CSS

Table of Content :-)

CSS (Styling)

Overview

- Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.
- CSS handles the look and feel part of a web page, using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colours are used, as well as a variety of other effects.
- CSS is easy to learn and understand but it provides a powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup language HTML or XHTML.

Syntax

Introduction

{···} A CSS comprises of style rules that are interpreted by the browser and then applied to the corresponding elements in your document. A style rule is made of three parts:

- Selector: A selector is an HTML tag at which a style will be applied. This could be any tag like <h1> or <table> etc.
- Property: A property is a type of attribute of HTML tag. Put simply, all the HTML attributes are converted into CSS properties. They could be color, border, etc.
- Value: Values are assigned to properties. For example, color property can have the value either red or #F1F1F1 etc.

You can put CSS Style Rule Syntax as follows:

```
selector { property: value }
```

Example: You can define a table border as follows:

```
table{ border :1px solid #C00; }
```

Here table is a selector and border is a property and the given value 1px solid #C00 is the value of that property.

You can define selectors in various simple ways based on your comfort. Let me put these selectors one by one.

The Type Selectors

{··} This is the same selector we have seen above. Again, one more example to give a color to all level 1 headings:

```
h1 {
color: #36CFFF;
}
```

The Universal Selectors

{··} Rather than selecting elements of a specific type, the universal selector quite simply matches the name of any element type:

```
* {
color: #000000;
```

```
}
```

This rule renders the content of every element in our document in black.

The Descendant Selectors

{··} Suppose you want to apply a style rule to a particular element only when it lies inside a particular element. As given in the following example, the style rule will apply to element only when it lies inside the <ul> tag.

```
ul em {
color: #000000;
}
```

The Class Selectors

{··} You can define style rules based on the class attribute of the elements. All the elements having that class will be formatted according to the defined rule.

```
.black {
color: #000000;
}
```

This rule renders the content in black for every element with class attribute set to black in our document. You can make it a bit more particular. For example:

```
h1.black {
color: #000000;
}
```

This rule renders the content in black for only <h1> elements with class attribute set to black.

You can apply more than one class selectors to a given element. Consider the following example:

```
<p class="center bold">
This para will be styled by the classes center and bold.
</p>
```

The ID Selectors

{··} You can define style rules based on the id attribute of the elements. All the elements having that id will be formatted according to the defined rule.

```
#black {
color: #000000;
}
```

This rule renders the content in black for every element with id attribute set to black in our document. You can make it a bit more particular. For example:

```
 h1#black {
color: #000000;
}
```

This rule renders the content in black for only <h1> elements with id attribute set to black.

The true power of id selectors is when they are used as the foundation for descendant selectors. For example:

```
#black h2 {
color: #000000;
}
```

In this example, all level 2 headings will be displayed in black color when those headings will lie within tags having id attribute set to black.

The Child Selectors

{··} You have seen the descendant selectors. There is one more type of selector, which is very similar to descendants but have different functionality. Consider the following example:

```
body > p {
color: #000000;
}
```

This rule will render all the paragraphs in black if they are a direct child of the <body> element. Other paragraphs put inside other elements like or <td> would not have any effect of this rule.

The Attribute Selectors

{··} You can also apply styles to HTML elements with particular attributes. The style rule below will match all the input elements having a type attribute with a value of text:

```
input[type="text"]{
  color: #000000;
}
```

The advantage to this method is that the <input type="submit" /> element is unaffected, and the color applied only to the desired text fields.

There are following rules applied to attribute selector.

- p[lang] - Selects all paragraph elements with a lang attribute.
- p[lang="fr"] - Selects all paragraph elements whose lang attribute has a value of exactly "fr".
- p[lang~="fr"] - Selects all paragraph elements whose lang attribute contains the word "fr".
- p[lang|="en"] - Selects all paragraph elements whose lang attribute contains values that are exactly "en", or begin with "en-".

Multiple Style Rules

{··} You may need to define multiple style rules for a single element. You can define these rules to combine multiple properties and corresponding values into a single block as defined in the following example:

```
h1 {
color: #36C;
font-weight: normal;
letter-spacing: .4em;
margin-bottom: 1em;
text-transform: lowercase;
}
```

Here all the property and value pairs are separated by a semicolon (;). You can keep them in a single line or multiple lines. For better readability, we keep them in separate lines.

For a while, don't bother about the properties mentioned in the above block. These properties will be explained in the coming chapters and you can find the complete detail about properties in CSS References.

Grouping Selectors

{··} You can apply a style to many selectors if you like. Just separate the selectors with a comma, as given in the following example:

```
h1, h2, h3 {
color: #36C;
font-weight: normal;
letter-spacing: .4em;
margin-bottom: 1em;
text-transform: lowercase;
}
```

This define style rule will be applicable to h1, h2 and h3 element as well. The order of the list is irrelevant. All the elements in the selector will have the corresponding declarations applied to them.

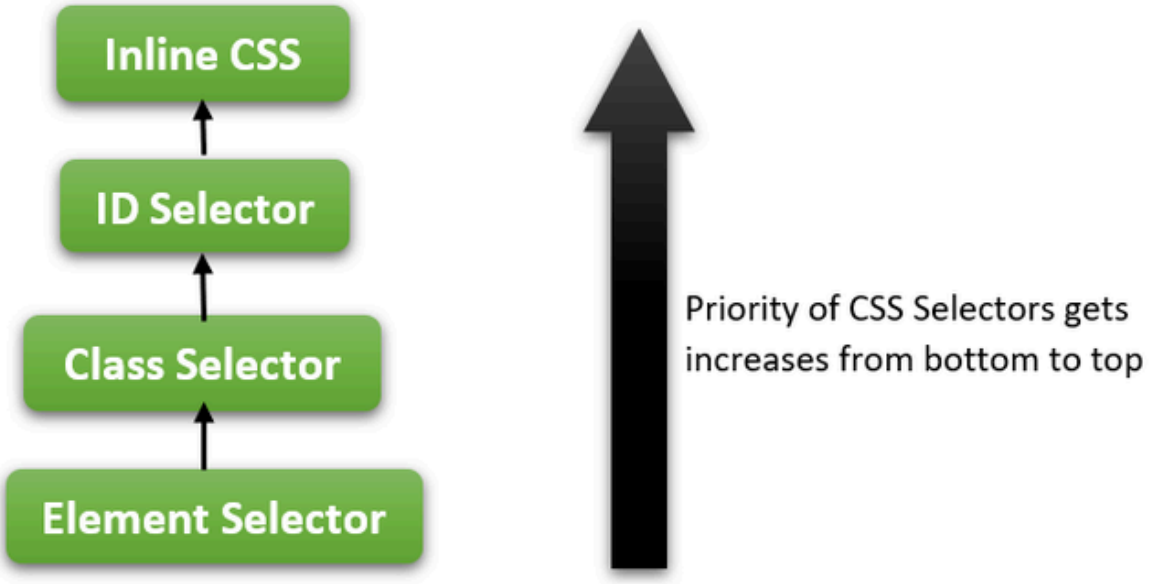
You can combine the various class selectors together as shown below:

```
#content, #footer, #supplement {
position: absolute;
left: 510px;
```

```
width: 200px;
}
```

Selector’s priority & Combining Selectors

Selector’s priority :-



Combining selectors :-

WITH SPACE

```
<!--HTML CODE-->
<div class= "container">
<h1 class= "title"> Hello World! </h1>
</div>
```

```
/*CSS CODE*/
   /*with space*/
.container .title{
color:red;
}
```

📌 This is telling the browser that the element with class called tilte in the div element with class called container, make it red.

WITH NO SPACE

```
<!--HTML CODE-->
<div class= "container">
<h1 class= "title"> Hello World! </h1>
</div>
```

```
/*CSS CODE*/
 /*with no space*/
.container.title{
color:red;
}
```

📌 This is telling the browser to look for an element that has a class called container but also a class of title and to make both red.

CONCLUSION

💡 Use id's sparingly, most use id's for sections and sections use only. classes for most of the element's div etc. While using classes only use one/same class two different element's and avoid using inline class, it's really bad habit and also one of the laziest.

Inclusion

Introduction

There are three ways to associate styles with your html document. Most commonly used methods are Inline css and External css.

Embedded CSS - The <style> Element

💡 You can put your CSS rules into an HTML document using the <style> element. This tag is placed inside the <head>...</head> tags. Rules defined using this syntax will be applied to all the elements available in the document. Here is the generic syntax:

```
<head>
<style>
Style Rules
.....
</style>
</head>
```

Inline CSS - The style Attribute

You can use style attribute of any HTML element to define style rules. These rules will be applied to that element only. Here is the generic syntax:

```
<element style="...style rules....">
```

```
Example:-
<h1 style= "color:red;">
...This is inline css...
</h1>
```

External CSS - The <link> Element

The <link> element can be used to include an external stylesheet file in your HTML document. An external style sheet is a separate text file with .css extension. You define all the Style rules within this text file and then you can include this file in any HTML document using <link> element.

Here is the generic syntax of including external CSS file:

```
<head>
<link type="text/css" href="..." media="..." />
</head>
```

CSS Rules Overriding

We have discussed three ways to include style sheet rules in an HTML document. Here is the rule to override any Style Sheet Rule.

- Any inline style sheet takes the highest priority. So, it will override any rule defined in <style>...</style> tags or the rules defined in any external style sheet file.
- Any rule defined in <style>...</style> tags will override the rules defined in any external style sheet file.
- Any rule defined in the external style sheet file takes the lowest priority, and the rules defined in this file will be applied only when the above two rules are not applicable.

CSS Comments

💡 Many times, you may need to put additional comments in your style sheet blocks. So, it is very easy to comment any part in the style sheet. You can simply put your comments inside CSS as follow :-

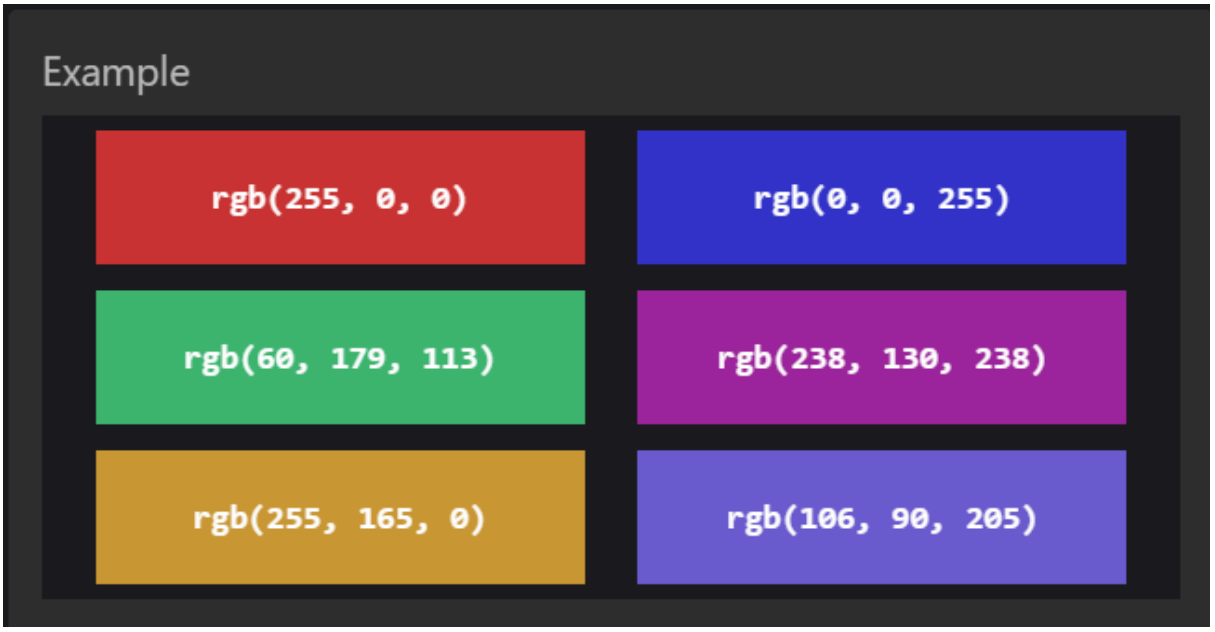
```
/* This is a comment in style sheet */
```

Colors in CSS

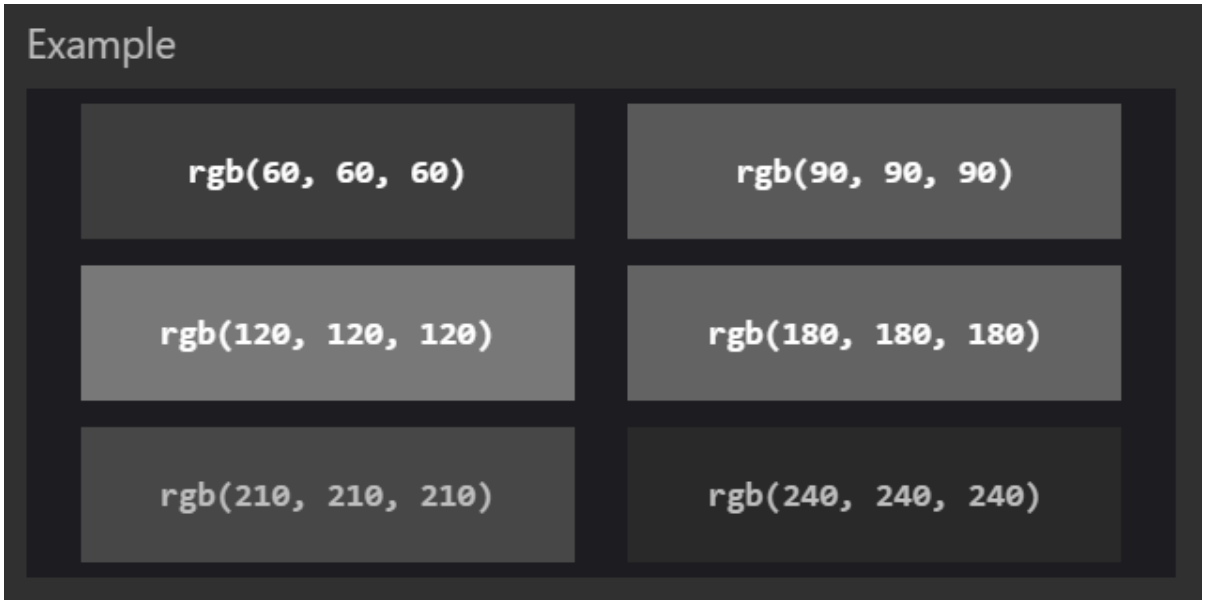
RGB Value

💡 An RGB color value represents RED, GREEN, and BLUE light sources.

- In CSS, a color can be specified as an RGB value, using this formula: **rgb(*red, green, blue*)**
- Each parameter (red, green, and blue) defines the intensity of the color between 0 and 255.
- For example, rgb(255, 0, 0) is displayed as red, because red is set to its highest value (255) and the others are set to 0.
- To display black, set all color parameters to 0, like this: rgb(0, 0, 0).
- To display white, set all color parameters to 255, like this: rgb(255, 255, 255).

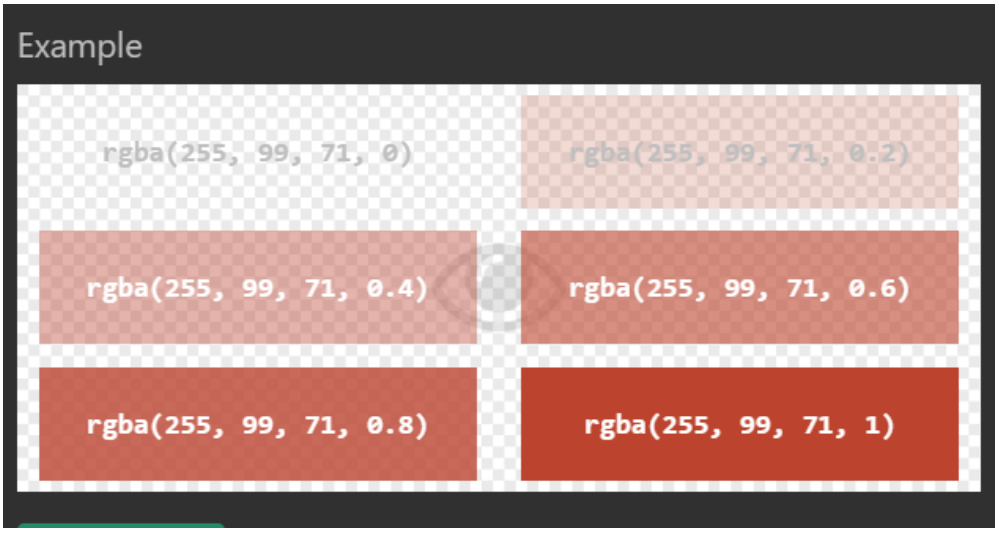


- Shades of gray are often defined using equal values for all the 3 light sources:



RGBA Value

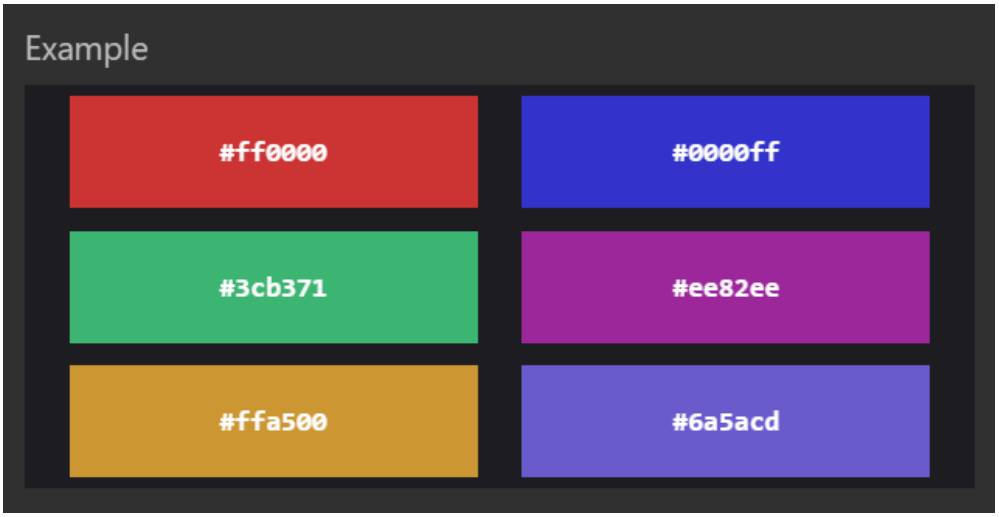
- RGBA color values are an extension of RGB color values with an alpha channel - which specifies the opacity for a color.
- An RGBA color value is specified with: **rgba(*red*, *green*, *blue*, *alpha*)**
- The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all):



HEX Value

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

- In CSS, a color can be specified using a hexadecimal value in the form: ***#rrggbb***
- Where rr (red), gg (green) and bb (blue) are hexadecimal values between 00 and ff (same as decimal 0-255).
- For example, #ff0000 is displayed as red, because red is set to its highest value (ff) and the others are set to the lowest value (00).
- To display black, set all values to 00, like this: #000000.
- To display white, set all values to ff, like this: #ffffff.



- Shades of gray are often defined using equal values for all the 3 light sources:

3 Digit HEX Value

- Sometimes you will see a 3-digit hex code in the CSS source.
- The 3-digit hex code is a shorthand for some 6-digit hex codes.
- The 3-digit hex code has the following form: ***#rgb***
- Where r, g, and b represent the red, green, and blue components with values between 0 and f.
- The 3-digit hex code can only be used when both the values (RR, GG, and BB) are the same for each component. So, if we have #ff00cc, it can be written like this: #f0c.

