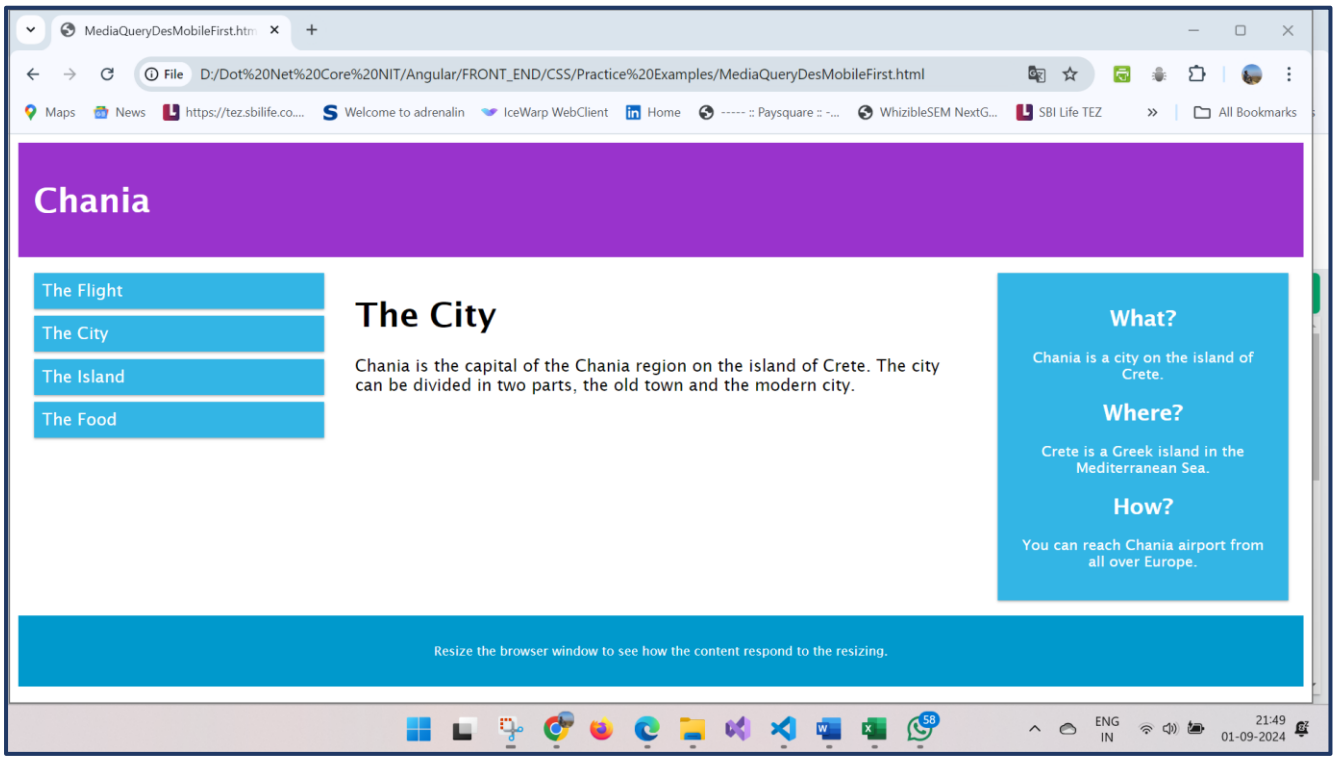
Example :

