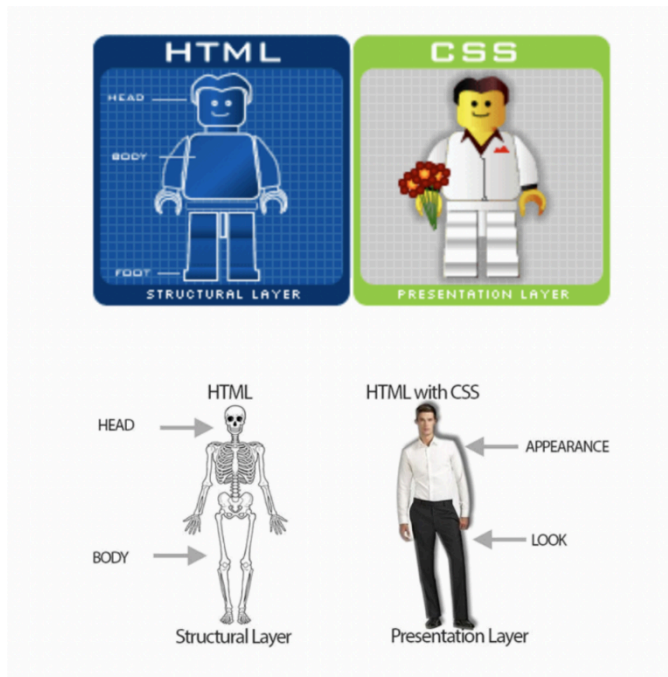


## Introduction to CSS

**Cascading Style Sheet:** The CSS concept is used to apply styles for the HTML elements.

If we want to customize the default appearance of the HTML elements. by using CSS, we can apply the required changes to any HTML elements, like colors, size, border, positions, alignment, etc.



Initial release	1996
Author	Håkon Wium Lie, Bert Bors [W3C]
File extension	.css
MIME Type	text/css
Latest version	CSS3 (June 2012)

**The CSS specifications specify rules that describe how the content of elements within our HTML document should appear.**

**Syntax:**

```
td {width: 36px;}
```

In this example, there are two parts

1. Selector: **td**
2. declaration: **{width: 36px;}**

**Selector:** which indicates which element or elements (if multiple elements, they must be separated by a comma (,)) the declaration applies to.

**Declaration:** It sets how the selector should be styled.

The declaration is also split into two parts, separated by a colon.

1. **Property**
2. **Value**

Example:

```
h1, h2, p {  
    font-weight: bold;  
    font-family: arial;  
    color: red;  
    background-color: pink;  
}
```

### **Implementation of CSS:**

We can implement CSS concept in three ways in our HTML document:

1. Inline CSS (tag level)
2. Page-level CSS/internal CSS (inside the HTML document)
3. External CSS (separate .css file)

### **Inline CSS:**

- For inline CSS implementation, we should use the "style" attribute inside any HTML element.

Example:

```
<h1 style="color: green; font-size: 50px">Welcome</h1>
```

### **Page-level CSS/Internal CSS:**

- If we want to apply the same style for the multiple tags on the page, we can use page-level styles.
- Page-level styles are implemented using **<style>** tag.
- **<style>** tag should be defined inside the **<head>** tag.

Example:

```
<head>

  <style>
    h1 {
      text-align: center;
      color: green;
      background-color: pink;
    }
  </style>

</head>

<body>

  <h1>Welcome to My class</h1>

  <h1>Welcome to Sage</h1>

</body>
```

### External CSS file:

- If we want to apply the same style for multiple pages in the website and make the HTML document neat and clear, so we should use an external css file.
- External CSS style is implemented by using a separate file (with .css extension)
- To refer to the .css file from HTML pages, we should use the **<link>** tag inside the **<head>** tag

Example:

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

### Inline Styles

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

### Internal Styles:

```
<head>
  <style>
    p {
      color: green;
      font-size: 18px;
    }
  </style>
</head>
<body>
  <p>This is a green text with 18px font size.</p>
</body>
```

### External Styles:

```
css
p {
  color: red;
  font-size: 20px;
}

In your HTML file:
html
<head>
  <link rel="stylesheet" type="text/css" href="styles.
</head>
<body>
  <p>This is a red text with 20px font size.</p>
</body>
```

### The major advantages of using external CSS are:

1. **Modularity and Reusability:** External CSS files can be reused across multiple web pages, promoting consistency in design and layout throughout a website. This modularity also simplifies maintenance as changes made to the external CSS file are automatically applied to all linked pages.
2. **Caching:** Once the browser downloads an external CSS file, it can cache it, which means subsequent page loads will be faster as the browser doesn't need to re-download the CSS file.
3. **Reduced Code Redundancy:** By separating CSS from HTML, you can avoid duplicating style definitions across multiple HTML files, leading to cleaner and more maintainable code.
4. **Ease of Maintenance:** External CSS files centralize styling rules, making it easier to manage and update styles across an entire website. This can save time and effort in the long run, especially for larger projects.

### Disadvantages:

1. **HTTP Requests:** Each external CSS file requires a separate HTTP request, which can slightly slow down the initial loading time of a webpage, especially if there are multiple CSS files to download.
2. **Dependency:** External CSS files are dependent on network connectivity. If the CSS file fails to load due to network issues or if the file is moved or deleted, it can affect the appearance and functionality of the website.
3. **Potential FOUC (Flash of Unstyled Content):** If the external CSS file takes time to load, users might briefly see unstyled content before the styles are applied, leading to a poor user experience.
4. **Increased Complexity:** Having separate CSS files may increase the complexity of managing a website, especially for smaller projects where inline or internal CSS may be more straightforward.

Overall, while external CSS files offer numerous benefits in terms of modularity, reusability, and ease of maintenance, they also come with some drawbacks related to performance and dependency on network connectivity. It's essential to consider these factors when deciding whether to use external CSS files in web development projects.

### CSS Selectors:

- CSS offers several types of selectors, each serving a unique purpose and allowing developers to target specific elements on a web page.
  - Here are some of the basic types of CSS selectors:
1. **Universal Selector:** It matches any element in the document. It is a wildcard selector that selects all elements, regardless of their type, class, ID, or other attributes.

Example:

```
* {  
  color: red;  
}
```

This rule renders the content of every element in our document in red.

- 2. Element Selector:** Selects HTML elements based on their element type. For example, `p` selects all `<p>` elements.

Example:

index.html: It will select all the `<p>` elements of this document.

```
<!DOCTYPE html>  
<html lang="en">  
  
  <head>  
  
    <style>  
      p {  
        background-color: green;  
      }  
    </style>  
  
  </head>  
  
  <body>  
  
    <p>This is the paragraph 1</p>  
    <p>This is the paragraph 2</p>  
    <p>This is the paragraph 3</p>  
    <p>This is the paragraph 4</p>  
    <p>This is the paragraph 5</p>  
    <p>This is the paragraph 6</p>  
  
  </body>  
  
</html>
```

- 3. ID Selector:** Selects an element based on its unique **id** attribute. It is denoted by a hash (`#`) followed by the id value. For example, `#x1` selects the element with `id="x1"`.

Example:

Index.html: It will select only that `<p>` element whose id is x1.

```
<!DOCTYPE html>  
<html lang="en">
```

```

<head>

    <style>
        #x1 {
            background-color: green;
        }
    </style>

</head>

<body>

    <p>This is the paragraph 1</p>
    <p>This is the paragraph 2</p>
    <p>This is the paragraph 3</p>
    <p id="x1">This is the paragraph 4</p>
    <p>This is the paragraph 5</p>
    <p>This is the paragraph 6</p>

</body>

</html>

```

**Note: The Id attribute should be unique for any element.**

4. **Class Selector:** Selects elements based on their class attribute. It is denoted by a period (.) followed by the class name. For example, **.btn** selects all elements with **class="btn"**.
  - If we want to group some HTML elements logically, we can apply the **class** attribute for those elements.
  - Using the same **class** name for different HTML elements we can group multiple elements logically.

