HTML

Head

```
<head>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width, initial scale=1.0" />    <!--For responsive-->
        <meta name="description" content="HTML CSS and JavaScript First Course-- Udemy" />    <!--While googling, this part will be displayed as heading-->
        <meta name="keywords" content="HTML CSS and JavaScript" />    <!--While googling, this part will be displayed as description, below the link-->
        <meta name="author" content="Swapnadip Saha" />    <!--In case of blog site, this could be helpful-->
        <meta name="robots" content="index, follow" />    <!--whether home page is index or no index, etc. -->
        <title>First HTML and CSS</title>
</head>
```

Tags

	-- For Bold
	-- For Italic
<mark>	-- For Highlighted
<del>	-- Deleted (Strike Through)
<ins>	-- Inserted (Under Strike)
<sup>	-- Superscripted
<sub>	-- Subscripted

Block vs Inline

https://www.w3schools.com/html/html_blocks.asp

These are Block elements

<address> <article> <aside> <blockquote> <canvas> <dd> <dl> <dt> <fieldset> <figcaption> <figure> <footer> <form> <h1> -<h6> <header> <hr> <main> <nav> <noscript> <p> <pre> <section> <table> <tfoot> <video>

These are Inline elements

<a> <abbr> <acronym> <b> <bdo> <big>
 <button> <cite> <code> <dfn> <i> <img> <input> <kbd> <label> <map> <object> <output> <q> <samp> <script> <select> <small>   <sub> <sup> <textarea> <time> <tt> <var>

Definition List

Like ol->li and ul->li this is dl->dt->dd

```
        <dl>
                <dt>Def 1</dt>
                <dd>Def 1 Item 1</dd>
                <dd>Def 1 Item 2</dd>
                <dt>Def 2</dt>
```

```
        <dd>Def 2 Item 1</dd>
        <dd>Def 2 Item 2</dd>
        <dt>Def 3</dt>
        <dd>Def 3 Item 1</dd>
        <dd>Def 3 Item 2</dd>
    </dl>
```

Def 1
    Def 1 Item 1
    Def 1 Item 2
Def 2
    Def 2 Item 1
    Def 2 Item 2
Def 3
    Def 3 Item 1
    Def 3 Item 2

Same Page Link

```
<a href="#SamePageLink">Go to same page</a>
```

This will redirect to the element of the same page. Having the mentioned id.

```
<p id="SamePageLink">Same Page link, will come here.</p>
```

Mail To

```
<a href="mailto:hms.applsupport@tmckolkata.com">Email Link</a>
```

Download File

```
<a href="../TextFile.txt" title="Download the document">Download</a>
```

Figure and Fig Caption

```
<figure>
        <img src="../images.jpg" alt="Image Alt" />
        <figcaption>Demo for figcaption</figcaption>
</figure>
```

PRE

```
<pre>            This will print the content in the same format, as provided.
        With the exact spaces and tabs.
```

use Class for CSS id for JavaScript...*recommended*

DIV

div is **block** span is **inline.**

```
<div contenteditable="true">
```

```
        The content under div is editable
</div>
<div draggable="true">
        The content under div is draggable
</div>

<div draggable="true">
        <input placeholder="This is draggable and auto focussed" type="text" autofocus />
</div>
```

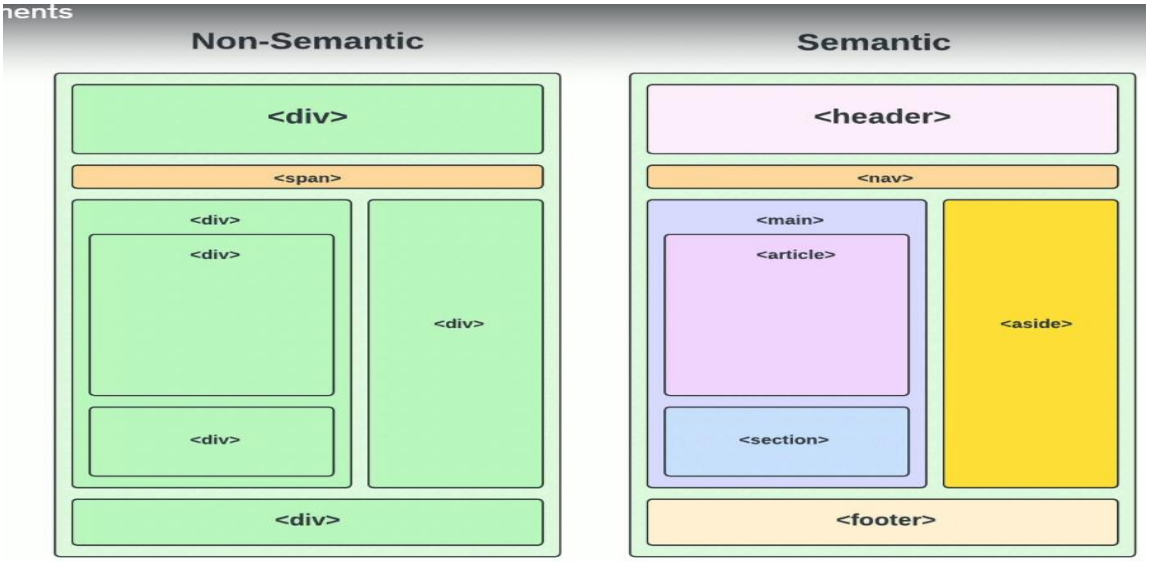
HTML Character Entities

Result	Description	Name	Number
	non-breaking space		
<	less than	&lt;	&#60;
>	greater than	&gt;	&#62;
&	ampersand	&amp;	&#38;
"	double quotation mark	&quot;	&#34;
'	single quotation mark	&apos;	&#39;
¢	cent	&cent;	&#162;
£	pound	&pound;	&#163;
¥	yen	&yen;	&#165;
€	euro	&euro;	&#8364;
©	copyright	&copy;	&#169;
®	trademark	&reg;	&#174;

Semantic Elements

Semantic Elements -- can be multiple in a page.

<header>	Header of the layout <footer> Footer of the layout
<nav>	Navigation area
<main>	The main content area
<article>	Publication area
<section>	Grouped area
<aside>	Sidebar/secondary content



Form Related

<input> Element Types

Text	Single-line text field
Email	Single line email
Password	Single line password with hidden characters
Number	Single line numbers
Textarea	Multi-line text field
Select	Dropdown list of options
Date	Date picker
Checkbox	Checkbox field with multiple options
Radio	Radio boxes with single option
File	File upload field
Submit	Submit button
Range	Slider input to select from a range
Color	Color picker

Label and Normal Textbox

```
            <label for="name">Name</label
            <input placeholder="Name" for="name" type="text" name="name" id="name"
required minlength="5" maxlength="200" />
        </div>
```

Label and Email

```
        <div>
            <label for="email">Email</label>
            <input placeholder="abc@xyz.com" for="email" type="email" name="email"
id="email" required />
        </div>
```

Label and Password

```
        <div>
            <label for="password">Password</label>
            <input placeholder="Password" for="password" type="password"
name="password" id="password" minlength="5" maxlength="20" required />
        </div>
```

Label and DDL

Multiple to select multiple options
Size is to set, how many items from the ddl will be visible

```
        <div>
            <label for="product">Product</label>
            <select for="product" id="product" name="product" multiple size= ”2”>
                <option value="0">IPhone</option>
                <option value="1" selected>iMac</option>
                <option value="2">Macbook</option>
                <option value="3">Macbook Pro</option>
            </select>
        </div>
```

Label and TextArea

```
<div>
    <label for="message">Message</label>
    <textarea id="message" name="message" for="Message" placeholder="Enter your message here....." rows="5" cols="50"></textarea>
</div>
```

Label and Checkboxes

```
<h2>Checkboxes</h2>
<div>
    <label for="html">
    <input for="html" type="checkbox" name="html" id="html" />HTML
    </label>
</div>
<div>
    <label for="JavaScript">
        <input for="JavaScript" type="checkbox" name="JavaScript" id="JavaScript" checked />JavaScript
    </label>
</div>
<div>
    <label for="css">
        <input for="css" type="checkbox" name="css" id="css" />CSS
    </label>
</div>
```

Label and Radio Buttons

```
    <h2>Radio Buttons</h2>
<div>
    <label for="small">
        <input for="small" type="radio" name="small" id="small" />Small
    </label>
</div>

<div>
    <label for="medium">
        <input for="medium" type="radio" name="medium" id="medium" checked />Medium
    </label>
</div>

<div>
    <label for="large">
        <input for="large" type="radio" name="large" id="large" />Large
    </label>
</div>
```

Label and Color Pickers

```
    <h2>Color Pickers</h2>
<div>
    <label for="color">Color      </label>
    <input for="color" type="color" name="color" id="color" />
</div>
```

Label and Date Time Week Month

```
        <h2>Date and time</h2>
    <div>
        <label for="date">Date      </label>
        <input for="date" type="date" name="date" id="date" />
    </div>

    <div>
        <label for="time">Time      </label>
        <input for="time" type="time" name="time" id="time" />
        </div>

        <div>
            <label for="week">Week      </label>
            <input for="week" type="week" name="week" id="week" />
        </div>

        <div>
            <label for="month">Month      </label>
            <input for="month" type="month" name="month" id="month" />
        </div>
```

Label and Range

```
        <div>
            <label for="range">Range      </label>
            <input for="range" type="range" name="range" id="range" min="0"
max="100" step="5" />
        </div>
```

Label and URL

```
        <div>
            <label for="url">URL      </label>
            <input for="url" type="url" name="url" id="url" />
        </div>
```

Button

```
        <div>
            <input type="submit" id="submit" value="Submit Input" />
            <button id="submitbtn">Submit Button </button>
        </div>
```

DataList

Combination of DDL and TextBox

Normal

```
        <label for="favLang">Favourite Languages</label>
        <input list="languages" id="favLang" name="FavLang" /> <!--list=language and
datalist id should be same-->
        <datalist id="languages">
            <option value="JavaScript"></option>
            <option value="PHP"></option>
            <option value="C#"></option>
```

```
                <option value="Python"></option>
                <option value="Ruby"></option>
            </datalist>
```

Time

```
            <label for="favLang">Popular Hours</label>
            <input type="time" list="popularHrs" /> <!--list=language and datalist id should be same-->
            <datalist id="popularHrs">
                <option value="10:00"></option>
                <option value="12:00"></option>
                <option value="15:00"></option>
                <option value="20:00"></option>
                <option value="21:00"></option>
            </datalist>
```

Range

```
<label for="tick">Tip Amount</label>
            <input type="range" list="tickMarks" id="tick" min="10" max="100" /> <!--list=language and datalist id should be same-->
            <datalist id="tickMarks">
                <option value="0"></option>
                <option value="10"></option>
                <option value="20"></option>
                <option value="30"></option>
                <option value="40"></option>
                <option value="50"></option>
                <option value="60"></option>
                <option value="70"></option>
                <option value="80"></option>
                <option value="90"></option>
                <option value="100"></option>
            </datalist>
```

Color Picker

```
<label for="color">Pick a color</label>
            <input type="color" list="chooseColors" id="color" /> <!--list=language and datalist id should be same-->
            <datalist id="chooseColors">
                <option value="#ffffff"></option>
                <option value="#800000"></option>
                <option value="#8B0000"></option>
                <option value="#A52A2A"></option>
                <option value="#DC143C"></option>


            </datalist>
```

Image Map

```
<img src="../workplace.jpg" alt="Image map Example" usemap="#computerMap" />
        <map name="computerMap">
            <area target="_blank" shape="rect" coords="34,44,270,350" href="https://www.dell.com/en-in/shop/work-from-home-deals/g15-gaming-laptop/spd/g-series-15-5530-laptop/gn5530n7hy001orb1o" alt="My laptop" />
```

```
                <area target="_blank" shape="rect" coords="290,172,333,350"
href="https://www.realme.com/in/realme-6-pro" alt="My Phone" />
                <area target="_blank" shape="circle" coords="337,300,45"
href="https://www.zinodavidoff.com/coffee" alt="Coffee" />
        </map>
```

