
Web Developer Bootcamp

Internet: global network of networks, infrastructure that carries. It enables www. Wwww is a information system where documents and other resources are available over the internet. Resources are identified by Url.

Documents are transferred via Http

front end: focus on the tools that runs on the browser.

Back end: runs on a server.

<https://developer.mozilla.org/en-US/docs/Web>

html entity codes to use {>,&,spades,integration}

All look like `&<Some-char>`; link:-[html entity code](#)

Semantic markup: [Semanitic markup](#)

Its adding meaning to the markup.

when drawing ur page, dont use meaningless markup

e.g use `<main>` instead of `<div>`, does the same exact thing, but also tells us this is the main content.

`<section>`, `<nav>` for navigation links, `<footer>`

Some of the benefits from writing semantic markup are as follows:

- Search engines will consider its contents as important keywords to influence the page's search rankings (see [SEO](#))
- Screen readers can use it as a signpost to help visually impaired users navigate a page
- Finding blocks of meaningful code is significantly easier than searching through endless `divs` with or without semantic or namespaced classes
- Suggests to the developer the type of data that will be populated
- Semantic naming mirrors proper custom element/component naming

When approaching which markup to use, ask yourself, "What element(s) best describe/represent the data that I'm going to populate?" For example, is it a list of data?; ordered, unordered?; is it an article with sections and an aside of related information?; does it list out definitions?; is it a figure or image that needs a caption?; should it have a header and a footer in addition to the global site-wide header and footer?; etc.

`<nav>`:- navigation links could be internal links or external resource.

`<article>`:- self independent composition which is intended to be highly distributed, e.g in syndication

`<aside>`:- not essential or maybe indirectly related. so when u using ur cellphone that's the part that disappears

`<time>`, `<figure>`....

Emmet [Emmet](#) :

built in vs-code

Tables:

```
<table><thead><tr><td><tbody><tr><td>
```

Forms: The action attribute specifies where the data to be sent. This attribute specifies which http method to be used.

Html5 input types [Html5 input types](#)

```
<input type='email' >
```

You can also use the `multiple` attribute in combination with the `email` input type to allow several email addresses to be entered in the same input (separated by commas):

```
<input type='tel'> //telephone dialpad
```

```
<input type='url'>
```

```
<input type='number'>
```

slider:

```
<input type='range'>
```

problem with slider: doesn't offer what the current value is

That's why, `<output>` it works similar as label.

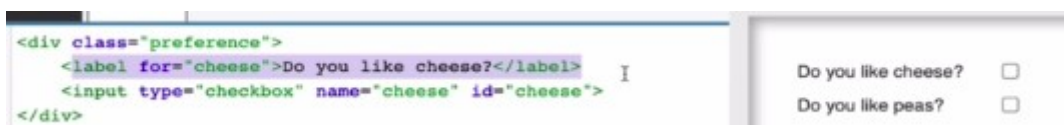
It take a `for-attribute` that allows you to associate with elements that the input value came from.

Date and time pickers:

```
<input type='datetime-local'> //no specific time zone
```

others are month,time,week.

Use label like,



If u click on the label on the webpage, the associated elements will get focused.

If u have button in a form, its always gonna submit by default.

That is: `<button></button>`

which is same as `<button type='submit'></button>`

So, for ur button to perform as a button only

Then: `<button type='button'></button>`

Other way is `<input type='button' value='click me'>`

when you submit data of a form,you'll see the data in the url, like:

url?q='youtube'

If multiple terms then url?name=joey,password=food

You group radiobuttons by giving them all same name;

If all the radiobutton in a group doesnt have a **value-attribute**,so after clicking any, it will send groupname=on. So use **value-attribute**.To send data associated to that radiobutton.Use **id-attribute** to identify the radiobutton.