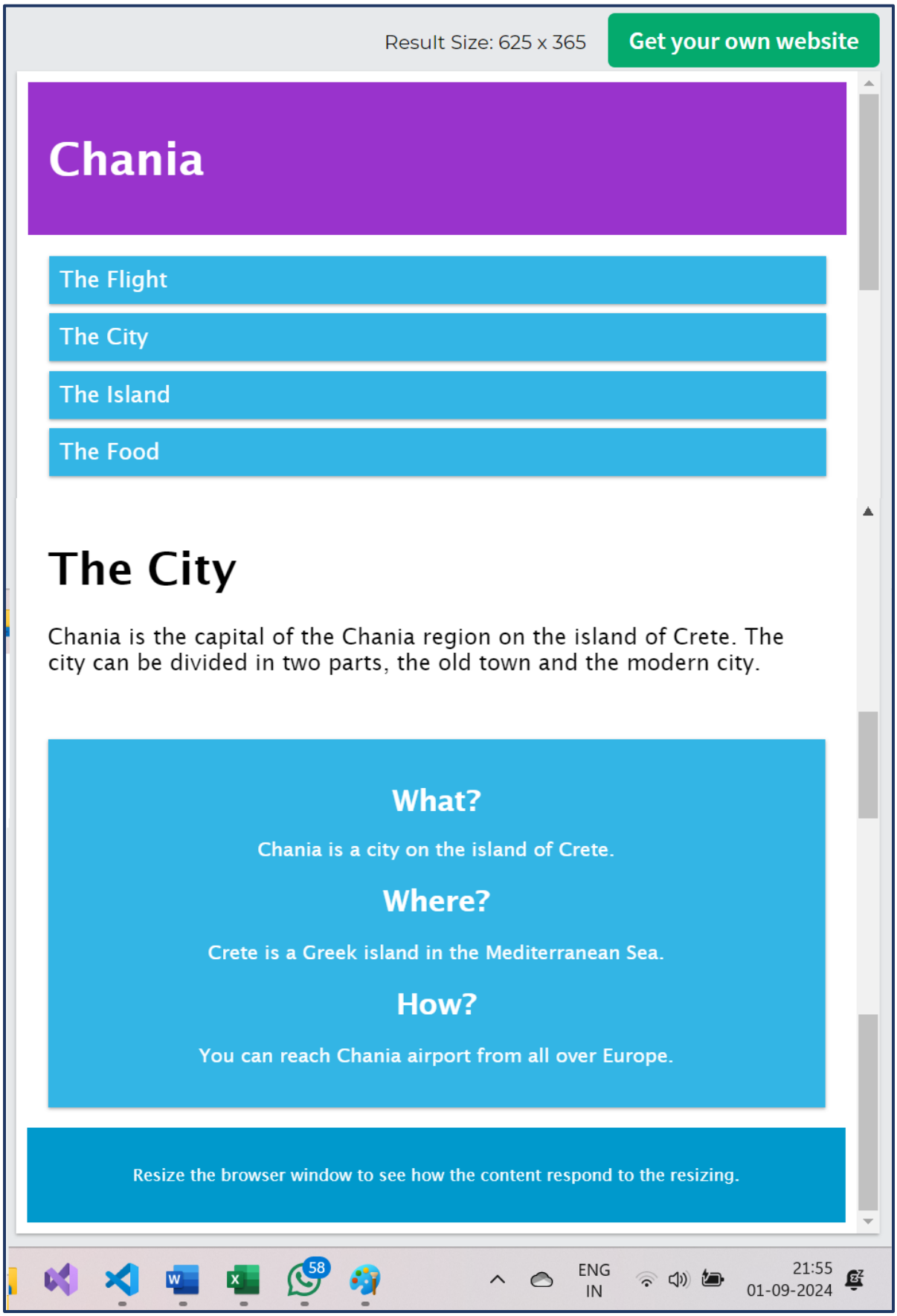
```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
* {
  box-sizing: border-box;
}

.row::after {
  content: "";
  clear: both;
  display: table;
}

[class*="col-"] {
  float: left;
  padding: 15px;
}

html {
  font-family: "Lucida Sans", sans-serif;
}

.header {
  background-color: #9933cc;
  color: #ffffff;
  padding: 15px;
}

.menu ul {
  list-style-type: none;
```

```
  margin: 0;
  padding: 0;
}

.menu li {
  padding: 8px;
  margin-bottom: 7px;
  background-color: #33b5e5;
  color: #ffffff;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}

.menu li:hover {
  background-color: #0099cc;
}

.aside {
  background-color: #33b5e5;
  padding: 15px;
  color: #ffffff;
  text-align: center;
  font-size: 14px;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}

.footer {
  background-color: #0099cc;
  color: #ffffff;
  text-align: center;
  font-size: 12px;
  padding: 15px;
}

/* For mobile phones: */
[class*="col-"] {
  width: 100%;
}

@media only screen and (min-width: 768px) {
  /* For desktop: */
  .col-1 {width: 8.33%;}
  .col-2 {width: 16.66%;}
  .col-3 {width: 25%;}
  .col-4 {width: 33.33%;}
  .col-5 {width: 41.66%;}
  .col-6 {width: 50%;}
  .col-7 {width: 58.33%;}
  .col-8 {width: 66.66%;}
  .col-9 {width: 75%;}
  .col-10 {width: 83.33%;}
  .col-11 {width: 91.66%;}
```

```
  .col-12 {width: 100%;}
}
</style>
</head>
<body>

<div class="header">
  <h1>Chania</h1>
</div>

<div class="row">
  <div class="col-3 menu">
    <ul>
      <li>The Flight</li>
      <li>The City</li>
      <li>The Island</li>
      <li>The Food</li>
    </ul>
  </div>

  <div class="col-6">
    <h1>The City</h1>
    <p>Chania is the capital of the Chania region on the island of Crete. The city
can be divided in two parts, the old town and the modern city.</p>
  </div>

  <div class="col-3 right">
    <div class="aside">
      <h2>What?</h2>
      <p>Chania is a city on the island of Crete.</p>
      <h2>Where?</h2>
      <p>Crete is a Greek island in the Mediterranean Sea.</p>
      <h2>How?</h2>
      <p>You can reach Chania airport from all over Europe.</p>
    </div>
  </div>
</div>

<div class="footer">
  <p>Resize the browser window to see how the content respond to the resizing.</p>
</div>

</body>
</html>
```

Output:


MediaQueryDesMobileFirst.htm
File D:/Dot%20Net%20Core%20NIT/Angular/FRONT_END/CSS/Practice%20Examples/MediaQueryDesMobileFirst.html
Maps
News
https://tez.sbilife.co....
Welcome to adrenalin
IceWarp WebClient
Home
----- :: Paysquare :: -...
WhizibleSEM NextG...
SBI Life TEZ
All Bookmarks
Chania
The Flight
The City
The Island
The Food
The City
Chania is the capital of the Chania region on the island of Crete. The city can be divided in two parts, the old town and the modern city.
What?
Chania is a city on the island of Crete.
Where?
Crete is a Greek island in the Mediterranean Sea.
How?
You can reach Chania airport from all over Europe.
Resize the browser window to see how the content respond to the resizing.
ENG
IN
21:49
01-09-2024

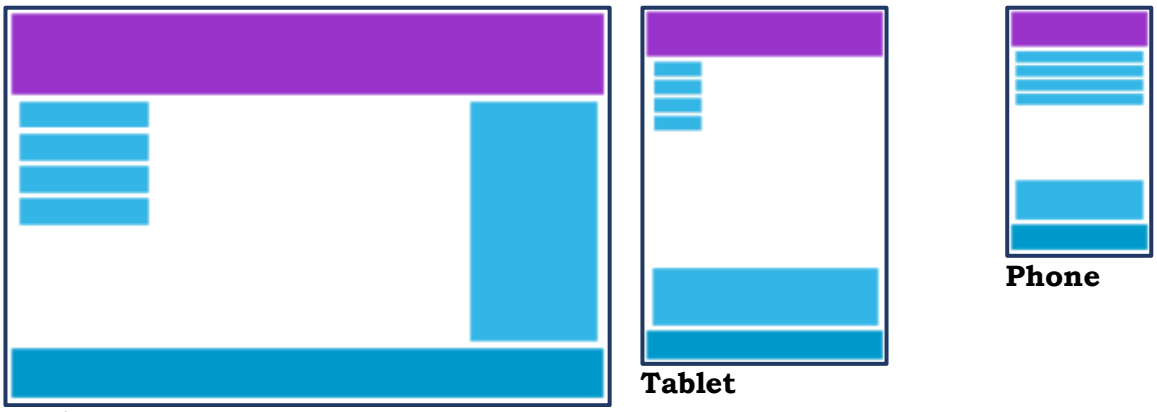


Example:

Another Breakpoint

You can add as many breakpoints as you like.

We will also insert a breakpoint between tablets and mobile phones.



We do this by adding one more media query (at 600px), and a set of new classes for devices larger than 600px (but smaller than 768px):

Example

Note that the two sets of classes are almost identical, the only difference is the name (col- and col-s-):

```
/* For mobile phones: */
[class*="col-"] {
  width: 100%;
}

@media only screen and (min-width: 600px) {
  /* For tablets: */
  .col-s-1 {width: 8.33%;}
  .col-s-2 {width: 16.66%;}
  .col-s-3 {width: 25%;}
  .col-s-4 {width: 33.33%;}
  .col-s-5 {width: 41.66%;}
  .col-s-6 {width: 50%;}
  .col-s-7 {width: 58.33%;}
  .col-s-8 {width: 66.66%;}
  .col-s-9 {width: 75%;}
  .col-s-10 {width: 83.33%;}
  .col-s-11 {width: 91.66%;}
  .col-s-12 {width: 100%;}
}

@media only screen and (min-width: 768px) {
  /* For desktop: */
  .col-1 {width: 8.33%;}
  .col-2 {width: 16.66%;}
  .col-3 {width: 25%;}
  .col-4 {width: 33.33%;}
  .col-5 {width: 41.66%;}
  .col-6 {width: 50%;}
  .col-7 {width: 58.33%;}
  .col-8 {width: 66.66%;}
  .col-9 {width: 75%;}
  .col-10 {width: 83.33%;}
  .col-11 {width: 91.66%;}
```

```
  .col-12 {width: 100%;}
}
```

It might seem odd that we have two sets of identical classes, but it gives us the opportunity *in HTML*, to decide what will happen with the columns at each breakpoint:
HTML Example

For desktop:

The first and the third section will both span 3 columns each. The middle section will span 6 columns.

For tablets:

The first section will span 3 columns, the second will span 9, and the third section will be displayed below the first two sections, and it will span 12 columns:

```
<div class="row">
  <div class="col-3 col-s-3">...</div>
  <div class="col-6 col-s-9">...</div>
  <div class="col-3 col-s-12">...</div>
</div>
```

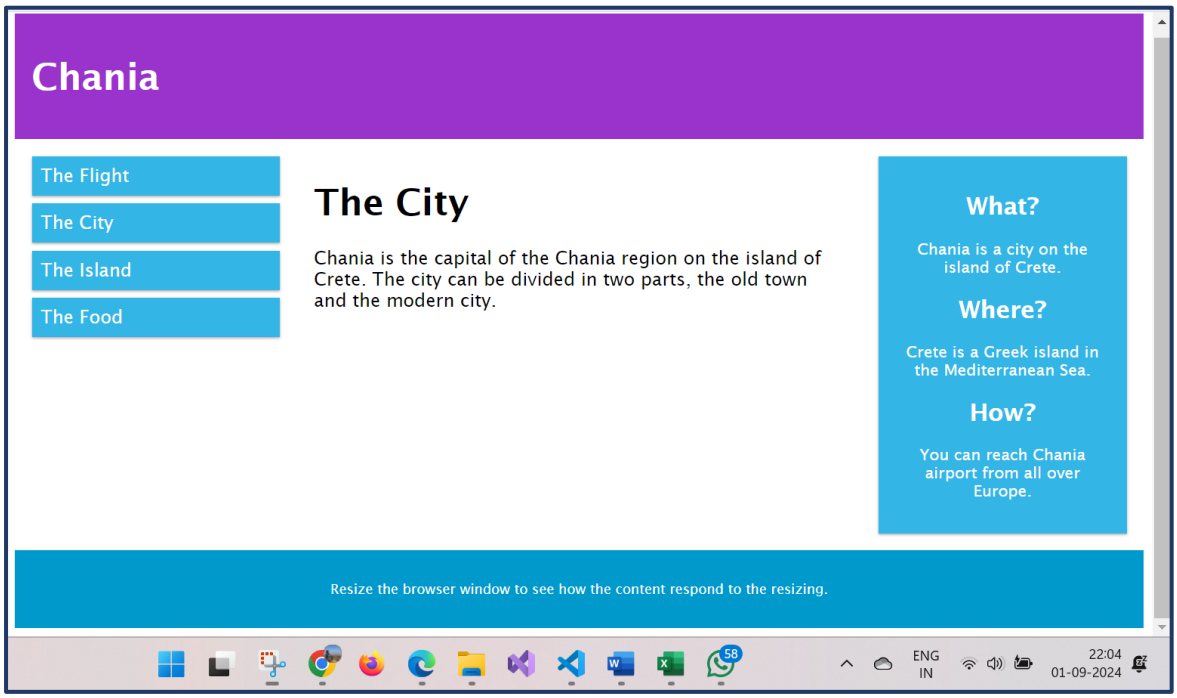
Example:

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
* {
  box-sizing: border-box;
}
.row::after {
  content: "";
  clear: both;
  display: table;
}
[class*="col-"] {
  float: left;
  padding: 15px;
}
html {
  font-family: "Lucida Sans", sans-serif;
}
.header {
  background-color: #9933cc;
  color: #ffffff;
  padding: 15px;
}
.menu ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}
.menu li {
  padding: 8px;
  margin-bottom: 7px;
  background-color: #33b5e5;
```

```
  color: #ffffff;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}
.menu li:hover {
  background-color: #0099cc;
}
.aside {
  background-color: #33b5e5;
  padding: 15px;
  color: #ffffff;
  text-align: center;
  font-size: 14px;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}
.footer {
  background-color: #0099cc;
  color: #ffffff;
  text-align: center;
  font-size: 12px;
  padding: 15px;
}
/* For mobile phones: */
[class*="col-"] {
  width: 100%;
}
@media only screen and (min-width: 600px) {
  /* For tablets: */
  .col-s-1 {width: 8.33%;}
  .col-s-2 {width: 16.66%;}
  .col-s-3 {width: 25%;}
  .col-s-4 {width: 33.33%;}
  .col-s-5 {width: 41.66%;}
  .col-s-6 {width: 50%;}
  .col-s-7 {width: 58.33%;}
  .col-s-8 {width: 66.66%;}
  .col-s-9 {width: 75%;}
  .col-s-10 {width: 83.33%;}
  .col-s-11 {width: 91.66%;}
  .col-s-12 {width: 100%;}
}
@media only screen and (min-width: 768px) {
  /* For desktop: */
  .col-1 {width: 8.33%;}
  .col-2 {width: 16.66%;}
  .col-3 {width: 25%;}
  .col-4 {width: 33.33%;}
  .col-5 {width: 41.66%;}
  .col-6 {width: 50%;}
  .col-7 {width: 58.33%;}
  .col-8 {width: 66.66%;}
  .col-9 {width: 75%;}
```