IFrame

Creates a place/frame to open another webpage

```
<iframe src="PriceTag.html">PriceTag</iframe>

        <iframe src="https://youtube.com/embed/8sXRyHI3bLw">Youtube</iframe>
        <embed src="https://youtube.com/embed/8sXRyHI3bLw" />Youtube Embed tag
        <object data="https://youtube.com/embed/8sXRyHI3bLw" />Youtube Object tag

        <iframe
src="https://www.google.com/maps/embed/place/GMN+Towers/@22.627434,88.4228151,17z/dat
a=!3m1!4b1!4m6!3m5!1s0x39f89fd6de521221:0xa9db9fd64aa1c50f!8m2!3d22.627434!4d88.42539!
16s%2Fg%2F11ny0pyx4s?entry=ttu">Home</iframe>

                <iframe src="../c238861723_sis_doc.pdf">pdf</iframe>
```

SVG

Creates a custom graphics

```
        <svg width="200" height="200">

                <circle cx="70" cy="70" r="60" fill="blue" stroke="black" stroke-
width="5"></circle>
                <text x="15" y="75" font-size="20" font-family="verdana" fill="white">SVG
Circle</text>
        </svg>
        <a href="https://www.blobmaker.app/" target="_blank">Blobmaker SVG</a>
        <svg width="200" height="200" viewBox="0 0 200 200"
xmlns="http://www.w3.org/2000/svg">
                <path fill="#FF0066" d="M31.5,-27C39.7,-14.8,44.6,-
1.7,45.2,16.4C45.9,34.5,42.4,57.7,30.8,62.8C19.2,67.9,-0.5,55,-14.4,43.2C-28.2,31.4,-36.2,20.8,-42.4,6.1C-
48.7,-8.6,-53.1,-27.3,-45.8,-39.3C-38.4,-51.3,-19.2,-56.7,-3.8,-53.7C11.6,-50.7,23.2,-39.2,31.5,-27Z"
transform="translate(100 100)" />
        </svg>
        <svg width="200" height="200" viewBox="0 0 200 200"
xmlns="http://www.w3.org/2000/svg">
                <path fill="#24A148" d="M14.7,-13C22.2,-
2.2,33.7,4,33.5,8.8C33.3,13.5,21.4,16.6,12.9,16.9C4.4,17.2,-0.7,14.6,-8.6,12.7C-16.6,10.9,-27.4,9.8,-38.4,-
1.8C-49.5,-13.4,-60.8,-35.4,-54.4,-45.9C-48.1,-56.4,-24,-55.3,-10.2,-47.2C3.6,-39,7.1,-23.8,14.7,-13Z"
transform="translate(100 100)" />
        </svg>
```

Link

https://www.blobmaker.app/

Pop Over and Details

Pop Over: On Click, Popover to on page pop up

```
<button popovertarget="mypopover-1">Open Top Over 1</button>
<div id="mypopover-1" popover>Popover content 1</div>

<button popovertarget="mypopover-2">Open Top Over 2</button>
<div id="mypopover-2" popover>
	<h3>This is Pop Over 2 H3</h3>
	<p>Hello from popover 2</p>
</div>
```

Details: On screen expand and collapse

```
<details>
	<summary>Details</summary>
	Lorem ipsum dolor sit amet, consectetur
</details>

<details>
	<summary>More Details</summary>
	Lorem ipsum dolor sit amet, consectetur
</details>
```

Progress

```
<label for="file">Progress</label>
	<progress id="file" max="100" min="50">50%</progress>
```

Meter

```
<label for="fuel">Fuel</label>
	<meter value="50" max="100" min="0" low="20" high="80"
optimum="75">50/100</meter>
```

Media

Audio

Controls is necessary, without it the audio will not be displayed
Autoplay-- controls not included. Automatically start play. No controls will be there
Autoplay-- controls and loop included. Automatically start play in loop.

```
<audio src="../Audio.mp3" controls type="audio/mp3"></audio>

	<audio controls>
		<source src="../Sample_BeeMoved_96kHz24bit.flac" />
		<p>Browser is not supporting the audio</p>
	</audio>


	<!-- <audio src="../Audio.mp3" autoplay type="audio/mp3"></audio> -->
	<audio src="../Audio.mp3" type="audio/mp3"></audio>

	<!-- <audio src="../Audio.mp3" loop controls autoplay type="audio/mp3"></audio>
-->
	<audio src="../Audio.mp3" loop controls type="audio/mp3"></audio>



	<figure>
		<figcaption>
			<audio src="../Audio.mp3" controls type="audio/mp3"></audio>
			<a href="../Audio.mp3">Download</a>
```

```
            </figcaption>
        </figure>
```

Video

```
        <video width="50%" height="50%" src="../Samplemp4.mp4" controls
type="video/mp4"></video>

        <video width="50%" height="50%" controls>
            <source src="../Samplemp4.mp4" />
            <p>Browser is not supporting the video</p>
        </video>

        <video width="50%" height="50%" src=".../Samplemp4.mp4" autoplay
type="video/mp4"></video>

        <video width="50%" height="50%" src="../Samplemp4.mp4" loop controls autoplay
type="video/mp4"></video>

        <video width="50%" height="50%" src="../Samplemp4.mp4" muted loop controls
autoplay type="video/mp4"></video>

        <figure>
            <figcaption>
                <video width="50%" height="50%" src="../Samplemp4.mp4" controls
type="video/mp4"></video>
                <a href="../Samplemp4.mp4">Download</a>
            </figcaption>
        </figure>
```

CSS

```
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;500;600&display=swap"
rel="stylesheet">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.5.2/css/all.min.css" integrity="sha512-
SnH5WK+bZxgPHs44uWIX+LLJAJ9/2PkPKZ5QiAj6Ta86w+fsb2TkcmfRyVX3pBnMFcV7oQPJkl9QevSCWr3W6A
==" crossorigin="anonymous" referrerpolicy="no-referrer" />
```

Preference

1st Inline (inside style attribute)
2nd Internal <style></style>
3rd External File reference

Specificity

1st Inline (inside style attribute)
2nd ID i.e. #
3rd Class i.e. .
4th Element

putting !important in a CSS element in .css will have the highest Specificity

Universal Selector

```
*{

}
```

Style for all element, generally used for,

```
box-shadow:
box-sizing:
font-family:
padding:0;
margin:0;
```

etc. The styles defined here will be applied for all elements in the page. Need to override for the individual classes or Ids or elements, if required.

Multiple

```
.sporty, .casual, .formal {
```

Descendent

```
#container p {
background: blue;   All <p>s inside container will be blue
}

.main-menu li.last-item {
```

- **.main-menu**: This part selects any element with the class main-menu. It could be a , <ul>, or any other HTML element with this class.
- **li**: This specifies that the selector is targeting <li> elements within the element selected by .main-menu
- **.last-item**: This is a class applied to an <li> element. It selects the list item with the class last-item.

Important Links

Web font

Need to be imported into a project using a @font-face rule or a service like Google Fonts
Can be imported to the HTML file using <link> or the CSS file using @import
https://fonts.google.com/

Search for a font → Get Font → Get embedded code → Paste the link in html

Colors
https://htmlcolorcodes.com/color-names/

Font Awesome
https://fontawesome.com/

```
<h2>Icon</h2>
    <i class="fa-solid fa-check-to-slot"></i>
    <i class="fas fa-check"></i>
    <i class="fas fa-times"></i>
    <i class="fas fa-exclamation"></i>
    <i class="fas fa-exclamation-triangle"></i>
    <i class="fas fa-exclamation-circle"></i>
    <i class="fas fa-info"></i>
    <i class="fas fa-info-circle"></i>
    <i class="fas fa-question"></i>
    <i class="fas fa-question-circle"></i>
    <h2>Social Media Icon</h2>
    <i class="fab fa-facebook"></i>
    <i class="fab fa-twitter"></i>
    <i class="fab fa-instagram"></i>
    <i class="fab fa-whatsapp"></i>
    <i class="fab fa-youtube"></i>
    <i class="fab fa-linkedin"></i>
    <h2>Icon Sizes</h2>
    <i class="fas fa-check "></i>
    <i class="fas fa-check fa-2x"></i>
    <i class="fas fa-check fa-3x"></i>
    <i class="fas fa-check fa-4x"></i>
    <i class="fas fa-check fa-10x"></i> 10x is highest
    <i class="fas fa-check fa-11x"></i>
```

```
.fa-facebook, .fa-twitter, .fa-linkedin {
color: blue;
font-size: 50px; /* Can set font size here or predefined classes like fa-2x etc*/
}

.fa-youtube, .fa-instagram {
```

```
color: red;
font-size: 50px; /* Can set font size here or predefined classes like fa-2x etc*/
}

.fa-whatsapp {
color: lightgreen;
font-size: 50px; /* Can set font size here or predefined classes like fa-2x etc*/
}
```

Texture or Gradient

https://cssgradient.io/

CDN js

https://cdnjs.com/

favicon

https://favicon.io/favicon-generator/

Logo Maker

rem and em

```
line-height: 1.6;
     /*This is rem. Relative unit, Multiplier of root HTML font size. By default,
              16px, so 1.6=16*1.6=25.6px.  */

html
{
font-size: 10px;
/* Defining this the default will be now 10px and 1.6=16px*/
}
```

em is Multiplier of parent.

Font

```
font-size: x-large;
font-weight: bold;
font-style: italic;
font-variant: small-caps;
```

same as

```
font: italic bold x-large small-caps verdana;


font-family: Verdana, sans-serif;
```

Text

```
text-transform: capitalize; /* To capital the First characters*/
text-indent: 20px; /* To move the characters right 20px*/
letter-spacing: 2px; /* To provide space between letters*/
word-spacing: 5px; /* To provide space between words*/
text-align: center; /* To move the text at center, it may be left or right*/
line-height: 40px; /* Space between lines. Usually in universal selectors*/
```

Opacity or Alpha

```
opacity: 1; /*in rgba–a i.e. alpha for color opacity or transparency. Opacity  for
             text) */
background: rgba(0,0,0,0.5); /* 0 is no transparent, 1 is max transparent */
```

Overflow

```
overflow: hidden;    /* After setting the height and width, the extra text outside the
                     container will be truncated*/
overflow:scroll;     /* After setting the height and width, the extra text outside the
                     container will be truncated. And a scroll bar will be applied to
                     view the full text */
overflow-x: scroll; /* Scrolling will be applied for horizontal */
overflow-y: scroll; /* Scrolling will be applied for vertical */
```

Background

```
background-color: coral;
```

Texture

```
background-image: url('../contents/images/texture.jpg');
background-size: 200px;  /* If size is not specified, then the image in bg will be
             abnormally fitted*/
```

image

```
background-image: url('../contents/images/hero.png');
background-size: 200px; /* This will set the size of the image*/
```

or

```
background-size: cover; /* This will cover the single image, for the entire screen*/
background-repeat: no-repeat; /*By default repeat. To set the image is not repeating*/
background-position: center; /*Image will be at the center*/
```

or

```
background-position: -10px -500px;
```

In single line (order is important)

```
background: url('../contents/images/hero.png') no-repeat center/cover;
```

attachment

```
background-attachment: scroll;/*scroll is default. In this case background image will
move on scroll */
background-attachment:fixed ;/*the background image is fixed when scroll*/
```

Linear Gradient

```
  background: linear-gradient(to right, lightblue, darkblue);
  background: linear-gradient(to left, lightgreen, darkgreen);
  background: linear-gradient(to top, lightcoral, darkkhaki);
  background: linear-gradient(to bottom, lightcyan, darkcyan);
  background: linear-gradient(to bottom left, lightblue, darkblue);
```

Box Model

Content - Text, images, etc.
Padding - Space between content and border.
Border - Separates the padding & margin.
Margin - Space outside of border.

Box Model Properties

- width & height, max-width, max-height, min-width, min-height
- padding, padding-top, padding-right, padding-bottom, padding-left
- border, border-style, border-color, border-width
- margin, margin-top, margin-right, margin-bottom, margin-left
- box-sizing

If `max-width` is 600px it is the widest to go to 600px, but it can be smaller than that. But if `width` is 600px, then it is going to be strictly 600px, no matter what.

While setting the size in pixels, that will be fixed, as the number of pixels is fixed in a screen, while setting them in %, will decrease or increase the size of the element, on decreasing of increasing the browser screen, as % will calculate on the size of the container of the element.

