Developer Tools and Debugging

Browser Developer Tools

- You learned about chrome developer tools. Many **modern browsers** that you see today have **developer tools built into them**, these are called browser developer tools. They **all work in a similar way**. For example Mozilla Firefox, Microsoft Edge, Brave, etc.
- These are **powerful tools for web developers** for a variety of purposes.

Uses of Browser Developer Tools

- They help you understand a web page structure (basic HTML), and styling (CSS) of various elements by inspecting them.
- They help in modifying the HTML and CSS of the page to experiment with the looks of the webpage for trial purposes.
- They help in debugging code.

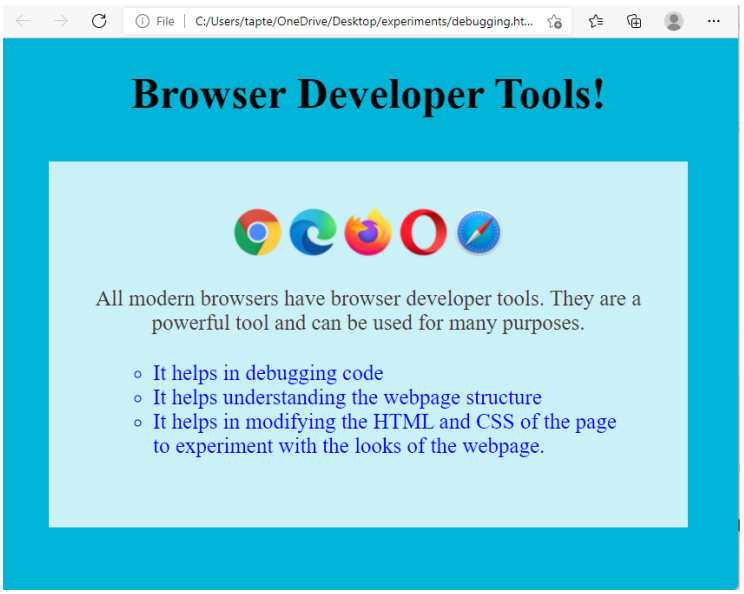
Note: *They also help you understand the javascript code and functionality associated with it. You'll learn more about this in the upcoming lectures.*

Debugging

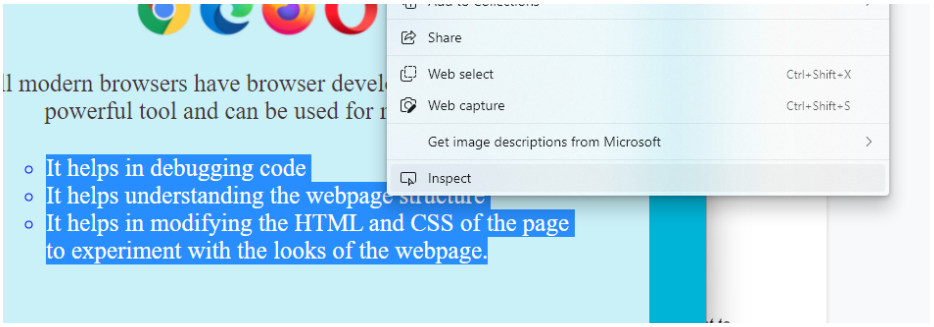
Sometimes our code doesn't behave the way we wanted to.
For example, we applied black text color to a heading but it appears to be blue in the browser.
So what went wrong over here?
How do we know the underlying reason for it?
Lastly, how to fix it?
So browser developer tools are very helpful in such cases where debugging is required.

How to debug CSS code?

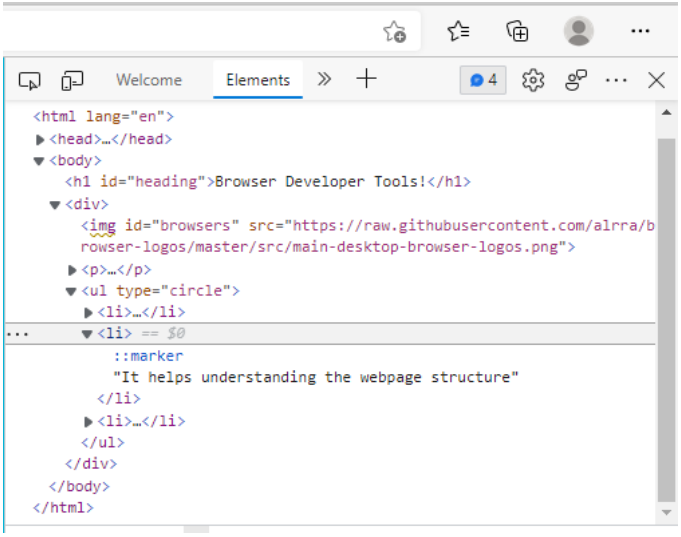
Let's say you have a simple web page as shown below. The text color of the list items is blue. But you coded it to be green. Now you wish to understand why it is appearing blue while you coded it to be green. Here we have used Microsoft Edge as our browser. Similarly, you can use developer tools in other browsers as well.



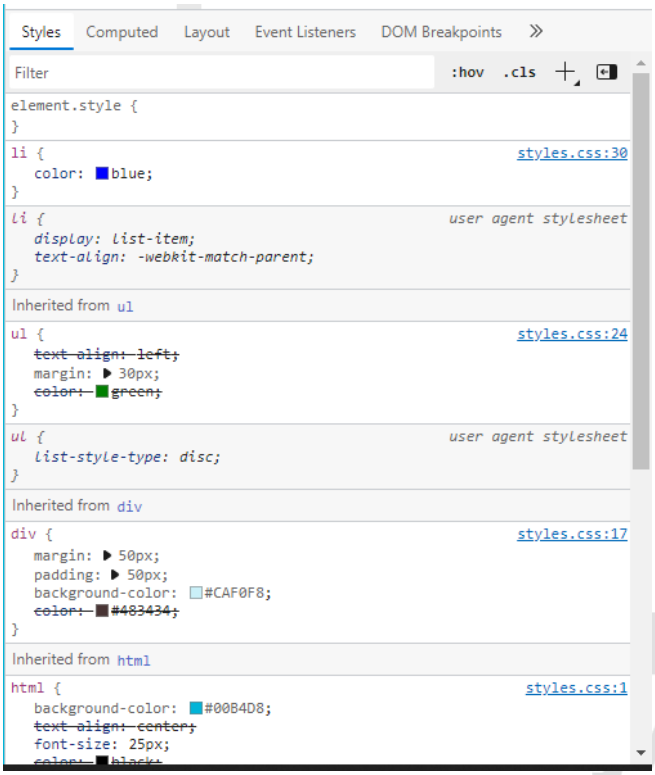
- **Step 1:** Select the HTML element that you want to fix, then right-click and select “inspect”



- **Step 2:** Under ‘Elements’ you’ll be able to see the HTML code highlighted with respect to the HTML element that you selected to inspect. Here one <li>.. is selected. You can click on it to expand and see its contents.



Below you can see ‘Styles’ where you’ll be able to see all the styles applied to the selected element.



- **Step 3:** Examine the styles applied **carefully.**
  Points to Note:
  - You can see the CSS rules applied to the selected element are in a specific order of most-to-least specific. As:
  - The <li> is present inside <ul>. The <ul> is present inside a . The is present inside <html>.
  - The most specific is the CSS rules applied to <li>.
  - The least specific is the CSS rules applied to <html>.

- **Step 4:** Come to a conclusion after examining the styles
  - The text color green is applied to <ul>, but its content still has blue color because <li> is more specific than <ul>. The color applied to <li> is blue, hence the final color that appears is blue. Now we know where the problem is, and we can fix it by changing the text color of <li> to green.

In conclusion,

- ***Browser developer tools arrange styles applied to an element in order of most-to-least specific. The above example was very simple. But in huge and complex websites it gets very easy to figure out styling-related problems by inspecting.***

- ***In Step 3, you can see some styles have strikethroughs. For example in <ul>, ~~color: green;~~***
  This means you coded <ul> contents to have the color green, but the browser didn't apply this color, since some other more specific styling (that is applied to <li>) overrode this value. Hence developer tools help us figure out such scenarios helping in debugging.

Note:

You don't have to master using Browser Developer tools in one go. As you keep practising you'll learn with time.

In general, if you ever face any issues with Chrome Developer Tools or if you want to learn more, you can refer to its documentation whenever required.

- *Documentation*
- *MDN Article on Chrome Dev Tools*

Introduction to CSS

Adding CSS to your websites will make them presentable and somewhat like the sites you see on the internet.

- **CSS is a shorthand for Cascading Style Sheets.**
- **CSS describes how the HTML elements are to be displayed.**
- **CSS is used to control the layout of multiple web pages at once.**
- **Includes the addition of visuals such as colour, fonts, layouts, etc.**

The syntax for CSS is:

```
selector {
    property: values;
}
```

- Each property ends with a semicolon.
- Each property includes a name for the css property and a value separated by a colon.

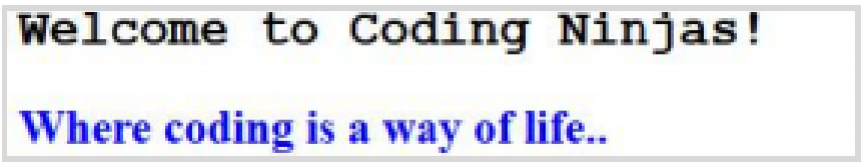
Below is an example, to see how CSS modifies the HTML code:

```
<h1>Welcome to Coding Ninjas!</h1>

<h2>Where coding is a way of life..</h2>
```

Now, adding CSS to the above HTML code:

```
h1{
   font-family: monospace;
}
h2{
    color: blue;
}
```

The code snippet would now look like this:



CSS COMMENTS

Comments are code that is ignored by the browser. They help make code understandable, hence helping in changes in code that may be required at a later stage easily, by defining the various sections of HTML element styles.

A CSS comment initiates with `/*` and ends with `*/`. Comments can also span multiple lines.

ADDING CSS TO HTML PAGE

The browser formats the HTML document based upon the information in the stylesheet. The browser will access the stylesheets in the HTML document itself. There are 3 ways to add CSS styles to your document:

- Inline Styles
- External Styles
- Internal Styles

Cascading order decides which styles will be applied to elements when multiple styles are used.

Cascading order priority is given as:

Inline > (internal≅ external) > browser default.

The browser treats both internal and external CSS equally, but the order in which they are defined, determines which property gets priority.

- If the link to internal CSS is defined *before the external CSS*, then properties of external CSS will get preference over internal CSS, i.e. **external CSS > internal CSS.**
- If the link to internal CSS is *defined after the external CSS*, then properties of internal CSS will get the preference, i.e **internal CSS > external CSS.**

Inline Styles

- The **style attribute** is used to apply an inline stylesheet directly to our HTML code.
- The inline stylesheet syntax will have the properties specified inside the style attribute.
- Multiple properties can be specified at a time.
- To **apply a unique style to a single element**, use inline styling.

You can use inline styles like this:

```
<p  style="color:blue;font-size:40px">Inline CSS</p>
```

Internal Styles

An **internal** or **in-page** stylesheet:

- It contains the CSS style code for the web page.
- It provides the styles for that particular HTML document only.
- It cannot be reused.
- It can be specified with the help of the **<style>** tag inside the **<head>** tag.

You can use internal styles like this:

```
<style>
        h1 {
        color: blue;
        }
        h2 {
        color: red;
        }
        p {
        color: green;
        }
</style>
```

External styles

- You should use an external style sheet if you want to apply styling to a website using just one file.
- Although the **syntax is similar to internal stylesheets**, it is implemented

using a **separate CSS file.**

- The '**.css' extension** is used to save it. Eg. 'styles.css'.

To use an external stylesheet, a reference is provided to file inside the **<link>** element:

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

Attribute	Value
rel	Defines the linked document relationship
href	Specifies where the linked document is located
type	Defines the type of media of the linked document

NOTE: *The link is placed within the head. CSS syntax code is contained in the 'styles.css' file only. The name of the file could be anything followed by .css extension, and the same should be specified while linking it to the html file using <link> tag.*

SELECTORS

Selectors are used to point to the HTML element that needs to be styled. Selectors are used in both internal and external stylesheets.

Styles are applied using three different types of selectors:

- **Element selector**
- **Class Selector**
- **Id selector**

When multiple styles are applied to an element, specificity determines which style will be used.

The latest rule is applied, if the specificity is the same.
Specificity order:
inline > id selector > class selector > tag selector > browser default

NOTE: *If the same property is defined inside the same type of selector, then the property which is defined at the last will be used by the browser.*

Element Selector

The element selector will help us to select all elements with the same mentioned element name. This will select all the elements in the HTML document with the given name, but most of the time this is not our requirement. So, to apply styles to only some specific elements we need to have some restrictions. We will take a look at this later in this section only.

Syntax: `element { css declarations; }`

Eg., applying style to h2 tag like this:

```
<h1>Blue Color</h1>
```

and applying CSS like this:

```
h1{
    color: blue;
}
```

will show on the browser like this:

Class Selector

Multiple elements with a specific class attribute are selected using the class selector. To select elements with a specific class, type a period (.) followed by the class name.

Syntax: `.class-name {css declarations; }`

To use the class selector, the **class attribute** is used in the element's opening tag. The value of the class attribute contains the name of the class. There can be **multiple classes** added to the tag by giving space in between.

Eg., defining the classes in the HTML file like this:

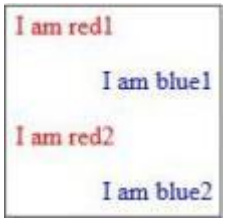
```
<p class="red">I am red1</p>
```

```
<p class="blue right">I am blue1</p>
<p class="red">I am red2</p>
<p class="blue right">I am blue2</p>
```

The CSS code file

```
.red{
    color: red ;
}
.blue{
    color: blue;
}
.right{
    text-align: right;
}
```

will show on the browser like this:



Id selector

The id selector will help us to select only one element with that specific id. We need to write a hash(#) character and then id name to select an element with a specific id.

Syntax: `#class-name{css declarations;}`

To make use of the id selector, the **id attribute** is defined in the element's opening tag. The value of the id attribute will have the name of the id. The id is **unique** on an HTML page.
There can only be **one id** in the tag. If another element is having the same id, the styles would not be applied by the browser.

Eg., defining the ids in the HTML file like this:

```
<p id="one">This is id one!</p>

<p id="two">This is id two!</p>

<p id="three">This is id three!</p>

<p id="four">This is id four!</p>
```

and applying css like this:

```
#one {
    color: blue;
}
#two {
    background-color: teal;
}
#three {
    color: green;
}
#four {
    background-color: lightgrey;
}
```

will show on the browser like this:

This is id one!

This is id two!

This is id three!

This is id four!

Grouping Selectors

We usually use the same CSS for multiple elements, and we can't have too many classes,as too many classes would become difficult to manage.
So, CSS helps us with a grouping feature where you can define the CSS rules to multiple elements with the use of a combination of either class, tag, or id.
We **need to use a comma separator for the different selectors** for grouping

Here are a few examples of how grouping can be used:

- `p, .class-name { CSS properties }` - apply styles to 'para' and element with class as 'class-name'
- `#id1, #id2, span { CSS properties }` - apply styles to 'span' and elements with ids as 'id1'and 'id2'

- `.class-name, #id1, div { CSS properties }` - apply styles to 'div' and elements with class 'class-name' and id as 'id1'.

Nesting Selectors

Whenever we require to target elements inside a particular section of our HTML page. Instead of using the classes there, we can use nesting that works like a hierarchy and is easier to understand.

To use nesting, you need to **add space between the selectors**. Hence the sequence formed represents a **hierarchy starting from the top**.

These are just a few examples:

- `.class-name span { CSS declarations }` - this will apply styles to only those 'span', which are present inside the element with class 'class-name
- `#id1 .class-name span { CSS declarations }` - this will apply styles to only
  those 'span', which are present inside the element with class 'class-name' and 'class- name' is inside the element with id 'id1'

Chaining Selectors

There are times when we want to have the same class for multiple elements and we want to apply styles to them. In this scenario, we can use chaining selectors.

To use chaining we take the help of the combination of selectors without putting any space in between them.
Eg., we have a class 'header-style' applied to every heading. We can apply different styles to them like this:

```
<html>
    <head>
       <style>
          .header-style{
            background-color: aqua;
            display: inline-block;
           }
       </style>
    </head>
    <body>
```

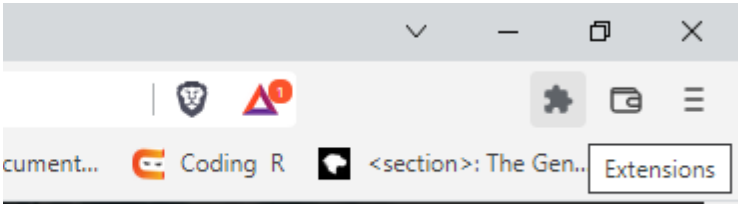
```
        <h1 class="header-style">Hi</h1>
        <h1 class="header-style">Hello</h1>
    </body>
</html>
```

Now, as the class **header-style** is given to both **<h1> tags**, we can apply styles to both of them together as done in the above example.

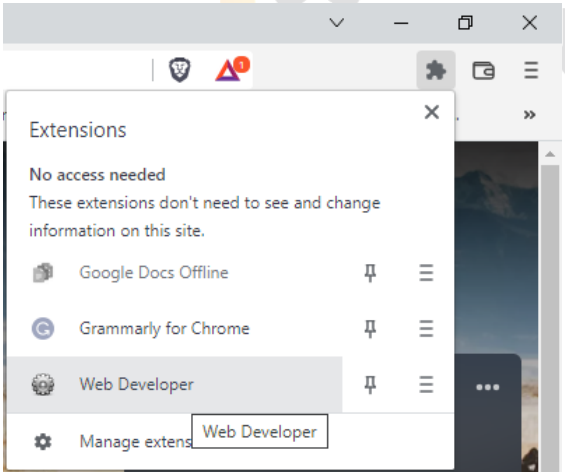
Chrome Extension for Disabling CSS

If you want you can add this chrome extension to disable the CSS of any website. Then check and learn how important CSS is for styling.

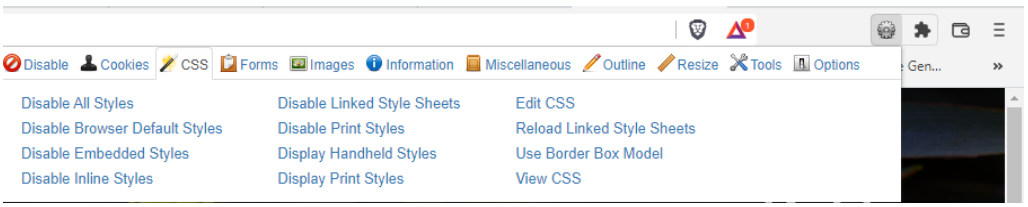
- Go to this link and add this extension to your browser Link.
- Open the website whose CSS you want to disable.
- Click on the icon of the extension on the top left corner of your browser.



- Select the Web Developer extension that you just added as shown.



- Select CSS → Disable All Styles



Styling with CSS

COLOR

- The color property is used to set the decoration and **foreground color of an element's text context**.

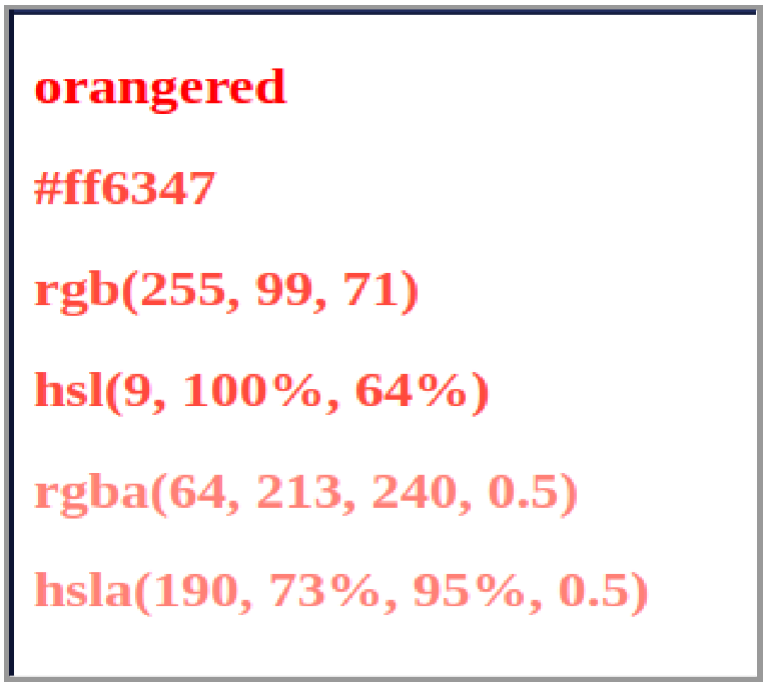
- The color property can be mentioned in 6 different ways. Each of the given ways provides some difference from the other.

Although the color property can also be used for the backgrounds and borders, we will discuss them later.

E.g. the following code:

```
<h2 style="color:orangered;">orangered</h2>
<h2 style="color:#ff6347;">#ff6347</h2>
<h2 style="color:rgb(255, 99, 71);">rgb(255, 99, 71)</h2>
<h2 style="color:hsl(9, 100%, 64%);">hsl(9, 100%, 64%)</h2>
<h2 style="color:rgba(255, 99, 71, 0.7);">rgba(255, 99, 71, 0.7)</h2>
<h2 style="color:hsla(9, 100%, 64%, 0.7);">hsla(9, 100%, 64%, 0.7)</h2>
```

shows the following output, which makes the difference quite easy to understand:



By name

- All the modern browsers support 140 different colors named in CSS. Unlike HTML, unknown keywords are totally ignored by CSS.
- The keywords of color are all simple, solid colors without transparency. *E.g., orangered, green, blue, lightgrey etc.*

Using RGB

- RGB is an abbreviation for **R**ed, **G**reen, and **B**lue (RGB). It's a color model in which a color is created by **combining red, green, and blue**.
- The intensity of each color has values ranging from 0 to 255. This gives us a very large number of colors in the dataset.
    - E.g. The RGB value for **black** is: `rgb(0, 0, 0)` and for **white** is: `rgb(255, 255, 255)`.

By HEX Code

- The colors can also be given by **6 digits hexadecimal code**.
- The hex codes are made with the help of the 3 *colors(Red, Green, and Blue).* The first 2 digits represent red, the next 2 are for green and the last 2 are blue.
    - So, the syntax for hex code can be given as `#RRGGBB`.
- Each hexadecimal code value between **00 - FF** is similar to **0 - 255**.
    - E.g. `#000000` represents **black** and `#FFFFFF` is **white**.

Using HSL

The HSL (**H**ue, **S**aturation, and **L**ightness) components can also be used to specify the color.

- **Hue** is a number that ranges from **0** to **360** on the color wheel. **0** represents **red**, **120** represents **green**, and **240** represents **blue**.
- The amount of saturation in a color is represented by **saturation**. It's a percentage value; 0% represents a shade of **grey**, and 100% represents the **entire color spectrum.**
- The amount of **light** in a color is represented by its **lightness**. It's also a percentage, with 0% being **black**, 50% being **neither light nor dark**, and 100% being **white**.

Using rgba

- RGBA (**R**ed, **G**reen, **B**lue, **A**lpha) is a color space that combines **RGB** and **alpha transparency**. The opacity of the RGB-defined color is determined by this alpha value.
- The alpha parameter is a number that ranges from 0.0 (transparent) to 1.0 (opacity) (opaque).
    - For example, rgba(255, 0, 0, 0.6) is a red color, and with 0.6 opacity, it will appear as:

Using HSLA

- Similar to RGBA, **HSLA** (**H**ue, **S**aturation, **L**ightness, **A**lpha) is a **variant** of HSL that includes **alpha transparency**.
- The property and alpha value are the same as in RGBA.
    - For example, hsla(0, 100%, 50%, 0.6) is also a red color with 0.6 opacity and the same color:

Note: Every color can be represented using RGB values. A certain color will always be a mix of RGB, red, green, and blue of various proportions.

To know more about CSS color value you can refer to this link:

https://developer.mozilla.org/en-US/docs/Web/CSS/color_value

You can also refer to this website tool for finding beautiful colors and color palettes: https://colorhunt.co/

CSS UNITS

- In CSS, a unit can be represented with **a value followed by the unit symbol.**
    - For eg: 10px, 5mm, 2%, etc.

- Some units are shared between properties, while some are restricted to certain properties.

- The unit can be omitted if the value is 0. Whitespace should be omitted between the number and the unit.

- Negative lengths are allowed for some CSS properties. For instance, consider the margin property. Absolute and relative length units are the two types of length units.

Absolute Units

- The absolute units are a fixed size/length of the element. Because screen sizes vary so much, absolute length units are not recommended to be used on screens.
- Absolute units are made up of the following:

Unit	Description
cm	centimetre
mm	millimetre
in	inches (1 in = 96px = 2.54cm)
px	pixels (1px = 1/96th of 1in)
pt	points (1pt = 1/72 of 1in)
pc	picas (1pc = 12pt)

Relative Units

- Relative length units describe how long something is in comparison to another length property.

- The following are some examples of relative units:

Unit	Description
em	relative to the font size of the parent element (3em means 3 times the size of the current font).
rem	relative to the font size of the root element.
vw	relative to 1% of the width of the viewport (browser width)
vh	relative to 1% of the height of the viewport.
%	relative to the parent element.

BORDER

- The element's border is controlled by the **border** property.
- The **style**, **width**, and **color** of an element's border can all be specified with CSS borders.
- Border-width, border-style, and border-color are written together as the **border** which is a **shorthand** for all the properties mentioned above.

  E.g. applying border property to a div like this:

```
border: 4px solid red;
```

will show like:

We will look into the different properties of the border.

Border Width:

- The width of the four borders is determined by the **border-width** property.
- The width can be given by using one of the three predefined values: *thin, medium, or thick* or as an absolute or relative size

  Syntax:
  - The CSS border-width property can be expressed with one, two, three, or four values provided.

  Syntax - One Value

border-width : all; (i.e top, right, bottom, left)

- When one single value is provided, the border-width value will apply to all four sides of the box.

Syntax - Two Values

border-width : top_bottom left_right;

- When two values are provided, the first value will apply to the top and bottom of the box. The second value will apply to the left and right sides of the box.

Syntax - Three Values

border-width : top right_left bottom;

- When three values are provided, the first value will apply to the top of the box. The second value will apply to the right and left sides of the box and the third value will apply to the bottom of the box.

Syntax - Four Values

border-width : top right bottom left;

- When four values are provided, the first value will apply to the top of the box, the Second value will apply to the right side of the box, the third value will apply to the bottom side of the box and the fourth value will apply to the left side of the box

Note: The same syntax is valid for the below properties.

Border Style:

- This property determines the type of border that will be displayed. Dotted, solid, dashed, ridge, double, groove, none, inset, outset, hidden are the border-style values.
    - Eg., `border-style: dotted dashed solid double;` will have dotted top border, dashed right border, solid bottom border, and double lined left border.

***NOTE**: It is necessary to add border-style property else no other border properties will work.*

Border Color:

- The color of the four borders is specified by the **border-color** property.
- The property's value is identical to that of the **color property**. However, different colors can now be assigned to different border sides. If border-color isn't specified, the element's color is used.
- E.g. `border-color: red blue`; The top and bottom borders will be red, while the left and right borders will be blue.

Border Individual Sides:

- We can provide width, style, and color to each border separately using the border property, but we must still give some value to each side of the border.
- CSS border also has the ability to assign a border value to each of the border sides separately.
- The sides' border properties are
    - **border-top**
    - **border-right**
    - **border-bottom**
    - **border-left**

- This is further broken down to give each border side its own style, width, and color. **border-top-style, border-right-width, border-left-color**, and so on are some of them.

Border Radius:

- This property is used to give an element rounded borders. This property can have an absolute (e.g. in px) or relative (e.g. in percent) value.
    - Eg., If we add border-radius: 10px; the first border example will show as follows:

- The border-radius can also be in the **elliptical form**. That is why you need to specify horizontal and vertical radius differently.
- This is done with the help of a slash ("/") between horizontal and vertical radius.

Here is an example:

```
div{
    border-radius: 40px/20px;
    background: teal;
    width: 20%;
    height: 20%;
    padding: 40px;
}
```

Here is another interesting example:

TEXT AND FONT STYLING

To change the look and style of text in the HTML document, various properties are specified. These styles will apply only to the text content of any element.

Let us have a look at some of the most used text and font styling properties.

- Font properties such as font-family, boldness, size, and style define how a font looks.
- **Text properties** are used to define the presentation and layout of the text on the HTML page.

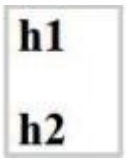
font-size:

- This **sets the text size**.
- The font-size value can be either absolute or relative, i.e., values in px, percent, em, and so on.

E.g.

```
h1{
    font-size: 30px;
}
h2{
    font-size: 1.875em;
}
```

will show



font-family:

- The font family of a text helps us to set the **font-family** property.
- As a "**fallback**" system, the font-family property holds several font names. If the first font isn't supported, the browser moves on to the next font, and so on.

Begin with your desired font and end with a generic family to let your browser

select a similar font from the generic family, unless there are other fonts.

E.g.

```
h1{
    font-family: sans-serif, monospace, serif;
}
```

will show

***NOTE**: If a font family's name contains more than one word, it must be enclosed in quotation marks, such as "Times New Roman."*

font-weight:

- The **font-weight** property is used to define the **weight/thickness** of the given font.
- The weight lies between light to bold. Bold, bolder, inherit, initial, lighter, normal, and unset are all possible values. Alternatively, we can define the font weight using numeric values ranging from 100 to 900.
  E.g.

```
h1{
    font-weight: lighter;
}
```

will show

font-style:

- The **font-style** property is used to define the **style for a text**.
- The general values for this property are: **normal, oblique, italic, initial, inherit**.

color:

- We have already discussed applying **color** to text using the color property.
- The color can be defined either by name, hex code, rgb, rgba, hsl, or hsla.

text-align:

- The **text-align** property is used to **define the horizontal alignment of a text**.
- A text can be aligned left, right, centered, or justified.

Value	Description
left	It is used to align the text to the left.
right	It is used to align the text to the right.
center	It centers the inline text.
justify	It stretches the element's content in order to display the equal width of every line. (like in newspaper and magazine)

E.g.

```
p#center{
    text-align: center;
}
p#left {
    text-align: left;
}
p#right{
    text-align: right;
}
```

Will show:

text-indent:

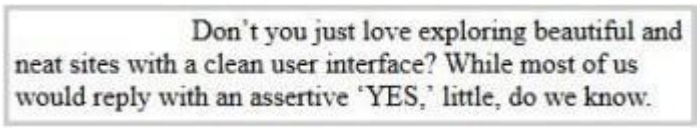
- The **text-indent** property controls the **indentation** of a **text block's first line**.
- **Negative numbers** are **permitted**. If the value is negative, the first line will be

indented to the left.