Here is an example:

```
body {
  background-color: #fc9; /* same as #ffcc99 */
}

h1 {
  color: #f0f; /* same as #ff00ff */
}

p {
  color: #b58; /* same as #bb5588 */
}
```

HSL Value

HSL stands for hue, saturation, and lightness.

- In CSS, a color can be specified using hue, saturation, and lightness (HSL) in the form: **hsl(*hue*, *saturation*, *lightness*)**
- Hue is a degree on the color wheel from 0 to 360. 0 is red, 120 is green, and 240 is blue.
- Saturation is a percentage value. 0% means a shade of gray, and 100% is the full color.
- Lightness is also a percentage. 0% is black, 50% is neither light or dark, 100% is white

Saturation

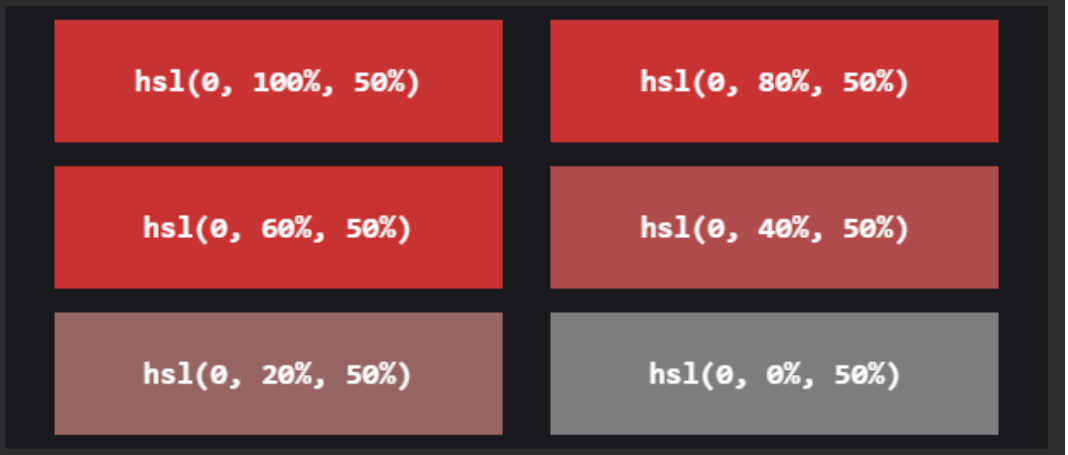
Saturation can be described as the intensity of a color.

100% is pure color, no shades of gray.

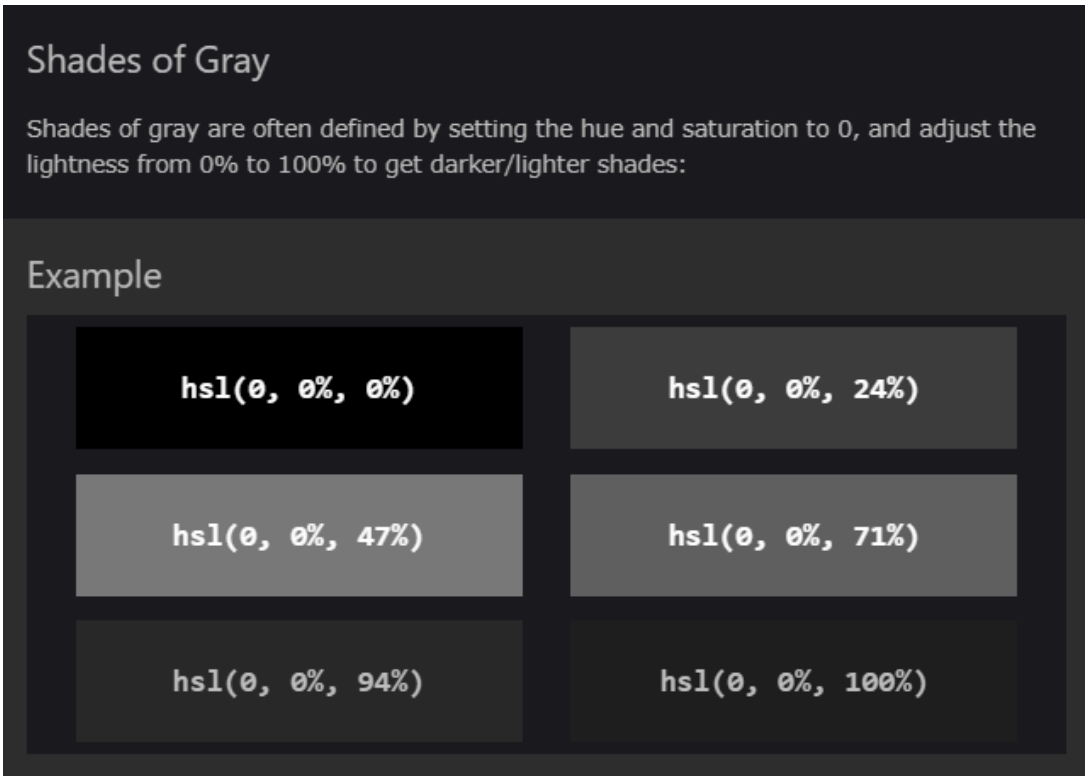
50% is 50% gray, but you can still see the color.

0% is completely gray; you can no longer see the color.

Example

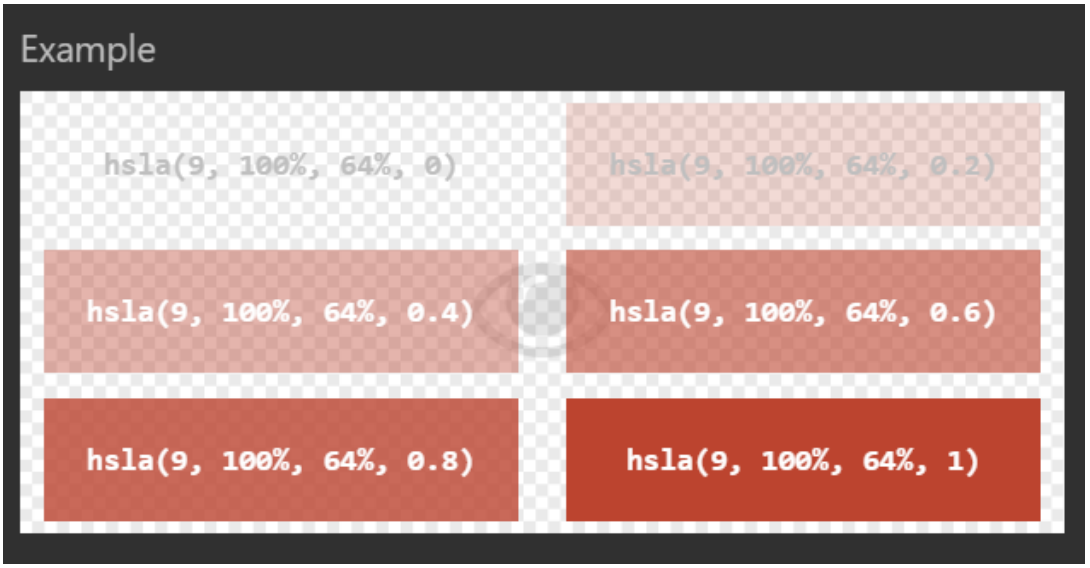



Lightness
The lightness of a color can be described as how much light you want to give the color, where 0% means no light (black), 50% means 50% light (neither dark nor light) and 100% means full lightness (white).
Example
hsl(0, 100%, 0%)
hsl(0, 100%, 25%)
hsl(0, 100%, 50%)
hsl(0, 100%, 75%)
hsl(0, 100%, 90%)
hsl(0, 100%, 100%)


Shades of Gray
Shades of gray are often defined by setting the hue and saturation to 0, and adjust the lightness from 0% to 100% to get darker/lighter shades:
Example
hsl(0, 0%, 0%)
hsl(0, 0%, 24%)
hsl(0, 0%, 47%)
hsl(0, 0%, 71%)
hsl(0, 0%, 94%)
hsl(0, 0%, 100%)

HSLA Value

- HSLA color values are an extension of HSL color values with an alpha channel - which specifies the opacity for a color.
- An HSLA color value is specified with: **hsla(*hue, saturation, lightness, alpha*)**
- The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all):



CSS Background Property

The CSS background properties are used to add background effects for elements.

CSS background-color

The `background-color` property specifies the background color of an element.

```
body {
  background-color: lightblue;
}
```

With CSS, a color is most often specified by:

- a valid color name - like "red"
- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)"

CSS background-image

The `background-image` property specifies an image to use as the background of an element.

By default, the image is repeated x & y direction, so it covers the entire element.

```
body {
  background-image: url("paper.gif");
}
```

CSS background-repeat

By default, the `background-image` property repeats an image both horizontally and vertically.

Some images should be repeated only horizontally or vertically, or they will look strange, like this:

```
body {
  background-image: url("gradient_bg.png");
}
```

If the image above is repeated only horizontally (`background-repeat: repeat-x;`), the background will look better:

```
body {
  background-image: url("gradient_bg.png");
  background-repeat: repeat-x;
}
```

Tip :- To repeat an image vertically, set `background-repeat: repeat-y;`

CSS background-repeat: no-repeat

Showing the background image only once is also specified by the `background-repeat` property:

```
/*Show the background image only once:*/
body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
}
```

In the example above, the background image is placed in the same place as the text. We want to change the position of the image, so that it does not disturb the text too much.

CSS background-position

The `background-position` property is used to specify the position of the background image.

```
body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
}
```

CSS background-attachment

The `background-attachment` property specifies whether the background image should scroll or be fixed (will not scroll with the rest of the page):

```
body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
  background-attachment: fixed;
}
```

```
/*Specify that the background image should scroll with the rest of the page:*/
body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
  background-attachment: scroll;
}
```

CSS background - Shorthand property

To shorten the code, it is also possible to specify all the background properties in one single property. This is called a shorthand property.

```
/*Instead of writing:*/
body {
  background-color: #ffffff;
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
```

```
  background-position: right top;
}
```

```
/*You can use the shorthand property background:*/
body {
  background: #ffffff url("img_tree.png") no-repeat right top;
}
```

When using the shorthand property the order of the property values is:

- `background-color`
- `background-image`
- `background-repeat`
- `background-attachment`
- `background-position`

It does not matter if one of the property values is missing, as long as the other ones are in this order. Note that we do not use the background-attachment property in the examples above, as it does not have a value.

CSS background-size Property

- The `background-size` property specifies the size of the background images.
- There are four different syntaxes you can use with this property: the keyword syntax ("auto", "cover" and "contain"), the one-value syntax (sets the width of the image (height becomes "auto"), the two-value syntax (first value: width of the image, second value: height), and the multiple background syntax (separated with comma).
- **Cover VS Contain** :-cover tells the browser to make sure the image always covers the entire container, even if it has to stretch the image or cut a little bit off one of the edges. contain , on the other hand, says to always show the whole image, even if that leaves a little space to the sides or bottom.

```
#example1 {
  background: url(mountain.jpg);
  background-repeat: no-repeat;
  background-size: auto; /*sets Default image size*/
}

#example2 {
```

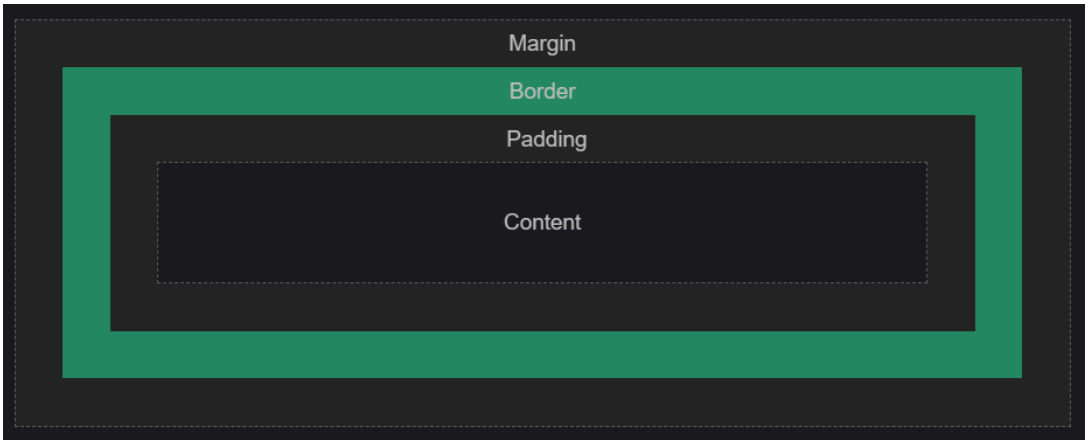
```
  background: url(mountain.jpg);
  background-repeat: no-repeat;
  background-size: cover; /*Fits and no empty space remains even if it has to cut a little bit of imgage*/
}

#example2 {
  background: url(mountain.jpg);
  background-repeat: no-repeat;
  background-size: contain; /*Fits and show the whole image even if it has to lea ve a little space*/
}

#example2 {
  background: url(mountain.jpg);
  background-repeat: no-repeat;
  background-size: 300px ; /*sets width & height becomes auto*/
}

#example2 {
  background: url(mountain.jpg);
  background-repeat: no-repeat;
  background-size: 300px 100px; /*sets width & height*/
}
```

CSS Box Model

Introduction

In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model :



Explanation of the different parts :

- **Content** - The content of the box, where text and images appear
- **Padding** - Clears an area around the content. The padding is transparent
- **Border** - A border that goes around the padding and content
- **Margin** - Clears an area outside the border. The margin is transparent

The box model allows us to add a border around elements, and to define space between elements.

Setting Width & Height

We can set width and height in CSS as follows

```
#box {
        height: 70px;
        width: 70px;
}
```

Note that the total width/height is calculated as follows :

Total height = height + top/bottom padding + top/bottom border + top/bottom margin

Setting Margin & Padding

We can set margin and padding as follows:

```
.box{
        margin: 3px;        /* Sets top, bottom, left & right values*/
        padding: 4px;       /* Sets top, bottom, left & right values*/
}
```

Copy

```
.boxMain{
        margin: 7px 0px 2px 11px;      /*top, right, bottom, left*/
}
```

Copy

```
.boxLast{
        margin: 7px 3px;          /*(top & bottom) (left & right)*/
}
```

We can also set individual margins/padding like this:

margin-top: 70px

margin-bottom: 3px

margin-left: 8px

margin-right: 9px

#Same goes with padding also

Setting Borders

We can set the border as follows

```
.bx{
        border-width: 2px;
        border-style: solid;
        border-color: red;
}
```

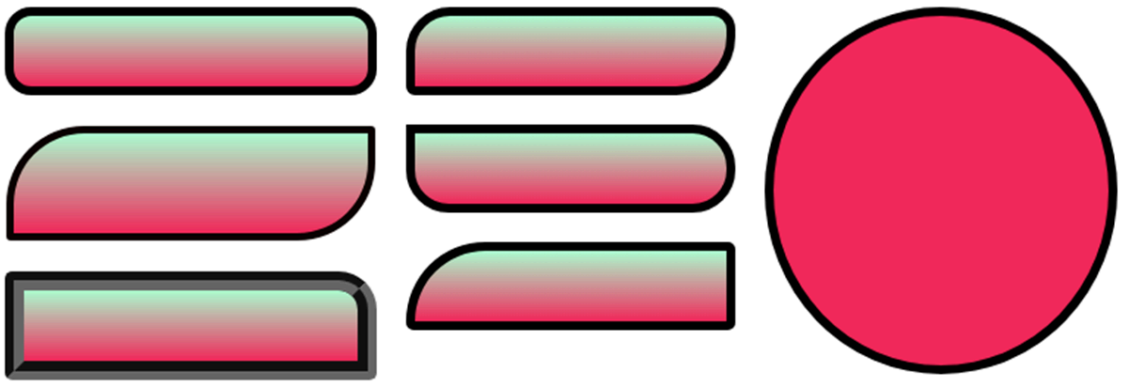
Shorthand for above codes,

```
set border: 2px solid red; /*widht style color*/
```

Border Radius

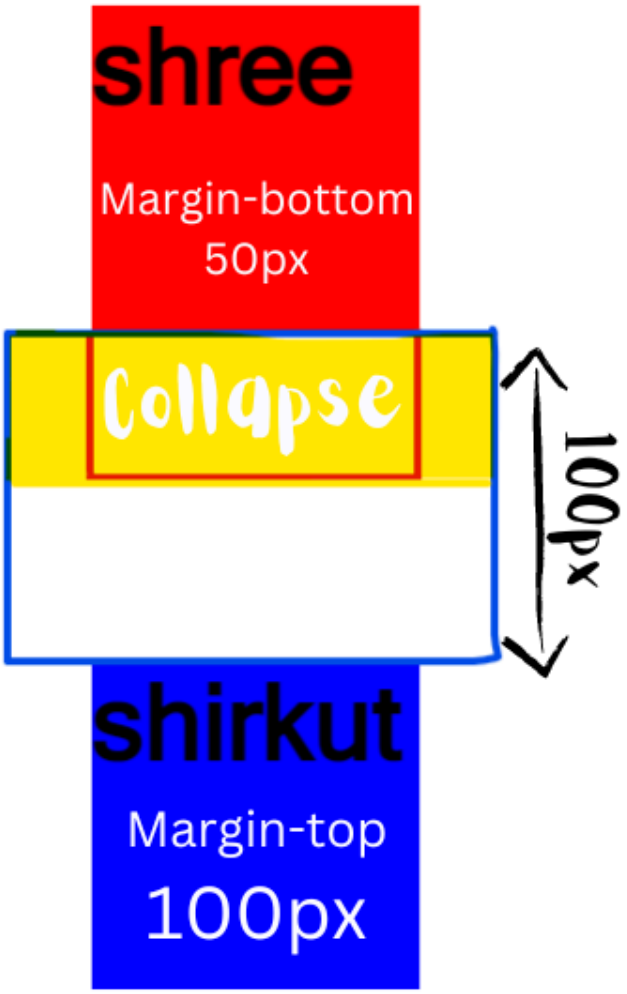
We can set border-radius to create rounded borders

```
.div2{
        border-radius: 7px;
}
```



Margin Collapse

When two margins from different elements overlap, the equivalent margin is the greater of the two. This is called margin collapse.