After setting width of a div, if the contents of the div is going outside of the div, then need to use `overflow.`

```
overflow: hidden; /* It will cut the extra contents off*/

overflow: scroll; /* It will create scroll, for that div*/

overflow-x: scroll; /* It will create scroll only for x-axis or horizontal, for that div*/

overflow-y: scroll; /* It will create scroll only for y-axis or vertical, for that div*/
```

Padding and Margin:

```
padding: 20px 40px 100px 60px;

margin: 20px 40px 100px 60px;
```

meaning top 20 Right 40 Bottom 100 Left 60**. Order is important, TRBL**

```
padding: 20px 40px;

margin: 20px 40px;
```

meaning top and bottom 20 left and right 40. **Order is important, TB-LR**

```
padding: 20px 40px 30px;

margin: 20px 40px 30px;
```

meaning top 20 left and right 40 and bottom 30. Order is important, **T-LR-B**

```
margin-left: -20px;
```

margin can also be negative, but not padding. In case of -ve values it will act opposite of +ve.

```
margin: auto; /* This will set the content at the middle of the page/container. But there will be margin in top and margin */

margin: auto 100px; /* This will set the margin left and right as 100px. But there will be margin in top and margin */

margin: 100px auto; /* This will set the content at the middle of the page/container. with margin in top and margin 100px each*/
```

Box Sizing

Generally, define in universal selector

*{

}

By default, the width and height of an element is calculated like this:

width + padding + border = actual width of an element
height + padding + border = actual height of an element

This means: When you set the width/height of an element, the element often appears bigger than you have set (because the element's border and padding are added to the element's specified width/height).

The `box-sizing` property allows us to include the padding and border in an element's total width and height.

`box-sizing: border-box;` padding and border are included in the width and height properties (and min/max properties).

`box-sizing: content-box;` Default. The width and height properties (and min/max properties) include only the content. Border and padding are not included

Display

Specifies the display behavior (type of rendering box) of an

element

none	flex	table-cell
block	grid	list-item
inline	table	inherit
inline-block	table-row	initial

1. `display: none;` Here the text will disappear and there will be no space. Such that there was nothing
2. `visibility: hidden;` Here the text will disappear but there will be space. Such that there is something, but it is hidden

3. `display: inline;` Default for p is Block, will be replaced with inline.
4. `display: block;` Default for a is inline, will be replaced with block

For **inline elements** there would be **no vertical (top Bottom), only horizontal margin** and also **width and height are not working**

To make these works inline property needs to be change in block

5. `display: inline-block;` This will place all elements in **same line and the margin, width and height will work**

Position

Position an element w.rt. rest of the layout

```
.box-1 {
  position: static;
  left:30px; /* Not working. TRBL will not work*/
}
```

left right top bottom will not work for static.

Elements to be positioned accordingly to the normal flow of the document. The static position is not affected by the top, right, bottom, and left values. The position is fixed in its natural position.

```
.box-2 {
  position: relative;
  left:10px;
}
```

positions the element relative to the original position in the document. The element is positioned with the top, right, bottom, and left values. In this case the element will be shifted 10px left to its natural position, same for right, bottom and top.

```
.box-6 {
    position: absolute;
    bottom: 20px;
    right: 20px;
}
```

Position will be wrt the parent. Can be viewed by changing the value of top bottom left right. But if the values exceed the height and width of the parent. It will go outside

```
.box-3 {
  position: fixed;
  left:30px;
}
```

While scrolling, it will be fixed in a single position. **TRBL** works as usual like in relative.

```
.box-4 {
  position: sticky;
  top:300px;
}
```

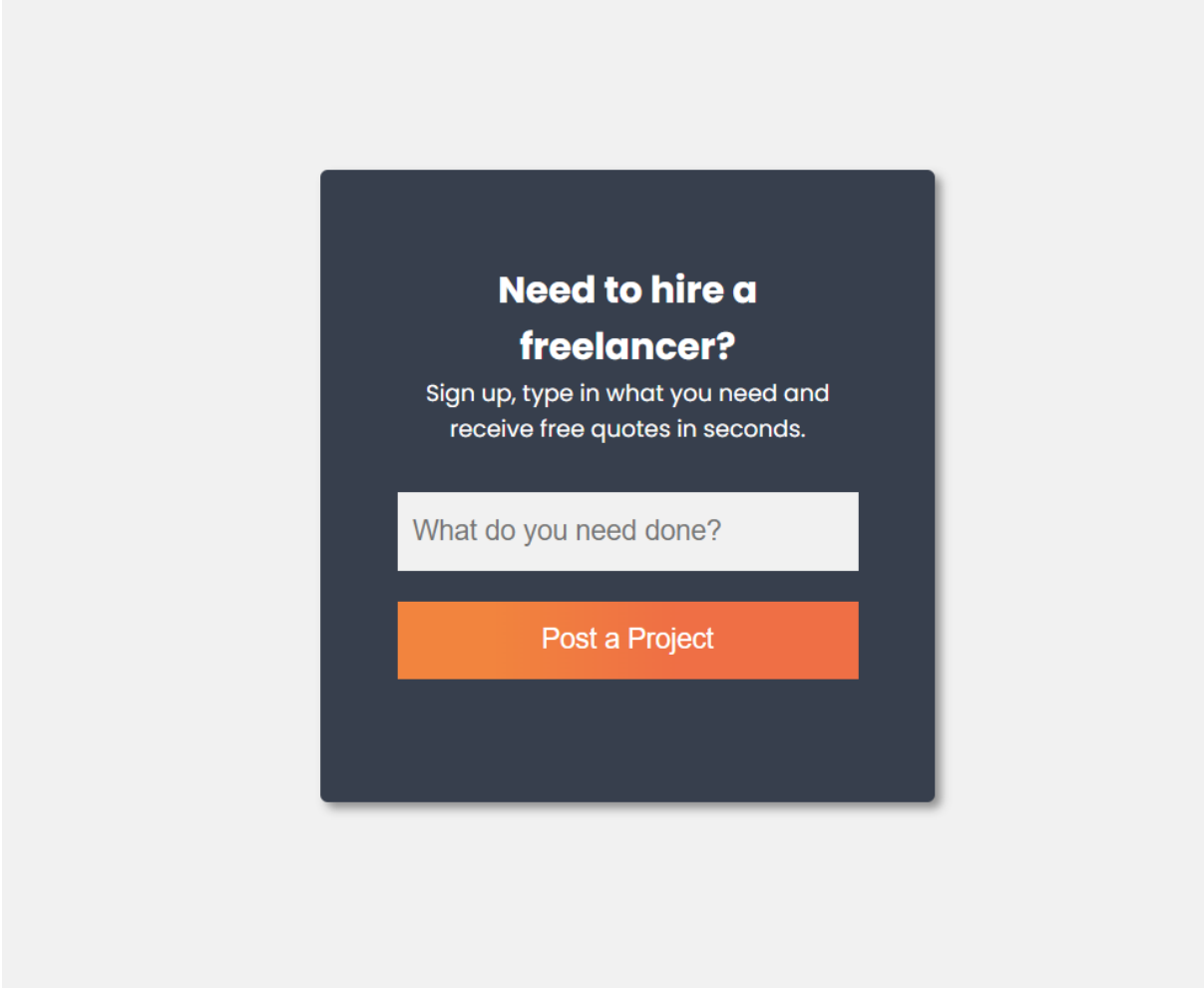
Combination of relative and fixed. It will be fixed from top 300px. Once reached, at that point while scrolling. It will be not fixed anymore

Z-index

If Z-index of fixed is higher than any other element(box-5 here), then the fixed element will go behind, else come forward. z-index can be negative also. **Will not be worked in static.**

A Simple Page using CSS

Sample Output



Html code

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
```

```
  <meta name="viewport" content="width=device-width, initial scale=1.0" />
<!--For responsive-->
  <meta name="description" content="HTML CSS and JavaScript First Course-- Udemy"
/>     <!--While searching, this part will be displayed as heading-->
  <meta name="keywords" content="HTML CSS and JavaScript" />     <!--While
searching, this part will be displayed as description, below the link-->
  <meta name="author" content="Swapnadip Saha" />     <!--In case of blog site,
this could be helpful-->
  <meta name="robots" content="index, follow" />    <!--whether home page is
index or no index, etc. -->

  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link
href="https://fonts.googleapis.com/css2?family=Lato:ital,wght@0,100;0,300;0,400;0
,700;0,900;1,100;1,300;1,400;1,700;1,900&display=swap" rel="stylesheet">

  <link href="styles/StyleSheet4.css" rel="stylesheet" />
  <title>Free Form Freelance</title>
</head>
<body>
  <div class="container">
    <div class="card">
      <h2>Need to hire a freelancer?</h2>
      <p>Sign up, type in what you need and receive free quotes in seconds. </p>
      <form>
        <div class="form-group">
          <input type="text" id="txtNeed" name="need" placeholder="What do you
need done?" />
        </div>
        <div class="form-group">
            <button type="submit">Post a Project</button>
        </div>
      </form>
 </div>
  </div>


</body>
</html>
```

CSS code

```
/*The universal selector. These styles will be applied to all individual elements
of the page. Need to override them (if required), whyle styling the other
elements.*/
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}


body {
    font-family: 'Poppins', sans-serif;
    background: #f1f1f1;
}

.container {
    width: 400px;
    margin: 150px auto; /*Generaly to make the element at the center of its
parent/container*/
```

```
}

.card {
    background: #373f4d;
    color: #fff;
    border-radius: 5px; /*To make rounded border*/
    text-align: center;
    padding: 60px 50px;
    box-shadow: 4px 4px 6px rgba(0, 0, 0, 0.4);
}

    .card h1 {
        font-size: 22px;
        margin-bottom: 15px;
        font-weight: 300;
    }

    .card p {
        font-size: 15px;
        margin-bottom: 30px;
    }

    .card .form-group {
        margin-bottom: 20px;
    }

    .card input[type='text'] { /*For input types*/
        width: 100%;
        padding: 15px 10px;
        font-size: large;
        border: none;
        background-color: #f1f1f1;
    }

    .card button {
        width: 100%;
        padding: 15px 10px;
        font-size: large;
        border: none;
        background: linear-gradient( 90deg, rgba(242, 132, 62, 1) 21%, rgba(239,
111, 69, 1) 61% ); /*from cssgradient.io*/
        color: #fff;
        cursor: pointer;
    }

        .card button:hover {
            background: linear-gradient( 90deg, rgba(232, 132, 62, 1) 21%,
rgba(229, 111, 69, 1) 61% );
        }
```

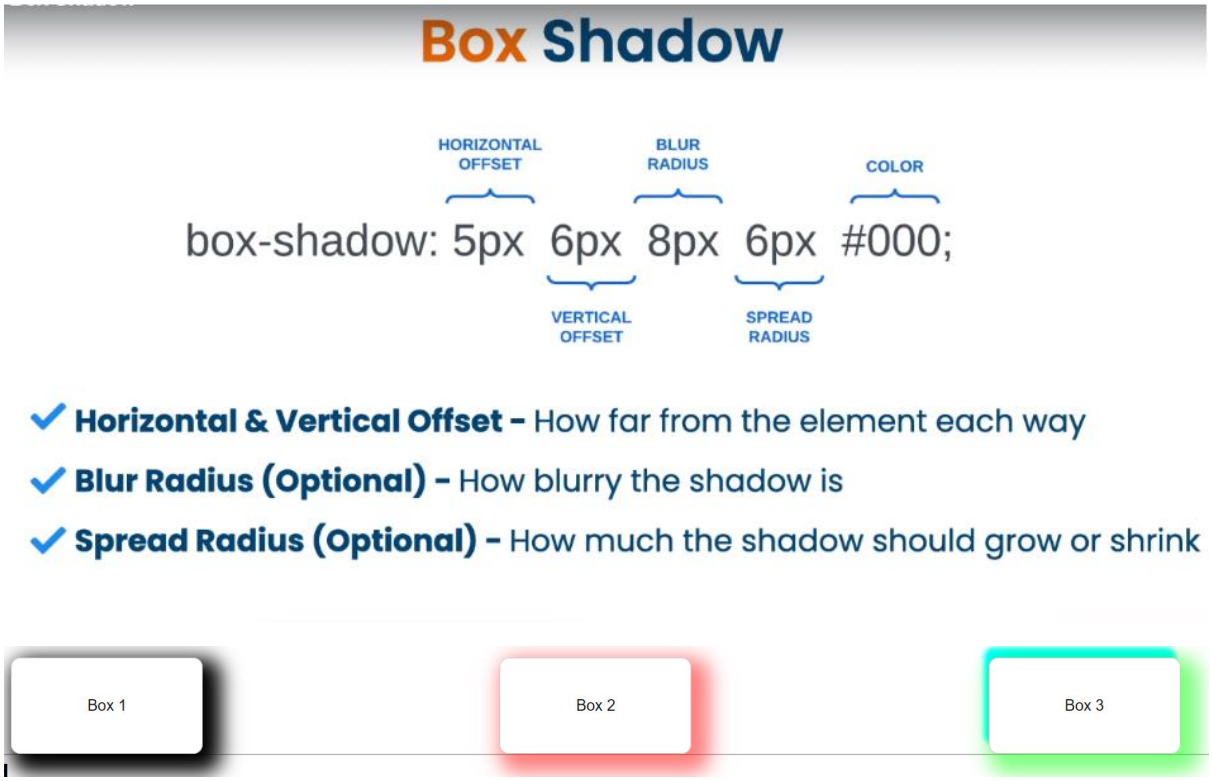
CSS cntd.

Box Shadow

Generally, define in [universal selector]