E.g.

```
<p style="text-indent: 100px;">Don;t you just exploring
beautiful and neat sites with a clean user interface? while most of us
would reply with an assertive "YES", little do we know</p>
```

Will show:


Don't you just love exploring beautiful and neat sites with a clean user interface? While most of us would reply with an assertive 'YES,' little, do we know.

text-transform:

- The **text-transform** property defines the **case of the letters in a text**. It can be used to turn text to:
  - **uppercase** - this will turn every character to uppercase.
  - **lowercase** - this will turn every character to lowercase.
  - **capitalize** - this will turn each word's first letter to uppercase, while the rest is converted to lowercase.
  - **none** - It's the default setting. The text is rendered exactly as it is.

Text-decoration:

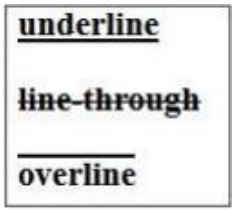
- The **text-decoration** property is used to **add and remove text decorations**.
- *To remove underlines from links*, the value **text-decoration: none;** is frequently used.
- The text-decoration is used to decorate the text.
- It has 4 values**:**
  - **underline -** puts a line under the text
  - **overline -** puts a line above the text
  - **line-through -** puts a line through the text
  - **none -** removes any of the above decoration

  E.g.

```
h3#underline {
    text-decoration: underline;
}
h3#line-through{
    text-decoration: line-through;
}
```

```
h3#overline {
    text-decoration: overline;
}
```

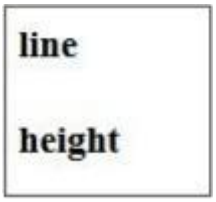
will make the text show like



line-height:

- The **line-height** property controls the **height of an element's lines**. It has **no effect** on the **font's size.**
- The font-size value can be an absolute or relative size, i.e., values can be applied in px, %, em, etc. The value is multiplied by the font-size of the element if no unit is specified.

E.g. wil show the lines like:

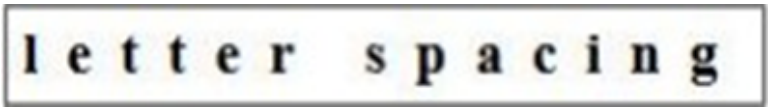


letter-spacing:

- If you want to **modify the space between the letters,** you can use the **letter-spacing** property.
- The space between the letters can be **increased** or **decreased** as per convenience.

E.g. `h3 { line-height:1.5; }`

Will show the text like



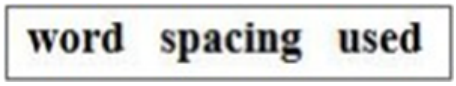
word-spacing:

- The **word-spacing** property can be used to change the amount of space between words.
- You can use this to **increase** or **decrease** the space between the words.
- The value of word-spacing can be absolute or relative.

  E.g.

```
h3 {word-spacing: 10px; }
```

Will show the text like:



text-shadow:

- The **text-shadow** property **adds a shadow to text.**
- The **position of the horizontal shadow**, the **position of the vertical shadow**, and the **color of the shadow** are all contained within this value.

E.g.

```
h2 { text-shadow: 2px 1px red; }
```

Will show the text like:

BACKGROUND

You can also edit and modify the background of an element.

- The background includes an **element's dimensions**, **the padding** and **border** but **excludes the margin.**
- Backgrounds in CSS can be **colors** or they can be **images**.

CSS Background properties:

- **background-size**

- **background-attachment**
- **background-image**
- **background-repeat**
- **background-position**

E.g. the below CSS code when applied to a web page.

```
body {
    background-image:
url("https://blog.codingninjas.in/wpcontent/uploads/2017/01/cropped-Final_logo_sw
itchtoc ode-01.png"),lineargradient(#a3f7ff, #fff58e);
    background-repeat: repeat-x;
    background-position:center;
    background-attachment: fixedd;
}
```

Will show the web page like this:

***NOTE**: You can see about other background properties from:*
https://developer.mozilla.org/en-US/docs/Web/CSS/background

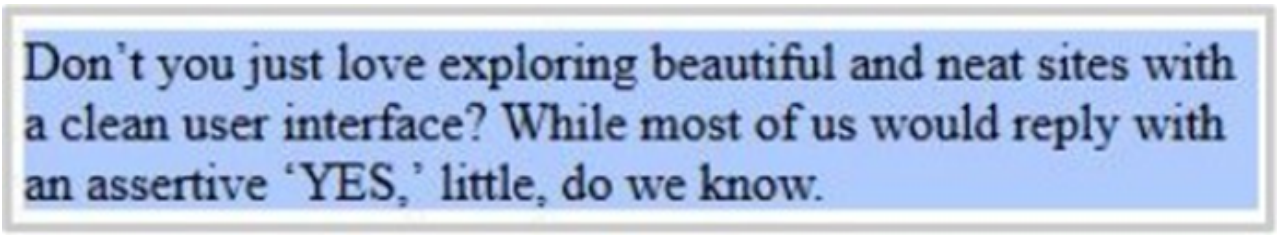
background-color:

- The **background-color** property specifies the **color of an element's background**.
- Its value is the **same as that of the color property.**

E.g.

```
<p style="background-color: #afcbff;"> Don't you just love exploring beautiful and neat sites with a clean user interface? While most of us would reply with an assertive 'YES,' little, do we know. </p>
```

Wil show like this:



background-image:

- To **specify an image to use as the background of an element** the **background-image** property is used.
- This property can be used to give an element one or more background images.
- By default, a background image is placed at the top-left corner of an element, and it is repeated so that it covers the entire element in both vertical and horizontal directions.

  The values it can take are

  - **url('URL')** - This specifies the image's URL. You can specify multiple images by using a comma to separate the URLs.
  - **none** - This is the normal setting. There will be no background image.
  - **linear-gradient()** - As the background image, this creates a linear gradient. A minimum of two colors will have to be mentioned (default direction is top to bottom).
  - **radial-gradient()** - As the background image, this creates a radial gradient. A minimum of two colors must be mentioned (the default is from the centre to the edges).
  - **repeating-linear-gradient()** - repeats a linear gradient
  - **repeating-radial-gradient()** - repeats a radial gradient

Background-repeat

- You can use the **background-repeat** property to control **how** and **if** a background image is replicated.
- By default, a background image repeats both **vertically** and **horizontally**, but
  background-repeat allows you to control how the image repeats.

The values this property can take are

- **repeat -** This is the standard-setting. Both vertically and horizontally, the background image is repeated. If the last image does not fit, it will be cropped.
- **repeat-x -** Only the horizontal portion of the image is repeated.
- **repeat-y -** Only the vertical portion of the image is repeated.
- **no-repeat -** the image will only be shown once
- **space -** There is no clipping on the background image. With the first and last images pinned to the sides of the element, the remaining space is evenly distributed between the images.
- **round -** this makes the image to be repeated and shrink or stretch to fill the space.

background-position

- A background image's initial position is specified using the **background-position** property.
- The **background-position** property can be used to **change the position of a background image**, which is by **default in the top-left corner** of an element.
- The values this property can take are**(X represents horizontal position and Y represents vertical position):**
  - **X Y -** they both can each take value from one of the following - ***left, right, top, bottom, center.*** If only one value is specified, the default value is "centre."
  - **Xpos Ypos -** specifies the horizontal and vertical position relative to the viewport. Any of the CSS units can be used as units. If only one value is specified, the other value will be set to 50%.

background-Size:

- The **background-size** property is used to **define the image size for the background.**
- The values it can take are
    - ***auto* -** This is the default setting. The image is shown in its original dimensions
    - ***length* -** sets the background image's width and height. The width is determined by the first value, while the height is determined by the second value.
    - ***percentage* -** sets the background image's width and height in percent. The width is determined by the first value, while the height is determined by the second. The second value is set to "auto" if only one is provided.
    - ***cover* -** resizes the background image to fill the container's horizontal width
    - ***contain* -** ensures that the background image is fully visible by resizing it

background-attachment

- This property determines whether a background image is fixed or scrolls with the rest of the page.
- The values it can take are:
    - **scroll -** this is the default value.The background image will follow the page as it scrolls.
    - **fixed -** The background image will not follow the page as it scrolls.
    - **local -** The background image will scroll along with the content of the element.

MARGIN

- The CSS margin properties are used to create **space between the borders and the other surrounding elements.**
- You can also provide **negative margins** as well.
- There are two other values that are used for the margin:
    - **auto** - the margin is applied by the browser itself only to horizontal margins
    - **none** - to remove any margins from the element or makes the value of margin equal to zero

Just like the border property, you can provide margins separately to the sides.

The margin property for the sides are:

- margin-bottom
- margin-left
- margin-top
- margin-right

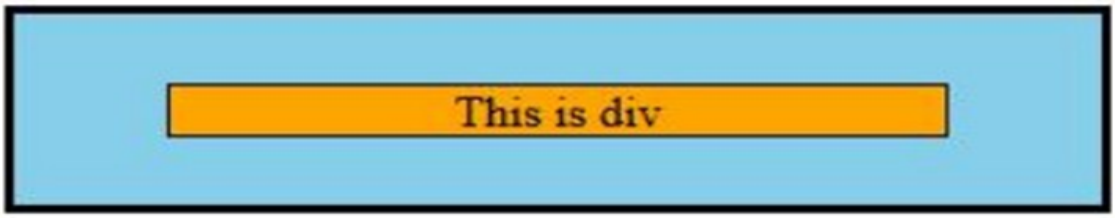
E.g: for the below HTML code:

```
<div class="blue-box"><div class="orange-box">This is div</div></div>
```

and applying the following CSS to the code:

```
.blue-box {
    border: 3px solid black;
    background-color: skyblue;
}

.orange-box {
    border: 1px solid black;
    margin: 25px 50px;
    text-align: center;
    background-color: orange;
}
```

we will see something like this on the browser:



NOTE: *When two elements are stacked vertically, their top and bottom margins are collapsed into a single margin equal to the larger of the two margins. Negative values can be found in margin.*

Negative margin

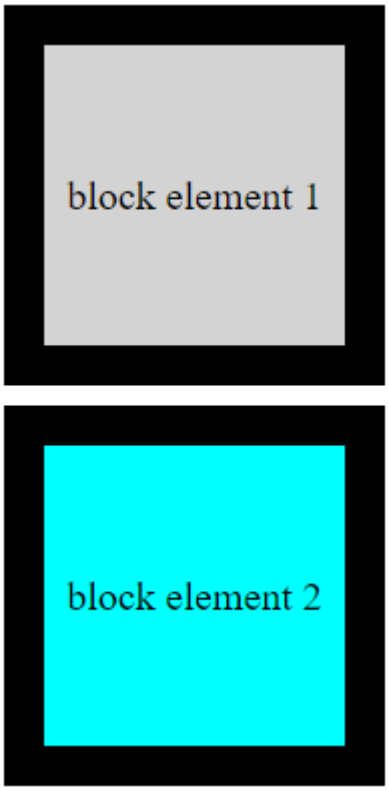
- It is possible to give margins a negative value. This allows you to draw the element closer to its top or left neighbour, or draw its right and bottom.

  In other words, A negative margin allows us to move elements in the opposite direction that usually occurs when using positive margins.

For e.g

```
#block-element-1{
    background-color: lightgray;
    font-size: 20px;
    text-align: center;
    line-height: 150px;
    width: 150px;
    height: 150px;
    border: 20px solid black;
    margin: 10px 10px 100px 10px;

}

#block-element-2{
    background-color: cyan;
    font-size: 20px;
    text-align: center;
    line-height: 150px;
    width: 150px;
    height: 150px;
    border:20px solid black;
    margin: 10px 10px 10px 10px;
}
```

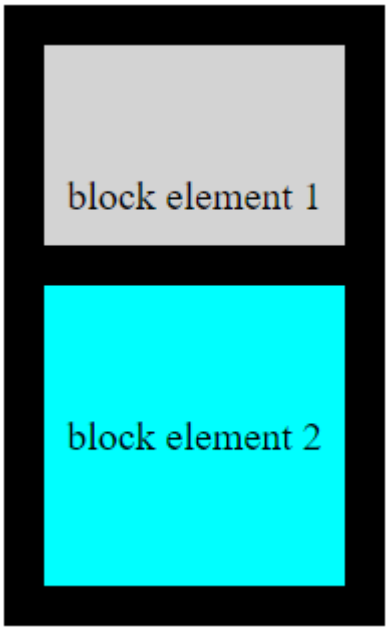
we will see something like this on the browser:



Now set negative margin-bottom to -80px

```css
#block-element-1{
    background-color: lightgray;
    font-size: 20px;
    text-align: center;
    line-height: 150px;
    width: 150px;
    height: 150px;
    border: 20px solid black;
    margin: 10px 10px -80px 10px;

}

#block-element-2{
    background-color: cyan;
    font-size: 20px;
    text-align: center;
    line-height: 150px;
    width: 150px;
    height: 150px;
    border:20px solid black;
    margin: 0px 10px 10px 10px;
}
```

Now we will see something like this on the browser:



This shows that negative margin elements pushed up the block element 2 from the bottom up into the block element 1 on the top.

PADDING

- The CSS padding properties are used to create space around the content and borders of an element.
- You can provide margins to the sides separately, just like the border property. The sides' margin properties are:
    - padding-top
    - padding-right
    - padding-bottom
    - Padding-left

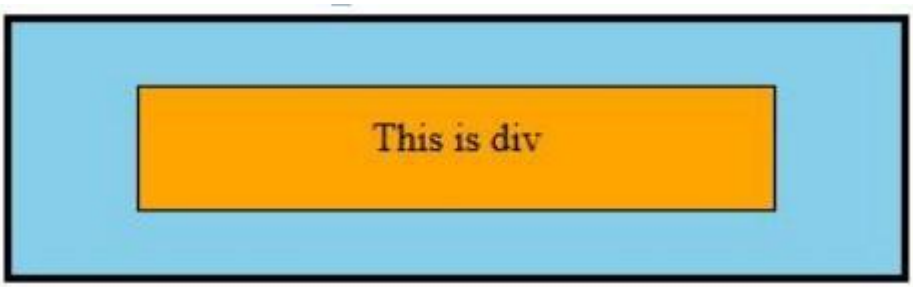
For the below HTML code:

```
<div class="blue-box">
<div class="orange-box">This is div</div>
</div>
```

and applying the following css code:

```
.blue-box {
    border: 3px solid black;
    background-color: skyblue;
}
.orange-box {
    border: 1px solid black;
    margin: 25px 50px;
    padding: 10px 0 20px 0;
    text-align: center;
    background-color: orange;
}
```

we will see something like this on the browser:



DISPLAY

- This one is a **crucial** CSS property for layout control. It determines **whether or not an element is displayed.**
- Many of the elements have default display property values as inline or block (inline and block elements in HTML).
- The display property has the following values:
  - inline
  - block
  - inline-block
  - none

Inline Display

- When display: inline; is used the following happens to the element:
- The element doesn't start in a new line
- only takes up as much width as necessary, so cannot set height and width.
- The vertical margins do not work.

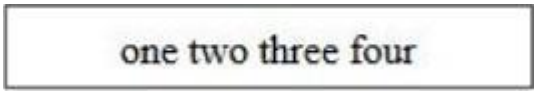
E.g originally a list looks like this:

- one
- two
- three
- four

making the list inline:

```
li { display: inline; }
```

and the list will now be shown in a single line like this(space before the list is because ul element has a default padding value):



Block Display

- When ***display: block;*** is used the following happens to the element:
  - Element starts in a new line
  - Occupies the entire width of the parent element

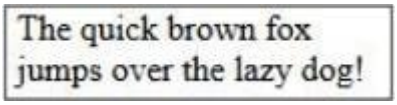
E.g. originally the 2 spans look like this:

```
<span>The quick brown fox</span> <span>jumps over the lazy dog!</span>
```

Adding the block property to them like this:

```
li {
    display: block;
}
```

and it will shown like this now:



inline-block Display

- ***display: inline-block;*** is a combination of the properties of the inline and block elements.
- These are the advantages of inline-block:
  - The height and width of the element can be set now.
  - vertical margins are allowed.

- The element can sit next to each other.

The elements next to each other have a space between them.
E.g. there are two empty div element:

```
<div class="div"></div>
<div class="div"></div>
```

and the following properties have been applied to them:

```
.div{
    display: inline-block;
    border: 1px dashed black;
    width: 100px;
    height: 100px;
    background-color: lightgrey;
}
```

two divs will be displayed next to each other like this:

You can notice a *space between* these 2 divs. It is because inline elements have ***word-spacing in them*** and it also defaults in inline-block elements.

No Display

- The `display: none;` property hides the elements in a browser.
- It actually removes the element from the HTML page and nothing is shown in its place.
- This is similar to another property - `visibility: hidden;`.
- The difference is that the element is not removed from the page and still occupies the space.

POSITION

- The **position** property is for specifying the **positioning of elements on the page.**
- This fixes the position of the element **relative to the web page or parent element.**

- The position property provides 5 ways to position the element
  - **static -** default
  - **relative**
  - **absolute**
  - **fixed**
  - **sticky**

This property is not enough to manipulate the position of the element. The position property just specifies the behavior.

Therefore, to move the elements we have these 4 properties:

- **left** - The element is shifted to the left side of the element as a result of this.
- **right** - The element is shifted to the right side of the element as a result of this.
- **top** - This causes the element to shift in relation to the element's top side.
- **bottom** - This causes the element to shift in relation to the element's bottom side.

The above properties, i.e. **left, right, top** and **bottom** have numerical values in both relative and absolute units.
The value can be both ***positive and negative***. Depending on the position value, they also work differently.

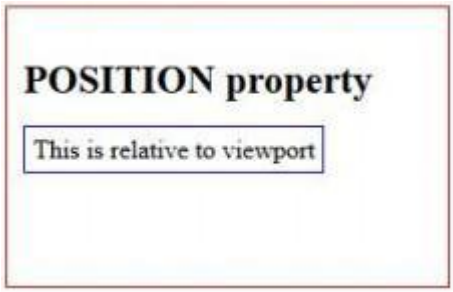
E.g. for the layout like this:

```
<div class="outer">
    <h2>POSITION property</h2>
    <span class="inner">This is relative to viewport</span>
</div>
```

and making div's position relative with CSS code like:

```css
.outer {
    position: relative;
    width: 250px;
    height: 150px;
    border: 1px solid red;
    padding: 10px;
}

.inner {
    border: 1px solid blue;
    padding: 5px;
    top: 10px;
    left: 40px;
}
```

will show on the web page like this:



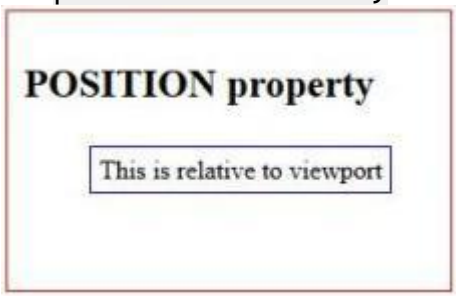
static

- The element is positioned using the **static** value in accordance with the page's normal flow, not in any special way.
- This is the **default** property. Properties like a top, right, bottom, left do not work when this is used.
- You can alternatively use `position: static;` to apply this property and the layout will look the same.

relative

- The element is positioned **relative** to its **first positioned** (i.e. not static) **ancestor element** by the **relative value**.
- It's **similar to a static value**, but now the element's properties such as top, right, bottom, and left will work.
- ***The element's original position is still occupied.***

Use `position: relative;` to apply this property and the layout will look like this:



absolute

- The **absolute** value positions the element **relative to its closest positioned ancestor.** If there is no positioned ancestor, then the position would be relative to the browser's window.
- ***The element doesn't occupy its original space.***

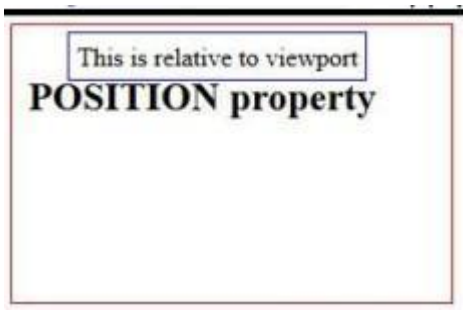
Use `position: absolute;` to apply this property and the layout will look like this:

fixed

- The **fixed** value **fixes the position of the element relative to the viewport**,which means it will always stay in the same place, regardless of how far down the page is scrolled.

  In addition, the element does not leave a blank space on the page; it will overlap with another element.

  Use `position: fixed;` to apply this property and the layout will look like this:



Notice, the ending of the black line is the start of the web page and the distance between the black line is 10px.

sticky

- The **sticky** value is **initially positioned as in the HTML source.** It is then **fixed relative to the viewport as soon as it reaches the desired position.**

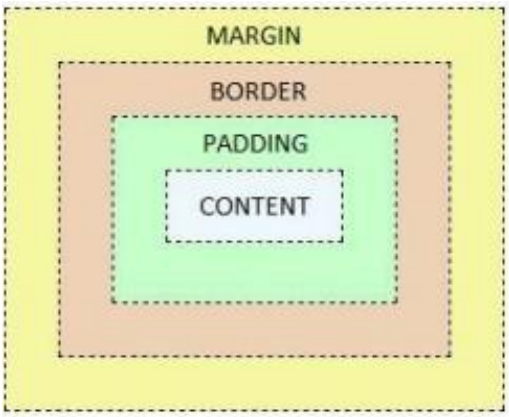
- Depending on the scroll position, a sticky element switches between relative and fixed states. It moves relative to the viewport until a certain offset is reached, at which point it "sticks" in place (like position: fixed).

Use `position: sticky;` to apply this property and the layout initially be same as that of relative positioned and move with the page on scrolling until the final position (i.e. same as fixed position) is reached.

BOX MODEL

- Box model is used to specify the layout of the elements.
- The HTML elements are considered boxes.
- **Every HTML element is surrounded by a box** in the **CSS box model**.
- It is made up of the following elements: **margins**, **borders**, **padding**, and **content**.

The image below illustrates the box model:



Box-Sizing:

- The box-sizing property defines how the width and height get applied on an element.
- It can be useful to play with box-sizing.
  E.g. An element needs to take a precise amount of space on a page, with its padding and border included.

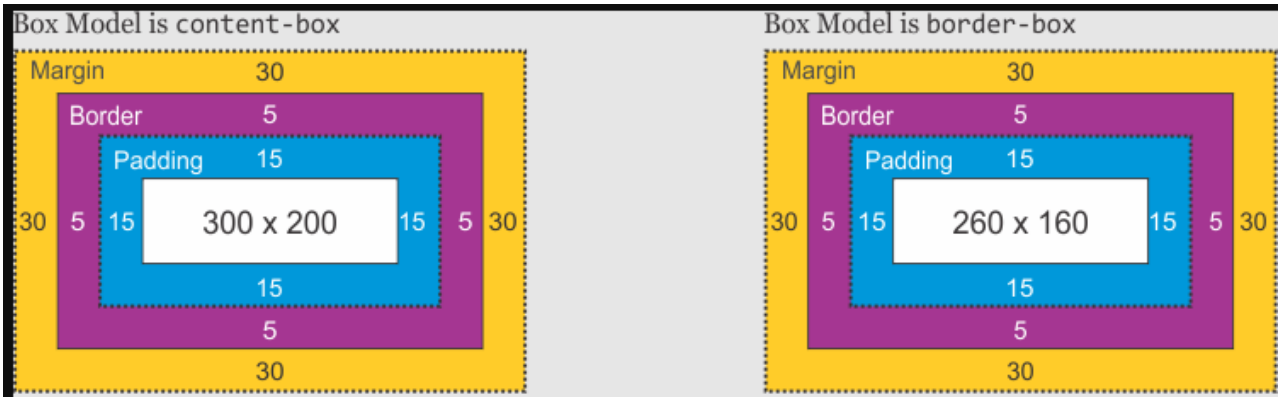
It can take two possible values:

content-box:

- The width and height of the element include only the content.i.e the border, padding and margin are not part of the width or height.
- This is the default value.

border-box:

- The width and height of the element include content, padding and border.




```
.div{
    width: 300px;
    height: 200px;
    padding: 15px;
    border: 5px solid grey;
    margin: 30px;
    box-sizing: content-box;
}
```

```
.div{
    width: 300px;
    height: 200px;
    padding: 15px;
    border: 5px solid grey;
    margin: 30px;
    box-sizing:border-box;
}
```

MIN/MAX WIDTH

- The width property is used to set the width of the element, to a specific size. But the size of the becomes fixed with this and this brings the problem of smaller devices. The browser then gets a scrollbar to scroll through the entire content.

  So, to overcome this problem, CSS provides a **max-width** property.

- **max-width** property specifies the **maximum width that an element can have**. If the browser window's width becomes smaller than the width of the element, the element width adjusts with the browser width.

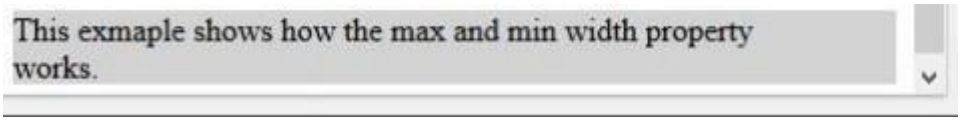
However, a small element is extremely difficult to read. So, another property **min-width** is provided by the CSS that specifies the minimum width that the element can have.

Eg.we have two divs like this:

```
<div id="div1">
    <div id="div2">This example shows how the max and min width property works.
    </div>
</div>
```

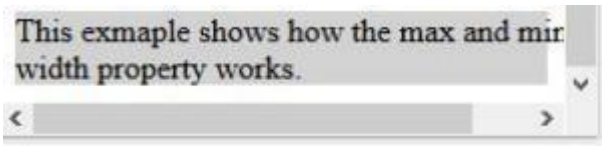
and the CSS is applied to them:

```
#div1 {
    background-color: lightgrey;
}

#div2 {
    max-width: 400px;
    min-width: 300px;
}
```



will show like this when the browser width is more than 400px

and like this when the browser width is less than 300px:



OVERFLOW

- The **overflow** property defines **what will happen if the content of an element overflows**, i.e. the height or width of the content is larger than the element's height or width.
- When the content of an element is too large to fit in a specified area, this property adds a scroll bar or clips the content.

The values that the overflow property can take are:

- **visible** - This is the normal setting. The content overflows here and is seen outside the box
- **hidden** - only the content that fits inside the box is visible and the overflow is clipped
- **scroll** - all the content is visible through a scroll-bar added to the box
- **auto** - a scroll-bar gets added if content overflows

E.g. when we a larger content than that can be fitted inside the element like:

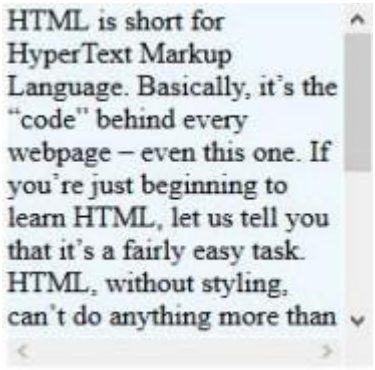
```
<div>HTML is short for HyperText Markup Language. It's basically the "code" that
powers every website, including this one. If you're just getting started with
HTML, rest assured that it's a relatively simple task. HTML, without styling,
can't do anything more than setting a layout, drawing a table, or creating frames
- but it is handy as it helps you structure the content correctly, which is
important when you sit down to add style to your HTML.</div>
```

with fixed height and width as:

```
div{
    background-color: aliceblue;
    width: 200px;
    height: 200px;
```

```
    overflow: scroll;
}
```

will show the box in the browser like this:



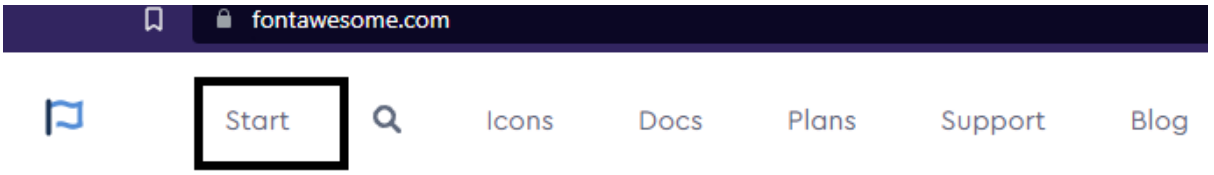
NOTE: *Only block elements with a specified height can use the overflow property.*

How to link icons on the web page?

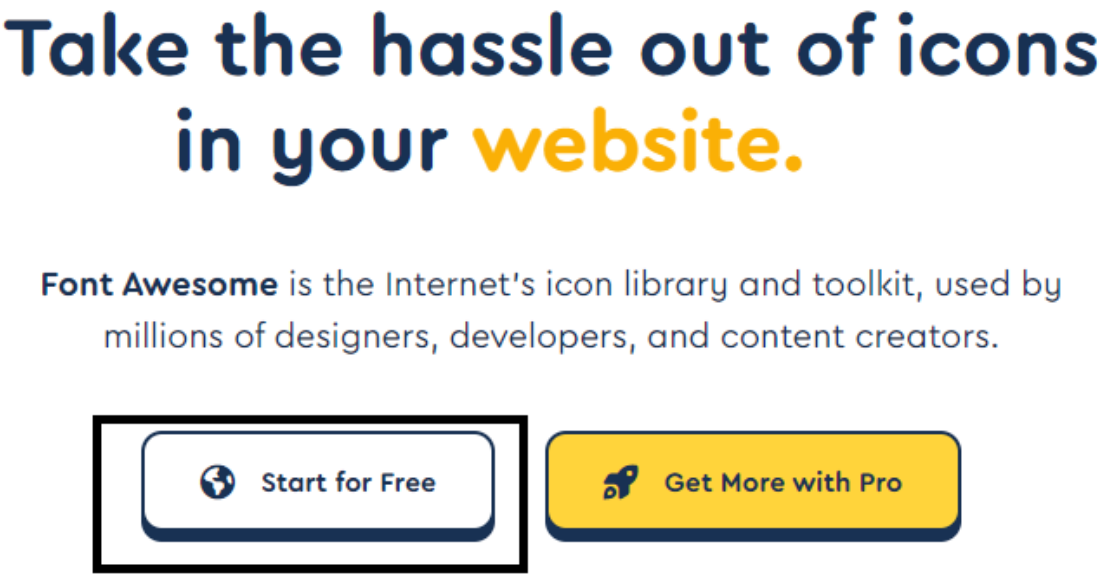
You can link various icons from the fontawesome website, where you can pick icons instead of images since they load fast and take less space than images.

Steps to follow:

1) Go to this link - https://fontawesome.com/ and click on start.

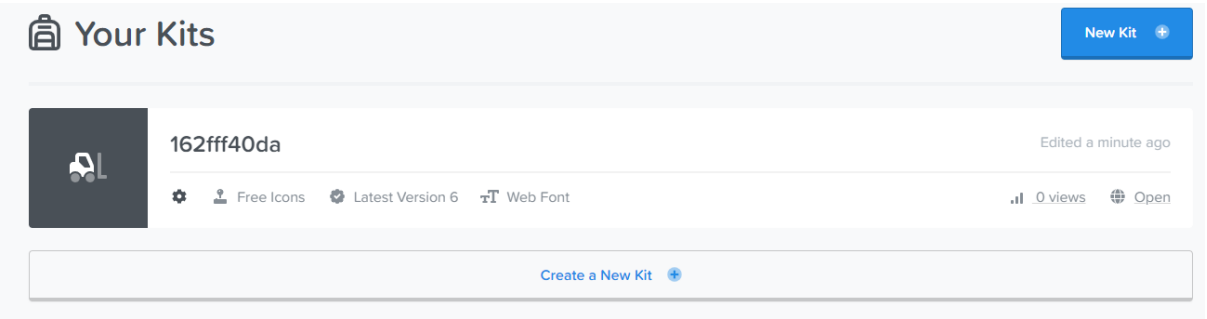


2) Now click on start free



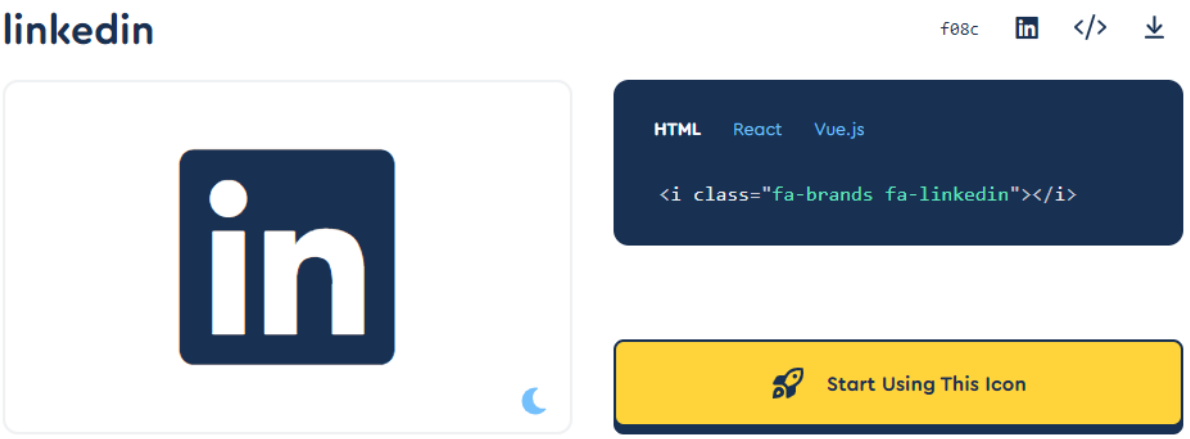
3) Enter your email for starting with a free kit and confirm your font awesome account email address.

4) Now click on your kits.



5) Now add your kit's code into the <head> section in the HTML file.

6) Now search for the icons over there and add this code to your file like below.
****



7) Now you can add style to the icon as CSS classes to an HTML tag.

We have already learnt about some CSS Selectors. But that’ s not it, there are other selectors as well which you can use while building the project. So here is a small reading assignment for you to explore them on your own. Do give them a read as you will then have to answer MCQs based on them!

Below are some links to read about selectors –

1. https://code.tutsplus.com/tutorials/the-30-css-selectors-you-must-memorize--net-16048

2. You can learn about CSS selectors from this game - http://flukeout.github.io/

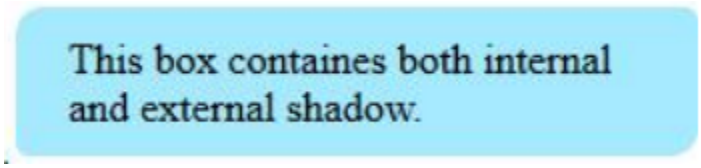
You can also create custom attributes using the 'data-*' attribute. You will find about it in the links above.

Some More Styles

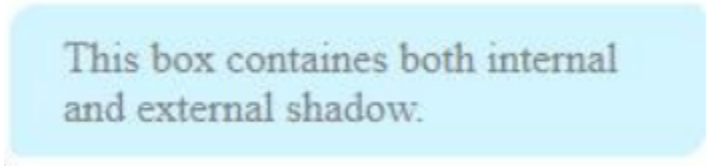
OPACITY

The **opacity** property is used to **set the transparency of an element**. This can take a value ranging from **(0.0 - 1.0).** The lower the value, the more transparent the element will become.

Eg., applying `opacity: 0.5;` to the element below:



will show the element like this when opacity gets applied:



So, when adding transparency to the background of an element, all of its **child elements will also inherit the same transparency.** This makes the text inside transparent as well.

You can use the '**rgba()**' property to provide color along with opacity.

Look at the example below where we set opacity along with color using ***rgb()*** property:

```
#box {
    width: 100px;
    height: 100px;
    background-color: rgb(255, 0, 0);opacity: 0.6;
}
```

With ***rgba(),*** you can give the opacity value, along with the rgb values:

```
#box {
    width: 100px;
    height: 100px;
    background-color: rgba(255, 0, 0, 0.6);
}
```

Try this on your own!!

TRANSITION

The transition property is used to change the value of a property to some other value over a given duration. You can provide multiple transitions to a single element by using a comma.

The CSS syntax is -

`transition: property duration timing-function delay;`

The transition property is a shorthand property for:

- **transition-property** -specifies the name of the CSS property to apply a transition to
- **transition-duration** - specifies the seconds it would take to complete the transition
- **transition-timing-function** - specifies the speed of the transition over the duration
- **transition-delay** - specifies the wait before the start of the transition effect

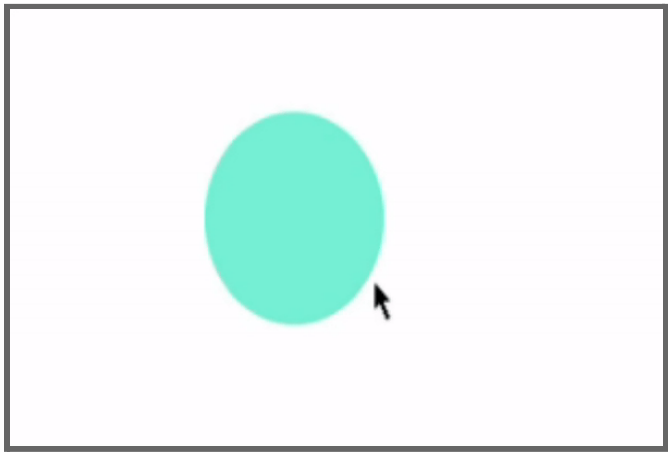
Eg: You can apply all these properties like this:

```
transition: 0.5s ease-in-out;
```

Or specify styles separately:

```
transition-delay: 0.5s;
transition-duration: 1s;
transition-timing-function: ease-in-out;
```

will change the look of the element on hovering like this:



You'll learn about transitions in a lot more detail, later in the course.

EXTRA:

You can see other 'transition-timing-function' value from the below link :

https://developer.mozilla.org/en-US/docs/Web/CSS/transition-timing-function

BOX SHADOW

The box-shadow property is used to produce a shadow-like effect for an element. You can also, give multiple shadows to an element.

The CSS syntax for attaching shadow to element is -

`box-shadow: none | h-offset v-offset blur spread color;`

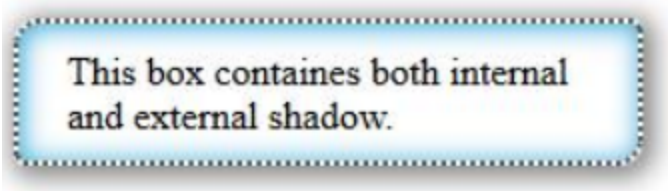
The meaning of the above options is -

- **none** - This is the **default** value. No shadow is displayed
- **h-offset** - this is a **required** value. It sets the horizontal point of the start of the shadow. The value can be either a positive or negative number.
- **v-offset** - this is also a **required** value. It sets the vertical point of the start of the shadow. The value can be either a positive or negative number.
- **blur** - this option is **optional**. This blurs the shadow. The higher the number, the more blurred the shadow will be
- **spread** - this option is also **optional**. This sets the size of the shadow. The value can be either a positive or negative number.
- **color** - this option is also optional. This sets the color of the shadow. The default value will be the text color.

Eg., adding the show to a paragraph like this:

```
p {
    border: 2px dotted #555555;
    box-shadow: 1px 1px 10px 1px #3faddf inset, 2px 2px 10px 3px #AAAAAA;
}
```

will show the para like:



Now, you can see 2 shadows -

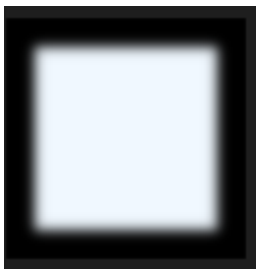
- One is outside the border.
- Other is inside the border.

We can provide **inner shadow** using the **'inset' option**, which is **optional**. This option changes the shadow from an outer shadow to an inner shadow

For example:

```
<html>
    <head>
        <style>
            #box{
               height: 100px;
               width: 100px;
               background-color: aliceblue;
               box-shadow: 0px 0px 5px 12px inset black;
            }
        </style>
    </head>
    <body>
        <div id="box">

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

Output: You can see that now the shadow is set using the ***inset*** option.



Flex

INTRODUCTION

The Flexbox layout provides an efficient way to layout, align and distribute space among items in a container. This is really helpful when the size of the elements in the container is unknown and/or dynamic.

Using the flex in a container gives the **ability to alter its item's width/height and order as well to best fit in the space available**. A flex container expands items to fill available free space or shrinks them to prevent overflow.

The flex is a value for the display property. It has to be **provided in the container** for the flex to work. Only if it is defined inside the container, flex properties will work. **Flex properties are defined on the child elements**.

To make the container to be flex, add this property in the container: `display: flex;` or `display: inline-flex;` for the inline variation.

FLEX-DIRECTION

The **flex-direction** property defines in which **direction** the container lays out the flex-items. It may take 4 values:

- row - default
- column
- row-reverse
- column-reverse

The flex-direction property lays out the flex-items either **horizontally, vertically or reversed** in both directions. The horizontal axis is called the main-axis and the vertical axis is called **cross axis.** Eg., for the layout like this:

```
<div class="flex-container">
    <div id="div1">div1</div>
    <div id="div2">div2</div>
    <div id="div3">div3</div>
</div>
```

we can use like this:

```
.flex-container {
    display: flex;
    width: 250px;
    height: 250px;
    background-color: lightgrey;
}
.flex-container div {
    width: 70px;
    height: 70px;
}
#div1 {
    background-color: #7ff9ae;
}
#div2 {
    background-color: #76bbfc;
}
#div3 {
    background-color: #ff7f8e;
}
```

It'll look like this:

- **row**

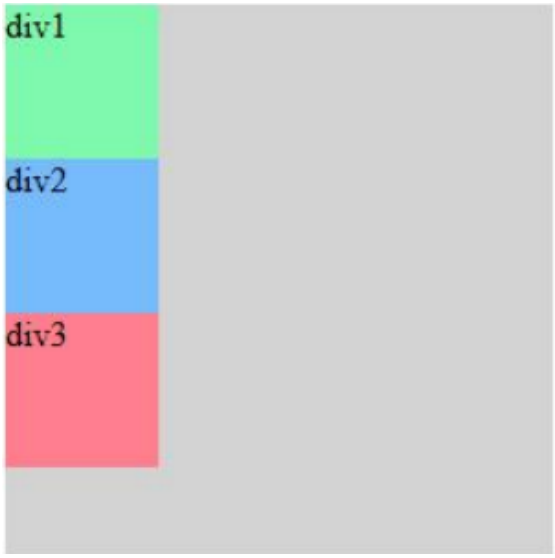
  The ***row*** value stacks the flex items horizontally, from left to right.

  For **forward direction**, add the `flex-direction: row;` to the flex-container class.

  It will not change anything as **this is the default value** and the layout will still look the same.

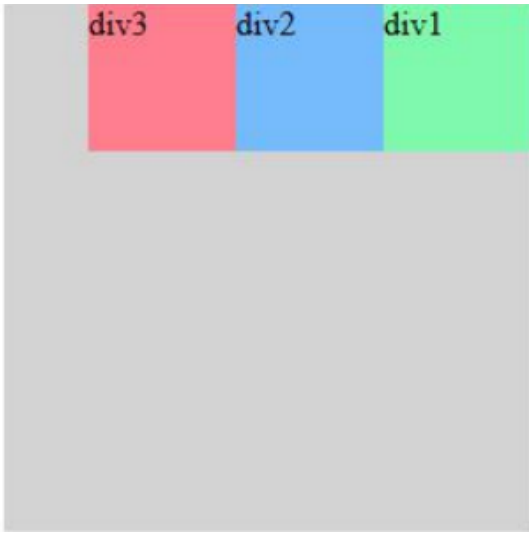
- **column**

  The ***column*** value stacks the flex items vertically, from top to bottom. For **downward direction,** add the `flex-direction: column;` to the flex-container class and the layout will look like this:



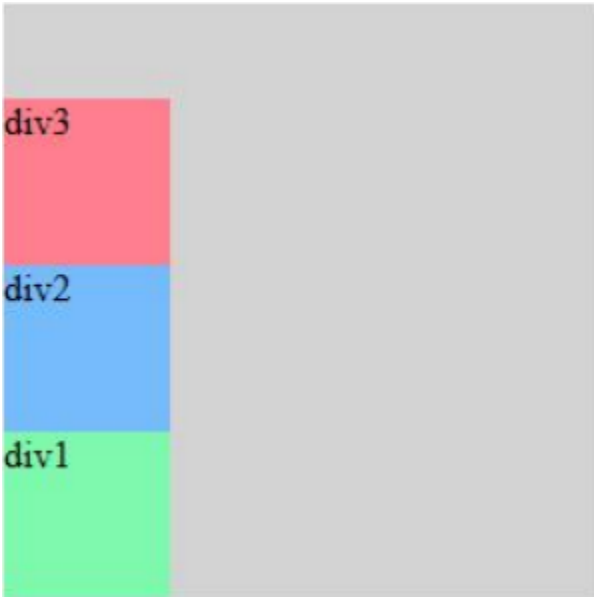
row-reverse

The ***row-reverse*** value stacks the flex items horizontally, from right to left. For **backward direction,** add the `flex-direction: row-reverse;` to the flex- container class and the layout will look like this:



- **column-reverse**

  The **column-reverse** value stacks the flex items **vertically**, from bottom to top. For **upward direction**, add the `flex-direction: column-reverse;` to the flex- container class and the layout will look like this:



ORDER

The order property is used to **specify the order of the flex items** in the flex container. This property value must be a whole number. By default, the number is 0(zero). The higher the number the latter would the flex item appear in the flex container.

The flex items are displayed in the order like:

- First, the items ***not having order property*** or ***order:0;*** property is displayed in sequence in which they appear in the source order.
- then the items are displayed in ***ascending order of the value of the order property.*** The items having the ***same order value*** are displayed in the sequence in which they appear in the source order.

Eg., for the layout:

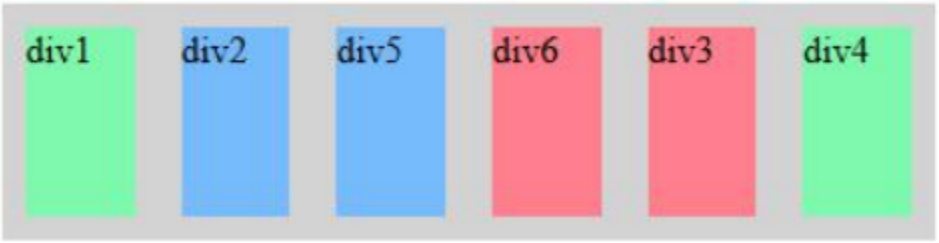
```
<div class="flex-container">
    <div class="div1">div1</div>
    <div class="div2" style="order:0;">div2</div>
    <div class="div3" style="order:3;">div3</div>
    <div class="div1" style="order:3;">div4</div>
    <div class="div2">div5</div>
    <div class="div3" style="order:2;">div6</div>
</div>
```

the css is applied to this as:

```
.flex-container {
    display: flex;
    width: 400px;
    height: 100px;
    background-color: lightgrey;
}
.flex-container div {
    width: 70px;
    min-height: 70px;
    margin: 10px;
}
.div1 {
    background-color: #7ff9ae;
}
.div2 {
    background-color: #76bbfc;
}
```

```
.div3 {
    background-color: #ff7f8e;
}
```

will display the flex items like:



FLEX WRAP

The flex-wrap property specifies whether the flex items should wrap or not. By default, flex items try to fit into one line. This property allows you to change that and allow the items to flow into multiple lines as needed with this property.

- **nowrap** - default
- **wrap**
- **wrap-reverse**

For the layout:

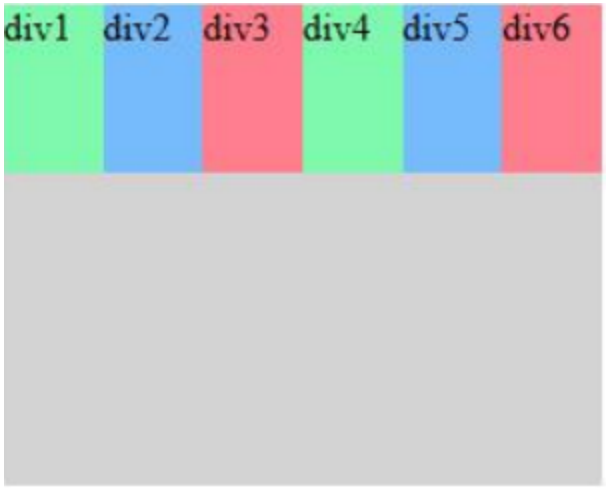
```
<div class="flex-container">
    <div class="div1">div1</div>
    <div class="div2">div2</div>
    <div class="div3">div3</div>
    <div class="div1">div4</div>
    <div class="div2">div5</div>
    <div class="div3">div6</div>
 </div>
```

and with the following styles applied to them:

```
.flex-container {
    display: flex;
    width: 250px;
    height: 200px;
    background-color: lightgrey;
}
.flex-container div {
    width: 70px;
```

```
    height: 70px;
}
#div1 {
    background-color: #7ff9ae;
}
#div2 {
    background-color: #76bbfc;
}
#div3 {
    background-color: #ff7f8e;
}
```

the layout(original container width is 250px and original total div's width is 420px) will look like this:



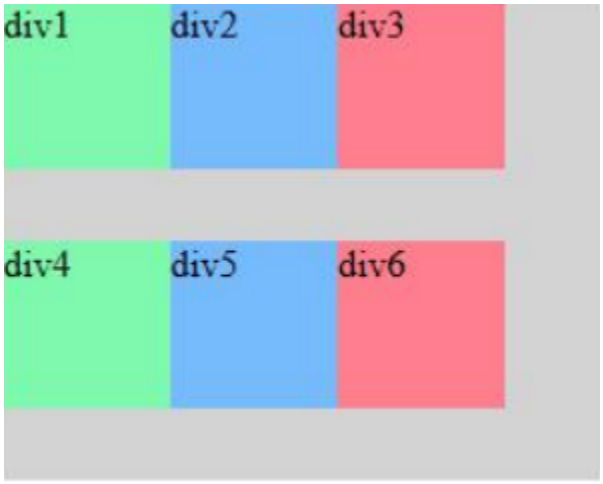
- **nowrap**

  The ***nowrap*** value specifies that the flex items will not wrap, i.e. it will make all the flex-items appear in a single line, no matter how many items are present inside the flex-container. It is a **default value** but you can use `flex-wrap: nowrap;` as well in the flex-container class.

- **wrap**

  The ***wrap*** value specifies that the flex items will wrap if necessary, i.e. if the total flex items are width is larger than the width of the container, the items will arrange themselves in the next column. The **vertical spacing of the items will be automatically decided** by this property.

Use `flex-wrap: wrap;` in the flex-container class to wrap the items like this:

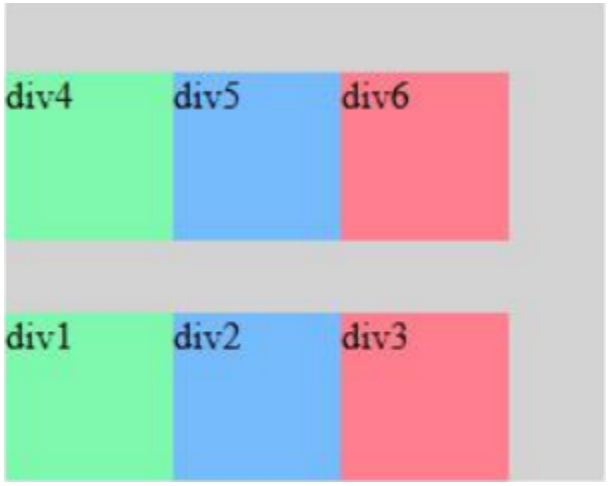


- **wrap-reverse**

  The ***wrap-reverse*** value specifies that the flex items will wrap in reverse order if necessary, i.e. if the total flex items are width is larger than the width of the container, the items will arrange themselves in the next column but the items will start from the bottom of the container.

  Use `flex-wrap: wrap-reverse;` in the flex-container

  class to wrap the items like this:



Try to use flex-direction and flex-wrap properties together and see what happens!!

FLEX GROW

The **flex-grow** property specifies the ratio with which a **flex item grows relative to the other flex items** when there is some extra space available. This controls the extent how much a flex-item grows, with respect to other flex items. This takes up any value from **0(zero) to any positive number. 0(zero) means that the flex items' width would not change.**

Eg., for a simple layout:

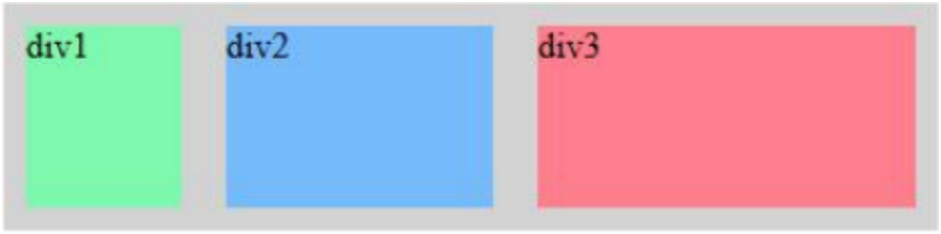
```
<div class="flex-container">
    <div id="div1">div1</div>
    <div id="div2">div2</div>
    <div id="div3">div3</div>
</div>
```

the css applied is:

```
.flex-container {
    display: flex;
    width: 270px;
    height: 100px;
    background-color: lightgrey;
}
.flex-container div {
    width: 70px;
    margin: 10px;
}
#div1 {
    background-color: #7ff9ae;
    flex-grow: 0;
}
#div2 {
    background-color: #76bbfc;
    flex-grow: 2;
}
#div3 {
    background-color: #ff7f8e;
    flex-grow: 4;
}
```

this will look like this:

Now, if the width of the flex-container is increased by 150px to **width:420px**; then the layout will now look like this:



Here, only the div2 and div3 grow, when the width of the container is increased. Also, div3 increases twice as much as div2. Now, the width of **div1 is 70px, div2 is 120px** and **div3 is 170px.**

FLEX SHRINK

The **flex-grow** property specifies the ratio with which a **flex item shrinks relative to the other flex items** when there is some extra space available. This is just the opposite of flex-grow. Here, also the range of value is from **0(zero) to any positive number.** Zero means width would not change.

Eg., for a simple layout:

```
<div class="flex-container">
    <div id="div1">div1</div>
    <div id="div2">div2</div>
    <div id="div3">div3</div>
</div>
```

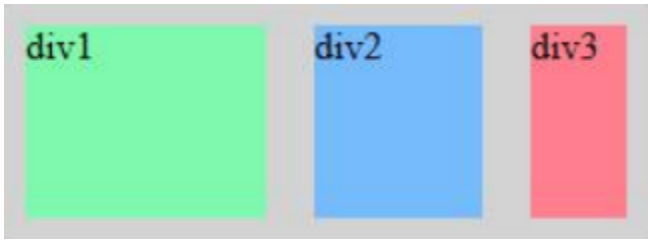
the css applied is:

```
.flex-container {
    display: flex;
    width: 360px;
    height: 100px;
    background-color: lightgrey;
}
.flex-container div {
    width: 70px;
    margin: 10px;
}
#div1 {
```

```
    background-color: #7ff9ae;
    flex-shrink: 0;
}
#div2 {
    background-color: #76bbfc;
    flex-shrink: 2;
}
#div3 {
    background-color: #ff7f8e;
    flex-shrink: 4;
}
```

this will look like this:

Now, if the width of the flex-container is decreased by **90px** to ***width:270px;,*** then the layout will now look like this:



Here, only the div2 and div3 shrinks, when the width of the container is decreased. Also, div3 decreases twice as much as div2. Now, the width of **div1 is 100px, div2 is 70px and div3 is 40px.**

FLEX - JUSTIFY CONTENT

The **justify-content** property is used to **align the flex items along the main axis.** This defines the alignment along the main axis. This property distributes extra free space inside the layout between the elements.

The justify-content property takes on any of the values below:

- **flex-start** - default
- **flex-end**
- **center**
- **space-between**
- **space-around**
- **space-evenly**

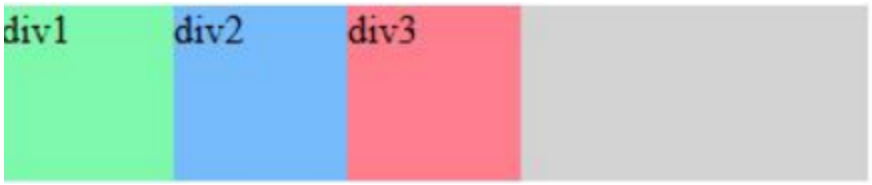
Eg., for the layout like this:

```
<div class="flex-container">
    <div id="div1">div1</div>
    <div id="div2">div2</div>
    <div id="div3">div3</div>
</div>
```

we apply this css to it:

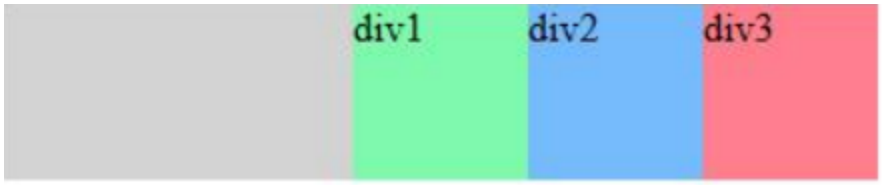
```
.flex-container {
    display: flex;
    width: 350px;
    background-color: lightgrey;
}

.flex-container div {
    width: 70px;
    height: 70px;
}

#div1 {
    background-color: #7ff9ae;
}

#div2 {
    background-color: #76bbfc;
}

#div3 {
    background-color: #ff7f8e;
}
```

the layout will look like this:

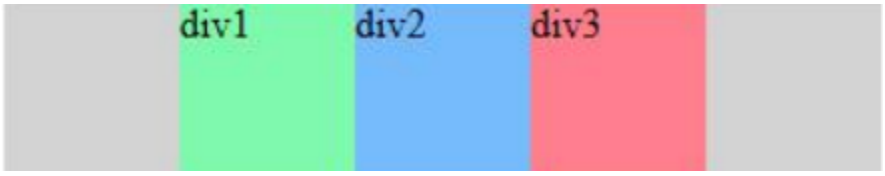


flex-start: The **flex-start** value aligns the items towards the **start of the main axis.** This is the **default value.** Else use `justify-content: flex-start;` in the flex-container class.

flex-end: The **flex-end** value aligns the items towards the **end of the main axis**. Use `justify-content: flex-end;` in the flex-container class and the flex items will look like this:



center: The **center** value aligns the items towards the **center of the main axis**. Use `justify-content: center;` in the flex-container class and the flex items will look like this:



space-between: The **space-between** value distributes the remaining **space between the items evenly along the main axis**. No space is provided towards the start of the first container and the end of the last container. So, there is some spacing between the items.

Use `justify-content: space-between;` in the flex-container class and the flex items will look like this:

space-around: The **space-around** value distributes the remaining **space around the items evenly along the main axis**.

Visually the spaces do not seem to be equal. This is because all the flex items have equal space on both sides.

Eg., suppose each item gets a spacing of 10px on both sides of each other. So, the first item will have 10px on the left side. The first item will have 10px on the right side and the second item will have 10px on its left side. This will total to 20px spacing between them. Similarly, the last item has only 10px spacing on its right side.

Use `justify-content: space-around;` in the flex-container class and the flex items will look like this:

space-evenly: The **space-evenly** value distributes the remaining **space between the items and edges of the container evenly along the main axis.**

This provides a visual look at evenly spread items inside the container.

Use `justify-content: space-evenly;` in the flex-container class and the flex items will look like this:

FLEX - ALIGN ITEMS

The **align-items** property is used to align the **flex items along the cross-axis.** The cross-axis is perpendicular to the main axis.

If the main-axis is horizontal, then the cross-axis is vertical. If the main-axis is vertical, then the cross- axis is horizontal.

Align-items can be set to any of these values:

- **flex-start**
- **flex-end**
- **center**
- **stretch -** default
- **baseline**

Eg., for the layout like this:

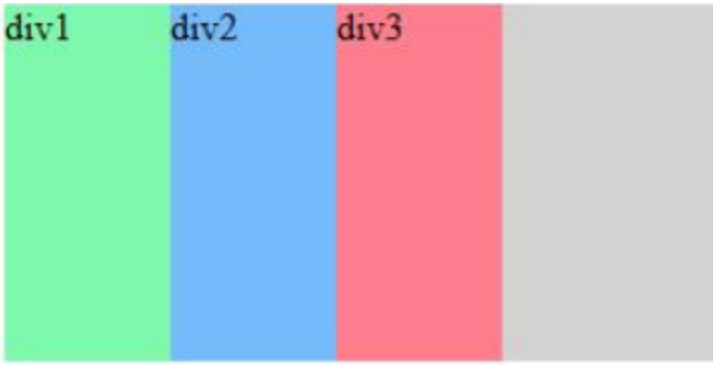
```
<div class="flex-container">
    <div id="div1">div1</div>
    <div id="div2">div2</div>
    <div id="div3">div3</div>
</div>
```

we can use like this:

```
.flex-container {
    display: flex;
    width: 300px;
    height: 150px;
    background-color: lightgrey;
}
.flex-container div {
    width: 70px;
    min-height: 70px;
}
#div1 {
    background-color: #7ff9ae;
}
#div2 {
    background-color: #76bbfc;
}
#div3 {
```

```
    background-color: #ff7f8e;
}
```

the layout will look like this:



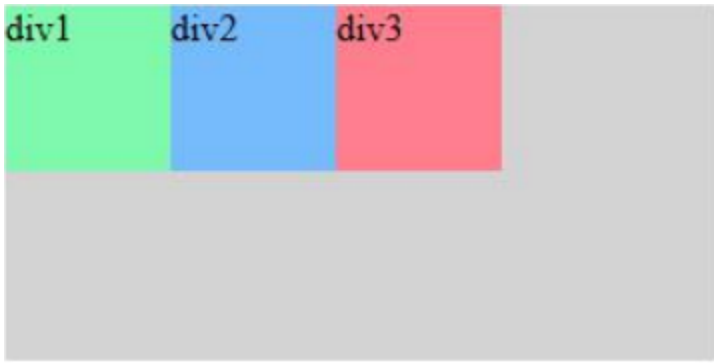
***NOTE**: Remember that the default behavior will stretch the items only if the **height of the item is not fixed.** But if the size is fixed, all the other values other than the stretch value will work.*

- **stretch**

  The **stretch** value stretches the flex items to **fill the container along the cross axis.** This is the default value. Else use `align-items: stretch;` in the flex-container class.

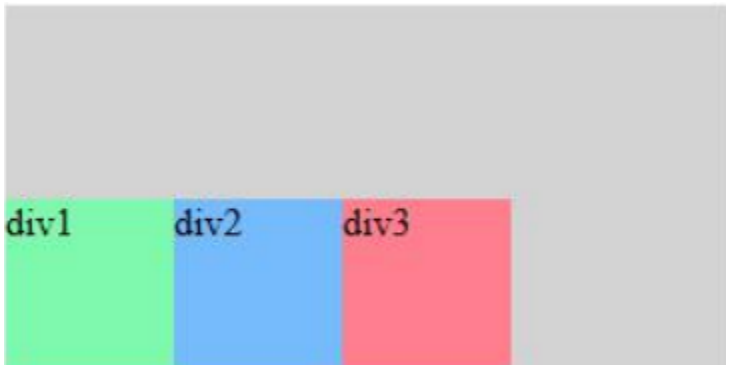
- **flex-start**

  The **flex-start** value aligns the items towards the **start of the cross axis.** Use `align-items: flex-start;` in the flex-container class and the flex items will look like this:



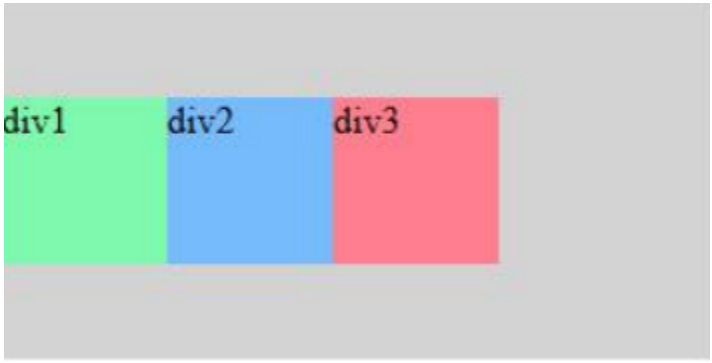
- **flex-end**

  The **flex-end** value aligns the items towards the end of the cross axis. Use `align-items: flex-end;` in the flex-container class and the flex items will look like this:



- **center**

  The **center** value aligns the items towards the **center of the cross axis.** Use `align-items: center;` in the flex-container class and the flex items will look like this:



- **baseline**

  The **baseline** value aligns with the flex items such as the **baselines of the content inside the items.** This means that the bottom base of the content inside the flex item will be the axis to align the items. To work with this property, we need to make some variations in the above code.

  The changes are highlighted:

```
<div class="flex-container">
    <div id="div1">
        <h5>h1</h5>
    </div>
    <div id="div2">
        <h2>h2</h2>
    </div>
    <div id="div3">
        <h1>h3</h1>
    </div>
</div>
```

we can use like this:

```
.flex-container {
    display: flex;
    width: 300px;
    height: 150px;
    align-items: baseline;
    background-color: lightgrey;
}
.flex-container div {
    width: 70px;
    min-height: 70px;
}
#div1 {
    background-color: #7ff9ae;
}
#div2 {
    background-color: #76bbfc;
}
#div3 {
    background-color: #ff7f8e;
}
```

the layout will look like this:

Here, see that the items are aligned based on the contents (i.e. h1, h2, and h3) baseline (or bottom of the text).

FLEX - ALIGN CONTENT

The **align-content** property is used to **align the flex lines,** i.e., multiple lines of flex items. This property aligns flex lines when there is extra space in the cross-axis, similar to how justify-content aligns individual items within the main-axis.

Align-content can be set to any of these values:

- **flex-start**
- **flex-end**
- **center**
- **space-between**
- **space-around**
- **stretch** - default

Eg., for the layout like this:

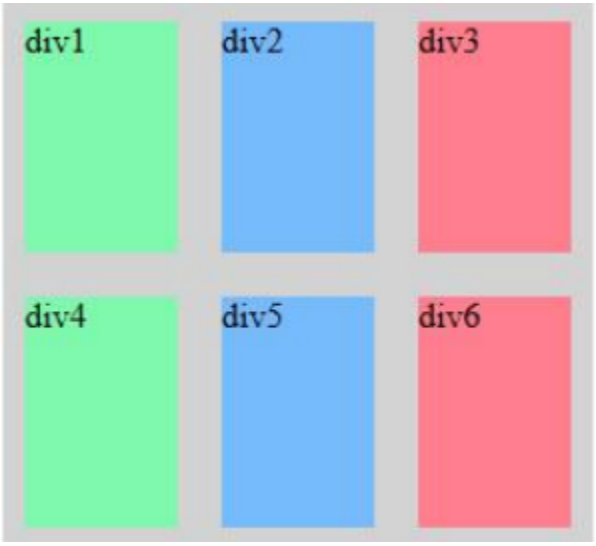
```
<div class="flex-container">
    <div class="div1">div1</div>
    <div class="div2">div2</div>
    <div class="div3">div3</div>
    <div class="div1">div4</div>
    <div class="div2">div5</div>
    <div class="div3">div6</div>
</div>
```

we apply this css to it:

```
.flex-container {
    display: flex;
    width: 270px;
    height: 250px;
    flex-wrap: wrap;
    background-color: lightgrey;
```

```
}
.flex-container div {
    width: 70px;
    min-height: 70px;
    margin: 10px;
}
.div1 {
    background-color: #7ff9ae;
}
.div2 {
    background-color: #76bbfc;
}
.div3 {
    background-color: #ff7f8e;
}
```

the layout will look like this:



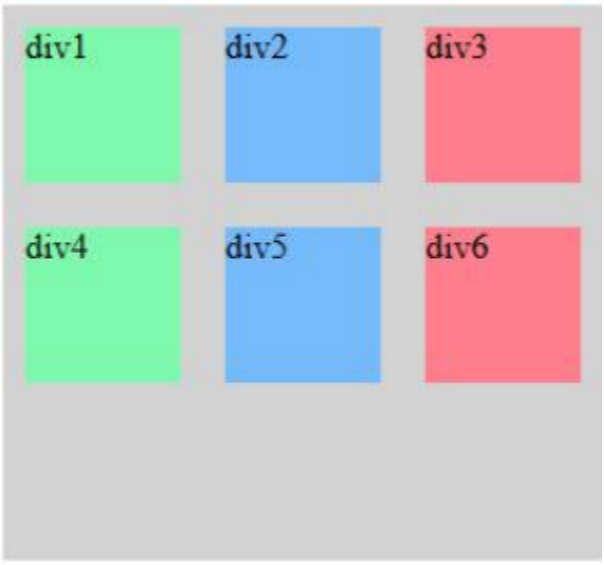
***NOTE**: This property works only when there are multiple lines of flex items. So, it has no effect when there is only one line of flex items.*

- **stretch**

  The **stretch** value stretches the flex items present in multiple lines to **fill the container equally along the cross axis.** This is the default value. Else use `align-content: stretch;` in the flex-container class.

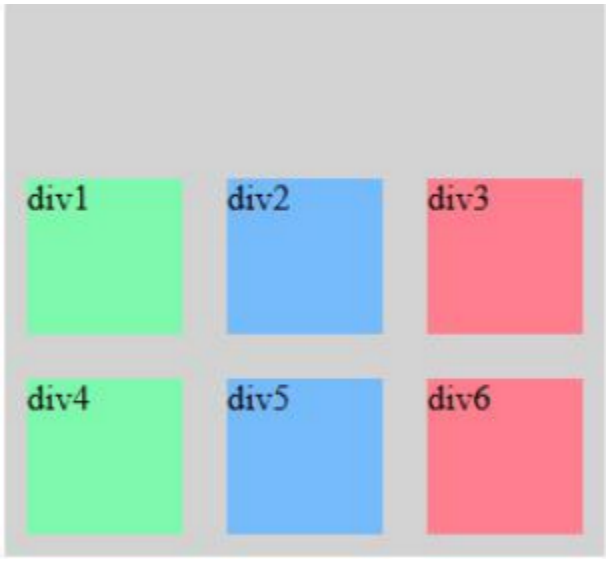
- **flex-start**

The **flex-start** value aligns the multiple lines of flex items towards the **start of the cross axis.** Use `align-content: flex-start;` in the flex-container class and the flex items will look like this:



- **flex-end**

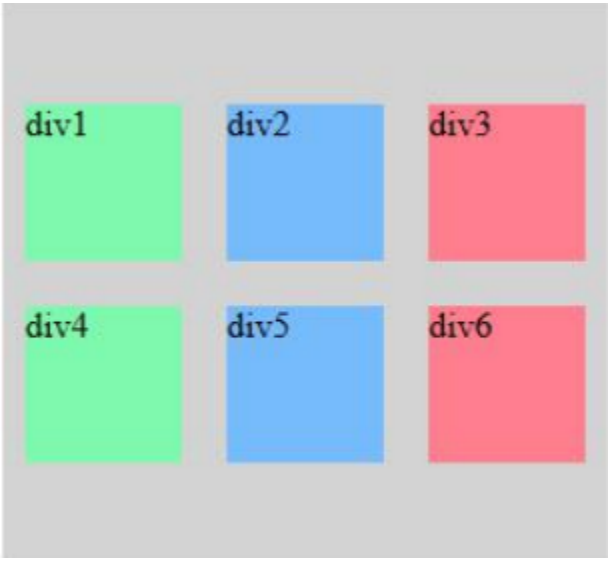
  The **flex-end** value aligns the multiple lines of flex items towards the **end of the cross axis.** Use `align-content: flex-end;` in the flex-container class and the flex items will look like this:



- **center**

  The **center** value aligns the multiple lines of flex items towards the **center of the cross axis.**

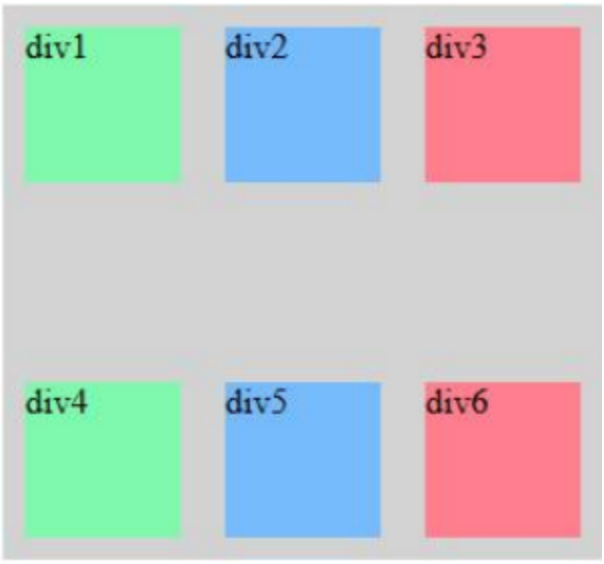
  Use `align-content: center;` in the flex-container class and the flex items will look like this:



- **space-between**

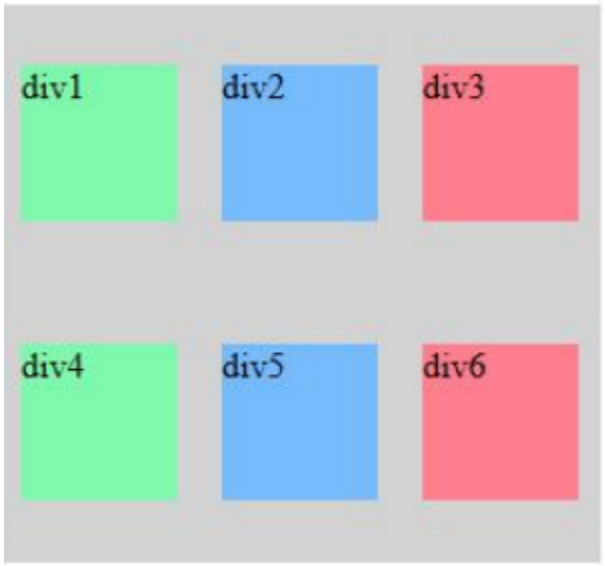
  The **space-between** value distributes the remaining **space between the multiple lines of flex items evenly along the cross axis.** No space is provided towards the start of the first container and the end of the last container. The space at the top and bottom of the container is because of the margin of the flex items.

  Use `align-content: space-between;` in the flex-container class and the flex items will look like this:



- **space-around**

  The **space-around** value distributes the remaining **space around the multiple lines of flex items evenly along the main axis.** Use `align-content: space-around;` in the flex-container class and the flex items will look like this:



FLEX - ALIGN SELF

The **align-self** property is used for the **alignment of selected flex items** inside the flex container. This overrides the default alignment set by the flex container.

Align-self can be set to any of these values:

- **auto** - default
- **flex-start**
- **flex-end**
- **center**
- **stretch**
- **baseline**

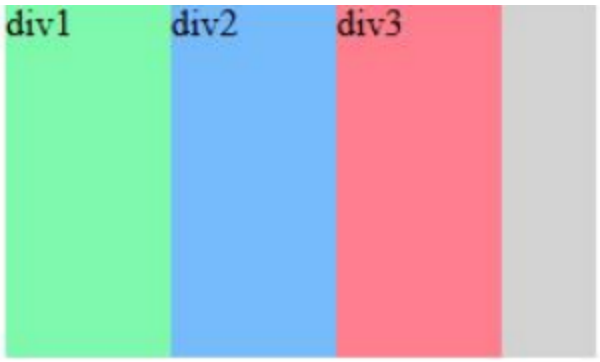
Eg., for this simple layout:

```
<div class="flex-container">
    <div id="div1">div1</div>
    <div id="div2">div2</div>
    <div id="div3">div3</div>
</div>
```

we apply this to it:

```
.flex-container {
    display: flex;
    width: 250px;
    height: 150px;
    background-color: lightgrey;
}
.flex-container div {
    width: 70px;
    min-height: 70px;
}
#div1 {
    background-color: #7ff9ae;
}
#div2 {
    background-color: #76bbfc;
}
#div3 {
    background-color: #ff7f8e;
}
```

and it will look like this in the browser:

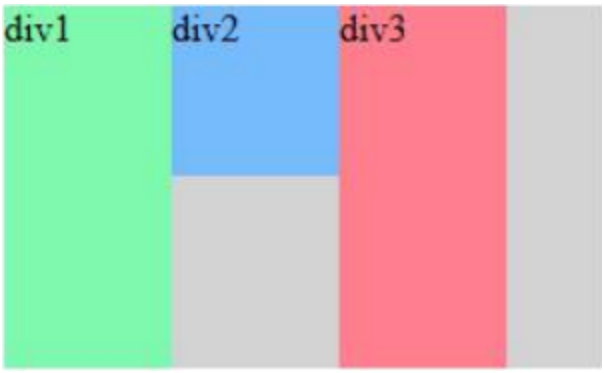


- **auto**

  The auto value inherits the flex container align-items property, or **by default stretch** is applied if the align-items property is not set. This is the default value. Else use `align-self: auto;` in the flex-container class.

- **flex-start**

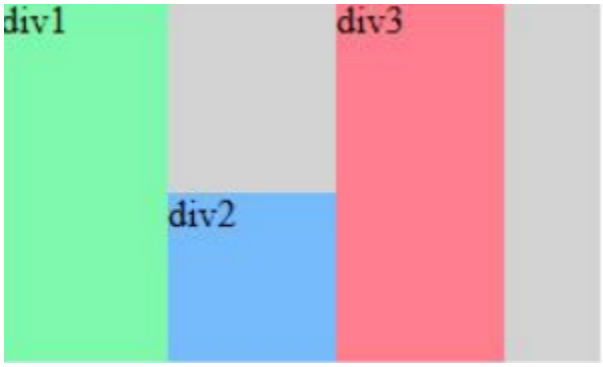
  The **flex-start** value aligns the selected flex items towards the start of the cross axis. Use `align-self: flex-start;` with the selected flex items and it will look like this:



  Here, the property is applied to div2.

- **flex-end**

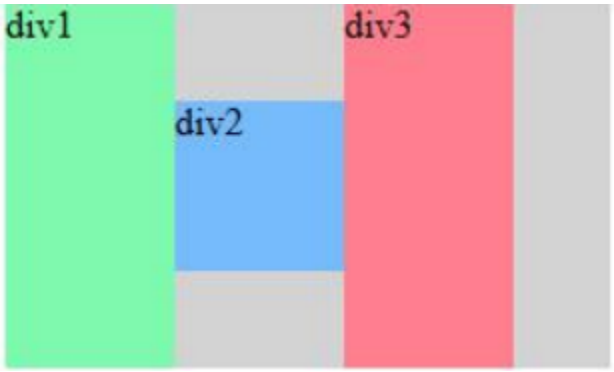
The **flex-end** value aligns the selected flex items towards the end of the cross axis. Use `align-self: flex-end;` with the selected flex items and it will look like this:



Here, the property is applied to div2.

- **center**

The **center** value aligns the selected flex items towards the **center of the cross axis.** Use `align-self: center;` with the selected flex items and it will look like this:

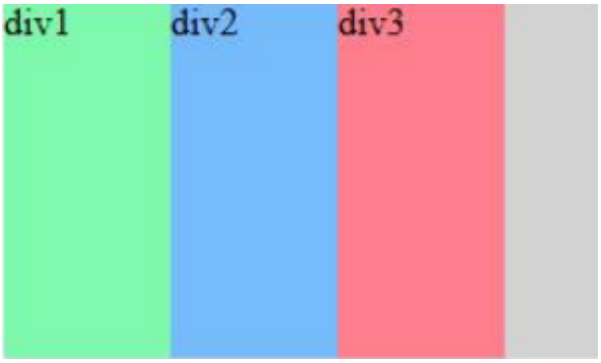


Here, the property is applied to div2.

- **stretch**

The **stretch** value stretches the selected flex items to **fill the container along the cross axis.**

Use `align-self: stretch;` with the selected flex items and it will look like this:



For this example, the auto is also set to stretch by default. That's why both auto and stretch values are the same.

- **baseline**

The **baseline** value aligns the flex items such as the **baselines of the content inside the items.** This means that the bottom base of the content inside the flex item will be the axis to align the items.

To see this property work, we need to make some variation in the above code. The changes are highlighted:

```
<div class="flex-container">
    <div id="div1">
        <h5>h1</h5>
    </div>
    <div id="div2">
        <h2>h2</h2>
    </div>
    <div id="div3">
        <h1>h3</h1>
    </div>
</div>
```

we can use like this:

```
.flex-container {
    display: flex;
    width: 300px;
    height: 150px;
```

```
    background-color: lightgrey;
}
.flex-container div {
    width: 70px;
    min-height: 70px;
}
#div1 {
    background-color: #7ff9ae;
    align-self: baseline;
}
#div2 {
    background-color: #76bbfc;
}
#div3 {
    background-color: #ff7f8e;
    align-self: baseline;
}
```

the layout will look like this:

Here, h2 is not given the baseline property. Therefore, it is not aligned and is stretched by default.

FLEX BASIS

The **flex-basis** is used to specify an initial **length to the flex-item.** The length of the item can be provided in absolute as well as in relative values. It will first apply the length of the item before applying flex-grow or flex-shrink to the items.

Eg., for a simple layout:

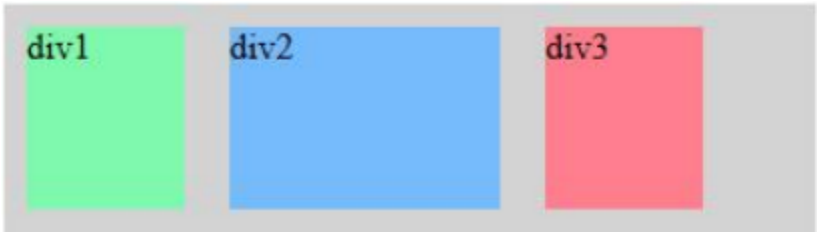
```
<div class="flex-container">
    <div id="div1">div1</div>
```

```
    <div id="div2">div2</div>
    <div id="div3">div3</div>
</div>
```

the css applied is:

```
.flex-container {
    display: flex;
    width: 360px;
    height: 100px;
    background-color: lightgrey;
}

.flex-container div {
    width: 70px;
    margin: 10px;
}

#div1 {
    background-color: #7ff9ae;
}

#div2 {
    background-color: #76bbfc;
    flex-basis: 120px;
}

#div3 {
    background-color: #ff7f8e;
}
```

this will look like this:



SHORTHAND PROPERTIES OF FLEX

Shorthand properties are used to **combine some flex properties together** so that they can be defined in one line.

- **Flex Flow:** The **flex-flow** property is a shorthand property to combine **flex-direction** and **flex-wrap** properties in one property.

  CSS Syntax is - `flex-flow: flex-direction flex-wrap | initial | inherit;`

  The meaning of the other 2 values is:

  - **initial** - represents flex-flow property as (row nowrap)
  - **inherit** - inherits this property from its parent element

- **Flex**

  The **flex** property is a shorthand property to combine the flex child properties, i.e. **flex-grow**, **flex-shrink** and **flex-basis**.

  CSS Syntax is - `flex: flex-grow flex-shrink flex-basis | auto | initial | none | inherit;`

  The meaning of the other 4 values is:

  - auto - represents flex property as (1 1 auto)
  - initial - represents flex property as (0 1 auto)
  - none - represents flex property as (0 0 auto)
  - inherit - inherits this property from its parent element

Reading Assignment

Flex

Now that we have studied flex in depth, I hope you have some confidence now. Here is another fun way to learn flex so you can try this –

https://flexboxfroggy.com/

And this is another one which you can go through -

https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Grid

The **CSS Grid Layout** provides a grid-based(rows and columns) layout system, which makes it easier to design web pages without having to use floats and positioning.

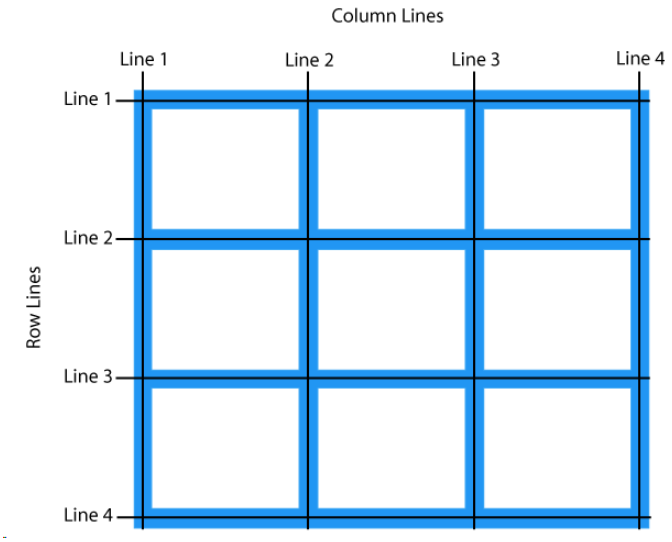
A grid layout consists of a **parent element**, with **one or more child elements**, just like flex.

You can make an element a grid container by setting the display property to ***grid*** or ***inline-grid***. All direct children of the grid container then becomes **grid items**.

Some terminologies in grid are -

- Columns - the vertical line of grid items.
- Row - the horizontal line of grid items.
- Gaps - the space between each column/row.

You can see Grid Layout in image below. The blue(darker) color shows the gap. You can use the **grid-template-columns** property to define the number of columns in your grid layout, and it can also define the width of each column.



Below are some links to learn about grid -

- *This is a guide for Grid Layout, explained using examples - https://learncssgrid.com/*
- *This contains videos to learn Grid Layout and are absolutely free - https://cssgrid.io/*
- *This link contains step by step way to learn Grids - https://learn.freecodecamp.org/responsive-web-design/css-grid/*
- *This is another guide - https://css-tricks.com/snippets/css/complete-guide-grid/*

Nth Pseudo class

The **:n-th** child and **:n-th type** pseudo-classes are interesting compared to other types of pseudo-classes because they select elements based on their position among a group of siblings.

Whereas some pseudo-classes (e.g hover, active, target) select a specific state of an element.

The :nth-child() Pseudo class:

A more specific way to select specific children is using nth-child. Where n is represented by a single argument, and it can be either a number, a keyword, or a formula.

Important components to consider for the nth-child pseudo-class:

- The element/elements selected that will have the pseudo-class applied to it/them.
- The value passed to the pseudo-class

:nth-child() Syntax-

There are few options for what values you can pass the nth-child pseudo-class.

- **:nth-child(odd/even)**: The keyword odd or even can be passed to select whose numeric position is odd and even.

- **:nth-child(An+B):** It can be read as the An+Bth element of a list.

For e.g. 2n+1 where n will start from 0 and increment from there.
It will iterate through your selected element by updating the values: 2(0)+1 = 1, 2(1)+1= 3, 2(2)+2 = 4, and so on.

e.g.

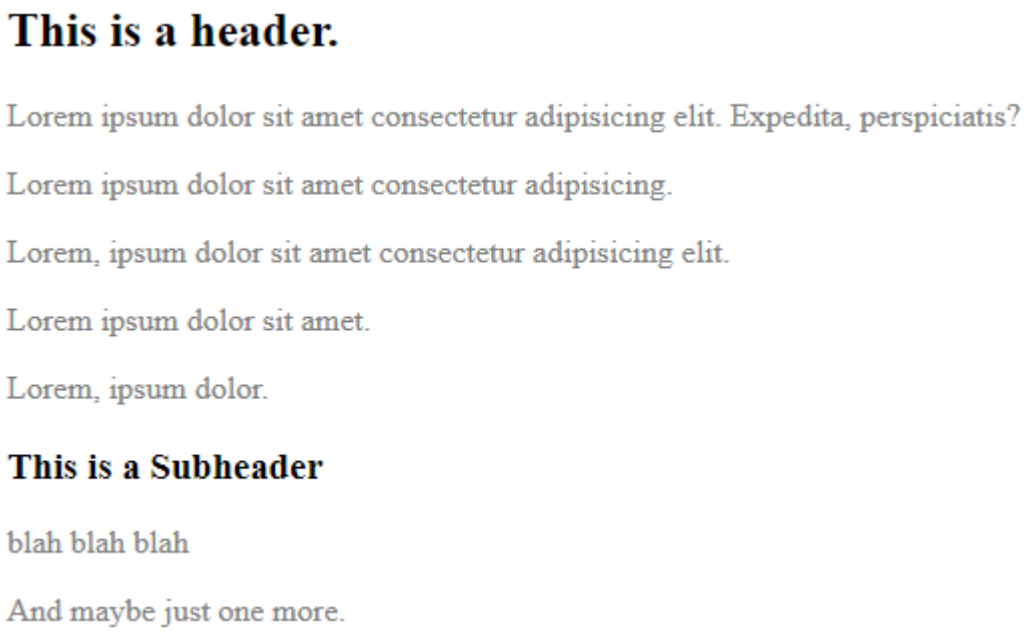
```
<div>
        <h1>This is a header.</h1>
        <p>Lorem ipsum dolor sit amet consectetur adipisicing
elit. Expedita, perspiciatis?</p>
        <p>Lorem ipsum dolor sit amet consectetur
adipisicing.</p>
```

```
        <p>Lorem, ipsum dolor sit amet consectetur adipisicing
elit.</p>
        <p>Lorem ipsum dolor sit amet.</p>
        <p>Lorem, ipsum dolor.</p>
        <h3>This is a Subheader</h3>
        <p>blah blah blah</p>
        <p>And maybe just one more.</p>
    </section>
```

Now apply the following style on the paragraph element and make the font color grey like :

```
div p {
    color: grey;
}
```

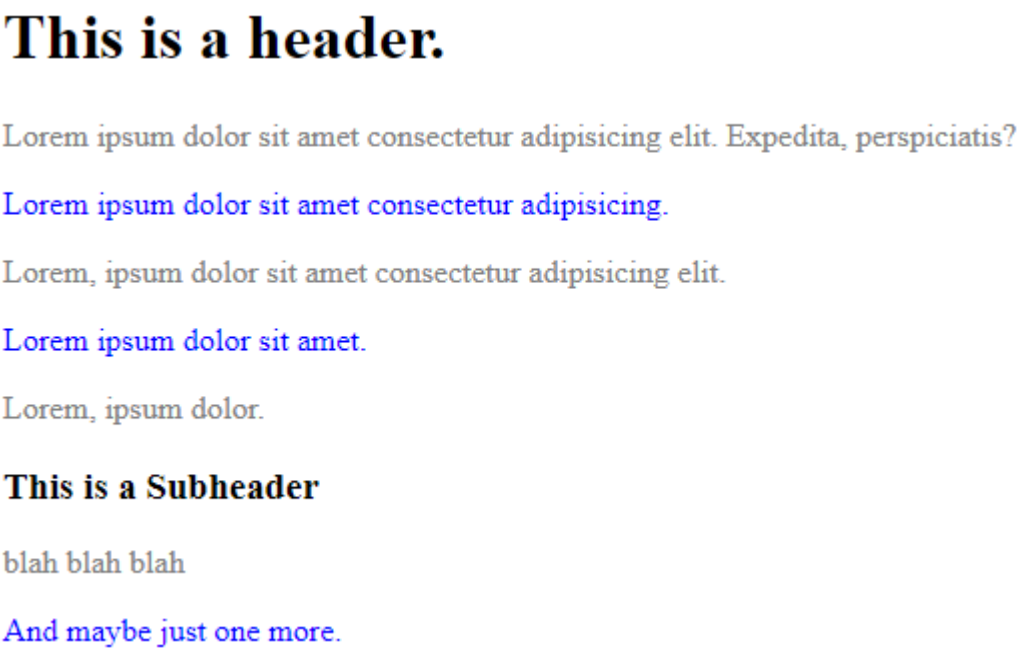
Will show up as:



Let's say you want every odd-numbered paragraph element to be a green color with the help of the **nth-child** pseudo-class.

```
div p:nth-child(odd) {
    color: blue;
}
```

Will show up as:



Now our odd-numbered paragraphs are blue in color. Did you notice what happened after the sub-header?
The grey color was repeated and then switched back to blue.
Let's look at why that happened.

Determining Which Elements are Selected with:

In the above example, the paragraphs that match our **p:nth-child(odd)** styling have to **nth-child** meet the following criteria:

- They are an odd child of the parent element.
- They are paragraph elements.

The odd or even check looks at all the children in the parent element and then looks to select all the paragraphs that are considered odd elements.

To know more about pseudo-class and pseudo-elements click here
here

Additional Notes

Z-index

- Z-index property in CSS is used to specify the stacking order of the positioned elements (elements whose position value is either fixed, absolute, relative, or sticky).
- Z-index only affects positioned elements. Setting a z-index on an unpositioned element does nothing.

Syntax:

z-index : auto | number;

Possible values

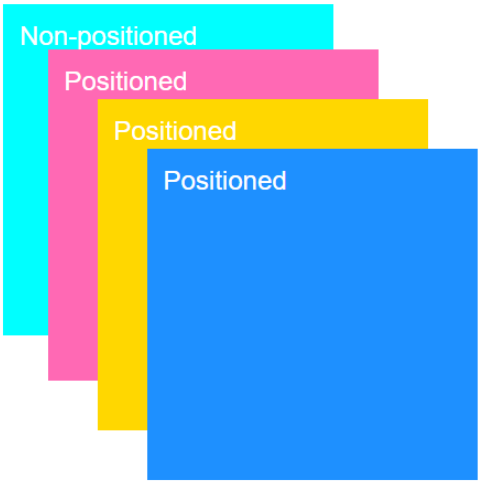
- **auto:** (default) It means that the order of the stack is equivalent to the parent.
- **number:** It means that the element's stack level is set to the given value. It also allows negative values.
  E.g. When no Z-index is applied.
  HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Z-index Demonstration</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="pink">Positioned
        <div class="orange">Positioned</div>
    </div>
    <div class="blue">Positioned</div>
    <div class="cyan">Non-positioned</div>
</body>
</html>
```

CSS:

```css
div {
    width: 200px;
    height: 200px;
    padding: 10px;
    box-sizing: border-box;
    font-family: Arial, sans-serif;
    color: white;
}
.pink, .blue, .orange {
    position: absolute;
}
.pink {
    background-color: #ff69b4;
    top: 35px;
    left: 35px;
}
.orange {
    background-color: #ffd700;
    top: 30px;
    left: 30px;
}
.blue {
    background-color: #1e90ff;
    top: 95px;
    left: 95px;
}
.cyan {
    background-color: #00ffff;
}
```

Output:



In this case, the stacking order is determined by the HTML structure. The cyan div appears on top because it's last in the HTML and not positioned.

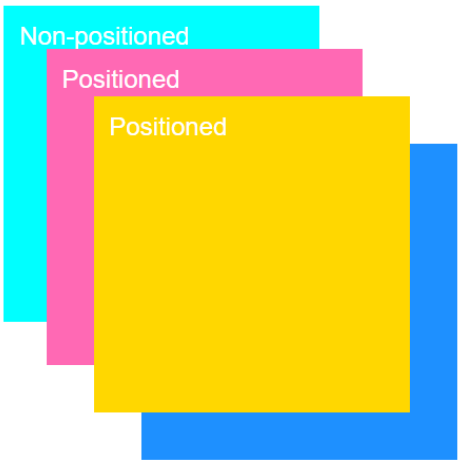
Suppose you want to change the stacking order of these elements by using the z-index property.

- An element with a higher z-index will be displayed in front of an element with a lower z-index.

Apply this styling:

```
div {
    width: 200px;
    height: 200px;
    padding: 10px;
    box-sizing: border-box;
    font-family: Arial, sans-serif;
    color: white;
}
.pink, .blue, .orange {
    position: absolute;
}
.pink {
    background-color: #ff69b4;
    top: 35px;
    left: 35px;
}
.orange {
    background-color: #ffd700;
    top: 30px;
    left: 30px;
    z-index: 3;
}
.blue {
    background-color: #1e90ff;
    top: 95px;
    left: 95px;
    z-index: 2;
}
.cyan {
    background-color: #00ffff;
    z-index: 1;
}
```

Output:



In this case, the z-index property controls the stacking order:

- The orange div (z-index: 3) appears on top
- The blue div (z-index: 2) is next
- The cyan div (z-index: 1) is at the bottom
- The pink div has no specified z-index, so it's treated as if it had z-index: auto, placing it below elements with positive z-index

Transforms

- The transform property allows you to manipulate an element visually.
- It has the following methods: rotate(),translate(), scale(), move(), skew() and matrix().

rotate():

- By using the rotate() method, you can rotate an element clockwise or anticlockwise from its current position.
- A positive value will rotate it in a clockwise direction while a negative value will rotate it in the anti-clockwise direction.

e.g. if you want to rotate an element in a clockwise direction with 90 degrees:

```
div {
  transform: rotate(90deg);
}
```

Note: you will be learning transform property in detail in the upcoming lecture.

More CSS Concepts

OVERFLOW-WRAP

If there is an overflow of text in an element, it can be fixed using the **overflow-wrap** property which allows the element to break the ***breakable word*** in order to wrap it in the next line.

Syntax:

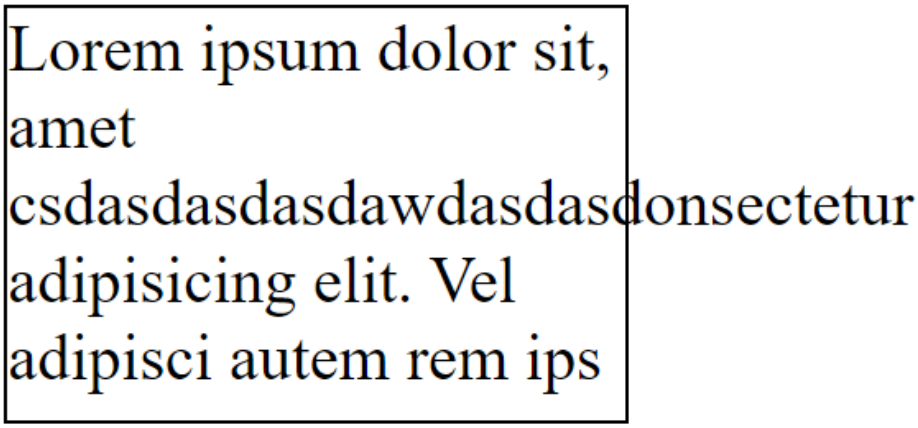
```
overflow-wrap: break-word;
overflow-wrap: normal;
overflow-wrap: inherit;
overflow-wrap: initial;
```

- **break-word:** It allows words to be broken
- **normal**: The browser will break words according to the normal rules. It's the default value.
- **initial**: This will set this property to its default value
- **inherit**: The element will inherit this property from its parent element.

For eg.,

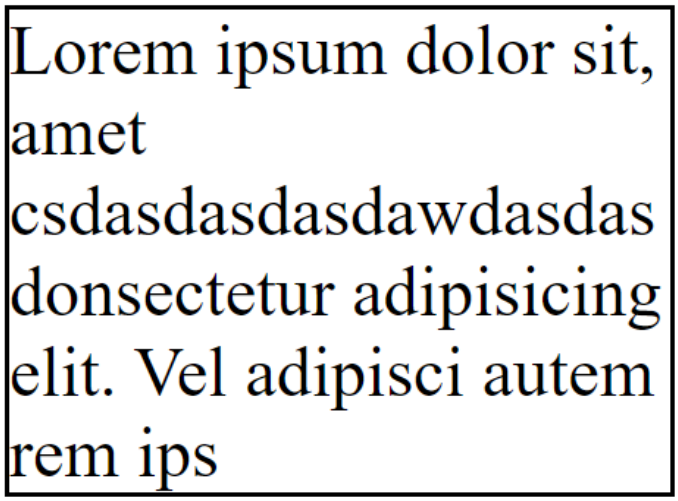
```
<head>

<style>
    p {
        width: 150px;
        height: 100px;
        border: 1px solid black;
      }
</style>

</head>

<body>

<p>
    Lorem ipsum dolor sit, amet csdasdasdasdawdasdasdonsectetur adipisicing elit.
Vel adipisci autem rem ips
</p>

</body>
```

Output:



As you can see that the text is ***overflowing out of the container*** and it's not looking good.
This can be fixed by using **overflow-wrap:**

```
width: 150px;
height: 100px;
border: 1px solid black;
overflow-wrap: break-word;
```



CSS GRADIENTS

With **CSS gradients**, you can create ***beautiful transitions between two colors or more***.
Generally, gradients are used as ***backgrounds for your HTML page***.
Gradients are of two types:

- **Linear Gradients:** Linear gradients are color bands that progress in straight lines. ie., up, down, left, right, and diagonally
- **Radial Gradients:** They radiate from a central point; they are defined by their center.

Linear Gradients:

This is the most basic type of gradient as here, you just **need to mention two colors.**

For eg.,

```
  div {
        width: 50px;
        height: 50px;
        background: linear-gradient(black, red);
    }
```

Output:

You can also create horizontal ones:

```
div {
    width: 50px;
    height: 50px;
    background: linear-gradient(to right, black, red);
}
```

You can also create **diagonal-running gradients** by ***specifying both horizontal and vertical starting positions:***

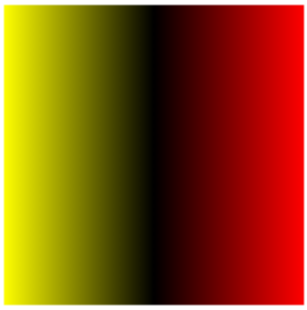
```
div {
    width: 50px;
    height: 50px;
    background: linear-gradient(to top right, black, red);
}
```

To have a **greater amount of control** on the direction, you can specify ***the angle in degrees,*** where a **positive angle means clockwise direction** and a **negative value means anticlockwise direction*:***

```
div {
    width: 50px;
    height: 50px;
    background: linear-gradient( 60deg, black, red);
}
```

You can also specify more than two colors:

```
div {
    width: 50px;
    height: 50px;
    background: linear-gradient( to right, yellow, black, red);}
```



You can use as many as colors you want. As you can see in the above example, ***by default, all the colors are equally spaced along the length of the gradient.***

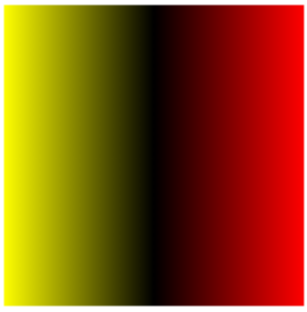
If needed, you can **change the location of each color** and **hard code the values to achieve the desired effect**.

If you give the values in percentages then ***0% means the starting point*** and ***100% means the ending point.*** You can also use values outside this range and if you don't specify the position for a color, it'll be automatically assigned to it.

For eg.,

```
div {
    width: 50px;
    height: 50px;
    background: linear-gradient( yellow 0%, black 50%, red 100%);
}
```

Will produce this same effect:



There are ***repeating-linear-gradients*** as well, which will take the same values as linear gradients, but they'll **repeat the gradient effect repeatedly, so as to cover the entire space of the container.**

For eg.,

```
div {
    width: 50px;
    height: 50px;
    background: repeating-linear-gradient( yellow, black 5%, red 20%);
}
```

Output:

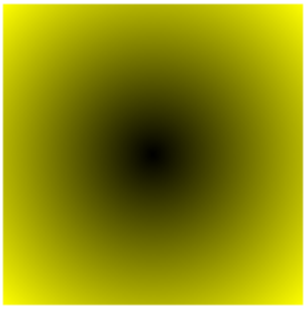
Radial Gradients:

Similar to the linear gradients, you can also create radial ones. The only difference is that they start to radiate from the element's center, ending towards the edges.
All the rules of the linear gradient are also applicable to the radial one.

For eg.,

```
div {
     width: 50px;
     height: 50px;
     background: radial-gradient(red, black, yellow);
}
```

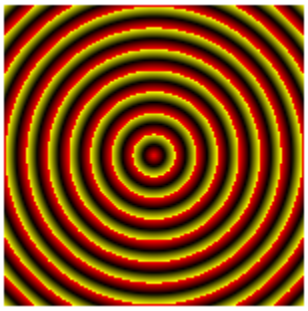
Output:



Similarly, you can create repeating-radial-gradients:

```
div {
    width: 50px;
    height: 50px;
    background: repeating-radial-gradient(red 0%, black 5%, yellow 10%);
}
```

Output :



Extra:

Read this to get more information on CSS gradients:

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Images/Using_CSS_gradients

LINK RELATED PSEUDO CLASSES

You probably know about ***:hover*** and ***:nth-child*** pseudo-classes. We'll discuss some more pseudo-classes related to hyperlinks!

To style and design what happens with hyperlinks as they are clicked, CSS has these pseudo-classes depending on the current state of a link:

- **:link -** This denotes an unvisited link.
- **:active -** This state comes just after a link is clicked.
- **:visited -** This denotes links that have already been visited.
- **:target -** It's used to style a link's target element.

- **:link**

  This pseudo-class is used for unvisited links. By default, the color of unvisited links is blue.

  For eg;

```
<p>
    take me to <a href="google.com">this</a> page.
</p>
```

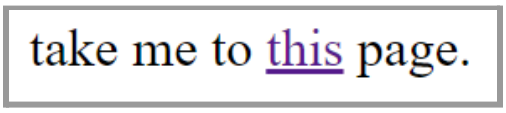
will show up as:

You can change this as per your style.

```
a:link {
    color: chartreuse;
}
```

- **:visited**

For links already visited, the default color is purple.
If we visit the link in the above example then the link will look like this:



We can use the **:visted** pseudo-class to change this:

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

Output:

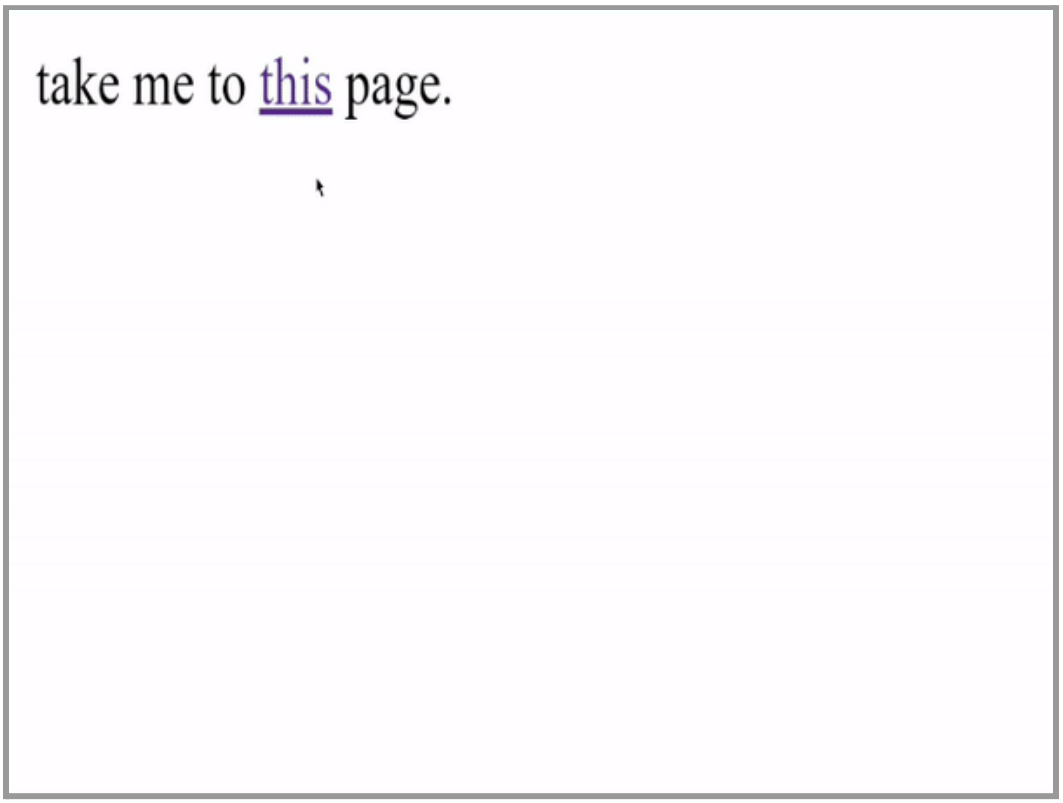
- **:active**

This state occurs when you click on a link. It is the very brief moment between someone clicking a link and the page being redirected to another

page. You can see this state easily if you click on a link and don't release the mouse button.

For eg., this code will set the color **red** for the ***active link***:

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



As you can see, after clicking and holding the mouse button for a couple of seconds, the **link becomes active and its color changes to red**. As soon as the ***mouse button is released, the page is redirected to the specified website***.

- **:target**

The URLs can have URLs as a **#** followed by **an element's id,** which makes that element **a target element.**
You can use this to style a ***link's current active target element.***

Look at the HTML code below:

```
<a href="#p1">link1</a>
<a href="#p2">link2</a>
<p id="p1">I'm first</p>
<p id="p2">I'm second</p>
```

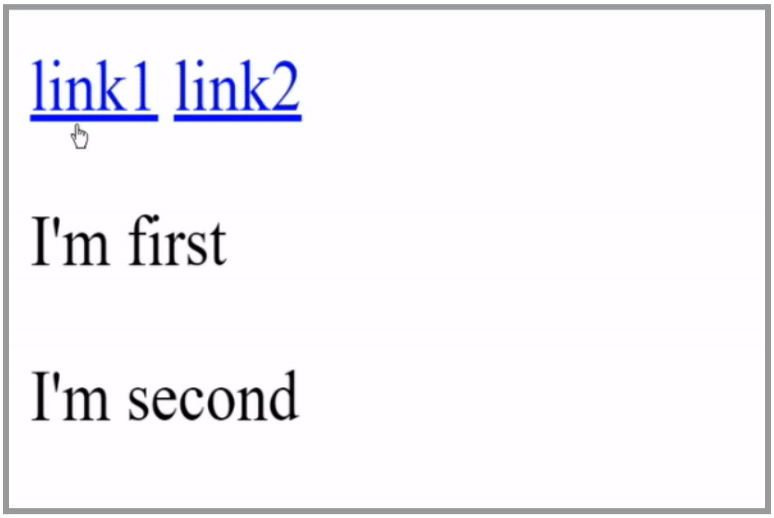
Here, we have two anchor tags followed by two <p> tags, and both the anchor tags ***contain the ids of both the <p> tags respectively, which makes both <p> tags targets for the respective links.***

Now, we can style these target elements, **when they become active.**

We can add styles like this, to style the target element, when the link is clicked:

```
#p1:target {
    background-color: darkslateblue;
}
#p2:target {
    background-color: deeppink;
}
```

Output:



As you can, as soon as the link becomes active, the styles are applied to the respective target elements.

PSEUDO ELEMENTS

If you want to style **specific parts of an element** like a ***line, or the first letter,or insert stuff before and after elements***, you can use something called **pseudo-elements.**

Syntax:

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

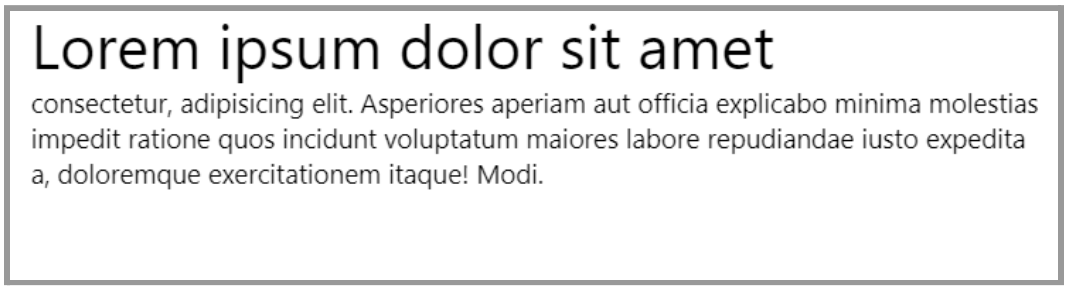
::first-line

This pseudo-element can be used to style the first line of an element.

For eg.,

```
<html>
<head>
   <style>
       p::first-line {
       font-size: 30px;
    }
   </style>
</head>
<body>
<p>
    Lorem ipsum dolor sit amet consectetur, adipisicing elit. Asperiores aperiam aut officia explicabo minima molestias impedit ratione quos incidunt voluptatum maiores labore repudiandae iusto expedita a, doloremque exercitationem itaque! Modi.
</p>
</body>
</html>
```

Output:



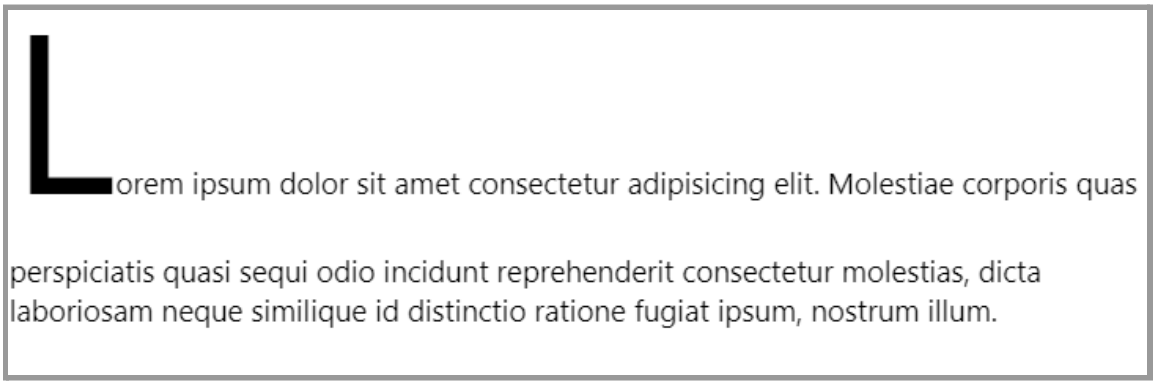
::first-letter

This pseudo-element can be used to add styling to the first letter of the text.

For eg.,

```
<html>
<head>
    <style>
        p::first-letter {
            font-size: 100px;
        }
    </style>
</head>
<body>
    <p>
Lorem ipsum dolor sit amet consectetur adipisicing elit. Molestiae
corporis quas perspiciatis quasi sequi odio incidunt reprehenderit
consectetur molestias, dicta laboriosam neque similique id distinctio
ratione fugiat ipsum, nostrum illum.
    </p>
</body>
</html>
```

Output:



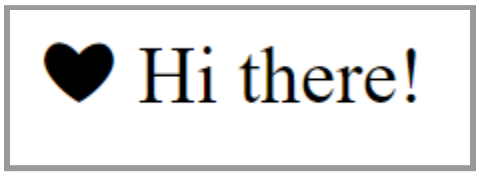
::before

You can use **::before** pseudo-element add content before an element's content. The content you want to add before the element will be **specified as a value to the content property.**

For eg.,

```
<html>
<head>
    <style>
        p::before {
            content: "❤";
            ;
        }
    </style>
</head>
<body>
    <p>
        Hi there!
    </p>
</body>
</html>
```

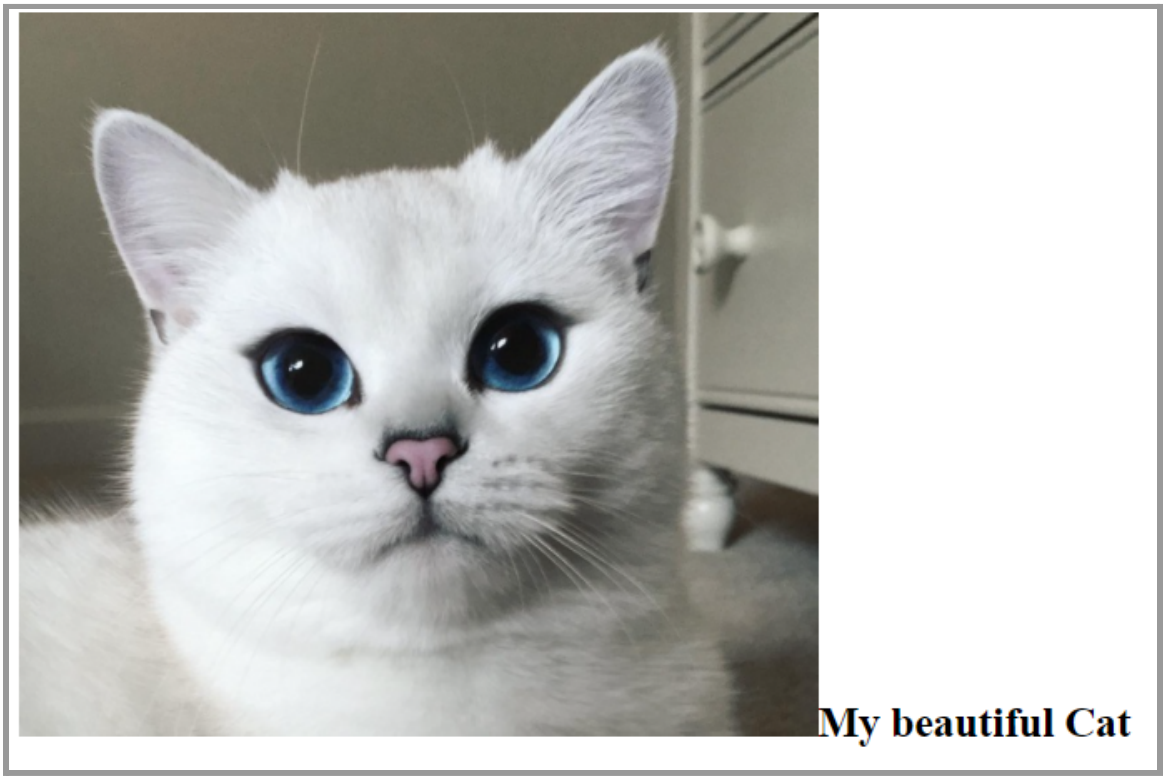
Output:



You can also add images like this:

```
<html>
<head>
    <style>
        h1::before {
            content:
url(https://dc-cdn.s3-ap-southeast-1.amazonaws.com/dc-Cover-67f1l9a958qafg
rmudoqel15k0-20160224121040.original.jpeg);
        }
    </style>
</head>
<body>
    <h1>My beautiful Cat</h1>
</body>
</html>
```

Output:



::after

Contrary to the ::before pseudo-element, the ::after pseudo-element is used to add content after an element's content.

For eg.,

```
<html>
<head>
    <style>
       h1::after {
       content:   url(https://i.ytimg.com/vi/3uazpaoz7tk/hqdefault.jpg);
      }
    </style>
</head>
<body>
    <h1>
        ♥♥ My cutipies ♥♥
    </h1>
</body>
</html>
```

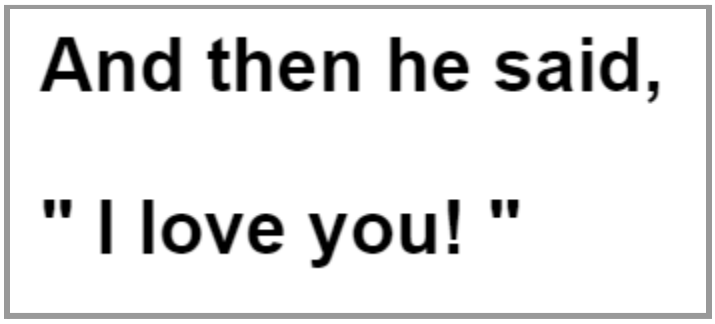
Output:

You can use the **::before** and **::after** pseudo-elements ***together to add content both before and after the element's content.***

For example, you can add quotation marks in a sentence using both **::before** and **::after** pseudo-elements:

```
<html>
<head>
    <style>
        p::before {
            content: ' " ';
        }

        p::after {
            content: ' " ';
        }
    </style>
</head>
<body>
    <h1>
        <span>And then he said,</span>
        <p>I love you!</p>
    </h1>
</body>
</html>
```

Output:



Extra:

To know more about pseudo-elements, follow this link:

https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-elements

Meta Element

Meta Element

- **Metadata** means data about data i.e. data that gives information about some other data.
- The **meta element** is used to specify metadata - some additional information about the HTML document.

The two main attributes of the meta element are:

- **name:**
  - It specifies ***what*** kind of metadata the element is representing.
  - This attribute has a specific meaning for the meta element.
  - Hence it can take only certain defined values that the meta element will understand.
  - Values it can take are: “keywords”, “description”, “viewport”, etc.

- **content:**
  - It specifies the ***actual value*** for the name attribute.
  - It can take up any value or a defined value depending on the name attribute.

For example,

```
<meta name=“keywords” content=“Frontend, HTML, Meta Tags”>
```

It means the meta element is representing keywords. Keywords are used to highlight the main crux of the HTML document. It helps search engines to index your webpage using the keywords specified using the ‘content’ attribute. It means your webpage is about frontend, HTML, and meta tags.

Viewport Meta Element

Setting the viewport meta tag, lets us control the width and scaling of the pages so that they are sized according to the device size.

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

It means,

Attribute	Value	Meaning
name	viewport	It means viewport meta tag is being used
content	width=device-width	**width** is used to control the viewport width. It means the width of the viewport will be equal to the actual width of the device on which you access the webpage. It can also be set to a specific number of pixels like width=300.
content	initial-scale=1.0	**initial-scale** is used to define the zoom level of the webpage when it is first loaded. It means the zoom factor of the webpage is 1.0 when it is first loaded i.e. neither zoomed out nor zoomed in.

Types of CSS Layouts

Broadly, CSS layouts can be categorised into three types:

Fixed Layout	Fluid Layout	Elastic Layout
The dimensions of HTML content are specified using **absolute units** like px, in, cm, etc.	The dimensions of HTML content are specified using **relative units** like %, vh, vw, etc	The dimensions of HTML content are specified using a **relative typographic unit** used for font sizes i.e. **em.**
Contents of the website do not adjust according to the browser window i.e. their **dimensions are fixed** across all screen resolutions.	Contents of the website adjust according to the browser window i.e. their **dimensions adjust** according to different screen resolutions (different devices).	Contents of the website **adjust according to the font size.** It means the widths of elements are represented using *em,* which depends on the font size of its parent element.
Hence it's **not flexible** with respect to different devices.	Hence it's **flexible** with respect to different devices.	Hence it's **flexible with respect to font size, not screen resolution.**

Note:

*There is a **Hybrid CSS layout** as well, which can be a **mix of any of the layouts specified above.** Usually responsive websites that we see use Hybrid CSS layout. They use the most appropriate units for different HTML elements according to the requirement or use case.*

Responsive Designs

INTRODUCTION

Responsive Design is a graphic design approach that is used to create content that can adjust to different screen sizes so that it looks good on devices with different screen sizes.

You need responsive designs because you want your website to look good on different devices without having to code different websites for each device type. It offers an optimized browsing experience. Your website will look great and work well on a desktop (or laptop), a tablet, and a mobile phone's browser as it automatically scales its content and elements to match the screen size on which it is viewed.

You can achieve this by three methods:

1. Responsive Design – This means the browser will show the same HTML code, from the same URL, regardless of the device on which it's being viewed.
2. Dedicated Website for each device – You can make a dedicated separate website for mobile-only devices, which will serve the same content, on a different URL.
3. Dynamic Serving – With the use of Dynamic serving, your website will use different HTML and CSS codes for different users, depending on their device type, on the same URL.

Here we will see Responsive Design using HTML and CSS.

Measurement units for Responsive Design

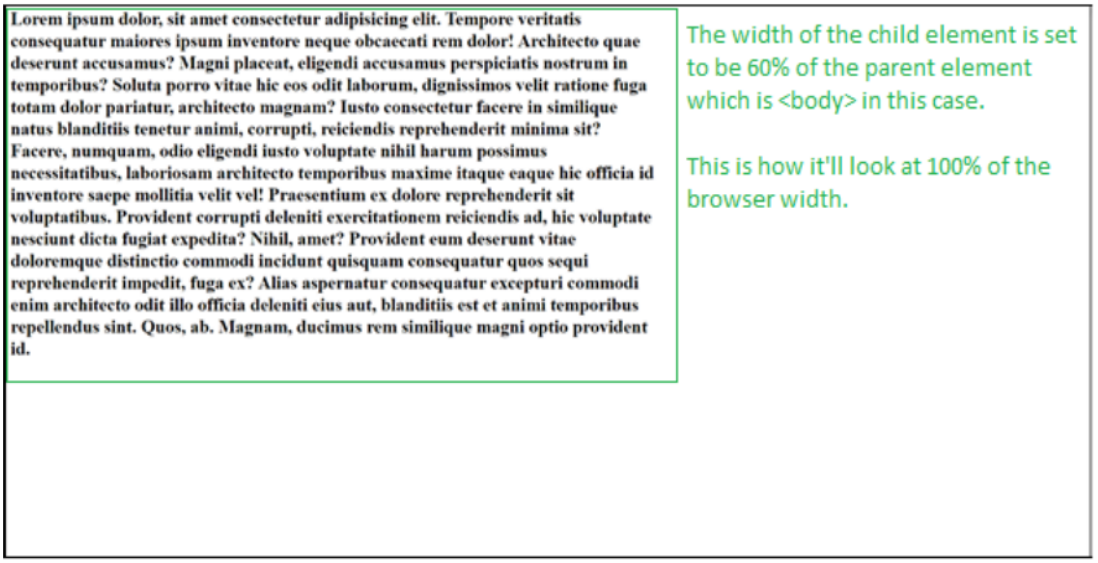
As we all know CSS units are of two types: Absolute and Relative

- Absolute Unit – It's a fixed-size unit across all devices and does not vary with different screen resolutions. For e.g., px(pixel), in(inch), mm(millimeter)
- Relative Unit – It doesn't have a fixed size. It's relative to something else like the font size of a parent element or the size of the viewport. E.g., % (relative

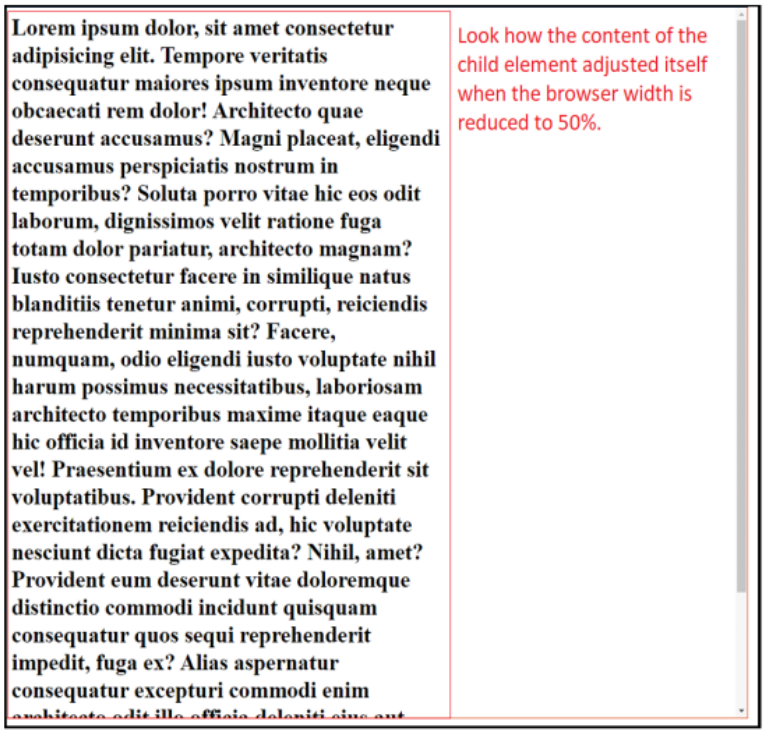
to parent element), rem (relative to the root element), em(relative to the parent element),

We should use relative units in fonts and images while keeping responsiveness in mind. To make things a bit clearer, we have explained the use of the percentage unit below.

The percentage unit is useful when creating a responsive fluid layout. We can adjust the percentage of width an element occupies which will be set according to the width of the parent element. Let's say we have a <p> tag inside the <body>, and we adjust the width of the child element to be 60%, then if we reduce the browser width, it should adjust itself and scale accordingly so that we'll be able to see the entire content of the element.



If we reduce the browser width, the child element will adjust and down-scale itself accordingly.



Viewport meta element

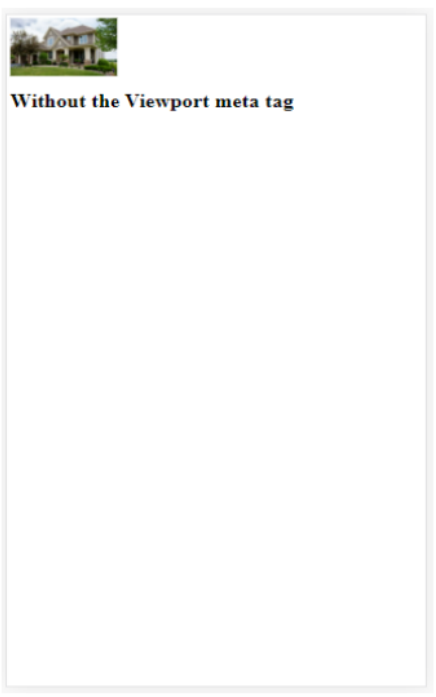
First of all, you'll need to add a viewport meta tag, without which, the mobile devices first render the pages at a typical desktop resolution and then scale them down, which makes it difficult to read as all the text and images will shrink and we'll have to zoom in to see the content.

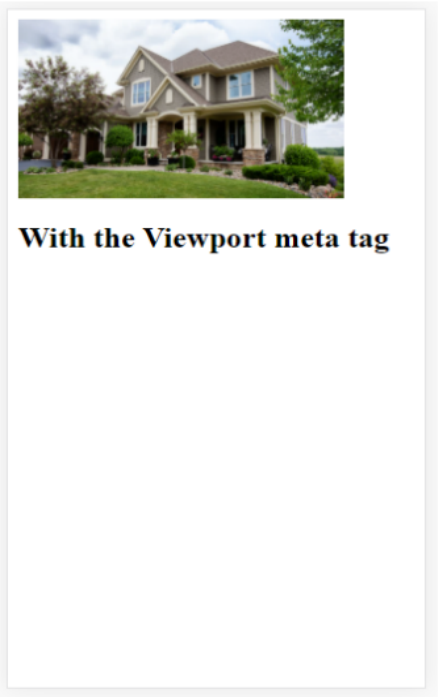
Setting the viewport meta tag, lets us control the width and scaling of the pages so that they are sized according to the device size.

You should include the following viewport meta element in all your web pages

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

Here is an example of a web page without the viewport meta tag, and the same web page with the viewport meta tag, when viewed on an iPhone SE:



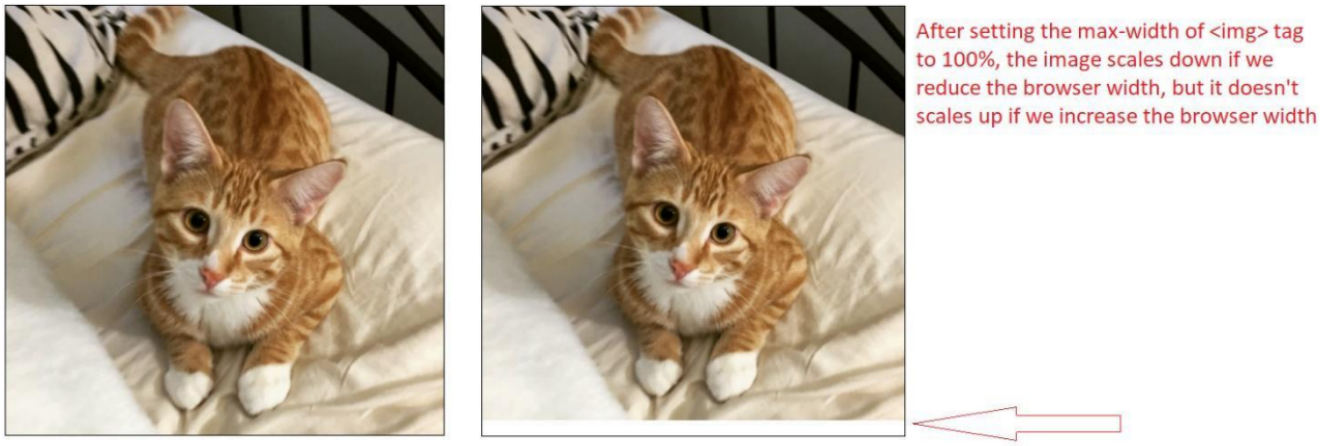


Responsive Images

Images can be responsive if they scale and adapt nicely to fit any browser window. We can set the width and height in percentages after which, the image will be responsive and it'll scale up and down according to the size of the browser.

The max-width and min-width properties can also be used, which will make sure the image never scales more than and less than its original size respectively. By setting the max-width property to 100%, an image would scale down smaller if its container becomes narrower than the image's intrinsic size, but never grows larger. This enables an image to scale down to fit in a flexibly-sized container, rather than overflow it, and become pixelated if the column becomes wider than the image.

```
<div>
    <img src="image.jpg" alt="" style="max-width: 100%;">
</div>
```



Media Queries

Media Queries were introduced in CSS3 and they are very useful for creating responsive websites, as they give the designer a lot of freedom and flexibility.

Media queries are useful when you want to modify your website or web app depending on the device type and characteristics.
It involves the use of the @media rule to include a block of CSS properties only if a certain condition is true.

A media query consists of two parts: media types and media features.

Media types: They describe the category of devices. Except for using logical operators, the media type is optional, and all types will be set as default.

- all: Default. Used for all media type devices
- print: Used for printers
- screen: Used for laptops, computer screens, tablets, smartphones, etc.
- speech: Used for screen-readers that “reads” the page content out loud.

Media features: Media features are used in media queries, which allow us to apply different styles depending on the capabilities of the output device. They must be surrounded by parentheses.
E.g., height - Height of the viewport.

Syntax:

```
@media not|only mediatype and (mediafeature and|or|not mediafeature) {
  CSS-Code;
}
```

The not, only, and are logical keywords.

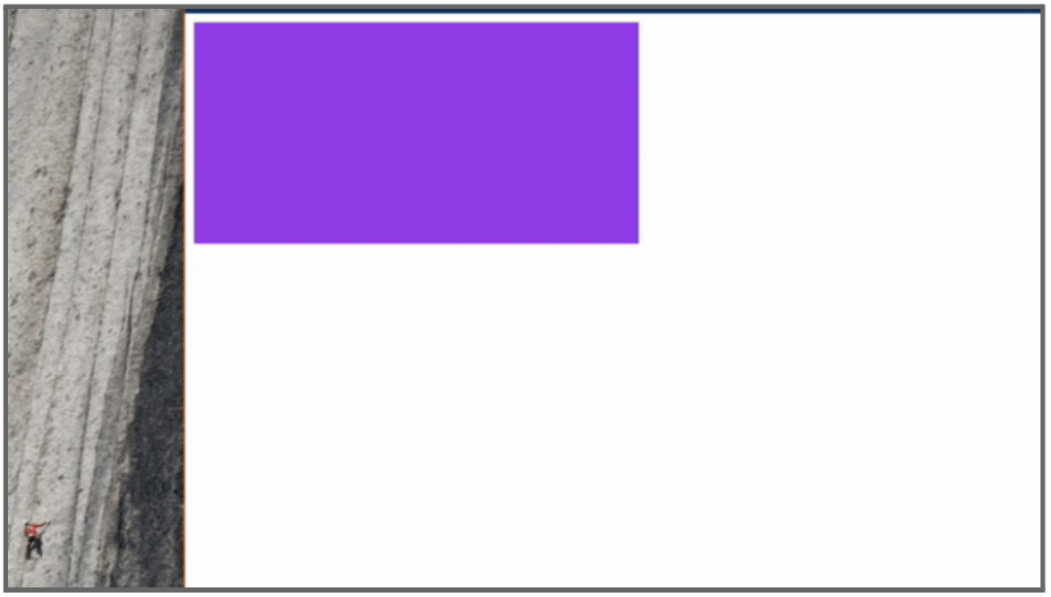
- not: It inverts the meaning of an entire media query.
- only: It will keep older browsers from applying the specified styles if they don't support media queries.
- and: it is used to specify a media feature and media type together

These all are optional and if you use not and only, you'll need to specify a media type.

Examples:

Change an element's color when the browser's width is 500px wide or less:

```
@media screen and (max-width: 500px) {
  #element {
    background-color: royalblue;
  }
}
```



2. Use media queries to set the background-color of an element to **pink** if the viewport is 700 pixels wide or wider, to **darkgreen** if the viewport is between 300 and 699 pixels wide. If the viewport is smaller than 300 pixels, the background-color is **grey:**

```
#element {
    background-color: grey;
    width: 50px;
    height: 50px;
}

@media screen and (min-width: 300px) {
   #element {
       background-color: darkgreen;
   }
}

@media screen and (min-width: 700px) {
   #element {
      background-color: pink;
   }
}
```

Hide an element when the browser's width is 800px wide or less and show it when the browser’s width is 1600px or more:

```
@media screen and (max-width: 800px) {
      #element {
        display: none;
      }
}
@media screen and (min-width: 1600px) {
      #element {
       display: block;
      }
}
```

Responsive Typography

Typography is the technique of arranging text to make written language legible, readable and appealing when displayed. There are different ways to achieve this. Generally, you can use pixels when the website you're working on, has a fixed width.

All the text and font in a responsive website are also responsive. ie It should change according to the size of the device browser. The font of your website should have a size that is related to its parent's container width so that it can adapt to the screen of the client and be easily readable on mobile devices.

CSS3 includes different relative units which can help us achieve relative typography.

- You can use 'vw' to set the text size. It means the 'viewport width' and its 1 unit's value is 1% of the total viewport width. As you'll use this, the size of the text gets itself adjusted according to the size of the browser window as the viewport is the browser window size.

```
<h2 style="font-size: 5vw;">
  Hello World!
</h2>
```

- We can also change font size with media queries. We can use rem. They are relative to the HTML element, which makes them a lot easier to use. We can define responsive font sizes as shown below:

```
@media (min-width: 640px) { body {font-size:1rem;} }
@media (min-width:960px) { body {font-size:1.2rem;} }
@media (min-width:1100px) { body {font-size:1.5rem;} }
```

Mobile-first Vs Desktop first

Desktop-first Responsive Strategy - This strategy involves designing a display that can show the maximum amount of information so that you can communicate with your audience as much as you want, provided that the target audience uses big screen devices like laptops and desktop PCs.

Mobile-first Responsive Strategy - As smaller screens have less space, you'll need to hide and adjust some elements while working with them. Generally, there are no changes in the functionality of the website. We'll have to focus on the user experience along with the proper implementation of the website on mobile devices. The mobile-first strategy needs a lot of research as compared to the Desktop-first strategy so that we can organize the content according to priority and space.

Most web content today is consumed with mobile devices so it just makes sense to develop mobile-first. Whether your app is mobile accessible or not, it should depend on your target audience.

Responsive design is the standard practice. Once upon a time, people called it mobile-first. It's easy to make your website responsive nowadays with CSS frameworks like bootstrap. The advantage of responsive design is that you don't have to try to add messy code to retrofit your website if you decide to make it mobile later on which makes it simpler to grow and maintain. Therefore, it all depends on the priority you'll give to the target audience. You can also use certain frameworks that can help you make responsive websites with much ease.

Responsive Frameworks

There are many free front-end frameworks that offer responsive design like Bootstrap and Foundation.

You can read about them in detail from the links given below:

1. https://getbootstrap.com/
2. https://get.foundation/

Animate.css

The links mentioned for the Animate.css library in the video have changed. Here are the new links

- Animate.css GitHub repo - https://github.com/animate-css/animate.css
- Animate.css website - https://animate.style/
- Animate.css raw file from Github - https://github.com/animate-css/animate.css/blob/main/animate.css

Note:

You have to add "animate__" before every property for it to actually work!!

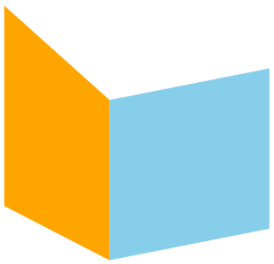
For example:

Solution - Building a Cube

Cube has 6 faces. **Make sure you have coded the first two sides of the cube!** You already built two faces as follows:

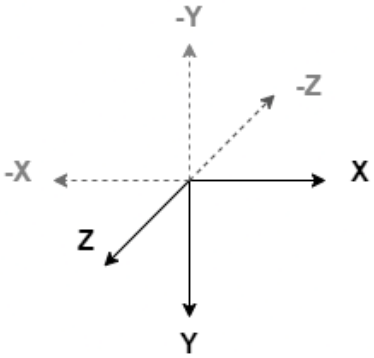
```
.one{
    background-color: skyblue;
}

.two{
    background-color: orange;
    transform: rotateY(90deg) translateZ(-100px)
translateX(-100px);
}
```

Currently, the cube has two sides - the back side and the left side. It looks like



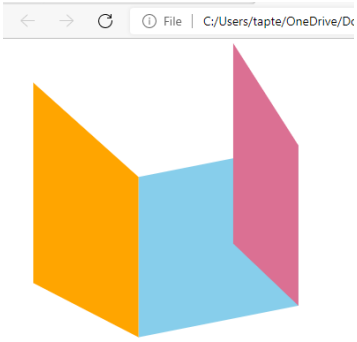
Now you'll be building the rest of the faces. You can generate these faces in any order.

Note: The cartesian coordinates that CSS uses has their Y-axis inverted as shown:



Face 3 (Right Side)

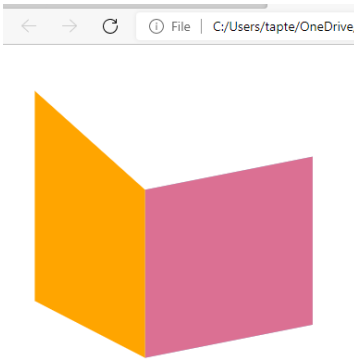
We aim at making this

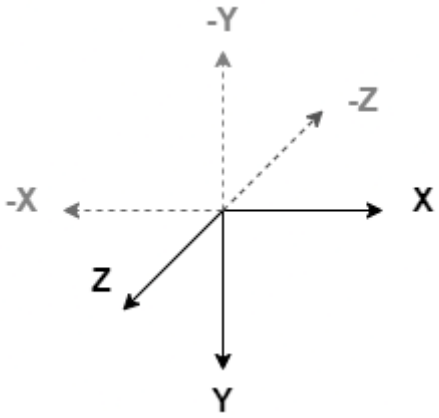


Step 1: Give background color to the face 3.

```
.three{
    background-color: palevioletred;
}
```

It'll generate:
(The pink side covers the blue side in the XY plane, hence the blue side is not visible yet.)

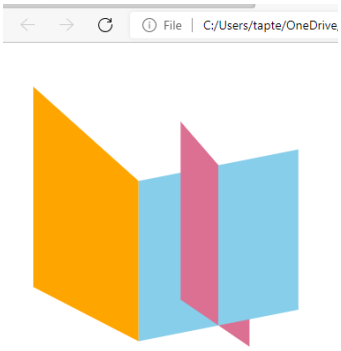


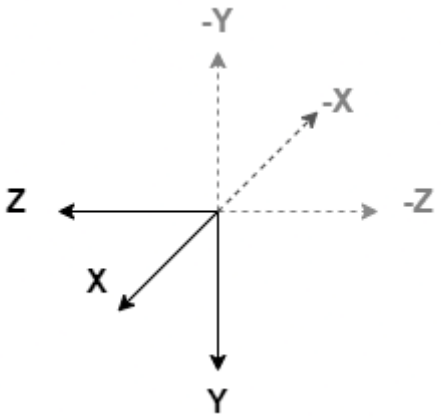


Step 2: Rotate the pink side along the Y direction.

```
transform: rotateY(-90deg) ;
```

The cube will look as shown.
The axis of the pink face will rotate as shown.



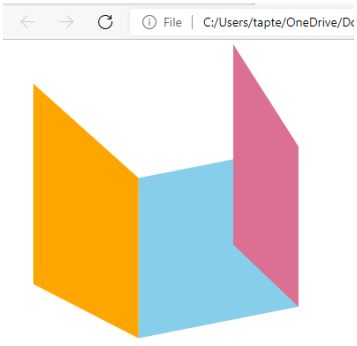


Step 3: Now translate the pink face 100px towards the -ve Z direction, and 100px towards the +ve X direction.

The final code will be

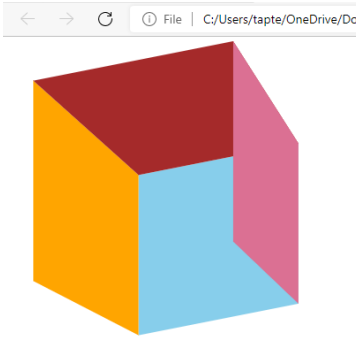
```
.three{
  background-color: palevioletred;
  transform: rotateY(-90deg) translateZ(-100px) translateX(100px);
}
```

Final cube after adding face 3



Face 4 (Top Side)

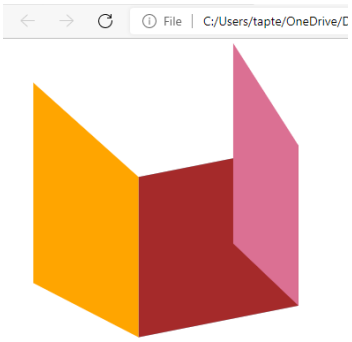
We aim at making this

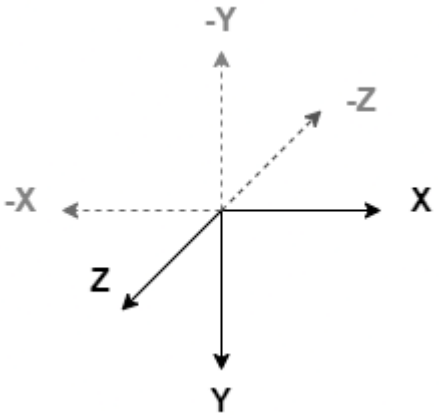


Step 1: Give background color to the face 4

```
.four{
    background-color: brown;
}
```

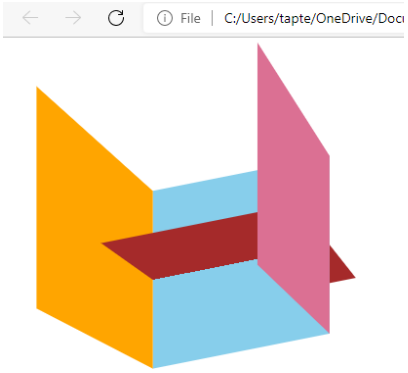
It'll generate:
(The brown side covers the blue side in the XY plane, hence the blue side is not visible yet.)

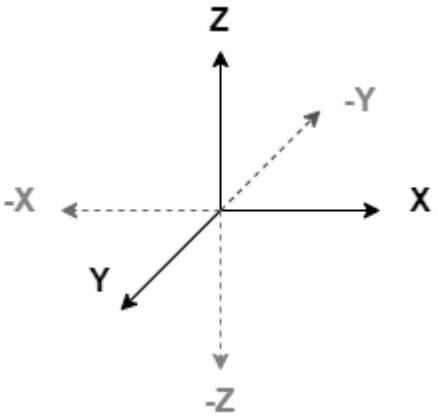




Step 2: Rotate the brown side along the X-direction.

```
transform: rotateX(90deg) ;
```



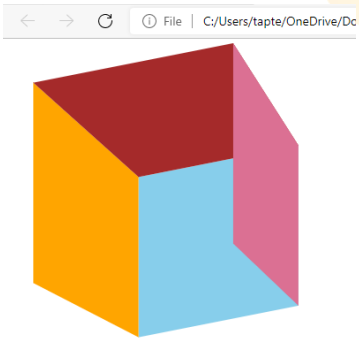


Step 3: Now translate the brown side 100px both in Y and Z directions

Final code:

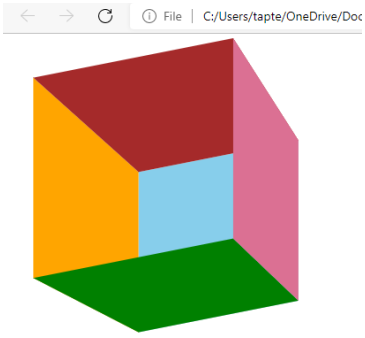
```
.four{
   background-color: brown;
   transform: rotateX(90deg) translateY(100px) translateZ(100px);
}
```

Final cube after adding Face 4



Face 5 (Bottom Side)

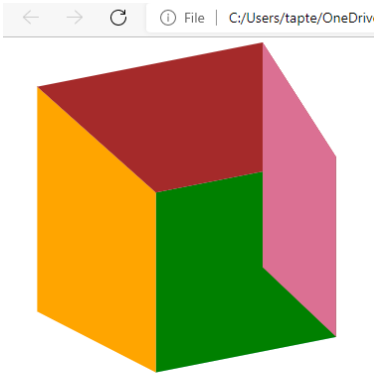
We aim at making this

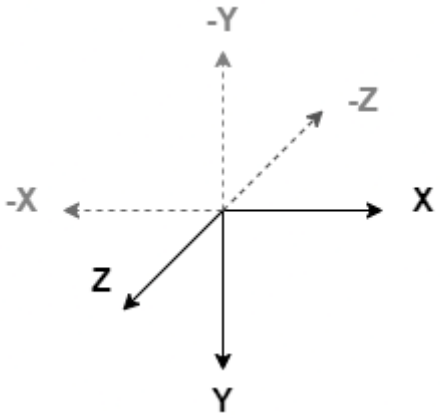


Step 1: Give background color to the face 5

```
.five{
    background-color: green;
}
```

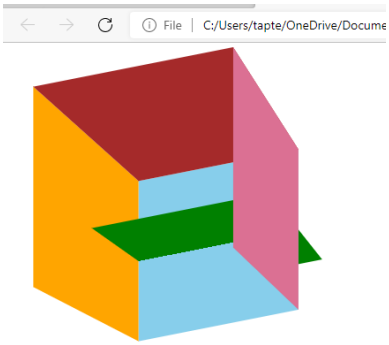
It'll generate:
(The green side covers the blue side in the XY plane, hence the blue side is not visible yet.)

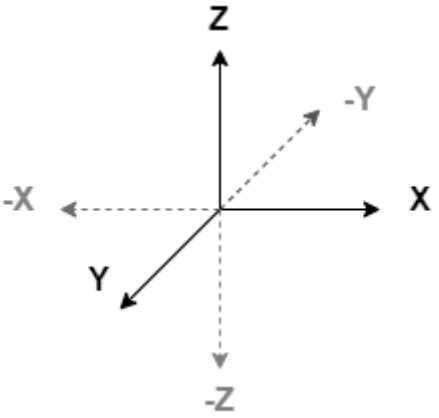




Step 2: Rotate it along the X-axis

```
transform: rotateX(90deg) ;
```



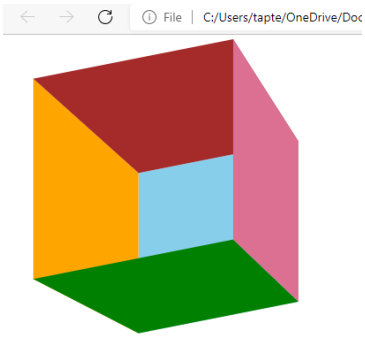


Step 3: Translate it -100px in Z-direction and 100px in Y-direction

Final code:

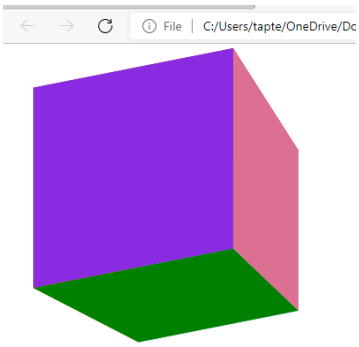
```
.five{
   background-color: green;
   transform: rotateX(90deg) translateZ(-100px) translateY(100px);
}
```

Final cube after adding face 5



Face 6 (Front Side)

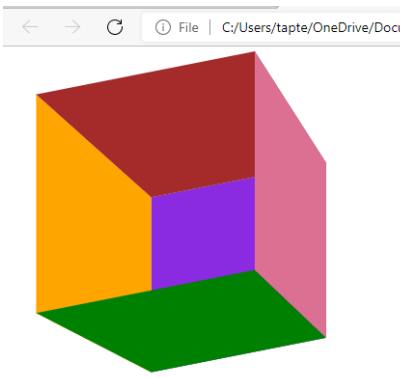
We aim at making this

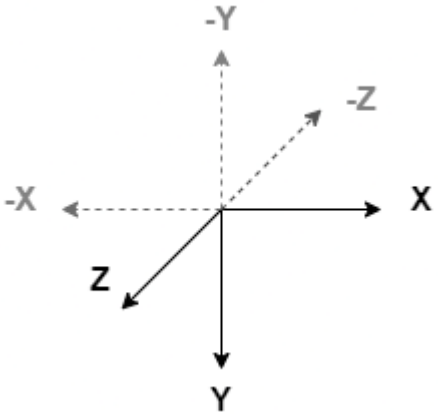


Step 1: Give background color to the face 6

```
.six{
    background-color: blueviolet;
}
```

It'll generate:
(The blueviolet side covers the blue side in the XY plane, hence the blue side is not visible yet.)



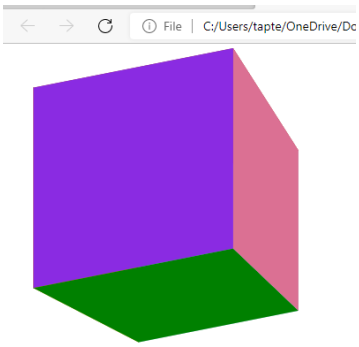


Step 2: Translate it along the Z-axis

Final Code:

```css
.six{
    background-color: blueviolet;
    transform: translateZ(200px);
}
```

Our 3D Cube is ready!!



NOTE:

We can rotate a cube using CSS. Although we should avoid adding such functionalities using CSS. The rotation can be achieved using javascript which you'll learn later.

Animations and 3D Space

Transitions and Transforms

Transforms:
You can visually manipulate an element using the CSS **transform** property.
It has the following methods:

- translate()
- rotate()
- scale()
- skew()
- matrix()

Translate(): Using this method you can move an element sideways or up and down from its current position. It requires two parameters for the X-axis and the Y-axis.

For example, if you want to move an element 100px from the top and 50px from the left:

```
div{
   transform: translate(50px, 100px);
}
```

If you want to move only in one direction, you can do it using the **translateX()** and **translateY()** methods.

Rotate(): Using this method, you can rotate an element anticlockwise or clockwise from its current position. A positive value will rotate it in a clockwise direction whilst a negative value will rotate it in the anti-clockwise direction. If you want to rotate an element along a specific axis, you can use the **rotateX(), rotateY()** and **rotateZ()** methods which rotate along the X, Y, and Z-axis respectively.

For example if you want an element to rotate an element in clockwise direction with 100 degrees:

```
div {
     transform: rotate(100deg);
}
```

Scale(): You can use this method to increase or decrease the size of an element. It requires two parameters that represent the amount of scaling to be applied in the X-axis and the Y-axis respectively. This method only scales in 2D. The **scale3d()** can be used to scale in 3D.

For example, if you want to scale an element by increasing its width 3 times its original width and increasing its height 5 times its original height:

```
div {
    transform: scale(3,5);
}
```

If you want to increase only the width or only the height of an element, you can use the **scaleX()** and the **scaleY()** methods respectively.