Example:

**index.html:** If we want to select 1<sup>st</sup>, 4<sup>th</sup>, and 6<sup>th</sup> <p> tag, we need to group these <p> tag with the same class name.

```

<!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>
    .c1 {
        background-color: green;
    }
</style>

</head>

<body>

    <p class="c1">This is the paragraph 1</p>
    <p>This is the paragraph 2</p>
    <p>This is the paragraph 3</p>
    <p class="c1">This is the paragraph 4</p>
    <p>This is the paragraph 5</p>
    <p class="c1">This is the paragraph 6</p>

</body>

</html>
```

Note: We can apply an id attribute and class attribute to an element same time. And if multiple classes are there then they will be separated with a space.

Example:

```
<p id="x1" class="c1 c2">This is a Paragraph</p>
```

## Priority of selectors:

In CSS, the priority of selectors determines which style rules are applied when multiple rules target the same element. The priority is determined by the specificity of selectors and the order they appear in the stylesheet.

1. **Inline Styles:** Styles applied directly to an HTML element using the **style** attribute. These have the highest priority and override all other styles.
2. **IDs:** Selectors targeting elements by their **id** attribute. They have a higher specificity than classes and elements.
3. **Classes, Attributes, and Pseudo-Classes:** Selectors targeting elements by class name, attribute, or pseudo-class (:hover, :focus, etc.). These are less specific than IDs but more specific than element selectors.
4. **Element Selectors:** Selectors targeting elements by their type (e.g., **div**, **p**, **h1**). These have the lowest specificity.
5. **!important:** This is a special flag that can be added to a CSS declaration to give it the highest priority, even above inline styles. However, its use is generally discouraged because it can make styles difficult to manage and override.
6. **Specificity:** If two selectors apply conflicting styles to the same element, the one with higher specificity wins. Specificity is calculated based on the number of IDs, classes, and elements in the selector.
7. **Order of Appearance:** If two conflicting rules have the same specificity, the one that appears last in the stylesheet will take precedence.
- 8.

## Scores of various selectors

Selector Name	Score
Inline	1000
Id	100
Class	10
Tag	1
Universal	0



Style > Id > Class > Type > Universal

(1000)    (100)    (10)    (1)    (0)

**Specificity Value:**

1. Inline (1000)
2. Id (100)
3. Class & pseudo-classes (10)
4. Tag & pseudo-elements (1)
5. Universal (0)

Follow this order **SICTU**

- a. S - inline style
- b. I - Id selector
- c. C - Class selector
- d. T - Type/Tag selector
- e. U - Universal selector

**1. When different selectors:**

- A. Different properties: All properties getting applied.
- B. Same properties: The selector with higher specificity score getting applied.

**2. When same selectors :**

- A. Different properties: All properties getting applied.
- B. Same properties: The property at the bottom getting applied.

**The highest value represents the highest priority. If two selectors have the same specificity, the one defined later in the stylesheet takes precedence (unless overridden by !important).**

**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>
    /* specificity value (100 + 1) = 101 */
    #box>p {
      color: red;
    }

    /* specificity value = 100 */
    #p1 {
      color: green;
    }
  </style>

</head>

<body>

  <h1>Welcome to Simple Webpage </h1>

  <div id="box">

    <p>Para 1</p>
    <p>Para 2</p>
    <p id="p1">Para 3</p>
    <p>Para 4</p>

  </div>
</body>

</html>

```

## CSS Styling properties:

### CSS Color:

CSS uses color values to specify a color. Typically, these are used to set a color either for the foreground of an element (i.e. its text) or else for the background of the element. They can also be used to affect the color of borders and other decorative effects.

We can use color in the following ways:

- a. Predefined color name (more than 100 names we can google)
- b. RGB
- c. RGBA
- d. HEX
- e. HSL
- f. HSLA

**RGB:** Colors can be defined using the `rgb()` function, which takes three parameters representing the red, green, and blue values.

**`rgb(red, green, blue)`**

Values from 0 to 255

Example

`rgb(0,0,0)`: black

`rgb(255, 255, 255)`: white

`rgb(255, 0, 0)`: red

`rgb(0, 255, 0)`: green

`rgb(0, 0, 255)`: blue

**RGBA:** Here **A** means Alpha, the alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent)

Example: **`rgba(255,99,71,0.5)`**

**NOTE:** All browsers do not support the `rgb()` property of color so it is recommended not to use it.

**HEX:** It is similar to `rgb()`, but A hexadecimal is a 6-digit representation of a color. The first two digits(rr) represent a red value, the next two are a green value(gg), and the last is a blue value(bb). Each hexadecimal code will be preceded by a pound or hash sign '#'.  
These symbols or values are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F

**Note:** To specify the hexadecimal codes, you can use upper-case or lower-case letters.

Example: **`#rrggbb`**

`#000000`: black

`#ff0000`: red

`#ffffff`: white

`#00ff00`: green

`#0000ff`: blue

#fffacd: Lemon Chiffon

**Short Hexadecimal Codes:** This is a shorter form of the six-digit notation. In this format, each digit is replicated to arrive at an equivalent six-digit value. For example: **#6A7** becomes **#66AA77**.

### HSL: Hue, Saturation, Lightness

This color value is specified using the `hsl()` function.

**Hue:** It is a degree on the color wheel from 0 to 360. where

0 = red

120 = green

240 = blue

**Saturation:** It is a percentage value, 0% means a shade of grey and 100% means full color.

**Lightness:** It is also a percentage value, 0% is black 50% is neither light nor dark and 100% is white.

Example: `hsl(0, 0%, 50%)`

**HSLA:** Here **A** represents Alpha (transparency), similar to `rgba`.

Example: `hsla(355, 70%, 50%, 0.4)`

### CSS Background:

The background property of CSS is used to set the background of an element. It can be used to apply a single background image or multiple background images, as well as define the background color, size, position, repeat behavior, and other related properties.

It is a versatile tool for styling the appearance of elements and adding visual interest to web pages.

The **background** is a shorthand for the following properties:

1. **background-color:** Sets the background color of an element.
2. **background-image:** Sets one or more background image(s) on an element. Here image path we mention using the **url()** function.

Example:

```
background-image: url(./images/f1.jpg);
```

3. **background-repeat:** Controls the repetition of an image in the background. By default image is repeated in both x and y directions.
  - repeat-x:** The background image will be repeated only horizontally
  - repeat-y:** The background image will be repeated only vertically
  - no-repeat:** The background-image will not be repeated
  - repeat:** The background image will be repeated both vertically and horizontally. This is the default.
4. **background-attachment:** Specifies the position of the background relative to the viewport, either fixed or scrollable.
  - scroll:** The background scrolls along with the element. This is default
  - fixed:** The background is fixed with regard to the viewport.
5. **background-size:** Controls the size of the background image.
  - auto:** Default value, The background image is displayed in its original size.
  - cover:** Resize the background image to cover the entire container, even if it has to stretch the image or cut a little bit off one of the edges
  - contain:** Resize the background image to make sure the image is fully visible.
6. **background-position:** This property sets the starting position of a background image. Default value: left top (It works with the x and y axis), we can give these values in pixels and percent also.

The following values place the background image accordingly.

**left top**  
**left center**  
**left bottom**  
**right top**  
**right center**  
**right bottom**  
**center top**  
**center center**  
**center bottom**

Example:

```
background-position: left top;  
background-position: 50px 100px;
```

**x% y%:** The first value is the horizontal position and the second value is the vertical. The left top is 0% 0% and the right bottom is 100% 100%.

