Project – 5

CSS Basics

Submitted By:

Nishigandha Patil

INDEX

INTRODUCTION TO CSS

➢ What is CSS?

- CSS stands for **Cascading Style Sheets**. It is a language we use to style a Web page.
- CSS defines how HTML elements are displayed on a web page.
- It's responsible for the visual aspects like layout, colors, fonts, spacing, and more. By using CSS, you can control how elements on a webpage are displayed on different devices and screen sizes.

➢ Syntax:

- CSS consists of selectors and declarations.
- The whole structure is called a ruleset.

1. **Selector:** Selectors target HTML elements (in this example, <p> element is the selector)
2. **Declaration:** Written in a **'property: value;'** format. (e.g., 'color: red;')
3. **Properties:** Control various styling aspects like color, font, margin, padding, etc.
4. **Property value:** Values define how the properties should be applied (e.g., color: **red**; , font-size: **16px**;).

Important parts of the syntax:

- Each ruleset must be wrapped in curly braces ({}).
- Within each declaration, you must use a colon (**:**) to separate the property from its value or values.
- Within each ruleset, you must use a semicolon (**;**) to separate each declaration from the next one.

Saving a CSS File: Enter a file name with the **.css** extension. For example, **'styles.css'**.

➢ Types of CSS:

1. Inline CSS:

- Implemented directly within HTML tags using the 'style' attribute.
- Overrides external and internal CSS.
- Not recommended for large-scale use due to reduced maintainability.

```
<body>
    <p style="background-color: yellow; color: red; font-size: 16px;">This is a
    paragraph with inline CSS.</p>
</body>
```

This is a paragraph with inline CSS.

2. **Internal CSS:**
   - Written within the **<style>** tag in the HTML <head> section.
   - Applies styles to the entire document or specific sections.
   - Overrides external CSS but can be overridden by inline CSS.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Internal CSS</title>
<style>
    p {
        color: red;
        font-size: 18px;
    }
</style>
</head>
<body>
    <p>This is a paragraph with internal CSS.</p>
</body>
</html>
```

This is a paragraph with internal CSS.

3. **External CSS:**
   - Defined in a separate **.css** file (e.g. styles.css)
   - Linked to HTML documents using the **<link>** tag in the <head> section.
   - Overrides inline and internal CSS.

HTML File:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0">
    <link rel="stylesheet" href="styles.css">
    <title>External CSS</title>
</head>
<body>
    <p>This is a paragraph with External CSS</p>
</form>
</body>
</html>
```

CSS File (styles.css):

```css
CSS Practice > # styles.css > ...
1  p {
2    color: blue;
3    background-color: pink;
4  }
```

This is a paragraph with External CSS

➢ Types of CSS Selector:

Selectors are used to target HTML elements that you want to style. They can target elements, classes, IDs, attributes, etc.

1. **Element Selector:** Selects HTML elements by their name (e.g., p targets all <p> paragraphs).

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0">
    <title>Element Selector</title>
<style>
  h1{
    font-style: italic;
  }
  p {
  color: blue;
  font-family: 'Courier New', Courier, monospace;
  }
  div {
  background-color: pink;
  }
</style>
</head>
<body>
  <h1>Welcome</h1>
  <p>This is a paragraph.</p>
  <div>This is a division.</div>
</form>
</body>
</html>
```

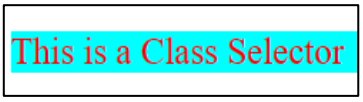
Welcome

This is a paragraph.

This is a division.

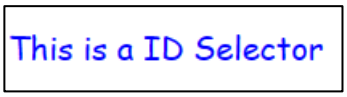
2. **Class Selector:** Targets elements with a specific class. Defined using a **dot (.)** followed by the class name.
(e.g., **.box** targets elements with class="box").

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="styles.css">
    <title>Class Selector</title>
<style>
    .box{
        background-color: aqua;
        color: red;
    }
</style>
</head>
<body>
    <p class="box">This is a Class Selector</p>
</body>
</html>
```



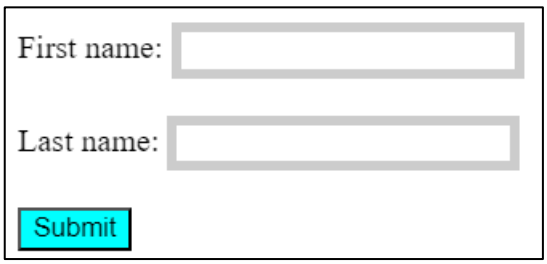
3. **ID Selector:** Selects a single element based on its unique ID attribute.
Defined using a **hash (#)** followed by the ID name.
(e.g., #first targets an element with id="first").

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="styles.css">
    <title>ID Selector</title>
<style>
    #first{
        color: blue;
        font-family:cursive;
    }
</style>
</head>
<body>
    <p id="first">This is a ID Selector</p>
</body>
</html>
```



4. **Attribute Selector:** Selects elements based on their attributes.
e.g., [type="text"] selects input elements with type="text").

```html
    <title>Attribute Selector</title>
<style>
  input[type="text"] {
  border: 5px solid #ccc;
  }
  input[type="submit"]{
    background-color: aqua;
  }
</style>
</head>
<body>
  <form>
    <label for="fname">First name:</label>
    <input type="text" id="fname" name="fname"><br><br>
    <label for="lname">Last name:</label>
    <input type="text" id="lname" name="lname"><br><br>
    <input type="submit" value="Submit">
  </form>
</form>
</body>
</html>
```



5. **Universal selector:** Defined using a *. It selects all elements in an HTML document.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
    content="width=device-width,
    initial-scale=1.0">
    <link rel="stylesheet"
    href="styles.css">
    <title>Pseudo-element</title>
</head>
<body>
  <h1>Welcome</h1>
  <p>This is a paragraph with some
  content.</p>
</body>
</html>
```

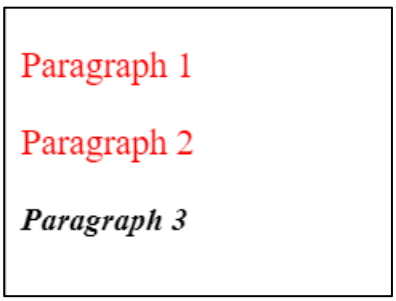
```css
* {
  margin: 0;
  padding: 0;
  font-family: 'Lucida Sans', 'Lucida Sans
  Regular', 'Lucida Grande', 'Lucida Sans
  Unicode', Geneva, Verdana, sans-serif;
}

p{
  color: red;
}
```

6. **Descendant Selector:** Allows you to style elements that are nested inside other elements in the HTML structure

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0">
    <title>Element Selector</title>
<style>
  .container p {
  color: red;
  font-size: 16px;
}
</style>
</head>
<body>
  <div class="container">
    <h2>Welcome</h2>
    <p>This is a paragraph inside the container.</p>
    <ul>
      <li>List item 1</li>
      <li>List item 2</li>
    </ul>
  </div>
</form>
</body>
</html>
```

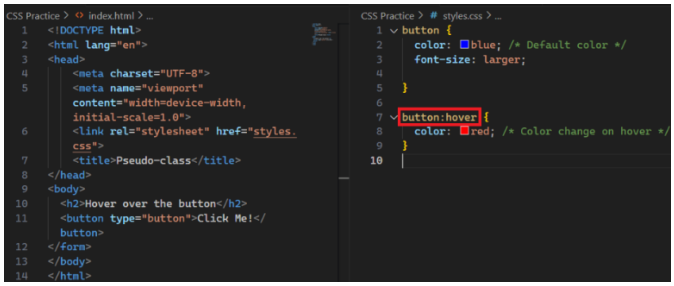
7. **Combinators:** Specify relationships between selectors to target specific elements. Includes the descendant combinator ( ), child combinator (>), adjacent sibling combinator (+), and general sibling combinator (~).

```html
<style>
    div p{
    /* Styles applied to all <p> elements that are
    descendants of <div> elements */
        color: red;
    }

    div > p{
    /* Styles applied to all <p> elements that are direct
    children of <div> elements */
       font-size: large;
    }

    div + p{
    /* Styles applied to <p> elements that directly follow
    <div> elements */
        font-style: italic;
    }

    div ~ p{
    /* Styles applied to all <p> elements that follow <div>
    elements */
        font-weight: bold;
    }
</style>
</head>
<body>
    <div>
        <p>Paragraph 1</p>
        <p>Paragraph 2</p>
    </div>
    <p>Paragraph 3</p>
</body>
</html>
```



8. **Pseudo Classes:** Selects elements based on their state or position.
Begins with a **colon (:)** followed by a pseudo-class keyword.

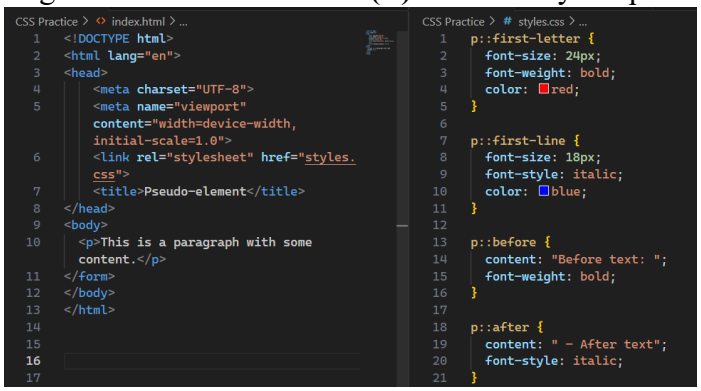
Sr. no	Pseudo classes	Description
1	: hover	Applies styles when the mouse is over an element.
2	: active	Applies styles to an element being activated by the user (usually when the mouse button is pressed down).
3	: focus	Applies styles to an element that currently has focus (e.g., an input field).
4	: nth-child(n)	Selects the nth child of its parent.
5	: nth-of-type(n)	Selects the nth child of its type within its parent.
6	: first-child	Selects the first child of its parent.
7	: last-child	Selects the last child of its parent.

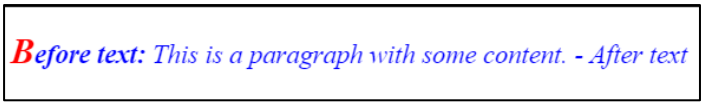


Before Hover:

After hover:

9. **Pseudo Elements:** Targets specific parts of elements, like the first letter or line.
Begins with a **double colon (::)** followed by the pseudo-element name.





- **Comments in CSS:** Comments are enclosed between /* and */ and can span across multiple lines.

```
div p{
    color: red;  /* setting font color red */
}
```

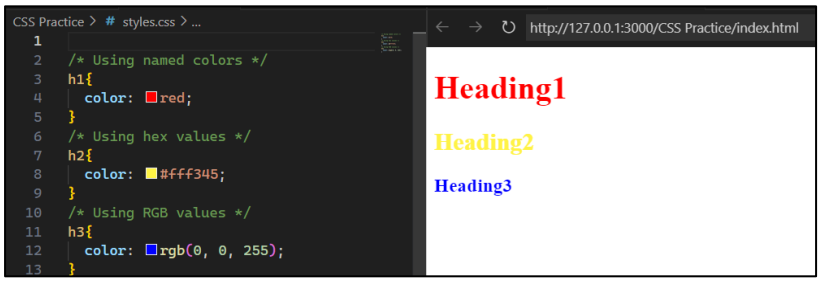
COLOR AND BACKGROUND

➢ Types of color Values:

- **Named Colors:** These are pre-defined color names, such as "red," "blue," "green," etc
- **RGB:** It uses three values (0-255) representing the intensity of each color channel. For example, rgb(255, 0, 0) represents pure red.
- **Hex:** Represented by a hash symbol followed by a combination of six characters/digits (0-9 and A-F). For example, #FF0000 represents pure red.
- **HSL (Hue, Saturation, Lightness):** Represents colors using three values - hue (0-360), saturation (0-100%), and lightness (0-100%)

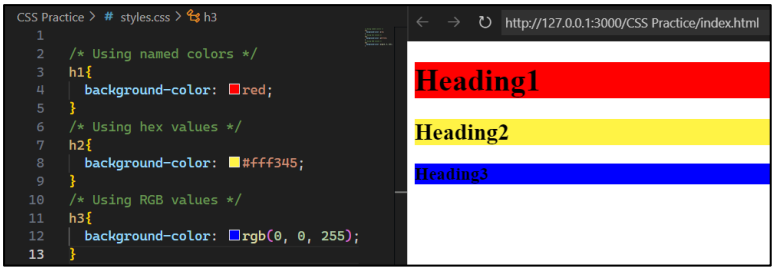
You can style the color and background of HTML elements using various properties. Here are some common properties and their usage:

➢ Color: Sets the text color of an element.

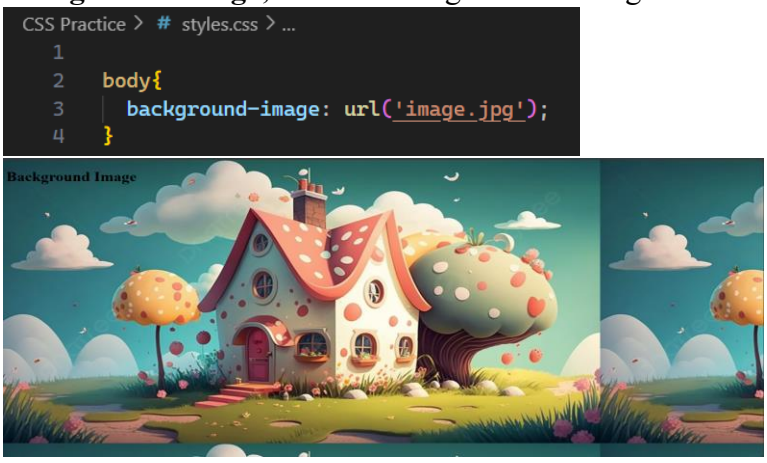


➢ Background Properties:

1. **background-color; -** Sets the background color of an element.



2. **background-image; -** Sets an image as the background of an element.



3. **background-repeat:** Specifies how a background image should be repeated.

Sr. no	background-repeat values	Description
1	background-repeat: repeat;	Tiles the background image both horizontally and vertically to fill the element's content area.
2	background-repeat: repeat-x;	Repeat background in horizontal direction (x-axix).
3	background-repeat: repeat-y;	Repeat background in vertical direction (y-axix).
4	background-repeat: no-repeat;	It's used to specify that a background image should not be repeated.

Example: background-repeat: repeat;

```
body{
  background-image: url('image.jpg');
  background-repeat: repeat;
}
```

background-repeat: no-repeat;

```
body{
  background-image: url('image.jpg');
  background-repeat: no-repeat;
}
```

background-repeat: repeat-y;

4. **background-size:**

Syntax: background-size: cover | contain | auto |;

Sr. no	background-size values	Description
1	background-size: auto;	The default value. The background image retains its original size.
2	background-size: cover;	Resizes the background image to cover the entire container, maintaining aspect ratio. Some parts of the image might be cropped.
3	background-size: contain;	Resizes the background image to fit within the container without exceeding its dimensions. Empty space may appear if aspect ratios differ.
4	Using specific dimensions: E.g. background-size:50%;	Sets the width and height of the background image using specific length values, such as pixels or percentages

Example: background-size:cover;

```
body{
  background-image: url('image.jpg');
  background-size:cover;
}
```

Using Specific Dimensions:

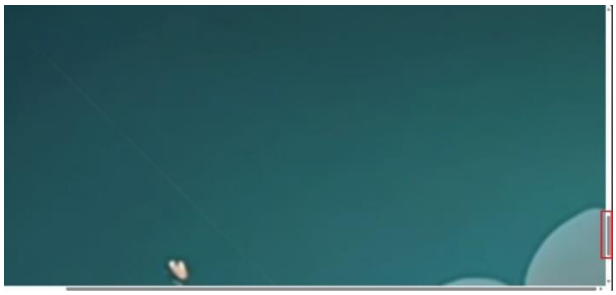
```
body{
  background-image: url('image.jpg');
  background-size:50%;
}
```

5. **background-attachment:**

Sr.no	background-attachment values	Description
1	background-attachment: scroll;	This is the default value. The background image scrolls along with the content when the user scrolls through the webpage.
2	background-attachment: fixed;	Image doesn't move when the content is scrolled, creating a "fixed" background effect.
3	background-attachment: local;	This value is less commonly used and is not fully supported across all browsers. It's similar to scroll, but the background scrolls along with the element's contents rather than the entire page.

Example: background-attachment:scroll;

background-attachment:fixed;



6. **background-position:** Controls the starting position of a background image within its container.
   Keyword Values: These include keywords like ‘top’, ‘bottom’, ‘left’, ‘right’, and ‘center’, used to position the background image accordingly.

```
body{
  background-image: url('image.jpg');
  background-position:center;
}
```

Percentage Values: These values represent the percentage distance from the top and left edges of the container. For instance, **background-position: 50% 25%;** would position the image 50% from the top and 25% from the left of the container.

```
body{
  background-image: url('image.jpg');
  background-position:50% 25%;
}
```

➢ **Background shorthand:**

Syntax:

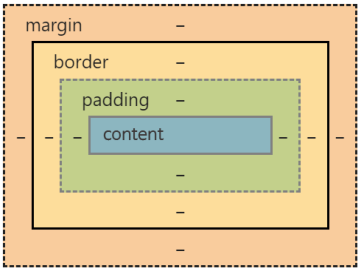
background: [background-color] [background-image] [background-repeat] [background-attachment] [background-position] / [background-size];

However, not all values are required, and they can be specified in any order.

```
body{
  width: 330px;
  height: 900px;
  background: red url('image.jpg') no-repeat scroll right top;
}
```

BOX MODEL

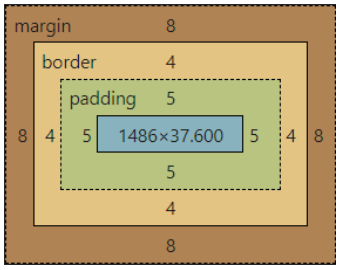
The CSS box model is a fundamental concept that defines how elements are rendered in a browser. It consists of four main components:



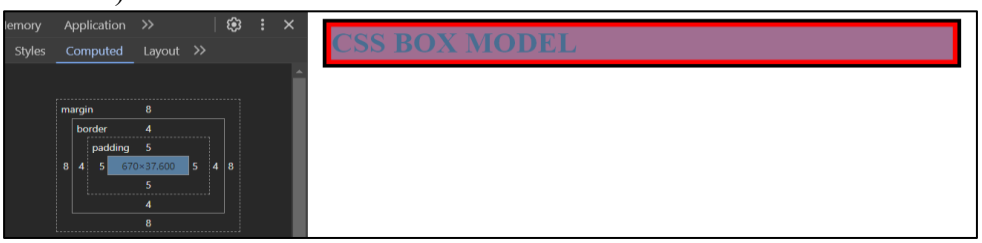
Example: Setting padding, border, margin to <h1> element.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0">
    <link rel="stylesheet" href="styles.css">
    <title>Box model</title>
</head>
<body>
    <h1>CSS BOX MODEL</h1>
</body>
</html>
```

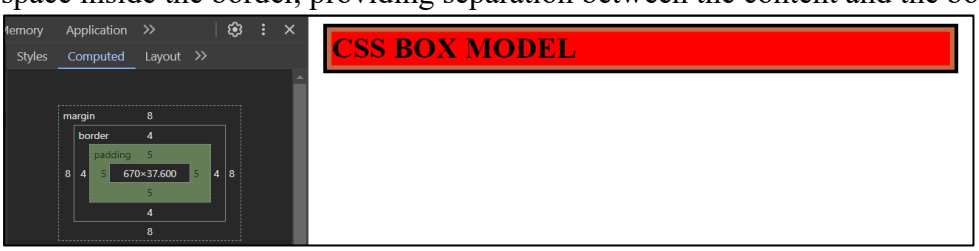
```css
h1{
  background-color: red;
  margin: 8px;
  border-width: 4px;
  border-style:solid;
  border-color: black;
  padding: 5px;
}
```



1. **Content:** The actual content of the HTML element, such as text, images, or other media, which is surrounded by padding, border, and margin. (eg., CSS BOX MODEL)



2. **Padding:** The space between the content and the element's border. It helps create space inside the border, providing separation between the content and the border.(5px)



We can also set individual padding like this (same for margin):

```css
h1{
  padding-top: 2px;
  padding-right: 3px;
  padding-bottom: 2px;
  padding-left: 3px;
}
```

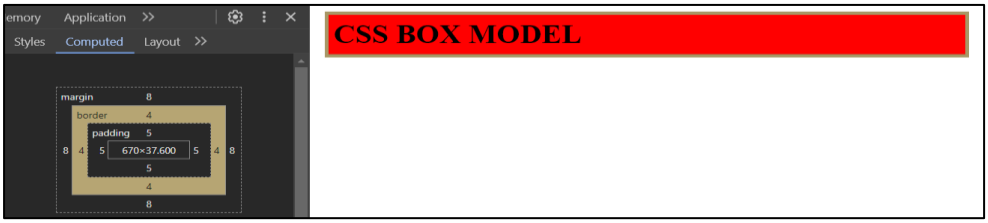
Margin with 2 values:

```css
h1{
 padding:2px 3px ; /* top-bottom left-right */
}
```

padding Shorthand: [padding-top] [padding-right] padding-bottom] padding-left];

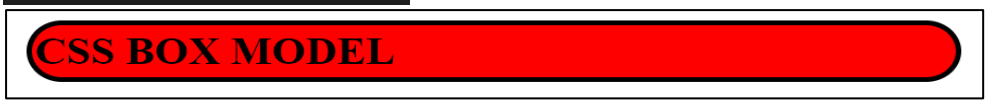
```css
h1{
  padding:2px 3px 2px 3px; /* top right bottom left */
}
```

3. **Border:** A line that surrounds the padding and content. It defines the visible boundary of the element.



Border Radius: Creates rounded corners on elements.

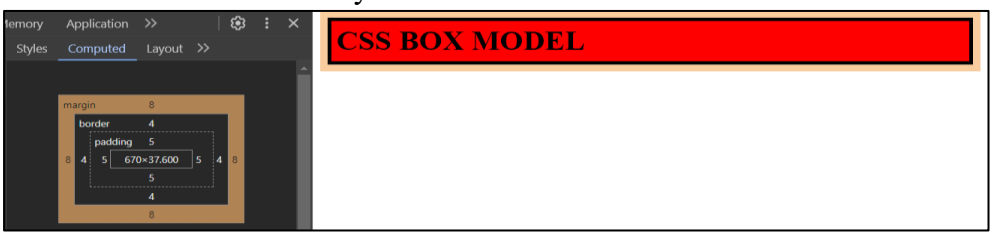
```css
border-radius:40px;
```



Border Shorthand: [border-width] [border-style] [border-color];

```css
h1{
  border: 2px double black; /* width style color */
}
```

4. **Margin:** The space outside the element's border. It creates space between the element and other elements in the layout.



