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.

Specificity Value:

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

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 behaviour, 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.

Example1 :

```
<!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 {
            background-image: url(./images/f12.jpg);
            background-repeat: no-repeat;
            background-attachment: fixed;
```

```
        }
    </style>

</head>

<body>

    <p>lorem5000</p>

</body>

</html>
```

5. **background-size:** Controls the size of the background image.

   The background-size property is specified in one of the following ways:

   - Using the keyword values **contain** or **cover** or **auto**.
   - Using a **width** value only, in which case the height defaults to auto.
   - Using both a **width** and a **height** value, in which case the first sets the width and the second sets the height. Each value can be a <length>(px), a <percentage>, or auto.

   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.

Example1 :

```
<!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 {
            background-image: url(./images/fl2.jpg);
            background-repeat: no-repeat;
            background-size: cover;
            /* background-size: 500px; */
            /* background-size: 400px 200px; */

        }
    </style>

</head>

<body>
</body>

</html>
```

6. **background-position:** This property sets the starting position of a background image. Default value: left top (It works with the x (horizontal)and y (vertical) 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

Note: If you only specify one keyword, the other value will be "center".

Note: to position the image properly, make use of **background-attachment: fixed;** without background-attachment: fixed, the background image can move along with the content as you scroll, causing the background-position to behave differently than expected.

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. And The left top is 0% 0% and the right bottom is 100% 100%.

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

Example:

```
<style>

    body {
        background-image: url(./images/nature.jpg);
        background-repeat: no-repeat;
        background-position: bottom right;
        background-attachment: fixed;

    }

</style>
```

Example: Creating a box of 300 * 300 and fitting a background image to cover the entire box:

```
<style>
    div {
        height: 300px;
```

```
                width: 300px;
                background-image: url(./images/sh1.jpg);
                background-position: center;
                background-size: cover;
                background-repeat: no-repeat;
            }
        </style>
    </head>
    <body>
        <div></div>
    </body>
```

Background shorthand:

```
        body {
            background: lightblue url("img_tree.gif") no-repeat
fixed center;
        }
```

CSS height and width:

- The default **height** for all the HTML elements is **auto,** which is the minimal height, and it will be increased based on its content. If we add the content then it will be increased by default.
- Whereas **width** of the HTML element will be dependent upon the type of the elements. In case of **block-level elements** (e.g , <p>, <h1>, <header>), default width will be the entire width of the page and for the **inline elements** (e.g <a>, <img>,) the width will be determined by its content.
-
- 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 explicitly. Its value can be set using **px, %,** or **auto.**

- 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.

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** (centimetres)**, mm** (millimetres)**:** 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.

Min-Height and Max-Height: These properties allow you to set a range for an element's height. min-height ensures the element does not shrink below a specified value, while max-height limits how tall the element can become.

Min-Width and Max-Width: Similarly, min-width and max-width constrain the width of an element.

Overflow: If an element's content exceeds its set width or height, the overflow property determines how the extra content is handled (visible, hidden, scroll, or auto).

Example1: width with px, %, vw

```
<!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 {
      background-color: gold;
      border: 5px solid black;
    }

    #p1 {
      /* here if we minimize the screen below 800px width, the scroll bar
will        appear at    the bottom  */
      width: 800px;
    }

    #p2 {
      width: 50%;
```

```
        /* 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: 100%; */
      }

      #p3 {
        /* here content will be adjusted to 50% of the entire viewport */
        width: 50vw;
      }
    </style>

</head>
<body>
  <p id="p1"> width using 800px </p>
  <p id="p2"> width using 50% </p>
  <p id="p3"> width using 50vw </p>
</body>

</html>
```

Difference between vw and % for width:

1. **Reference:**

%: The percentage % width is always relative to the width of the parent container. For example, if a parent container is 1000px wide, a width of 50% will result in 500px.

vw: It is relative to the viewport's width (entire browser window).

2. **Responsiveness:**

   %: Resizes based on the parent container's size.

   vw: Resizes based on the browser window's size, independent of the parent.

3. **Use Case**:

   %: Ideal when you need the element to adjust according to its container.

   vw: Best when the element needs to respond directly to the viewport size, providing consistency across different containers.

4. **Behavior**:

   %: Changes when the parent container's size changes.

   vw: Changes when the browser window is resized, regardless of the parent container's size.