```
background-position: 5% 10%;
```

## CSS height and width:

All the HTML elements follow the box model. The height and width properties in CSS are used to set the height and width of the boxes. Its value can be set using **px**, **%**, or **auto**.

Example:

**Index.html:**

```
<!DOCTYPE html>
<html lang="en">

  <head>

    <style>
      div {

        background-color: blue;
        height: 200px;
        width: 200px;
      }
    </style>

  </head>

  <body>

    <div>

    </div>

  </body>

</html>
```

We can specify the height using **vh** and the width using **vw** also.

- vh: Viewport height
- vw: Viewport width

The viewport is the visible area of a web page. It varies depending on the device and the size of the browser window. The viewport size determines how much content can be displayed without scrolling.

A value of 1vh is equal to 1% of the viewport height. A value of 100vh is equal to 100% of the viewport height.

Example:

```
div {  
    height: 50vh;  
    /* 50% of the viewport's height */  
    width: 100vw;  
    /* 100% of the viewport's width */  
    background-color: lightblue;  
}
```

## Different types of measurement units in CSS:

### 1. Absolute Length Units:

- **px** (pixels): A pixel is a unit of measurement that represents a single dot on a screen. It is a fixed-size unit and does not change with the user's settings.
- **in** (inches), **cm** (centimeters), **mm** (millimeters): These units are absolute physical units and are mostly used for printing.
- **pt** (points), **pc** (picas): These units are commonly used in print design.
- *1 inch = 72 points = 6 picas*

### 2. Relative Length Units:

- **em**: The 'em' unit is relative to the **font size** of the element. If the font size of the parent element is 16px, then 1em is equal to 16px.
- **rem**: Similar to 'em', but it is relative to the root element's **font size**.
- **%**: Represents a percentage of the parent element's size.

### 3. Viewport Percentage Lengths:

- **vw** (viewport width): Represents a percentage of the viewport's width.
- **vh** (viewport height): Represents a percentage of the viewport's height.

**Note:** The default height of the `<html>` and `<body>` elements in most modern web browsers is determined by the content they contain.

By default, the height of these elements will expand to accommodate their content. If there is no content, or if the content does not have a specific height defined, the height of these elements will be minimal.

However, in terms of CSS, the default height of the `<html>` and `<body>` elements is typically set to "auto", meaning their heights will adjust dynamically based on their content.

so generally we set the height of any content in the term of px or vh, but in order to make use of % we need to give the height of the `<html>` and `<body>` element as 100%

Example: to cover the entire viewport area:

```
html, body {  
    height: 100%;  
    margin: 0%;  
}
```

**The default width of the <html> and <body> elements in most modern web browsers is typically set to fill the entire width of the viewport.**

so for any other element we can define the width in term of % also.

**Example: height and width using %**

```
<!DOCTYPE html>  
  
<html lang="en">  
  
<head>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>Height in Percentage Example</title>  
    <style>  
        html, body {  
            height: 100%;  
            margin: 0;  
        }  
        .container {  
            height: 50%;  
            width: 50%;  
            /* 50% of the viewport height and width */  
            border: 1px solid black;  
            /* Just for demonstration */  
        }  
    </style>  
</head>  
</html>
```



```

        .inner-div {
            height: 50%;
            width: 50%;

            /* 50% of the height and width of its parent container */
            background-color: lightblue;

            /* Just for demonstration */
        }
    </style>
</head>

<body>

    <div class="container">

        <div class="inner-div"></div>

    </div>
</body>
</html>

```

Note: you can see the effect by minimizing the browser window size.

**Example height and width using px, vh, vw:**

```

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Height and Width Example</title>

```

```
<style>

  div {

    background-color: lightblue;

    border: 2px solid blue;

    margin: 20px;

  }

  /* Set height and width using different units */

  .box1 {

    height: 300px;

    /* 300 pixels */

    width: 700px;

    /* 700 pixels */

  }

  .box2 {

    height: 50vh;

    /* 50% of viewport height */

    width: 50vw;

    /* 50% of viewport width */

  }

</style>

</head>

<body>

  <div class="box1">Height: 300px, Width: 700px</div>

  <div class="box2">Height: 50vh, Width: 50vw</div>

</body>

</html>
```

**Note: you can see the effect by minimizing the browser window size.**

Example : **width, max-width, min-width**

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Height and Width Example</title>

    <style>

        .container {

            background-color: lightblue;

            border: 2px solid blue;

            /* width: 1500px; */

            /* here if we minimize the screen, the scroll bar will appear at
the bottom */

            /* width: 3000px; */

            /* here scroll bar will appear in the bottom */

            /* width: 100%; */

            /* it will take the 100% width of the entire viewport which is the
default behaviour, we can see the effect by minimizing the browser window */

            /* width: 50vw; */

            /* here content will be adjusted to 50% of the entire viewport */
```

```

        /* width: 50%; */

        /* here content will be adjusted to 50% of the entire body */

        max-width: 500px;

        /* if the content will increases still the max-width will be 500
        px */

        min-width: 800px;

        /* if we minimize the browser window till 800px the content will
        be adjusted, below that one scroll bar will appear */
    }
</style>
</head>
<body>

    <div class="container">

        Lorem ipsum dolor sit amet consectetur adipisicing elit. Quae facilis
        quisquam excepturi minus ea aspernatur

        dolorum? Quod, ad, natus pariatur aliquid debitis consequuntur
        repellat quibusdam vel dolore maiores dolor sit!

        Lorem ipsum dolor sit amet consectetur adipisicing elit. Autem numquam
        doloribus at esse dolor. Cum eos placeat,

        assumenda obcaecati ipsam distinctio cumque perferendis perspiciatis
        corrupti ad nemo dolorem necessitatibus.

        Distinctio.

    </div>
</body>
</html>

```

Example : **height, max-height, min-height**

if we take the height of a div 400px and if we put more content in that then the content will start overflowing from that div.

here if we don't take the height property then the height of the div will be automatically expanded according to the content.

but if we want that the height of the div will automatically expanded according to the content till the certain point only (after that text may overflow) then we can make use of **max-height** property.

**min-height** will represent the minimum initial height, and if we add the content then its size will automatically grow. (till the **max-height** point if it is defined).

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>

<style>

    div {

        border: 2px solid red;

        width: 50%;

        /* setting the initial height */

        min-height: 100px;

        /* setting the maximum height that can expand */

        max-height: 300px;

    }

</style>

<body>

    <div>
```

Lorem ipsum dolor sit amet consectetur adipisicing elit. Ipsum, corrupti? Ut vel ipsa beatae fuga animi fugit libero nam esse, pariatur cumque soluta amet cupiditate perferendis obcaecati itaque sunt voluptatem.

Lorem ipsum dolor sit amet consectetur adipisicing elit. Dolor pariatur, voluptatibus asperiores tempore error id rem quia blanditiis, repudiandae, voluptatum praesentium nesciunt corrupti ratione mollitia eos harum rerum aperiam minus.

```
</div>

</body>

</html>
```

Note: we can see the effect by adding some more content inside the `<div>` tag

Example: **em**, **rem**

```
<!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>

    /* by default the body and html font size is 16px */

    /* changing the font-size of the body from 16px to 32px */

    body {

      font-size: 32px;

    }

    #p1 {
```

```

        font-size: 2em;

        /* this size will be calculated based on its parent 2em = 32*2 px
*/
    }

    #p2 {

        font-size: 2rem;

        /* this size will be calculated based on the root 2rem = 16*2 px
*/

    }

</style>
</head>
<body>

    <p id="p1">Lorem ipsum dolor sit amet consectetur adipisicing elit.
Accusamus quo deserunt pariatur quibusdam

        voluptates voluptatum dolor repellat voluptatibus magnam. Nobis ad
omnis quo aspernatur eligendi numquam fugit,

        consequatur incidunt ratione?

    </p>

    <p id="p2">Lorem ipsum dolor sit amet consectetur adipisicing elit.
Accusamus quo deserunt pariatur quibusdam

        voluptates voluptatum dolor repellat voluptatibus magnam. Nobis ad
omnis quo aspernatur eligendi numquam fugit,

        consequatur incidunt ratione?