```
*{

}
```




```
.sBox-1 {
  box-shadow: 10px 10px 20px 10px #000; /*Right Bottom Blur Spread Color */
  box-shadow: -20px -10px black; /*Left Top */
  box-shadow: 20px -10px black; /* Right Top */
  box-shadow: -20px 10px black; /* Left Bottom*/
}
.sBox-2 {
  box-shadow: 10px 10px 20px 10px rgba(255,0,0,0.5); /*   Right Bottom Blur Spread Color(rgba)*/
}
.sBox-3 {
  box-shadow: 10px 10px 20px 10px rgba(0,255,0,0.5), -5px -10px 5px cyan; /*  Multiple Shadows*/
}
```

Flexbox

Flexbox is short of Flexible Box Layout.

Flexbox is a layout model in CSS that provides an efficient way to arrange, align and distribute elements

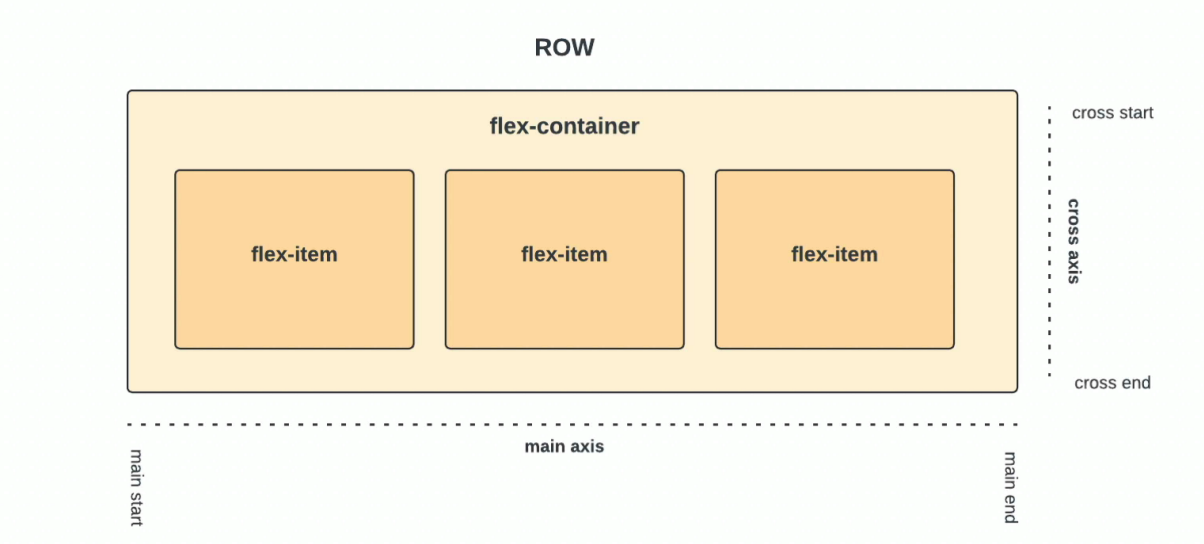
Great for dealing with dynamic or unknown sizes

One-dimensional layout model

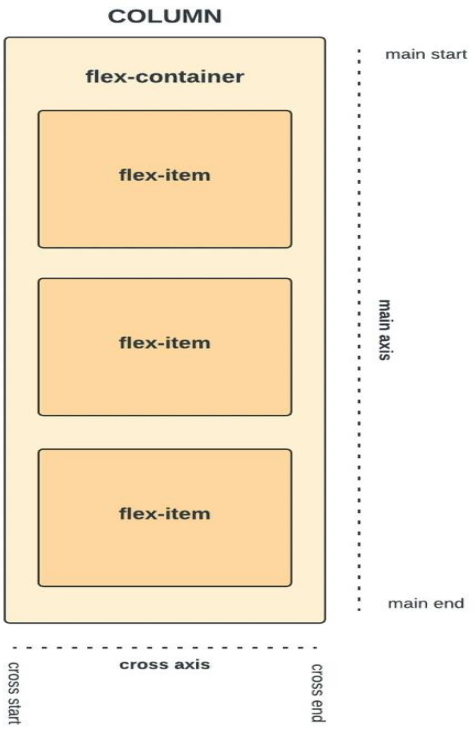
Flex items are put in a flex container and applied only to the direct children.

Flex container could be anything like, div or ul etc. Setting the display style flex, it will automatically apply the flex css on flex items like, P or li etc as shown in images.. In case of **Row** they will place horizontally. In Case of **Column** Vertically.

FlexBox Row(Default)



FlexBox Column




```
.container {
  display: flex; /* Place in the container folder, to apply flex property to direct children */
```

flex-direction: row; /* Default. Reverse will arrange in reverse order */

flex-wrap: wrap; /* Otherwise, the actual height and width will be changed. wrap will make these fixed. Reverse will arrange in reverse order */

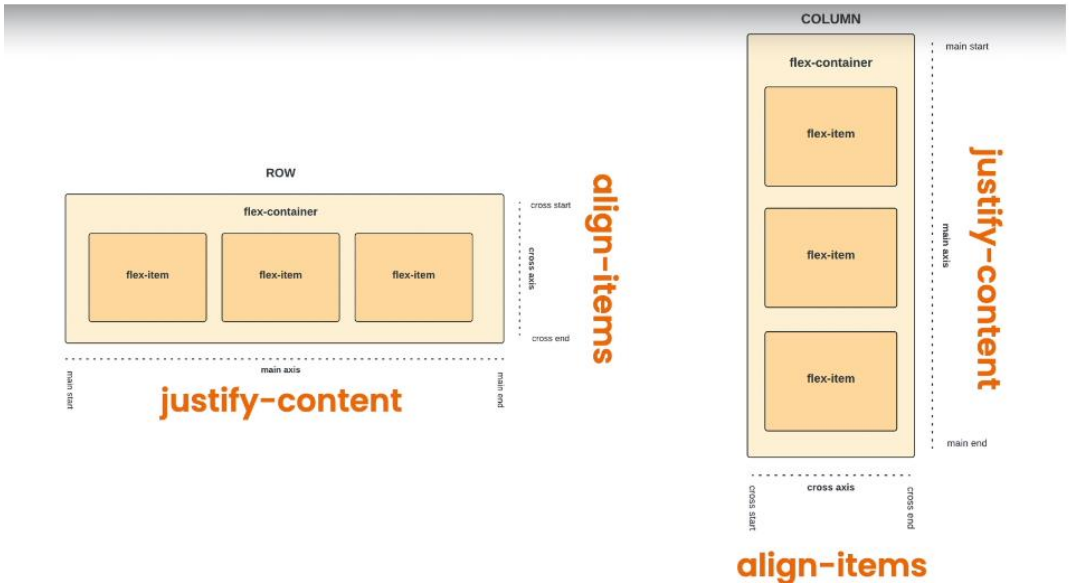
gap: 10px; /* Make row and column gaps between the children elements. row-gap and column-gap properties are also there, to make gaps for only rows and columns */

justify-content: flex-end;

align-content: center;

}

Align and Justify



justify-content and align-content are generally used in the container. In Item level, align-self is used.

Flex Properties and Dynamic Sizing

Items grows/shrinks as Container grows/shrinks

- flex-shrink: 1; Default value. Shrinks the items as the Container or browser shrink.
- flex-shrink: 0; This will never shrink.
- flex-shrink: 2; This will shrink at the double rate of other items
- flex-grow: 0; Default value. flex-grow will not work if width is set
- flex-grow: 1; It will grow, and will take up the remaining places. There should be no width set to grow to work
- flex-shrink: 0;
- flex-basis: 200px; will always grow upto 200px. Always be 200px. basis Specifies the initial size before any available space is distributed.
- flex: 1 0 300px; order Grow Shrink Basis (GSB) All together
- display: flex; Giving flex the flex items will come side by side in row automatically
- flex-direction: column; →By Default flex is row. Giving the value as column, it will arrange vertically. column-reverse or row-reverse will reverse the arrangement.

- flex-wrap: wrap; → The size of flex items is 200px. They are all wrapping into a single row/column. Causes the size of flex-items become smaller.
  wrap, will create multiple rows/columns, keeping the size same.
  wrap-reverse will do the same in reverse order.

- gap:10px; →this is the same thing as margin 10px in flex-item
- row-gap: 10px; → Only for Horizontal gap
- column-gap: 10px; → Only for Vertical gap
- align-items: stretch; → Default. Also need to remove the height of the container element.

These are item level flex properties.

These can be written as: flex: 1 0 200px; Grow, shrink and Basis in the order. Ideally no basis. So, commonly used as
flex: 1 0 0; This is again similar as
flex: 1;

Order

Flex Order

Item 2 - order: 1;	Item 1 - order: 2;	Item 3 - order: 3;	Item 6 - order: 4;	Item 5 - order: 5;	Item 4 - order: 6;

```
.ord-item-1 {
 order: 2;
}
.ord-item-2 {
 order: 1;
}
.ord-item-3 {
  order: 3;
}
.ord-item-4 {
  order: 6;
}
.ord-item-5 {
  order: 5;
}
.ord-item-6 {
  order: 4;
}
```

CSS Flexbox Layout Guide

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

Background

The Flexbox Layout (Flexible Box) module (a W3C Candidate Recommendation as of October 2017) aims at providing a more efficient way to lay out, align and distribute space among items in a container, even when their size is unknown and/or dynamic (thus the word “flex”).

The main idea behind the flex layout is to give the container the ability to alter its items’ width/height (and order) to best fill the available space (mostly to accommodate to all kind of display devices and screen sizes). A flex container expands items to fill available free space or shrinks them to prevent overflow.

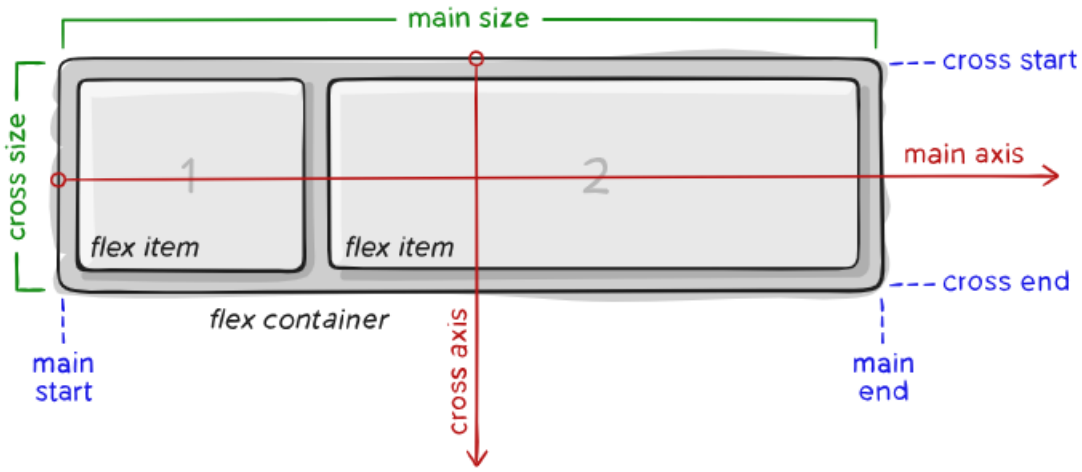
Most importantly, the flexbox layout is direction-agnostic as opposed to the regular layouts (block which is vertically-based and inline which is horizontally-based). While those work well for pages, they lack flexibility (no pun intended) to support large or complex applications (especially when it comes to orientation changing, resizing, stretching, shrinking, etc.).

Note: Flexbox layout is most appropriate to the components of an application, and small-scale layouts, while the Grid layout is intended for larger scale layouts.

Basics and terminology

Since flexbox is a whole module and not a single property, it involves a lot of things including its whole set of properties. Some of them are meant to be set on the container (parent element, known as "flex container") whereas the others are meant to be set on the children (said "flex items").

If "regular" layout is based on both block and inline flow directions, the flex layout is based on "flex-flow directions". Please have a look at this figure from the specification, explaining the main idea behind the flex layout.



Items will be laid out following either the `main axis` (from `main-start` to `main-end`) or the cross axis (from `cross-start` to `cross-end`).

main axis – The main axis of a flex container is the primary axis along which flex items are laid out. Beware, it is not necessarily horizontal; it depends on the `flex-direction` property (see below).

main-start | main-end – The flex items are placed within the container starting from main-start and going to main-end.

main size – A flex item's width or height, whichever is in the main dimension, is the item's main size. The flex item's main size property is either the 'width' or 'height' property, whichever is in the main dimension.

cross axis – The axis perpendicular to the main axis is called the cross axis. Its direction depends on the main axis direction.

cross-start | cross-end – Flex lines are filled with items and placed into the container starting on the cross-start side of the flex container and going toward the cross-end side.

cross size – The width or height of a flex item, whichever is in the cross dimension, is the item's cross size. The cross-size property is whichever of 'width' or 'height' that is in the cross dimension.

Flexbox properties

Properties for the Parent (flex container)

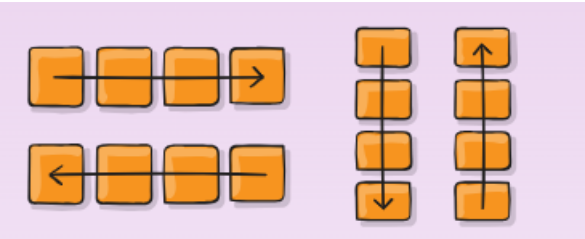
display

This defines a flex container; inline or block depending on the given value. It enables a flex context for all its direct children.