Example2: vw and %

```
<!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>
    /* The container div has a width of 50% relative to the browser window or parent container */
    #box {
      width: 50%;
```

```
      border: 5px solid green;
    }

    /* The first paragraph (#p1) has a width of 40% relative to the container
(#box) */
    #p1 {
      width: 40%;
      background-color: red;
    }

    /* The second paragraph (#p2) has a width of 40vw, which is 40% of the
viewport width, independent of its parent container */
    #p2 {
      width: 40vw;
      background-color: aqua;
    }
  </style>

</head>

<body>

  <!-- This is the container div, which takes 50% of the browser window width -->
  <div id="box">

    <!-- This paragraph's width is 40% of the container (#box) -->
    <p id="p1">Using %</p>
```

```
    <!-- This paragraph's width is 40% of the viewport width, independent of the
container -->
    <p id="p2">Using vw</p>

  </div>

</body>

</html>
```

Example3: min-width and max-width:

- **width:** sets a specific width for an element.
- **max-width:** ensures that the element doesn't exceed a specified width, even if its container allows it to grow larger.
- **min-width:** ensures that the element doesn't shrink smaller than a specified width, even if its container becomes smaller.

```
<!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 with fixed width of 400px */
    #box1 {
      width: 400px;
      background-color: lightcoral;
      border: 2px solid black;
    }

    /* Box with a flexible width that has a max-width of 600px */
    #box2 {
      max-width: 800px;
      background-color: lightblue;
      border: 2px solid black;
    }

    /* Box with a flexible width but can't be smaller than 300px */
    #box3 {
      min-width: 800px;
      background-color: lightgreen;
      border: 2px solid black;
    }
  </style>

</head>
```

```
<body>


  <!-- Fixed width: the box will always be 400px wide, regardless of the viewport
size -->
  <div id="box1">
    <p>Box with fixed width: 400px</p>
  </div>


  <!-- Max-width: the box will expand to fill the container, but will not exceed
800px -->
  <div id="box2">
    <p>Box with max-width: 800px,
      Lorem ipsum dolor sit amet consectetur adipisicing elit. Magni repellendus,
porro laboriosam dicta, dolores deleniti inventore illum mollitia accusantium
obcaecati impedit eum nam facilis. Labore, ex accusamus dolore reprehenderit
tempora, dicta neque commodi et debitis qui eos delectus voluptate ab
exercitationem incidunt praesentium dolorem maxime suscipit voluptatibus
laboriosam quod veritatis.
    </p>
  </div>


  <!-- Min-width: the box will shrink, but not below 800px -->
  <div id="box3">
    <p>Box with min-width: 800px, can't shrink below the minimum
      Lorem ipsum dolor sit amet consectetur adipisicing elit. Hic quae quia
maiores exercitationem porro iusto tempore nam tempora magni dolorem saepe
voluptatibus, quibusdam aperiam sequi voluptatum in dignissimos distinctio fugiat
error totam minus quaerat quam. Excepturi, consectetur laudantium. Unde dolorem
eum necessitatibus numquam eos. Alias consequuntur eum modi excepturi voluptas.
    </p>
```

```
  </div>

</body>

</html>
```

Example4: Height with % and vh

- As we know that all the HTML elements have the minimal height and it will increase based on the content. In order to give height in % we must give the initial height to the parent container.

```
<!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>
    html, body {
      height: 500px;
    }

    #p1 {
      border: 2px solid;
```

```
        background-color: aquamarine;
        height: 50%;
      }

      #p2 {
        border: 2px solid;
        background-color: red;
        height: 50vh;
      }
    </style>

  </head>

  <body>

    <p id="p1">Using the %</p>
    <p id="p2">Using the vh</p>
  </body>
  </html>
```

Note: we can see the difference by commenting the html, body height.

Example 5:

```
<!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 {
      height: 400px;
      border: 2px solid;
      background-color: antiquewhite;
    }

    #p1 {
      /* Here the height will be 25% of parent box which is 400px */
      height: 25%;
      border: 2px solid;
      background-color: aqua;
    }

    #p2 {
      /* Here the height will be 25vh with respect to the viewport */
      height: 25vh;
      border: 2px solid;
      background-color: aqua;
    }
  </style>
```

```
</head>


<body>


  <div id="box">

    <p id="p1">Inner paragraph using % </p>

    <p id="p2">Inner paragraph using vh </p>

  </div>


</body>

</html>
```

Example6: min-height and max-height

min-height: Ensures that the element's height is not less than a specified value, but it can grow larger if needed. It sets the initial height of an element.

max-height: Limits the element's height to a maximum value; beyond this, the element will not grow taller, and the content will either overflow or become scrollable if overflow is set.

```
<!DOCTYPE html>

<html lang="en">


<head>

  <meta charset="UTF-8">

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

```
  <title>Min-Height and Max-Height Example</title>

  <style>
    /* Box with a min-height: it cannot be smaller than 200px */
    #box1 {
      background-color: lightgreen;
      min-height: 200px; /* Ensures the height is at least 200px */
      border: 2px solid black;
    }

    /* Box with a max-height: it cannot be taller than 300px */
    #box2 {
      background-color: lightcoral;
      max-height: 300px; /* Ensures the height doesn't exceed 300px */
      border: 2px solid black;
    }
  </style>
</head>

<body>

  <!-- Box with min-height: height will grow if content is larger, but won't be smaller than 200px -->
  <div id="box1">
    <p>This box has a min-height of 200px. Even if there's less content, the height will not shrink below 200px.</p>
  </div>
```

```
  <br>

  <!-- Box with max-height: height will be capped at 300px, content will be scrollable if it exceeds the max height -->
  <div id="box2">