    </p>
</body>
</html>

```

## CSS Border:

CSS borders are essential elements in websites, representing the edges of various components and elements. CSS Borders refer to the lines that surround elements, defining

their edges. Borders can be styled, colored, and sized using CSS properties such as border style, border color, border width, and border-radius. borders can be styled with the top border, the right border, the bottom border, and the left border.

#### Border properties:

1. **border-style:** Specifies what kind of border to display.
  - none:** No border
  - dotted:** Defines a dotted border
  - dashed:** Defines a dashed border
  - solid:** Defines a solid border
2. **border-width:** Sets the width of the border.
3. **border-color:** Sets the color of the border.
4. **border-radius:** Creates rounded corners for the border.

#### Shortcut for applying border:

```
border: 10px solid green;
```

We can apply style for all 4 sides of the border. It works as top, right, bottom, left (clockwise).

Example:

```
border-style: solid solid dashed solid;
```

If we want to apply the color of the border for all 4 sides:

```
border-color: blue green pink yellow;
```

If we can apply the width for all the 4 sides:

```
border-width: 5px 10px 2px 7px;
```

We can apply styles for a border on all sides individually also.

```
border-top-style: solid;
```

```
border-top-width: 5px;
```

```
border-top-color: red;
```

To give the radius:

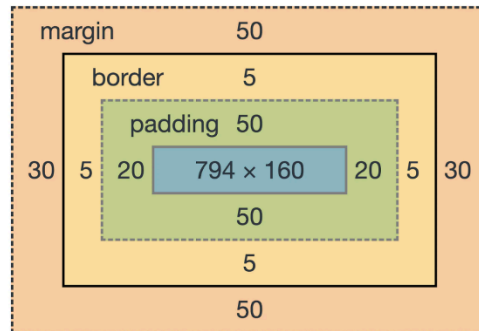
```
border-radius: 10px; //for all 4 side radius.
```

```
border-top-left-radius: 10px //for only the top left radius.
```

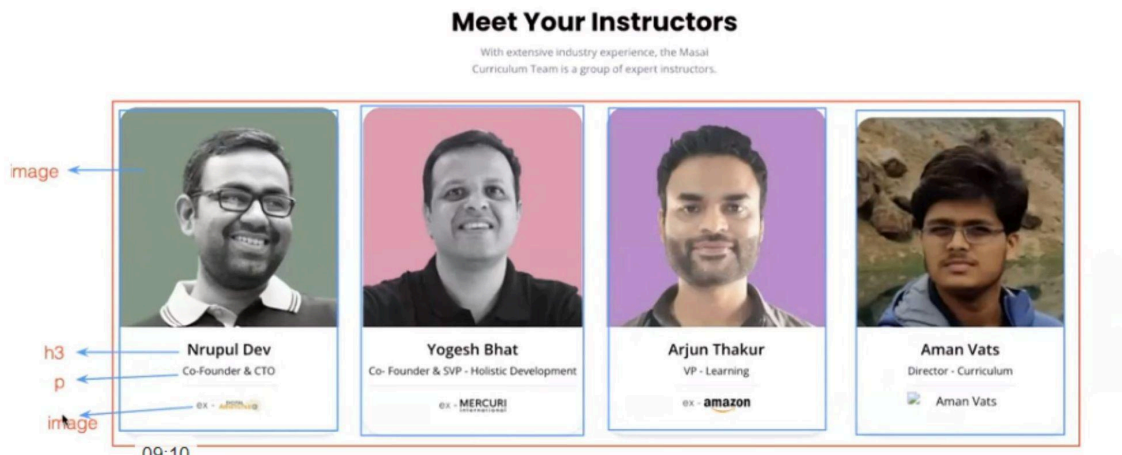


## CSS Box Model:

The CSS box model is a container that contains multiple properties including borders, margin, padding, and the content itself. It is used to create the design and layout of web pages.



**Visualizing the CSS box model:**



## Properties of the Box Model:

- **Content:** The content area consists of content like images, text, or other forms of media content. The height and width properties help to modify the box dimensions.
- **Padding:** The padding area is the space around the content area and within the border- box. It can be applied to all sides of the box or to the specific, selected side(s) -top, right, bottom, and/or left.
- **Margin:** Margin is a CSS property that defines the space around an HTML element. Margins ensure that the specified region around an element remains unoccupied by any neighboring element.

## CSS Padding:

The CSS padding properties define the space between the element border and the element content.

**padding: 25px;** It takes padding 25px from all 4 sides.

**padding: 25px 50px;** It takes 25px top and bottom and 50px left and right.

**padding: 25px 30px 40px;** It takes 25px top, 30px left and right 40px bottom.

`padding: 25px 30px 30px 40px;` It takes all 4 sides clockwise from top.

It is possible to set the padding for every side.

```
padding-top: 25px;
padding-bottom: 25px;
padding-right: 25px;
padding-left: 25px;
```

## CSS Margin:

- Margins are used to create space around elements, outside of any defined borders.

`margin: 25px;` It takes 25px margin from all 4 sides.

`margin: 25px 50px;` It takes 25px top and bottom and 50px left and right.

`margin: 25px 30px 40px;` It takes 25px top, 30px left and right 40px bottom.

`margin: 25px 30px 30px 40px;` It takes all 4 sides clockwise from top.

It is possible to set the margin for every side also.

```
margin-top: 25px;
margin-bottom: 25px;
margin-right: 25px;
margin-left: 25px;
```

**Note:** padding value should always be positive but margin value can be negative also, if we give the negative value then it will overlap in the other element.

Example:

```
<style>
    #p1 {
        border: 2px solid red;
        width: 400px;
        height: 200px;
    }

    #p2 {
        border: 2px solid green;
        width: 400px;
        height: 200px;
    }
```

```

        margin-top: -50px;
    }
</style>

<body>

    <p id="p1">paragraph1</p>
    <p id="p2">paragraph2</p>

</body>

```

### Margin-collapse:

It means the margin will collapse if we give the margin for one element from the bottom as 50px and the margin for the below element as 20px from the top then it will not take a total of 70px, It will take the max value i.e 50px only.

Example

```

<style>
    #p1 {
        border: 2px solid red;
        width: 400px;
        height: 200px;
        margin-bottom: 50px;
    }

    #p2 {
        border: 2px solid green;
        width: 400px;
        height: 200px;
        margin-top: 50px;
    }
</style>

<body>

    <p id="p1">paragraph1</p>
    <p id="p2">paragraph2</p>

```

</body>

## **Block, Inline, and Inline-block Elements:**

- **Inline Elements:** Inline elements do not start on a new line and only take up as much width as necessary. They typically flow within the content and do not create a new "block" of content.

**Examples:**

- **<span>**
- **<a>** (anchor)
- **<strong>** and **<em>** (text emphasis)
- **<img>** (image)
- **<br>** (line break)
- **<i>** (italic)
- **<b>** (bold)
- **<u>** (underline)
- **<small>** (small text)
- **<code>** (code)
- **<input>**

**Note:** In Inline elements **height** and **width** property will not work.

**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>
```

```

span {
    background-color: red;

    /* Here height and width property will not work */
    height: 200px;
    width: 200px;
}
</style>
</head>

<body>
    <span>Welcome</span>
</body>
</html>

```

- **Block Level Elements:** Block elements typically start on a new line and take up the full width available.

#### Examples:

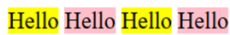
- **<div>**
- **<p>** (paragraph)
- **<h1>, <h2>, ..., <h6>** (headings)
- **<ul>, <ol>, <li>** (lists)
- **<table>, <tr>, <th>, <td>** (table elements)
- **<form>** (form element)
- **<hr>** (horizontal rule)
- **<header>, <footer>, <section>, <article>, <nav>, <aside>** (HTML5 structural elements)

- **Inline-Block Level Elements:** Inline-block elements combine aspects of both inline and block elements. They flow like inline elements but can have block-level properties like setting width and height.