We can also set individual margin like this:

```css
h1{
  margin-top: 2px;
  margin-right: 3px;
  margin-bottom: 2px;
  margin-left: 3px;
}
```

Margin with 2 values:

```css
h1{
  margin:2px 3px;  /* top-bottom left-right */
}
```

Margin Shorthand: [margin-top] [margin-right] [margin-bottom] [margin-left];

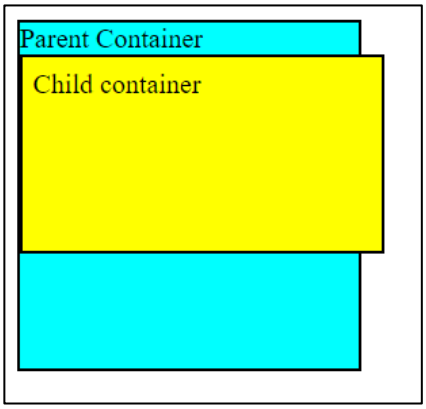
```css
h1{
  margin:2px 3px 2px 3px; /* top right bottom left */
}
```

➢ Box Sizing:

The box-sizing property in CSS controls how the width and height of an element are calculated, taking into account padding and borders. It has two main values:

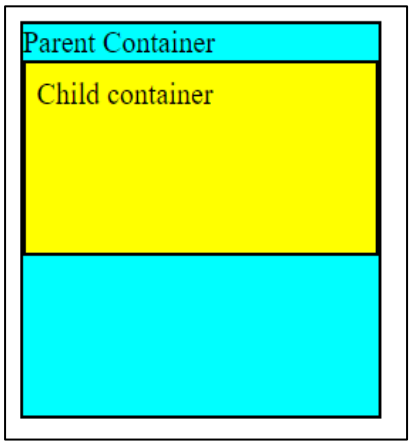
1. **content-box:** This is the default value. When you set a width or height for an element (width: 200px;, for example), any padding or border added to the element will increase its overall size. In other words, the width and height you set will apply only to the content area, and padding and border will be added to that size.

```css
.parent-container{
  background-color: aqua;
  height: 200px;
  width:200px;
  border: 2px solid black;
}

.child-container{
  box-sizing: content-box;
  background-color: yellow;
  height: 100px;
  width:100%;
  border: 2px solid black;
  padding:6px
}
```



2. **border-box:** With this value, the width and height you set (width: 200px;) include padding and borders. The padding and border widths are subtracted from the specified width and height, ensuring that the content area remains the specified size.

```css
.parent-container{
  background-color: aqua;
  height: 200px;
  width:200px;
  border: 2px solid black;
}

.child-container{
  box-sizing: border-box;
  background-color: yellow;
  height: 100px;
  width:100%;
  border: 2px solid black;
  padding:6px;
}
```



FONTS AND DISPLAY

➢ Fonts:

You can control fonts and text display in various ways. Properties of Font are as follows:

```css
p{
 background-color: pink;
 font-family:'Tilt Prism', cursive;
 font-size: 20px;
 font-style: italic;
 font-weight: bold;
}
```

Lorem ipsum dolor sit amet consectetur, adipisicing elit. Voluptates saepe nihil sunt expedita cum sed dolorum ea vitae aliquam aliquid quisquam ad minima libero dolore consectetur dignissimos obcaecati totam porro, consequuntur nam maxime magnam!

Text Display Properties:

```css
p{
  background-color: pink;
  text-align: center;
  text-transform:uppercase;
  text-decoration: underline;
  line-height: 2;
}
```

LOREM IPSUM DOLOR SIT AMET CONSECTETUR, ADIPISICING ELIT. VOLUPTATES SAEPE NIHIL SUNT EXPEDITA CUM SED DOLORUM EA VITAE ALIQUAM ALIQUID QUISQUAM AD MINIMA LIBERO DOLORE CONSECTETUR DIGNISSIMOS OBCAECATI TOTAM PORRO, CONSEQUUNTUR NAM MAXIME MAGNAM!

Google Fonts (External Fonts):

Import external fonts from Google Fonts or other sources using the @import rule or a <link> tag in HTML.

```css
@import url('https://fonts.googleapis.com/css2?family=Lora&family=Tektur:wght@500&display=swap');

p{
  background-color: pink;
  font-family: 'Lora', serif;
  font-family: 'Tektur', sans-serif;
}
```

Using <link> tag:

```html
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?family=Lora&family=Tektur:wght@500&display=swap" rel="stylesheet">
```

➢ Display:

Determines how an HTML element behaves in terms of its layout and rendering on the webpage.

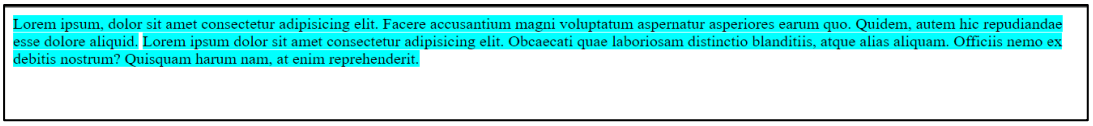
1. **display: inline;**

   Occupy only necessary width. Don't start on a new line; they flow within the content.

   Setting height and width is not allowed.

   You can set left-right margin and padding (Top and bottom margin-padding is not applicable).

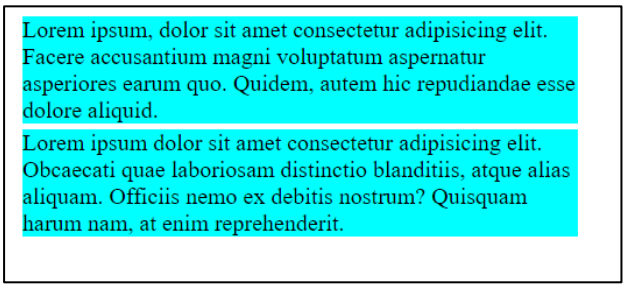
```css
p{
  display: inline;
  background-color: aqua;
}
```



2. **display: block;**

   Take up the full width available. Leaves a newline before and after the element.

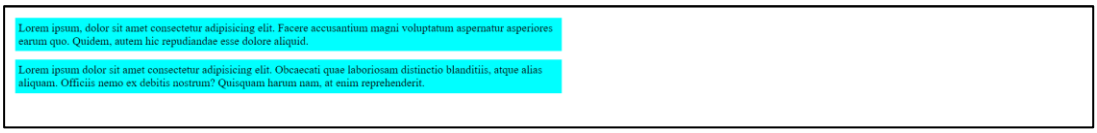
```css
p{
  display: block;
  background-color: aqua;
  margin: 4px;
  width: 50%;
  height: 30%;
}
```



3. **display: inline-block;**

   Similar to inline elements but allow for setting width, height, padding, and margins. Elements can sit next to each other.

```css
p{
  display: inline-block;
  background-color: aqua;
  width: 50%;
  height: 50%;
  margin: 6px;
  padding: 5px;
}
```



4. **display: none;**
   Completely hides the element from the page.
   It's as if the element doesn't exist in the document.

```css
h1{
 display: none;
}
p{
  display: block;
  background-color: aqua;
  width: 50%;
  height: 50%;
  margin: 6px;
  padding: 5px;
}
```

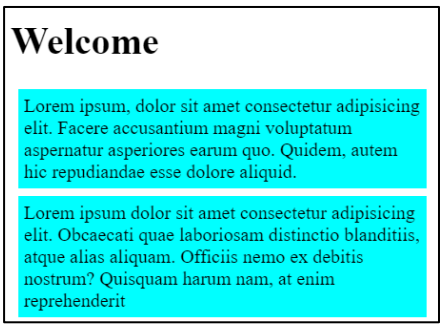
Lorem ipsum, dolor sit amet consectetur adipisicing elit. Facere accusantium magni voluptatum aspernatur asperiores earum quo. Quidem, autem hic repudiandae esse dolore aliquid.

Lorem ipsum dolor sit amet consectetur adipisicing elit. Obcaecati quae laboriosam distinctio blanditiis, atque alias aliquam. Officiis nemo ex debitis nostrum? Quisquam harum nam, at enim reprehenderit

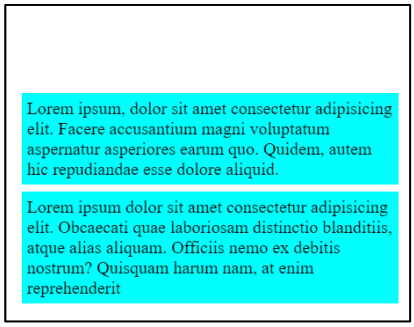
5. **visibility: hidden;** Element is hidden but its space is reserved.

```css
h1{
  visibility: hidden;
}
p{
  display: block;
  background-color: aqua;
  width: 50%;
  height: 50%;
  margin: 6px;
  padding: 5px;
}
```

Before:



After visibility:hidden;

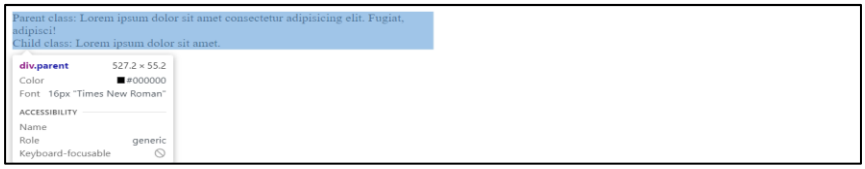


SIZES, POSITION AND LISTS

➢ Sizes:

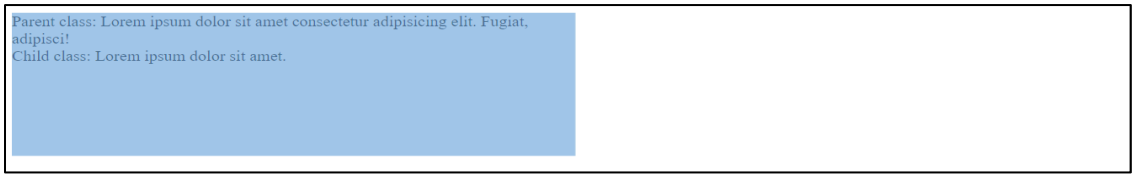
1. **Pixels (px):** It's a fixed-size unit that doesn't change with screen density. For example, width: 200px;
2. **Percentage (%):** Relative to the parent element. '50%' means half the size of the parent.

```
.parent{
 width:50%
}
```



3. **Viewport Width (vw) and Viewport Height (vh):** Relative to the size of the browser window. '50vw' means half of the window's width.

```
.parent{
 width:50vw;
 height: 50vh;
}
```



4. **Em (em):** Relative to the font size of the element's parent. '2em' would be twice the size of the parent's font.

```
.parent{
   font-size: 20px;
}
.child{
   font-size: 2em;
}
```

Parent class: Lorem ipsum dolor sit amet consectetur adipisicing elit. Fugiat, adipisci!
Child class: Lorem ipsum dolor sit amet.

5. **Rem (root em):** Similar to 'em', but it's based on the root element's font size ('html'). '2rem' means twice the size of the root font.

```
div{
   font-size: 2rem;
}
```

Parent class: Lorem ipsum dolor sit amet consectetur adipisicing elit. Fugiat, adipisci!
Child class: Lorem ipsum dolor sit amet.

6. **Viewport-Percentage Lengths (vmin, vmax):** Relative to the viewport's smallest ('vmin') or largest ('vmax') dimension. (eg width:50vmin;)

7. **Absolute Lengths (in, cm, mm, pt, pc):** Physical measurements like inches, centimeters, etc. '2in' means 2 inches.

8. **Flexible Lengths (fr):** Used in CSS Grid Layout to distribute space within a grid container based on fractions.

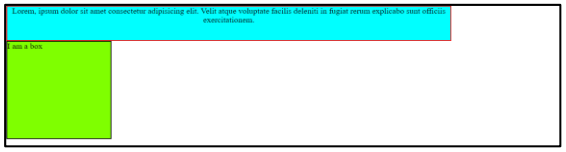
➢ The Position Property:

The position property in CSS determines how an element is positioned within its parent or the document flow.

1. **position: static; -** Default value. Elements are positioned according to the normal document flow.

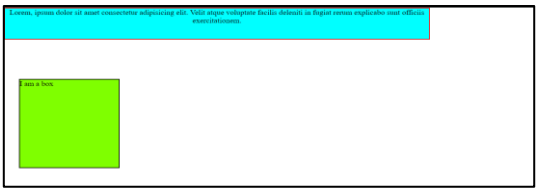
   The top, right, bottom, left, and z-index properties don't affect static-positioned elements.

```css
.container{
    background-color: aqua;
    text-align: center;
    height: 10vh;
    width: 80vw;
    border: 2px solid red;

}
.box{
    height: 200px;
    width: 200px;
    background-color: chartreuse;
    border: 2px solid black;
    position:static;
    top: 90px;
    right: 10px;
    left: 30px;
}
```



2. **position: relative; -** Positioned relative to its normal position. Element can be shifted using the top, right, bottom, and left properties. It will remain static if these properties are not applied. If applied it just moves from its original position.

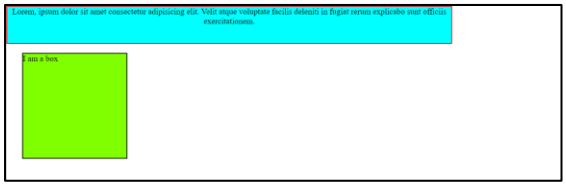
```css
.container{
    background-color: aqua;
    text-align: center;
    height: 10vh;
    width: 80vw;
    border: 2px solid red;

}
.box{
    height: 200px;
    width: 200px;
    background-color: chartreuse;
    border: 2px solid black;
    position:relative;
    top: 90px;
    right: 10px;
    left: 30px;
}
```



3. **position: absolute;**

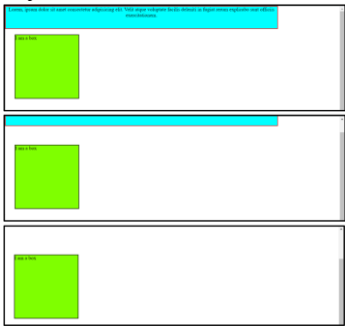
   An element positioned with respect to its parent. If no parent is present, then it uses the document body as parent. The element is taken out of the normal flow, and its position is determined by the top, right, bottom, or left properties.

```css
.container{
    background-color: aqua;
    text-align: center;
    height: 10vh;
    width: 80vw;
    border: 2px solid red;

}
.box{
    height: 200px;
    width: 200px;
    background-color: chartreuse;
    border: 2px solid black;
    position:absolute;
    top: 90px;
    right: 10px;
    left: 30px;
}
```



4. **position: fixed; -** Similar to absolute positioning, but it's positioned relative to the browser window, so it stays fixed even when the user scrolls the page.

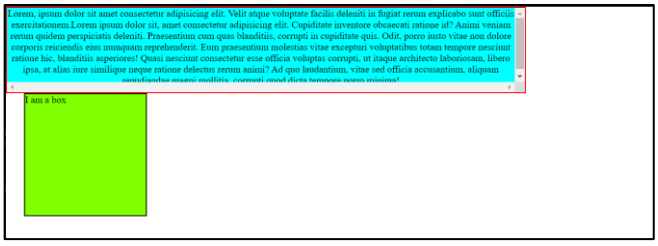
```css
.container{
    background-color: aqua;
    text-align: center;
    height: 10vh;
    width: 80vw;
    border: 2px solid red;

}
.box{
    height: 200px;
    width: 200px;
    background-color: chartreuse;
    border: 2px solid black;
    position:fixed;
    top: 90px;
    right: 10px;
    left: 30px;
}
```



5. **position: sticky; -**

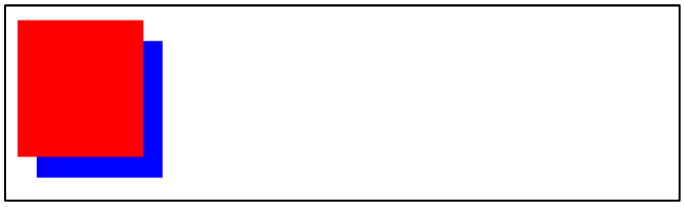
Acts like *relative* within its container until a given scroll point, then becomes fixed for the rest of the scroll. Sticky element "sticks" to its nearest ancestor that has a "scrolling mechanism" (created when overflow is hidden, scroll, auto, or overlay).

```css
.container{
    background-color: aqua;
    text-align: center;
    height: 20vh;
    width: 80vw;
    border: 2px solid red;
    overflow: scroll;

}
.box{
    height: 200px;
    width: 200px;
    background-color: chartreuse;
    border: 2px solid black;
    position:sticky;
    top: 90px;
    right: 10px;
    left: 30px;
}
```



- **z-index:** The z-index property in CSS specifies the stack and order of an element. An element with greater stack order is always in front of element with a lower stack order.

```html
<style>
  .red-box {
    width: 200px;
    height: 200px;
    background-color: red;
    position: absolute;
    top: 20px;
    left: 20px;
    z-index: 2; /* Higher z-index */
  }

  .blue-box {
    width: 200px;
    height: 200px;
    background-color: blue;
    position: absolute;
    top: 50px;
    left: 50px;
    z-index: 1; /* Lower z-index */
  }
</style>
</head>
<body>
<div class="red-box"></div>
<div class="blue-box"></div>
</body>
</html>
```



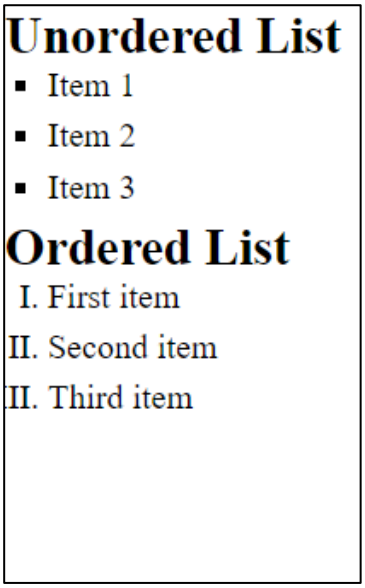
```html
<style>
  .red-box {
    width: 200px;
    height: 200px;
    background-color: red;
    position: absolute;
    top: 20px;
    left: 20px;
    z-index: 2; /* lower z-index */
  }

  .blue-box {
    width: 200px;
    height: 200px;
    background-color: blue;
    position: absolute;
    top: 50px;
    left: 50px;
    z-index: 3; /* higher z-index */
  }
</style>
</head>
<body>
<div class="red-box"></div>
<div class="blue-box"></div>
</body>
</html>
```

➢ Lists:

CSS provides various properties to style lists. Here are some commonly used list-related properties:

1. **list-style-type:**
   Defines the appearance of the list item marker.
   For unordered lists (<ul>), values like disc, circle, and square are commonly used. (default value is 'bullet')
   For ordered lists (<ol>), values like decimal, lower-alpha, upper-roman determine the numbering style. (default value is 'decimal')

```
<style>
    ul {
      list-style-type: square;
      margin-left: 20px;
    }

    ol {
      list-style-type:upper-roman;
      margin-left: 20px;
    }

    li {
      margin-bottom: 5px;
    }
  </style>
</head>
<body>
  <h2>Unordered List</h2>
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
  </ul>

  <h2>Ordered List</h2>
  <ol>
    <li>First item</li>
    <li>Second item</li>
    <li>Third item</li>
  </ol>
</body>
</html>
```



2. **list-style-image:**
   Allows you to use a custom image as the list item marker instead of the default marker specified by list-style-type.

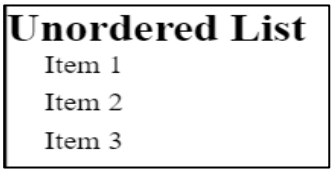
```
ul {
  margin-left: 20px;
  list-style-image: url("image.jpg");
}
```

3. **list-style-position:**
   Determines where the list item marker appears in relation to the content.
   outside places the marker outside the content flow, and inside places it inside the content flow.

```
ul {
  list-style-type:square ;
  margin-left: 20px;
  list-style-position: inside;
}
```

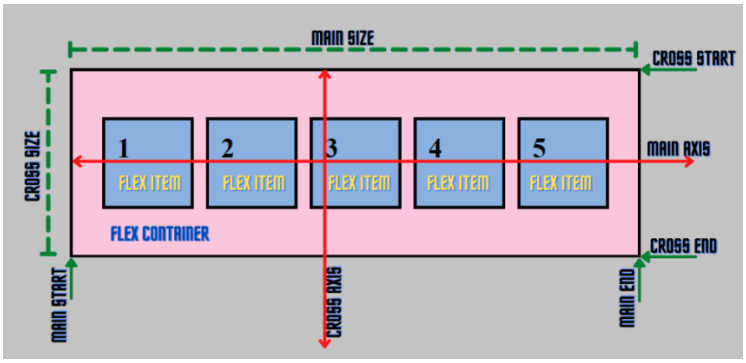
4. **list-style-none:** remove the default markers.

```
ul {
  list-style-type:none ;
  margin-left: 20px;
}
```



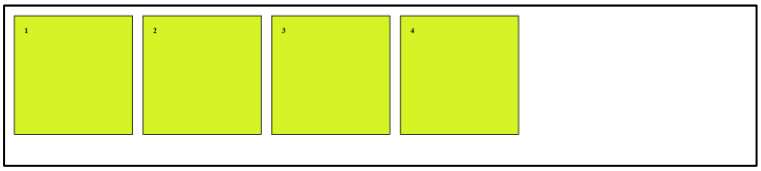
FLEX BOX

CSS Flexbox is a layout model that allows you to design more flexible and efficient page layouts in a two-dimensional space – either as rows or columns. It provides a way to distribute space and align content in a container, even when the size of the items is unknown or dynamic.



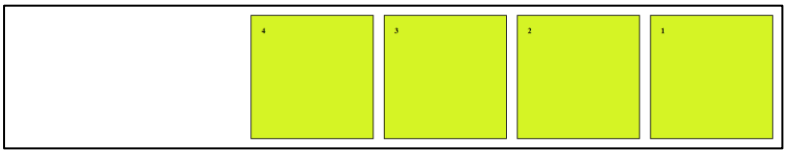
1. **Flex Container:**
   To start using the Flexbox model, you need to first define a flex container using **display: flex;**. This turns all direct children of the container into flex items.

```
<style>
    .container{
        display: flex;

    }

    .box{
        height: 200px;
        width: 200px;
        background-color: rgb(213, 244, 37);
        border: 2px solid black;
        margin: 10px;
        padding: 20px;
        font-weight: bold;
    }
    .box1{
      order: 1;
      background-color: aqua;
    }
    .box2{
      background-color: red;
      order:4;

    }
</style>
</head>
<body>
  <div class="container">
      <div class="box box1">1</div>
      <div class="box box2">2</div>
      <div class="box box3">3</div>
      <div class="box box4">4</div>
  </div>
```



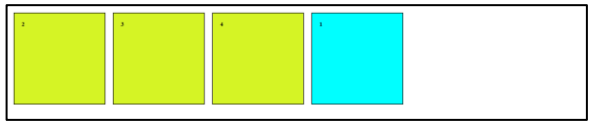
2. **Flex Direction:**
   It defines the main axis along which the flex items are placed inside the flex container: **row (default), row-reverse, column,** or **column-reverse**.

```
.container{
    display: flex;
    flex-direction: row-reverse;
}
```



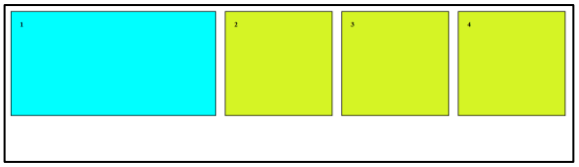
3. **Flex Items:**
   Elements inside the flex container become flex items and can be aligned and ordered using various flex properties.
   Flex item Properties:
   - **order:** Change the order of flex items according to order specified. By default, all flex items have an order value of 0.

```
.box1{
  background-color: aqua;
  order: 1;
}
```



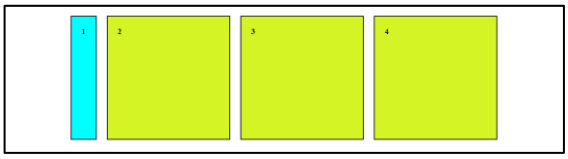
   - **flex-grow:** Determines how much a flex item should grow relative to the other items in the container.

```
.box1{
  background-color: aqua;
  flex-grow: 2;
}
```



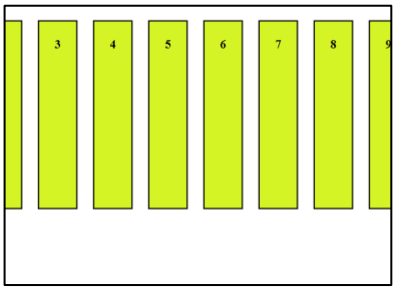
   - **flex-shrink:** Specifies how much a flex item can shrink relative to the other items in the container.
   - **flex-basis:** Sets the initial size of a flex item.

```
.box1{
  background-color: aqua;
  flex-basis:content;
}
```

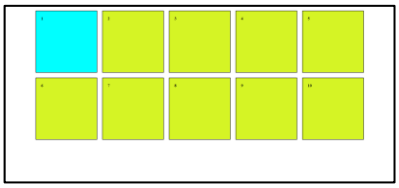


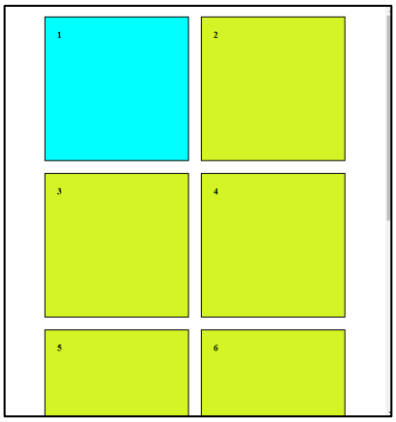
➢ Alignment and Spacing:

1. **flex-wrap:** The flex-wrap property in Flexbox controls whether flex items are forced onto a single line or can wrap onto multiple lines within a flex container when there isn't enough space along the main axis to accommodate them all.
   - **flex-wrap:nowrap;** All items stay in a single line even if they don't fit, causing some to overflow or get squished.

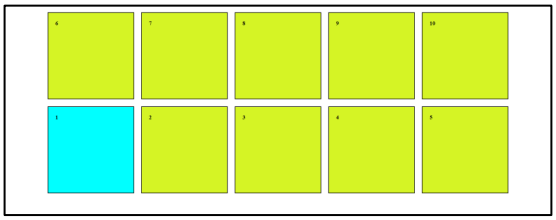


- **flex-wrap:wrap;** If the box isn't wide enough, items will move onto the next line, creating multiple lines to fit them all.



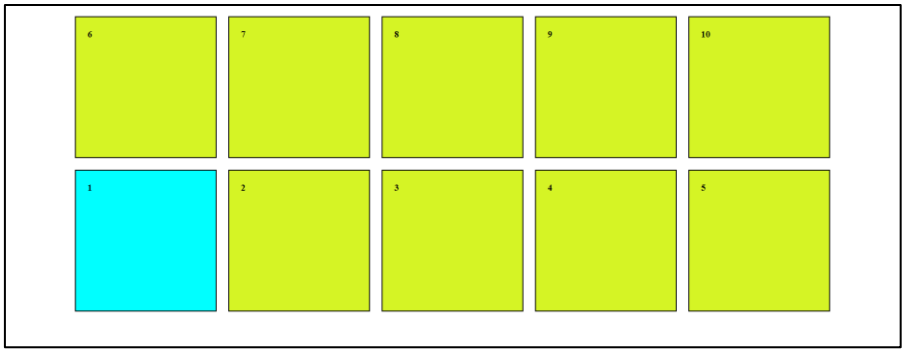


- **flex-wrap: wrap-reverse; -** Similar to wrapping, but items wrap onto multiple lines in the opposite order, starting from the bottom (or end) and moving upwards.



2. **justify-content:** Controls the alignment of flex items along the main axis. (center, flex-start, flex-end, baseline, end, left, right etc)
3. **align-items**: Defines how flex items are aligned along the cross axis of the flex container.
4. **align-content:** Similar to align-items but for multiple lines of flex items within the flex container.

```
.container{
  display: flex;
      flex-direction:row;
      flex-wrap:wrap;
      justify-content: center;
      align-items: center;
      align-content: center;
}
```

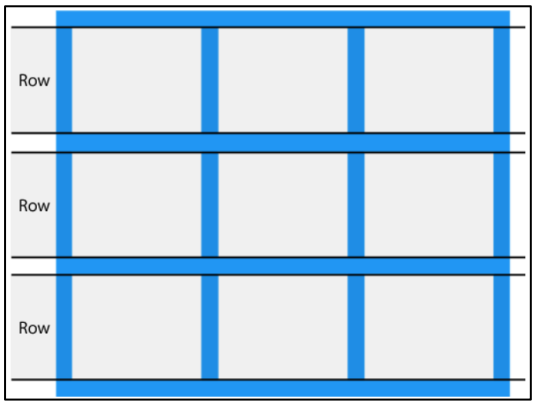


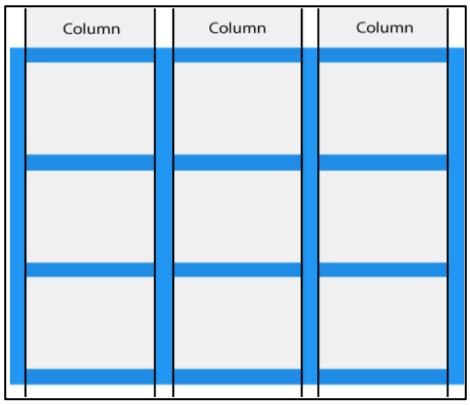
CSS GRID

CSS Grid is a powerful layout system that allows you to create two-dimensional layouts using rows and columns. It's fantastic for arranging content on a web page with precise control over placement and sizing.

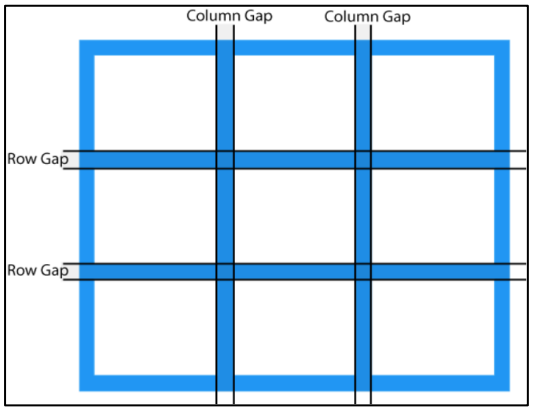
➢ Introduction to CSS Grid:

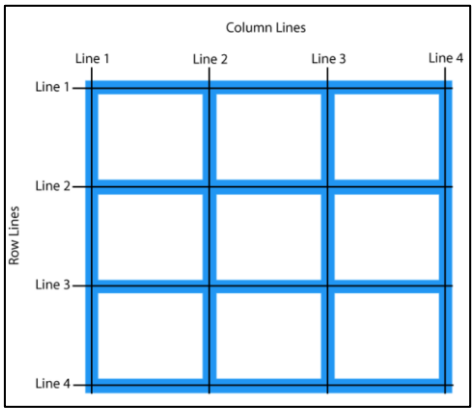
Grid Rows and columns:





Row/Column Gap and Row/Column Lines:





display: grid; - An HTML element becomes a grid container when its *display* property is set to *grid or inline-grid.*

```
<style>
.grid-container {
  display: grid;
  background-color: #2196F3;
  padding: 5px;
}

.grid-container > div {
  background-color: pink;
  text-align: center;
  border: 1px solid #2196F3;
}
</style>
</head>
<body>
<h1>CSS Grid</h1>
<div class="grid-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
  <div>7</div>
  <div>8</div>
</div>
</body>
</html>
```

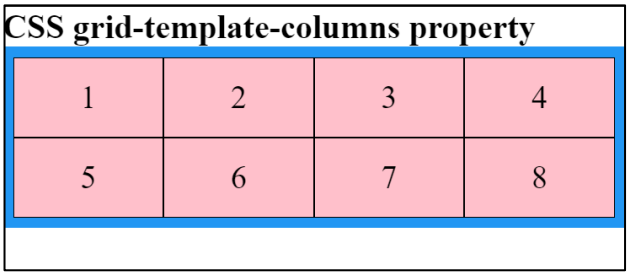
CSS Grid

1
2
3
4
5
6
7
8

Grid container properties:

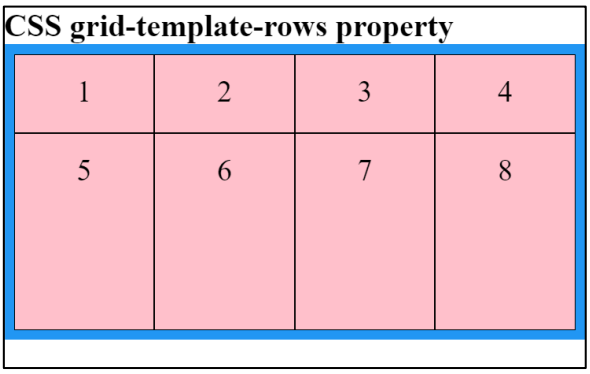
1. **grid-template-columns:** Specify the number of columns in your grid layout.
   Eg: '**grid-template-columns: auto auto;**' means 2 columns.
   Or you can manually assign width (30px 30px…)
   Or you can also set it as '**grid-template-columns: repeat (2, 200px);**' sets up 2 columns.

```css
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto auto;
  background-color: #2196F3;
  padding: 10px;
}

.grid-container > div {
  background-color: pink;
  text-align:center;
  padding: 20px 0;
  font-size: 30px;
  border: 1px solid black;
}
```



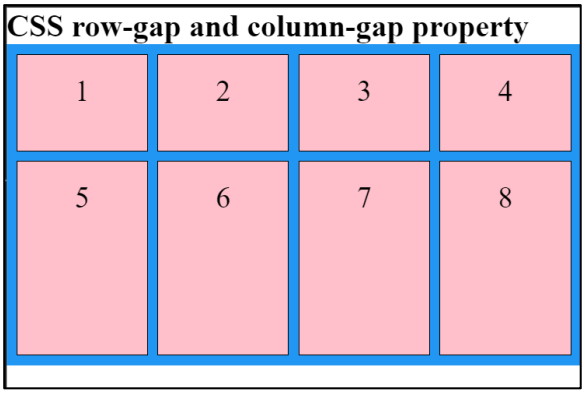
2. **grid-template-rows:** allows defining row sizes. Use the grid-template-rows property to specify the size (height) of each row.

```css
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto auto;
  grid-template-rows: 80px 200px;
  background-color: #2196F3;
  padding: 10px;
}

.grid-container > div {
  background-color: pink;
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
  border: 1px solid black;
}
```



3. **grid-row-gap;** and **grid-column-gap;:**

```css
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto auto;
  grid-template-rows: 80px 200px;
  row-gap: 10px;
  column-gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}
.grid-container > div {
  background-color: pink;
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
  border: 1px solid black;
}
```

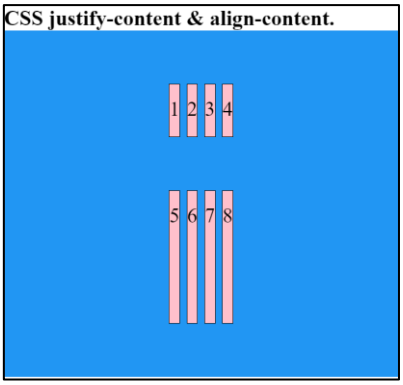


gap is a shorthand property for grid-row-gap and grid-column-gap.
gap: 10px; > for both row and column gap.