    <p>Lorem ipsum dolor sit amet consectetur, adipisicing elit. Tenetur, blanditiis necessitatibus eum voluptate rerum eaque aut fugit totam maiores! Ipsa, maxime omnis. Voluptates, dolor libero deserunt id qui numquam vel unde sint labore aspernatur optio, quas sed fugiat voluptate architecto vero odit ipsum doloribus reprehenderit assumenda temporibus placeat. Et, praesentium.</p>

  </div>

</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>
```

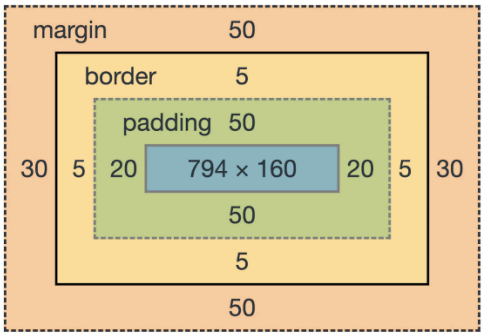
Example: em, rem

```
<!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 */

        }
```

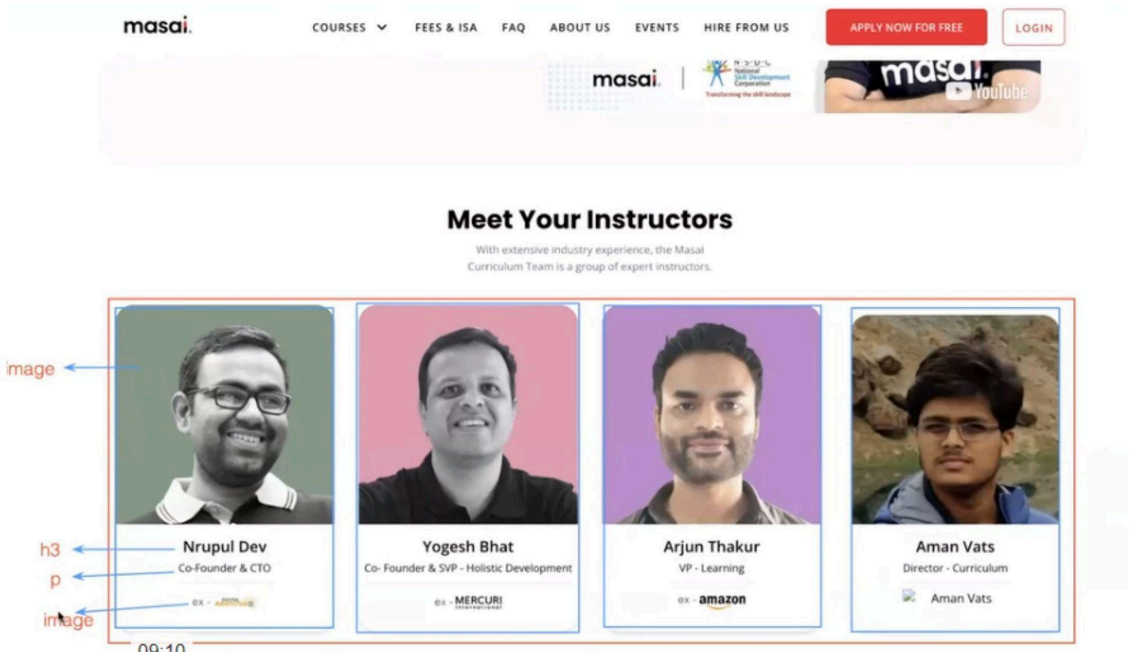
```
    </style>
</head>
<body>
    <p id="p1">Lorem ipsum dolor sit amet consectetur adipisicing elit. Accusamus quo deserunt pariatur quibusdam
        voluptates voluptatum dolor repellat voluptatibus magnam. Nobis ad omnis quo aspernatur eligendi numquam fugit,
        consequatur incidunt ratione?
    </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 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 neighbouring element.
- **Border -** A border that goes around the padding and content
- **Margin -** Clears an area outside the border. The margin is transparent

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 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-top: 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-top: 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:
    - ****
    - **<a>** (anchor)
    - **** and **** (text emphasis)
    - **<img>** (image)
    - **
** (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
<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:

  - ** **
  - **<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)

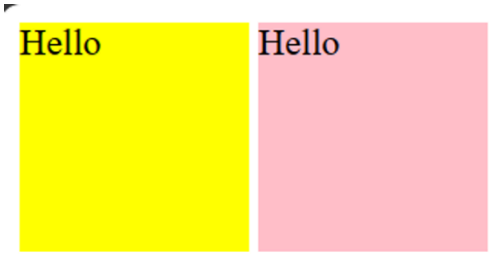
- **Inline-Block Level Elements:** Inline-block elements combine aspects of both inline and block elements. They flow like inline elements but can have block-level properties like setting width and height.

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>
```

Hello Hello Hello Hello

```
<!-- 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: