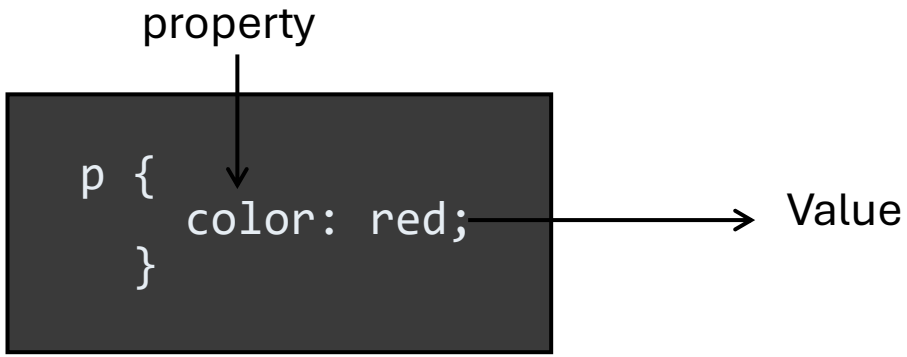
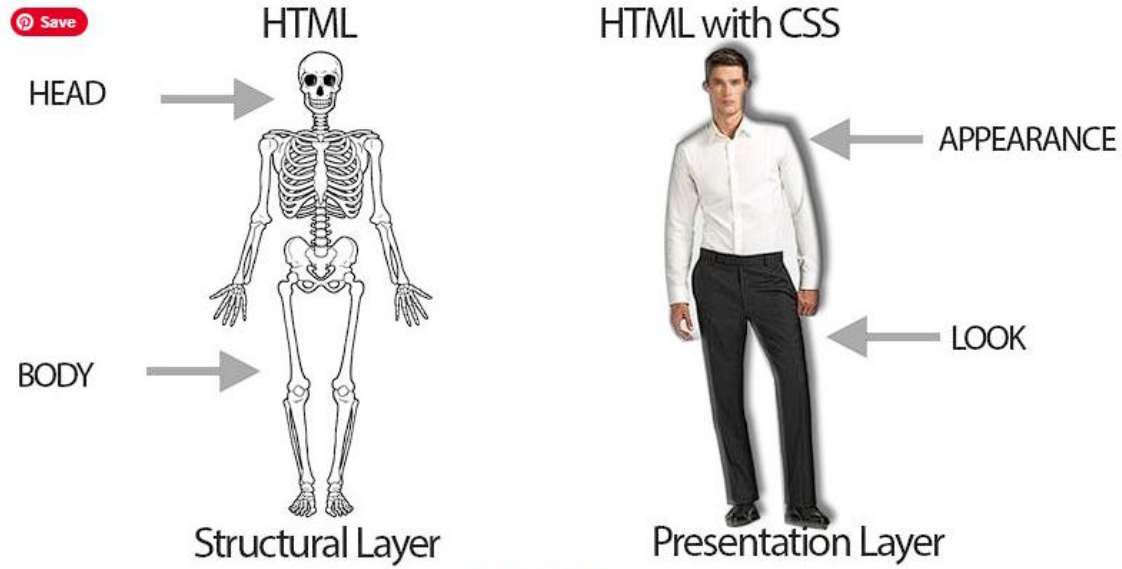
CSS Tutorial

Cascading Style Sheet

CSS Intro

Like HTML, CSS is not a programming language. It's not a markup language either. CSS is a style sheet language. CSS is what you use to selectively style HTML elements. For example, this CSS selects paragraph text, setting the color to red:



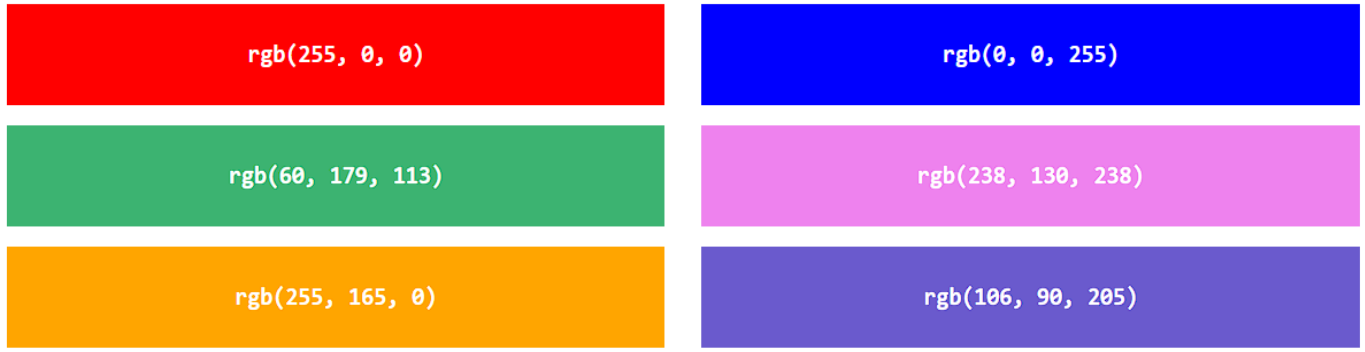


CSS Colors

RGB - **rgb(*red, green, blue*)**

To display black, set all color parameters to 0, like this: rgb(0, 0, 0).

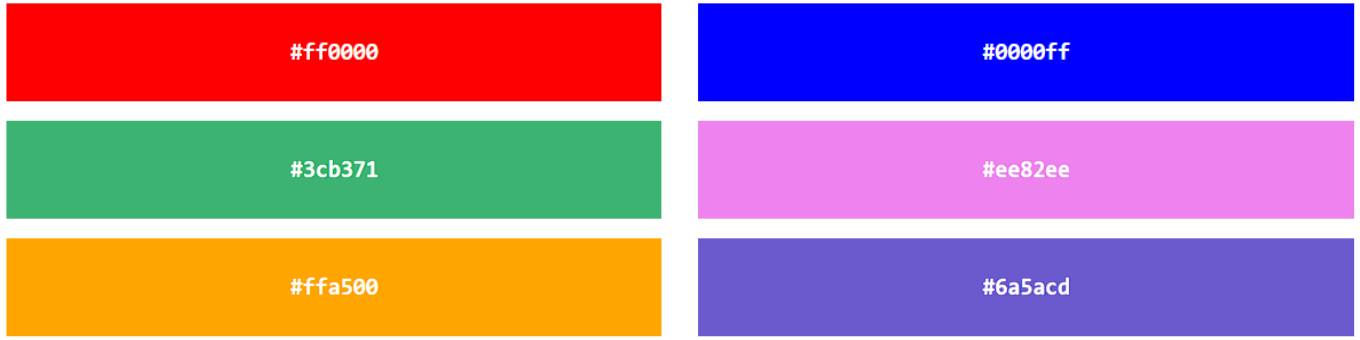
To display white, set all color parameters to 255, like this: rgb(255, 255, 255).



A hexadecimal color is specified with: #RRGGBB, where the RR (red), GG (green) and BB (blue)

To display black, set all values to 00, like this: #000000.

To display white, set all values to ff, like this: #ffffff



Implementing Css

Internal Css

Used In single HTML file

```
<!DOCTYPE html>
<html>
<head>
   <style>
      body {
        background-color:
red;
      }
       </style>
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

Inline CSS

Used In single HTML element

```
<!DOCTYPE html>
<html>
<head>
</head>
<body style='background-color:
red'>
<h1>This is a heading</h1>
<p >This is a paragraph.</p>
</body>
</html>
```

External Css

Can be used In many HTML element

```
<html>
<head>
  <link rel="stylesheet"
href="styles.css">
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

External CSS

styles.css

```
body {
  background-color: blue;
}
h1 {
  color: blue;
}
p {
  color: red;
}
```

External CSS

An external style sheet is used to define the style for many HTML pages.

With an external style sheet, you can change the look of an entire web site, by changing one file!

To use an external style sheet, add a link to it in the <head> section of the HTML page:

An external style sheet can be written in any text editor. The file must not contain any HTML code, and must be saved with a .css extension.

Here is how the "styles.css" looks:

Example

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet“
href="styles.css">
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

CSS Text and fonts

CSS Texts and Fonts

- Text Align
- Text Decoration
- Text Transform
- Text Overflow
- letter-spacing
- Line-height
- Word-spacing
- White-space

- Font-family
- Font-style
- Font-size
- Font-weight

```
h1 {
    color:red;
 }
```

Text color

```
 .heading
 {

  text-align:
centre/left/
right/justify;

  }
```

text-align(horizontal alignment)

```
p.uppercase {
  text-transform: uppercase;
}

p.lowercase {
  text-transform: lowercase;
}

p.capitalize {
  text-transform: capitalize;
}
```

The text-transform property is used to specify uppercase and lowercase letters in a text

Text Decoration Examples

CODE EXAMPLE

Text Decoration Overline

Text Decoration underline

~~Text Decoration line-through~~

Text Decoration none

```
<style>
    .text5{text-decoration-line:overline;}
    .text6{text-decoration-line:underline;}
    .text7{text-decoration-line:line-through;}
    .text8{text-decoration-line:none;}
</style>

<h4 class="text5">Text Decoration overline</h4>
<h4 class="text6">Text Decoration underline</h4>
<h4 class="text7">Text Decoration line-through</h4>
<h4 class="text8">Text Decoration none</h4>
```

Text Decoration Thickness

text-decoration-thickness change thickness of text decoration line. The value scales.

Thickness is always works with text decoration line.

CODE EXAMPLE

Text Decoration Thickness 1px

Text Decoration Thickness 5px

Text Decoration Thickness 1rem

```
<style>
  .p1{ text-decoration: underline solid 1px #000;}
  .p2{ text-decoration: underline solid 5px #00f;}
  .p3{ text-decoration: underline solid 1rem #f00;}
</style>

<p class="p1">Text Decoration Thickness 1px</p>
<p class="p2">Text Decoration Thickness 5px</p>
<p class="p3">Text Decoration Thickness 1rem</p>
```

Text Decoration Style

Text Decoration Style change style of text decoration. Default value is solid.

text-decoration-style	Value	Example
solid	create a solid line	Text Decoration Style solid
dashed	create a dashed line	Text Decoration Style dashed
dotted	create a dotted line	Text Decoration Style dotted
double	create a double line	Text Decoration Style double

Text Decoration Color

text-decoration-color change color of text decoration line. Default value is currentColor.

text-decoration-color	Value	Example
black	create a solid line with black color	Text Decoration color
red	create a solid line with red color	Text Decoration Color

Text Decoration

- Text Decoration
- **CSS text Decoration** add or remove decoration of text,
- like **underline**, **overline**, **line-through** and **none**.
- **Text Decoration** shorthand is the combination of four properties, i.e
- text decoration line (compulsory)
- text decoration style
- text decoration color
- text decoration thickness

Text Overflow

text-overflow property can add ellipsis (...) after text if text is overflowing from line. To add ellipsis, we also need to add **overflow:hidden** and **white-space:nowrap** to text element

```
<style>

.t1{ text-overflow: ellipsis;
max-width: 400px;
overflow: hidden;
 white-space: nowrap;
}
<style>

<p class='t1'>This is example to show the text overflow. This is example to show the text overflow</p>

o/p---- This is example to show the text overflow….
```

letter spacing

The letter-spacing property is used to specify the space between the characters in a text

```
<style>
.t1{  letter-spacing: 5px;;
}
<style>

<p class='t1'>This is example to show the text overflow. This is example to show the text overflow</p>
```

word spacing

The word-spacing property is used to specify the space between the words in a text

```
<style>
.t1{  word-spacing: 5px;;
}
<style>

<p class='t1'>This is example to show the text overflow. This is example to show the text overflow</p>
```

line height

The line-height property is used to specify the space between the lines

```
<style>
.t1{  line-height: 5px;;
}
<style>

<p class='t1'>This is example to show the text overflow. This is example to show the text overflow</p>
```

font-family

```
.p1 {
  font-family: "Times New Roman", Times, serif;
}

.p2 {
  font-family: Arial, Helvetica, sans-serif;
}
```

font-style

```
<style>
p {
  font-style: normal;
}
p {
  font-style: italic;
}
p {
  font-style: oblique;
}<style>

<p class='t1'>This is example to show the text overflow. This is example to show the text overflow</p
```

font-weight

In CSS, **Font weight** is used to give **Bold** or **Bolder** appearance to font are

Font Weight value in number
100 or thin
200 or extralight
300 or light
400 or regular
500 or medium
600 or semibold
700 or bold
800 or extrabold
900 or black

```
<style>
.t1{  font-weight: normal
}
<style>

<p class='t1'>This is example to show the text overflow. This is example to show the text overflow</p>
```

font-size

Font size property in css can change the **font size** of fonts. By default, all html elements are having their own font size set by useragent (browser).

Default Font size of root element *html* is 16px, p tag is 1em and h1 tag is 2em.

The popular units for **CSS fonts size** are *em* and *px*. The Top 5 **font-size font-size** units are.

Font Size values in CSS

•Pixels (px) → As per height of screen pixels

•Em(em) → Relative to nearest parent.

•percentage (%) → Relative to parent element.

•rem (rem) → relative to root parent (*HTML Tag*)

```
<div style="font-size:16px">
<p style="font-size:16px">Font size 16px </p>
 <p style="font-size:1em">Font size 1em </p>
<p style="font-size:100%">Font size 100% </p>
<p style="font-size:1rem">Font size 1rem </p>
</div>
```

Font size 16px
Font size 1 em
Font size 100%
Font size 1rem

Specificity

- Specificity is an algorithm that calculates the weight that is applied to a given CSS declaration .
  - important
  - Inline CSS(style attribute)
  - id#
  - Class
  - internal
  - external

CSS Display Property

CSS Display Property

CSS Display property is used to sets an element as block or inline in **normal document flow**. **Display property** can also **change display of an HTML Element**, like from **block to inline** and **inline to block** etc.

Value	Description	Example
`block`	Displays the element as a block-level element. It starts on a new line and takes up the full width available.	`<div style="display: block;">Block Element</div>`
`inline`	Displays the element as an inline element. It does not start on a new line and only takes up as much width as needed.	`<span style="display: inline;">Inline Element</span>`
`inline-block`	Behaves like `inline` but allows setting width and height like `block`.	`<div style="display: inline-block; width: 100px;">Inline Block</div>`
`none`	Hides the element, making it not appear in the document at all (no layout space taken).	`<div style="display: none;">Hidden Element</div>`
`flex`	Displays the element as a flex container, enabling flexbox layout for its children.	`<div style="display: flex;">Flex Container</div>`
`inline-flex`	Behaves like `inline`, but the element is a flex container.	`<div style="display: inline-flex;">Inline Flex</div>`
`grid`	Displays the element as a grid container, enabling grid layout for its children.	`<div style="display: grid;">Grid Container</div>`
`inline-grid`	Behaves like `inline`, but the element is a grid container.	`<div style="display: inline-grid;">Inline Grid</div>`
`table`	Displays the element as a table (like `<table>` element).	`<div style="display: table;">Table Container</div>`

Display inline

- **Display inline** is default display property of <a>, <img>, , <b>, <i>, <s>, etc. These elements occupy space of content only and does not break line
- Inline elements of text type (a, span, b, i, strong, em, small) doesn't support width property
- Properties of inline Elements
  - Occupy width of content only .
  - Need
 to break line.
  - Allow padding and margin of left and right side only.
  - Width and Height are not supported.


```
  <style>
    .blockToInline {
      display: inline;
      background: red;
    }
    .blockClass {
      background: pink;
    }
  </style>
 </head>
 <body>
  <!-- display properties -->
  <h1 class="blockClass">This is block element converting to inline</h1>
  <h1 class="blockToInline">This is block element converting to inline</h1>
 </body>
```

Display inline block

- **Display inline block** works same like **display inline**, but **inline-block** elements can have width and height.
- We can also add padding in **inline-block** element

```
<!-- <h1 class= blockClass >This is block element converting to inline</h1>
<h1 class="blockToInline">This is block element converting to inline</h1>
<h1 class="blockToInlineBlock">
  This is block element converting to inline Block
</h1>
```

This is block element converting to inline

This is block element converting to inline Block

```
<style>
  .blockToInline {
    display: inline;
    background: red;
    width: 200px;
    padding: 20px;
  }
  /* .blockClass {
    background: pink;
  } */
  .blockToInlineBlock {
    display: inline-block;
    background: red;
    width: 400px;
    padding: 20px;
  }
</style>
```

Display block

Display Block allow inline elements to behave like block Elements. , <h1> to <h6>, <p>, <address>, <pre>, <ul>, <ol>, <hr> all are **block level elements**. HTML Block elements can have both inline level and block level as child.

Properties of block elements

- Occupy Full width (100%) of the parent element.
- Starts from a new row
- Next element will come in next row.
- can have both inline and block elements as child.

Inline elment

Inline element to block

```
<!-- inline -->
<span class="inline">Inline elment</span>
<span class="inlinetoblock">Inline element to block</span>
```

```
<style>

.inline {
  background: green;
}
.inlinetoblock {
  background: purple;
  display: block;
}
</style>
```

Display none

Display none hide an html element from user. Thus it doesn't occupy any space. We can change display of these elements on hover of parent element.

```
<!-- inline -->
<span class="inline">Inline elment</span>
<span class="inlinetoblock">Inline element to block</span>
```

```
<style>

.inline {
  display :none
}
</style>
```

CSS Background

Background Properties

- Background Color
- Background Image
- Background Repeat
- Background Position
- Background Attachment
- Background

Background

- CSS can use both colors and images in backgrounds
- **Background colors** can have any colorname, hexacode or rgb color code. **Background Colors** occupy full width of Border Box.
- **Background images** are also used with background. Images like JPG, PNG and GIF can be used as **background images**. These images can repeat in x-axis, y-axis or no repeat, can change position and can be attached fixed to screen.

Background Color

Background color property is used to change **background color**. **Background color** can have either **color name** or **color code**.

```
<div class="backgroundClass">
  <h1>background image</h1>
</div>
```

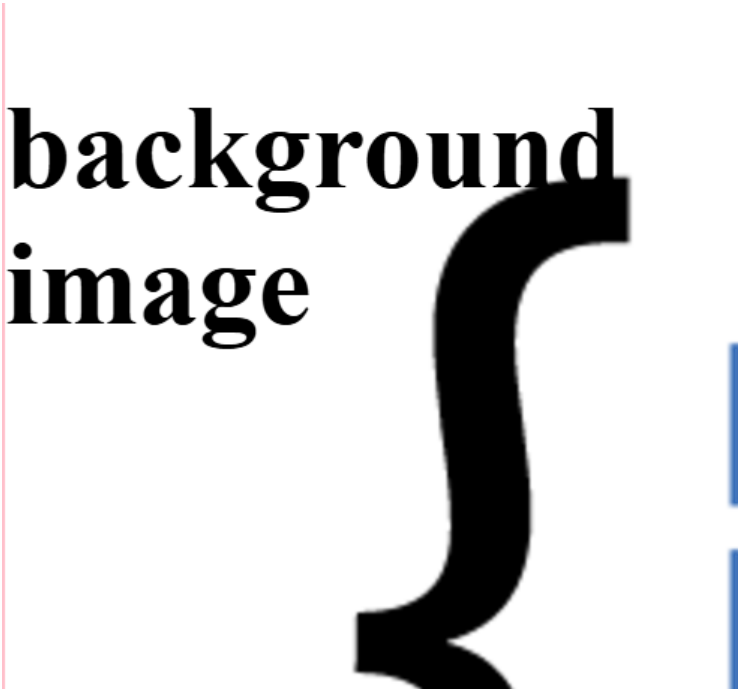
```
<style>
      .backgroundClass {
        background-color: blue;
      }
    </style>
```

Background Image

Background images are also used to set backgrounds in css. Popular image extension like, JPG, PNG, GIF and svg can be used for **background images**.

```
<style>
      .backgroundImage {
       background-image: url("./logoLight.png");
       width: 200px;
       height: 200px;
       border: 1px solid pink;
     }
   </style>
```

```
<div class="backgroundImage">
  <h1>background image</h1>
</div>
```



Background repeat

background-repeat: no-repeat can stop image from repeating in x and y direction of background. Thus remaining area is transparent.

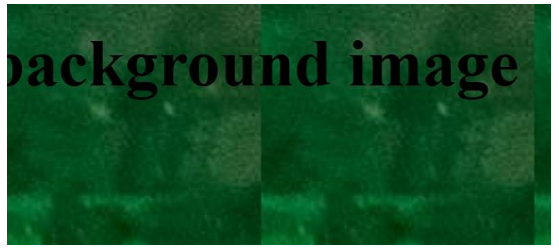
Background repeat values

- repeat
- no-repeat
- repeat-x
- repeat-y
- round
- space

```
<style>
        background-
image:url("./smallImage.jpg");
        width: 100vw;
        height: 100vh;
        border: 1px solid pink;
        /* repeat is by default */
        background-repeat: repeat;
    }
</style>
```

```
<div class="backgroundImage">
    <h1>background image</h1>
</div>
```

background-repeat: repeat;



background-repeat: no-repeat;

Background position

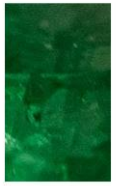
Background Position tells position of background image. The default the **background position** is left top.

```
<style>
        background-image:url("./smallImage.jpg");
        width: 100vw;
        height: 100vh;
        border: 1px solid pink;
        background-position: center;
        background-repeat: no-repeat;

    }
</style>
```

```
<div class="backgroundImage">
    <h1>background image</h1>
</div>
```

background-position: left top;(default)

background-position: center;



Background attachment

Background Attachment property tells whether a background image should scroll or remain fixed on scrolling down window. The default value is scroll..

```
<style>
      background-image:url("./smallImage.jpg");
      width: 100vw;
      height: 100vh;
      border: 1px solid pink;
      background-attachment:fixed;
    }
  </style>
```

```
<div class="backgroundImage">
  <h1>background image</h1>
</div>
```

Background

The **background property** is actually a shortcut of **background-image**, **background-repeat**, **background-position**, **background-attachment** and **background-color**.

```
<style>
      background-image:url("./smallImage.jpg");
      width: 100vw;
      height: 100vh;
      border: 1px solid pink;
      background-attachment:fixed;
    }
  </style>
```

```
<div class="backgroundImage">
  <h1>background image</h1>
</div>
```

CSS Styling Links

Styling Links

States

✔ a:link	Normal unvisited link
✔ a:hover	Link when hovered
✔ a:active	Moment link is clicked
✔ a:focus	Moment link receives focus
✔ a:visited	Link user has visited

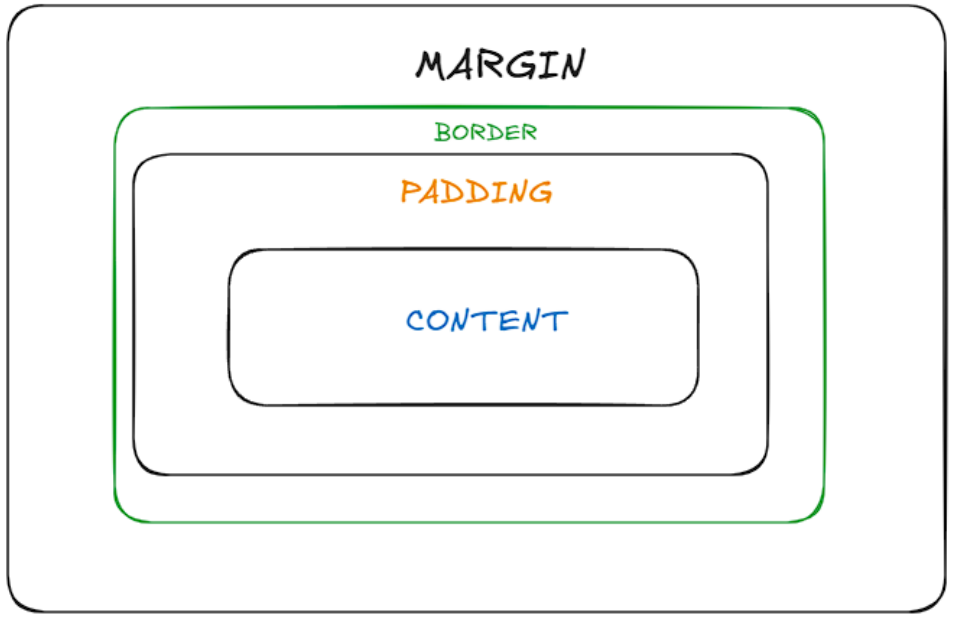
Styling Link

```
<style>
      /* on hover */
      a:hover {
        color: brown;
      }

      /* link is clicked */
      a:active {
        color: blueviolet;
      }

      /* click on link  focus */
      a:focus {
        background-color: red;
        color: cadetblue;
      }

      /* Link user visted */
      a:visited {
        background-color: burlywood;
      }
</style>
```

```
<body>
  <a href="">styling Links</a>
</body>
```

CSS BoxModal

Box Modal



- **CSS Box Model** is the backbone of **css layout design**.
- **BoxModel** includes **Content**, **padding**, **border** and **margin** which all together create a rectangular box, that's why its called **Box Model**
- **Content**: This is where the actual content, such as text or images, resides. It's the core of the box.
- **Padding**: Creates an internal buffer around the content. It ensures that the content does not touch the border.
- **Border**: A visible line or design surrounding the padding and content. Its thickness, style, and color can be customized.
- **Margin**: This defines the space between the element and its neighboring elements, helping in layout and positioning.

CSS Boxsizing

1. **box-sizing** is a CSS property that defines how the width and height of an element are calculated, affecting the overall size of the box.
2. There are two main values for **box-sizing: content-box (default) and border-box.**

Value	Description
content-box	**Default value.** Only the content's width and height are included. Padding and border are added *outside* these dimensions.
border-box	The padding and border are included in the element's total width and height, so the content area shrinks accordingly.

content-box (default):
•If you set width: 200px, the content will be 200px wide.
•Padding and border are added on top of this width, increasing the total size.
Total width = 200px + 20px (padding left) + 20px (padding right) + 5px (border left) + 5px (border right) = **250px**.

```
div {
  width: 200px;
  padding: 20px;
  border: 5px solid black;
  box-sizing: content-box;
}
```

Border-box :
•If you set width: 200px, the content will be 200px wide.
•Padding and border are added on top of this width, increasing the total size.
•Total width = **200px**

```
div {
  width: 200px;
  padding: 20px;
  border: 5px solid black;
  box-sizing: border-box;
}
```

CSS overflow

1. **CSS Overflow Property** controls the behavior of overflowing content from an element.
2. The possible values are **auto, visible, scroll and hidden**.
3. Overflow can be used when width or height of content is more than parent element.

overflow visible

overflow visible is usually *default value* of overflow property in most of the browsers. This will visible overflowing content from a element.

overflow auto

overflow auto is used when width or height of content is more and we want scrollbar in element. If width is more, scroll will comes in x-axis and if height is more, scroll will comes in y-axis.

overflow hidden

overflow hidden is used to hide overflowing content from an element.

overflow scroll

overflow scroll is used to see visible scrollbar even if content is not overflowing.

Overflow:visible

Lorem ipsum dolor sit amet consectetur adipisicing elit. Voluptate enim necessitatibus exercitationem quibusdam molestiae cum distinctio numquam suscipit doloremque dolorum quod ipsa nam soluta voluptatibus beatae illo veritatis, facilis error repudiandae mollitia omnis pariatur, commodi tempore sapiente. Quo amet voluptas totam architecto recusandae nobis velit officia corporis molestias dolores repudiandae maxime, eveniet, inventore, nisi ab ipsam. Error, dolores assumenda. Ex nostrum expedita numquam voluptatem tenetur! Recusandae, tempora maxime repudiandae temporibus facilis consequatur rerum doloremque, adipisci sint debitis quia mollitia perferendis dolore incidunt natus ex, odio dicta illum eligendi voluptatum delectus quis hic iure expedita! Veniam, dolorem excepturi?

Overflow:auto

Lorem ipsum dolor sit amet consectetur adipisicing elit. Voluptate enim necessitatibus exercitationem quibusdam molestiae cum distinctio numquam suscipit doloremque dolorum quod ipsa nam soluta voluptatibus beatae illo veritatis, facilis error repudiandae mollitia omnis pariatur, commodi tempore sapiente. Quo amet voluptas totam architecto recusandae nobis velit officia corporis molestias dolores repudiandae maxime, eveniet, inventore, nisi ab ipsam. Error, dolores assumenda. Ex nostrum expedita numquam voluptatem tenetur! Recusandae, tempora maxime repudiandae temporibus facilis consequatur

Overflow:scroll

Lorem ipsum dolor sit amet consectetur adipisicing elit. Voluptate enim necessitatibus exercitationem quibusdam molestiae cum distinctio numquam suscipit doloremque dolorum quod ipsa nam soluta voluptatibus beatae illo veritatis, facilis error repudiandae mollitia omnis pariatur, commodi tempore sapiente. Quo amet voluptas totam architecto recusandae nobis velit officia corporis molestias dolores repudiandae maxime, eveniet, inventore, nisi ab ipsam. Error, dolores assumenda. Ex nostrum expedita numquam voluptatem tenetur! Recusandae, tempora maxime repudiandae temporibus facilis consequatur

Overflow:hidden

sapiente. Quo amet voluptas totam architecto recusandae nobis velit officia corporis molestias dolores repudiandae maxime, eveniet, inventore, nisi ab ipsam. Error, dolores assumenda. Ex nostrum expedita numquam voluptatem tenetur! Recusandae, tempora maxime repudiandae temporibus facilis consequatur rerum doloremque,

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Expedita incidunt sequi ipsam alias officiis veniam praesentium eos! Eos in quaerat dolore accusantium accusamus rerum. Vero ducimus velit harum voluptatem quos?

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Expedita incidunt sequi ipsam alias officiis veniam praesentium eos! Eos in quaerat dolore accusantium accusamus rerum. Vero ducimus velit harum voluptatem quos?

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Expedita incidunt sequi ipsam alias officiis veniam praesentium eos! Eos in quaerat dolore accusantium accusamus rerum. Vero ducimus velit harum voluptatem quos?

```
<style>
    .box1 {
      width: 40%;
      background-color: aqua;

      /* padding-top: 10px;
      padding-right: 20px;
      padding-bottom: 5px;
      padding-left: 20px; */
      padding: 10px 20px 5px 20px;
    }
    .box2 {
      background-color: blueviolet;
      max-width: 500px;
      min-width: 400px;
      /* margin-top: 10px;
      margin-right: 20px;
      margin-bottom: 5px;
      margin-left: 20px; */
      margin: 10px 20px 5px 20px;
    }
    .box3 {
      background-color: red;
    }
</style>
```

CSS some universal selector

Universal selector – at start of your coding

```
*{
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    list-style: none;
  }
```

- * selects all elements.
- margin: 0; and padding: 0; remove the default margin and padding.
- box-sizing: border-box; ensures that width and height include padding and border.

CSS Float and clear

1. **CSS Float** is a positioning property in css used to float an element to the left or right corner of parent element and the next element or text wrapping around the left or right to it .
2. A floating element doesn't occupy space in normal flow.
3. By default, all html elements are non floating.

4. CSS float Values
    a. float: left
    b. float: right
    c. float: none

Float Left

CSS **Float Left** push an element to the left corner of parent element, and other elements wrap around it. A floating element doesn't occupy space in flow. Thus next non floating element will wrap around floating element.

Float Right

CSS **Float Left** push an element to the left corner of parent element, and other elements wrap around it. A floating element doesn't occupy space in flow. Thus next non floating element will wrap around floating element

CSS Clear Property

CSS Clear property is used to stop next element to wrap around the adjacent floating elements. Clear can have clear left, clear right or clear both values.

Values of clear property

None	*Default*
Left	*Clear left floating only*
Right	*Clear right floating only*
Both	*Clear Both left & right floating*

```
 <style>
      .imgFloat {
        /* float: right; */
        float: left;
      }
      .clear {
        clear: both;
      }
    </style>
<body>
    <h1>Float</h1>
    <!-- its like this becoz after float the image
leaves its position float lefts
    /right from parent and  for div only <p> exists
    and if overflow :hidden it will remove clear and
will be proper
        -->
    <div style="background: lightblue">
      <p class="imgFloat">
        the image will float left/right and paragraph
will set accordingly
      </p>
      <img src="./smallImage.jpg" alt="demo"
class="imgFloat" />
      <p class="clear">I don't want this to float</p>
    </div>
  </body>
```

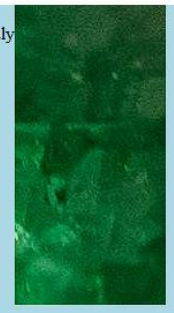
Float

the image will float left/right and paragraph will set accordingly

I dont want this to float

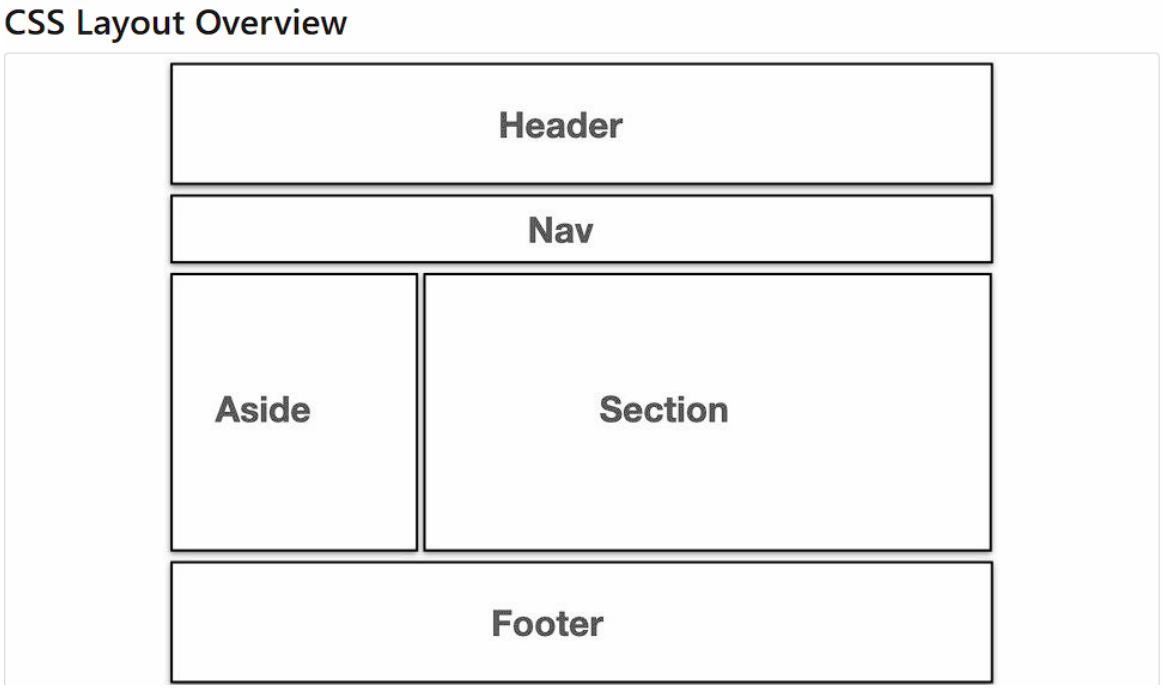
Float and clear

the image will float left/right and paragraph will set accordingly



I dont want this to float

Lets Try to make this layout using float

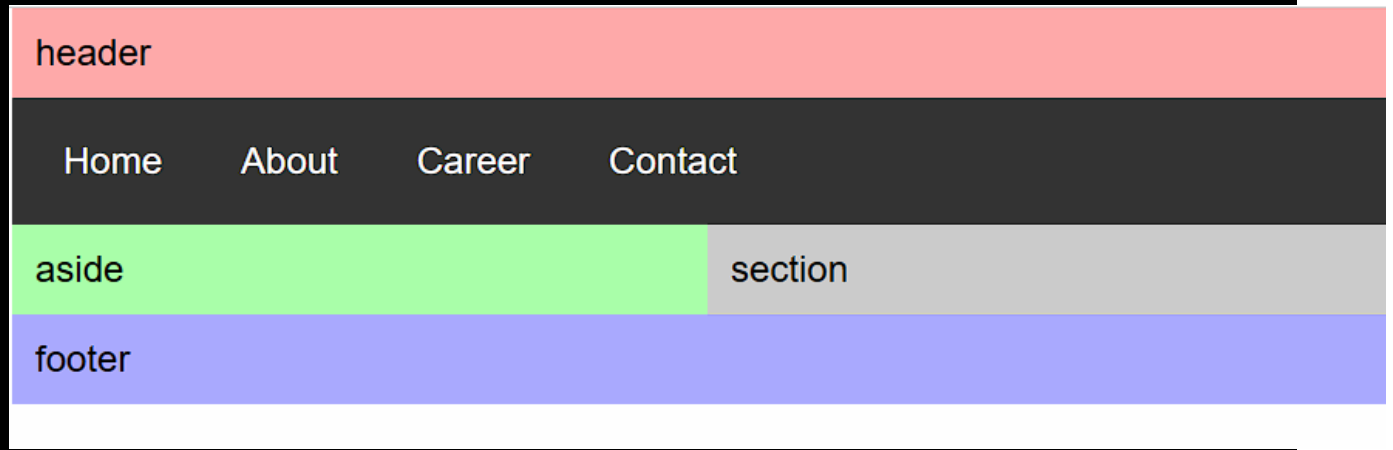


```
<style>
    * {
      margin: 0;
      box-sizing: border-box;
    }

    body {
      font: 16px sans-serif;
    }

    .container {
      max-width: 1200px;
      margin: auto;
    }

    header {
      background: #faa;
      padding: 10px;
    }
    nav {
      background: #333;
      padding: 10px;
      color: #fff;
    }
```

```
  nav ul {
       list-style: none;
       padding: 0;
      }

nav ul li {
       float: left;
       margin-right: 10px;
     }
nav ul li a {
       text-decoration: none;
       color: inherit;
       display: block;
       padding: 8px 12px;
     }
     main {
       background: #afa;
     }
     aside {
       width: 25%;
       float: left;
       padding: 10px;
       background: #afa;
     }
```

```
    section {
      width: 75%;
      float: left;
      padding: 10px;
      background: #ccc;
    }
    .clear {
      clear: both;
    }

footer {
          background: #aaf;
          padding: 10px;
        }
      </style>
```

```html
<body>
   <div class="container">
     <header>header</header>
     <nav>
       <ul>
         <li><a href="">Home</a></li>
         <li><a href="">About</a></li>
         <li><a href="">Career</a></li>
         <li><a href="">Contact</a></li>
       </ul>
       <div class="clear"></div>
     </nav>
     <main>
       <aside>aside</aside>
       <section>section</section>
       <div class="clear"></div>
     </main>
     <footer>footer</footer>
   </div>
 </body>
```



CSS Border

CSS borders are used to define the boundaries around elements and can significantly enhance the layout and design of a webpage. Here's a detailed breakdown of border properties

Property	Description	Values	Example
border	Shorthand for setting width, style, and color of the border.	<border-width> <border-style> <border-color>	border: 2px solid red;
border-width	Sets the width of the border.	thin, medium, thick, or any length value (e.g., 5px, 1em).	border-width: 5px;
border-style	Defines the style of the border.	none, solid, dashed, dotted, double, groove, ridge, inset, outset.	border-style: dashed;
border-color	Sets the color of the border.	Named colors, hex (#ffffff), RGB, or HSL values.	border-color: #3498db;
border-top	Shorthand for the top border. Sets width, style, and color.	<border-width> <border-style> <border-color>	border-top: 3px solid blue;
border-right	Shorthand for the right border. Sets width, style, and color.	<border-width> <border-style> <border-color>	border-right: 2px dashed green;
border-bottom	Shorthand for the bottom border. Sets width, style, and color.	<border-width> <border-style> <border-color>	border-bottom: 4px dotted red;
border-left	Shorthand for the left border. Sets width, style, and color.	<border-width> <border-style> <border-color>	border-left: 5px double black;
border-radius	Rounds the corners of the border.	Length values (e.g., 10px, 50% for circular corners).	border-radius: 10px;
border-image	Allows the use of an image as the border.	url(image) slice repeat stretch	border-image: url(border.png) 30;
border-spacing	Sets spacing between borders of adjacent cells in a table.	Length values (e.g., 10px, 1em).	border-spacing: 5px;
border-collapse	Specifies whether table cell borders should collapse into a single border or remain separate.	collapse, separate.	border-collapse: collapse;

Example

Example Box with All Border Properties

Table with Borders

Property	Description	Example
border	Shorthand for setting all border properties.	`border: 2px solid blue;`
border-radius	Rounds the corners of the element.	`border-radius: 15px;`
border-image	Sets an image as the border.	`border-image: url('border.png') 30 stretch;`

```
<body>
    <div class="example-box">Example Box with All Border
Properties</div>

    <h3>Table with Borders</h3>
    <table>
      <tr>
        <th>Property</th>
        <th>Description</th>
        <th>Example</th>
      </tr>
      <tr>
        <td>border</td>
        <td>Shorthand for setting all border properties.</td>
        <td><code>border: 2px solid blue;</code></td>
      </tr>
      <tr>
        <td>border-radius</td>
        <td>Rounds the corners of the element.</td>
        <td><code>border-radius: 15px;</code></td>
      </tr>
      <tr>
        <td>border-image</td>
        <td>Sets an image as the border.</td>
        <td><code>border-image: url('border.png') 30
stretch;</code></td>
      </tr>
    </table>
  </body>
```

```
<style>
      .example-box {
        width: 300px;
        height: 150px;
        border: 5px solid #3498db; /* Shorthand */
        border-radius: 15px; /* Rounded corners */
        /* border-image: url("./smallImage.jpg") 30 stretch; */
        border-top: 4px dashed red; /* Individual side */
      }

      table {
        border-collapse: collapse;
        width: 100%;
        border: 2px solid black;
      }

      th,
      td {
        border: 1px solid #ddd;
        padding: 8px;
      }

      th {
        border-bottom: 2px solid #3498db;
        background-color: #f2f2f2;
      }
    </style>
```

CSS Positions

1. **CSS Position** means where an element should be located.
2. **Positions** are used to specify the location of an HTML element on a page layout. By default, all elements in html are positioned statically .
3. Static position is position on an element to page or document flow.
4. By using **css positions**, they can be moved from their position relatively or absolutely.

Type of CSS positions

- Static
- Relative
- Absolute
- Fixed
- Sticky

Position Properties

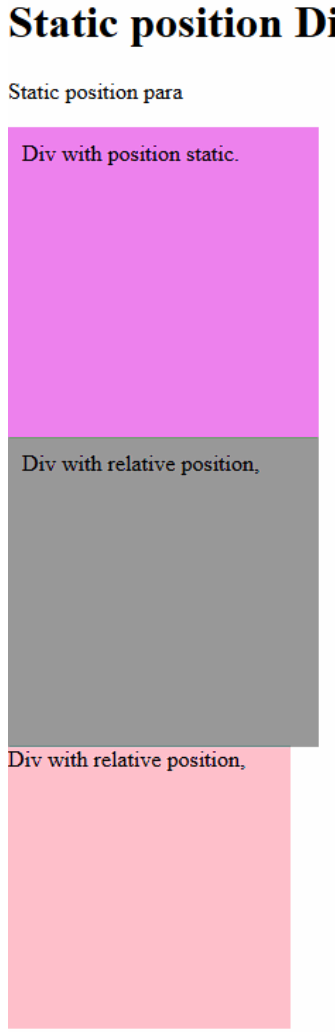
These 6 position properties are applicable to non static positions only. Except position static, any position can use them.

Property	Units	Role
Left	px, %, em, rem	To move element from left
Right	px, %, em, rem	To move element from right
Top	px, %, em, rem	To move element from top
bottom	px, %, em, rem	To move element from bottom
inset	px, %, em, rem	shortcut for top, right, bottom and left.
z-index	number	To change z axis of element

1. **Static position** is the default position of all HTML elements (*except dialog tag*). Static means position according to the HTML Order or position taken by an element by its own.
2. In static position, there will be no impact when using top, bottom, left, right and z-index properties.

```
  <body>
    <div>
      <h1>Static position Div</h1>
      <p>Static position para</p>
    </div>

    <div class="box1">Div with position
static.</div>
    <div class="box2">
      Div with static position
    </div>
    <div class="box3">
      Div with static position
    </div>
  </body>
```

```
 <style>
       .box1 {
         width: 200px;
         height: 200px;
         background:
violet;
         padding: 10px;
       }
       .box2 {
         width: 200px;
         height: 200px;
         background: #999;
         padding: 10px;
       }
       .box3 {
         width: 200px;
         height: 200px;
         background: pink;
       }
    </style>
```



1. **Relative Position** means position of an HTML element with respect to its actual position. **Relative position** can overlap over another elements. Relative elements moves, but their space remains.
2. By default, relative elements are above static elements on z axis. But when two or more elements are relative, the last relative element in stack will come to the top(z axis).

```
    <body>
    <!-- relative -->
    <h1>Relative</h1>
    <div class="box1">Div with position
static.</div>
    <div class="box2">Div with relative
position, left 200px and top 100px</div>
    <div class="box3">Div with relative
position, left 200px</div>
    <div class="box4">
      Div with relative position, left
200px and bottom 150px
    </div>
  </body>
```

```
<style>
      .box1 {
        width: 200px;
        height: 200px;
        background: violet;
        padding: 10px;
      }
      .box2 {
        width: 200px;
        height: 200px;
        background: #999;
        padding: 10px;
        position: relative;
        left: 100px;
        top: 100px;
      }
```

```
      .box3 {
        width: 200px;
        height: 200px;
        background: pink;
        position: relative;
        left: 100px;
      }
      .box4 {
        width: 200px;
        height: 200px;
        background: blue;
        position: relative;
        left: 100px;
        bottom: 150px;
      }
    </style>
```

Relative

Div with position static.

Div with relative position, left 200px and top 100px

Div with relative position, left 200px

Div with relative position, left 200px and bottom 150px

1. **Position Absolute** means position of an HTML element in a specific location outside of normal document flow. **Position Absolute** remove the element from normal document flow and shows all of its own.

2. An element with **position absolute** can be placed anywhere on the page using properties top, left, right and bottom. Absolute Element can also overlap other elements by using **z-index** property.

3. The left, top, bottom and right of absolute element is with respect to nearest non-static parent or viewport.

Properties of Position Absolute

1.Leave no space in normal document flow.

2.Occupy width of content.

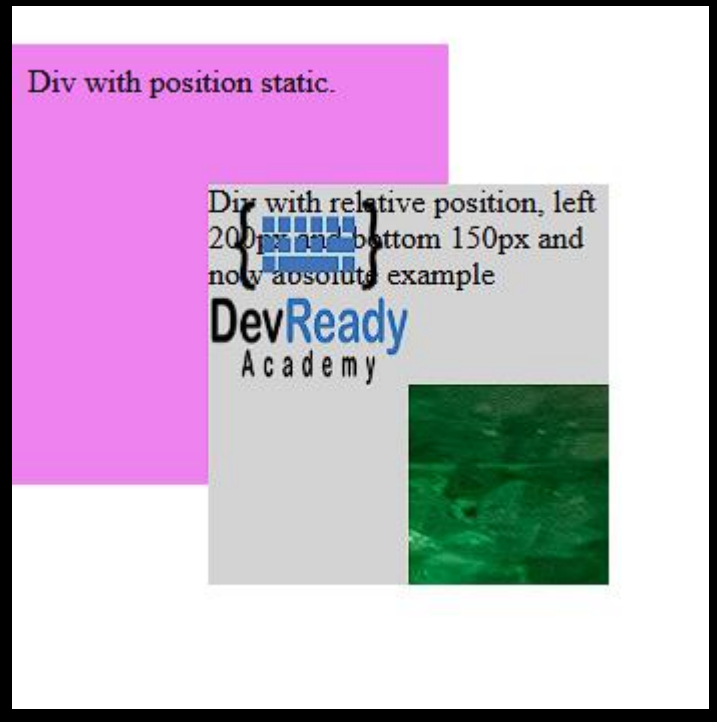
3.Can move relative to its nearest non static parent.

4.width 100% and height 100% of absolute element is actually 100% of its non static parent element.

5.If all parent elements are static, absolute element moves relative to viewport window.

```
<style>
      .box1 {
        width: 200px;
        height: 200px;
        background: violet;
        padding: 10px;
      }

      .box4 {
        width: 200px;
        height: 200px;
        background: lightgray;
        position: relative;
        left: 100px;
        bottom: 150px;
      }
      .imgAbsolute1 {
        position: absolute;
        top: 100px;
        left: 100px;
      }
      .imgAbsolute2 {
        position: absolute;
        top: 0px;
        left: 0px;
      }
    </style>
```

```
 <body>
    <!-- relative -->
    <h1>Relative</h1>
    <div class="box1">Div with position static.</div>
    <!-- <div class="box2">Div with relative position, left 200px and top
100px</div>
    <div class="box3">Div with relative position, left 200px</div> -->
    <div class="box4">
      Div with relative position, left 200px and bottom 150px and now
absolute
      example
      <img
        src="./smallImage.jpg"
        class="imgAbsolute1"
        width="100"
        height="100"
      />
      <img
        src="./logoLight.png"
        class="imgAbsolute2"
        width="100"
        height="100"
      />
    </div>
  </body>
```



1. **Fixed Position** means position of an HTML element with respect to **viewport**. It doesn't move even if we scroll window up or down.
   For Exp: the Top Menu of this web-page is positioned fixed to the top. It doesn't move even after scrolling down or up.

2. We can use top, left, bottom, right and z-index after giving position fixed to an HTML element.

```
<nav class="navbarcontainer">
    <ul>
      <li>HOME</li>
      <li>ABOUT</li>
      <li>Contact</li>
    </ul>
  </nav>
```

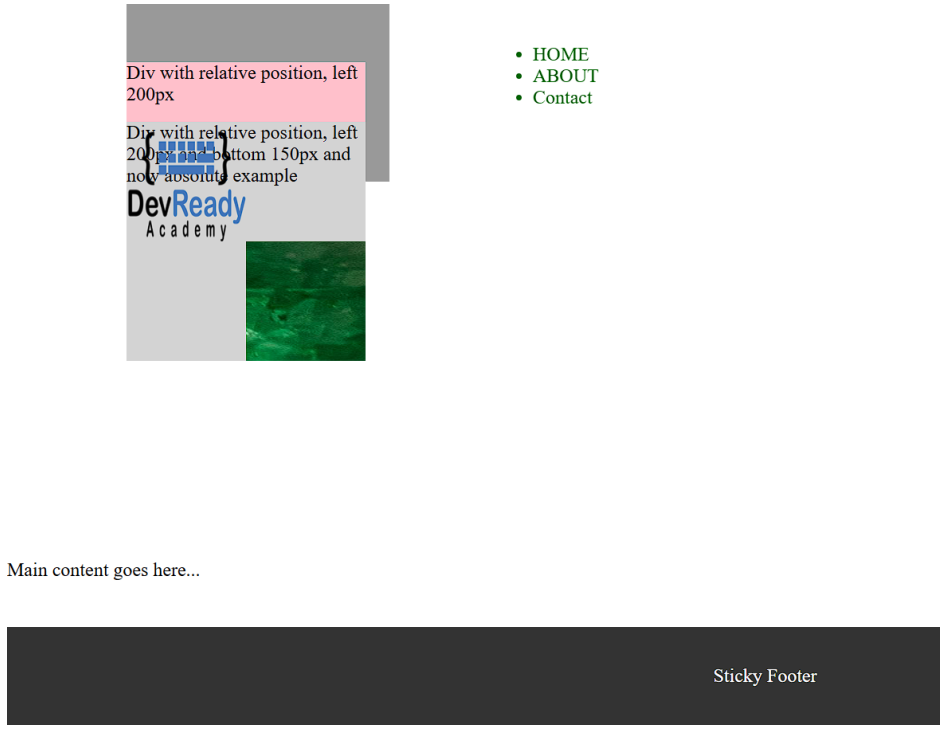
```
.navbarcontainer {
    position: fixed;
    z-index: 100;
    color: darkgreen;
    left: 400px;
}
```

1. **Position Sticky** was introduces in CSS3. Sticky element normally behave like relative positioned, but once reaching the corner or viewport, it sticks. That's why its called Sticky.

2. Its compulsory to use either top, bottom, left or right to sticky element. Otherwise it will not stick.

```
  <footer>
      <p>Sticky Footer</p>
    </footer>
```

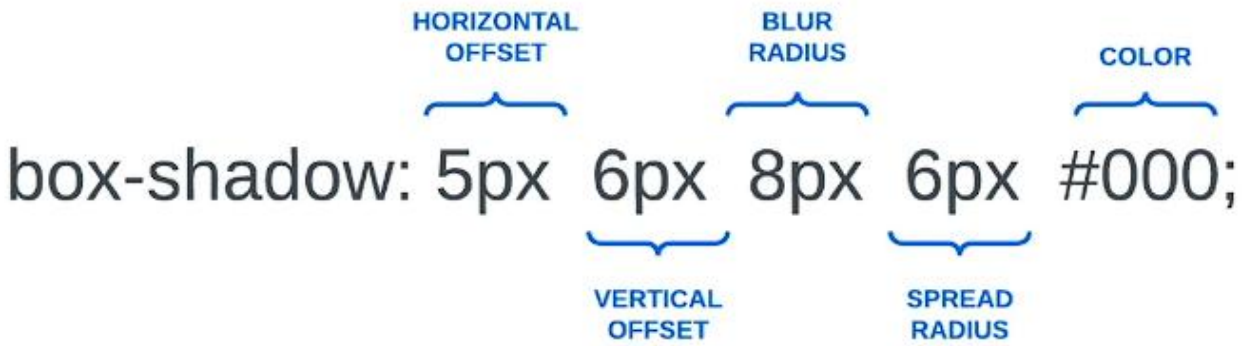
```
  .content {
        height: 100vh;
      }

      footer {
        position: sticky;
        bottom: 0; /* Stick the footer to the bottom
*/
        background-color: #333;
        color: white;
        padding: 1rem;
        }
```



CSS Box shadow

CSS3 Box Shadow is a new property to add **shadow effects** to any html Element. You can choose your own **color**, **offset**, **blur**, **spread**, **offset** and **repetition**.

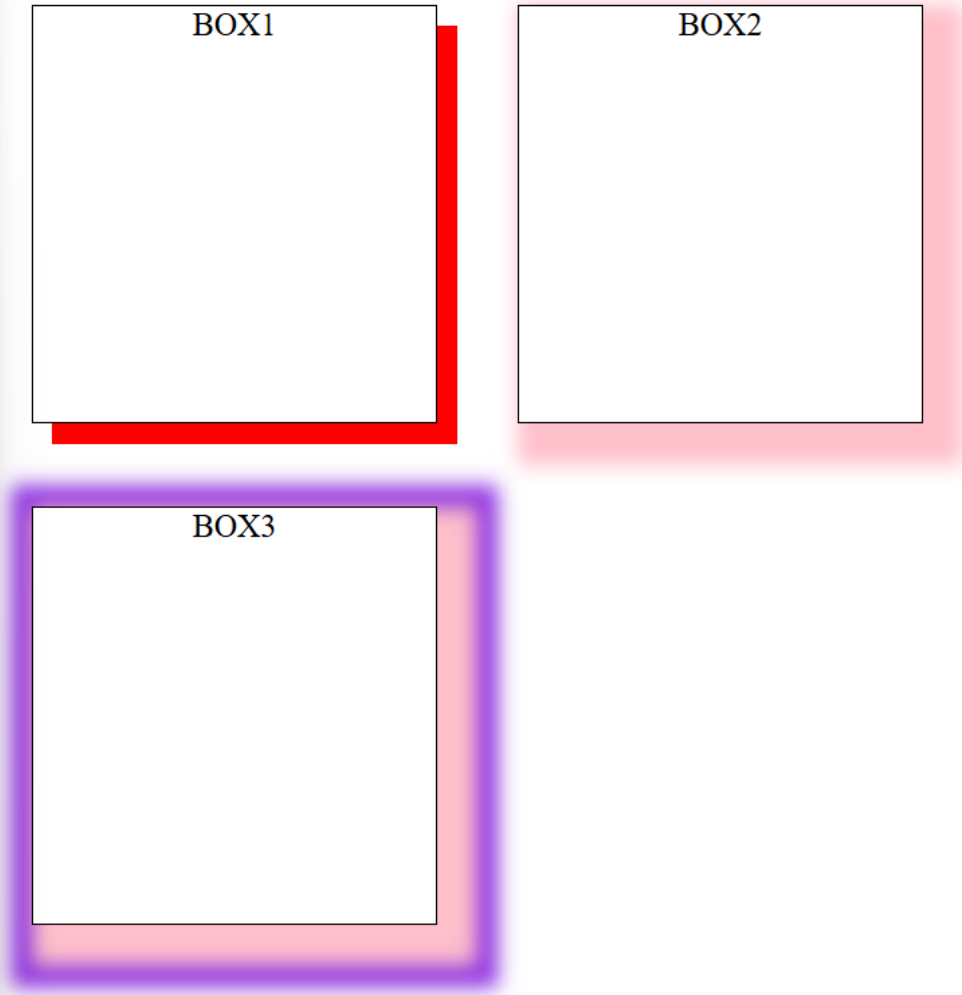
Box Shadow



- **Horizontal & Vertical Offset –** How far from the element each way
- **Blur Radius (Optional) –** How blurry the shadow is
- **Spread Radius (Optional) –** How much the shadow should grow or shrink

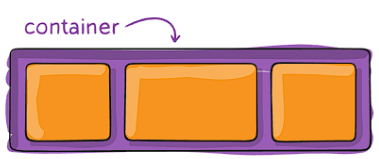
```html
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Document</title>
<style>
  .box1,
  .box2,
  .box3 {
    width: 200px;
    height: 200px;
    border: 1px solid black;
    float: left;
    margin-right: 20px;
    margin-top: 40px;
    margin-left: 20px;
    text-align: center;
  }
  .box1 {
    box-shadow: 10px 10px red;
  }
  .box2 {
    box-shadow: 10px 10px 10px 10px pink;
  }
  .box3 {
    box-shadow: 10px 10px 10px 10px pink, 10px 10px 10px 20px blueviolet;
  }
</style>
</head>
<body>
  <div class="box1">BOX1</div>
  <div class="box2">BOX2</div>
  <div class="box3">BOX3</div>
```

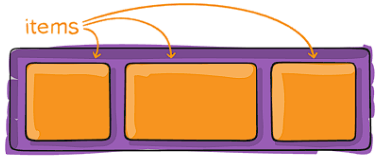
OUTPUT DEBUG CONSOLE TERMINAL PORTS

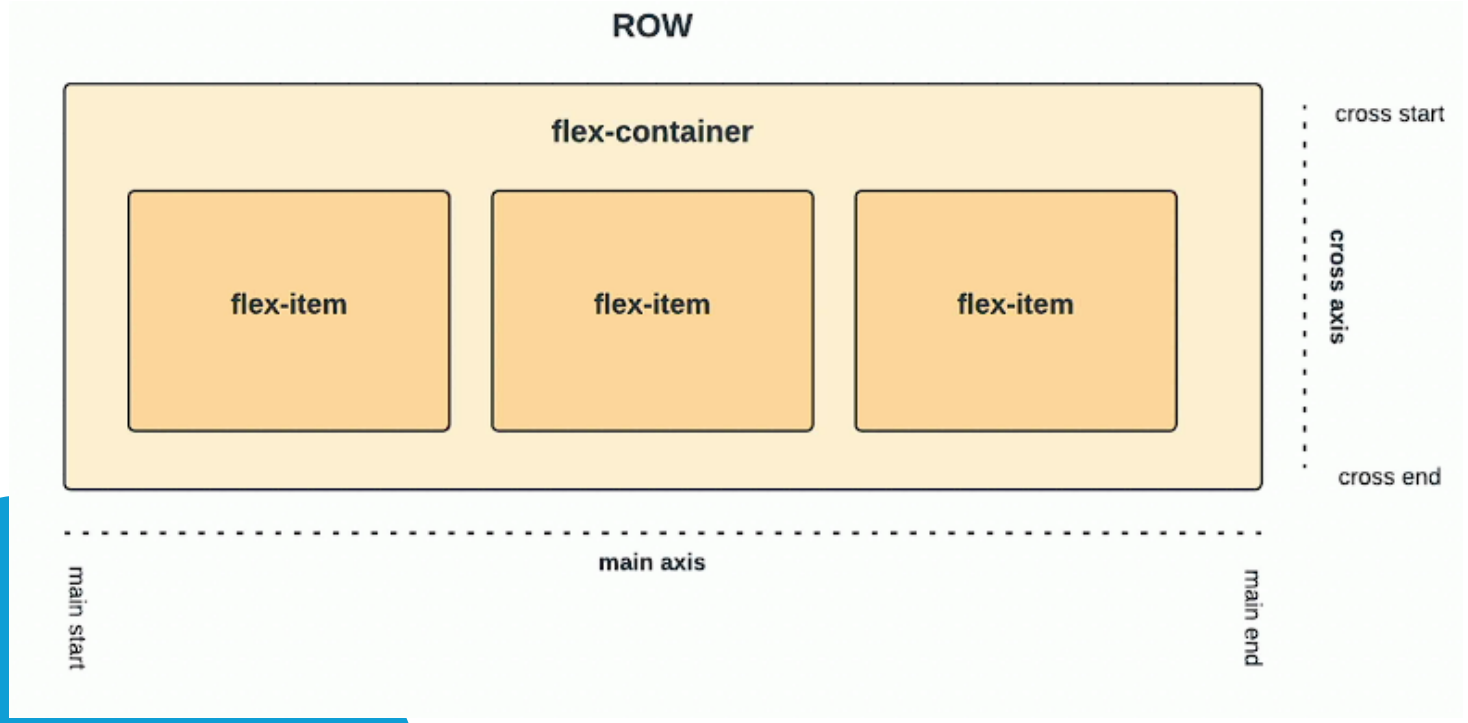


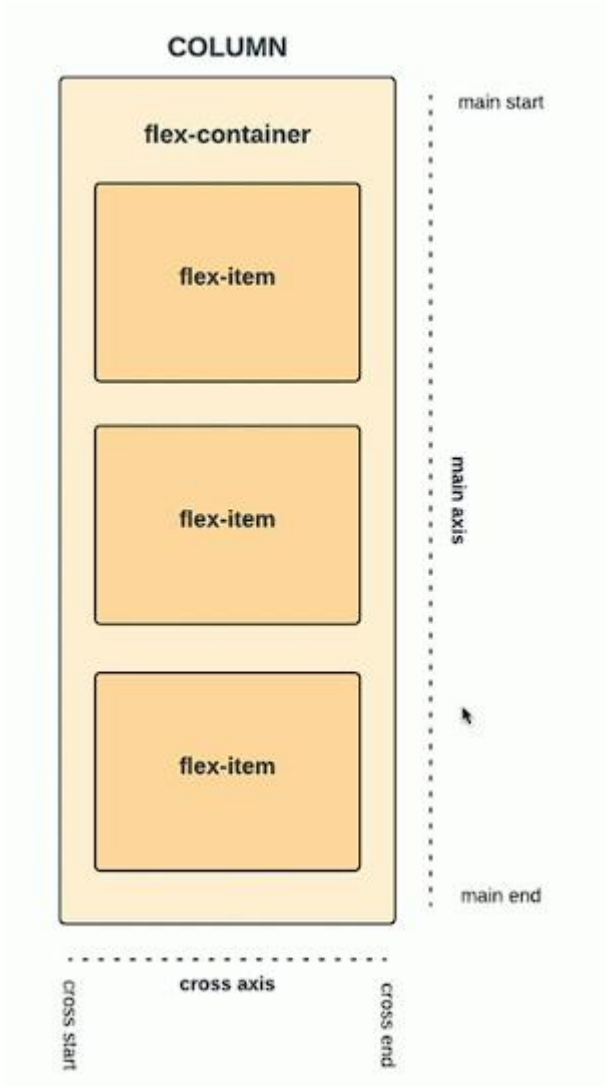
CSS FlexBox

1. Flexbox, or the **CSS Flexible Box Layout Module**, is a layout model designed to provide a more efficient way to align, distribute, and space elements in a container, even if their size is dynamic or unknown.
2. It's ideal for creating responsive designs and controlling the layout direction and alignment of items.
3. Its one-dimensional layout model
4. Flex items are put in flex container









Properties for the Parent

• Display Flex

- **Display flex** is the property of flex container to use flexbox.
- **CSS Display** property can have value **flex** or **inline-flex**.
- By using display flex or inline-flex to parent container, the children automatically enable flex content.
- By default all the items will be **arranged row wise**. Only with a single property display:flex

```
<body>
  <div class="mainContainer">
    <div class="box1">
      <h1>BOX1</h1>
    </div>
    <div class="box2">
      <h1>BOX2</h1>
    </div>
    <div class="box3">
      <h1>Box 3</h1>
    </div>
  </div>
</body>
```

```
.mainContainer {
        background: violet;
        margin: 2rem;
        display: flex;
    }
```

a.Flexbox basics.

Flex-direction

Flex Direction property is given to **flex container** to **change direction of flex items**. By default, flex direction is row.

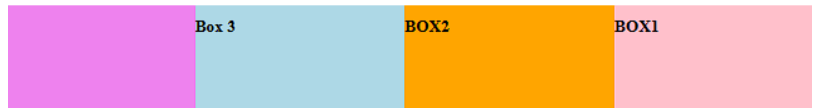
1. **row *(default)***
2. **row-reverse**
3. **column**
4. **column-reverse**

flex-direction:row(default)

```
.mainContainer {
        background: violet;
        margin: 2rem;
        display: flex;
    }
```

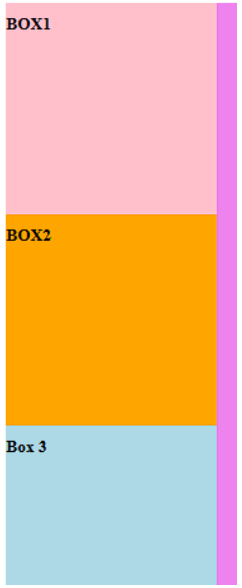
flex-direction:row-reverse

```
.mainContainer {
        background: violet;
        margin: 2rem;
        display: flex;
        flex-direction: row-reverse;
    }
```



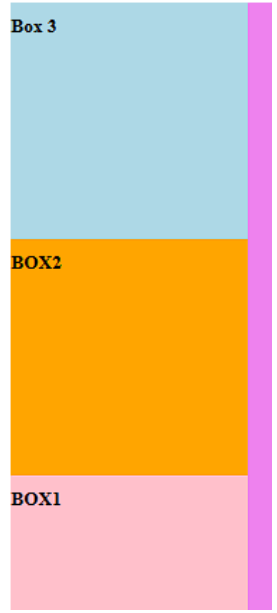
flex-direction:column

```
.mainContainer {
        background: violet;
        margin: 2rem;
        display: flex;
        flex-direction: column;
    }
```



flex-direction:column-reverse

```
.mainContainer {
        background: violet;
        margin: 2rem;
        display: flex;
        flex-direction: column-reverse;
    }
```



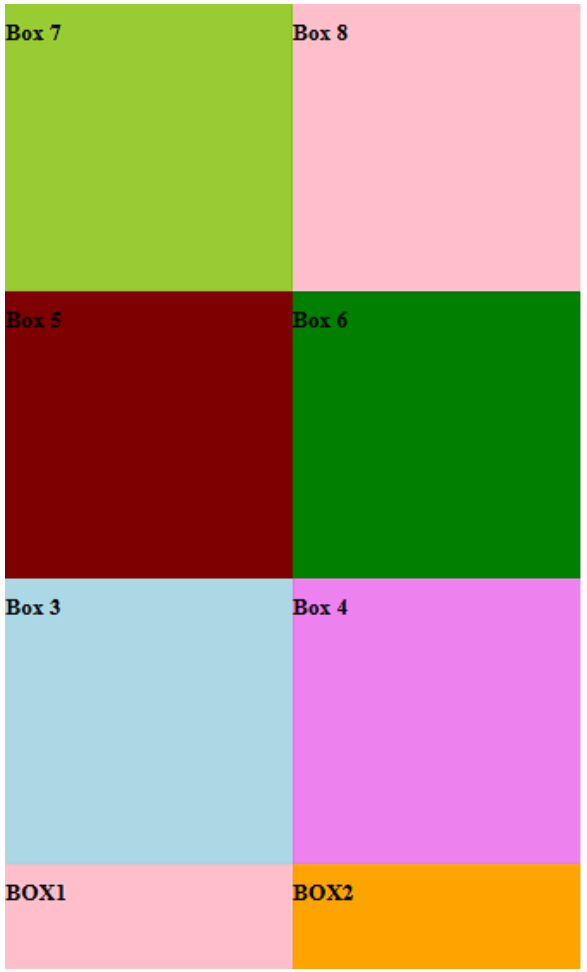
flex-wrap

By default, flex items will all try to fit onto one line. You can change that and allow the items to wrap as needed with this property..

.main.container { flex-wrap: nowrap | wrap | wrap-reverse; }

flex-wrap:nowrap(default)

flex-wrap: wrap-reverse(default)



flex-wrap:nowrap(default)

flex-flow

This is a shorthand for the flex-direction and flex-wrap properties, which together define the flex container's main and cross axes. The default value is row nowrap.

```
.maincontainer  { flex-flow: column wrap; }
```

b.Align and justify items.

justify-content

1. Aligns items along the main axis.
2. This can adjust items to left, center, right or add space in between.

   justify-content values—

   1 flex-start

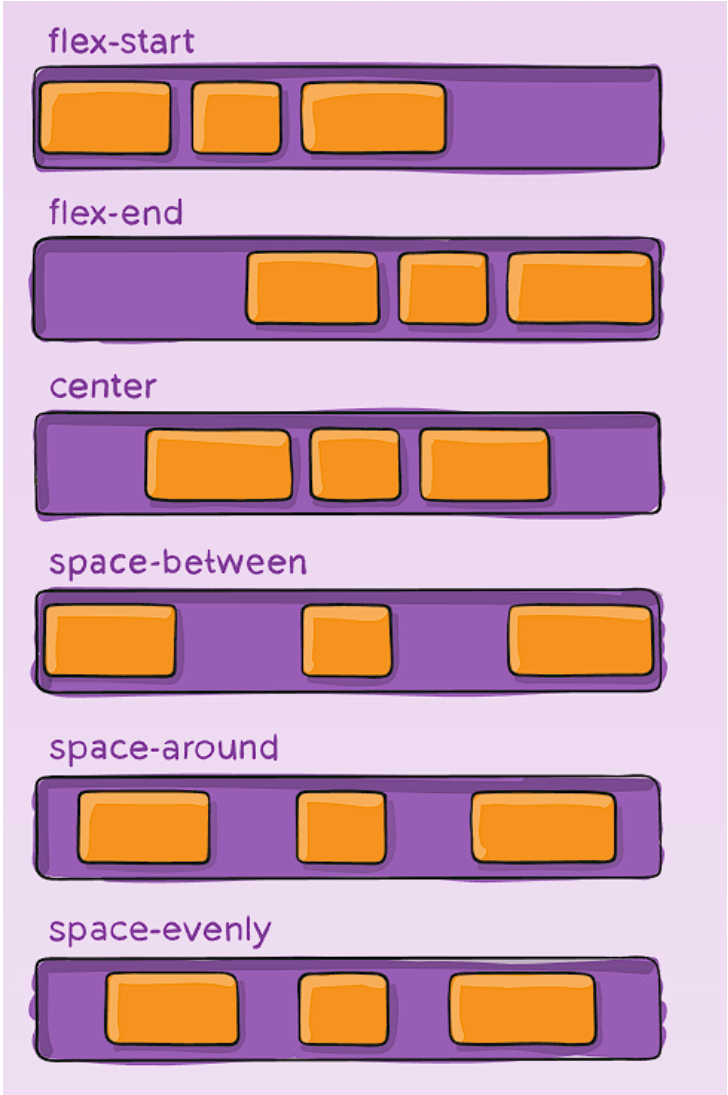
   2. Center

   3. flex-end

   4. space-between

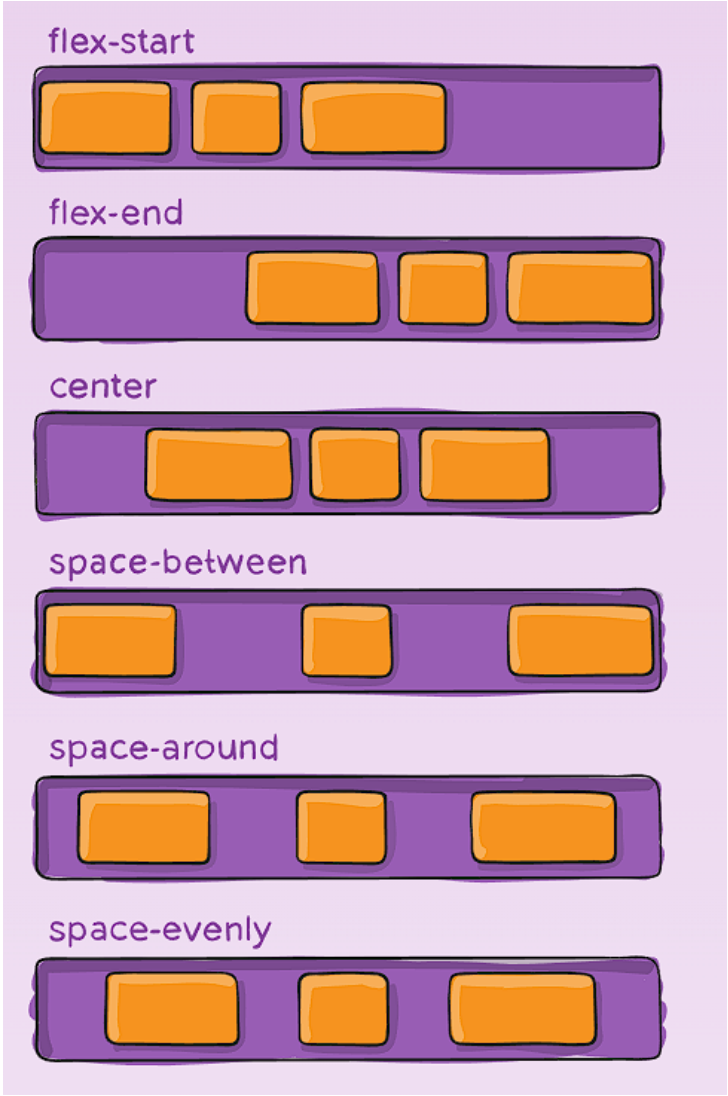
   5. space-around

   6. space-evenly.



1. **flex-start** (default): items are packed toward the start of the flex-direction.
2. **flex-end**: items are packed toward the end of the flex-direction.
3. **center**: items are centered along the line
4. **space-between**: items are evenly distributed in the line; first item is on the start line, last item on the end line.
5. **space-evenly**:
   - Distributes the space between the items evenly.
   - The spaces before, between, and after the items are equal.
6. **space-around**:
   - Distributes the space around the items.
   - Each item will have an equal amount of space around it.

   However, the space between two adjacent items is twice as much as the space at the ends of the container.



align-items

1. This defines the default behavior for how flex items are laid out along the **cross axis** on the current line.
2. **Align-items** property define the behavior of how flex item laid out across horizontal axis. By-default, the value is stretch.

   justify-content values—

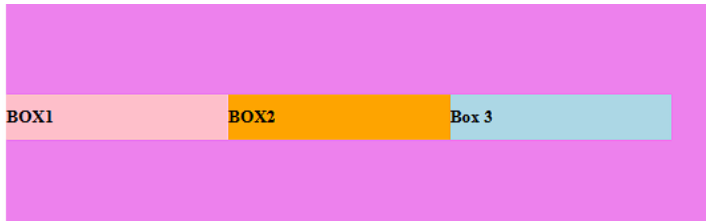
   1 flex-start/start/self-start

   2. center

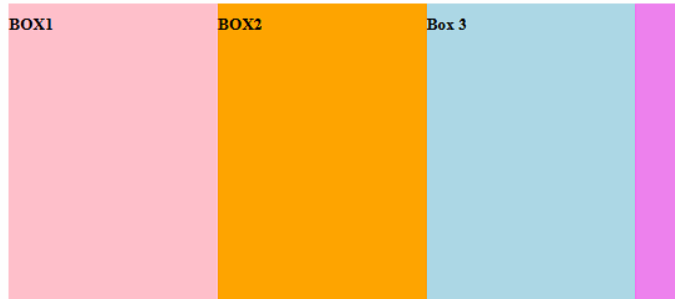
   3. flex-end/end/self-end

   4. stretch.

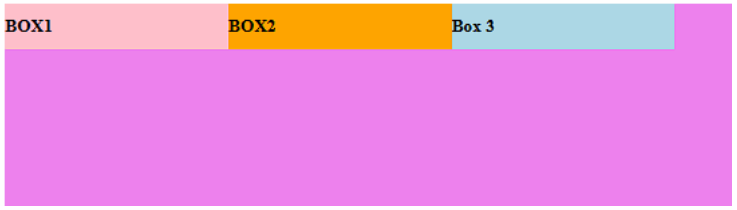
center



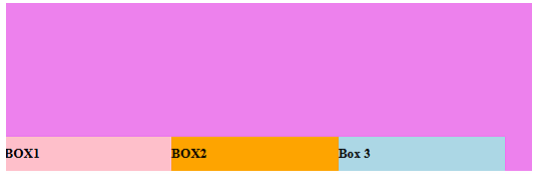
Stretch(default)



Flex-start

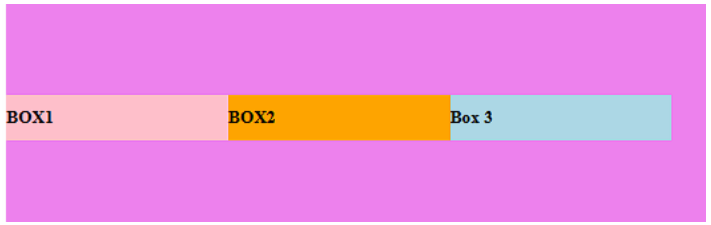


Flex-end

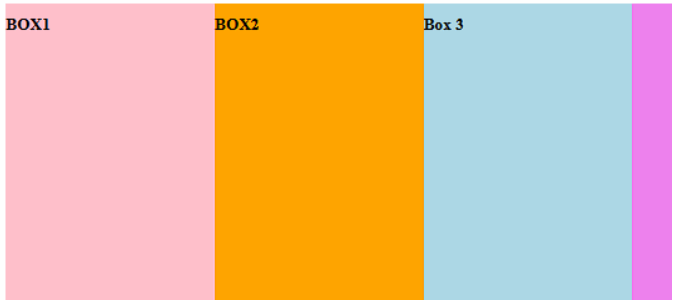


1. **flex-start** : items are packed toward the start of the flex-direction.
2. **flex-end**: items are packed toward the end of the flex-direction.
3. **center**: items are centered along the line
4. **Stretch:**Items stretch to fill the container along the cross axis, unless a specific height or width is set.

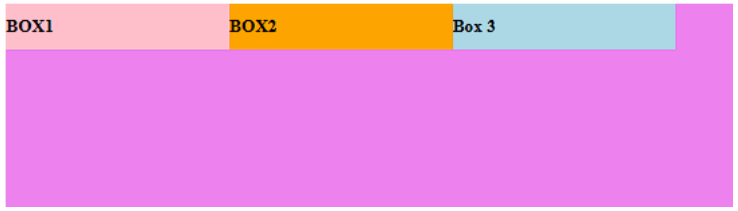
center



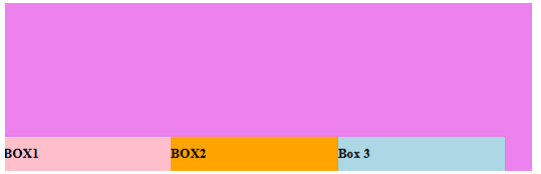
Stretch(default)



Flex-start



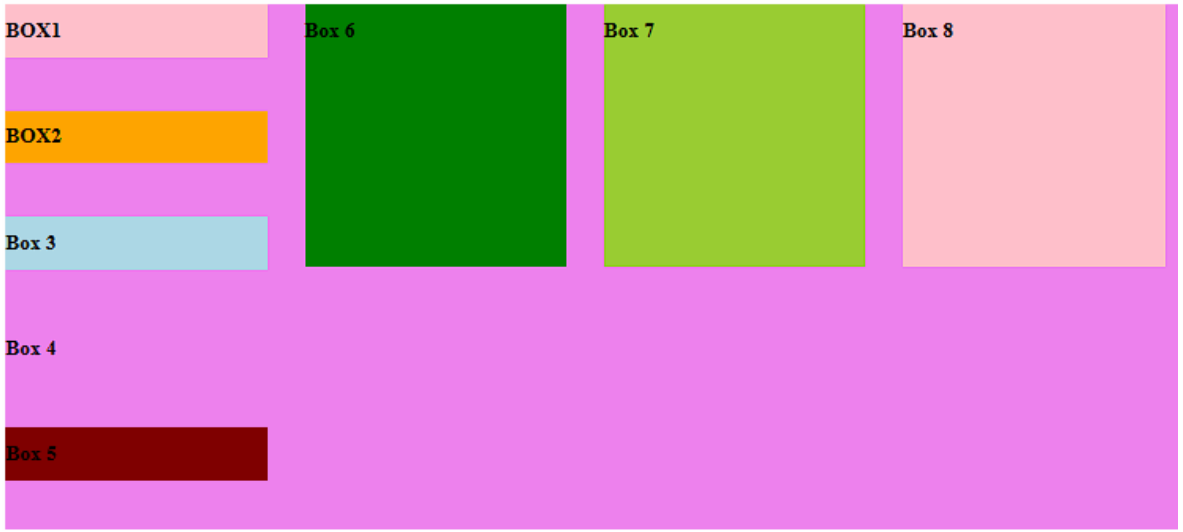
Flex-end



c.CSS properties
And dynamic sizing.

gap

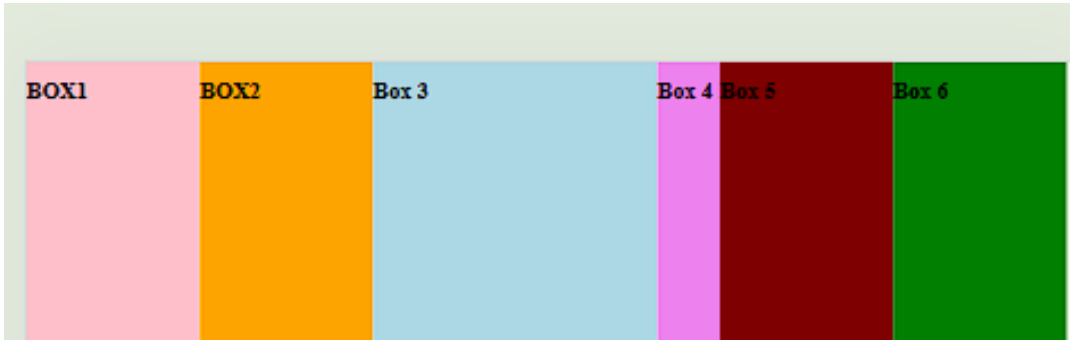
1. This defines the gap between flex-items.
2. **gap/row-gap/column-gap**




```css
.mainContainer {

    display:flex;
        /* only gap between rows */
     row-gap: 80px;
     /* only gap between column */
     column-gap: 300px;
     /* both row and column */
     gap: 30px;
     /* row column */
     gap: 80px 30px;
  }
```

Flex-shrink

1.flex-shrink is a CSS property used in flexible layouts to define how much a flex item should shrink relative to the rest of the flex items in the same container when there is not enough space .
2.flex-shrink :1 (by default)
3.flex-shrink:0 – the item where it is added will not shrink.
4.flex-shrink:2 – the item will shrink double of other items.




```
.box3 {
        background: lightblue;
        width: 400px;
        height: 400px;
        /* if shrink 0 then no shrink */
        flex-shrink: 0;
      }
      .box4 {
        background: violet;
        width: 400px;
        height: 400px;
        /* if shrink 2 then  shrink will be double */
        flex-shrink: 2;
      }
```

Flex-grow

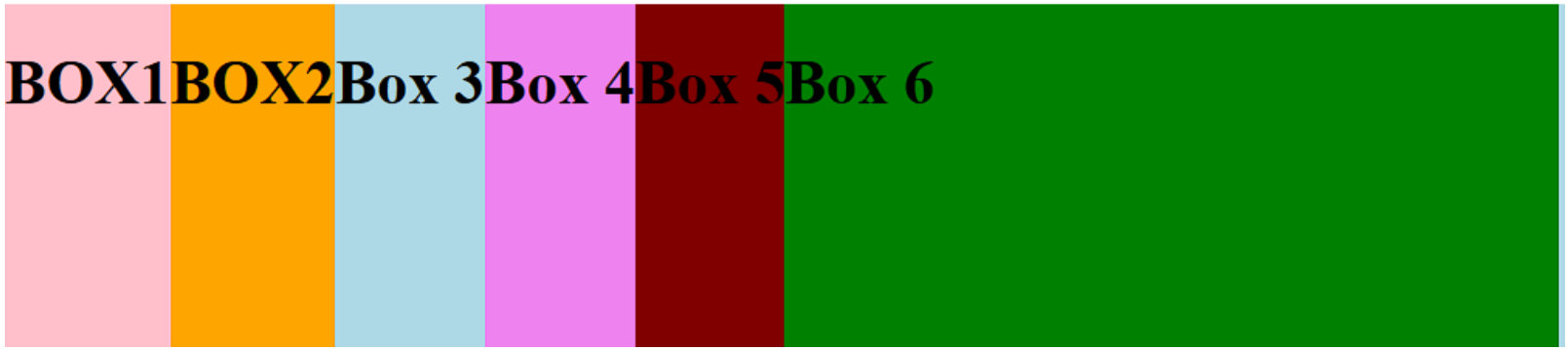
1. flex-grow is a CSS property that defines how much a flex item will grow relative to the other items in a flex container. It determines the ability of a flex item to fill the available space in the container.
2.flex-grow :0 (by default)

```
.box3 {
      background: lightblue;
      /* if grow 1 then this item will grow and take rest space */
      flex-grow: 1;
    }
    .box4 {
      background: violet;
      /* if add in 2 flex grow they will occupy the existing space */
      flex-grow: 1;
    }

    /* if add in 2 flex grow they will occupy the existing space */

    .box5 {
      background: maroon;
      /* twice of other item space */
      flex-grow: 2;
    }
```

Flex-basis

1. flex-basis is a CSS property that defines the initial main size of a flex item. It determines the default size of an element before any space distribution is applied by the flex-grow or flex-shrink properties.

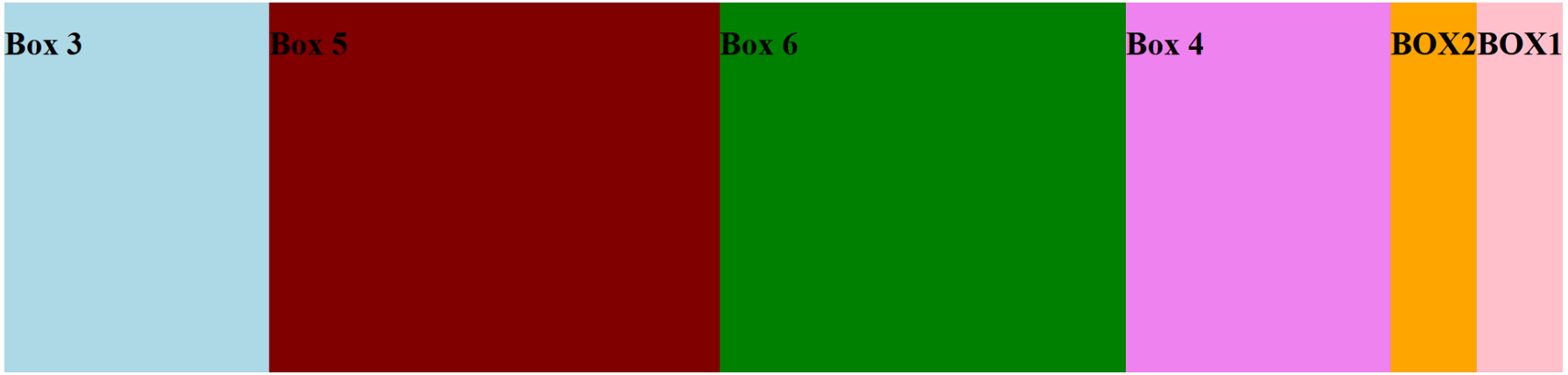
```css
.box6 {
    background: green;
    flex-basis: 400px;
    flex-shrink: 0;
}
```



order

The order property in CSS is used in Flexbox and CSS Grid layouts to control the order in which flex or grid items appear in the container. It allows you to change the visual order of items without altering the HTML structure

```css
.box1 {
      background: pink;
      height: 400px;
      order: 3;
    }
    .box2 {
      background: orange;
      height: 400px;
      order: 2;
    }
    .box3 {
      background: lightblue;
      /* if grow 1 then this item will grow and take rest space */
      flex-grow: 1;
    }
    .box4 {
      background: violet;
      order: 1;
      /* if add in 2 flex grow they will occupy the existing space */
      flex-grow: 1;
    }
```



Games for display :flex

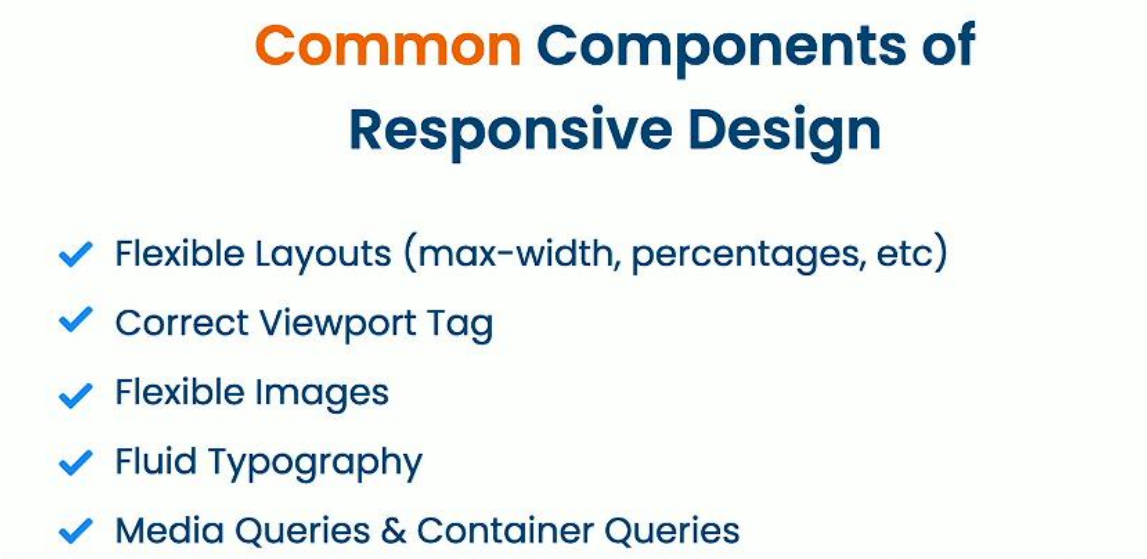
https://flexboxfroggy.com/

https://codingfantasy.com/games/flexboxadventure

Responsive.

Why need a responsive design ????

Reason	Description
Enhanced User Experience	Ensures consistent, accessible, and user-friendly interfaces across devices.
Improved Mobile Traffic	Caters to increasing mobile users, boosting engagement and retention.
Cost-Effectiveness	Single codebase reduces development and maintenance costs.
SEO Benefits	Improves search rankings and reduces bounce rates by being mobile-friendly.
Future-Proofing	Adapts to new devices and screen sizes, ensuring longevity.
Increased Conversion Rates	Enhances user engagement, leading to higher conversion rates.
Competitive Advantage	Keeps the site modern, improving brand perception and staying ahead of competitors.



a. Flexible layout and percentages

Percentages and Max Width: Elements are set to use percentage widths for flexibility, with max-width to limit how large they can grow.

Max Height: You can use max-height to adjust padding or element sizes when the screen height is limited, improving usability on devices like mobile phones.

```html
<body>
  <div class="container">
    <img src="./smallImage.jpg" />
    <h1>
      Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nesciunt
      eveniet temporibus provident impedit sequi quod quam adipisci
      consectetur ipsa exercitationem?
    </h1>
  </div>
</body>
```

```html
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Document</title>
  <style>
    .container {
      /* width: 1000px; */
      max-width: 1000px;
      width: 60%;
      margin: auto;
    }
    /* if we add only width then in mobile screens it will show scroll */
    /* for these we can use max-width or width in percentage */

    /* if image is max-width is 100 them image will not move out of width */
    img{
      max-width: 100%;
    }

  </style>
</head>
```

vh (viewport height) and vw (viewport width) are CSS units used to size elements relative to the dimensions of the viewport, which is the visible area of a web page. Here’s how they work:

- **1vh**: Represents 1% of the viewport's height. For example, if the viewport height is 800px, 1vh would be 8px.
- **1vw**: Represents 1% of the viewport's width. For example, if the viewport width is 1200px, 1vw would be 12px

Use Cases:

- **Full-page sections**: Use 100vh to make an element (like a hero section) take up the full height of the viewport.
- **Responsive layouts**: vh and vw can create responsive elements that adjust their size based on the viewport, avoiding fixed pixel values.
- **Typography**: Sometimes used for fluid typography that scales with the viewport

b.Media queries

1. Media queries are a CSS feature that allows you to apply styles based on specific conditions such as screen size, resolution, orientation, and more.
2. They are fundamental for building responsive designs that adapt to different devices and screen sizes.

```
@media screen and (max-width: 600px) {
  /* CSS rules */
}
```

Aspect	Description
@media Rule	Defines different style rules based on conditions.
Media Features	Common features include min/max-height, min/max-width, and orientation.
Media Types	Includes types like screen, print, speech, etc.

Breakpoint	Device Type
576px	Smartphones
768px	Tablets
992px	Desktop
1024px	Landscape
1200px	Desktop/Widescreen

CSS Selectors

CSS Selectors

```
h1 {
    color:red;
}
```

Type Selectors

```
.heading
{
 color:red;
 }
```

Class Selectors

```
#heading
{
 color:red;
 }
```

ID Selectors

```
h1,p {
    color:red;
}
```

Multiple Selectors

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

universal Selectors

```
input[type="tex
t"]
 {
  color:red;
  }
```

Attribute Selectors

```
p::first-line
{
 color:red;
 }
```

Pseudo Element

```
a:visited {
    color:red;
}
```

Pseudo Classes

Attribute Selectors

styles.css

```
a[target="_blank"] {
  background-color: yellow;
}

input[type="text"]{
  background-color: yellow;
}


input[type="button"]{
background-color: yellow;
}
```

The [attribute] selector is used to select elements with a specified attribute.

`[attribute="value"]` -- select elements with the specified attribute, whose value equal to specified value.

`[attribute^="value"]` -- select elements with the specified attribute, whose value starts with the specified value.

`[attribute$="value"]` – select elements with the specified attribute, whose value ends with the specified value

`[attribute*="value"]` – select elements with the specified attribute, whose value contains with the specified value

Example

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet“
href="styles.css">
</head>
<body>
<a href="http://www.disney.com"
target="_blank">disney.com</a>
</body>
</html>
```

Pseudo Element

```
label.is-required::after {
  content: '*';
  color: red;
  margin-left: 5px;
  font-size: 1.2rem;
}
```

Image overlay on hero

```
.hero::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: rgba(0, 0, 0, 0.5);
}
```

```
p::first-line {
 color: #ff0000;
}

p:: first-letter  {
 color: #ff0000;
}

h1::before {
  content: url(smiley.gif);
}
h1::after {
  content: url(smiley.gif);
}

p::selection {
 color: red;
 background: yellow;
}

Input :: placeholder{
color:lightblue
}
```

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

`p::first-line` -- add a special style to the first line of a text.

`p::first-letter` -- add a special style to the first letter of a text.

`::before`- insert some content before the content of an element.

`::after`- insert some content after the content of an element.

::placeholders

::required

::file-selector-button

Example

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet“
href="styles.css">
</head>
<body>
<a href="http://www.disney.com"
target="_blank">disney.com</a>
</body>
</html>
```

Pseudo Class

```css
/* Using pseudo classes with pseudo elements */
.container p:first-of-type::first-letter {
  font-weight: bold;
  font-size: 2rem;
  color: red;
}
```

```
/* unvisited link */
a:link {
 color: red;
}

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

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

/* selected link */
a:active {
 color: black;
}

button:hover{
 color: black;
}

ul li:nth-child(even/2/odd){ color:#a94442}
}
```

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

:nth-child(n)-Targets only **nth child** of parent element.

:nth-child(odd/even)--Targets only **odd children** of parent element.

(first-child/last-child/nth-child(4n) – (every 4th item)

Same is (first-of-type/last-of-type/nth-of-type)

Example

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet“
href="styles.css">
</head>
<body>
<a
href="http://www.disney.com"
target="_blank">disney.com</
a>
<ul> <li>List 1</li> <li>List 2</li>
<li>List 3</li> <li>List 4</li> <li>List
5</li> </ul>
 </body>
</html>
```

Pseudo-classes and Pseudo-elements

Pseudo-classes enable you to target an element when it's in a particular state(hover,active), as if you had added a class for that state to the DOM. Pseudo-elements act as if you had added a whole new element to the DOM,(::before,::firstchild) and enable you to style that.

CSS Combinators

```
ul.my-things > li  {
    color:red;
 }
//li that are direct children
```

Child combinator

```
 ul.my-things  li
{
 color:red;
 }
/* List items that are descendants of the "my-things"
list */
```

Descendant combinator

```
Div ~ span {
     color:red;
  }
//span that are all sibling of
same parent
```

Subsequent-sibling combinator combinator

```
Div + span {
     color:red;
  }
//span that are direct sibling
 of same parent and immediate
```

Next-sibling combinator

```
 div , span,p,h1
{
 color:red;
 }
//select all matching nodes
```

Selector list combinator

CSS Object fit

The object-fit property in CSS specifies how the content of an <img>, <video>, or other media elements should be resized to fit its container. It controls the alignment and scaling of the media within its containing element.

Syntax:

object-fit: value;

```css
img {
  width: 100%;
  height: 100%;
  object-fit: cover;
}
```

Values:

- **fill**: The default value. The content is resized to fill the container, potentially distorting the aspect ratio.
- **contain**: The content is resized to fit within the container while maintaining its aspect ratio. It might leave some space in the container.
- **cover**: The content is resized to fill the container while maintaining its aspect ratio. Parts of the content might be clipped to fit.
- **none**: The content is not resized and maintains its original size, which might cause overflow.
- **scale-down**: The content is sized as if none or contain was used, whichever results in a smaller size.

CSS Linear Gradient

Linear Gradients are gradients with combinations of two or more colors in linear direction. These directions are top to bottom, left to right, top left corner to bottom right corner etc. Even direction can be controlled using angle(in degrees). These gradients are used in background only, not as font color.

Linear Gradient with angle

The default linear gradient starts from top to bottom. To change direction, we can use degree from 0 to 359. The default degree in 90.

Degree in linear gradients

degree	direction
0deg	bottom to top
45deg	let-bottom to top-right
90deg	right to left
180deg	top to bottom
270deg	left to right

```
<style>
    .gradient-2{
    height:300px;
    background:linear-gradient(0deg, #ff0000, #ffffff, #0f0);text-shadow:1px 1px white";
    }
</style>

 <div class="gradient-2"> Linear Gradient 2, starting from left top corner </div>
```

```
<style>
.gradient-3{
    height:50px;
    background:linear-gradient(to right, #f00, #ff0, #0f0, #0ff, #00f, #f0f);
    }
</style>

<div class="gradient-3">Linear Gradient 3, starting from bottom with multiple colors </div>
```

Custom Properties

CSS Custom Properties (CSS variables) Syntax:

1.Defining a Custom Property:

css

scope { --variable-name: variablevalue; }

- **scope**: This is the selector where the custom property is defined. It can be any CSS selector such as a class, ID, or element.
- **--variable-name**: This is the name of the custom property. Custom properties always start with -- and can be named anything following standard CSS naming conventions.
- **variablevalue**: This is the value assigned to the custom property. It can be any valid CSS value such as a color, size, or even a complex value like a gradient.

2.Using a Custom Property:

css

selector { property: var(--variable-name); }

- **selector**: This is the CSS selector where the custom property is being used.
- **property**: This is any standard CSS property where the custom property's value will be applied.
- **var(--variable-name)**: This is the function used to retrieve the value of the custom property --variable-name. The var() function allows you to use the custom property's value wherever needed.

```css
:root {
  --main-color: #3498db;
  --padding-size: 20px;
}

body {
  background-color: var(--main-color);
  padding: var(--padding-size);
}
```

CSS filters

1. CSS filters allow you to apply graphical effects like blurring, color shifting, or brightness adjustments to elements, similar to image editing software.
2. These effects can be applied to images, text, or any other HTML elements, making them visually dynamic.

Filter Function	Description	Example
`blur()`	Applies a Gaussian blur to the element.	`filter: blur(5px);`
`brightness()`	Adjusts the brightness of the element.	`filter: brightness(150%);`
`contrast()`	Adjusts the contrast of the element.	`filter: contrast(200%);`
`grayscale()`	Converts the element to grayscale.	`filter: grayscale(100%);`
`hue-rotate()`	Rotates the hue of the element.	`filter: hue-rotate(90deg);`
`invert()`	Inverts the colors of the element.	`filter: invert(100%);`
`opacity()`	Adjusts the transparency level of the element.	`filter: opacity(50%);`

```css
.image {
  filter: blur(2px) brightness(120%)
contrast(90%);
}
```

CSS Smooth Scroll

```css
html,
body {
  font-family: 'Poppins', sans-serif;
  font-size: 18px;
  line-height: 1.6;
  scroll-behavior: smooth;
}
```

CSS Calc function

The calc() function in CSS allows you to perform calculations to determine CSS property values. This function can combine different units (like percentages and pixels) and perform mathematical operations like addition, subtraction, multiplication, and division.

Feature	Description	Example
Addition (`+`)	Adds values together, combining different units or types of values.	`width: calc(100% + 20px);`
Subtraction (`-`)	Subtracts one value from another, useful for dynamic layouts.	`width: calc(100% - 50px);`
Multiplication (`*`)	Multiplies values, typically used for scaling dimensions.	`width: calc(50px * 2);`
Division (`/`)	Divides one value by another, useful for creating proportional values.	`width: calc(100px / 2);`
Combining Units	Combines different units like percentages and pixels to create flexible designs.	`height: calc(100vh - 60px);`
Responsive Typography	Adjusts font size dynamically based on viewport size.	`font-size: calc(16px + 1vw);`
Dynamic Layouts	Allows for layout adjustments that respond to container or viewport dimensions.	`padding: calc(10px + 2%);`
Operator Precedence	Multiplication and division are calculated before addition and subtraction.	`width: calc(50px + 20px * 2);`

No Nesting

```css
.header {
  background: lightcoral;
  padding: 0 20px;
}

.header h1 {
  font-size: 4rem;
}

.header p {
  font-size: 2rem;
  margin: 25px 0px 25px;
}
```

Nesting

```css
.header {
  background: lightcoral;
  padding: 0 20px;

  h1 {
    font-size: 4rem;
  }

  p {
    font-size: 2rem;
    margin: 25px 0px 25px;
  }
}
```

CSS Transitions

1. **CSS transitions** allow you to change property values smoothly (over a given duration) instead of having the changes take effect immediately.
2. This makes for a more interactive and visually appealing user experience.

Syntax:

transition: property duration timing-function delay;

Components:

1. **property**: Specifies the CSS property to which the transition is applied (e.g., background-color, width). You can also use all to apply the transition to all properties.
2. **duration**: Specifies the duration of the transition (e.g., 2s for 2 seconds).
3. **timing-function**: Specifies the speed curve of the transition (e.g., ease, linear, ease-in, ease-out, cubic-bezier).
4. **delay**: Specifies the delay before the transition starts (e.g., 1s).
5. Transition use on

   1.:hover

   2.:focus

   3.:active

```html
<head>
    <title>Css Transition</title>
    <style>
      .hoverDivContainer {
        width: 300px;
        height: 100px;
        background: lightblue;
        transition: all 500ms ease-in-out 1s;
      }
      .hoverDivContainer:hover {
        background: lightcoral;
        width: 400px;
      }
      .buttonHover {
        width: 150px;
        height: 50px;
        text-align: center;
        transition: all 200ms ease-in-out 100ms;
      }
      .buttonHover:hover {
        height: 60px;
      }
      input[type="text"] {
        width: 150px;
        transition: all 400ms ease-in-out 100ms;
      }
      input[type="text"]:focus {
        width: 250px;
      }
    </style>
  </head>
```

```html
<body>
  <div class="hoverDivContainer">
    <h1>hover on me to see the transition</h1>
  </div>
  <button class="buttonHover">Hover on me</button>
  <input type="text" id="name" />
</body>
```

CSS Transform

1. The **transform** property in CSS allows you to apply 2D or 3D transformations to an element. These transformations can move, rotate, scale, or skew an element, and they do not affect the document flow, making them ideal for creating dynamic and visually engaging layouts.

Syntax:

transform: function(value);

Function	Description	Example
`translate(tx, ty)`	Moves the element by the given X (tx) and Y (ty) values.	`transform: translate(50px, 100px);`
`scale(sx, sy)`	Scales the element by the given X (sx) and Y (sy) factors.	`transform: scale(1.5, 1.5);`
`rotate(angle)`	Rotates the element by the specified angle.	`transform: rotate(45deg);`
`skew(x-angle, y-angle)`	Skews the element along the X and Y axes by the specified angles.	`transform: skew(20deg, 10deg);`
`matrix(a, b, c, d, tx, ty)`	Applies a combination of `translate`, `scale`, `rotate`, and `skew` using a transformation matrix.	`transform: matrix(1, 0, 0, 1, 50, 100);`
`translate3d(tx, ty, tz)`	Moves the element in 3D space along X, Y, and Z axes.	`transform: translate3d(50px, 100px, 30px);`
`rotateX(angle)`	Rotates the element around the X-axis in 3D space.	`transform: rotateX(45deg);`
`rotateY(angle)`	Rotates the element around the Y-axis in 3D space.	`transform: rotateY(45deg);`

```html
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>CSS transform</title>
    <style>
      .box {
        width: 100px;
        height: 100px;
        background-color: lightblue;
        /* The element is moved 50 pixels to the right and 50 pixels down. */
        transform: translate(50px, 50px);
        transition: all 0.8s ease-in;
      }
      .box:hover {
        /* transform: translateX(150px); */
        /* The circle is resized to 1.5 times its original size. */
        /* transform: scale(1.5); */

        /* The element is rotated 45 degrees clockwise. */
        /* transform: rotate(45deg); */

        /*  The skew CSS function is used to distort an element along its X and/or Y axes,
        creating a slanted or "skewed" effect. This can make elements appear as
        if they are being viewed from an angle, rather than straight on.*/
        /* transform: skew(20deg, 15deg); */

        /* combine */
        /* The element is moved, rotated, and scaled in one combined transformation. */
        transform: translate(250px, 80px) rotate(45deg) scale(1.5);
      }
    </style>
  </head>
  <body>
    <div class="box"></div>
  </body>
</html>
```

Centre the box when position absolute

```css
.container {
  position: relative;
  max-width: 900px;
  margin: 130px auto;
  background: #f4f4f4;
  height: 600px;
}

.logo {
  max-width: 30%;
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```

CSS Animation

CSS **animations** allow elements to transition between various states smoothly over time. They are defined using keyframes and controlled by various animation properties.

Basic Structure

1. **Keyframes**: Define the states and transformations the element will go through.
2. **Animation properties**: Control the behavior of the animation (e.g., duration, timing function).

Key CSS Animation Properties

Property	Description
`animation-name`	Specifies the name of the `@keyframes` defined for the animation.
`animation-duration`	Defines how long the animation lasts.
`animation-timing-function`	Specifies the speed curve of the animation (e.g., `linear`, `ease`, `ease-in-out`).
`animation-delay`	Defines a delay before the animation starts.
`animation-iteration-count`	Specifies how many times the animation should run (e.g., `infinite`, `1`, `2`).
`animation-direction`	Specifies whether the animation should alternate direction (e.g., `normal`, `reverse`, `alternate`).

```html
<body>
    <div class="fade-in"><h1>hello animation</h1></div>
    <div class="bounce">Bounce div</div>
  </body>
```

```css
<style>
      @keyframes fadeIn {
        from {
          opacity: 0;
          transform: translate(0px, 0px);
        }
        to {
          opacity: 1;
          transform: translate(150px, 150px);
        }
      }

      @keyframes bounce {
        0%,
        100% {
          transform: translate(60px, 60px);
        }
        50% {
          transform: translate(260px, 260px);
        }
      }

      .bounce {
        animation: bounce 2s infinite;
        width: 100px;
        height: 100px;
        background: lightblue;
      }

      .fade-in {
        animation: fadeIn 2s ease-in infinite;
      }
    </style>
```