```
.container {
  display: flex; /* or inline-flex */
}
```

Note that CSS columns have no effect on a flex container.

flex-direction

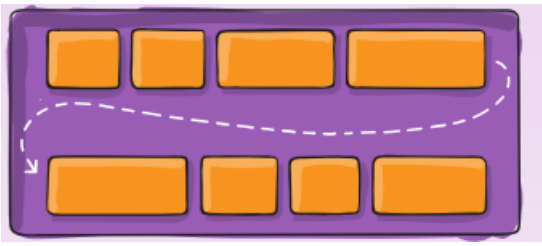


This establishes the main-axis, thus defining the direction flex items are placed in the flex container. Flexbox is (aside from optional wrapping) a single-direction layout concept. Think of flex items as primarily laying out either in horizontal rows or vertical columns.

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

- row (default): left to right in ltr; right to left in rtl
- row-reverse: right to left in ltr; left to right in rtl
- column: same as row but top to bottom
- column-reverse: same as row-reverse but bottom to top

flex-wrap



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

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

- nowrap (default): all flex items will be on one line
- wrap: flex items will wrap onto multiple lines, from top to bottom.
- wrap-reverse: flex items will wrap onto multiple lines from bottom to top.

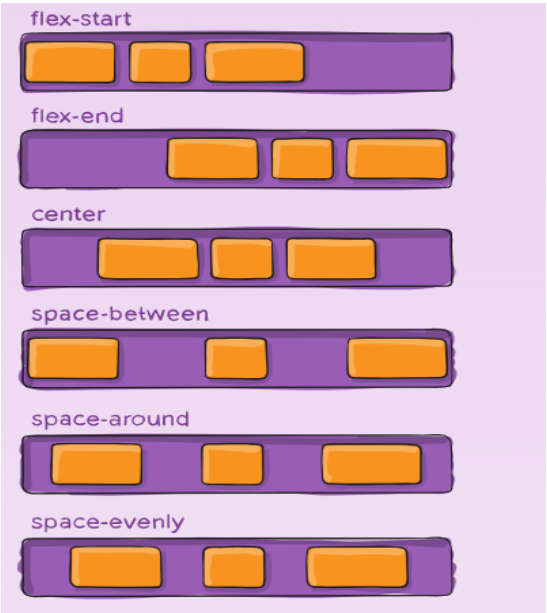
There are some visual demos of flex-wrap here.

flex-flow

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

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

justify-content



This defines the alignment along the main axis. It helps distribute extra free space leftover when either all the flex items on a line are inflexible, or are flexible but have reached their maximum size. It also exerts some control over the alignment of items when they overflow the line.

```
.container {
  justify-content: flex-start | flex-end | center | space-between |
space-around | space-evenly | start | end | left | right ... + safe
| unsafe;
}
```

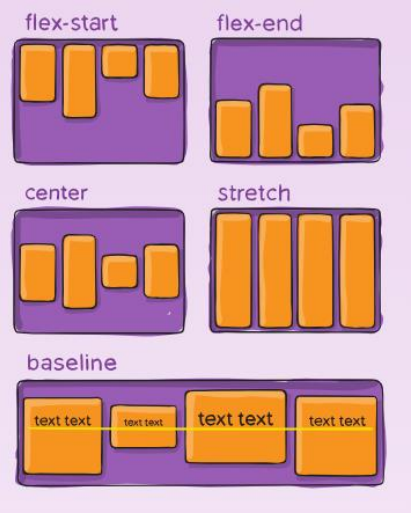
- flex-start (default): items are packed toward the start of the flex-direction.
- flex-end: items are packed toward the end of the flex-direction.
- start: items are packed toward the start of the writing-mode direction.
- end: items are packed toward the end of the writing-mode direction.
- left: items are packed toward left edge of the container, unless that doesn't make sense with the flex-direction, then it behaves like start.
- right: items are packed toward right edge of the container, unless that doesn't make sense with the flex-direction, then it behaves like end.
- center: items are centered along the line
- space-between: items are evenly distributed in the line; first item is on the start line, last item on the end line
- space-around: items are evenly distributed in the line with equal space around them. Note that visually the spaces aren't equal, since all the items have equal space on both sides. The first item will have one unit of space against the container edge, but two units of space between the next item because that next item has its own spacing that applies.
- space-evenly: items are distributed so that the spacing between any two items (and the space to the edges) is equal.

Note that that browser support for these values is nuanced. For example, space-between never got support from some versions of Edge, and start/end/left/right aren't in Chrome yet. MDN has detailed charts. The safest values are flex-start, flex-end, and center.

(https://developer.mozilla.org/en-US/docs/Web/CSS/justify-content)

There are also two additional keywords you can pair with these values: safe and unsafe. Using safe ensures that however you do this type of positioning, you can't push an element such that it renders off-screen (e.g. off the top) in such a way the content can't be scrolled too (called "data loss").

align-items



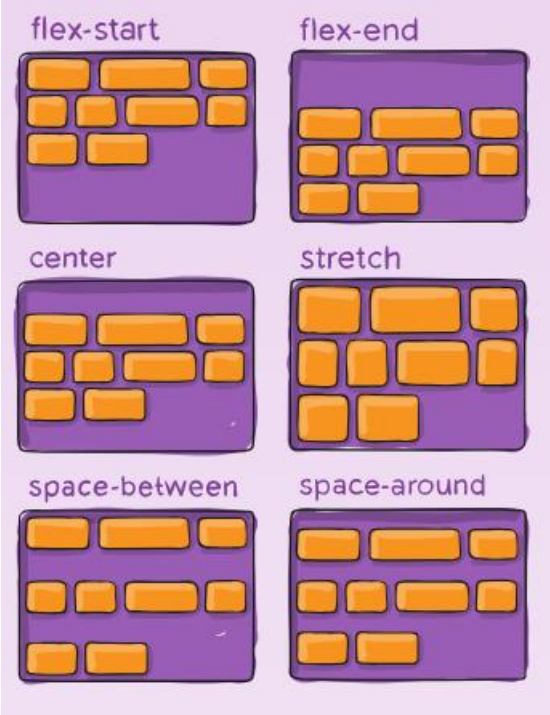
This defines the default behavior for how flex items are laid out along the **cross axis** on the current line. Think of it as the justify-content version for the cross-axis (perpendicular to the main-axis).

```
.container {
  align-items: stretch | flex-start | flex-end | center | baseline |
first baseline | last baseline | start | end | self-start | self-end
+ ... safe | unsafe;
}
```

- stretch (default): stretch to fill the container (still respect min-width/max-width)
- flex-start / start / self-start: items are placed at the start of the cross axis. The difference between these is subtle, and is about respecting the flex-direction rules or the writing-mode rules.
- flex-end / end / self-end: items are placed at the end of the cross axis. The difference again is subtle and is about respecting flex-direction rules vs. writing-mode rules.
- center: items are centered in the cross-axis
- baseline: items are aligned such as their baselines align

The safe and unsafe modifier keywords can be used in conjunction with all the rest of these keywords (although note browser support), and deal with helping you prevent aligning elements such that the content becomes inaccessible.

align-content



This aligns a flex container's lines within when there is extra space in the cross-axis, similar to how justify-content aligns individual items within the main-axis.

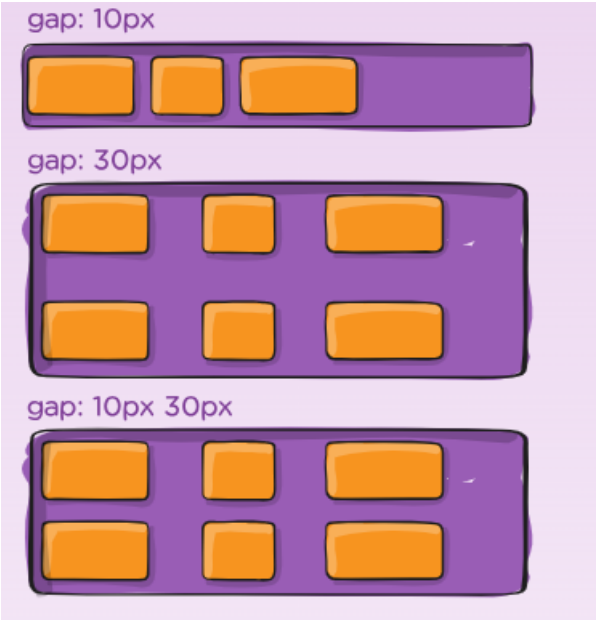
Note: This property only takes effect on multi-line flexible containers, where flex-wrap is set to either wrap or wrap-reverse). A single-line flexible container (i.e. where flex-wrap is set to its default value, no-wrap) will not reflect align-content.

```
.container {
  align-content: flex-start | flex-end | center | space-between |
space-around | space-evenly | stretch | start | end | baseline |
first baseline | last baseline + ... safe | unsafe;
}
```

- normal (default): items are packed in their default position as if no value was set.
- flex-start / start: items packed to the start of the container. The (more supported) flex-start honors the flex-direction while start honors the writing-mode direction.
- flex-end / end: items packed to the end of the container. The (more support) flex-end honors the flex-direction while end honors the writing-mode direction.
- center: items centered in the container
- space-between: items evenly distributed; the first line is at the start of the container while the last one is at the end
- space-around: items evenly distributed with equal space around each line
- space-evenly: items are evenly distributed with equal space around them
- stretch: lines stretch to take up the remaining space

The safe and unsafe modifier keywords can be used in conjunction with all the rest of these keywords (although note browser support), and deal with helping you prevent aligning elements such that the content becomes inaccessible.

gap, row-gap, column-gap



The gap property explicitly controls the space between flex items. It applies that spacing *only between items* not on the outer edges.

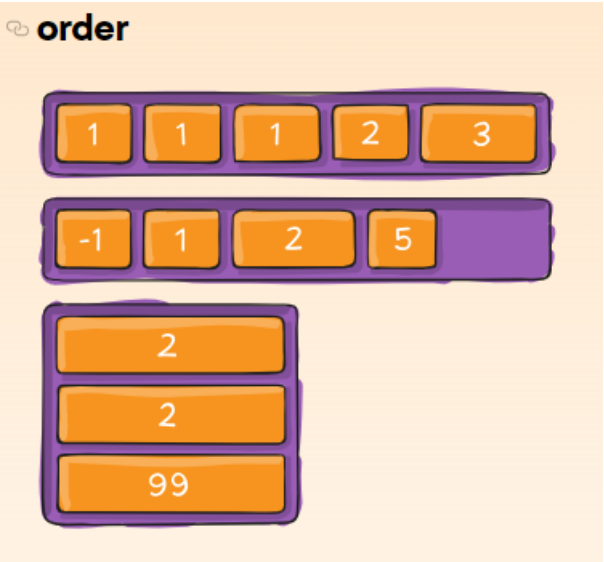
```
.container {
  display: flex;
  ...
  gap: 10px;
  gap: 10px 20px; /* row-gap column gap */
  row-gap: 10px;
  column-gap: 20px;
}
```

The behaviour could be thought of as a *minimum* gutter, as if the gutter is bigger somehow (because of something like justify-content: space-between;) then the gap will only take effect if that space would end up smaller.

It is not exclusively for flexbox, gap works in grid and multi-column layout as well.

Properties for the Children (flex items)

Order

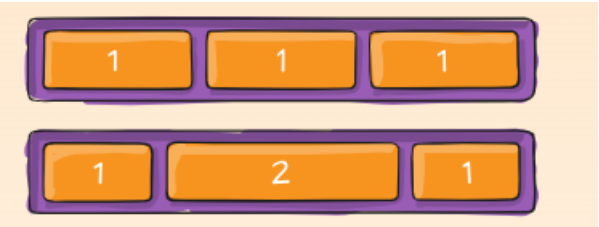


By default, flex items are laid out in the source order. However, the order property controls the order in which they appear in the flex container.