4. **justify-content and align-content:** aligns grid items along the row axis (horizontally).
   align-content: aligns grid items along the column axis (vertically).

Sr.no	Values	Description
1	start	Items are packed toward the start of the container.
2	end	Items are packed toward the end of the container.
3	center	Lines are centered along the cross-axis
4	stretch	Lines are stretched to fit the container.
5	space-between	Lines are evenly distributed in the container with extra space placed between them.
6	space-around	Lines are evenly distributed in the container with equal space around them.
7	space-evenly	Lines are evenly distributed in the container with equal space around them, including the edges.

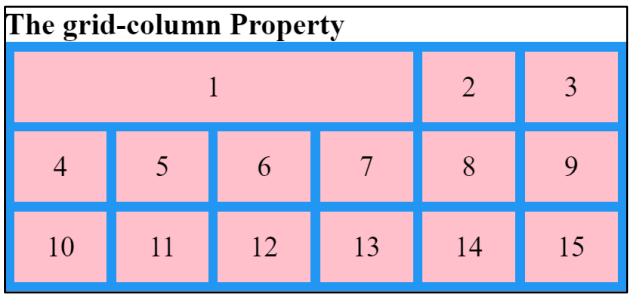
```css
.grid-container {
  display: grid;
  height: 500px;
  grid-template-columns: auto auto auto auto;
  grid-template-rows: 80px 200px;
  row-gap: 10px;
  column-gap: 10px;
  background-color: #2196F3;
  padding: 10px;
  justify-content: center;
  align-content:space-evenly;
}
.grid-container > div {
  background-color: pink;
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
  border: 1px solid black;
}
```



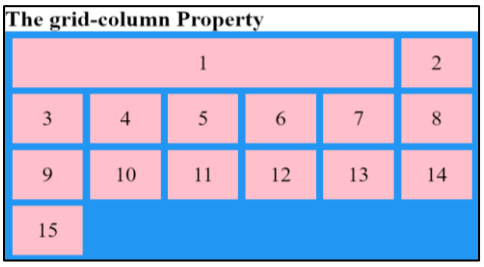
Properties for grid-items: A grid container contains grid items.

1. **grid-column:** Specifies the grid lines a grid item will span horizontally. It takes values in the format of **grid-column-start / grid-column-end** or using the **span** keyword.
E.g. grid-column: 1/5; Item1 will start on column 1 and end before column 5.

```css
<style>
  .grid-container {
    display: grid;
    grid-template-columns: auto auto auto auto auto auto;
    gap: 10px;
    background-color: #2196F3;
    padding: 10px;
  }

  .grid-container > div {
    background-color:pink;
    text-align: center;
    padding: 20px 0;
    font-size: 30px;
  }

  .item1 {
    grid-column: 1 / 5;
  }
</style>
```

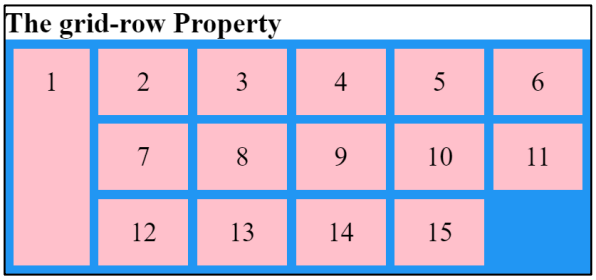


E.g. grid-column: 1 / span 5; Item1 will start on column-line 1 and span 5 columns

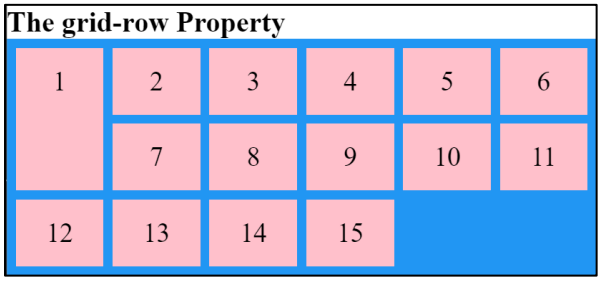


2. **grid-row:** Specifies the grid lines a grid item will span vertically. It takes values in the format of **grid-row-start / grid-row-end** or using the **span** keyword.
E.g. grid-row: 1/4; Item1 will start on row-line 1 and end on row-line 4:

```
<style>
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto auto auto auto;
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}

.grid-container > div {
  background-color: pink;
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}

.item1 {
  grid-row: 1 / 4;
}
  }
```



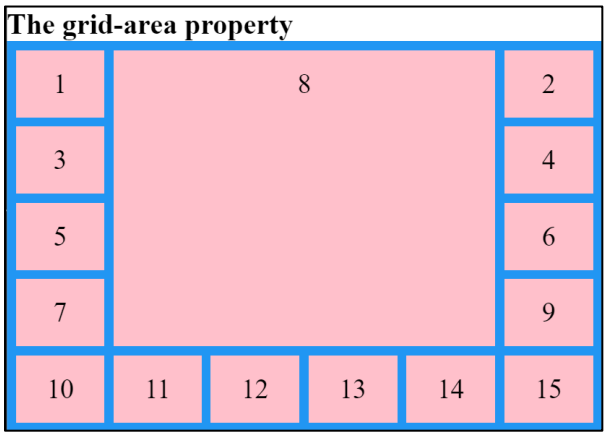
E.g. grid-row: 1 span/2; Item1 will start on row 1 and span 2 rows.



3. **grid-area:** You can use the grid-area property to specify where to place an item.
The syntax is:
grid-row-start / grid-column-start / grid-row-end / grid-column-end.

```
.item8 {
  grid-area: 1 / 2 / 5 / 6;
}
```

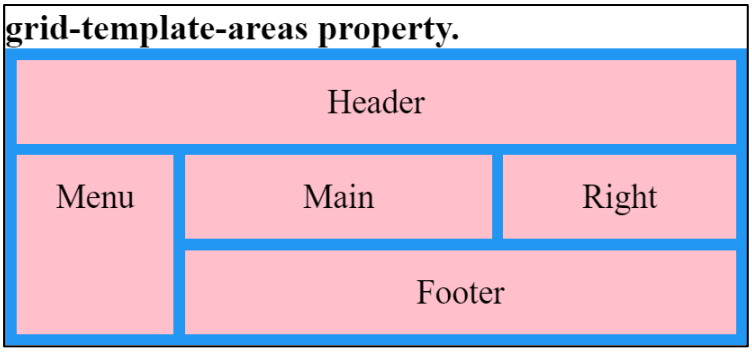
Item8 will start on row-line 1 and column-line 2, and end on row-line 5 column-line 6:



grid-area 2 / 1 / span 2 / span 3; - Item8 will start on row-line 2 and column-line 1, and span 2 rows and 3 columns:

4. **grid-template-areas:** you assign names to different areas of the grid using strings. Each string represents a row in the grid, and items within that string represent the columns.
   You assign grid items to these named areas using the **grid-area** property.

```
<style>
.item1 { grid-area: header; }
.item2 { grid-area: menu; }
.item3 { grid-area: main; }
.item4 { grid-area: right; }
.item5 { grid-area: footer; }

.grid-container {
  display: grid;
  grid-template-areas:
    'header header header header header header'
    'menu main main main right right'
    'menu footer footer footer footer footer';
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}

.grid-container > div {
  background-color: pink;
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
</style>
</head>
<body>
<h1>Grid Layout</h1>
<div class="grid-container">
  <div class="item1">Header</div>
  <div class="item2">Menu</div>
  <div class="item3">Main</div>
  <div class="item4">Right</div>
  <div class="item5">Footer</div>
</div>
</body>
</html>
```



CSS MEDIA QUERIES

CSS Media Queries are an essential part of responsive web design, allowing you to apply specific styles based on various device characteristics such as screen width, height, orientation, and more. They enable you to create layouts and designs that adapt to different devices and screen sizes.

Syntax:

@media 'media-type' and (media-feature) {
//CSS Code
}
@media: This is the syntax to indicate a media query in CSS.
media-type: Specifies the type of media being targeted (e.g., screen, print, all).
media-feature: Conditions based on various features like width, height, orientation, resolution, etc.
Inside the curly braces {}, you place the CSS rules that you want to apply when the conditions of the media query are met.

➢ CSS Media Types

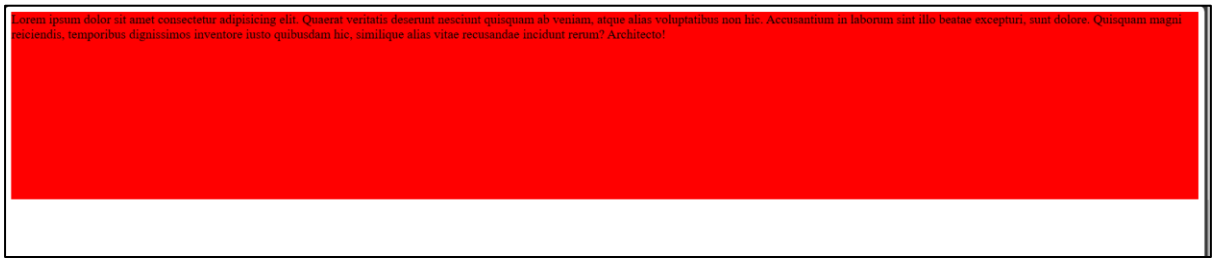
Sr.no	Value	Description
1	all	Used for all media type devices.
2	print	Used for print preview mode.
3	screen	Used for computer screens, tablets, smart-phones etc.

➢ CSS Common Media Features:

Sr.no	Value	Description
1	orientation	Orientation of the viewport. Landscape or portrait.
2	max-height	Maximum height of the viewport.
3	min-height	Minimum height of the viewport.
4	height	Height of the viewport (including scrollbar).
5	max-width	Maximum width of the viewport.
6	min-width	Minimum width of the viewport.
7	width	Width of the viewport (including scrollbar).

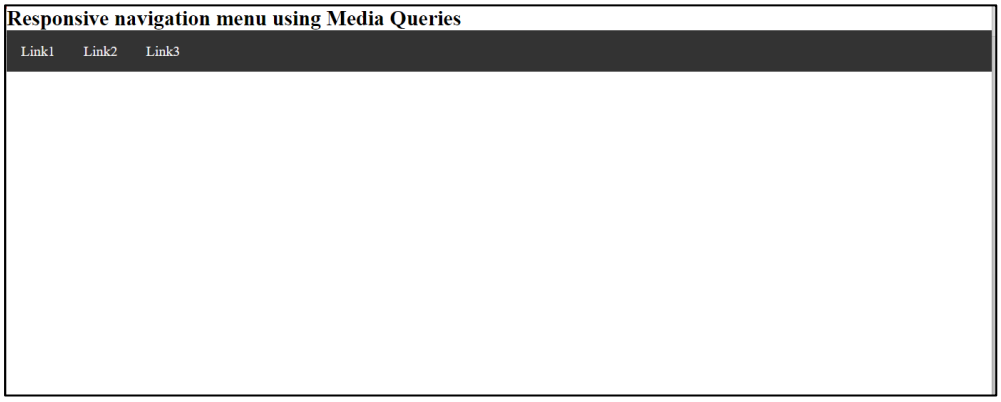
Example 1: Changing background color to yellow when max-width of screen is 800px or less.

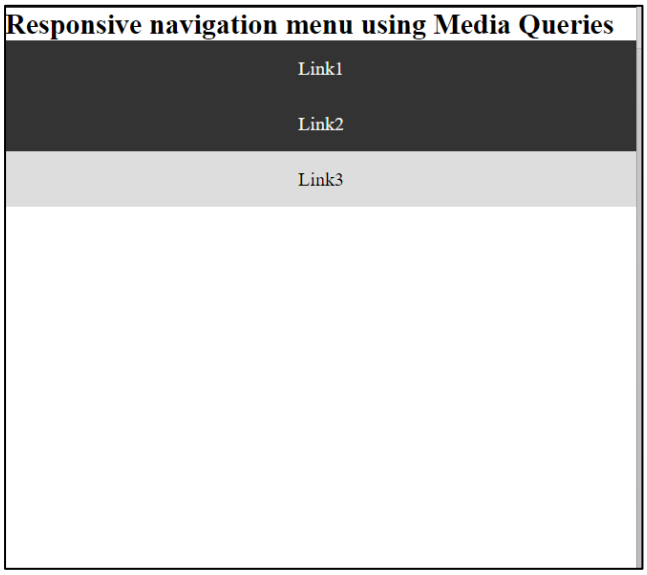
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0">
    <title>MediaQueries</title>
    <style>
        .container{
            background-color: red;
            height: 34vh;
        }
        @media screen and (max-width:800px){
            body{
                background-color: yellow;
            }

        }
    </style>
</head>
<body>
    <div class="container">Lorem ipsum dolor sit amet
    consectetur adipisicing elit. Quaerat veritatis
    deserunt nesciunt quisquam ab veniam, atque alias
    voluptatibus non hic. Accusantium in laborum sint
    illo beatae excepturi, sunt dolore. Quisquam magni
    reiciendis, temporibus dignissimos inventore iusto
    quibusdam hic, similique alias vitae recusandae
    incidunt rerum? Architecto!</div>

</body>
</html>
```



Example 2: Responsive navigation menu using media query.

```
<style>
* {
  box-sizing: border-box;
}
.nav {
  overflow: hidden;
  background-color: #333;
}
.nav a {
  float: left;
  display: block;
  color: #f2f2f2;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}
.nav a:hover {
  background-color: #ddd;
  color: black;
}

/* On screens that are 600px wide or less, make the menu links stack on top of
each other instead of next to each other */
@media screen and (max-width: 600px) {
  .nav a {
    float: none;
    width: 100%;
  }
}
</style>
</head>
<body>

<h2>Responsive navigation menu using Media Queries</h2>
<div class="nav">
  <a href="#">Link1</a>
  <a href="#">Link2</a>
  <a href="#">Link3</a>
</div>
</body>
</html>
```



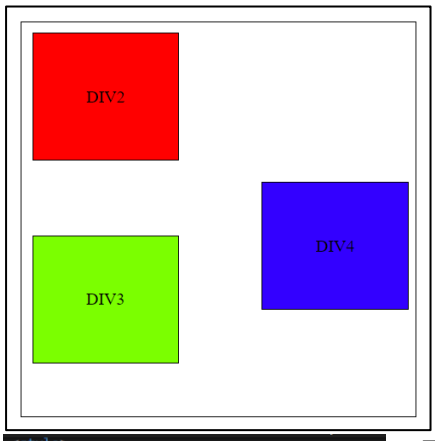


TRANSFORM, TRANSITION AND ANIMATION

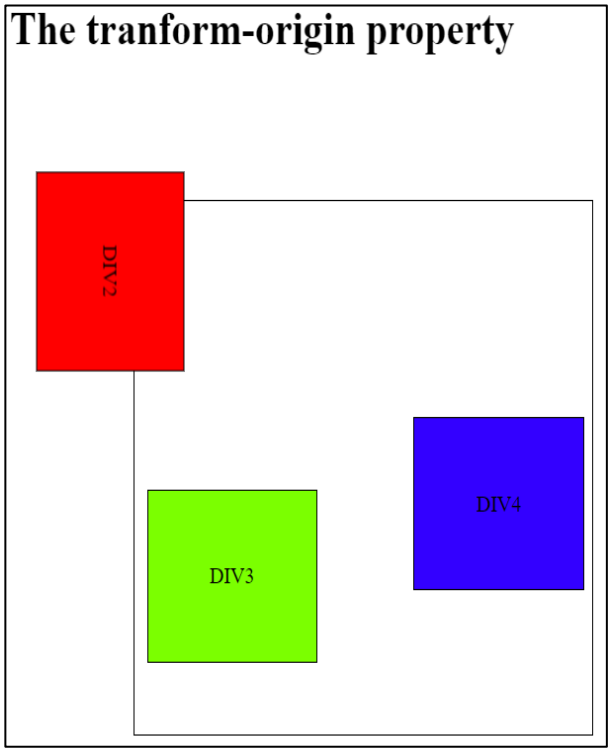
➢ Transform:

CSS transforms allow you to modify the appearance of an element in 2D or 3D space. They are used to translate, rotate, scale, and skew elements. **transform** property is used to apply 2D or 3D effects/transformations to an element.
2D transform: x and y axis
3D transform: x, y and z axis

1. **transform-origin:** allows you to change the position of 'transformed elements'.
   Syntax: transform-origin: *x-axis y-axis z-axis* | initial | inherit;
   Before transformation:

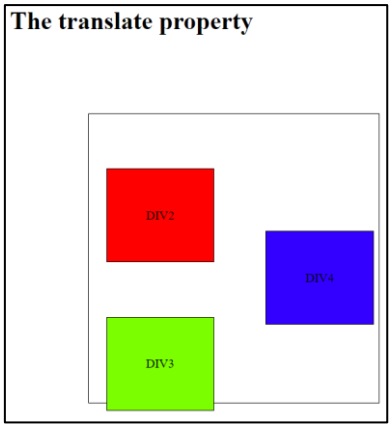



```
<style>
        #div1 {
          position: relative;
          height: 350px;
          width: 350px;
          margin: 100px;
          padding: 10px;
          border: 1px solid black;
        }
        #div2 {
          padding: 50px;
          position: absolute;
          border: 1px solid black;
          background-color: red;
          transform: rotate(90deg);
          transform-origin: 30px top;
        }
        #div3 {
          padding: 50px;
          position: absolute;
          bottom: 50px;
          border: 1px solid black;
          background-color: rgb(123, 255, 0);
        }
        #div4 {
          padding: 50px;
          position: absolute;
          right: 6px;
          bottom: 100px;
          border: 1px solid black;
          background-color: rgb(51, 0, 255);
        }
</style>
</head>
    <body>
        <h1>The tranform property</h1>
        <div id="div1">
          <div id="div2">DIV2</div>
          <div id="div3">DIV3</div>
          <div id="div4">DIV4</div>
        </div>
```



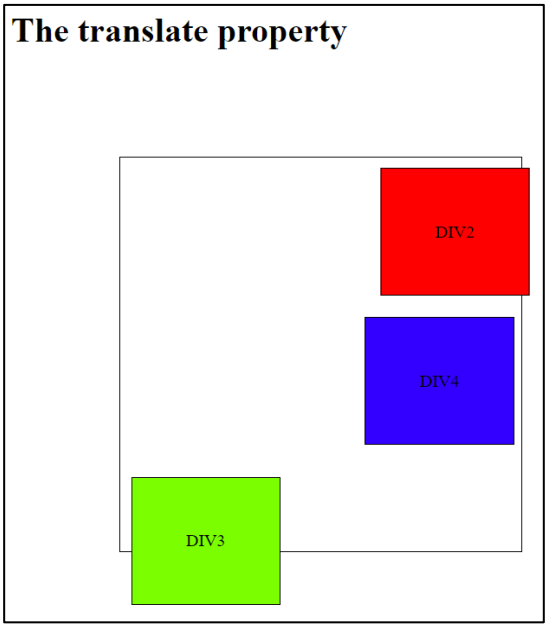
2. **translate ():**
   - **transform:translate(x,y);** Defines a 2D translation.
   - **transform:translate3d(x,y,z);** for 3D translation.

```css
#div2 {
  padding: 50px;
  position: absolute;
  border: 1px solid black;
  background-color: red;
  transform:translate(12px,50%)
}
#div3 {
  padding: 50px;
  position: absolute;
  bottom: 50px;
  border: 1px solid black;
  background-color: rgb(123, 255, 0);
  transform:translate3d(12px,50%,3em)
}
```



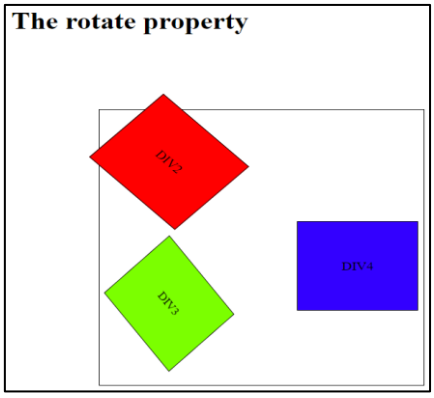
- **transform:translateX(x);** Defines a translation, using only the value for the x-axis
- **transform:translateY(y);** Defines a translation, using only the value for the y-axis
- **transform:translateZ(z);** Defines 3D translation, using only the value for the z-axis

```css
#div2 {
  padding: 50px;
  position: absolute;
  border: 1px solid black;
  background-color: red;
  transform:translateX(230px);
}
#div3 {
  padding: 50px;
  position: absolute;
  bottom: 50px;
  border: 1px solid black;
  background-color: rgb(123, 255, 0);
  transform:translateY(100px);
}
#div4 {
  padding: 50px;
  position: absolute;
  right: 6px;
  bottom: 100px;
  border: 1px solid black;
  background-color: rgb(51, 0, 255);
  transform:translateZ(5px);
}
```



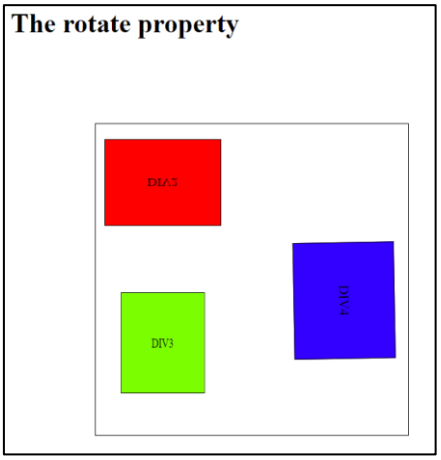
3. **rotate ():**
   - **transform:rotate(angle);** -Defines a 2D rotation, the angle is specified in the parameter.
   - **transform:rotate3d(x,y,z,angle); -** Defines a 3D rotation.