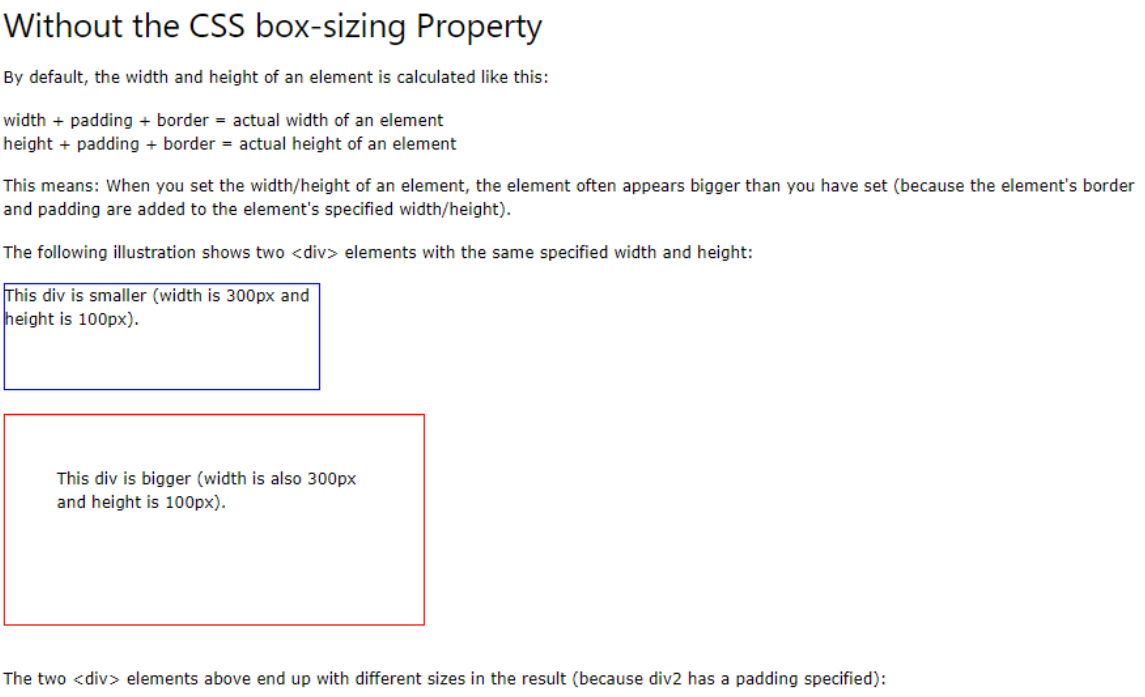
Highlited area is margin collapse.

```
.shree{
    height: 100px;
    width: 100px;
    background-color: red;
    margin-bottom: 50px;
}
.shirkut{
    height: 100px;
    width: 100px;
    background-color: blue;
```

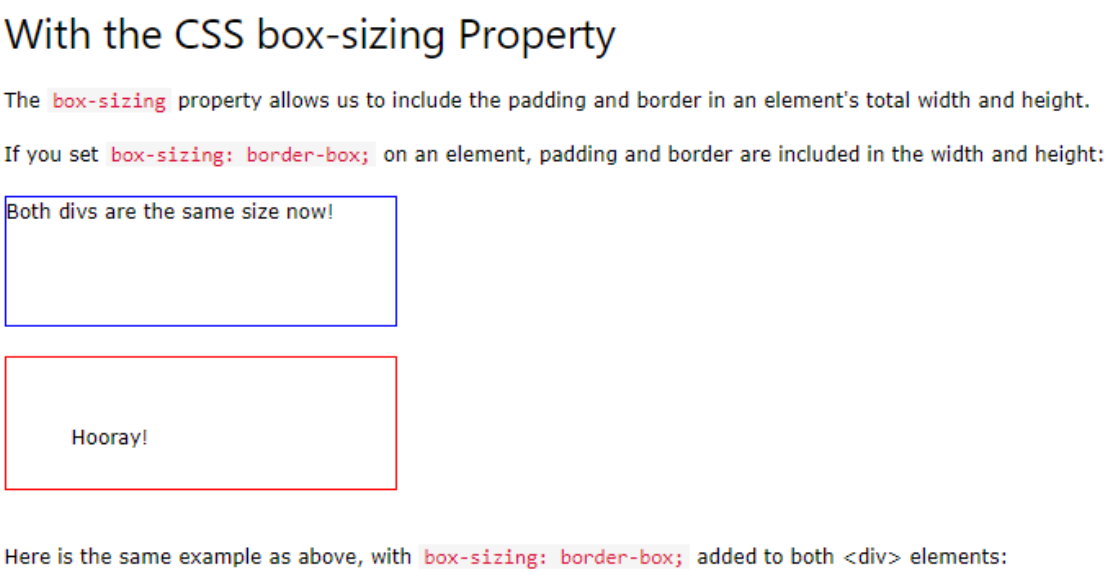
```
  margin-top: 100px;
}
```

Box Sizing

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




```
Example:-
.div1 {
  width: 300px;
  height: 100px;
  border: 1px solid blue;
}

.div2 {
  width: 300px;
  height: 100px;
  padding: 50px;
  border: 1px solid red;
}
```




```
Example:-
.div1 {
  width: 300px;
  height: 100px;
  border: 1px solid blue;
  box-sizing: border-box;
}

.div2 {
  width: 300px;
  height: 100px;
  padding: 50px;
  border: 1px solid red;
  box-sizing: border-box;
}
```

Since the result of using the `box-sizing: border-box;` is so much better, many developers want all elements on their pages to work this way.

The code below ensures that all elements are sized in this more intuitive way. Many browsers already use `box-sizing: border-box;` for many form elements (but not all - which is why inputs and text areas look different at width: 100%;).

Applying this to all elements is safe and wise:

CSS Fonts

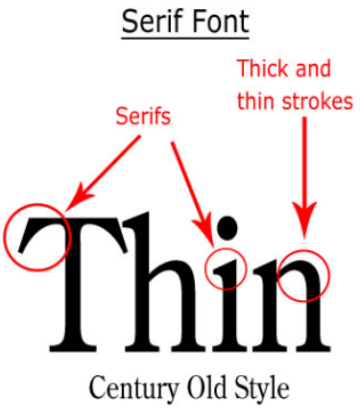
Introduction

Fonts are the most significant components we see on any website, so while building a website, choosing the correct/suitable font is very important. fonts enhance the visual appearance of any website and the use of appropriate font helps in increasing the readability and effectiveness of any website.

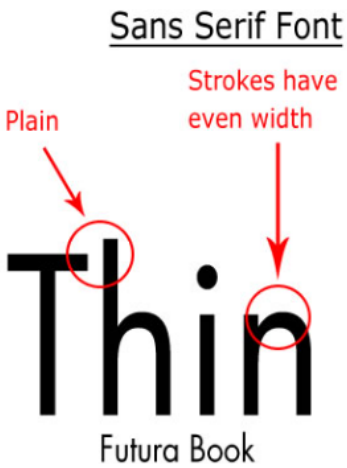
Generic font families in CSS

There are five types of font famillies in CSS ex:- Serif, Sans-serif, Monospace, Fantasy and Cursive. All these font families are used to specify the appearance/shape of the fonts.

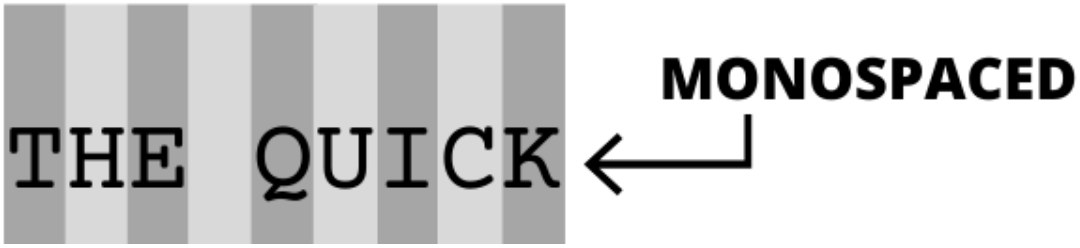
- **Serif Fonts :-** Have small stylish strokes at the edge of the letters.



- **Sans-serif :-** Fonts have plain letters, no stroke. These fonts are easily readable.



- **Monospace :-** Fonts are fixed-pitch fixed-width letters. Mostley used to represent the programming code.



- **Cursive :-** Fonts are very much similar to human writing.

- **Fantasy :-** Fonts have decorative/fancy look.

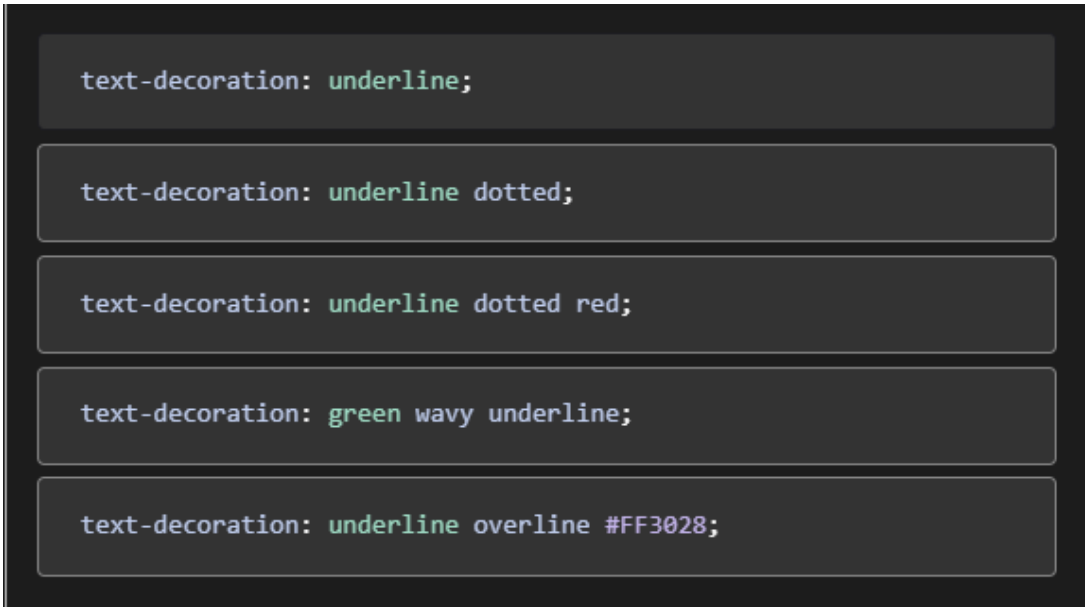
text-align property

Used to set the horizontal alignment of a text

```
.div1{
        text-align: center; /*left,center,right*/
}
```

text-decoration property

Used to decorate the text



☆ visibility: hidden Property →

With display: none, the element is removed from the document flow. Its space is not blocked. With visibility: hidden, the element is hidden but its space is reserved.

- Code →

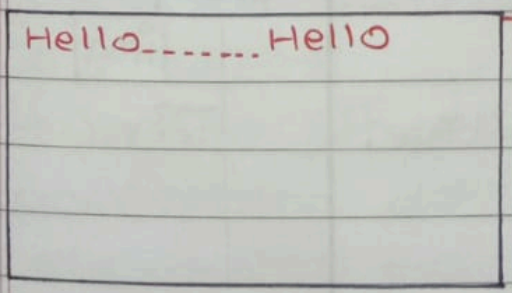
Html →

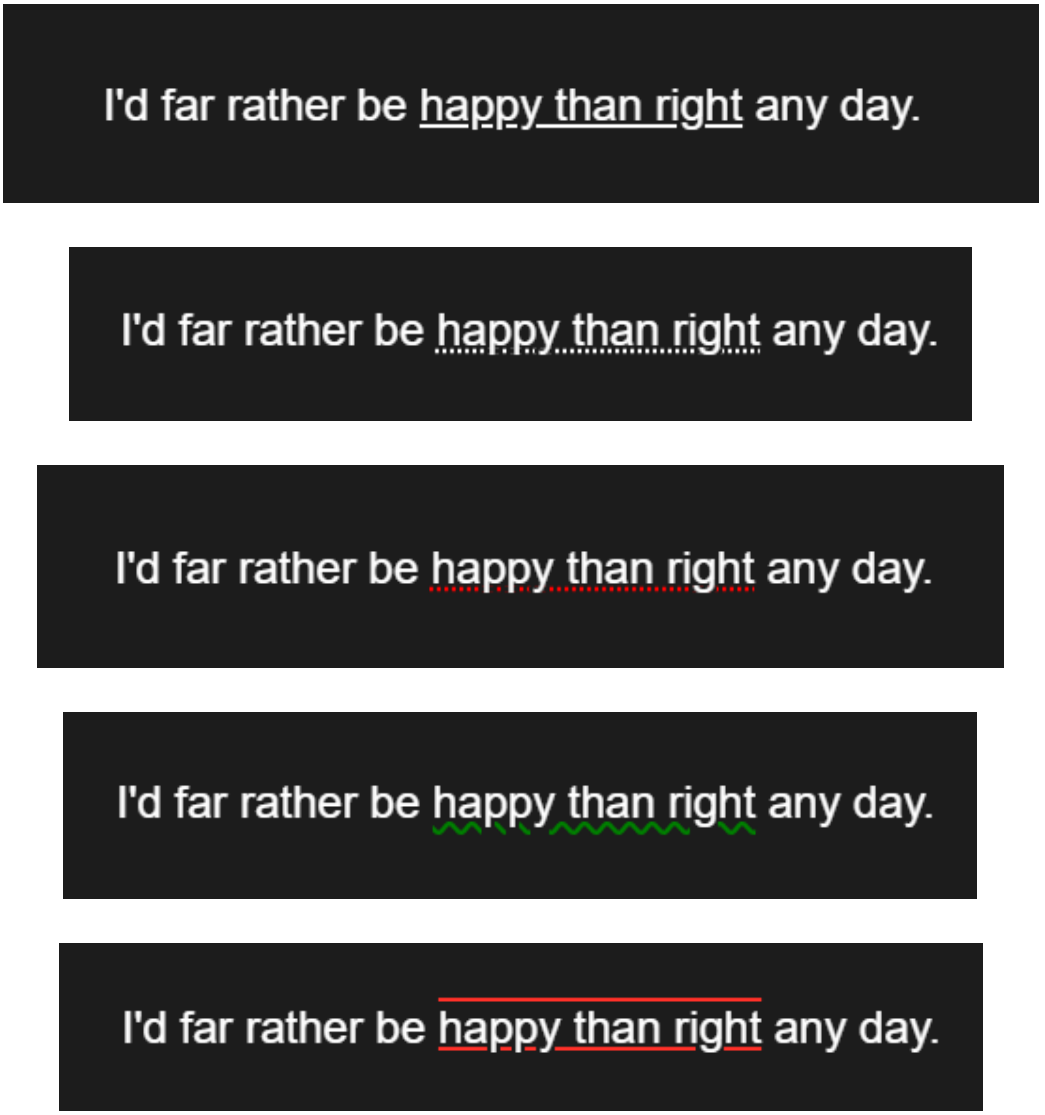
```
<p>Hello</p>
<p class="hidden">World <p>
<p>Hello</p>
```

- CSS →

```
.hidden {
    visibility: hidden;
}
```

- Output →





text-transform property

Used to specify uppercase and lowercase letters in a text

```
p.uppercase{
        text-transform: uppercase; /*lowercase*/
}
```

line-height property

Used to specify the space between lines

```
.Small{
        line-height: 0.7;
}
```

Font-family

Font family specifies the font of a text. It can hold multiple values as a “fallback” system.

```
p{
        font-family: “Times new Roman”, monospace;
}                       /*font 1*/     /*font 2*/
```

#always follow the above technique to ensure the correct font of your choice is rendered.

Other Font Properties

Some of the other font properties are listed below:

- font-size: Sets the size of the font
- font-style: Sets the font style
- font-variant: Sets whether the text is displayed in small-caps
- font-weight: Sets the weight of the font
- font-shorthand: Set’s all values in one line code.
- font-family: Set’s font family.

Web Safe fonts

Some of the browser’s have not all fonts download or in any case, your font don’t work, you have to use different technique’s to embed the font or you can give second option in code as follow’s:-

```
example:-
font-family: Vardana, Sans-serif;
                /*second font*/
```

Safe fonts/embedding fonts

If you do not want to change the feel of your website, you can use some safe font’s or technique’s to add that font in your html file.

1. CSS web safe fonts.
2. CSS font stack: web safe.
3. Google fonts (Recommended)

How to Embed Google fonts

To use google fonts, first you have to embed the font-family in your head section and secondly you have to specify the font family in CSS file.

Example

Here, we want to use a font named "Sofia" from Google Fonts:

```
<head>
<link rel="stylesheet" href="https://fonts.googleapis.com/css?
family=Sofia">
<style>
body {
  font-family: "Sofia", sans-serif;
}
</style>
</head>
```

Result:

Sofia Font

Lorem ipsum dolor sit amet.

123456790

Note:- 99% people will see this font because we are embedding it, instead of presuming that people will already have it.

CSS Measurement Units

There are two types of units you can use in CSS, relative and absolute units.

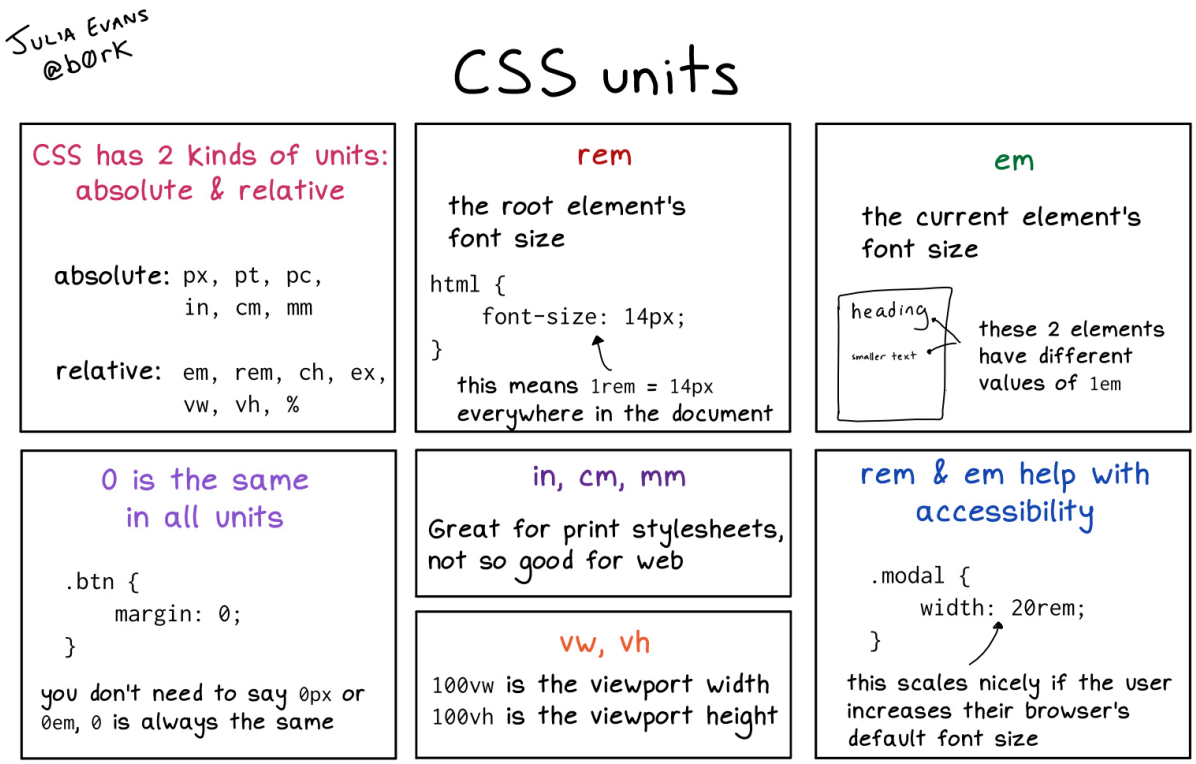
What is the difference between absolute and relative units of measurement?

💡 Relative units are based on the value of something else, such as the size of a parent element. while absolute units always indicate a fixed length regardless of the viewport or font size.

Why are relative units preferred over absolute units in CSS?

💡 Relative units are preferred over absolute units in CSS b**ecause different devices have differing physical sizes as well as different pixel resolutions**.

Units in depth



CSS UNITS

Absolute length

Units	Full Form	Font-size
px	pixel	font-size: 16px
pt	point	font-size: 12pt
pc	pica	font-size: 1pc
in	inches	font-size: 1in
cm	centimeter	font-size: 1cm
mm	millimeter	font-size: 10mm
q	quarter	font-size: 16q

Relative Length

Units	Definition
em	Relative to the font-size of the current element
ex	Relative to the font's x-height
%	Relative to the enclosing parent element in percent
ch	Relative to the width of the digit "0"
rem	Relative to the font-size of the root element
vw	Relative to 1% of the width of the viewport
vh	Relative to 1% of the height of the viewport
vmin	Relative to 1% of the viewport (smaller between vw & vh)
vmax	Relative to 1% of the viewport (bigger between vw & vh)

CSS Font Sizing (rendering)

Rendering of CSS units

(em) (px) (%) (rem)

case I
(em)

```
body{
  Font-size:20em;
}
```

```
h1{
  Font-size:30em;
}
```

Parent → 20em + child → 30em
↳ Rendered → 50em ↲

case II
(px)

```
body{
  Font-size:20px;
}
```

```
h1{
  Font-size;30px;
}
```

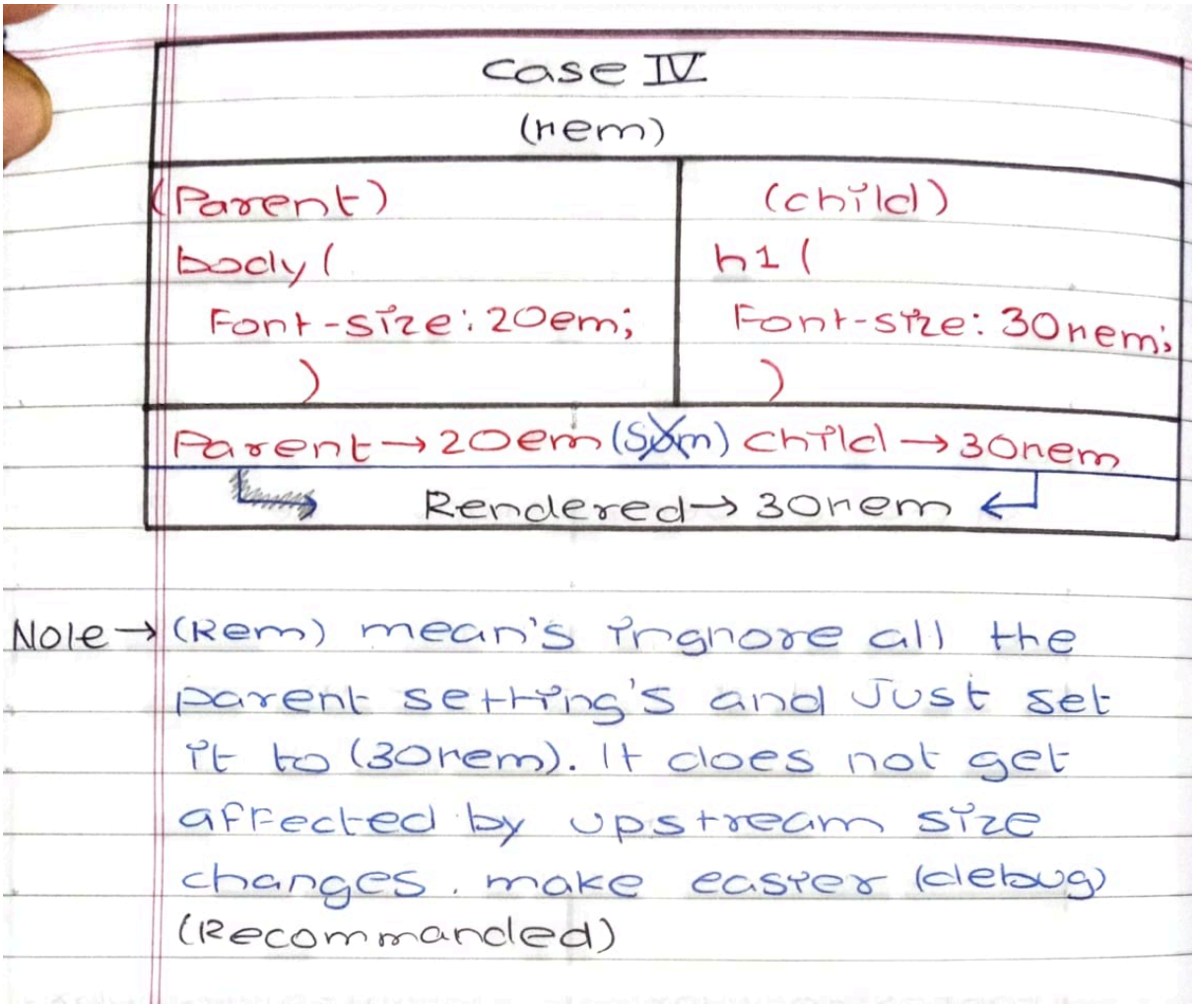
Parent → 20px (~~5em~~) child → 30px
Rendered → 30px ↲

case III
(%)

```
body{
  Font-size:20%;
}
```

```
h1{
  Font-size;30%;
}
```

Parent → 20% + child → 30%
↳ Rendered → 50% ↲



Useful tool's

- **Flaticon.com** ➡️ for cool icon more than 60000 thousand.
- **GIPHY.com** ➡️ Best animated videos.
- **Lorem ipsum** ➡️ you can use lorem before having any content to put on website.

CSS Display property

The display CSS property **sets whether an element is treated as a block or inline element and the layout used for its children, such as flow layout, grid or flex**. Formally, the display property sets an element's inner and outer display types.

☆ Display Block Property →

Block element's take's full width.

→ common Block Elements.

- Paragraphs <p>
- Headers (<h1> through <h6>)
- Division's
- Lists and list items (<ol>, <ul>, &)
- Forms <form>

☆ Display Inline property →

Inline element's take's width as per use.

→ Common Inline Elements.

- Spans
- Images <img>
- Anchors <a>

Q You would wonder why would I use block element, If we can simply use inline elements.

★ Problem →

But there is a problem with inline element ↓

	CSS →	Output →

(block)	p{
	bg-color: red;
	width: 100px;
	}
		Hello
		← 100px →

(R) Here we see <p> tag is a block element. and we have applied width of 100px. and it is applied in output.

	CSS →	Output →

inline)	span{
	bg-color: red;
	width: 100px;
	}
		Hello
		← remain → same

(R) Here we see <Span> tag is a inline element. and we have applied width of 100px. and it is not applied in the output.

★ Solution →

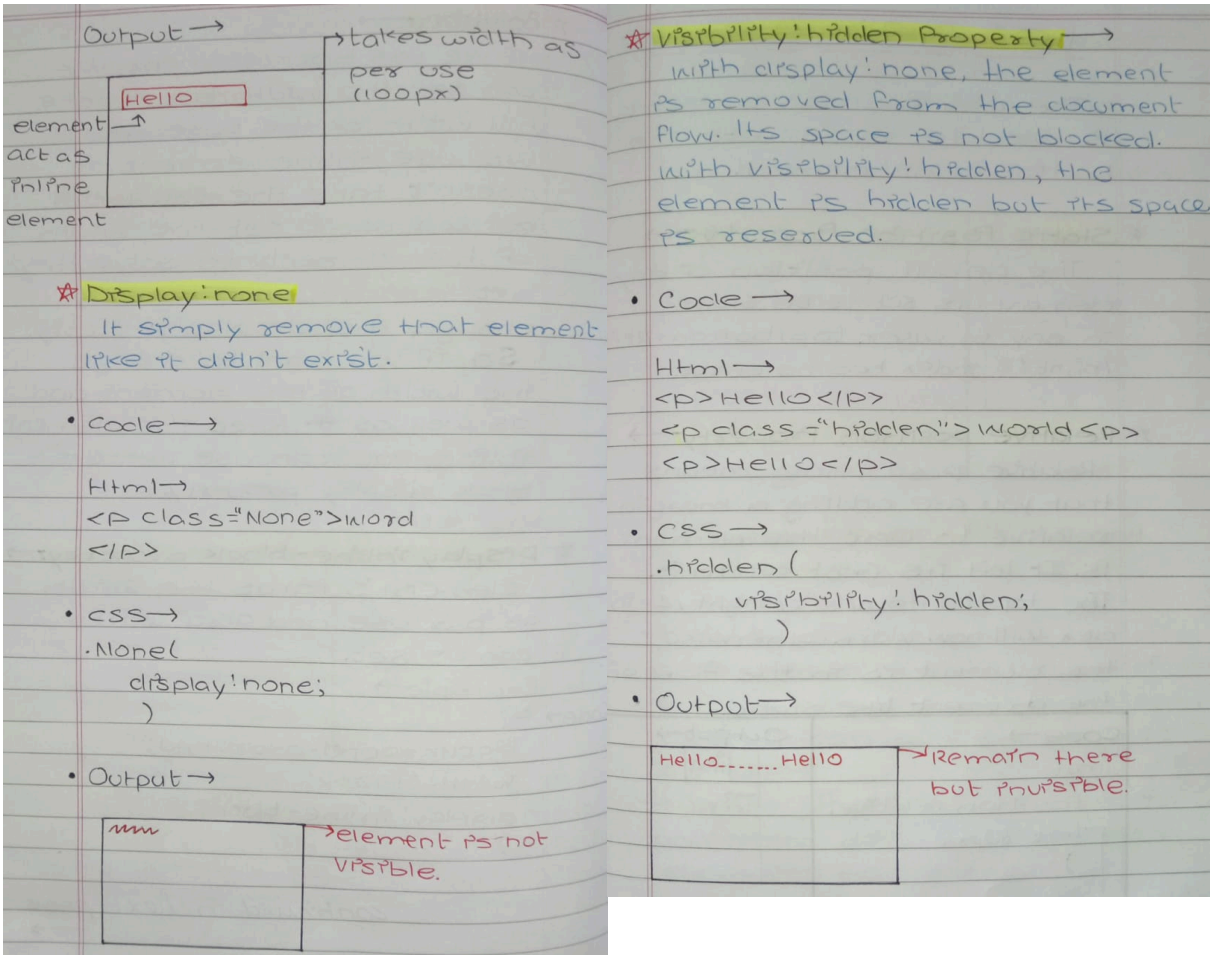
If we use block element, we can set the width but it take full width of the page. and If we use inline element, it doesn't take the full width, but we can't set the width of the element. to solve this problem you can actually change the display property. So, If you want to change the width of the element and as well as keep the element inline, you can use inline-block display property.

★ Display inline-block property: →

Element's takes the width as per use and also width can be set.

Example →

Code →
```
p{
background-color: red;
width: 100px;
display: inline-block;
}
```



💻Animated example:-

https://appbrewery.github.io/css-display/

CSS Position Property

The CSS position property is used to manipulate the location of an element.

Static Position Property

The Default position of any element is set to be static in any browser.
top/bottom/left/right/z-index has no effect.

Relative Position Property

Relative positioning means that you are adding a margin relative to were the element is , it left its Ghost behind it, the top/bottom/left/right/z-index will now work. otherwise, the element is in the flow of the document like static.

Absolute Position Property

Absolute positioning mean's that you are adding a margin to its parent element and top/bottom/left/right/z-index work's.

Fixed Position Property

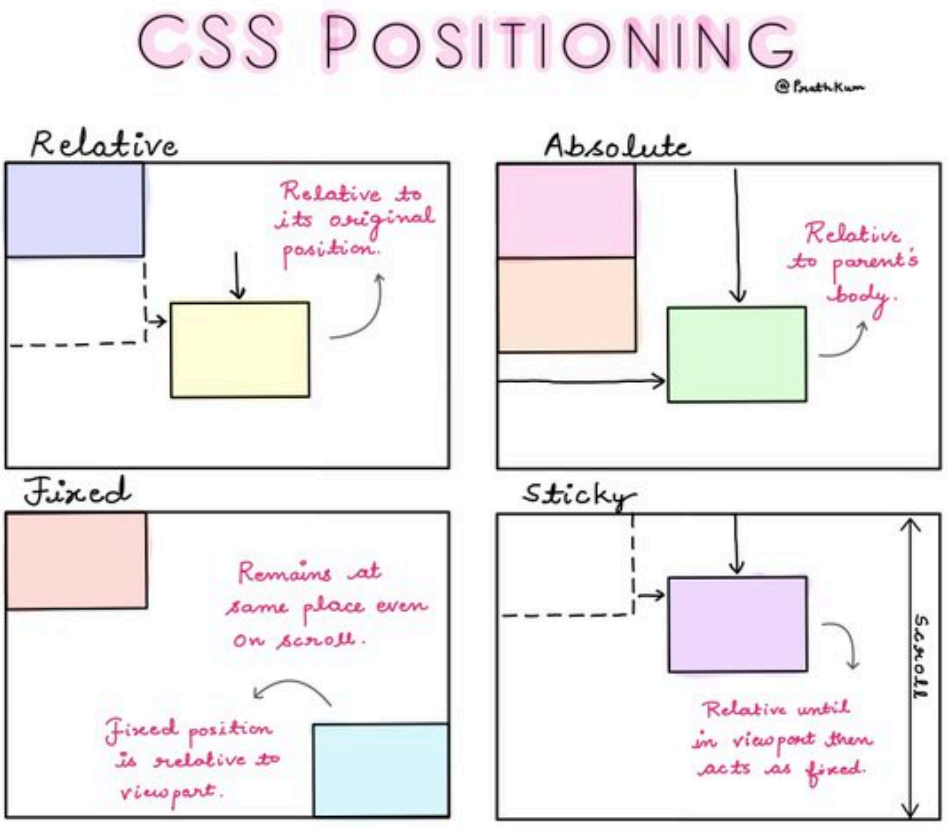
Fixed property is mostly used to keep a navbar or buy now button visible while scrawling.

Sticky Position Property

An element with `position: sticky;` is positioned based on the user's scroll position.

A sticky element toggles between `relative` and `fixed`, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position: fixed).

Animated example:-

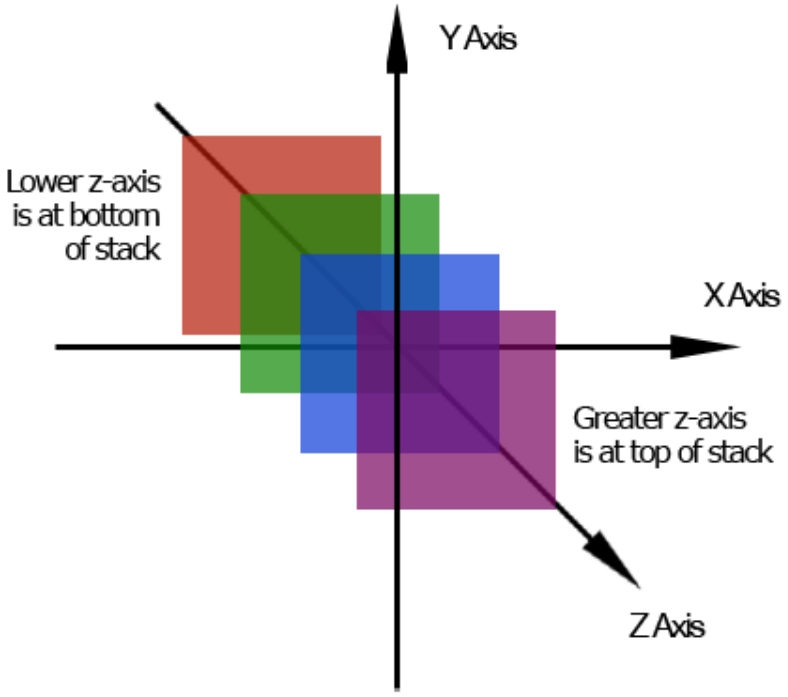


```
example:-
position:relative;
position:absolute;
position:fixed;
position: sticky;
```

CSS Z-index and stacking order.

Overview

We know that any object has two axis called **y-axis** and **x-axis**. but there is also one more axis called **z-axis**, which is shown as below:-



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

Key Notes:-

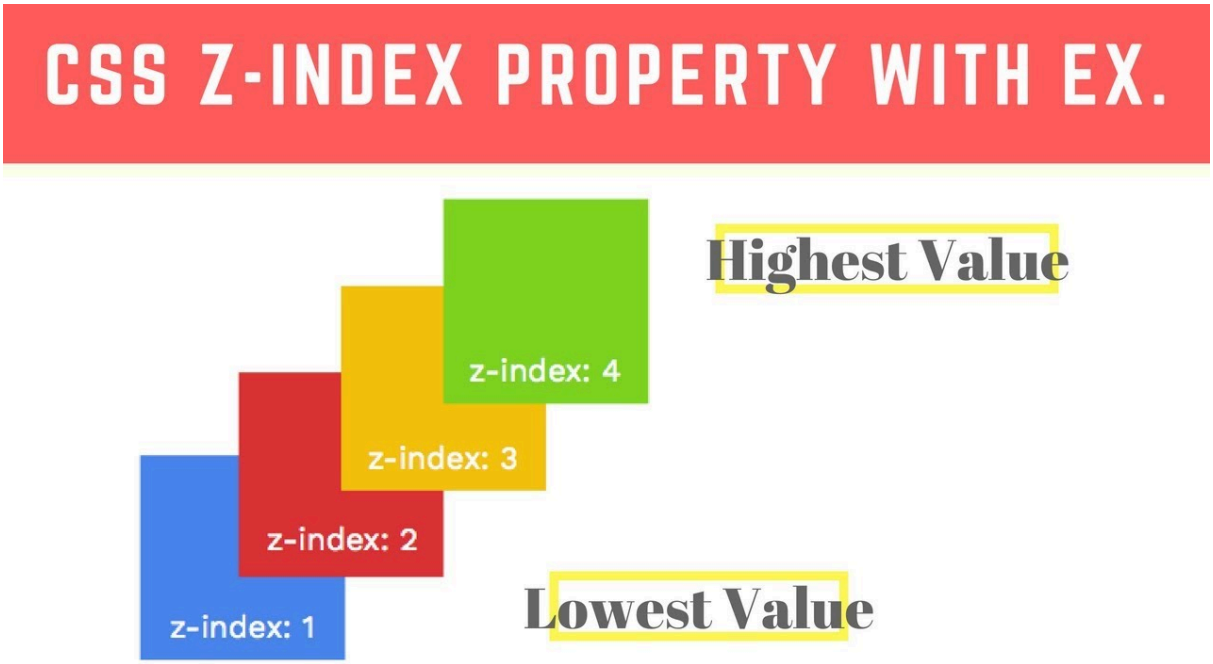
- The default **z-index** value for page element is 0.
- **z-index: 0** is always the **"default layer"** (the layer in which all elements without an explicit/clear z-index reside), and **z-index: auto** means **"set's the stack order equal to its parent"**.

- **z-index** only work's, if you have applied position property (any property other than static) and flex items (elements that are direct children of display: flex elements)

z-index demonstration

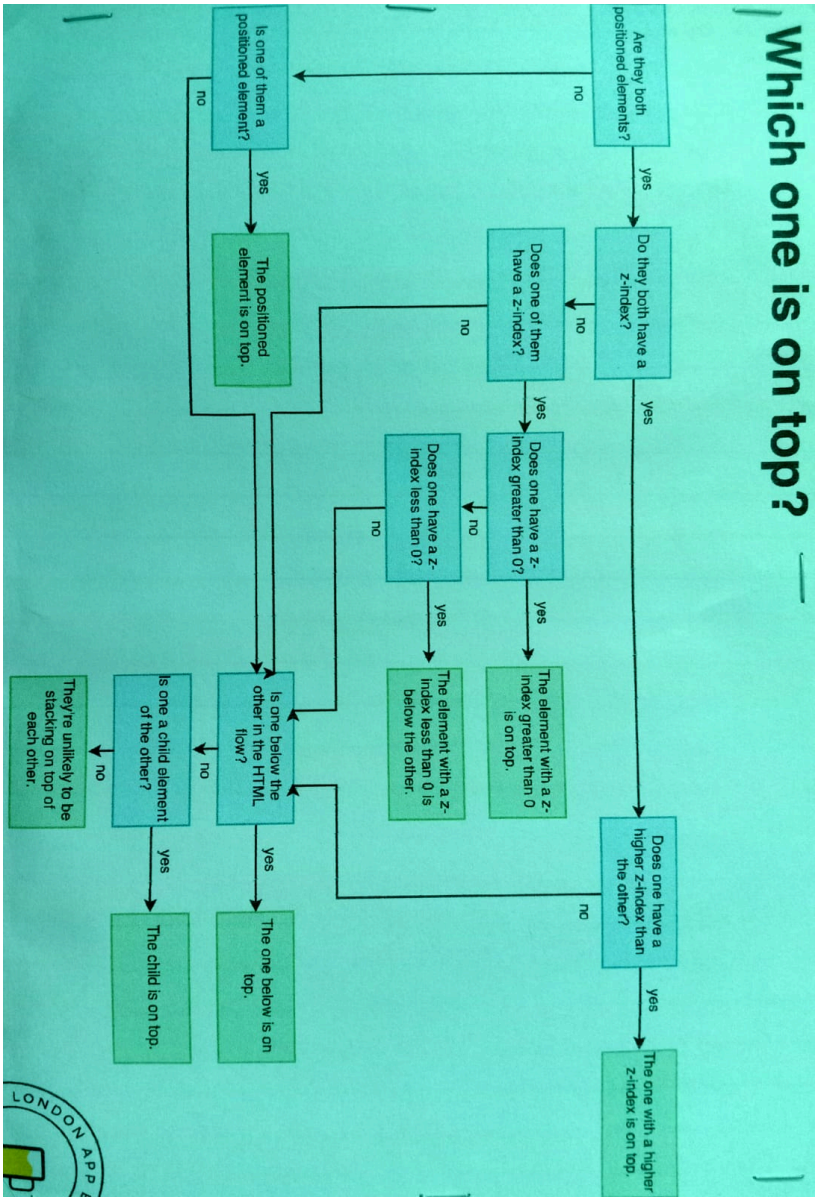
W3Schools CSS z-index demonstration

https://www.w3schools.com/cssref/playdemo.php?filename=playcss_z-index




```
/* Keyword value */
z-index: auto;

/* <integer> values */
z-index: 0;
z-index: 3;
z-index: 289;
z-index: -1; /* Negative values to lower the priority */
```

Which One is on top?

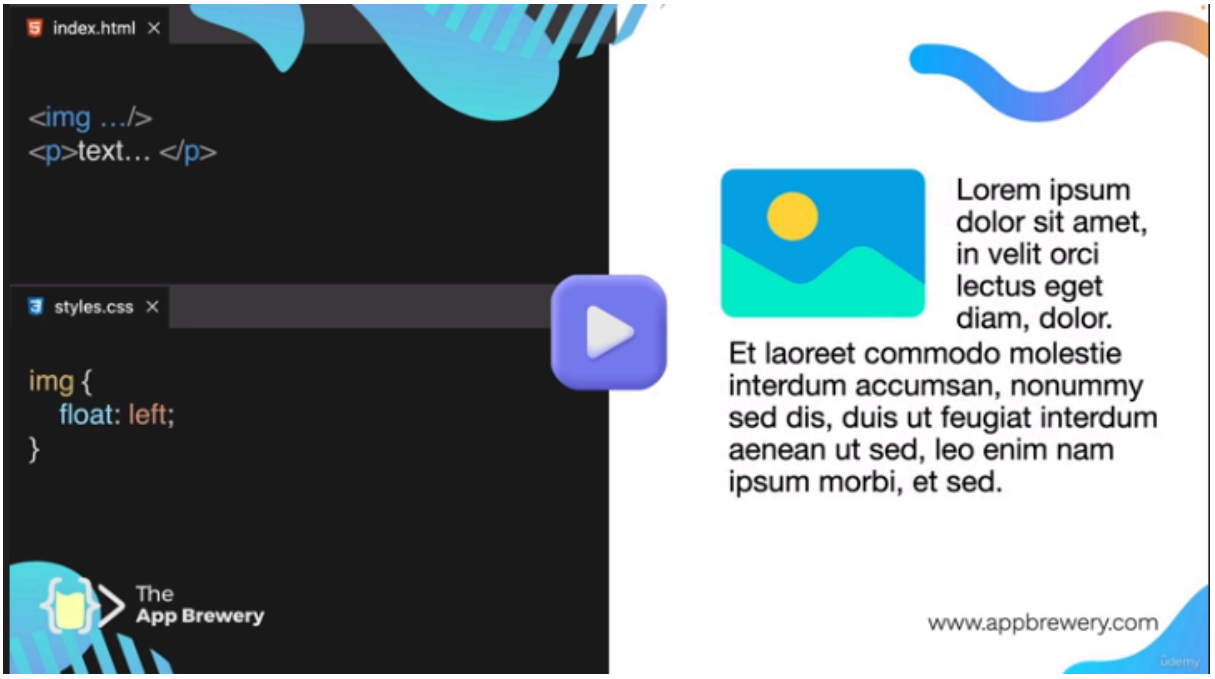


CSS Float Property

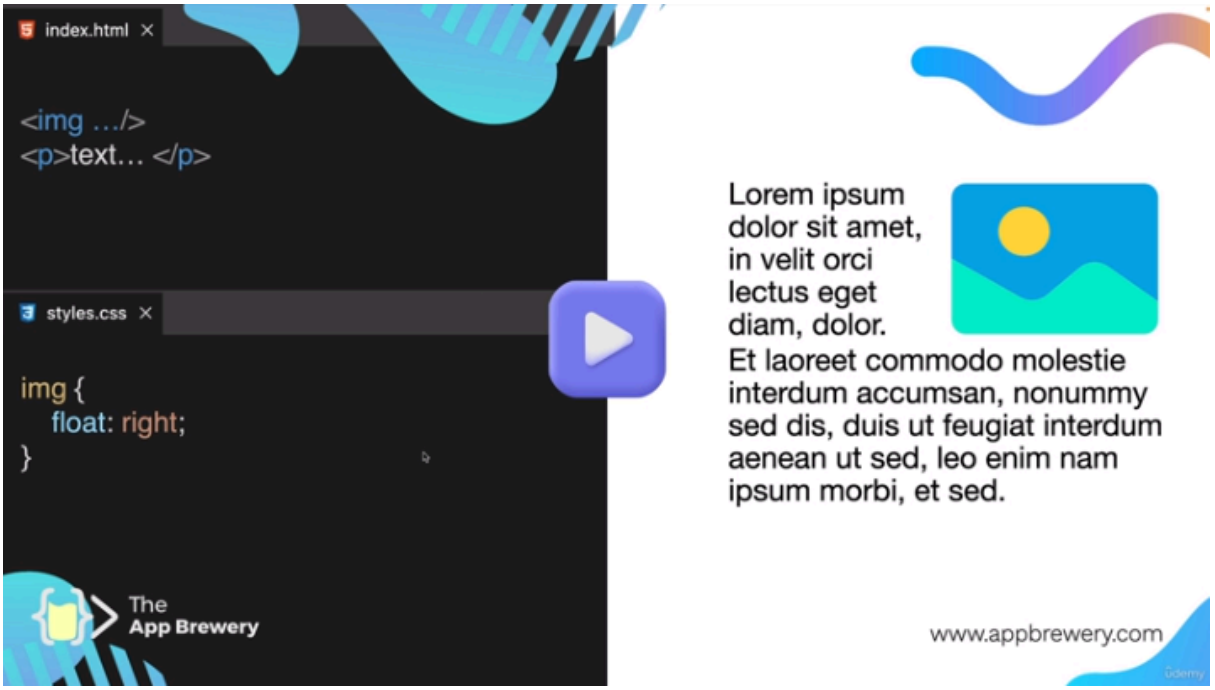
What is CSS float property?

The float CSS property **places an element on the left or right side of its container, allowing text and inline elements to wrap around it**. The element is removed from the normal flow of the page, though still remaining a part of the flow (in contrast to absolute positioning).

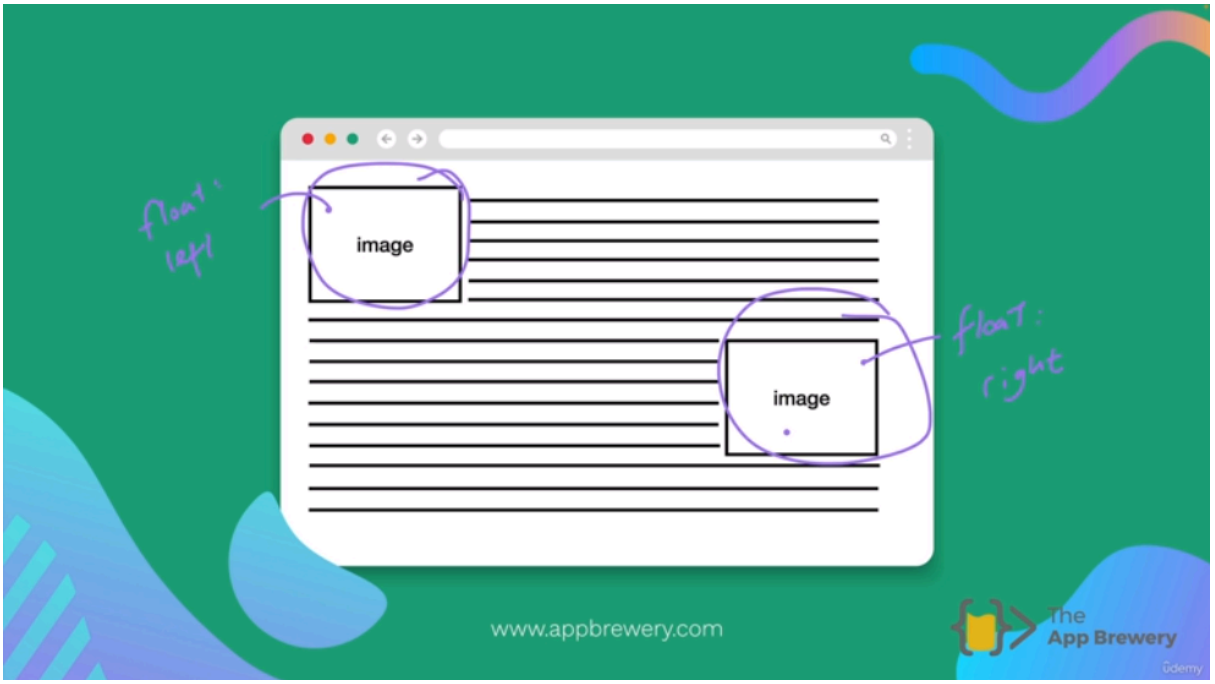
`Float: left;` Property



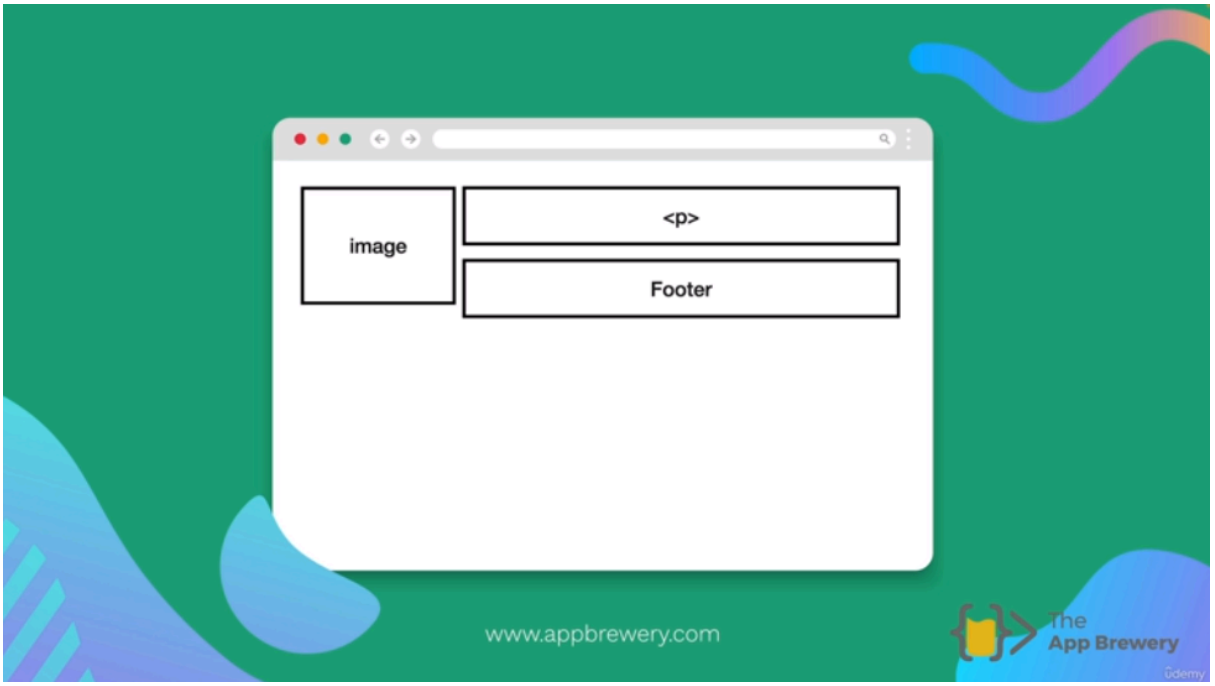
`Float: right;` Property

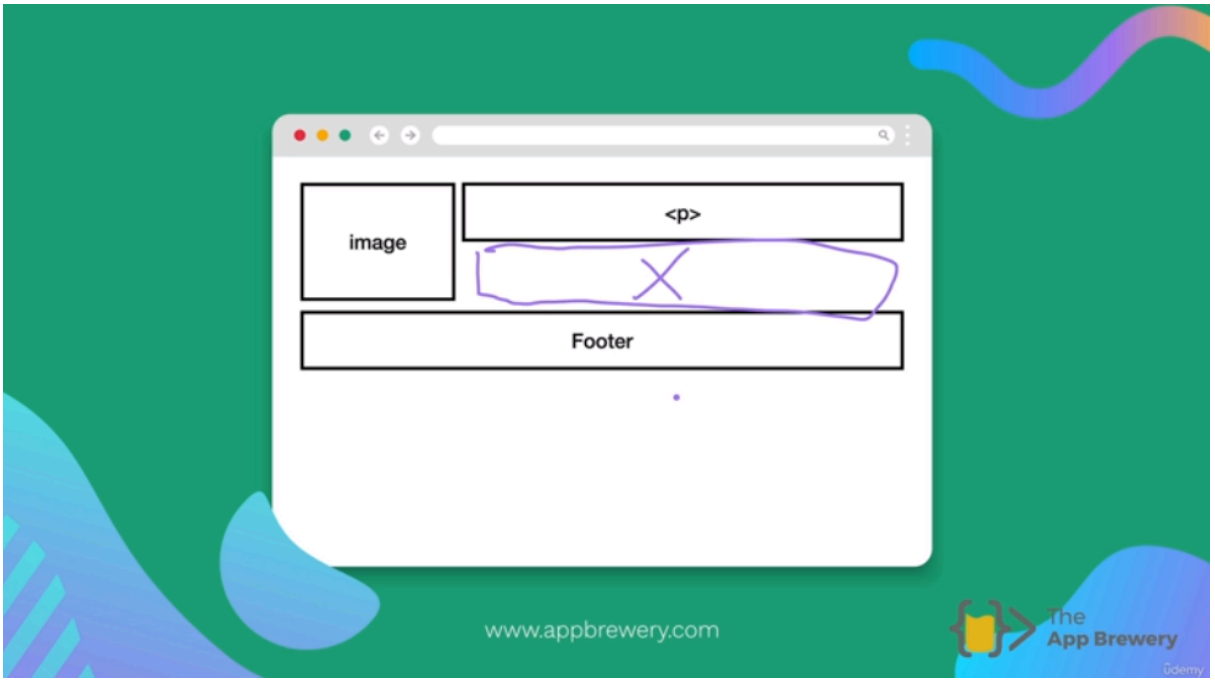


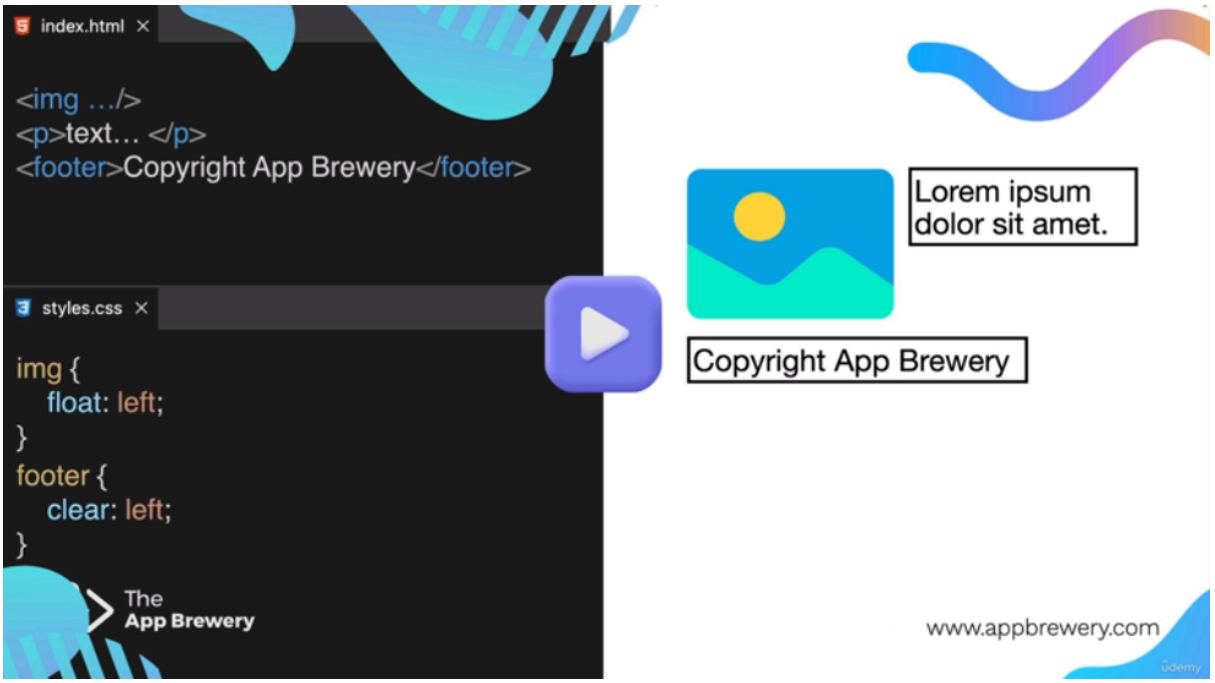
Use case of both `left` & `right` property's