```
.item {
  order: 5; /* default is 0 */
}
```

Items with the same order revert to source order.

flex-grow



This defines the ability for a flex item to grow if necessary. It accepts a unitless value that serves as a proportion. It dictates what amount of the available space inside the flex container the item should take up.

If all items have flex-grow set to 1, the remaining space in the container will be distributed equally to all children. If one of the children has a value of 2, that child would take up twice as much of the space as either one of the others (or it will try, at least).

```
.item {
  flex-grow: 4; /* default 0 */
}
```

Negative numbers are invalid.

flex-shrink

This defines the ability for a flex item to shrink if necessary.

```
.item {
  flex-shrink: 3; /* default 1 */
}
```

Negative numbers are invalid.

flex-basis

This defines the default size of an element before the remaining space is distributed. It can be a length (e.g. 20%, 5rem, etc.) or a keyword. The auto keyword means “look at my width or height property” (which was temporarily done by the main-size keyword until deprecated). The content keyword means “size it based on the item’s content” – this keyword isn’t well supported yet, so it’s hard to test and harder to know what its brethren max-content, min-content, and fit-content do.

```
.item {
  flex-basis:  | auto; /* default auto */
}
```

If set to 0, the extra space around content isn’t factored in. If set to auto, the extra space is distributed based on its flex-grow value. See this graphic.

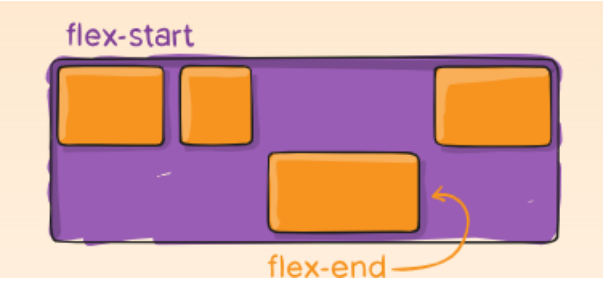
flex

This is the shorthand for flex-grow, flex-shrink and flex-basis combined. The second and third parameters (flex-shrink and flex-basis) are optional. The default is 0 1 auto, but if you set it with a single number value, like flex: 5;, that changes the flex-basis to 0%, so it’s like setting flex-grow: 5; flex-shrink: 1; flex-basis: 0%;.

```
.item {
  flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]
}
```

It is recommended that you use this shorthand property rather than set the individual properties. The shorthand sets the other values intelligently.

align-self



This allows the default alignment (or the one specified by align-items) to be overridden for individual flex items.

Please see the align-items explanation to understand the available values.

```
.item {
  align-self: auto | flex-start | flex-end | center | baseline |
stretch;
}
```

Note that float, clear and vertical-align have no effect on a flex item.

Prefixing Flexbox

Flexbox requires some vendor prefixing to support the most browsers possible. It doesn't just include prepending properties with the vendor prefix, but there are actually entirely different property and value names. This is because the Flexbox spec has changed over time, creating an "old", "tweener", and "new" versions.

Perhaps the best way to handle this is to write in the new (and final) syntax and run your CSS through Autoprefixer, which handles the fallbacks very well.

Alternatively, here's a Sass `@mixin` to help with some of the prefixing, which also gives you an idea of what kind of things need to be done:

```
@mixin flexbox() {
  display: -webkit-box;
  display: -moz-box;
  display: -ms-flexbox;
  display: -webkit-flex;
  display: flex;
}

@mixin flex($values) {
  -webkit-box-flex: $values;
  -moz-box-flex:  $values;
  -webkit-flex:  $values;
  -ms-flex:  $values;
```

```
  flex:  $values;
}

@mixin order($val) {
  -webkit-box-ordinal-group: $val;
  -moz-box-ordinal-group: $val;
  -ms-flex-order: $val;
  -webkit-order: $val;
  order: $val;
}

.wrapper {
  @include flexbox();
}

.item {
  @include flex(1 200px);
  @include order(2);
}
```

Examples

Let's start with a very simple example, solving an almost daily problem: perfect centering. It couldn't be any simpler if you use flexbox.

```
.parent {
  display: flex;
  height: 300px; /* Or whatever */
}


.child {

  width: 100px;  /* Or whatever */
  height: 100px; /* Or whatever */
  margin: auto;  /* Magic! */
}
```

This relies on the fact a margin set to auto in a flex container absorb extra space. So, setting a margin of auto will make the item perfectly centered in both axes.

Now let's use some more properties. Consider a list of 6 items, all with fixed dimensions, but can be auto-sized. We want them to be evenly distributed on the horizontal axis so that when we resize the browser, everything scales nicely, and without media queries.

```
.flex-container {
  /* We first create a flex layout context */
  display: flex;

  /* Then we define the flow direction
```

```
    and if we allow the items to wrap
   * Remember this is the same as:
   * flex-direction: row;
   * flex-wrap: wrap;
   */
  flex-flow: row wrap;

  /* Then we define how is distributed the remaining space */
  justify-content: space-around;
}
```

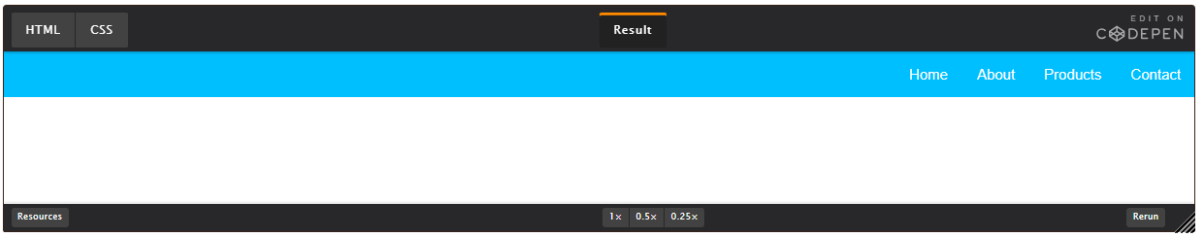
Done. Everything else is just some styling concern. Below is a pen featuring this example. Be sure to go to CodePen and try resizing your windows to see what happens.

Let's try something else. Imagine we have a right-aligned navigation element on the very top of our website, but we want it to be centered on medium-sized screens and single-columned on small devices. Easy enough.

```
/* Large */
.navigation {
  display: flex;
  flex-flow: row wrap;
  /* This aligns items to the end line on main-axis */
  justify-content: flex-end;
}

/* Medium screens */
@media all and (max-width: 800px) {
  .navigation {
    /* When on medium sized screens, we center it by evenly
distributing empty space around items */
    justify-content: space-around;
  }
}

/* Small screens */
@media all and (max-width: 500px) {
  .navigation {
    /* On small screens, we are no longer using row direction but
column */
    flex-direction: column;
  }
}
```



Let's try something even better by playing with flex items flexibility! What about a mobile-first 3-columns layout with full-width header and footer. And independent from source order.

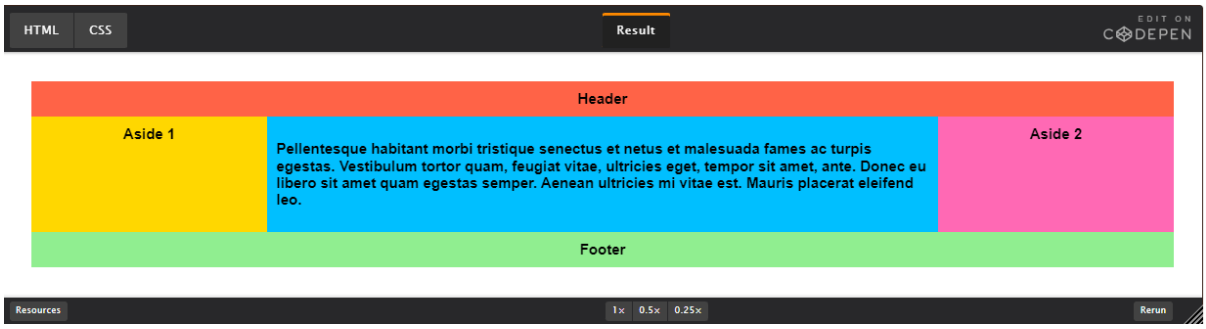
```
.wrapper {
  display: flex;
  flex-flow: row wrap;
}

/* We tell all items to be 100% width, via flex-basis */
.wrapper > * {
  flex: 1 100%;
}

/* We rely on source order for mobile-first approach
 * in this case:
 * 1. header
 * 2. article
 * 3. aside 1
 * 4. aside 2
 * 5. footer
 */

/* Medium screens */
@media all and (min-width: 600px) {
  /* We tell both sidebars to share a row */
  .aside { flex: 1 auto; }
}

/* Large screens */
@media all and (min-width: 800px) {
  /* We invert order of first sidebar and main
   * And tell the main element to take twice as much width as the
other two sidebars
   */
  .main { flex: 3 0px; }
  .aside-1 { order: 1; }
  .main    { order: 2; }
  .aside-2 { order: 3; }
  .footer  { order: 4; }
}
```



Responsive Design

When web pages & layouts respond and render well on a variety of screen sizes from smartphones to large TV monitors

- Improved User Experience
- Better Accessibility
- Enhanced SEO Performance
- Simplified Maintenance

Common Components of Responsive Design

- Flexible Layouts (max-width, percentages, etc)
- Correct Viewport Tag
- Flexible Images
- Fluid Typography
- Media Queries & Container Queries

width Max-width Min-width

The `width` property is used to dictate how wide an element will be when rendered by the browser. The element in question should be an image, `block` level element, or an `inline` element having `display` property of `block` or `inline-block`.

This `Min-width` property, as the name implies dictates the ***minimum*** *width* of a `block` level element (or an image) when rendered by the browser.

The `Max-width` property dictates the **maximum** width of a `block` level element (or image) when rendered on screen by the browser. This means **if a child element has a maximum width less than the parent's container it will occupy that specified width but if it has a maximum width equal to the width of the parent's container, it will occupy the entire parent's width.**

Similarities

All three properties are similar in the following ways:

- They dictate the width of an element.
- They take numbers and acceptable CSS units of measurement as their values.
- They work on `block` level elements or any elements having a **display property other than inline**. Image is an exception.

Differences

There are three main differences between these properties namely:

Sizing

width:

Fixed. Therefore, a child element having a width of 45%, 45em, or other values of the width property will be the same irrespective of the browser's viewport. At a lower viewport the width of the child element will be equal to the parent's container width. As the viewport gets smaller it can cause an overflow in the browser viewport or parent container.

min-width

The value of the `Min-width` property will allow the child element to occupy the entire `width` of its parent container if the value is large enough and the browser is at a larger viewport. At a lower viewport it will cause an overflow in the browser's viewport, but when the `min-width` value is smaller than the viewport then the `width` of the child element will be equal to the parent's `width`.

max-width

In most case, the child element having a `max-width` property will behave like the width property but when the browser's viewport is reduced you'll see the difference in size compared to the `width` and at a certain viewport size the width of the child element will be equal to the parent's `width`.

Media queries for Responsive Web Design

width:

Not good.

Min-width

Very Good

```
@media screen and (min-width: 48em) { /* Take note of the min-width property */
  /**
     * The code blocks within this media query
     * will only work for devices with at least
     * 48em. Assuming a default font size of 16px,
     * that will be 48 * 16 = 768px. 768px is the minimum
     * width of an iPad device (as of August 2020).
     */
}
```

Max-width

Good

```
@media screen and (max-width: 48em) { /* Take note of the max-width property */
  /**
     * The code blocks within this media query
     * will only work for devices with at most
     * 48em i.e. screen sizes of 768px and below.
     * Assuming a default font size of 16px.
     */
}
```

Responsive Images

width:

Good.