**Example:**

```
<!-- Inline -->
<span style="background-color: yellow;">Hello</span>
<span style="background-color: pink;">Hello</span>

<span style="background-color: yellow; width: 100px; height: 100px;">Hello</span>
<span style="background-color: pink; width: 100px; height: 100px;">Hello</span>
```



```
<!-- Inline block -->
<span style="display: inline-block; background-color: yellow; width: 100px; height: 100px;">Hello</span>
<div style="display: inline-block; background-color: pink; width: 100px; height: 100px;">Hello</div>
```



**Note:** Using the **display** property we can make any block-level elements to the inline or inline-block element and any inline-level elements to the block or inline-block also.

**Example:**

Index.html:

```
<!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>
    h1 {
      text-align: center;
    }

    #parent {

      width: 100%;
      height: 200px;
      border: 2px solid;
      background-color: pink;
      text-align: center;
    }

    .child {
      display: inline-block;
      width: 25%;
      height: 200px;
      background-color: yellow;
      border: 1px solid blue;
    }
  </style>

</head>

<body>

  <h1>Welcome to Simple Webpage </h1>

  <div id="parent">

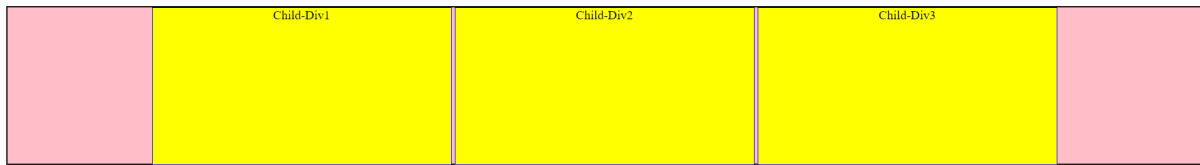
    <div class="child">Child-Div1</div>
    <div class="child">Child-Div2</div>
    <div class="child">Child-Div3</div>

  </div>

</body>
</html>
Output:
```



## Welcome to Simple Webpage



### Adding Google fonts in our webpage:

- Search the google font in google.
- Select the icon tab
- Copy the link for the static icon and paste it inside the <head>
- Insert the style related code inside the webpage.

to increase the size of the font:

set the font-size of the corresponding class inside the <style> tag.

### Applying the font:

- select the font tab
- choose the desired font
- click on Get font button
- click on Get Embed code
- follow the instructions

### CSS box-sizing property:

The box-sizing property defines how the width and height of an element are calculated: should they include padding and borders, or not?

**Values are:**

**content-box:** It is the default value. The width and height properties (and min/max properties) include only the content. Border and padding are not included

**border-box:** The width and height properties (and min/max properties) include content, padding and border.

Example:

```
<style>
#d1 {
  border: 2px solid red;
  width: 400px;
  height: 200px;
  /* padding: 50px; */
  box-sizing: border-box;
}
</style>
```

```
<div id="d1">
  <p>Lorem ipsum dolor, sit amet consectetur adipisicing elit. Libero soluta ut quibusdam
  tempora, quasi vero,
  cupiditate, quae voluptatem minima exercitationem maxime eveniet molestiae molestias
  inventore necessitatibus sint praesentium hic ipsam. </p>
</div>
```

Here if we don't use the property **box-sizing** then the width of the <div> tag will expand up to 400+50px.

## CSS Text Formatting Properties:

Property	Description	
color	Set the color of the text (use hex-code)	
text-align	Set the horizontal alignment of a text. values: left, right, justified or centered	
text-decoration	Short hand property for text-decoration-line, text-decoration-color and text-decoration-style	
text-transform	Specify uppercase and lowercase letters in a text. values: uppercase, lowercase, capitalize	
text-indent	Specify the indentation of the first line of a text (in px)	
letter-spacing	Specify gapping between two consecutive characters (in px)	
word-spacing	Specify gapping between two consecutive words (in px)	
line-height	Specify gapping between consecutive two lines (in %, default 100%)	
text-direction	Specify direction of text. Values: rtl,ltr	
white-space	Specifies how white-space inside an element is handled	
	Value	Description
	normal	Sequences of whitespace will collapse into a single whitespace. Text will wrap when necessary. This is default
	nowrap	Sequences of whitespace will collapse into a single whitespace. Text will never wrap to the next line. The text continues on the same line
	pre	Whitespace is preserved by the browser. Text will only wrap on line breaks. Acts like the <pre> tag in HTML
	pre-wrap	Whitespace is preserved by the browser. Text will wrap when necessary, and on line breaks

## CSS Font Properties:

Property	Description	
font-family	Set the font-family property. If the name of a font family is more than one word, it must be in quotation marks, like: "Times New Roman".	
font-style	Specify font-style	
	Value	Description
	normal	The text is shown normally
	italic	The text is shown in italics
font-size	Sets the size of the text. Default 16px = 1em	
font-weight	Sets how thick or thin characters in text should be displayed	
	Value	Description
	normal	Defines normal characters. This is default
	bold	Defines thick characters
font-variant	Sets the font variation	
	Value	Description
	normal	The browser displays a normal font. This is default
	small-caps	The browser displays a small-caps font. all lowercase letters are converted to uppercase letters. However, the converted uppercase letters appears in a smaller font size than the original uppercase letters in the text

## CSS list Properties:

Property	Description	
list-style-type	Specify type of list item marker some values are as follows	
	Value	Description
	disc	This is default value
	square	Use square symbol for list
	circle	Use circle symbol for list
	decimal	Use decimal numbers for list
	lower-alpha	Use lower case symbol for list
	upper-alpha	Use upper case symbol for list
list-style-image	Specify image as list item marker	

Example:

```
<!DOCTYPE html>

<html>

<head>
  <style>
    .a {
      list-style-type: circle;
    }

    .b {
      list-style-type: square;
    }

    .c {
      list-style-type: upper-roman;
    }

    .d {
      list-style-type: lower-alpha;
    }
  </style>
</head>

<body>

  <h2>The list-style-type Property</h2>

  <p>Example of unordered lists:</p>
  <ul class="a">
    <li>Coffee</li>
    <li>Tea</li>
    <li>Coca Cola</li>
  </ul>

  <ul class="b">
    <li>Coffee</li>
```

```

        <li>Tea</li>
        <li>Coca Cola</li>
    </ul>

    <p>Example of ordered lists:</p>
    <ol class="c">
        <li>Coffee</li>
        <li>Tea</li>
        <li>Coca Cola</li>
    </ol>

    <ol class="d">
        <li>Coffee</li>
        <li>Tea</li>
        <li>Coca Cola</li>
    </ol>

</body>

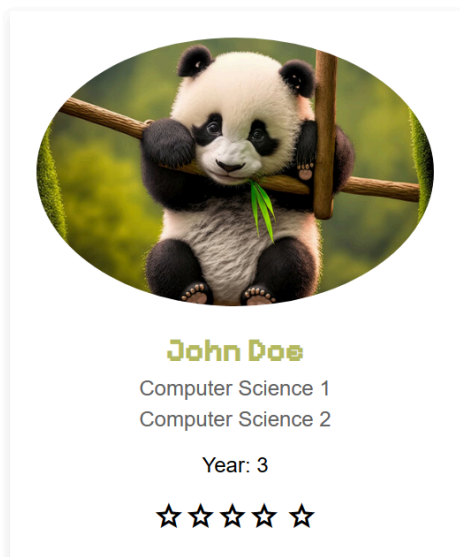
</html>

```

Experiment1: create the following layout using CSS:

## Title

### Panda Card



Experiment2: Create a rounded circle at the center of the webpage and apply a background image to that circle, image should fit properly inside that circle.

**Experiment3:** Create a rectangle box with 400\*400 px at the center of the webpage with background color aqua, and inside that box place another box with 100\*100 px at the center of the parent box with background color red.

**Experiment4:** create the following layout using CSS:



## CSS Combinators and pseudo-classes:

### CSS Combinators:

In CSS, combinators are used to specify relationships between elements in selectors. There are several types of combinators available:

1. **Descendant combinator (space):** Selects an element that is a descendant of another element. It selects all elements that are inside another specified element, no matter how deeply nested.

**Syntax:**

```
div p {  
    /* Selects all <p> elements that are descendants of <div> elements */  
}
```

**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>
/* It will target all the <p> which is the decendent of <div> tag */
```

```
    div p {
        color: red;
    }
</style>
```

```
</head>
```

```
<body>
```

```
    <div>
        <p>Lorem ipsum dolor sit amet.1</p>
        <div>
            <p>Lorem ipsum dolor sit amet.2</p>
            <div>
                <p>Lorem ipsum dolor sit amet.3</p>
            </div>
        </div>
    </div>
```

```
    <p>Lorem ipsum dolor sit amet.4</p>
```

```
</body>
```

```
</html>
```

2. **Child combinator (>):** Selects an element that is a direct child of another element. It selects only elements that are immediately nested inside the specified parent element.

**Syntax:**

```
div>p {
    /* Selects <p> elements that are direct children of <div> elements */
}
```

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>
/* It will target all the <p> which is the direct children of the <div> tag */
    .d1>p {
        color: red;
    }
    </style>

</head>

<body>

    <div class="d1">
        <p>Lorem ipsum dolor sit amet.1</p>
        <div>
            <p>Lorem ipsum dolor sit amet.2</p>
            <div>
                <p>Lorem ipsum dolor sit amet.3</p>
            </div>
        </div>
    </div>

    <p>Lorem ipsum dolor sit amet.4</p>

</body>

</html>

```

3. **Adjacent sibling combinator (+):** Selects an element that is immediately preceded by another element. It selects the first element immediately following another specified element.

#### Syntax:

```

h2+p {
/* Selects <p> elements that are immediately preceded by <h2> elements */
}

```

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>
  /* It will target all the <p> which is immediately preceded by the <div> tag */
  .d1+p {
    color: red;
  }
</style>

</head>

<body>

  <div class="d1">
    <p>Lorem ipsum dolor sit amet.1</p>
    <div>
      <p>Lorem ipsum dolor sit amet.2</p>
      <div>
        <p>Lorem ipsum dolor sit amet.3</p>
      </div>
    </div>
  </div>

  <p>Lorem ipsum dolor sit amet.4</p>
  <p>Lorem ipsum dolor sit amet.5</p>

</body>

</html>

```

4. **General sibling combinator (~):** Selects elements that are siblings of another element. It selects all elements that are siblings of a specified element and come after it in the HTML structure.

**Syntax:**

```

h2~p {
  /* Selects <p> elements that are siblings of <h2> elements */
}

```

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>
    /* It will target all the <p> which are the siblings of <div> tag */
    .d1~p {
      color: red;
    }
  </style>

</head>

<body>

  <div class="d1">
    <p>Lorem ipsum dolor sit amet.1</p>
    <div>
      <p>Lorem ipsum dolor sit amet.2</p>
      <div>
        <p>Lorem ipsum dolor sit amet.3</p>
      </div>
    </div>
  </div>

  <p>Lorem ipsum dolor sit amet.4</p>
  <p>Lorem ipsum dolor sit amet.5</p>

</body>

</html>

```

These combinators allow you to create more specific and targeted CSS selectors, making it easier to style elements based on their relationships with other elements in the HTML structure.

### CSS Positioning:

The **position** CSS property specifies how an element is positioned in a document. To position an element 'top, bottom, right, left' properties are used. Make sure to use the **position** property before using the **top, bottom, right & left**. Let us take a look at the positioning method:

## 1. position: static;

- HTML elements are positioned static by default.
- Static positioned elements are not affected by the top, bottom, left, and right properties.
- An element with `position: static;` is not positioned in any special way; it is always positioned according to the normal flow of the page:

### 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>
    .d1 {
      background-color: green;
      border: 2px solid red;
      width: 800px;
      height: 50px;
      position: static;
      /* not affected by top, right, left, bottom properties */
      /* top: 20px;
      right: 40px; */
    }
  </style>

</head>

<body>

  <h1>Welcome to Sage</h1>

  <p>
    An element with position: static; is not positioned in any special way; it is always
    positioned according to the
    normal flow of the page:
  </p>

  <div class="d1">
    This div element has position: static;
```

```
</div>

</body>

</html>
```

## 2. position: relative;

The element is positioned according to the normal flow of the document, and then offset *relative to itself* based on the values of **top, right, bottom, and left**. The offset does not affect the position of any other elements; thus, the space given for the element in the page layout is the same as if position were static.

It will move from its relative position(actual position).

## 3. position: absolute;

The element is removed from the normal document flow, and no space is created for the element in the page layout. The element is positioned relative to its closest positioned ancestor (if any) or to the initial containing block. Its final position is determined by the values of top, right, bottom, and left.

By default it will move the element with respect to its viewport, but if the parent element of this element is **absolute** then it will work **relative** according to its parent element.

Example1: **box with relative and absolute value**

```
<!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: 100px;

height: 100px;

border: 2px solid red;

background-color: aqua;

/\* it will move from its actual location and space created based on its movement will not be filled by other elements \*/

/\* position: relative; \*/

/\* It will move the element with respect to the viewport, element will come out from the document normal flow, and space created by its movement will be filled by other elements\*/

position: absolute;

top: 40px;

left: 40px;

}

</style>

</head>

<body>

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Accusantium qui dolor unde? Odio natus explicabo placea nam exercitationem iusto corrupti cupiditate consequuntur optio quasi iure dolor, sed cumque modi necessitatibus?</p>

<div class="box"></div>

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Accusantium qui dolor unde? Odio natus explicabo placeat nam exercitationem iusto corrupti cupiditate consequuntur optio quasi iure dolor, sed cumque modi necessitatibus?</p>

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Accusantium qui dolor unde? Odio natus explicabo placeat nam exercitationem iusto corrupti cupiditate consequuntur optio quasi iure dolor, sed cumque modi necessitatibus?</p>

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Accusantium qui dolor unde? Odio natus explicabo placeat nam exercitationem iusto corrupti cupiditate consequuntur optio quasi iure dolor, sed cumque modi necessitatibus?</p>

```
<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Accusantium qui dolor unde? Odio natus explicabo placeat nam exercitationem iusto corrupti cupiditate consequuntur optio quasi iure dolor, sed cumque modi necessitatibus?</p>
```

```
</body>
```

```
</html>
```

### Example2: box absolute with respect to its parent( if parent element position is relative)

In this case, the parent element acts as the reference point for positioning the child element. So if you move the parent element, the child element will move relative to it.

```
<!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 {
```

```
width: 400px;
```

```
height: 400px;
```

```
background-color: beige;
```

```
border: 2px solid black;
```

```
position: relative;
```

```
/* Now if we move the parent element then the child element will also move related to its parent; */
```

```
/* position: absolute; */
```

/\* it we make this element position to absolute then this element will come out from the normal document flow \*/

}

.box {

width: 100px;

height: 100px;

border: 2px solid red;

background-color: aqua;

/\* If the parent element position is relative then it will work respect to its parent \*/

position: absolute;

top: 40px;

left: 40px;

}

</style>

</head>

<body>

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Accusantium qui dolor unde? Odio natus explicabo placeat nam exercitationem iusto corrupti cupiditate consequuntur optio quasi iure dolor, sed cumque modi necessitatibus?</p>

<div class="container">

<div class="box"></div>

</div>

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Accusantium qui dolor unde? Odio natus explicabo placeat nam exercitationem iusto corrupti cupiditate consequuntur optio quasi iure dolor, sed cumque modi necessitatibus?</p>

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Accusantium qui dolor unde? Odio natus explicabo placeat nam exercitationem iusto corrupti cupiditate consequuntur optio quasi iure dolor, sed cumque modi necessitatibus?</p>

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Accusantium qui dolor unde? Odio natus explicabo placeat nam exercitationem iusto corrupti cupiditate consequuntur optio quasi iure dolor, sed cumque modi necessitatibus?</p>

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Accusantium qui dolor unde? Odio natus explicabo placeat nam exercitationem iusto corrupti cupiditate consequuntur optio quasi iure dolor, sed cumque modi necessitatibus?</p>

</body>

</html>

#### 4. position: fixed;

An element with **position: fixed;** is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.