```
  .col-10 {width: 83.33%;}
  .col-11 {width: 91.66%;}
  .col-12 {width: 100%;}
}
</style>
</head>
<body>
<div class="header">
  <h1>Chania</h1>
</div>
<div class="row">
  <div class="col-3 col-s-3 menu">
    <ul>
      <li>The Flight</li>
      <li>The City</li>
      <li>The Island</li>
      <li>The Food</li>
    </ul>
  </div>
  <div class="col-6 col-s-9">
    <h1>The City</h1>
    <p>Chania is the capital of the Chania region on the island of Crete. The city
can be divided in two parts, the old town and the modern city.</p>
  </div>
  <div class="col-3 col-s-12">
    <div class="aside">
      <h2>What?</h2>
      <p>Chania is a city on the island of Crete.</p>
      <h2>Where?</h2>
      <p>Crete is a Greek island in the Mediterranean Sea.</p>
      <h2>How?</h2>
      <p>You can reach Chania airport from all over Europe.</p>
    </div>
  </div>
</div>
<div class="footer">
  <p>Resize the browser window to see how the content respond to the resizing.</p>
</div>
</body>
</html>
```

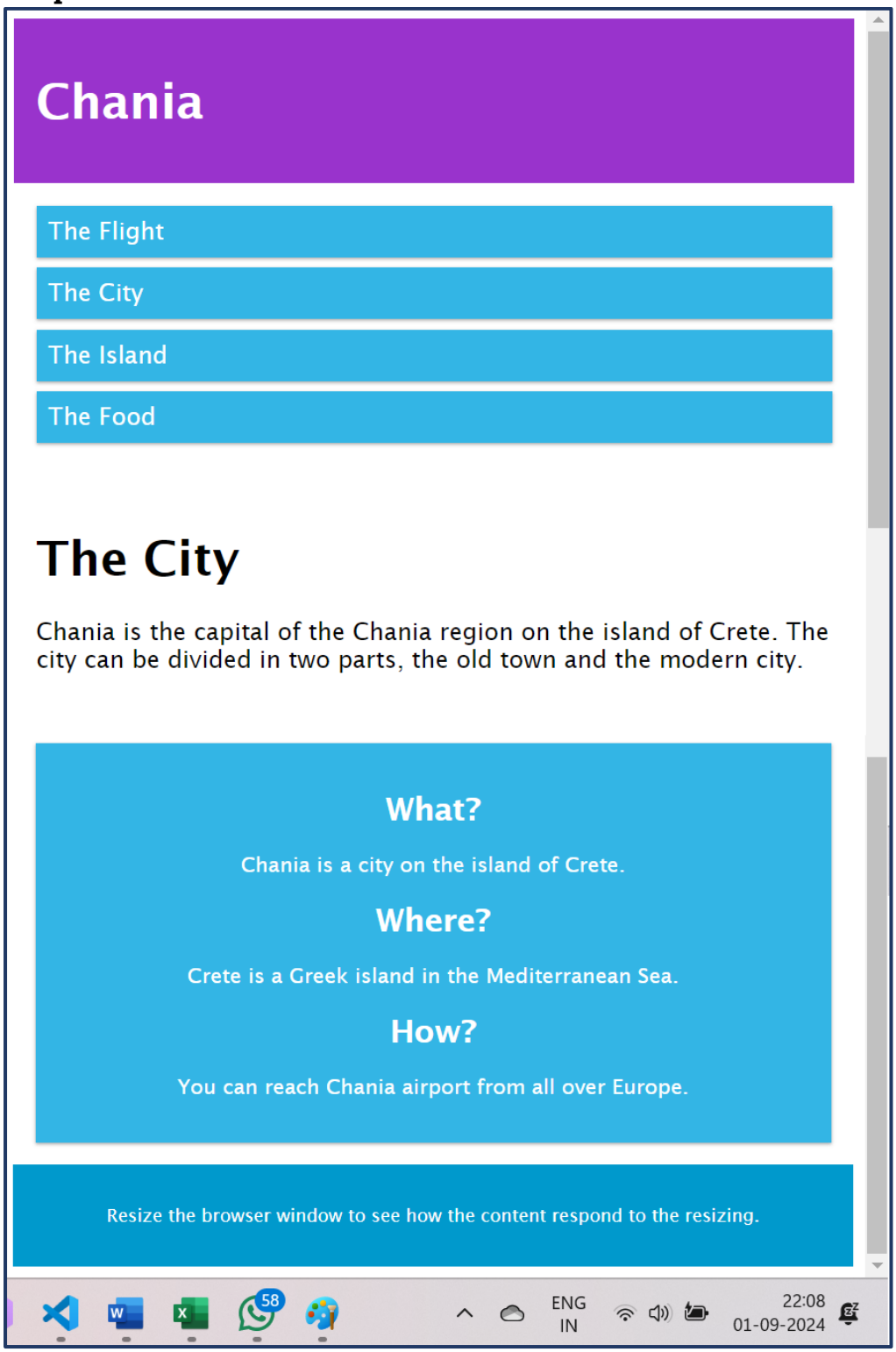
Output:


Chania
The Flight
The City
The Island
The Food
The City
Chania is the capital of the Chania region on the island of Crete. The city can be divided in two parts, the old town and the modern city.
What?
Chania is a city on the island of Crete.
Where?
Crete is a Greek island in the Mediterranean Sea.
How?
You can reach Chania airport from all over Europe.
Resize the browser window to see how the content respond to the resizing.
ENG
IN
22:04
01-09-2024

Output 2:


Chania
The Flight
The City
The Island
The Food
The City
Chania is the capital of the Chania region on the island of Crete. The city can be divided in two parts, the old town and the modern city.
What?
Chania is a city on the island of Crete.
Where?
Crete is a Greek island in the Mediterranean Sea.
How?
You can reach Chania airport from all over Europe.
Resize the browser window to see how the content respond to the resizing.
ENG
IN
22:05
01-09-2024

Output 3:



Typical Device Breakpoints

There are tons of screens and devices with different heights and widths, so it is hard to create an exact breakpoint for each device. To keep things simple you could target five groups:

Example

/* Extra small devices (phones, 600px and down) */

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

/* Small devices (portrait tablets and large phones, 600px and up) */

@media only screen and (min-width: 600px) {...}

/* Medium devices (landscape tablets, 768px and up) */
@media only screen and (min-width: 768px) {...}

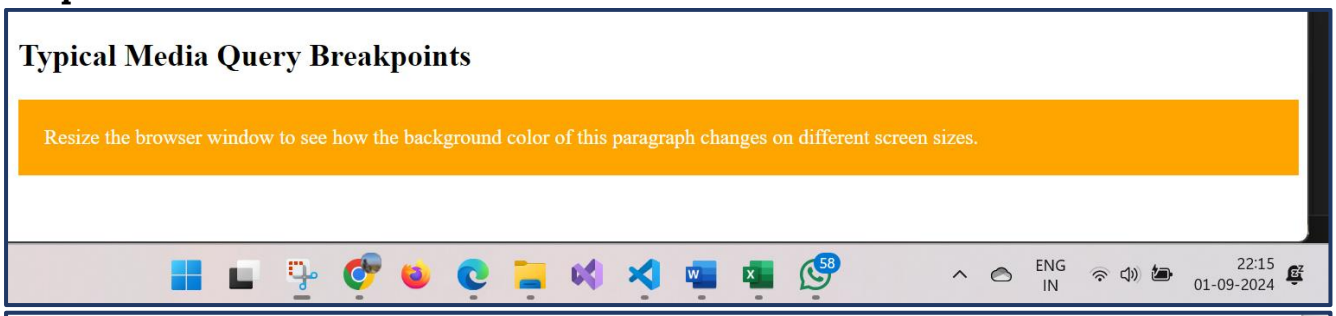
/* Large devices (laptops/desktops, 992px and up) */
@media only screen and (min-width: 992px) {...}

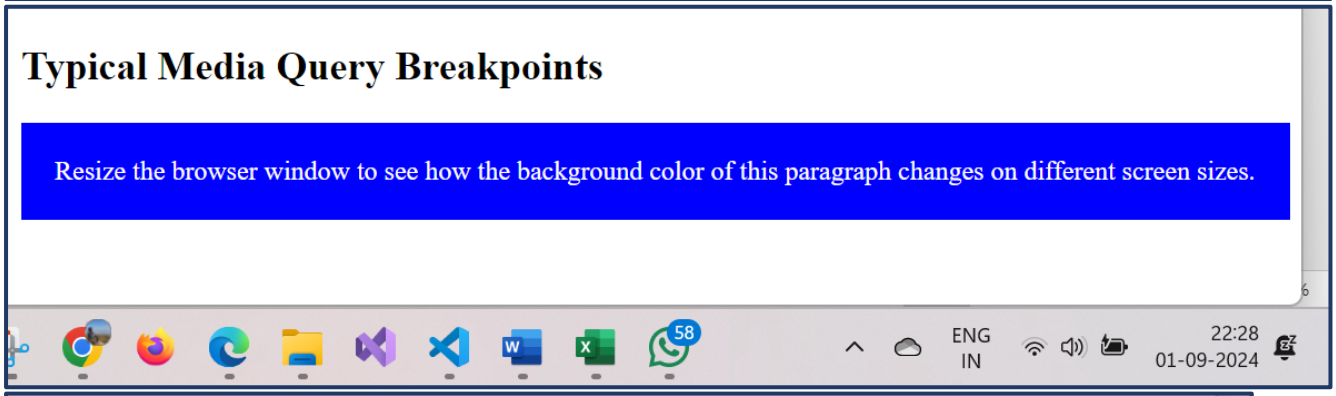
/* Extra large devices (large laptops and desktops, 1200px and up) */
@media only screen and (min-width: 1200px) {...}

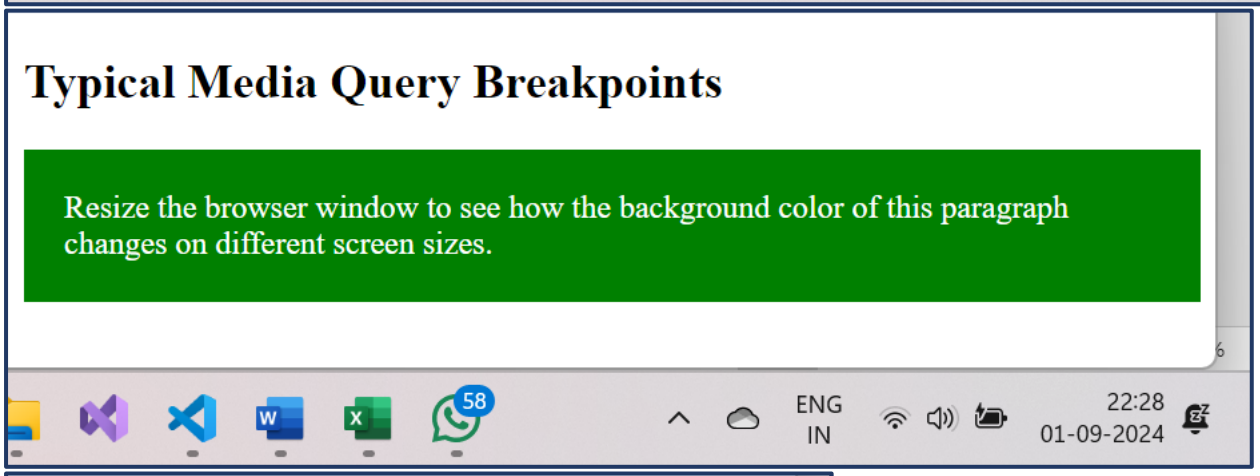
Example

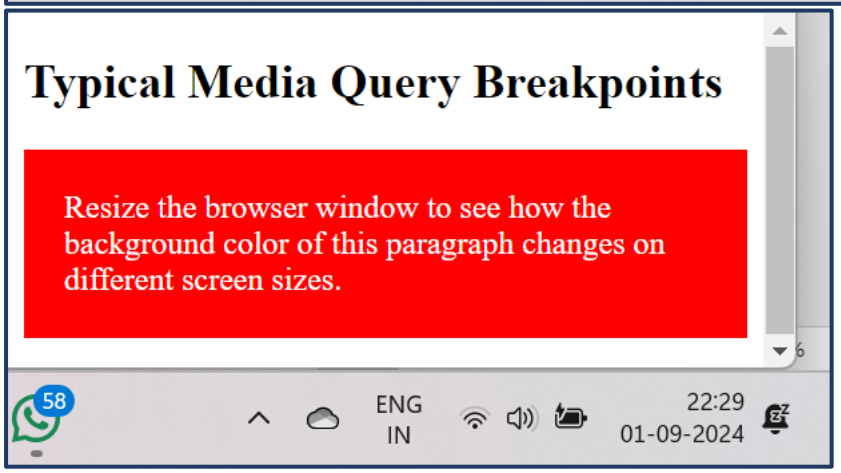
```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
.example {
  padding: 20px;
  color: white;
}
/* Extra small devices (phones, 600px and down) */
@media only screen and (max-width: 600px) {
  .example {background: red;}
}
/* Small devices (portrait tablets and large phones, 600px and up) */
@media only screen and (min-width: 600px) {
  .example {background: green;}
}
/* Medium devices (landscape tablets, 768px and up) */
@media only screen and (min-width: 768px) {
  .example {background: blue;}
}
/* Large devices (laptops/desktops, 992px and up) */
@media only screen and (min-width: 992px) {
  .example {background: orange;}
}
/* Extra large devices (large laptops and desktops, 1200px and up) */
@media only screen and (min-width: 1200px) {
  .example {background: pink;}
}
</style>
</head>
<body>
<h2>Typical Media Query Breakpoints</h2>
<p class="example">Resize the browser window to see how the background color of this
paragraph changes on different screen sizes.</p>
</body>
</html>
```

Output:


Typical Media Query Breakpoints
Resize the browser window to see how the background color of this paragraph changes on different screen sizes.
ENG
IN
22:15
01-09-2024


Typical Media Query Breakpoints
Resize the browser window to see how the background color of this paragraph changes on different screen sizes.
ENG
IN
22:28
01-09-2024


Typical Media Query Breakpoints
Resize the browser window to see how the background color of this paragraph changes on different screen sizes.
ENG
IN
22:28
01-09-2024


Typical Media Query Breakpoints
Resize the browser window to see how the background color of this paragraph changes on different screen sizes.
ENG
IN
22:29
01-09-2024

CSS Cascading and Inheritance:

CSS Cascading and **Inheritance** are core concepts that govern how CSS rules are applied to HTML elements. They determine the final style of elements when multiple CSS rules apply to the same element. Understanding these concepts helps in writing more efficient and predictable CSS code.

1. CSS Cascading

The term "cascading" refers to the process of resolving conflicts between multiple CSS rules that apply to the same element. When there are conflicting styles, the browser decides which one to apply by following a specific order of priority. The cascade involves three main factors:

- **Importance**: Rules marked with !important have the highest priority.
- **Specificity**: More specific selectors (e.g., #id over .class or element) take precedence.
- **Source Order**: When rules have equal specificity and importance, the rule that comes last in the stylesheet will be applied.

Example of Cascading:

```
/* Global styles */
p {
    color: blue;
}
/* Class selector */
.text-red {
    color: red;
}
/* ID selector */
#unique-paragraph {
    color: green;
}
/* Important rule */
.text-red-important {
    color: red !important;
}
<p>This paragraph is blue (global style).</p>
<p class="text-red">This paragraph is red (class style overrides global style).</p>
<p id="unique-paragraph">This paragraph is green (ID style overrides class and global styles).</p>
<p class="text-red-important" id="unique-paragraph">This paragraph is red (important overrides ID style).</p>
```

Explanation:

1. **Global Style**: The first paragraph is styled blue because the global rule p { color: blue; } applies.
2. **Class Selector**: The second paragraph uses the class .text-red, so it overrides the global style and becomes red.
3. **ID Selector**: The third paragraph uses the #unique-paragraph ID, which is more specific than a class, so it overrides the class and becomes green.
4. **Important Rule**: The fourth paragraph uses .text-red-important, which includes the !important declaration. This rule overrides even the more specific #unique-paragraph ID selector, making the text red.

2. CSS Inheritance

Inheritance is the process by which certain CSS properties are passed down from parent elements to their child elements. Not all properties are inherited; some must be explicitly set.

Inherited Properties:

Common properties that are inherited include:

- color

- font-family
- font-size
- line-height
- text-align

Non-Inherited Properties:

Properties like margin, padding, border, and background are not inherited by default.

Example of Inheritance:

```
/* Parent element style */
.parent {
    color: blue;
    font-family: Arial, sans-serif;
}
/* Child element style (no color defined) */
.child {
    font-size: 20px;
}
<div class="parent">
    This is the parent element.
    <p class="child">This is the child element, inheriting color and font-family from the parent.</p>
</div>
```

Explanation:

- The **parent element** has the color and font-family properties set.
- The **child element** does not have a color or font-family specified, so it inherits these properties from the parent. The child, however, has its own font-size defined.

Inheritance with inherit, initial, and unset:

You can control inheritance with special keywords:

- **inherit**: Forces a property to inherit the value from its parent.
- **initial**: Sets the property to its initial (default) value as defined by the browser.
- **unset**: Resets the property to its inherited value if it is inheritable, otherwise sets it to its initial value.

Example:

```
/* Parent element style */
.parent {
    color: blue;
    font-family: Arial, sans-serif;
}
/* Child element style */
.child {
    color: inherit; /* Inherits the color from the parent */
    font-size: initial; /* Resets to the default browser font size */
}
<div class="parent">
    This is the parent element.
    <p class="child">This is the child element, inheriting color and resetting font-size.</p>
</div>
```

Explanation:

- The **child element** explicitly inherits the color from the parent using the inherit keyword.
- The **child element** resets its font-size to the default value defined by the browser using the initial keyword.

Conclusion:

- **Cascading** resolves conflicts between multiple CSS rules based on importance, specificity, and source order.
- **Inheritance** passes down certain CSS properties from parent elements to child elements, but not all properties are inherited by default.

By understanding cascading and inheritance, you can better control how styles are applied across your website, making your CSS more predictable and easier to manage.

CSS Variables (Custom Properties):

CSS Variables, also known as **Custom Properties**, allow you to define values that can be reused throughout your stylesheet. This helps create consistent design and makes it easier to update styles globally. CSS Variables are defined using the -- prefix and can be accessed using the var() function.

Defining and Using CSS Variables

You can define CSS variables globally using the :root selector or within a specific element. The :root selector represents the highest level in the CSS hierarchy, so variables defined here are available throughout your entire stylesheet.

Example of Defining and Using CSS Variables:

```
/* Defining CSS variables in the :root pseudo-class */
:root {
  --primary-color: #3498db;
  --secondary-color: #2ecc71;
  --font-size-large: 20px;
  --padding-default: 10px;
}
body {
  background-color: var(--primary-color);
  font-size: var(--font-size-large);
  padding: var(--padding-default);
}
button {
  background-color: var(--secondary-color);
  color: white;
  padding: var(--padding-default);
  border: none;
  border-radius: 5px;
}
<body>
  <h1>Welcome to My Website</h1>
  <button>Click Me</button>
</body>
```

Explanation:

- **Defining Variables**: In the :root block, we've defined four custom properties: --primary-color, --secondary-color, --font-size-large, and --padding-default.
- **Using Variables**: We then use these variables in various CSS rules by referencing them with the var() function. For example, the background-color of the body is set to var(--primary-color), which resolves to #3498db.

Advantages of CSS Variables:

1. **Reusability**: You can reuse the same variable throughout your stylesheet, which helps maintain consistency.
2. **Easier Updates**: Changing the value of a variable in one place automatically updates all instances where it's used.

3. **Theming**: CSS Variables make it easier to create and manage themes, allowing you to change the look and feel of a site by adjusting just a few variables.

Scope of CSS Variables

CSS variables are scoped, meaning they can be defined globally (e.g., in :root) or within specific elements. Variables defined inside an element only apply to that element and its descendants.

Example of Scoped Variables:

```
:root {
  --primary-color: #3498db;
}

.container {
  --primary-color: #e74c3c; /* Redefining the variable within the container class */
}

button {
  background-color: var(--primary-color); /* Will be blue outside of .container, but red inside */
}
<div>
  <button>Outside Container</button>
</div>
<div class="container">
  <button>Inside Container</button>
</div>
```

Explanation:

- **Global Variable**: The --primary-color variable is initially defined globally as #3498db (blue).
- **Scoped Variable**: Inside the .container class, the --primary-color variable is redefined as #e74c3c (red). This scoped variable overrides the global one within the .container element, so the button inside .container will have a red background, while the one outside remains blue.

CSS Variable with Fallback Value

CSS Variables also support fallback values in case the variable is not defined.

```
p {
  color: var(--undefined-color, black); /* If --undefined-color is not set, use black */
}
```

In this case, if --undefined-color is not defined, the color of the paragraph will default to black.

Conclusion

CSS Variables (Custom Properties) offer a flexible and powerful way to manage styles. By using variables, you can make your CSS more maintainable and scalable, particularly in large projects or when implementing themes. They are a great addition to modern CSS practices.

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

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 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? #

The short answer is, in most cases, either.

The double colon (::) was introduced in CSS3 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.