```
.image-selector {
  width: 100%;
}
```

Min-width

Not Good

The `min-width` property is a no-go area for Responsive Images because there is high possibility the image will exceed its parent container and sometimes it can result in a distorted image.

Max-width

Very Good (*highly recommended*)

```
.image-selector {
  max-width: 100%; /* Never exceed your parent container. */
}
```

Flexible Layout and percentages

```
.container {
width: 1100px;
margin: 50px auto;
padding:20px;
}
```

Here as long as the browser is 1100px or more then, it will be ok. But once it is less, the text will be truncated. So, the solution is

```
.container {
Max-width: 1100px;
margin: 50px auto;
padding:20px;
}
```

In this case, the text will not be truncated, and will be responsive.

Also, we can use

```
.container {
width: 80%;
margin: 50px auto;
padding:20px;
}
```

This will always take 80% of the view port. So, a value with % also can be used. But in this case also if the browser expands beyond a large level, the text will be expanding. So, the max-width also should be there. Also, margin a padding values giving in px can also malfunction on smaller device. So, the it will be,

```
.container {
width: 80%;   /* It will always be *0% of the viewport, high large it may be */
max-width: 1100px;  /* Then the 80% will be max 1100px, not beyond that.*/
margin: 10% auto;
padding: 20px;
}
```

so, for images, inside `.container` class, the above features will not be working. So, for that

```
img {
max-width: 100%;
}
```

Here all the images will be max-width 100%. And for the `.container` images wit will be **100% of the 80%.**

When Not to Use Percentages

- Fixed Sized Elements
- Font Sizes
- Precise Positioning
- Nested Percentage Values

Rem and em units

Absolute	**Relative**
Pixels (px)	Percentages(%)
Centimeters (cm)	em & rem
Millimeters (mm)	Viewport Height/ Width (vh/vw)
Inches (in)	Viewport Max (vmax)
Points (pt)	Viewport Min (vmin)
Picas (pc)	
Percentages (%)	

```
.container-1 {

max-width: 900px;
margin: 20px auto;
padding: 20px;
border:1px solid black;
}

.container-1 p {
font-size: 1rem;
}
```

By default, the font size of a html page is 16px . `font-size: 1rem;` will be 16px, i.e. 1 time of 16px, i.e. The **R**oot **E**le**M**ent

```
.container-1 {
max-width: 900px;
margin: 20px auto;
padding: 20px;
border: 1px solid black;
font-size:10px;
}

.container-1 p {
font-size: 1.5em; /*(10*1.5)=15px*/
}
```

em is the multiplier of the parent, not root. In this case parent of p is `.container`-1, and there the font size is 10px. So, the p of `.container`-1 will be 1.5 time of 10px.

rem is **recommended**. Instead using px , using **rem** is better. If the font size of the browser is changed to small, large etc, then using px as font size will not affect. But using **rem** the font will be changed as per browser settings**.**

rem and **em** can also use in padding, margin etc. Also, since **em** is depended on its parent, for nested elements, the font size will depend on the parent child relation.

ViewPort units (vh and vw → viewport height and weight)

The viewport is the viewing pane or the browser. This is divided by 100 imaginary boxes row wise and column wise.

Now 30vh means, it will take 30 of those boxes vertically ie. Height and 30vw means. it will take 30 of those boxes horizontally. On increasing or decreasing of the viewport, this will adjust automatically.

Case 1:

```
.container {
background: coral;
height: 100vh;
}
```

This will make the background color, only for the `container` class.

Case 2:

```
.container {
background: coral;
height: 100vh;
}
```

This will make the background color, for the whole view port.

Case 3:

```
.container {
background: coral;
height: 50vh;
}
```

This will make the background color, for the half of the view port.

Case 4:

```
.container {
background: coral;
height: 100vh;
width: 50vw;

}
```

This will make the background color, for the half of the view port vertically

etc.

This can also be applied to as to the fonts.

```
.container-3 h1 {
font-size:7vw;
}
```

Absolute Units

PX: Pixels (px) are considered absolute units, although they are relative to the DPI and resolution of the viewing device. But on the device itself, the PX unit is fixed and does not change based on any other element. Using PX can be problematic for responsive sites, but they are useful for maintaining consistent sizing for some elements. If you have elements that should not be resized, then using PX is a good choice.

Relative Units

EM: Relative to the parent element

REM: Relative to the root element (HTML tag)

%: Relative to the parent element

VW: Relative to the viewport's width

VH: Relative to the viewport's height

%, VW, and VH

While PX, EM, and REM are primarily used for font sizing, %, VW, and VH are mostly used for margins, padding, spacing, and widths/heights.

To reiterate, VH stands for "viewport height", which is the viewable screen's height. 100VH would represent 100% of the viewport's height, or the full height of the screen. And of course, VW stands for "viewport width", which is the viewable screen's width. 100VW would represent 100% of the viewport's width, or the full width of the screen. % reflects a percentage of the parent element's size, regardless of the viewport's size.

Let's look at some examples of where Elementor gives %, VW, and VH options.

Column Widths: If you edit the layout of an Elementor Column, you'll notice that there is only one width sizing unit available – %. Column widths only work well and responsively when using percentages, so no other option is given.

Margins: A section's margins can be specified either in PX or %. Using % is usually preferable to ensure the margins don't get larger than the content when scaling down for a mobile device for instance. By using a percentage of the width of the device, your margins will remain relative to the size of the content, which is almost always preferable.

Padding: A section's padding can be specified either in PX, EM, or %. As with margins, it is often preferable to use either EM or % so the padding remains relative as the size of the page scales.

Font Size: If you edit the typography of an element, such as a Heading, you'll see four choices: PX, EM, REM, and VH

Have you ever created a large heading and admired how great it looked on desktop, only to realize it was far too large on mobile? (Admission of guilt: I have, more than once).

The key to solving this elegantly is to use either EM, REM, or VW instead of PX. Which you choose is dependent upon your particular situation. I usually choose EM because I want the size to be relative to the Heading's parent. But if you prefer to have the size be relative to the root (HTML) size, then choose REM instead. Or, you could set it to be relative to the viewport's width (VW) if that works better for your case.

Note that you could also set specific font size PX values per device by using the Device Icons to specify a size for Desktop, Tablet, and Mobile. But that still places limits on responsiveness and accessibility, so keep that in mind if you choose PX.

More About VW and VH

Viewport units represent a percentage of the current browser viewport (current browser size). While similar to % units, there is a difference. Viewport units are calculated as a percentage of the browser's current viewport size. Percentage units on the other hand are calculated as a percentage of the parent element, which may be different than the viewport size.

Let's consider an example of a mobile screen viewport that is 480px x 800px.

- 1 VW = 1% of the viewport's width (or 4.8px)
- 50 VW = 50% of the viewport's width (or 240px)
- 1 VH = 1% of the viewport's height (or 8px)
- 50 VH = 50% of the viewport's height (or 400px)
- If the viewport size changes, the element's size changes respectively.

When Should You Use One Unit Over Another?

Ultimately, there isn't a perfect answer for this question. In general, it is often best to choose one of the relative units over PX so that your web page has the best chance of rendering a beautifully responsive design. Choose PX however, if you need to ensure that an element never resizes at any breakpoint and remains the same regardless of whether a user has chosen a different default size. PX units ensure consistent results even if that's not ideal.

EM is relative to the parent element's font size, so if you wish to scale the element's size based on its parent's size, use EM.

REM is relative to the root (HTML) font size, so if you wish to scale the element's size based on the root size, no matter what the parent size is, use REM. If you've used EM and are finding sizing issues due to lots of nested elements, REM will probably be the better choice.

VW is useful for creating full width elements (100%) that fill up the entire viewport's width. Of course, you can use any percentage of the viewport's width to achieve other goals, such as 50% for half the width, etc.

VH is useful for creating full height elements (100%) that fill up the entire viewport's height. Of course, you can use any percentage of the viewport's height to achieve other goals, such as 50% for half the height, etc.

% is similar to VW and VH but it is not a length that is relative to viewport's width or height. Instead, it is a percentage of the parent element's width or height. Percentage units are often useful to set the width of margins, as an example.

Elementor makes it easy to choose the option that is best suited for your design. Ultimately, it's your choice.

Media Query

```
@media screen and (max-width: 600px) {

  /* CSS rules . In this case the rules will apply, if the screen is 600px or less*/

}
```

@media rule defines different style rules

Media features - min/max-height, min/max-width, orientation (Portrait, Landscape etc)

Media types - screen, print, speech, etc

Typical Breakpoints

- 576px. Smartphones
- 768px. Tablets
- 992px. Desktop
- 1024px. Landscape
- 1200px. Desktop/Widescreen

Define certain CSS styles for certain screen sizes.

In the syntax screen is optional.

Can be start as

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

So,in this case we're saying that the css rules will be applied if the screen is less than 600 pixels less, because that's the max width in this case.

```
@media (max-width:1200px) {
/*in case of max-width:1200px it won't work beyond 1200px */
  .widescreen {
    background-color: coral;
  }
}
```

Here the background of `widescreen` class will be coral if the width of the screen is <=1200px

```
@media (min-width:1200px) {
/*in case of min-width:1200px it won't work for less than 1200px */
  .widescreen {
    background-color: coral;
  }
}
```

Here the background of `widescreen` class will be coral if the width of the screen is >=1200px

`max-width`: This is recommended for desktop

`min-width:` This is recommended for mobile, tab etc

```
@media (orientation:landscape) {
  .landcsape {
    background-color: lightgoldenrodyellow;
       }
}
```

This will work only if the orientation is landscape. To check, try landscape mode in developer mode for mobile or other devices.

```
@media (max-height:600px) {
  .height {
    background-color: lightgray;
  }
}
```

Here the background of `height` class will be coral if the height of the screen is <=600px

```
@media (min-width:768px) and (max-width:991px) {
body {
background: rgb(0,217,242);
background: linear-gradient(90deg, rgba(0,217,242,1) 33%, rgba(0,153,166,1) 71%);
  }
}
```

This is the use of multiple media. In this case the background of `body` will be darkblue if the width of the screen is <=991px and >=768

Media Query in Flexbox

If a container has the property,

```
flex-wrap: wrap;
```

then the items will stack onto each other after a certain height or weight, but the size of the items may grow or shrink. But doing the below

```
@media (max-width:768px) {
       .pricing {
              flex-direction:column;
       }
}
```

recommended

Or

```
@media (max-width:768px) {
       .pricing {
              /* flex-direction:column;*/
              display:block;
       }
}
```

then the items will stack onto each other after a weight<=768px, also the size of the items will not grow or shrink.

Also, some padding and margin can be added. These padding and margin will be work when the width of the screen will be in that given range.