Skew(): If you want to skew an element along the X-axis and the Y-axis, you can use the skew() method. If you want to skew an element only along the X-axis or the Y-axis, use the **skewX()** and **skewY()** methods respectively.

The following example skews an element 20 degrees along the X-axis and 90 degrees along the Y-axis:

```
div {
    transform: skew(20deg, 100deg);
}
```

matrix(): This method combines all the above 2D transform methods and accepts six parameters, which allows you to scale, skew, translate, and rotate.

The parameters are given in this way:

matrix(scaleX(),skewY(),skewX(),scaleY(),translateX(),translateY())

```
div {
   transform: matrix(2,0,1,2,10,0);
}
```

Transitions: If you want to change the property of an element over a given period of time, you can use transitions. These are the following transition properties:

- transition
- transition-delay
- transition-duration
- transition-property
- transition-timing-function

To create a transition for an element, you'd need to specify two things:

- The CSS property you want to add a transition to
- The duration of the transition

If you don't specify the duration of the transition, then the transition will have no effect as the default value for the duration is 0.

Look at the example below, where after hovering on the element, its size will scale by a factor of 2 and all this will happen within 2s.

```
div:hover {
   transform: scale(2);
}

div {
   background-color: aquamarine;
   width: 50px;
   transition: 2s ease;
   height: 50px;
}
```


The dimensions of the element begin to increase as soon as we hover over it and it looks like a cool animation. You can create similar effects using the transition property.

Similarly, you can also create a transition effect for multiple properties. You'll have to mention the property names and the duration of each one separately :

```
div {
    transition: width 2s, height 2s;
}
```

You can also adjust the speed curve of the transition with the help of these additional parameters:

- **ease**: the transition will have a slow start, then it'll go fast, and then it'll end slowly
- **linear**: the transition will have a constant linear speed throughout its time duration
- **ease-in**: the transition will have a slow start
- **ease-out**: the transition will have a slow end
- **ease-in-out**: the transition will have a slow start as well as a slow end

Example:

```
 div {
    transition: 2s ease-in;
 }
```

These properties can also be defined separately in the **transition-timing-function** which is used to specify the speed curve of the transition:

```
div {
    transition: 2s;
    transition-timing-function: ease-in;
}
```

The **transition-delay** property will add a delay in the transition, after which the transition will occur.

```
div {
    transition-delay: 2s;
}
```

You can specify the property for which transition is required separately using the **transition-property**:

```
div {
    transition-property: width;
}
```

Similarly, the duration of the transition can be specified separately using the **transition-duration** property:

```
div {
    transition-duration: 4s;
}
```

Creating a simple animation

Simple animations can be created easily using the transition and transform properties. Look at the simple animation below:

Just with basic knowledge of transitions and transforms, animations like this can be created.

```
div:hover {
    transform: translateX(200px) scale(2);
}

div {
    margin: 100px;
    width: 50px;
    background-color: bisque;
    height: 50px;
    transition: 1s ease-in;
}
```

Explanation: When the mouse hovers over the , the **translateX** causes it to move 200px to the right and the **scale** property increases the dimensions of it and all of this occurs with a **transition** of 1s which makes it look like an animation.

If you want the above animation to complete in a specified number of steps, you can use the **steps** property in transition where you need to specify the number of steps and start|end based on your preference.

You can also achieve an animation like this one:

```
div:hover {
   transform: rotate(360deg) translateX(20px) scale(0.75);
   border-radius: 50%;
}

div {
   margin: 100px;
   width: 50px;
   background-color: bisque;
   height: 50px;
   transition: 1s;
}
```

This one also works similarly; as soon as the mouse hovers over the , the **transform:rotate()** property rotates it by 360 degrees along which **translateX** moves it to the right side a little bit and scale reduces its size by a factor of 0.75 and a border-radius of 50% is being applied to it, all within a 1s time frame.

CSS Animation Property

You can also create some advanced animations in CSS using the **animation** property. Each property has a name, which will be used along with the **@keyframe** rule.

@keyframe will define what will happen at specific moments during the animation.

Animation property has some sub-properties which are explained below:

- **animation-name**: it's a name for the keyframe animation
- **animation-duration**: it's the duration of the animation in seconds or milliseconds
- **animation-delay**: it specifies a delay, after which animation will start loading
- **animation-iteration-count**: it's the number of times the animation will be executed
- **animation-timing-function**: it sets a speed curve for the animation;
    - **ease**: the animation will have a slow start, then it'll go fast, then it'll have a slow end.
    - **linear**: the speed of the animation will remain constant throughout its duration
    - **ease-in**: the animation will have a slow start
    - **ease-out**: the animation will have a slow end
    - **ease-in-out**: both the end and the start of the animation will be slow
    - **cubic-bezier(n,n,n,n)**: you can define your own bezier curve using this
- **animation-direction**: it specifies the direction of the animation after each cycle;
    - **normal**: the animation will be played in the normal (default) direction

- **reverse**: the animation will be played in the reverse direction
- **alternate**: the animation will be played first in the normal direction, then in the reverse direction
- **alternate-reverse**: the animation will be played in the reverse direction first, then in the normal direction

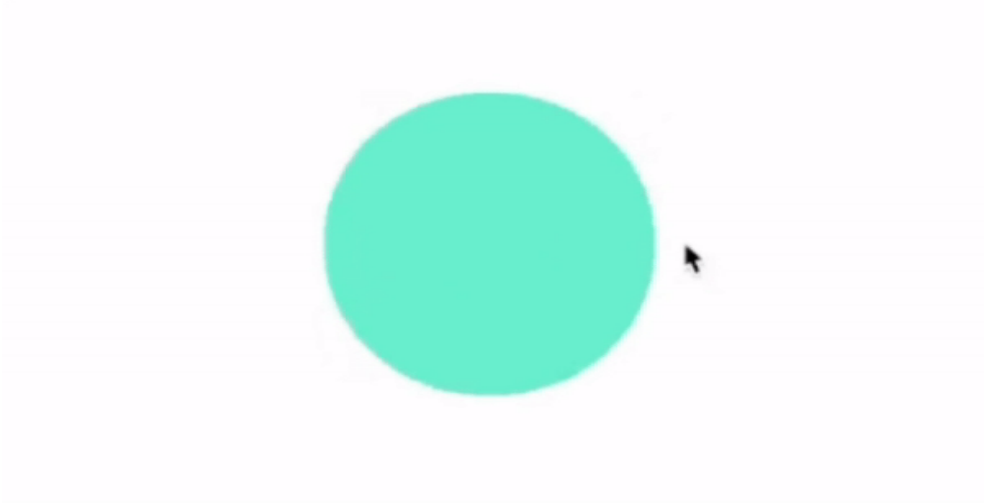
- **animation-fill-mode**: it sets how an animation will apply styles to its target, before and after its execution;
  - **none**: it's the default value, meaning that element will not retain any styles before or after the execution of the animation
  - **forwards:** the target element will retain the styles and values set by the last keyframe, which depends upon the values of animation-direction and animation-iteration-count.
  - **backwards:** the target element will retain the styles and values set by the first keyframe, which depends upon animation-direction
  - **both:** In this, the animation will follow both the above rules, so the target element will retain the styles and values in both directions.

@keyframe: It defines what will happen during different stages of animation. This gives you a lot more control over the sequence of the animation as you will define what happens at different sections of the animation. To use @keyframes, you'll need to create one along with a **name**, which is then used by the ***animation-name*** property to match its value with a keyframe declaration.

You'll need to specify the percentage of the time at which the keyframe should occur. The ***from*** and ***to*** keywords represent the initial (start) and the final (end) state of the animation respectively. You can use them instead of writing **0%** and **100%.**

For Example:

```
@keyframes changeShape {
    from {
        border-radius: 0%;
    }
    to {
        border-radius: 50%;
    }
}
div {
    height: 200px;
    width: 200px;
    background-color: aquamarine;
    animation-name: changeShape;
    animation-duration: 1s;
    animation-iteration-count: infinite;
}
```



Explanation: We created a keyframe named *changeShape* and we specified that when the animation starts, the border-radius is 0%, and when the animation ends, it's 50%. When the *animation-name* is matched with the one given to the keyframe, the animation starts to run. As the *animation-duration* is set to 1s, the full animation duration is only 1 second and then it starts to run again. The *animation-iteration-count* is set to **infinite**, which means after completing a cycle of 1s, this animation will continue to run forever.

If we set the *animation-direction* to **alternate**, then the keyframe animation will first run from 0% to 100% and then from 100% to 0%; the transition from rectangle to circle and circle to rectangle will happen in both directions which will look smoother as compared to earlier one.

Instead of **infinite**, if we put a number like 4 in the *animation-iteration-count,* the animation will run only four times in total. You can set the *animation-timing-function* as **ease-in-out,** which will cause the animation to run smoothly during the start and the end.

Instead of only specifying what to do at the start and end of the animation, you can also specify what to do at individual states of the animation. For example, if you want to change the background color of the when the animation reaches 50% of its duration:

```
@keyframes changeShape {
    from {
        border-radius: 0%;
    }
    50% {
        background-color: beige;
    }
    to {
        border-radius: 50%;
    }
}
```

Similarly, you can specify the actions you want to perform at any specific duration of time. You can also write **0%** and **100%** instead of *from* and *to* respectively.

Instead of specifying all the animation properties separately, you can use a shorthand property; *animation* where you can specify all other sub-properties at once. The above animation can also be written like this.

```
animation: changeShape 1s ease-in-out infinite;
```

CSS 3D Transforms

As you know by now, CSS was built to typically style documents, it's not ideal for 3D modelling. But as of 2009, some of the additional 3D features were added in CSS which can be used for some really good 3D animations. We'll explain these by taking examples. The ***perspective*** property is required to activate 3D space. We can apply this like:

```
transform: perspective(400px);
```

OR

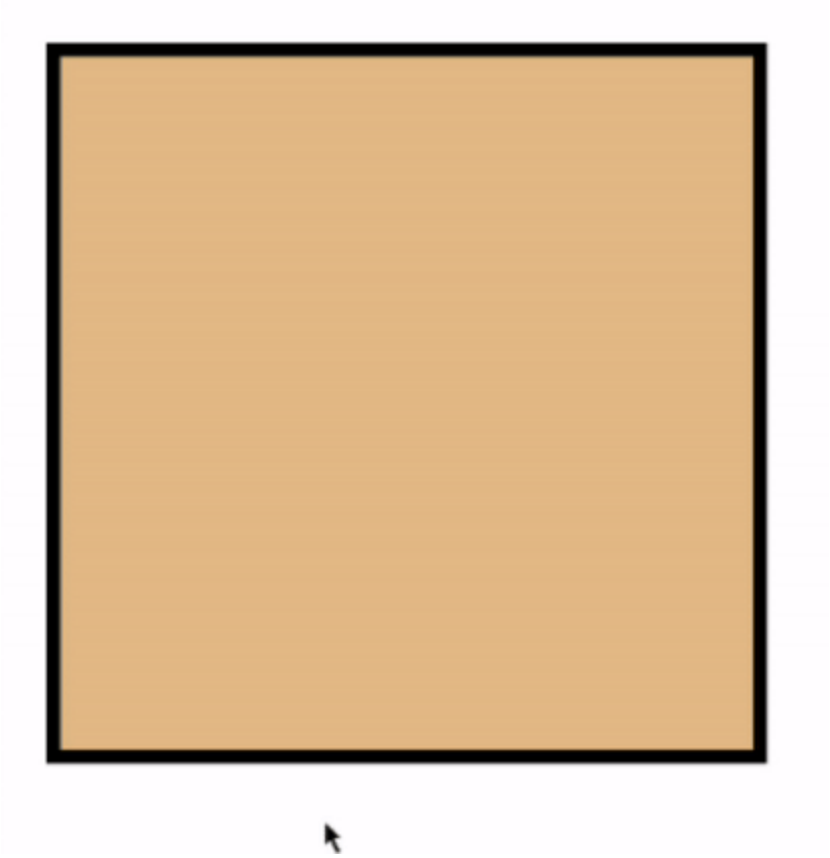
```
perspective: 400px;
```

The value of ***perspective*** property determines the ***intensity*** of the 3D effect. You can think of it as a distance from the viewer to the object. The greater the value, the further the distance, the less intense the visual effect. **perspective: 2000px** yields a ***subtle*** 3D effect, as if we are viewing an object from ***far away*** through binoculars whereas, **perspective: 100px** produces a ***tremendous*** 3D effect, like a tiny person viewing a massive object.

This will trigger a 3D space and thus, we can see the element moving and transforming in 3D space. For example:

```
<head>
    <style>
        #outer {
            width: 100px;
            height: 100px;
            border: 2px solid white;
            perspective: 400px;
        }

        #inner {
            background-color: burlywood;
            width: inherit;
            height: inherit;
        }

        #inner:hover {
            transform: rotateY(45deg);
        }
    </style>
</head>
<body>
    <div id="outer">
        <div id="inner">

        </div>
    </div>
</body>
```

We have two *** *** tags {inner and outer}, we applied ***perspective:400px*** on the ***outer .*** On hovering over the inner div, it will rotate along the Y-axis and you'll be able to see it:



The 3D transforms use the same `transform` property used for 2D transforms. As you already know about 2D transforms, this will easy for you:

- `rotateX( angle )`
- `rotateY( angle )`
- `rotateZ( angle )`
- `translateZ( tz )`
- `scaleZ( sz )`

Whereas `translateX()` moves an element along the horizontal X axis, `translateZ()` positions it along the Z axis. The rotate functions rotate along the corresponding axis.

In order for children to inherit a parent's perspective, and sustain in the same 3D space, the parent's perspective can passed along with `transform-style: preserve-3d`

We can create a card flipping animation like this:

This is the basic HTML for the card.

```
<body>
    <div class="scene">
        <div class="card">
            <div class="card_face card_face-front">front</div>
            <div class="card_face card_face-back">back</div>
        </div>
    </div>
</body>
```

Everything will happen inside the element with class ***.scene.*** The ***.card element*** will act as the 3D object. Two separate ***.card_face*** elements form the faces of the card.
First we will apply all the necessary ***perspective*** styling to the 3D space, which will contain all the stuff:

```
  .scene {
        width: 200px;
        height: 200px;
        border: 1px solid black;
        margin: 40px 0;
        perspective: 600px;
    }
```

Now, the ***.card*** can be transformed into its parent's (***.scene***) 3d space. We'll use ***position:relative*** so that the card faces are positioned correctly:

```
.card {
        width: 100%;
        height: 100%;
        transition: transform 0.5s;
        transform-style: preserve-3d;
        position: relative;
    }
```

To reset the position of the card faces in 2D space, we used ***position:absolute***:

```
.card_face {
        position: absolute;
        width: 100%;
        height: 100%;
        line-height: 200px;
        color: white;
        text-align: center;
        font-weight: bold;
        font-size: 40px;
}
```

Other styles are there to adjust the position and appearance of the text.

To flip the card on hover, we added a basic 3D transform which will rotate the cards along the Y-axis by 180 degrees:

```
 .card:hover {
        transform: rotateY(180deg);
    }
```

You can refer to the full code here:

```
<style>
    .scene {
        width: 200px;
        height: 200px;
        border: 1px solid black;
        margin: 40px 0;
        perspective: 600px;
    }
        .card {
        width: 100%;
        height: 100%;
        transition: transform 0.5s;
        transform-style: preserve-3d;
        cursor: pointer;
```

```
        position: relative;
    }
        .card:hover {
        transform: rotateY(180deg);
    }
        .card_face {
        position: absolute;
        width: 100%;
        height: 100%;
        line-height: 200px;
        color: white;
        text-align: center;
        font-weight: bold;
        font-size: 40px;
    }
        .card_face-front {
        background: maroon;
    }
        .card_face-back {
        background: royalblue;
        transform: rotateY(180deg);
    }
</style>
 <body>
    <div class="scene">
        <div class="card">
            <div class="card_face card_face-front">front</div>
            <div class="card_face card_face-back">back</div>
        </div>
    </div>
</body>
```

You can create many such animations using 3D transforms. Try creating a 3D Cube, which will test your skill a lot.

EXTRA:
You can refer to this link below to read more about 3D transforms:
https://blog.logrocket.com/the-noobs-guide-to-3d-transforms-with-css-7370aafd9edf/

Additional Notes

Bootstrap is a front-end framework.

Framework vs Library

Framework	Library
It provides a standard way and a collection of reusable code for software development.	It is a collection of reusable code used for software development.
It is the incharge of the flow of your application. It means you have to plug in your code inside the skeleton functionalities that it provides.	You are the incharge of the flow of your application. It means you can plug in the functionalities that it provides anywhere in your code. It gives you the freedom to choose when and where to use it.

Pros and Cons of Using Bootstrap

Pros

- It is a bunch of reusable code, hence it saves you time.
- It provides loads of features for styling purposes.
- It uses a mobile-first approach and provides inbuilt functionalities to make a website responsive.
- It has great support and documentation for exploring and learning.

Cons

- It is very bulky i.e. it has a lot of features which your website may or may not require.
- Since it is very popular, there are many websites that are built using it. So it may produce similar designs that lack creativity.
- It's less flexible as it may require a lot of customizations to adjust it according to the required use case.

Content Delivery Network (CDN)

The Idea Behind CDN

When we enter a URL in a browser, it sends a request to the corresponding server to fetch the website content. This website content can be HTML, CSS, and JS files. The time taken to fetch a website over the internet often depends on the distance between the client (in this case a browser) and the server. If a server is located in a far-off country from where you are, the loading time of the website will be high. Hence it'll create a bad user experience.

What is CDN?

CDNs solve the above problem. They are geographically distributed servers across the world. Although they do not host content but provide a cache for the content. Whenever you enter a URL in your browser, the request will be sent to the nearby CDN which will provide the content, hence reducing the loading time.

Bootstrap using CDNs

Bootstrap uses these Content Delivery Network services. The bootstrap framework has lots of CSS and JS code that it provides to its users. The need for CDNs arises because bootstrap users are located all over the world. That is why you see **"https://cdn"** prefixed with the bootstrap links which means the bootstrap files will be fetched from the nearest CDN according to your geographical location.

Script Tags for Bootstrap

Bootstrap uses javascript for some of its functionalities. **JS script tags for bootstrap must be placed at the end of the body section of the HTML page.** Browser scans an HTML document from top to bottom to render it on the screen. So the HTML, CSS must be rendered first only then the functionality code (JS) will operate correctly on it. Otherwise, if you add functionality before rendering content on the screen, the functionality won't be able to operate as there's no content to be operated on.

Bootstrap

Introduction

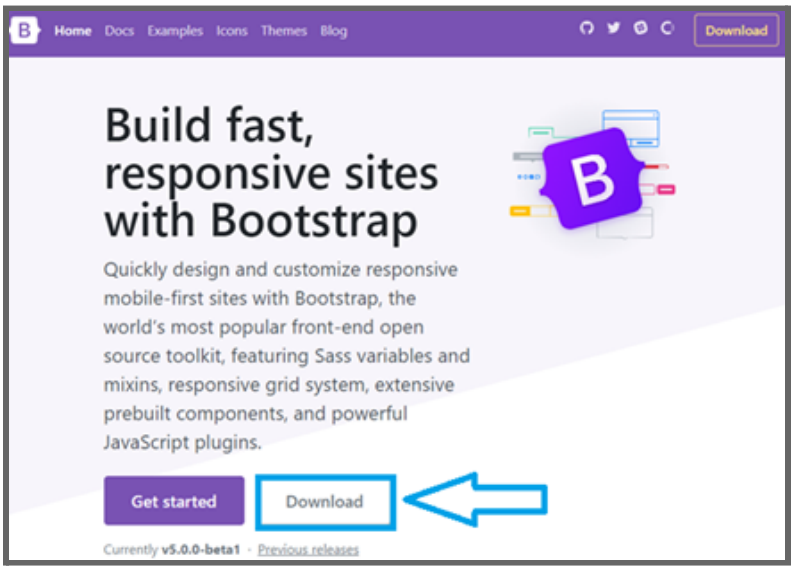
Bootstrap is a big collection of handy, reusable code written in HTML, CSS, and JavaScript. It's a front-end development framework that enables developers and designers to quickly build fully responsive websites. It is a free and open-source project, hosted on GitHub, and originally created by (and for) Twitter.

Bootstrap saves you from writing lots of CSS code, giving you more time to spend on designing web pages. It is flexible and easy to use. Its main advantages are that it is responsive by design, it maintains wide browser compatibility, it offers consistent design by using reusable components, and it is very easy to use and quick to learn.

Installing Bootstrap 4

You can install Bootstrap in two ways:

- The standard way is a very simple method where you can easily download combined and minified JavaScript and CSS bundles that can be later used for your web application project.



- Another way is through **CDNS** or Content Delivery Networks. If you use CDNs, then you don't even have to download Bootstrap to your PC. All you need to do is to add the following code into your existing HTML code:

1. Copy-paste this <link> into your <head> before all other stylesheets to load the Bootstrap CSS:

```
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/boots
trap.min.css" rel="stylesheet"
integrity="sha384-giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTf
AAsrekwKmP1" crossorigin="anonymous">
```

2. As many of the bootstrap components require the use of **JavaScript** to function, specifically, they require their own JavaScript plugins. Place the following <script> near the end of your pages, right before the closing </body> tag, to enable it.

```
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/js/bootstr
ap.bundle.min.js"
integrity="sha384-ygbV9kiqUc6oa4msXn9868pTtWMgiQaeYH7/t7LECLbyPA2x65Kgf
80OJFdroafW" crossorigin="anonymous"></script>
```

Bootstrap Breakpoints

Bootstrap includes six default breakpoints, sometimes referred to as grid levels, for building responsively. These breakpoints can also be customized, but for that, you'll need to change Bootstrap's CSS files.

In Bootstrap, breakpoints are the building blocks of making a responsive design. You can refer to this table for all the bootstrap breakpoints.

Breakpoint	Class infix	Dimentions
X-Small	None	0-576px
Small	sm	>=576px
Mediium	md	>=768px
Large	lg	>=992px
Extra Large	xl	>=1200px
Extra Extra large	xxl	>=1400px

You can use them to control the layout of your website according to different screen sizes.

You don't need to remember all these dimensions; you just need to remember the class names of respective breakpoints.

Bootstrap Grid System

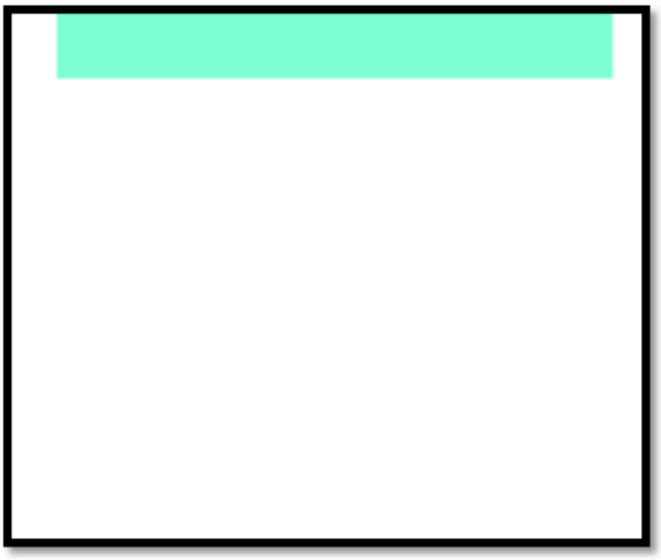
Bootstrap uses a flexbox-grid system which consists of a series of containers, rows, and columns to build its layout and align the content of the page.

- There is a 12-column grid system
- All the items must be in rows
- To prevent things from going full width use the wrapper class
- You can put rows inside of rows

Following is basic structure of Bootstrap grid −

```
<div class="container">
    <div class="row">
        <div class="col" style="background-color: aquamarine; height: 100px">
        </div>
    </div>
</div>
```

We can see in the above example, inside a container there can be rows, inside which there can be multiple elements. Output of the above code will look something like this:

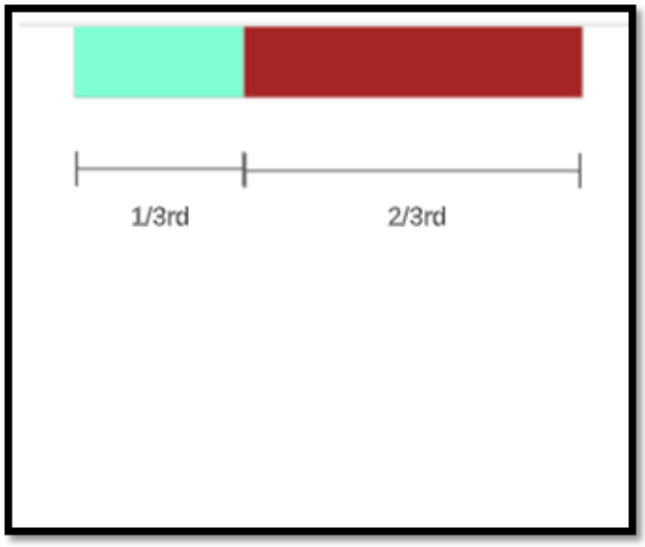


Bootstrap adds some padding by default. As we have not specified how wide this element will be that's why it will take the complete width available.

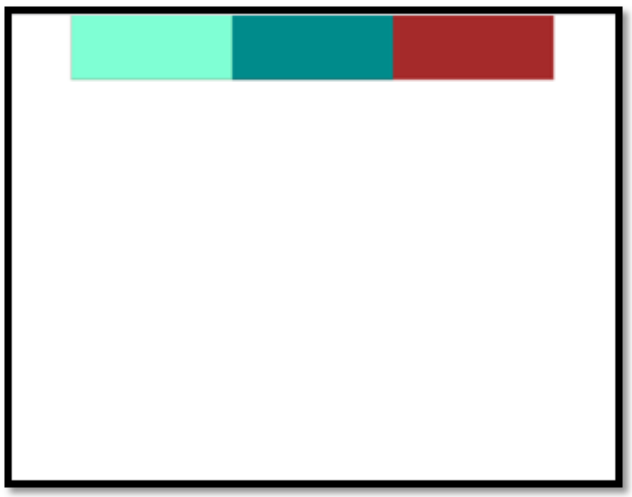
If you want the container to use the complete width of the browser, you can use container-fluid class, which will remove the padding.

If you want to specify the number of columns an element will occupy, you can write in this way:

```
<div class="container">
  <div class="row" style="background-color: brown;">
    <div class="col-4" style="background-color: aquamarine; height: 100px;">
    </div>
  </div>
</div>
```



As we can see, now it's occupying 4-cols or 1/3rd of the 12-cols. Let's add one more element which will occupy 4-cols:

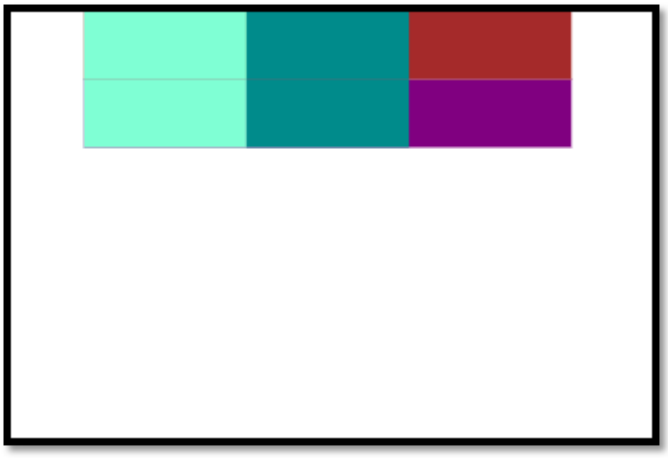


```
<div class="container">
    <div class="row" style="background-color: brown;">
```

```
        <div class="col-4" style="background-color: aquamarine; height:
100px;"></div>
        <div class="col-4" style="background-color: darkcyan; height:
100px;"></div>
    </div>
</div>
```

Now we can see that we have 2 elements occupying 1/3rd of the 12-col width each. Similarly, we can have multiple rows as well:

```
<div class="container">
    <div class="row" style="background-color: brown;">
        <div class="col-4" style="background-color: aquamarine; height: 100px;">
        </div>
        <div class="col-4" style="background-color: darkcyan; height: 100px;">
        </div>
    </div>

    <div class="row" style="background-color: purple;">
        <div class="col-4" style="background-color: aquamarine; height: 100px;">
        </div>
        <div class="col-4" style="background-color: darkcyan; height: 100px;">
        </div>
    </div>
</div>
```



By default, all your rules will be applicable to X-Small devices and to all other breakpoints beyond xs.

If you want to specify the number of columns an element occupies on different breakpoints, you can do that as well:

```
<div class="col-4 col-sm-6 col-lg-2" style="background-color: aquamarine; height: 100px;"> </div>
```

This means that the element will occupy 4-col width on x-small screens, and 6-col width on small screens and 2-col width on large screen and bigger screen devices.

Display Utilities

Now, we'll see different display utilities for layout in Bootstrap using which we can toggle the display values of components. Different values for display property are:

- none
- inline
- inline-block
- block
- table
- table-cell
- table-row
- flex
- inline-flex

You already know many of these properties, like none is used if you want to hide an element. Display utility classes that apply to all breakpoints, from xs to xl. If you want to set/change display properties, you'll need to add classes in this format:

.d-{value} for xs

.d-{breakpoint}-{value} for sm, md, lg, and xl.

where value is one of the properties mentioned above.

For example:

If you want to set the display of an element to block, you can do it this way:

```
<div class="d-block"></div>
```

If you want an element to be hidden on medium screens and visible on extra-small, small, large and extra-large screens:

```
<div class="d-block d-md-none d-lg-block"></div>
```

If you want an element to be hidden only on xs:

```
<div class="d-none d-sm-block"></div>
```

Spacing Utilities

You can use these to add responsive margin and padding utility classes to modify an element's appearance by assigning a responsive-friendly margin or padding values to an element.

Bootstrap also includes support for spacing properties like margin and padding. The classes are named using the format **{property}{sides}-{size}** for **xs** and **{property}{sides}-{breakpoint}-{size}** for **sm, md, lg, and xl** where:

property is one of:

- **m** – for classes that set **margin**
- **p** – for classes that set the **padding**

sides are one of:

- **t** - for classes that set **margin-top or padding-top**
- **b** - for classes that set **margin-bottom or padding-bottom**
- **l** - for classes that set **margin-left or padding-left**
- **r** - for classes that set **margin-right or padding-right**
- **x** - for classes that set both ***-left and *-right**
- **y** - for classes that set both ***-top and *-bottom**
- **blank** - for classes that set a margin or padding on **all 4 sides** of the element

and **size** is one of:

- 0 - for classes that eliminate the margin or padding by setting it to 0
- 1 - (by default) for classes that set the margin or padding to $spacer * .25
- 2 - (by default) for classes that set the margin or padding to $spacer * .5
- 3 - (by default) for classes that set the margin or padding to $spacer
- 4 - (by default) for classes that set the margin or padding to $spacer * 1.5
- 5 - (by default) for classes that set the margin or padding to $spacer * 3
- auto - for classes that set the margin to auto

where **$spacers** is a **Sass** variable, and its value can be customized.