```css
#div2 {
  padding: 50px;
  position: absolute;
  border: 1px solid black;
  background-color: red;
  transform:rotate(45deg)
}
#div3 {
  padding: 50px;
  position: absolute;
  bottom: 50px;
  border: 1px solid black;
  background-color: rgb(123, 255, 0);
  transform:rotate3d(1,2,3,60deg)
}
```



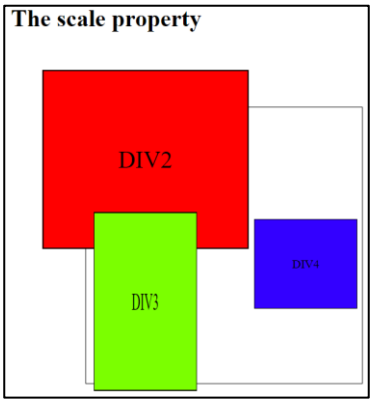
- **transform:rotateX(angle);** - Defines a 3D rotation along the X-axis.
- **transform:rotateY(angle);** - Defines a 3D rotation along the Y-axis.
- **transform:rotateZ(angle);** - Defines a 3D rotation along the Z-axis.

```
#div2 {
  padding: 50px;
  position: absolute;
  border: 1px solid black;
  background-color: red;
  transform:rotateX(149deg);
}
#div3 {
  padding: 50px;
  position: absolute;
  bottom: 50px;
  border: 1px solid black;
  background-color: rgb(123, 255, 0);
  transform:rotateY(44deg);
}
#div4 {
  padding: 50px;
  position: absolute;
  right: 6px;
  bottom: 100px;
  border: 1px solid black;
  background-color: rgb(51, 0, 255);
  transform:rotateZ(89deg);
}
```



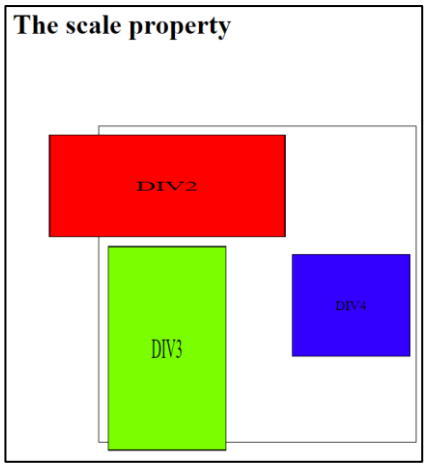
4. **scale():**
   - **transform:scale(x,y);** - Defines a 2D scale transformation
   - **transform:scale3d(x,y,z);** - Defines a 3D scale transformation.

```
#div2 {
  padding: 50px;
  position: absolute;
  border: 1px solid black;
  background-color: red;
  transform:scale(2,2);
}
#div3 {
  padding: 50px;
  position: absolute;
  bottom: 50px;
  border: 1px solid black;
  background-color: rgb(123, 255, 0);
  transform:scale3d(1,2,3);
}
```



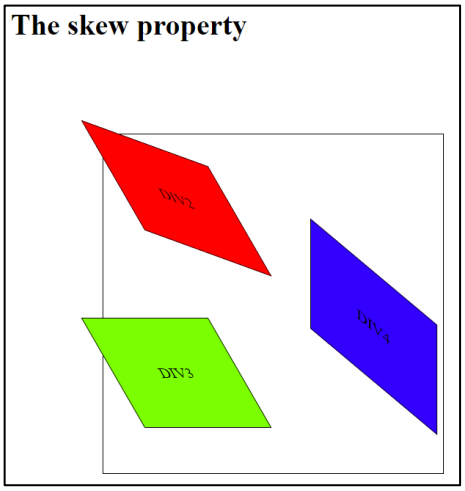
   - **transform:scaleX(x);** - Defines a scale transformation by giving a value for the X-axis
   - **transform:scaleY(y);** - Defines a scale transformation by giving a value for the Y-axis
   - **transform:scaleZ(z);** - Defines a 3D scale transformation by giving a value for the Z-axis

```
#div2 {
  padding: 50px;
  position: absolute;
  border: 1px solid black;
  background-color: red;
  transform:scaleX(2);
}
#div3 {
  padding: 50px;
  position: absolute;
  bottom: 50px;
  border: 1px solid black;
  background-color: rgb(123, 255, 0);
  transform:scaleY(2);
}
#div4 {
  padding: 50px;
  position: absolute;
  right: 6px;
  bottom: 100px;
  border: 1px solid black;
  background-color: rgb(51, 0, 255);
  transform:scaleZ(0.3);
}
```



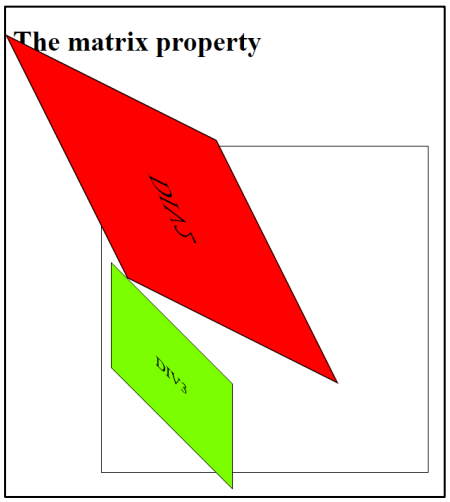
5. **skew():**
   - **skew(x-angle,y-angle); -**Defines a 2D skew transformation along the X- and the Y-axis
   - **skewX(angle); -**Defines a 2D skew transformation along the X-axis
   - **skewY(angle);**- Defines a 2D skew transformation along the Y-axis

```
#div2 {
  padding: 50px;
  position: absolute;
  border: 1px solid black;
  background-color: red;
  transform: skew(30deg,20deg);

}
#div3 {
  padding: 50px;
  position: absolute;
  bottom: 50px;
  border: 1px solid black;
  background-color: rgb(123, 255, 0);
  transform: skewX(30deg);

}
#div4 {
  padding: 50px;
  position: absolute;
  right: 6px;
  bottom: 100px;
  border: 1px solid black;
  background-color: rgb(51, 0, 255);
  transform:skewY(40deg);
}
```



6. **matrix():**
   - **transform:matrix(n,n,n,n,n,n); -** Defines a 2D transformation, using a matrix of six values.
   - **matrix3d(n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n)**;- Defines a 3D transformation, using a 4x4 matrix of 16 values

```
#div2 {
  padding: 50px;
  position: absolute;
  border: 1px solid black;
  background-color: red;
  transform: matrix(1,2,2,1,0,1);

}
#div3 {
  padding: 50px;
  position: absolute;
  bottom: 50px;
  border: 1px solid black;
  background-color: rgb(123, 255, 0);
  transform: matrix3d(1, 1, 3, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1);
}
```



➢ Transition:

The transition CSS property is a shorthand property for **transition-property**, **transition-duration**, **transition-timing-function**, **transition-delay**.

Sr.no	Value	Description
1	transition-property	Specifies the name of the CSS property the transition effect is for (the transition effect will start when the specified CSS property changes). A transition effect could typically occur when a user hover over an element. **Note:** Always specify the **transition-duration** property, otherwise the duration is 0, and the transition will have no effect.
2	transition-duration	Specifies how many seconds (s) or milliseconds (ms) a transition effect takes to complete.
3	transition-timing-function	Specifies the speed curve of the transition effect.
4	transition-delay	Specifies when the transition effect will start. Delay value is defined in seconds (s) or milliseconds (ms).

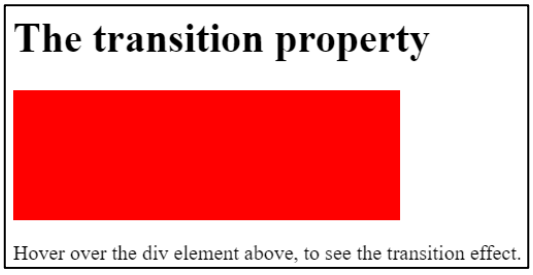
```
<style>
div {
  width: 100px;
  height: 100px;
  background: red;
  transition-property: width, height;
  transition-duration: 2s;
  transition-timing-function: ease-out;
  transition-delay: 3s;
}

div:hover {
  width: 300px;
  height: 300px;
}
```

Before hover: width-height(100px)

After hover (3sec): width-height(300px)

Transition Shorthand: transition: [transition-property] [transition-duration] [transition-timing-function] [transition-delay].

```
div {
  width: 100px;
  height: 100px;
  background: red;
  transition:width 2s ease-out 3s;
}

div:hover {
  width: 300px;
}
```



➢ Animation:

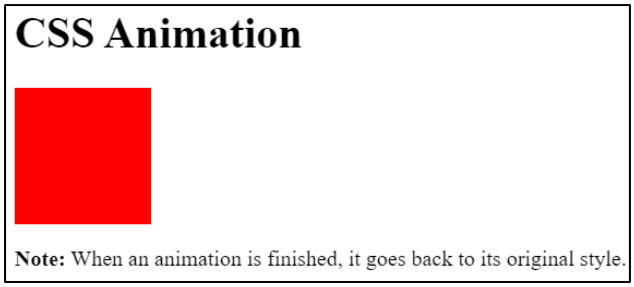
An animation lets an element gradually change from one style to another.
You can change as many CSS properties you want, as many times as you want.
To use CSS animation, you must first specify some keyframes for the animation.
Keyframes hold what styles the element will have at certain times.

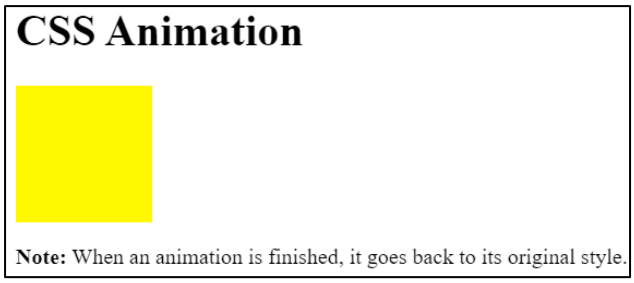
Sr.no	Value	Description
1	animation-name	Specifies the name of the keyframe you want to bind to the selector
2	animation-duration	Specifies how many seconds or milliseconds an animation takes to complete
3	animation-timing-function	Specifies the speed curve of the animation
4	animation-delay	Specifies a delay before the animation will start
5	animation-iteration-count	Specifies how many times an animation should be played
6	animation-direction	Specifies whether or not the animation should play in reverse on alternate cycles
7	animation-fill-mode	Specifies what values are applied by the animation outside the time it is executing
8	animation-play-state	Specifies whether the animation is running or paused
9	initial	Sets this property to its default value.
10	inherit	Inherits this property from its parent element.

➢ The @keyframes Rule

When you specify CSS styles inside the @keyframes rule, the animation will gradually change from the current style to the new style at certain times.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0">
    <title>Animation</title>
<style>
    div {
      width: 100px;
      height: 100px;
      background-color: red;
      animation-name: colorchange;
      animation-duration: 4s;
    }

    @keyframes colorchange {
      from {background-color: red;}
      to {background-color: yellow;}
    }
</style>
</head>
    <body>
    <h1>CSS Animation</h1>
    <div></div>
    <p><b>Note:</b> When an animation is finished, it
    goes back to its original style.</p>
    </body>
</html>
```





Animation Shorthand: animation: [animation-name] [animation-duration] [animation-timing-function] [animation-delay] [animation-iteration-count] [animation-direction] [animation-fill-mode];

```
animation:colorchange 2s ease-out 1s 2 normal forwards;
```

- **Conclusion:**

  CSS is a fundamental technology for web development. It separates the presentation from the structure of an HTML document, enabling the styling of web pages. CSS uses selectors to target HTML elements and defines their appearance through properties and values. Understanding the box model, units, and responsive design is crucial. Specificity plays a role in selector priority, and external stylesheets promote consistency across a website. Flexbox and Grid layouts enhance layout capabilities. CSS provides two powerful features, transform and transition, as well as a more advanced feature, animation, to enhance the visual experience of web pages. Mastery of these basics provides a solid foundation for creating visually appealing and responsive web pages.