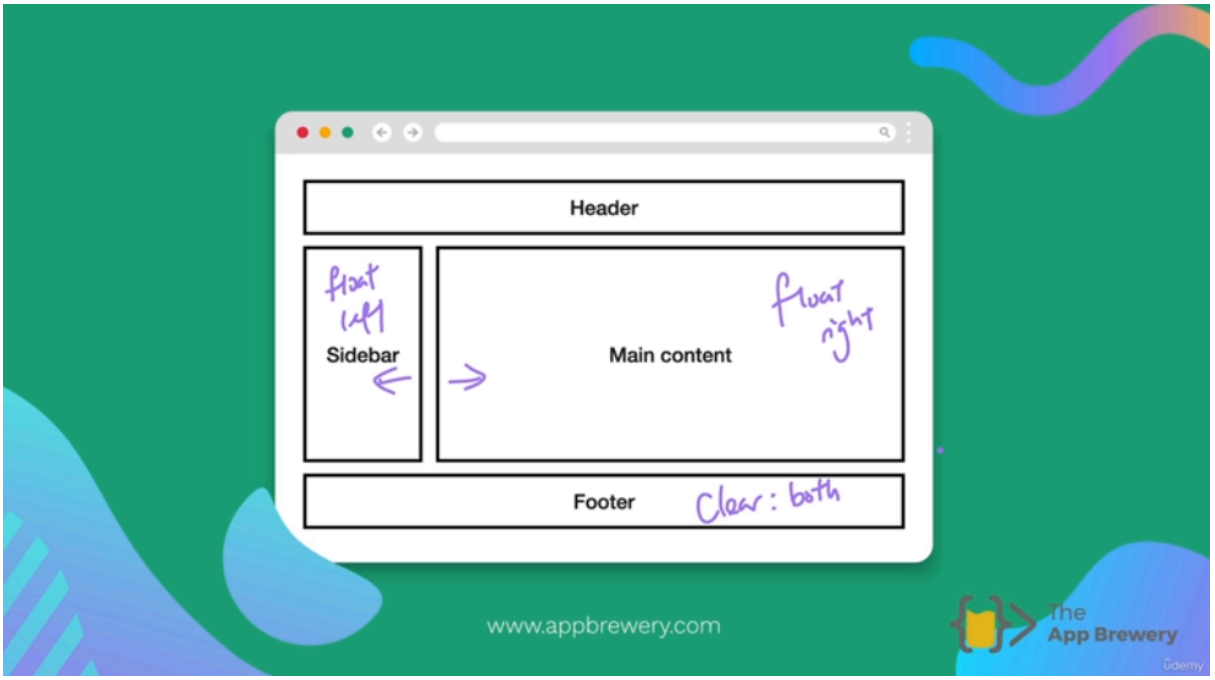
CSS `Clear` property





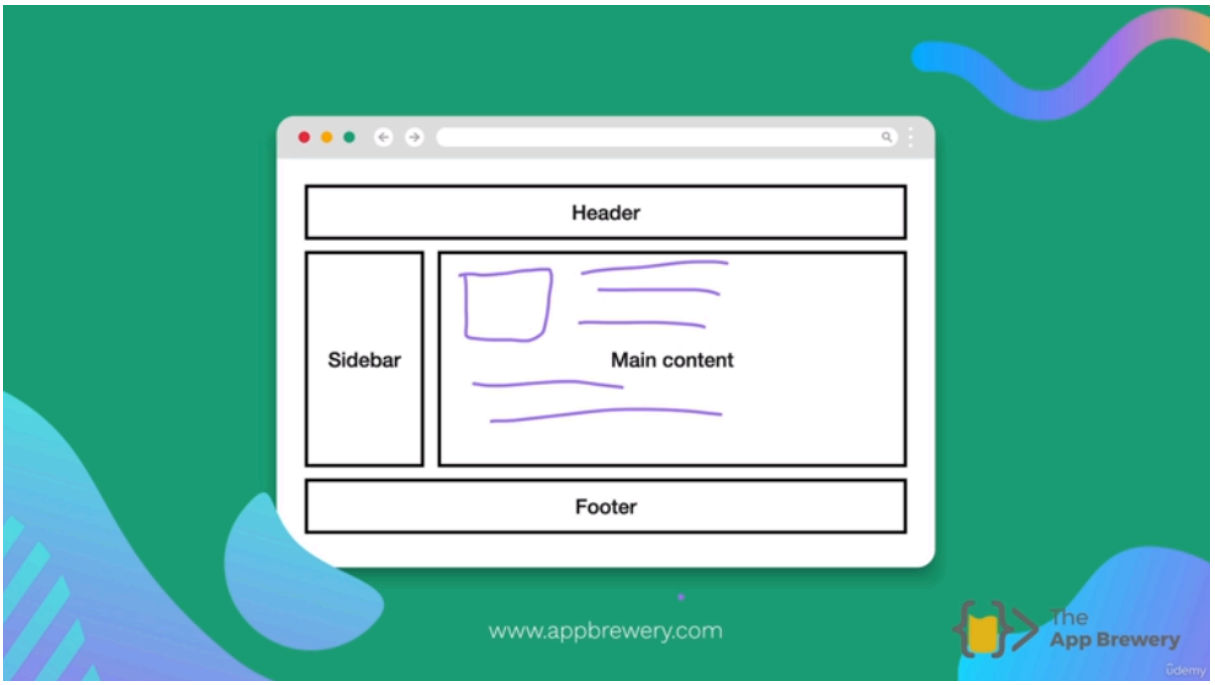


CSS `Clear: both;` property



Conclusion:-

Now I'm showing you this layout so you can use the clear property.



But in modern web development and web design, we don't actually use the float property to achieve these kind of designs because there are much better tools for us like Flexbox, grid bootstrap, whole load of other things that is a lot less complicated to use and to understand than using float. To achieve this, there can be very unexpected results if you're trying to achieve complex layouts using float. So I recommend to only use float when you want to wrap text around an image. And in

modern web design, this is what you're going to see most people doing as well. So it's an important concept to learn and understand, but try to not use it on everything that you see. We've got better techniques...

Thing's not to do🚫

Float is great when you want to float an image to the left or the right of a block of text to wrap the

text around it.

But don't use floats for layout instead.

Use something else like what?

We're going to explore Flexbox Grid Bootstrap, something that is designed specifically for creating

an overall page structure.

How to Create Responsive Websites

What is responsiveness?

Responsive design refers to **a site or application design that responds to the environment in which it is viewed**.

Because these days there are many different screen sizes desktops, laptops, iPads, mobile phones

and we want our website to look good on all of them.

And in order to do that, we need to make our websites responsive so it responds to the changes in the

screen size.

There are four main ways of making website responsive.

Deep dive in all of the four topics🏊

CSS Media Queries

▼ Introduction

> 💡 The **CSS Media Query** gives you a way to apply CSS only when the browser and device environment matches a rule that you specify, for example "viewport is wider than 480 pixels". Media queries are a key part of responsive web design, as they allow you to create different layouts depending on the size of the viewport, but they can also be used to detect other things about the environment your site is running on, for example whether the user is using a touchscreen rather than a mouse an so on.

▼ **Media Query Basics**

The simplest media query syntax looks like this:

```
@media media-type and (media-feature-rule) {
  /* CSS rules go here */
```

```
}
```

It consists of:

- A media type, which tells the browser what kind of media this code is for (e.g. print, or screen).
- A media expression, which is a rule, or test that must be passed for the contained CSS to be applied.
- A set of CSS rules that will be applied if the test passes and the media type is correct.

▼ **Media types**

The possible types of media you can specify are:

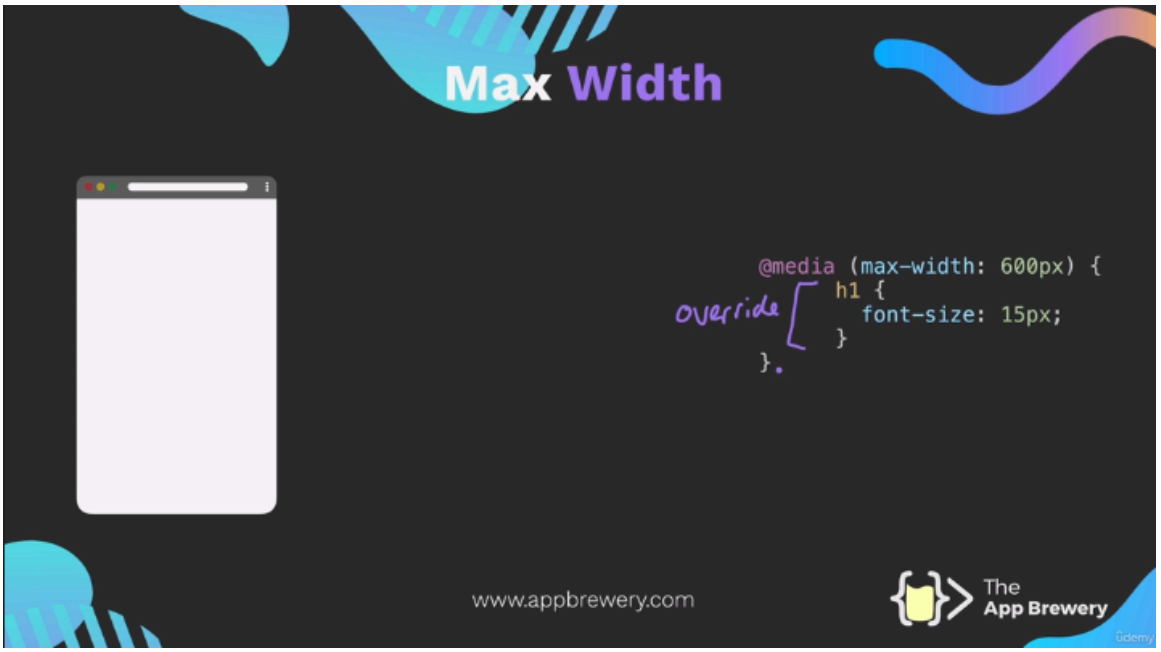
- `all`
- `print`
- `screen`

The following media query will only set the body to 12pt if the page is printed. It will not apply when the page is loaded in a browser.

```
@media print {
  body {
    font-size: 12pt;
  }
}
```

▼ Max-Width

We're using that media query max width in order to define any screen size that is 600 pixels or below.



So anything that is smaller or equal to 600 pixels, then it should have this particular kind of styling

and we provide an override to the default styling.

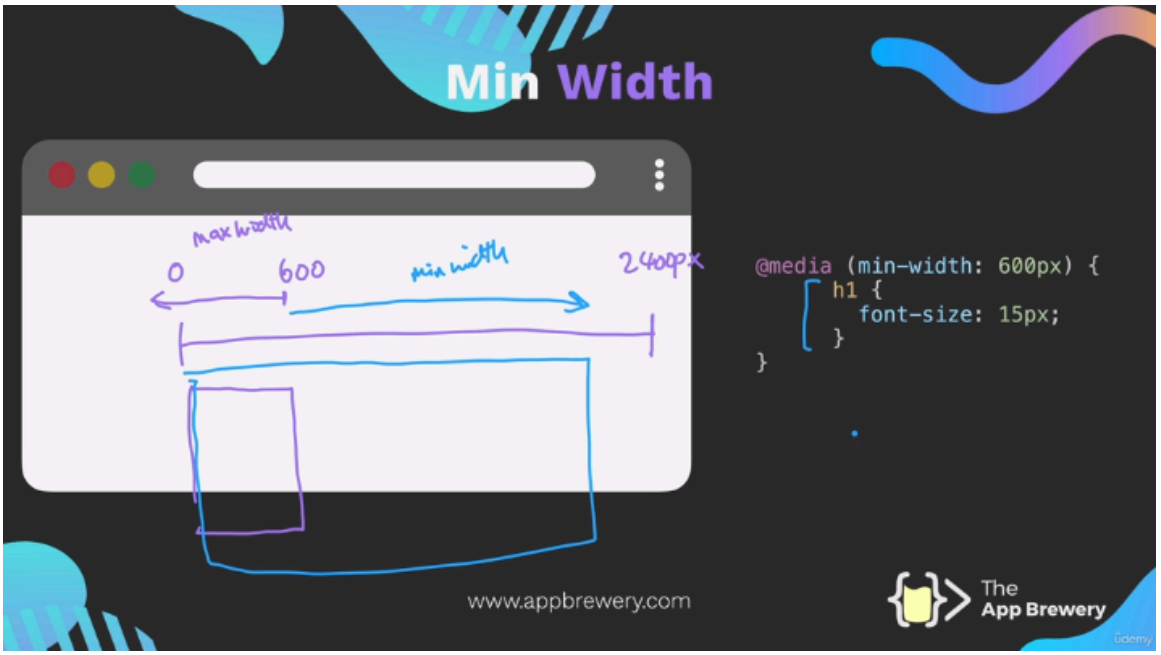
▼ Min-Width

Now alternatively, you can use a different keyword here.

You can use the min width, which is the minimum width, and by changing that from max to min, essentially

you're going the other direction of responsiveness.

So you're saying anything that is from 600 pixels upwards, then we should have a different styling.



So you're basically targeting bigger screens with this min width.

So if you think about the total possible size of a website, so from zero to say, I don't know, 2400

pixels, if you have a screen that is say only 600 pixels right here, then you can use the max width

to target this direction.

But if you want to target all the screens that are bigger, then you could use the min width to target

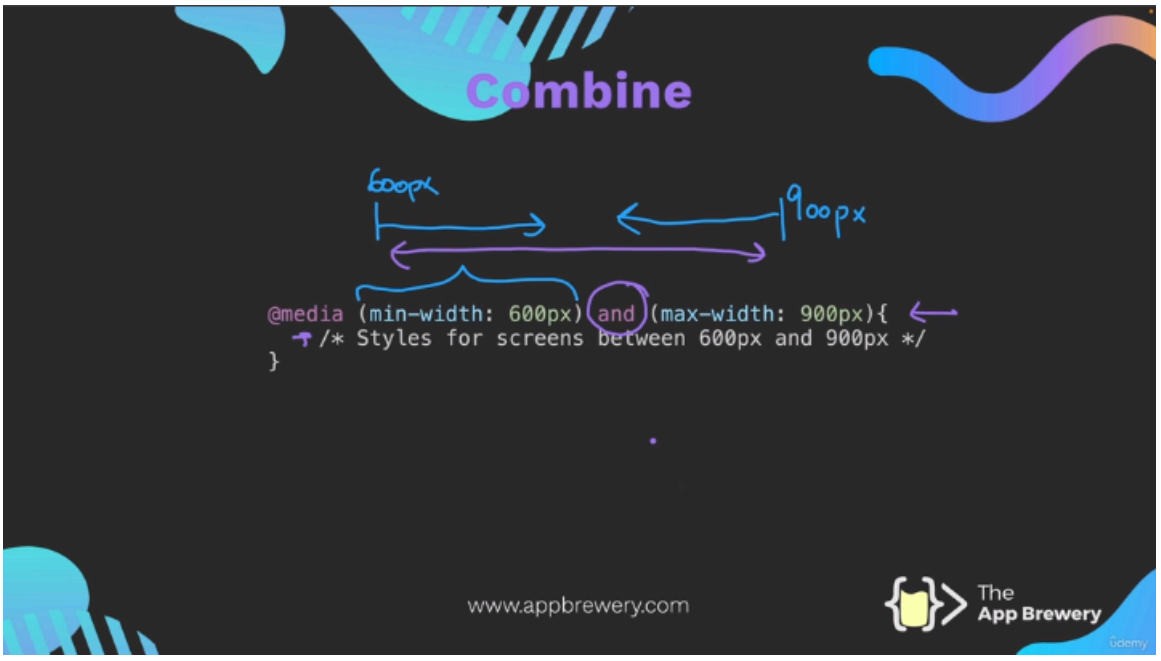
in this direction.

So you're essentially targeting different screen sizes and providing different styles for those different

screen sizes.

▼ Combine

Now you can also combine different breakpoints to target something that is a specific size.



So in this case, you're saying anything that is bigger than 600 pixels.

Using this particular line here.

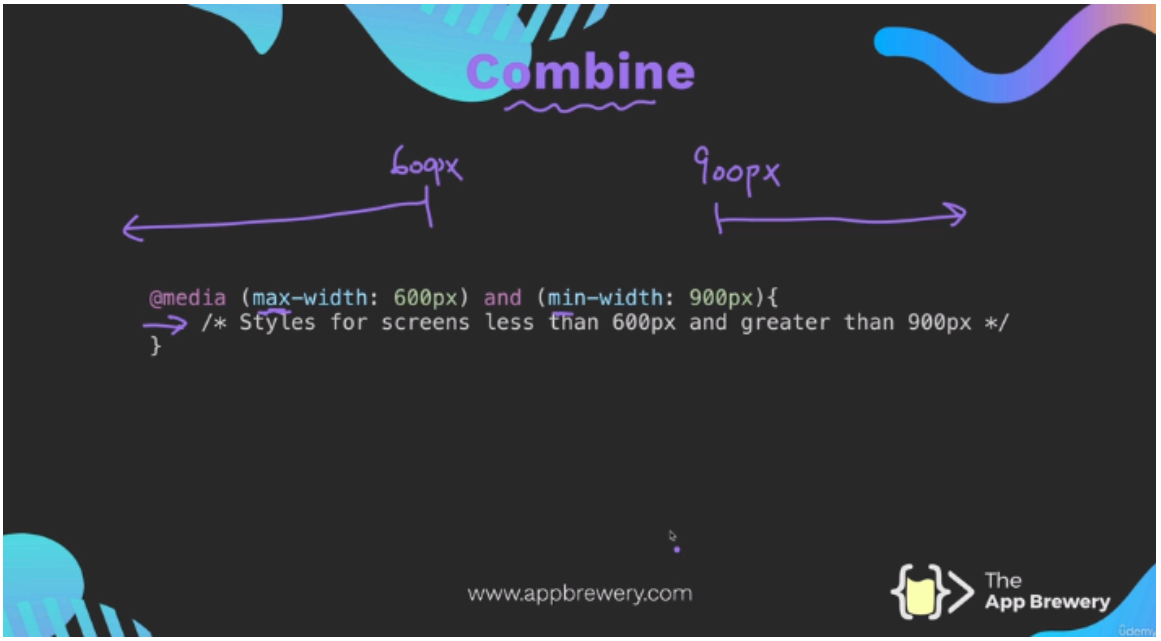
So minimum width for this style is 600 pixels and then maximum width for this style is 900 pixels.

So anything that's less than 900 pixels and in between these two points is where your specific style

is going to be applied in here.

So this means the screens, which are between 600 and 900, will be targeted using this combination

system by adding that and keyword between these two breakpoints.



Now if you flip that around and you change the smaller size to a max width, then you're taking a different

approach.

You're saying anything that is less than 600 pixels and anything that is greater than 900 pixels on

those screen sizes.

Let's apply these styles so you can combine things in many different ways.

You can chop it, change it however you like.

But the important thing is to understand this wording max width and min width, the maximum width where

this style will be applied and the minimum width where this style will be applied.

So when you read it that way, it becomes much simpler to understand.

▼ Device

In addition to max width and min width, you might also see this screen keyword being used, which is

not really necessary by default because it's going to apply it to all screens.

But if we think about the alternative to screen is a keyword called print.

And what this allows you to do is to use the media query to target only when your website is being printed

and to give it a different layout.

I don't normally recommend adding the screen keyword if you're just writing a normal media query, but

when you do see it out there in the wild, then you'll know what that means.

It's just saying targeting screens or targeting print.

For more on media queries and all of the different properties that you can change using media queries,

head over to the developer docs and take a look at using media queries.

You can see under the syntax section there are a lot of things that we've mentioned, such as orientation

or device height, device width or height and width and targeting different media types or targeting

different features.

And there's a lot more that you can do with media queries and you can have very complex media queries,

but most of the time you're going to be using pretty simple media queries such as max width and min

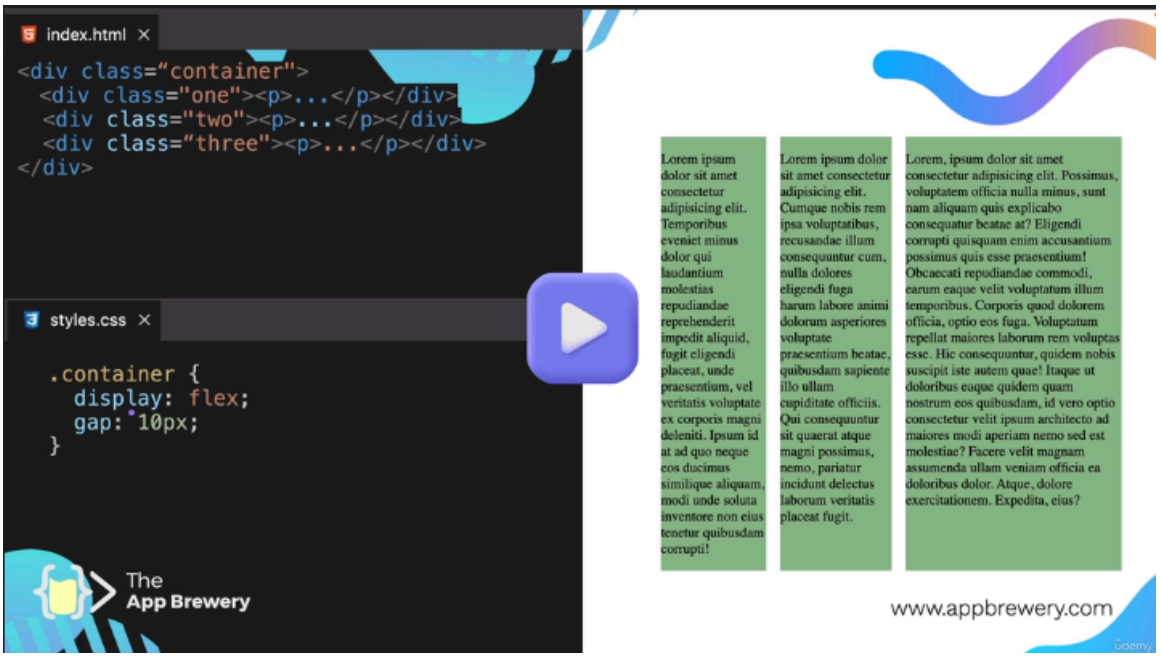
width, and you won't use a lot of these other features.