**Example:** We can see the effect by adding some more content to the <p> tag and by scrolling the window.

```
<!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: 100px;
```



```

    height: 100px;

    border: 2px solid red;

    background-color: aqua;

    position: fixed;

    top: 50px;

    left: 50px;

}

/* to put the paragraph after the div */

p {

    margin-top: 200px;

}

</style>

</head>

<body>

    <div class="box"></div>

    <p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Cupiditate
    eum, illum autem quisquam quo quis? Amet quasi expedita numquam nulla. Labore
    corrupti ratione amet. Asperiores vero dignissimos minus commodi placeat.

</p>

</body>

</html>

```

## 5. position: sticky;

- An element with **position: sticky;** is positioned based on the user's scroll position.

- A sticky element toggles between **relative** and **fixed**, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed).

**Example:** Add some content(1000) to the last <p> tag and scroll the page to see the effect:

```
<!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: 100px;
      height: 100px;
      border: 2px solid red;
      background-color: aqua;
      position: sticky;
      top: 20px;
      /* after adding the Lorem1000 content to the last <p>,then while
scrolling the page after reaching the 20px from the top the <div> will stick
*/
    }
  </style>
</head>

<body>

  <p>
    Lorem ipsum dolor sit amet consectetur adipisicing elit. Ab nesciunt
    laboriosam, molestiae eius, nostrum reprehenderit architecto saepe
    maiores consectetur sint vero porro ratione odit rem repellat nam
    placeat iusto voluptatem.

    Lorem, ipsum dolor sit amet consectetur adipisicing elit. Temporibus
    debitis harum rem tempora voluptate itaque beatae nobis magni minima,
```

commodi aspernatur fuga molestiae quae quo, eaque similique quaerat laudantium ab.

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Cumque, et velit accusantium ipsam deserunt rerum voluptates illum iure autem unde temporibus repellendus? Non impedit nobis natus vero, neque accusantium dicta.

</p>

<div class="box"></div>

<p>

Lorem ipsum, dolor sit amet consectetur adipisicing elit. Cupiditate eum, illum autem quisquam quo quis? Ametquasi expedita numquam nulla. Labore corrupti ratione amet. Asperiores vero dignissimos minus commodi placeat.

Lorem1000

</p>

</body>

</html>

## CSS Overflow:

The CSS **overflow** property controls what happens to content that is too big to fit into an area.

The **overflow** property specifies whether to clip the content or to add scrollbars when the content of an element is too big to fit in the specified area.

The **overflow** property has the following values:

- **visible** - Default. The overflow is not clipped. The content renders outside the element's box
- **hidden** - The overflow is clipped, and the rest of the content will be invisible
- **scroll** - The overflow is clipped, and a scrollbar is added to see the rest of the content
- **auto** - Similar to **scroll**, but it adds scrollbars only when necessary

**Note: The overflow property only works for block elements with a specified height.**

**Example:** Add some more content inside the <p> tag to see the effect.

```
<!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>
    p {
      border: 2px solid red;
      width: 500px;
      height: 200px;
      overflow: auto;
    }
  </style>
</head>

<body>

  <p>
    Lorem ipsum dolor sit amet, consectetur adipisicing elit. Id alias
    officiis voluptates soluta iusto! Voluptatum eligendi numquam corrupti
    animi, molestias beatae qui quod impedit voluptatem laborum officiis,
    magni nam sit.
  </p>

</body>

</html>
```

**CSS Float:**

The CSS **float** property specifies how an element should float.

The **float** property is used for positioning and formatting content e.g. let an image float left to the text in a container

.

The element is removed from the normal flow of the page, though still remaining part of it.

**The **float** property can have one of the following values:**

- **left** - The element floats to the left of its container
- **right** - The element floats to the right of its container
- **none** - The element does not float (will be displayed just where it occurs in the text). This is default
- **inherit** - The element inherits the float value of its parent

### Example

```
<!DOCTYPE html>
<html>

<head>
  <style>
    /* It will float the image to the left side so that remaining space
    will be accupied by the next block level element */
    img {
      float: left;
    }
  </style>
</head>

<body>
  <h2>Float Right</h2>

  <p>
    In this example, the image will float to the right in the paragraph,
    and the text in the paragraph will wrap around the image.
  </p>

  <div>
    
```

```
<p>
  Lorem ipsum dolor, sit amet consectetur adipisicing elit. Harum,
  consequatur et delectus voluptatibus animi molestiae quidem cumque
  corrupti eos nihil quibusdam nemo facere ullam ea fuga nisi. Neque,
  eaque cum. Lorem500
</p>

</div>

</body>

</html>
```

### **Example: Float Next to each other**

```
<!DOCTYPE html>
<html>

<head>
  <style>
    div {
      float: left;
      height: 200px;
      width: 200px;
      margin: 20px;
    }

    .div1 {
      background: red;
    }

    .div2 {
      background: yellow;
    }

    .div3 {
      background: green;
```

```
    }  
  </style>  
</head>
```

```
<body>
```

```
  <h2>Float Next To Each Other</h2>
```

```
  <p>In this example, the three divs will float next to each other.</p>
```

```
  <div class="div1">Div 1</div>  
  <div class="div2">Div 2</div>  
  <div class="div3">Div 3</div>
```

```
</body>
```

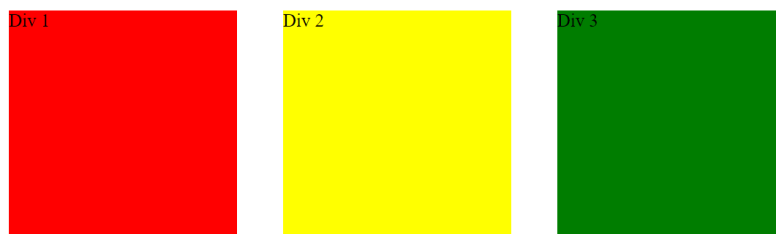
```
</html>
```

## **Output:**

### **Float Next To Each Other**

Getting Started  
<https://www.nx>

In this example, the three divs will float next to each other.



**Example:** float with clear property

```
<!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>
    #p1 {
      border: 2px solid red;
      background-color: aqua;
      width: 500px;
      /* This element will move out from the normal document flow and
float to the right side */
      float: left;
    }

    #p2 {
      border: 2px solid green;
      background-color: beige;
      width: 500px;
      /* This element will move out from the normal document flow and
float to the left side */
      float: right;
    }

    #p3 {
      border: 2px solid blue;
      background-color: violet;
      /* This p3 element will come after p1 and p2 and clear floats on
both sides */
      /* To see the effect of clear: left or clear: right, add more
content inside p1 or p2 */
      clear: both;
    }
  </style>

</head>

<body>
```



```
<p id="p1">Lorem ipsum dolor sit amet consectetur, adipisicing elit.  
Doloremque reprehenderit voluptatum illo  
    blanditiis soluta sapiente voluptas sint quia est cumque excepturi  
quibusdam sit accusamus explicabo velit  
    reiciendis, dolorum sed quod.  
</p>  
  
<p id="p2">Lorem ipsum dolor sit amet consectetur adipisicing elit.  
Commodi in consequatur et quaerat dolore!  
    Excepturi at ratione suscipit, sit voluptate magnam, perferendis non  
quisquam cumque voluptates beatae sapiente  
    nobis vel.  
</p>  
  
<p id="p3">Lorem ipsum dolor sit amet consectetur, adipisicing elit. Hic,  
itaque architecto alias, ipsam assumenda  
    animi neque magni maxime vel est rerum dolores ipsum eveniet modi quam  
quod error suscipit. Nulla?  
</p>  
  
</body>  
  
</html>
```

### **CSS pseudo-elements:**

A CSS pseudo-element is used to style specified parts of an element.