```
@media (max-width:768px) {
.pricing {
flex-direction: column;
/*display: block;*/
}

.container {
margin: 2rem;
padding: 40px 30px;
  }
}
```

Container Query

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

@container rule defines different style rules.

@container-type property creates a container (inline-size, size). **Size is for the width and height and inline-size is for the width**

container-name is used to name containers.

- **@media:** Responsive to the viewport size or device characteristics.
- **@container:** Responsive to the size of a specific container element, enabling more granular control of layout based on the container's dimensions.

@container is a newer feature and not yet fully supported across all browsers, so using **@media** is still common for broad, viewport-based responsive design.

`@media`

- **Purpose**: Used for responsive design based on the viewport size or other device characteristics.
- **Functionality**: Applies styles conditionally based on the characteristics of the device or viewport, such as width, height, or orientation.
- **Usage**: Commonly used to adapt layouts for different screen sizes, such as mobile, tablet, and desktop.

Example:

```
@media (max-width: 600px) {
  .container {
    background-color: lightblue;
  }
}
```

In this example, the background color of the .container class will change to light blue when the viewport width is 600 pixels or less.

`@container`

- **Purpose**: Used for container queries, allowing styles to be applied based on the size of a container element rather than the viewport.

- **Functionality**: Enables responsive design based on the dimensions of a container element, rather than the entire viewport.

- **Usage**: Useful for component-level responsiveness where the container's size, rather than the screen size, dictates the layout changes.

Example:

```
.container {
  container-type: inline-size;
}

@container (max-width: 400px) {
  .item {
    font-size: 14px;
  }
}
```

In this example, the .item class will have a font size of 14 pixels when the container's width is 400 pixels or less.

	Media Queries	Container Queries
Viewport-based vs container-based	Apply styles based on the size of the viewport (the entire browser window)	Apply styles based on the size of an element's parent container
Modularity and Flexibility	Adjusting all components at once doesn't work as individual components may have different style needs	Creates adaptive components that can be reused in various parts of a website without unexpected changes
Complexity and Maintenance	Managing numerous media queries can become cumbersome, difficult-to-manage codebase	Context-aware components are easier to maintain, leading to a more organized codebase

```
main {
    container-type: inline-size;
    width: 100%;
    margin: 30px auto;
    padding: 20px;
}

    main p {
        background-color: lightgreen;
        color: indianred;
        text-shadow: 1px 1px 0.5px black;
        box-shadow: 1px 1px 0.5px black;
    }
```

```
aside {
    container-type: inline-size;
    container-name: aside;
    flex: 0 0 30%;
    margin: 30px auto;
    padding: 20px;
}
    aside p {
        background-color: #ccc;
        width: 100%;
    }

@container(max-width:992px) {
    main p {
        background-color: indianred;
        color: black;
        text-shadow: none;
        box-shadow: none;
        text-align: center;
    }
}

@container aside (max-width:300px) {
    aside p{
        background-color: #ff6a00;
    }
}
```

Container Query Units

VW, VH, CQW, and CQH are all units of measurement in Cascading Style Sheets (CSS) that are used to set the size of elements:

VW

A viewport-relative unit that's relative to the width of the viewport. VW is useful for creating elements that fill the entire width of the viewport.

VH

A viewport-relative unit that's relative to the height of the viewport. VH is useful for creating elements that fill the entire height of the viewport.

CQW

A container query unit that's relative to the width of a container. CQW is equal to 1% of the container's width.

CQH

A container query unit that's relative to the height of a container. CQH is equal to 1% of the container's height.

Section Intro

- Custom Properties
- Vendor Prefixes
- Filters
- Smooth Scrolling
- Sticky Navbar
- calc() Function
- Nesting

Custom Properties

```
scope {
--variable-name: variablevalue;
}

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

- Custom properties can be defined and used throughout your CSS
- Define properties by scope. Use :root for the root scope

```
.container { /*This is the scope. The variable or custom property --primary-color
will only work inside container class.*/
--primary-color: lightblue;
}

.box-1 {
background-color: var(--primary-root-color);
}

.box-2 {
background-color: var(--secondary-root-color);
}

.box-3 {
background-color: var(--tertiary-root-color);
}

body {
background-color: var(--primary-color); /*This will not work, as body or header
is outside container class.*/
}

header {
background-color: var(--primary-color);
/*This will not work, as body or header is outside container class.
*/
}

header {
background-color: var(--primary-root-color);
/*This will work. As it is from root.
*/

}
```

```
:root { /*This is accessible everywhere. Generally, place after *{}  */
--primary-root-color: red;
--secondary-root-color: yellow;
--tertiary-root-color: pink;
--shadow-Opacity: 0.5;

}

.box {
flex: 1;
padding: 20px;
border-radius: 10px;
box-shadow: 0 0 10px rgba(0,0,0,var(--shadow-Opacity));

}
```

Vendor Prefixes

Prefixes add support for new experimental features in certain browsers

-webkit- Chrome, Safari, newer versions of Opera & Edge

-moz- Firefox

-o- Old pre-Webkit versions of Opera

-ms- Internet Explorer & Edge before Chromium 15

Custom Properties

```
selector {
-webkit-transition: all 4s ease;
-moz-transition: all 4s ease;
-ms-transition: all 4s ease;
-o-transition: all 4s ease;
transition: all 4s ease;
}
```

https://caniuse.com/

Filters

```
img {
max-width: 50%;
max-height: 50%;
margin: auto 70px;

-webkit-border-radius: 5px;
-moz-border-radius: 5px;
-o-border-radius: 5px;
-ms-border-radius: 5px;
border-radius: 5px;

/*For Filter Any one will be working.*/

filter: grayscale(100%);  /*Converted to black and white 100% is full black and
white*/
filter: blur(3px);  /*Converted to blur.*/
filter: brightness(50%);  /*100% is default.*/
filter: contrast(500%);  /*100% is default.*/
```

```
filter: drop-shadow(10px 10px 5px red);
filter:hue-rotate(90deg);
filter: invert(100%);  /*Color Inversion*/
filter: opacity(50%);  /*Transparency. 0 is min transparent. 100 is max
transparent*/
filter: saturate(200%); /* Color Saturation.100% is default*/
filter: sepia(100%);  /*Color to brownish. higher than 100. Higher the color
fades to brownish*/

}
img {

/*For multiple filters to work simultaneously*/
filter: grayscale(100%) blur(3px);


/*reset filters*/
filter: none;
}
```

Smooth Scrolling

```
html {
  scroll-behavior: smooth;
}

/*For same page smooth scrolling*/
```

Sticky Nav

```
.header {
position: sticky;
top:0;
z-index:1000;
}
```

Transparency

First Late us define a css class.

```
.header.transparent {
background: linear-gradient(45deg,rgba(255,0,0,0.5),rgba(0,0,255,0.5));
}
```

Now let us define a javascript

```
document.addEventListener('DOMContentLoaded', function () {
  const header = document.querySelector('.header');
      //console.log(header);
  function toggleHeaderTrans() {

      //console.log(123);
      //console.log(window.scrollY);

    if (window.scrollY > 0) {
        header.classList.add('transparent');
    }
    else {
        header.classList.remove('transparent');
    }
  }
```

```
  window.addEventListener('scroll', toggleHeaderTrans);
});
```

Overview

This code listens for two main events on a webpage:

- The page's **Document Object Model (DOM)** fully loading.
- The user scrolling down the page.

When the DOM loads, the code selects the header element and sets it up to respond to scroll events. When the user scrolls down, a new CSS class, `.transparent`, is added to the `.header` to make it transparent. If the user scrolls back up to the top, the `.transparent` class is removed.

DOMContentLoaded Event

- **What is the DOM?**
  - The **DOM** (Document Object Model) represents the structure of an HTML document in a tree-like format. All HTML elements are organized in this "tree," which JavaScript can interact with.

- **Listening for the DOM to Load**
  - We need to make sure that JavaScript interacts with elements only after the DOM has fully loaded.
  - To do this, we use `document.addEventListener` (`'DOMContentLoaded'`, ...).
  - `'DOMContentLoaded'` is an event that triggers when the HTML document has been completely loaded and parsed (but before stylesheets and images are fully loaded).

Selecting the Header Element

- **Selecting Elements in JavaScript**
- We use `document.querySelector` () to select elements. Here, `querySelector` takes a CSS selector (like a class name, ID, or tag) and returns the first matching element.
- We store this selected header in a variable called `header` for easy access.

Adding Scroll Event Listener

- **Listening for the Scroll Event**
- We want to detect when the user scrolls the page. To do this, we set up another event listener on `window` for the `scroll` event.
- Every time the user scrolls, the `toggleHeaderTrans()` function will run.

Creating the toggleHeaderTrans Function

- **Function Purpose**
  - `toggleHeaderTrans` checks the vertical scroll position (Y-axis) of the window.
  - If the user has scrolled down (scrollY > 0), it adds the `.transparent` class to `header`.
  - If the user scrolls back to the top (scrollY = 0), it removes the `.transparent` class from `header`.

- **Adding or Removing a Class**
  - We use `header.classList.add('transparent');` to add the .transparent class.
  - To remove it, we use `header.classList.remove('transparent');`

```
document.addEventListener('DOMContentLoaded', function () {
  // Select the header element when the DOM is fully loaded
  const header = document.querySelector('.header');
  console.log(header); // Logs the header element for debugging

  // Function to add/remove the 'transparent' class on scroll
  function toggleHeaderTrans() {
    console.log(123);           // Logs 123 while scrolling
    console.log(window.scrollY); // Logs current scroll position

    if (window.scrollY > 0) {
        header.classList.add('transparent'); // Add class if scrolled down
    } else {
        header.classList.remove('transparent'); // Remove class if at the top
    }
  }

  // Listen for the scroll event and run toggleHeaderTrans when it occurs
  window.addEventListener('scroll', toggleHeaderTrans);
});
```

calc() Function

```
.container {
width: calc(500px 100px) /* 400px */
}
```

- Perform simple operations within CSS
- Useful for responsive design or applying dynamic styles

Example

```
:root {
--container-width: 1000px;
}

.section {
width: calc(500px 100px); /* 400px */
width: calc(50% + 100px);
width: calc(50% + 3 rem);
width: calc(var(--container-width) / 2); /*  will work.*/
min-width: calc(calc(100%/5) - 20px);
}
```

Nesting (Not Required)

No Nesting

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

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

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

Nesting

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


  h1{
  font-size: 4rem;
  }

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

}
```

GIT

Version Control

System that tracks changes to files over time, allowing multiple people to collaborate on a project by managing different versions of those files

- Keep track of files & folders
- Collaborate with other developers
- Track who did what
- Backup your code

What Is Git?

Most popular VCS (version control system). Track, share, collaborate and push to remote repositories (GitHub, GitLab, etc).

Must be installed on your machine. You can use the CLI, GUI tools as well as editor extensions.

Many hosting services such as Netlify, Vercel and Render allow you to pull from your Git repository to deploy to their servers.

Git Config

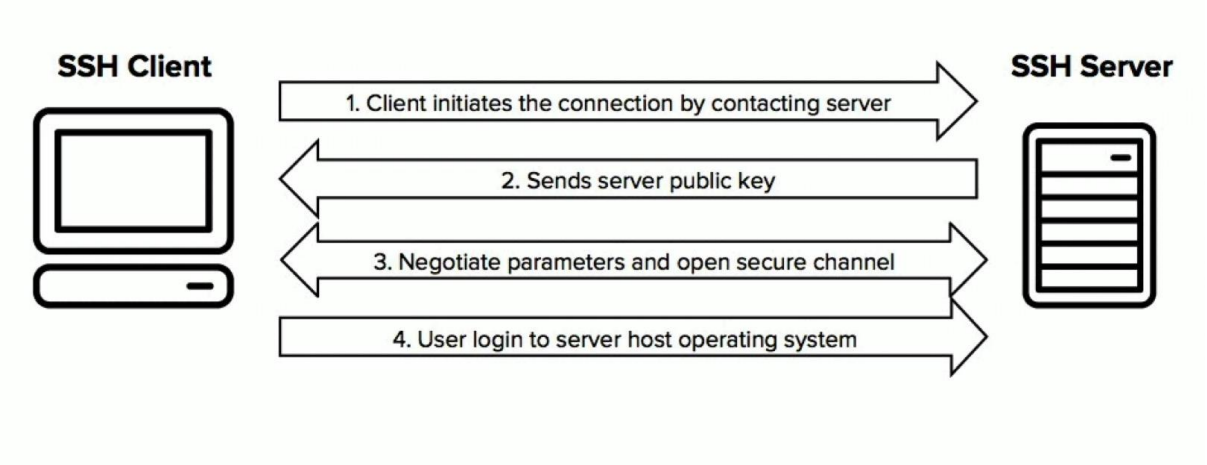
```
git config --global user.name "Swapnadip Saha"

git config --global user.email "swapnadip123.saha@gmail.com"
```

What Is SSH?

SSH (Secure Shell) is a cryptographic network protocol used to securely connect to and communicate with remote computers over an unsecured network, such as the internet.

You can generate SSH keys and use them to interact with your GitHub account without having to enter a password.



Generate SSH Keys

```
ssh-keygen -t ed25519 -C swapnadip123.saha@gmail.com
```

This will generate a private and public key file. You add the public key to Github via the settings.

To see the content

```
more <file_name>
```

Copy the key with email i.e. the full content. Go to Github→settings→SSH and GPG Keys→New SSH Keys→ Add the key

Git Workflow

Initialize a local repo	git init
Write some code, make changes, etc	
Stage your changes	git add or (git add .)to add all contents
Commit to local repo	git commit -m "Initial commit"
Push to remote repo	git push
To check status	git status

Then run the 3 commands provided in the Github page

```
git remote add origin https://github.com/SwapnadipSaha/Module-1.git
git branch -M main
git push -u origin main
```

Deploy website to Netlify

https://www.netlify.com/

login with github