But as always, it's good to be aware that they exist.

So have a quick read of this and see if there's anything else there that you want to pick up.

CSS Flexbox

▼ `Display: Flex` & `gap` Property



all you need to do is to wrap your divs

inside a container and then target that container in your CSS and set the display to flex.

That's pretty much it.

All you need to do in order to get them to be displayed in columns like this, as you would expect on

a modern website.

Now it's also got these great additional properties like, for example, Gap, where you can add in

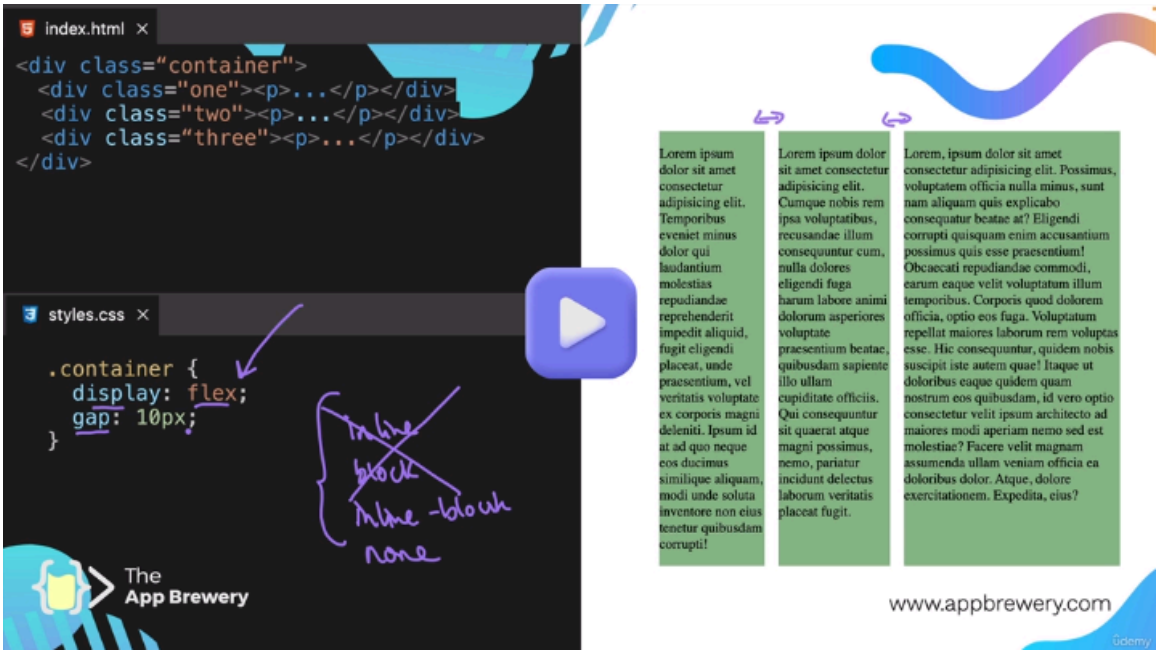
a gap between each of these items

and you're going to see just how flexible and easy it is to use Flexbox to create more complex layouts

on our website.

▼ Q. why flex is set on display property?

Now, one of the things that you might realize at this stage is this flex is set on the display property.



And we know that previously we've looked at display, inline display, block display, inline block

and all of that and display none.

This particular set, I want you to think of it as a completely different system.

When we're thinking about Flexbox, we have to throw all of this out of the window.

That doesn't mean we're not going to use it, because as you see, websites still continue to use these

different things.

But when you make something flex, it no longer abides by any of these rules.

Instead, it's a different system with different rules.

And we have to learn about it and think about it as a separate entity.

And essentially what happens when we declare something display flex is we give a little bit of the control

to Flexbox to lay out our content in a reasonable and commonly desired way.

But we have all of these different modifiers in order to tweak it, to have the exact layout that we

want.

▼ `Display: inline-flex` Property

By default, when you declare a container to be display flex, it's going to create a container that

is going to be actually more or less like a block element.

It's going to be 100% full width.

But there's also this thing where instead of setting the container to be display flex, you can actually

set it to be display inline, flex and similarly to the inline block.

What it does is it allows this Flexbox to occupy as much space as it needs, but it means that other

things can also go and occupy that same line.

So this is just two different versions of creating a Flexbox container.

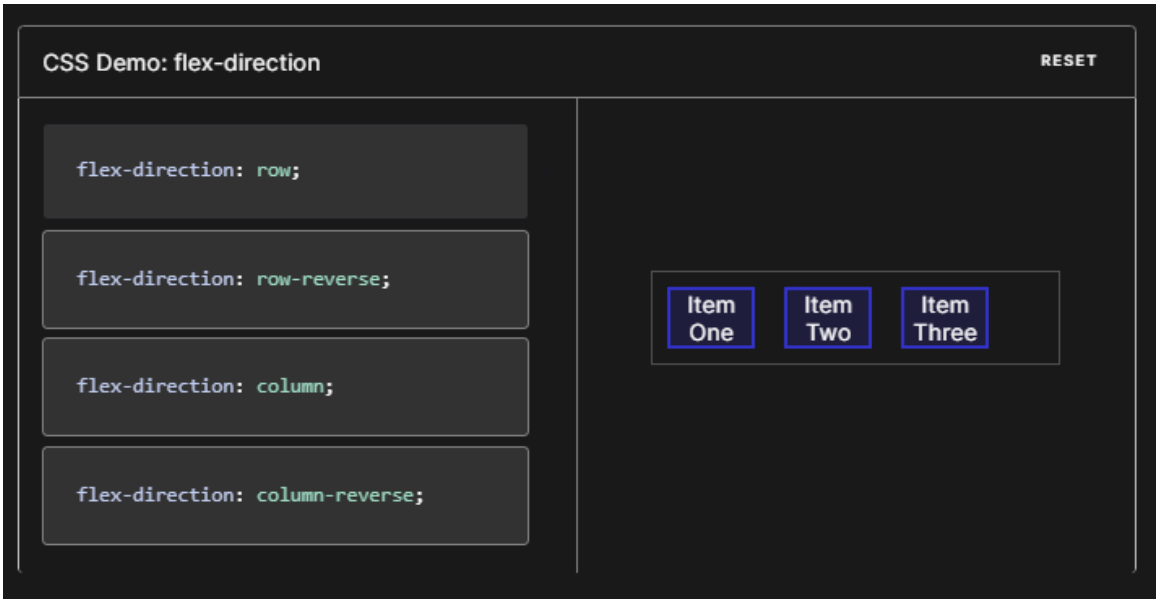
▼ `flex-direction` Property

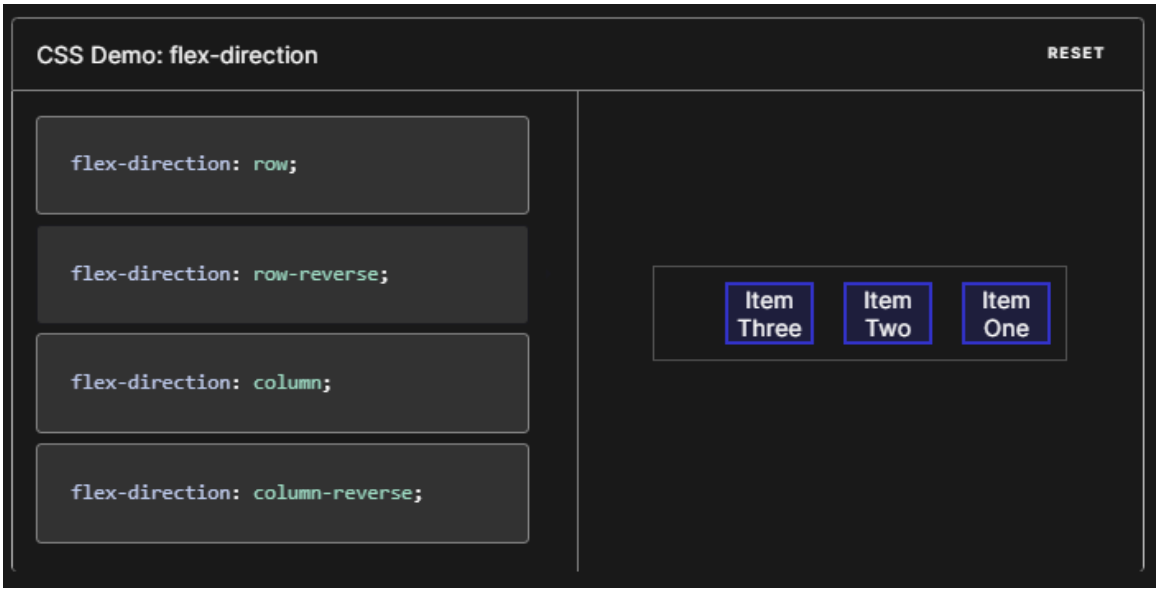
The flex-direction property is used to set the direction of the flexible items inside the flex container. its default value is row (left-to-right, top-to-bottom).

The other possible values are:

- row-reverse
- column
- column-reverse

Examples:-




CSS Demo: flex-direction
RESET
flex-direction: row;
flex-direction: row-reverse;
flex-direction: column;
flex-direction: column-reverse;
Item Three
Item Two
Item One

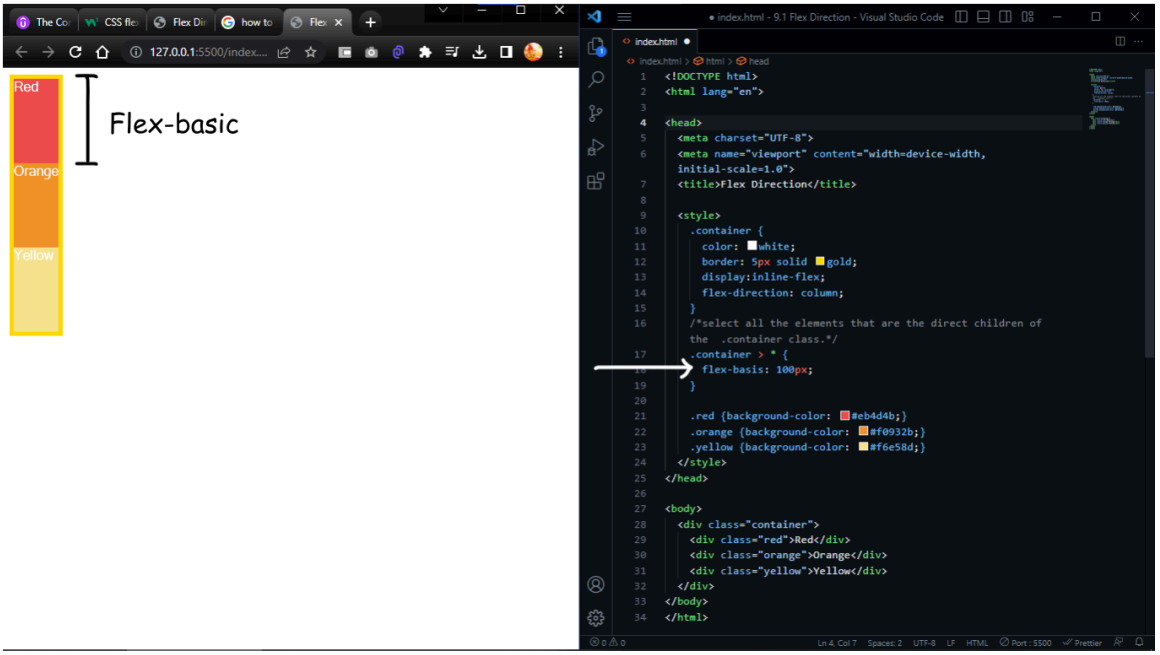

CSS Demo: flex-direction
RESET
flex-direction: row;
flex-direction: row-reverse;
flex-direction: column;
flex-direction: column-reverse;
Item One
Item Two
Item Three


CSS Demo: flex-direction
RESET
flex-direction: row;
flex-direction: row-reverse;
flex-direction: column;
flex-direction: column-reverse;
Item Three
Item Two
Item One

▼ `flex-basis` Property

The flex-basis CSS property **sets the initial main size of a flex item**. It sets the size of the content box unless otherwise set with box-sizing .

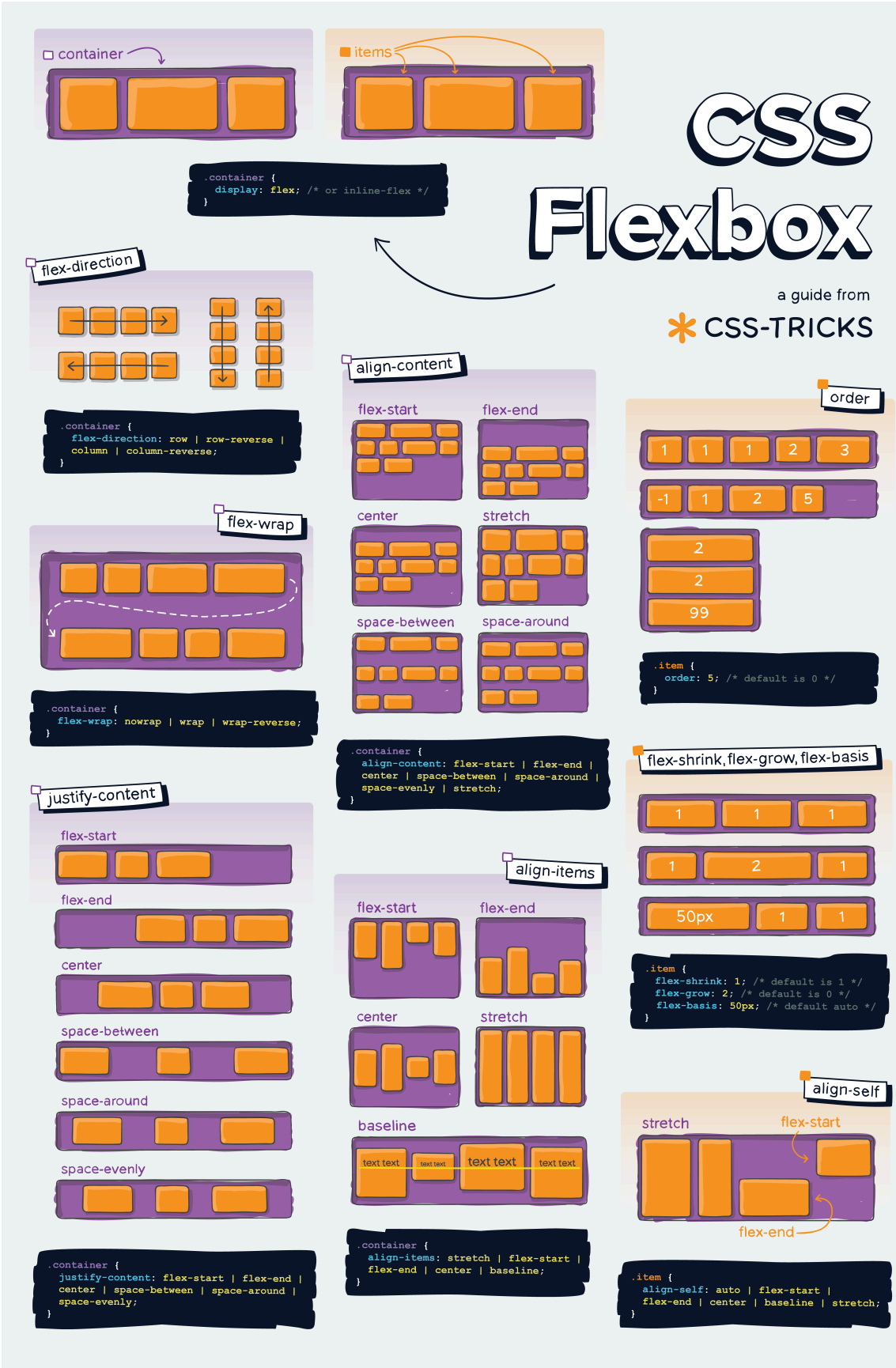
Example:-



▼ Flex Layout

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

POSTER:-


container
items
CSS
Flexbox
a guide from
CSS-TRICKS
.container {
display: flex; /* or inline-flex */
}
flex-direction
.container {
flex-direction: row | row-reverse |
column | column-reverse;
}
flex-wrap
.container {
flex-wrap: nowrap | wrap | wrap-reverse;
}
justify-content
flex-start
flex-end
center
space-between
space-around
space-evenly
.container {
justify-content: flex-start | flex-end |
center | space-between | space-around |
space-evenly;
}
align-content
flex-start
flex-end
center
stretch
space-between
space-around
.container {
align-content: flex-start | flex-end |
center | space-between | space-around |
space-evenly | stretch;
}
align-items
flex-start
flex-end
center
stretch
baseline
text text
.container {
align-items: stretch | flex-start |
flex-end | center | baseline;
}
order
1 1 1 2 3
-1 1 2 5
2
2
99
.item {
order: 5; /* default is 0 */
}
flex-shrink, flex-grow, flex-basis
1 1 1
1 2 1
50px 1 1
.item {
flex-shrink: 1; /* default is 1 */
flex-grow: 2; /* default is 0 */
flex-basis: 50px; /* default auto */
}
align-self
stretch
flex-start
flex-end
.item {
align-self: auto | flex-start |
flex-end | center | baseline | stretch;
}

css-flexbox-poster.png

Example:-

Website link:-https://appbrewery.github.io/flex-layout/

https://appbrewery.github.io/flex-layout/

▼ Flex Sizing

Overview:-

- `flex-grow`
- `flex-shrink`
- `flex-basis`

Udemy video:-

https://www.udemy.com/course/the-complete-web-development-bootcamp/learn/lecture/37368390?start=150#learning-tools

MDN link:-

Controlling ratios of flex items along the main axis - CSS: Cascading Style Sheets | MDN

In this guide we will be exploring the three properties that are applied to flex items, which enable us to control the size and flexibility of the items along the main axis — flex-grow, flex-shrink, and flex-basis. Fully understanding how these properties work with growing and shrinking items is the real

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

CSS Grid

▼ `CSS Grid` VS `CSS Flexbox`, when to use?