For example, it can be used to:

- Style the first letter, or line, of an element
- Insert content before, or after, the content of an element

Syntax:

```
selector::pseudo-element {  
    property: value;  
}
```

pseudo-elements	Example	Description
::first-letter	p::first-letter	Selects the first letter of every <p> element
::first-line	p::first-line	Selects the first line of every <p> element
::selection	::selection	Selects the portion of an element that is selected by a user
::after	p::after	Insert content after every <p> element
::before	p::before	Insert content before every <p> element

#### Example:

```
<!DOCTYPE html>
<html>

<head>
  <style>
    ::selection {
      color: red;
      background: yellow;
    }

    h3::before {
      content: "(Added using Before)";
    }

    h3::after {
      content: "(Added after)";
    }

    h1::first-letter {
      color: green;
    }

    p::first-line {
      color: aquamarine;
      font-size: 20px;
    }
  </style>
</head>

<body>

  <h1>Select some text on this page:</h1>

  <h3>This is a paragraph.</h3>
  <p>
    Lorem ipsum dolor sit amet consectetur adipisicing elit. Excepturi
    facilis nobis aperiam facere rem eligendi
```

```
        minima blanditiis consequatur nostrum possimus, nulla obcaecati
        accusantium sequi provident molestias amet nam.
        Earum, fugiat.
    </p>

</body>

</html>
```

### CSS pseudo-classes:

A pseudo-class is used to define a special state of an element.

For example, it can be used to:

- Style an element when a user mouses over it
- Style visited and unvisited links differently
- Style an element when it gets focus

Syntax:

```
selector:pseudo-class {
    property: value;
}
```

Take a look at some important pseudo classes.

pseudo-class	Example	Description
:active	a:active	Selects the active link
:link	a:link	Selects all unvisited links
:hover	a:hover	Selects links on mouse over
:visited	a:visited	Selects all visited links
:checked	input:checked	Selects every checked <input> element
:disabled	input:disabled	Selects every disabled <input> element
:enabled	input:enabled	Selects every enabled <input> element
:empty	p:empty	Selects every <p> element that has no children (including text nodes)
:first-child	p:first-child	Selects every <p> element that is the first child of its parent
:last-child	p:last-child	Selects every <p> element that is the last child of its parent
:first-of-type	p:first-of-type	Selects every <p> element that is the first <p> element of its parent

:last-of-type	p:last-of-type	Selects every <p> element that is the last <p> element of its parent
:focus	input:focus	Selects the input element which has focus
:read-only	input:read-only	Selects input elements with the "readonly" attribute specified
:in-range	input:in-range	Selects input elements with a value within a specified range
:invalid	input:invalid	Selects all input elements with an invalid value
:nth-child(n)	p:nth-child(2)	Selects every <p> element that is the second child of its parent

**Example:** Links can be displayed in different ways:

**Note:** `a:hover` MUST come after `a:link` and `a:visited` in the CSS definition in order to be effective! `a:active` MUST come after `a:hover`.

If we use them individually then order does not matter.

```
<!DOCTYPE html>
<html>

<head>
  <style>
    /* unvisited link */
    a:link {
      color: red;
    }

    /* visited link */
    a:visited {
      color: green;
    }

    /* mouse over link */
    a:hover {
      color: hotpink;
    }

    /* selected link */
    a:active {
      color: blue;
    }
  </style>
</head>

<body>
```

```

<h2>Styling a link depending on state</h2>

<a href="sucess.html" target="_blank">This is a link</a>

<p>
  <b>Note:</b> a:hover MUST come after a:link and a:visited in the CSS
  definition in order to be effective.
</p>
<p>
  <b>Note:</b> a:active MUST come after a:hover in the CSS definition in
  order to be effective.
</p>
</body>
</html>

```

**Note:** if the normal link color red not showing, it might be due the browser cache. clear the browser cache and check it.

### Combining the pseudo-classes with the combinators:

**Example:** Hover over a <div> element to show a <p> element (like a tooltip):

```

<!DOCTYPE html>
<html>

<head>
  <style>
    p {
      display: none;
      background-color: yellow;
      padding: 20px;
    }

    /* This selector, combining the above two parts, targets <p> elements that are
    descendants of <div> elements currently being hovered over. */

    div:hover p {
      display: block;
    }
  </style>

```

```
</head>

<body>

    <div>Hover over this div element to show the p element
        <p>Tada! Here I am!</p>
    </div>

</body>

</html>
```

Pseudo-classes can be combined with HTML classes:

**Example:**

```
<!DOCTYPE html>
<html>

<head>
    <style>
        p.p1:hover {
            background-color: yellow;
            padding: 20px;
        }
    </style>
</head>

<body>

    <p class="p1">This is the p1</p>

    <p>This is the p1</p>

</body>

</html>
```

**Example:** first-child, last-child and nth-child

The **first-child** CSS pseudo-class represents the first element among a group of sibling elements, Whereas The **last-child** CSS pseudo-class represents the last element among a group of sibling elements.

The **nth-child** represents which number of elements among a group of sibling elements.

```
<!DOCTYPE html>
<html>

<head>
  <style>

    div p:first-child{
      color: red;
      font-size: 20px;
      font-family: 'Times New Roman';
    }

    div p:last-child{
      color: green;
      font-size: 20px;
      font-family: 'Times New Roman';
    }

    div p:nth-child(3){
      color: violet;
      font-size: 20px;
      font-family: 'Times New Roman';
    }

  </style>
</head>

<body>
<div>
  <p>This is the Paragraph 1</p>
  <p>This is the Paragraph 2</p>
  <p>This is the Paragraph 3</p>
  <p>This is the Paragraph 4</p>
  <p>This is the Paragraph 5</p>
</div>
</body>

</html>
```

**Student Activity:** Creating the following layout:



### Solution: Using table

```
<!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>
    body {
      margin: 0px;
      padding: 0px;
    }

    table {
      width: 100%;
      height: 100vh;
    }
  </style>
</head>

<table>
  <tr>
    <th>Header</th>
  </tr>
  <tr>
    <td>
      <div>SideBar</div>
      <div>main</div>
    </td>
  </tr>
  <tr>
    <th>footer</th>
  </tr>
</table>
```



```
#header {  
    height: 10vh;  
    background-color: aqua;  
  
}  
  
#footer {  
    height: 10vh;  
    background-color: green;  
  
}  
  
#side {  
    width: 20%;  
    background-color: blueviolet;  
  
}  
  
td {  
    text-align: center;  
}  
  
#main {  
    width: 80%;  
    background-color: antiquewhite;  
  
}  
</style>
```

</head>

<body>

```
<table border="2px">  
  
    <tr id="header">  
        <td colspan="2">Header</td>  
    </tr>  
  
    <tr id="center">  
        <td id="side">SideBar</td>  
        <td id="main">main</td>
```

```
        </tr>
        <tr id="footer">
            <td colspan="2">footer</td>

        </tr>

    </table>

</body>

</html>
```

### Solution: Using div and float:

```
<!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>
        body {
            margin: 0px;
            padding: 0px;
        }

        .header {
            height: 10vh;
            background-color: aqua;
            text-align: center;
        }

        .side {
            text-align: center;
            width: 20%;
            background-color: blueviolet;
            height: 80vh;
```

```
        float: left;
    }

    .main {
        height: 80vh;
        background-color: antiquewhite;
        text-align: center;
    }

    .footer {
        height: 10vh;
        background-color: green;
        text-align: center;
    }
</style>
```

</head>

<body>

```
<div class="header">Header</div>
<div class="side">Sidebar</div>
<div class="main">Main</div>
<div class="footer">Footer</div>
```

<!-- Using semantic tags, here we need not take the class selector, we can target them by using their name -->

```
<!--
<header>Header</header>
<aside>Sidebar</aside>
<main>Main</main>
<footer>Footer</footer> -->
```

</body>

</html>