For e.g.:

If you want to set the margin-top to 0:

```
<div class="mt-0"></div>
```

If you want to set the both left and right to 0:

```
<div class="px-0"></div>
```

For more information, you can have a read from the link below:

https://getbootstrap.com/docs/5.0/utilities/spacing/

Other Utilities

Border: You can use border utilities to add or remove an element's borders. The different border classes available are:

Additive: These are used to add borders.

- .border: This class adds a border all around the element.
- .border-top : This class adds a border on the top edge of the element.
- .border-left : This class adds a border on the left edge of the element.

- .border-right : This class adds a border on the right of the element.
- .border-bottom : This class adds a border on the bottom of the element.

Subtractive: These are used to remove borders.

- .border-0 : removes the border from all around the element.
- .border-top-0: removes the border from the top of the element
- .border-left-0: removes the border from the left of the element.
- .border-right-0: removes the border from the right of the element
- .border-bottom-0: removes the border from the bottom of the element

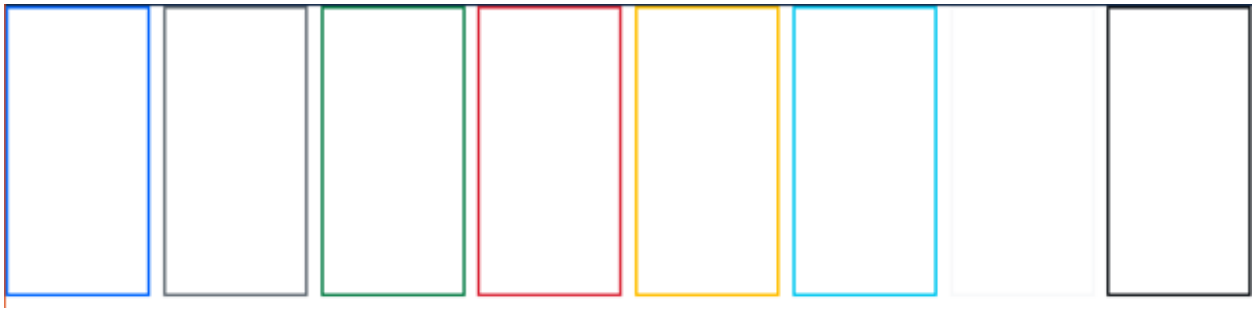
Border Color: Any color can be added to the border by using the following border-color classes:

- border border-primary
- border border-secondary
- border border-success
- border border-danger
- border border-warning
- border border-info
- border border-light
- border border-dark
- border border-white

Look at the code below, where we create spans with colored borders:

```
    <span class="border border-primary"></span>
    <span class="border border-secondary"></span>
    <span class="border border-success"></span>
    <span class="border border-danger"></span>
    <span class="border border-warning"></span>
    <span class="border border-info"></span>
    <span class="border border-light"></span>
    <span class="border border-dark"></span>
    <span class="border border-white"></span>
```

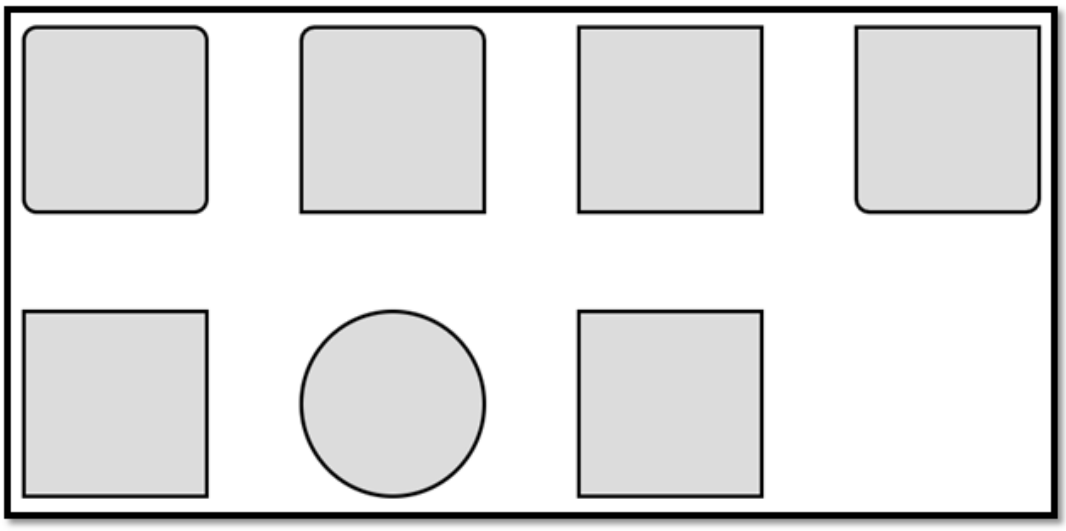
Output of the above code will look something like this:



Border Radius: A border-radius is used to make the corner of the border curved. The more is the radius, the more curved and round it will be. In bootstrap, the following classes as used in the code are used to implement radius at particular corners:

```
    <span class="rounded"></span>
    <span class="rounded-top"></span>
    <span class="rounded-right"></span>
    <span class="rounded-bottom"></span>
    <span class="rounded-left"></span>
    <span class="rounded-circle"></span>
    <span class="rounded-0"></span>
```

Output:



Colors: In Bootstrap, you can convey your meaning through colors, as it has a handful of contextual utility classes. The following classes can be used to color text:

```
<p class="text-primary">.text-primary</p>
    <p class="text-secondary">.text-secondary</p>
    <p class="text-success">.text-success</p>
```

```
    <p class="text-danger">.text-danger</p>
    <p class="text-warning">.text-warning</p>
    <p class="text-info">.text-info</p>
    <p class="text-light bg-dark">.text-light</p>
    <p class="text-dark">.text-dark</p>
    <p class="text-muted">.text-muted</p>
    <p class="text-white bg-dark">.text-white</p>
```

Output:

Background color: Similar to the text color classes, you can easily set the background color and text of elements using these classes:

```
<div class="bg-primary text-white">.bg-primary</div>
    <div class="bg-secondary text-white">.bg-secondary</div>
    <div class="bg-success text-white">.bg-success</div>
    <div class="bg-danger text-white">.bg-danger</div>
    <div class="bg-warning text-dark">.bg-warning</div>
```

```
    <div class="bg-info text-white">.bg-info</div>
    <div class="bg-light text-dark">.bg-light</div>
    <div class="bg-dark text-white">.bg-dark</div>
    <div class="bg-white text-dark">.bg-white</div>
```

Float and Clearfix: You can float an element to the right with the **.float-right** class or to the left with **.float-left,** and clear floats with the .**clearfix** class:

```
<div class="clearfix">
        <span class="float-left">Float left</span>
        <span class="float-right">Float right</span>
</div>
```

Width: You can set the width of an element with **w-*** classes:

```
    <div class="w-25 bg-primary">Width 25%</div>
    <div class="w-50 bg-warning">Width 50%</div>
    <div class="w-75 bg-danger">Width 75%</div>
    <div class="w-100 bg-secondary">Width 100%</div>
    <div class="mw-100 bg-dark">Max Width 100%</div>
```

Height: You can set height of an element using h-* classes:

```
<div>
    <div class="h-25 bg-secondary">Height 25%</div>
    <div class="h-50 bg-dark">Height 50%</div>
    <div class="h-75 bg-danger">Height 75%</div>
    <div class="h-100 bg-warning">Height 100%</div>
    <div class="mh-100 bg-primary">Height 100%</div>
</div>
```

Visibility: You can use classes **visible** and **invisible** to control the visibility of the elements.

```
<div class="visible">I'm here!</div>
<div class="invisible">I'm gone!</div>
```

If you want to know more about utilities, visit the link below:

https://getbootstrap.com/docs/5.0/utilities/api/

Bootstrap Components

Bootstrap has different components like Alerts, Dropdowns, Cards, Buttons, etc., which can be used in your website directly saving a lot of time.

Buttons: Bootstrap provides button classes to achieve button styles: .btn, .btn-default, .btn-primary, .btn-success, .btn-info, .btn-warning, .btn-danger, .btn-link.

Example:

```
<button type="button" class="btn btn-default">Default</button>
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-link">Link</button>
<button type="button" class="btn">Basic</button>
```

In Bootstrap, buttons can be of various sizes: **.btn-lg(large), .btn-sm(small), .btn-xs(extra-small):**

```
<button type="button" class="btn btn-primary btn-lg">Large</button>
<button type="button" class="btn btn-primary">Normal</button>
```

```
<button type="button" class="btn btn-primary btn-sm">Small</button>
<button type="button" class="btn btn-primary btn-xs">XSmall</button>
```

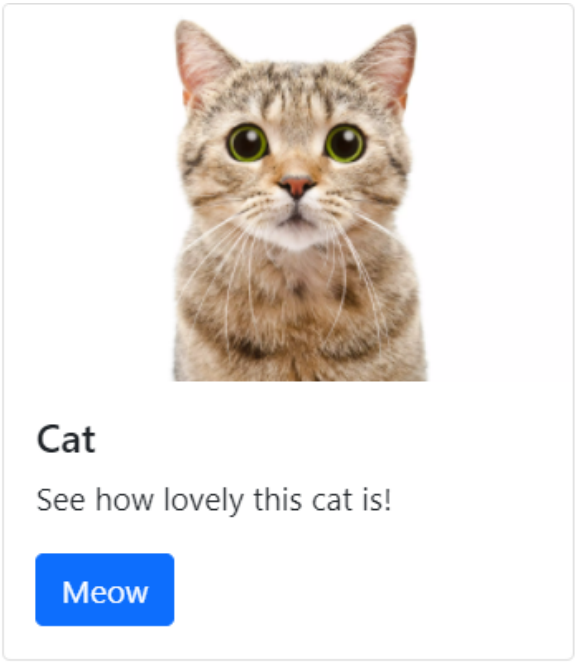
There are block level buttons that occupy the entire width of the parent element. You have to add the class **.btn-block** to create a block level button:

```
<button type="button" class="btn btn-secondary btn-block">Button 1</button>
```

A button can be set to be active or inactive. The classes **.active** and **.disabled** makes a button appear active or inactive respectively:

```
<button type="button" class="btn btn-default active">Active</button>
<button type="button" class="btn disabled">Disabled</button>
```

Cards: Bootstrap 4 has cards that are bordered boxes with some padding around their content. It includes options for headers, footers, content, colors, etc.



A card is a flexible and extensible content container. It includes options for headers and footers, a wide variety of content, contextual background colors, and powerful display options.

A basic card is created with the **.card** class, and the content inside the card has a **.card-body** class:

```
<div class="card">
    <div class="card-body">Basic card</div>
</div>
```

Card Header: The **.card-header** class adds a heading to the card and the **.card-footer** class adds a footer to the card:

```
<div class="card">
    <div class="card-header">Header</div>
    <div class="card-body">Content</div>
    <div class="card-footer">Footer</div>
</div>
```

You can also add background colors to cards by adding contextual classes **(.bg-primary, .bg-success, .bg-info, .bg-warning, .bg-danger, .bg-secondary, .bg-dark and .bg-light).**

Card images: The **.card-img-top** places an image to the top of the card. With **.card-text**, text can be added to the card. Text within .card-text can also be styled with the standard HTML tags.

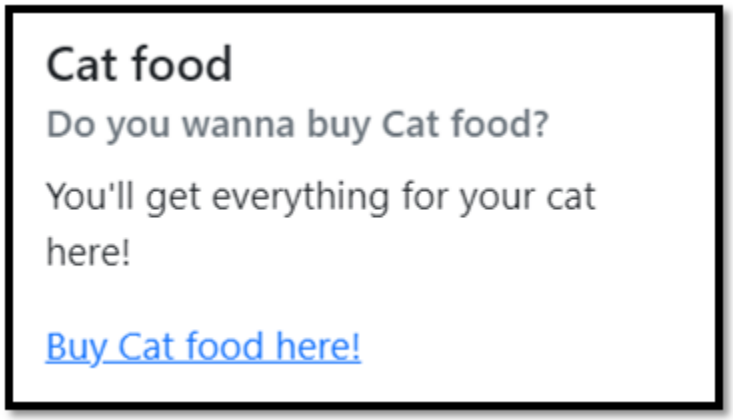
```
<div class="card">
    <img src="cat.png" class="card-img-top" alt="cat_image">
    <div class="card-body">
    <p class="card-text">See, how lovely this cat is!</p>
    </div>
</div>
```

Card Title, Text, and Links: Card titles can be used by adding a **.card-title** class to a <h*> tag.

In the same way, links are added and placed next to each other by adding **.card-link** to a <a> tag.

```
<div class="card" style="width: 300px;">
   <div class="card-body">
     <h5 class="card-title">Cat food</h5>
     <h6 class="card-subtitle mb-2 text-muted">Do you wanna buy Cat food?</h6>
     <p class="card-text">You'll get everything for your cat here!</p>
     <a href="#" class="card-link">Buy Cat food here!</a>
   </div>
</div>
```

Output:



Dropdowns: It's a toggleable menu that allows the user to choose one the predefined options. They're made interactive with the included Bootstrap dropdown JavaScript plugin.

```
<div class="dropdown">
<button   class="btn   btn-secondary   dropdown-toggle"   type="button"
data-toggle="dropdown">My dropdown<span class="caret"></span>
</button>
  <ul class="dropdown-menu">
        <li><a href="#">1</a></li>
        <li><a href="#">2</a></li>
        <li><a href="#">3</a></li>
  </ul></div>
```

The **.dropdown-header** class is used to add headers inside the dropdown menu:

```
<li class="dropdown-header">Dropdown header 1</li>
```

You can read more about components from the link below:

https://getbootstrap.com/docs/5.0/components/accordion/

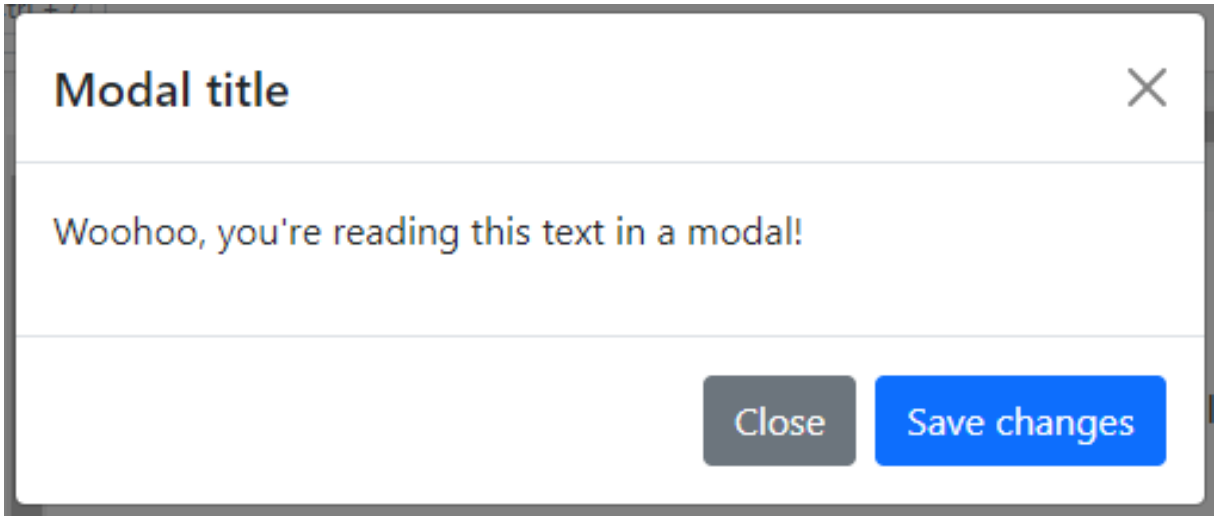
Modals

Modals are a dialog box/popup window that is displayed on top of the current page. They are built with HTML, CSS, and JavaScript. Bootstrap only supports one modal window at a time as nested modals are considered to produce a bad user experience.

Modal Components: These are the modal header, modal body (required for padding), and modal footer (optional).

A modal example:

```
 <!-- Button triggers the modal -->
    <button type="button" class="btn btn-primary" data-bs-toggle="modal"
data-bs-target="#exampleModal">
    Launch demo modal
  </button>

    <!-- Modal -->
    <div class="modal fade" id="exampleModal">
        <div class="modal-dialog">
      <!-- Modal content-->
            <div class="modal-content">
                <div class="modal-header">
                    <h5 class="modal-title" id="exampleModalLabel">Modal
title</h5>
                    <button type="button" class="btn-close"
data-bs-dismiss="modal" aria-label="Close"></button>
                </div>
                <div class="modal-body">
                    ...
                </div>
                <div class="modal-footer">
                            <button   type="button"   class="btn   btn-secondary"
data-bs-dismiss="modal">Close</button>
         <button type="button" class="btn btn-primary">Save changes</button>
                </div>
            </div>
       </div></div>
```



Modal explained:

The parent of the modal must have an **ID** that should be the **same** as the value of the data-target attribute used to trigger the modal ("exampleModal").

The **.modal** class identifies the content of as a modal and brings focus to it. The **.fade** class adds a **transition** effect that fades the modal in and out.

You can remove this class if you do not want this effect. The attribute role=" dialog" improves accessibility for people using **screen readers**. The **.modal-dialog** class sets the proper width and margin of the modal.

Modal content explained: The having class="modal-content" will style the modal (border, background-color, etc.). Add the modal's header, body, and footer inside this .

The .modal-header class is used to define the style for the header of the modal. The <button> inside the header has a data-dismiss="modal" attribute which closes the modal if you click on it. The **.close** class styles the close button, and the **.modal-title** class styles the header with a proper line-height.

The **.modal-body** class will be used to define the style for the body of the modal. You can add any HTML here; paragraphs, images, videos, etc. The **.modal-footer** class is used to define the style for the footer of the modal. You should remember that this area is right-aligned by default.

Modal Size: You can change the size of the modal by adding the **.modal-sm** class for small modals or **.modal-lg** class for large modals. You need to add this size class to the element with class **.modal-dialog.**

```
<div class="modal-dialog modal-lg">
```

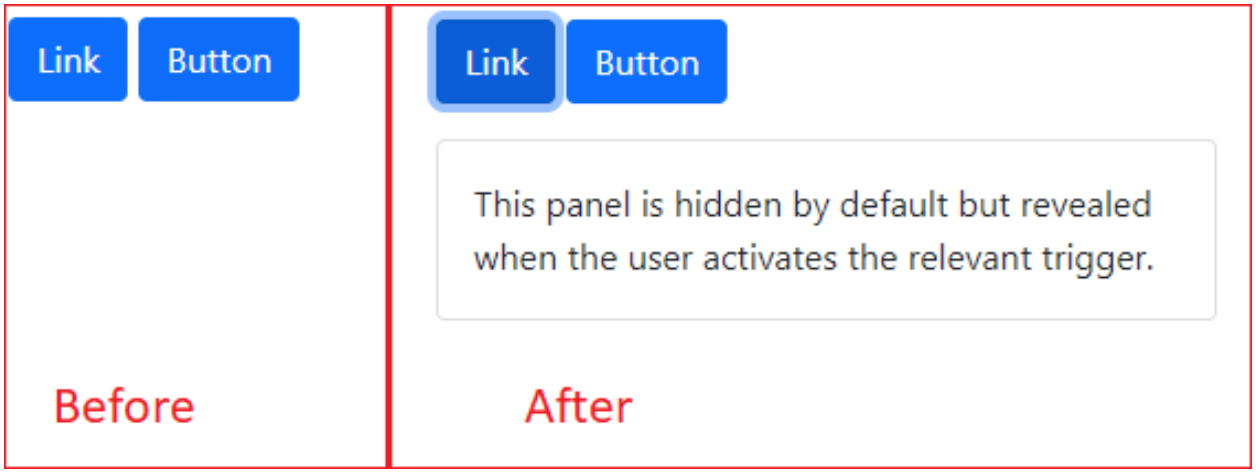
You can read more about modals from here:
https://getbootstrap.com/docs/5.0/components/modal/

Collapse

The collapse JavaScript plugin is used to show and hide content. Buttons or anchor tags are used as triggers that are bound to specific elements you toggle. Collapsing an element will animate the height from its current value to 0. Collapses are useful when you want to hide and show large amount of content:

```
  <p>
                    <a    class="btn    btn-primary"    data-bs-toggle="collapse"
href="#collapseExample"            role="button"            aria-expanded="false"
aria-controls="collapseExample">Link </a>
         <button class="btn btn-primary" type="button" data-bs-toggle="collapse"
data-bs-target="#collapseExample"                          aria-expanded="false"
aria-controls="collapseExample">
          Button
        </button>
    </p>
```

```
<div class="collapse" id="collapseExample">
 <div class="card card-body">
 This panel is hidden by default but revealed when the user activates the relevant trigger.
 </div>
</div>
```



The **.collapse** class indicates an element that can be **collapsed**. The content inside this element will be shown or hidden as per the click of a button. We can control (show/hide) the collapsible content by adding the **data-toggle="collapse"** attribute to an <a> or a <button> element. Then we can add the data-target="#id" attribute to connect the button with the collapsible content .

For <a> elements, you can use the href attribute instead of the data-target attribute:

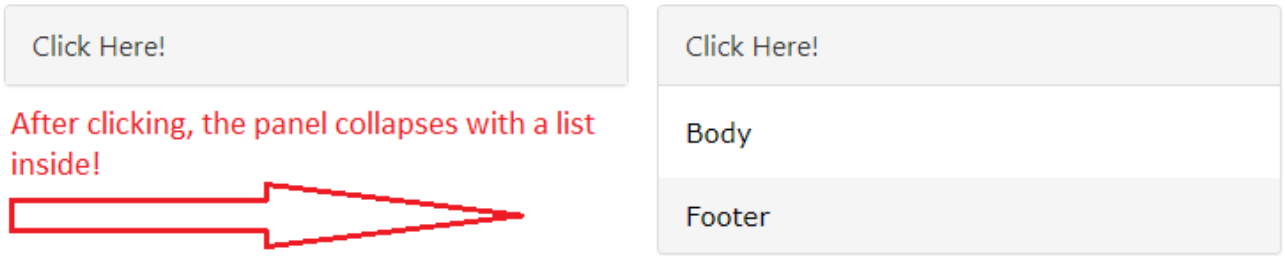
```
<a href="#collapseExample" data-toggle="collapse">Collapsible</a>

    <div id="collapseExample" class="collapse">
        Hello there!
    </div>
```

By default, the collapsible content is hidden. However, you can add the .**show** class to show the content by default:

```
<div id="collapseExample" class="collapse show">
    Random Text
</div>
```

There are collapsible panels also, which you can use:

```
<div class="panel-group">
    <div class="panel panel-default">
       <div class="panel-heading">
           <h4 class="panel-title">
               <a data-toggle="collapse" href="#collapse1">Click Here!</a>
            </h4>
        </div>
        <div id="collapse1" class="panel-collapse collapse">
             <div class="panel-body">Body</div>
             <div class="panel-footer">Footer</div>
        </div>
    </div>
</div>
```



Navs

If you want to create navigation menus, Bootstrap includes classes that can help you. Simple horizontal menu can be created by adding the class **.nav** to a <ul>, and **.nav-item** class to each <li>. If there are links inside <li>, add class **.nav-link** to them:


```
<ul class="nav">
        <li class="nav-item">
            <a class="nav-link" href="#">Link</a>
```

```
        </li>
        <li class="nav-item">
            <a class="nav-link disabled" href="#">Disabled</a>
        </li>
    </ul>
```

If you want to align the nav to the center, add the class **.justify-content-center,** and if you want to align it to the right-end, add the class **.justify-content-end.**

```
    <!-- Centered nav -->
    <ul class="nav justify-content-center">

    <!-- Right-aligned nav -->
    <ul class="nav justify-content-end">
```

You can create a vertical nav bar by adding the class **.flex-column**:

```
<ul class="nav flex-column">
  <li class="nav-item">
       <a class="nav-link" href="#">Link</a>
  </li>
  <li class="nav-item">
       <a class="nav-link disabled" href="#">Disabled</a>
  </li>
</ul>
```

You can also turn this navigation bar into navigation tabs by adding the .nav-tabs class. You'll need to add the class **.active** to the link you want to be active.


```
<ul class="nav nav-tabs">
   <li class="nav-item">
        <a class="nav-link active" href="#">ActiveLink 1</a>
   </li>
   <li class="nav-item">
        <a class="nav-link disabled" href="#">Disabled Link 1</a>
   </li>
</ul>
```

You can also turn your nav menu into **nav pills** by adding the .**nav-pills** class:

```
<ul class="nav nav-pills">
     <li class="nav-item">
         <a class="nav-link active" href="#">Link 1</a>
     </li>
     <li class="nav-item">
         <a class="nav-link disabled" href="#">Link 2</a>
     </li>
</ul>
```

You can read from the below link for more information:

https://getbootstrap.com/docs/4.0/components/navs/

Navbar

You can also use pre-built navbars which can extend or collapse, depending on the screen size. A basic horizontal navbar can be created using the **.navbar** class along with a responsive collapsing class: **.navbar-expand-xl|lg|md|sm** (stacks the navbar vertically on extra-large, large, medium, or small screens).

To add links inside the navbar, use a <ul> element with **class="navbar-nav"**. Then add <li> elements with a **.nav-item** class followed by an <a> element with a **.nav-link class**:


```
<nav class="navbar navbar-expand-sm bg-dark" style="margin-top: 10px;">
        <ul class="navbar-nav">
            <li class="nav-item">
                <a class="nav-link" href="#">Link 1</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Link 2</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Link 3</a>
            </li>
        </ul>
</nav>
```

You can remove the **.navbar-expand-xl|lg|md|sm** class to create a vertical navbar. You can add the justify-content-center class to center the navbar. To create colored navbars, you can use any of the .bg-color classes which will change the background color of the navbar (**.bg-primary, .bg-success, .bg-info, .bg-warning, .bg-danger, .bg-secondary, .bg-dark and .bg-light**).

The **.navbar-brand** class can be used to highlight the brand/logo/project name of your webpage:

```
<nav class="navbar navbar-expand-sm bg-light" style="margin-top: 10px;">
        <a class="navbar-brand" href="#">Brand Name</a>
        <ul class="navbar-nav">
            <li class="nav-item">
                <a class="nav-link" href="#">Link 1</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Link 2</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Link 3</a>
            </li>
        </ul>
    </nav>
```

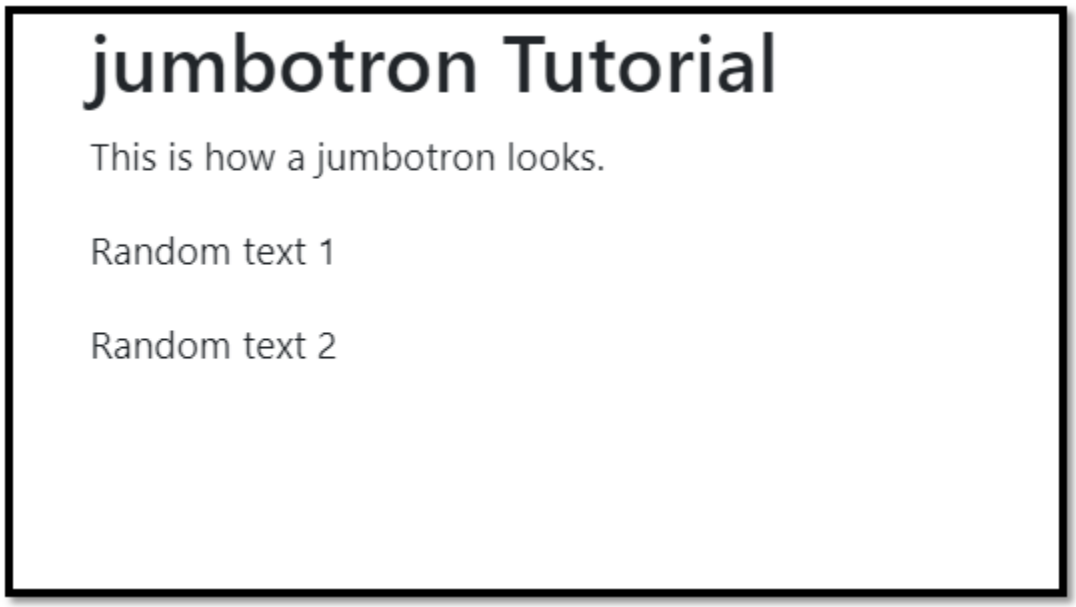
You can read more about navbars from this link: https://getbootstrap.com/docs/4.0/components/navbar/

Jumbotron

A jumbotron is like a big container for giving special attention to some content or information on the page. It's a grey box with rounded corners. The font size of the text inside the jumbotron is enlarged, and you can put all other bootstrap components discussed earlier, inside this jumbotron.

You can create a jumbotron with class **.jumbotron** using a :

```
<div class="container">
        <div class="jumbotron">
          <h1>jumbotron Tutorial</h1>
          <p>This is how a jumbotron looks.</p>
        </div>
        <p>Random text 1</p>
        <p>Random text 2</p>
      </div>
```

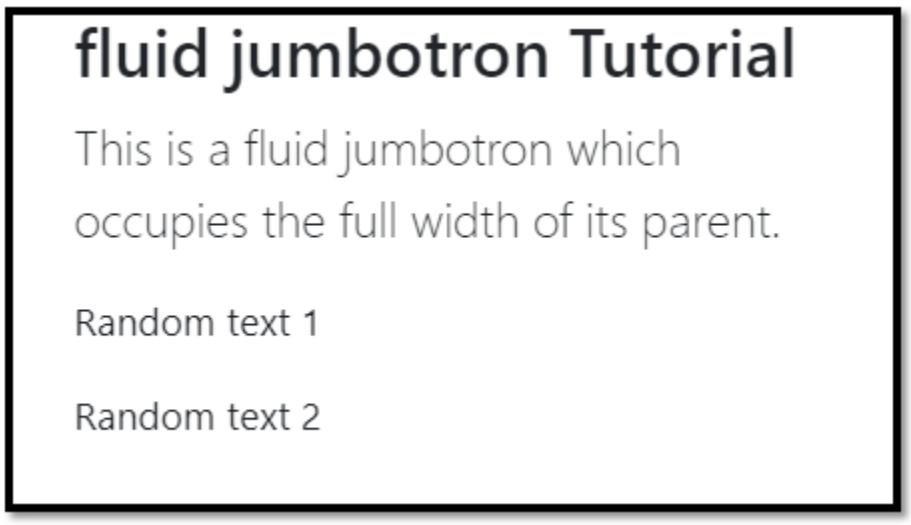


You can place the jumbotron inside a with .container class to prevent the jumbotron to extend the screen edges.

To make the jumbotron full width, and without rounded corners, you can add the **.jumbotron-fluid** modifier class and add a **.container** or **.container-fluid** within.

```
<div class="container-fluid">
        <div class="jumbotron jumbotron-fluid">
          <h1>fluid jumbotron Tutorial</h1>
                  <p class="lead">This is a fluid jumbotron which occupies the
full width of its parent.</p>
```

```
    </div>
    <p>Random text 1</p>
    <p>Random text 2</p>
  </div>
```



For a better understanding, you can look at the Jumbotron template, which uses some part of everything we've learned till now:

https://getbootstrap.com/docs/4.0/examples/jumbotron/#