🌐 When to use Flexbox and when to use CSS Grid...

https://blog.logrocket.com/css-flexbox-vs-css-grid/

Website link:-https://blog.logrocket.com/css-flexbox-vs-css-grid/

▼ Grid Sizing

Overview:-

- Fixed Size
- Auto Size
- Fractional Size (fr)
- Min Max Size
- Repeat
- Test
- Developer’s tool (Layout)

Udemy video:-

https://www.udemy.com/course/the-complete-web-development-bootcamp/learn/lecture/37368474?start=375#learning-tools

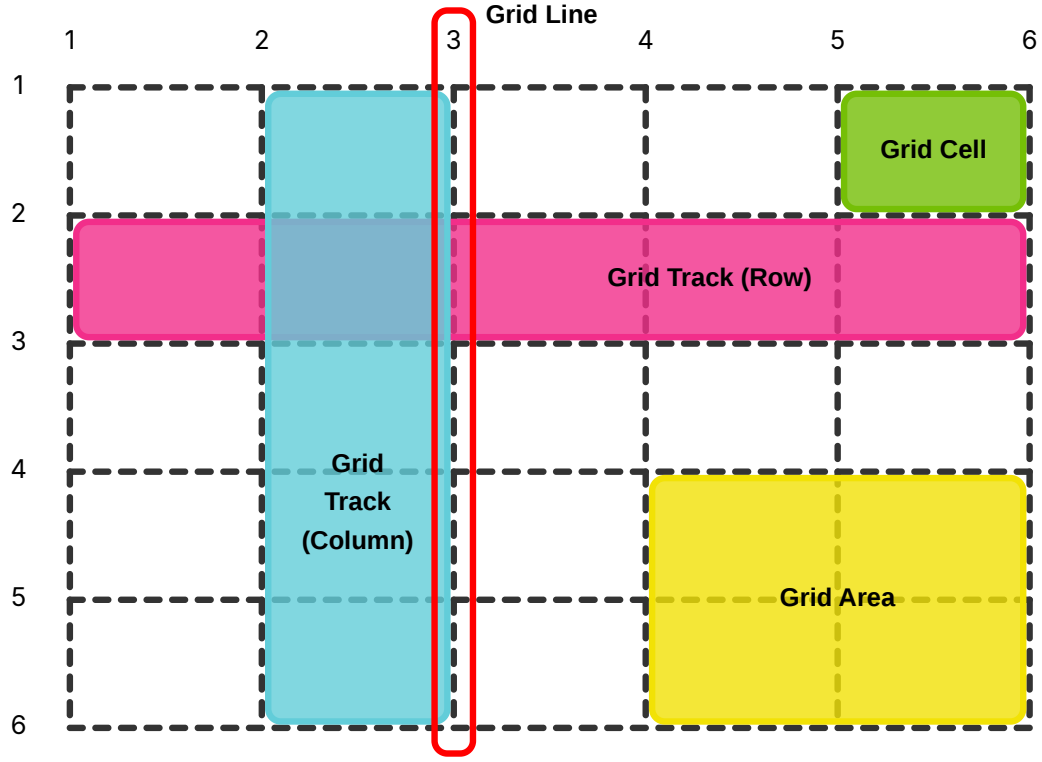
Demo website for CSS grid sizing practice:-

Grid Fundamentals

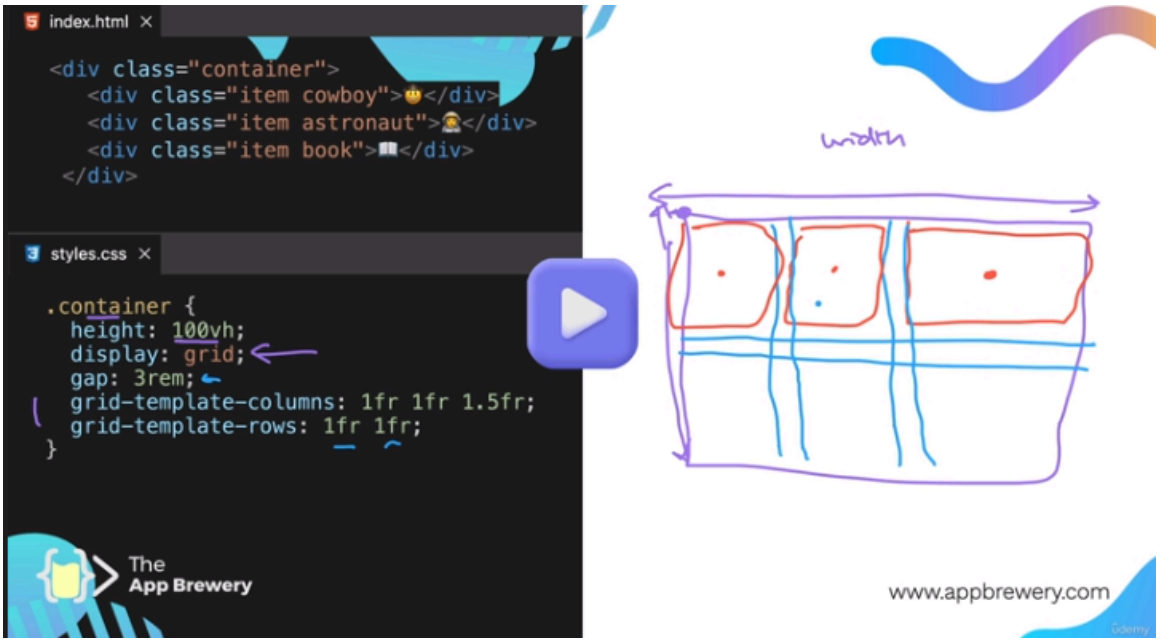
https://appbrewery.github.io/grid-sizing/

▼ Grid Item Placement

Grid-fundamental-concepts:-



Let's try and create that layout using actual code.



I've created a grid container and inside I've placed three grid items by creating three divs.

Now I'm going to update the CSS code for that grid container.

Make it use grid by setting display to grid.

In addition, I'm setting the height to 100 viewport height.

Remember that by default a div is going to span the entire width of the window, but it's only going

to occupy as much vertical space as there is content.

But by setting the height of our container to 100 viewport height, then it's also going to take that

container and make it stretch all the way down to the bottom of the window.

So now that we've got our grid container, we can create our tracks, and the tracks, if you remember,

are created using the grid template properties.

We're saying that there should be three columns.

The first two are about the same height as each other and the third one is a little bit wider.

And then we've said that there should be two rows of equal height so something like this would get created.

So we've got our tracks, we've got our lines, we've got our container.

Now the items get placed in by default.

It will start from the first and it will go into each of the cells that are available.

This cowboy will go into here.

The next one the astronaut will go into here and the book will take up the final cell.

This is before we specified any sort of positioning.

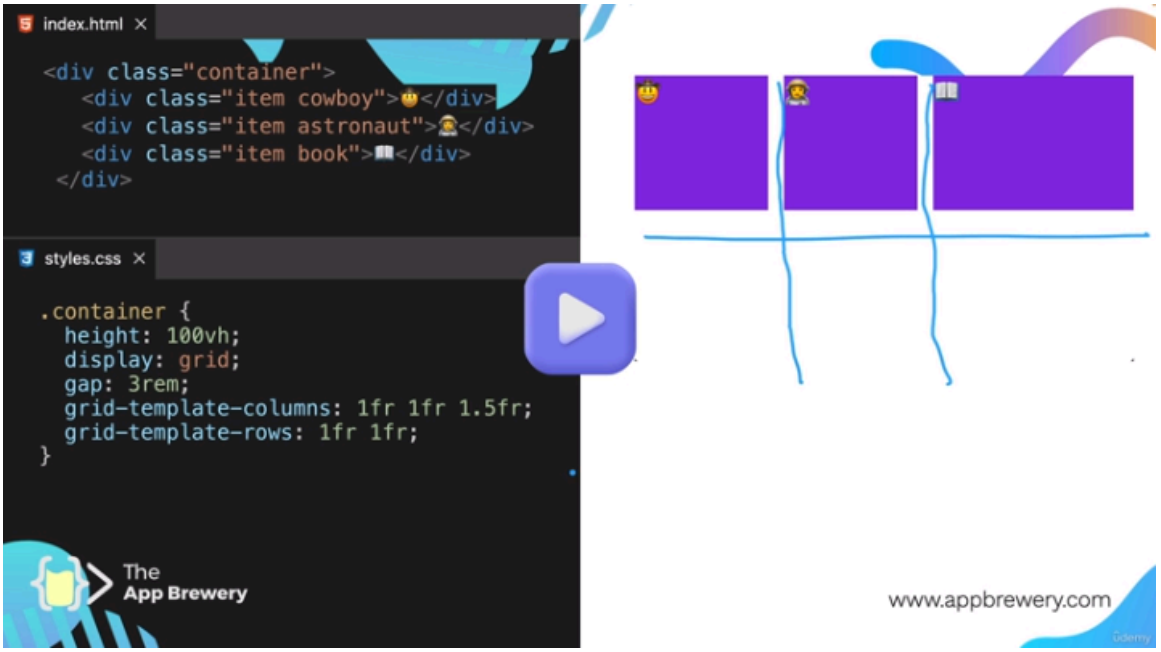
And the final thing that we've updated here is the gap.

We've said that this should be a three REM gap.

We're saying that each of these lines should be three REM thick, and that means that we don't have

to use padding or margin to separate the items in our grid.

And when we run all of this code, this is what we will see.



You'll notice the three REM gap between each of the grid items.

You'll notice the ratio of the width of our column 1 to 1 to 1.5.

But you'll also notice that there's actually a whole track down here that's unoccupied because as I

mentioned, by default, our grid is going to lay out our items into the unoccupied cells, one item

per cell.

Udemy video link:-

https://www.udemy.com/course/the-complete-web-development-bootcamp/learn/lecture/37368524#learning-tools

External Frameworks e.g. Bootstrap

▼ Overview

Now the very last method of creating responsiveness that I want to talk about is using the bootstrap

framework.

And the reason why it's called a framework is because it's external.

So it's not something that's built into CSS like Grid or Flexbox, but it's actually something that

we have to bring in so other people's code that we're going to use.

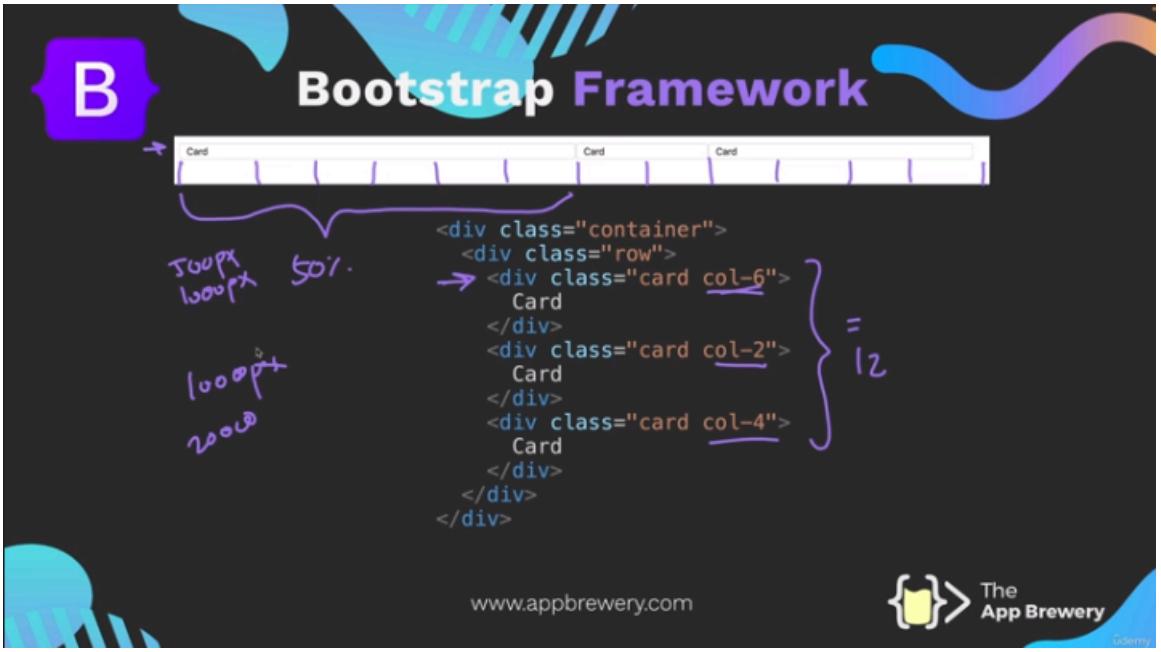
And essentially what they've done is they've defined a whole bunch of classes in their CSS.

So they've written an extensive amounts of CSS and they've put different rules for different selectors.

For example, you could have something called card.

You could have something called Row and you could have something called container.

And these all will point to some CSS code inside the bootstrap container, which has predefined layout.



For example, here I've got three cards and notice how they've all got rounded corners and how they

have a specific type of font and all of that styling is applied automatically to your code by simply

putting in the bootstrap framework and then adding a particular class name.

Now in terms of responsiveness, they also have a sort of flex system because bootstrap is in fact built

on top of Flexbox.

Now, this system is a 12 division system.

So let me explain this.

If the full width of your website can be divided into 12 equal portions, so then you have one, two,

three.

One, two, three.

One, two, three.

Now, if you count these, you'll find that there are 12 little sections and they divide up the entire

website.

Now you can say if I want a particular div that is going to take up half of the width, then I can use

this class called Col six.

So column dash six and it's basically going to take up six of these columns right here.

So six of these little sections.

And as you can see, this first card indeed does take up six of the sections.

Now I can create other cards in here.

So this one's going to take up two of the columns and this one's going to take up four of the columns

and added together they add up to 12, which means that it's going to take up the entire width.

So we've now got three cards taking up the entire width of the website.

But as we resize the website, then it's going to respond.

So a column six is going to be a 50% width.

If the width was 1000 pixels, then this one is going to be 500 pixels wide.

If the width was 2000 pixels, then the width will be 1000 pixels.

ET cetera.

And this is not only really good for responsiveness, but it's also really good because I mentioned

we can get styling brought into our website really easily just by using some of the pre-built components

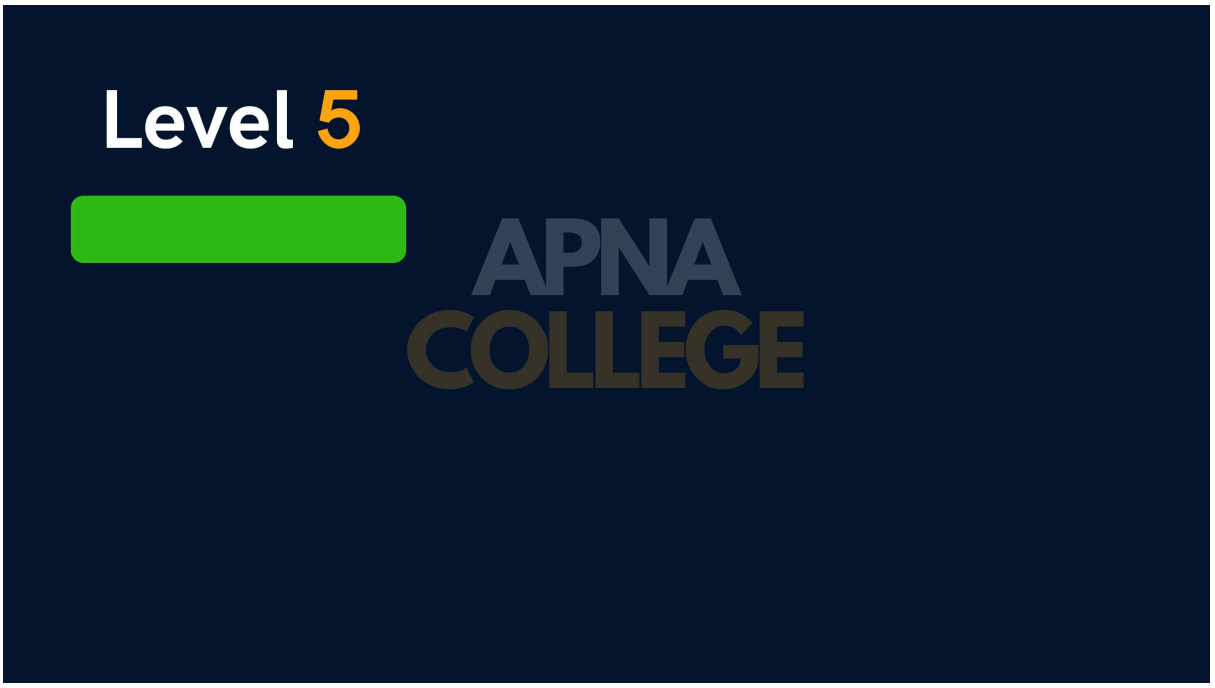
that comes with this framework similar to Grid Flexbox.

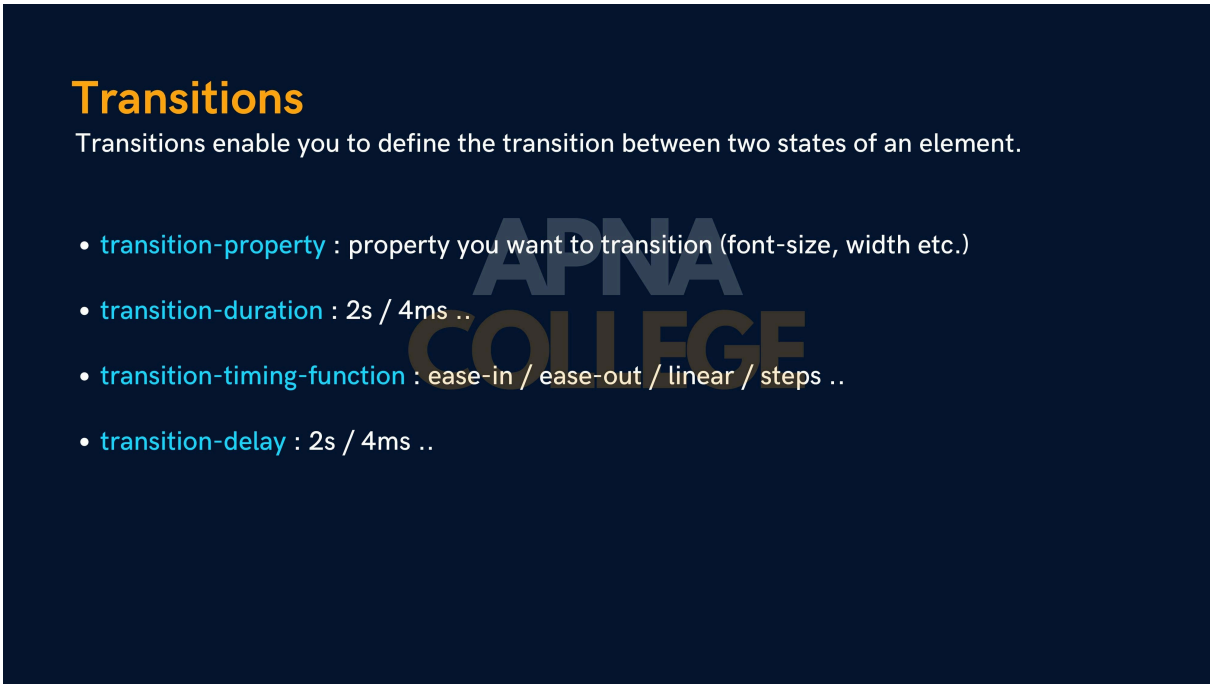
We're going to be doing a section on Bootstrap as well so that you can learn more about this system

and all of the features that are available to you once you understand how it works.

Advanced CSS:- Transitions/Transform/Animation...

https://youtu.be/ESnrn1kAD4E?t=17252





Transition Shorthand

property name | duration | timing-function | delay

```
transition: font-size 2s ease-in-out 0.2s;
```


CSS Transform

Used to apply 2D & 3D transformations to an element

- rotate

```
transform: rotate(45deg);
```

CSS Transform

- scale

```
transform: scale(2);
transform: scale(0.5);
transform: scale(1, 2);

transform: scaleX(0.5);
transform: scaleY(0.5);
```


CSS Transform

- translate

```
transform: translate(20px);
transform: translate(20px, 50px);

transform: translateX(20px);
transform: translateY(20px);
```

CSS Transform

- skew

```
transform: skew (30deg);
```


Animation

To animate CSS elements

```
@keyframe myName {
    from { font-size : 20px; }
    to { font-size : 40px; }
}
```

Animation Properties

- animation-name
- animation-duration
- animation-timing-function
- animation-delay
- animation-iteration-count
- animation-direction

Animation Shorthand

animation : myName 2s linear 3s infinite normal

% in Animation

```
@keyframe myName {
    0% { font-size : 20px; }
    50% { font-size : 30px; }
    100% { font-size : 40px; }
}
```

Go on top👆