Dropdown:

<select> <option>what user sees</option> </select>

color:<https://htmlcolorcodes.com/>

hexadecimal color:-#** ** *

| * belongs to {1-9}+{A-F}

e.g #c5e is same as #cc55ee

Properties

color:green

color:rgb()

background-color:yellow

font-family:"helvetica","sans-serif"

font-size:11px{even the headings got smaller cuz heading are relative unit "em" }

font-weight:bold

font-style:italic

OR

font:italic 13px fantasy

line-height:1.5em {spacing between line}

text-align:center

text-decoration:underline

width:300px

height:70%

overflow:visible,hidden,auto(adds a scrollbar)

overflow-x:hidden

margin:top right bottom left

margin:auto (center;equal margin on either side)

border:thickness(px) style(solid) color

border-top:10px solid purple

letter-spacing:2px

word-spacing:4px

rgba:(255,255,255,0.7)

hsl([0-360] ,50%,60%)

<!--first param:- color, second param:-
saturation , third param:- lightness

Css text properties [Css text properties](#)

line-height:2em // you'll see when you highlight the text on the page, the highlight extend far beyond the text,the height of the line.

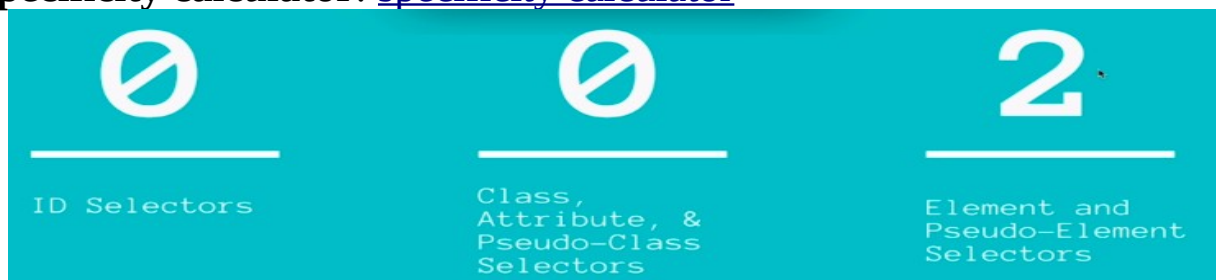
| Relative | Absolute |
|---|--|
| <ul style="list-style-type: none">- EM- REM- VH- VW- %- AND MORE ! | <ul style="list-style-type: none">- PX- PT- CM- IN- MM |

Color palette: [color palette](#)

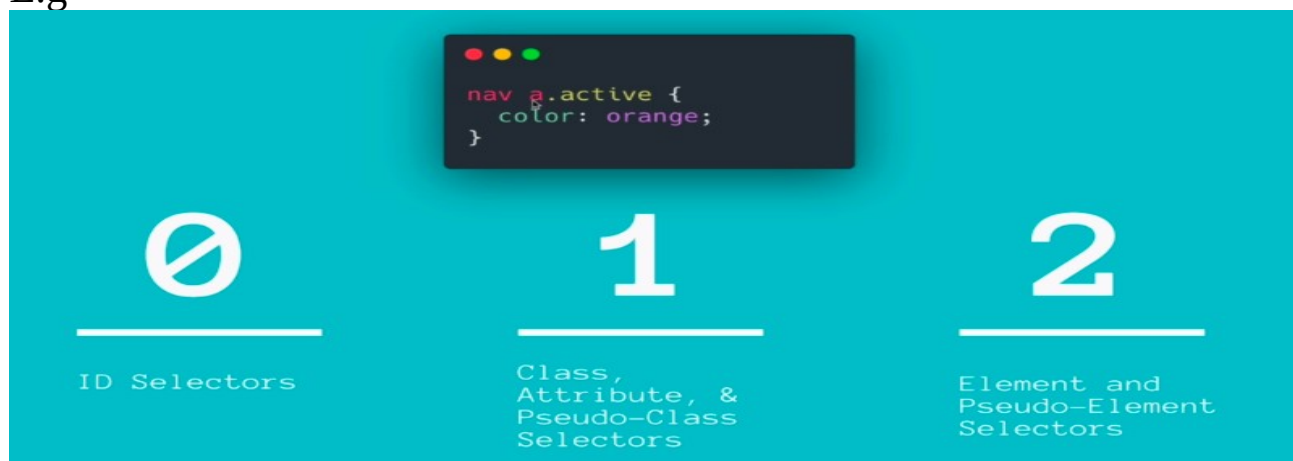
Controls: use alt+click to write multiple places in Vs.

Css specificity : [Css specificity](#)

Specificity calculator: [specificity calculator](#)



E.g



```
selector{  
    color: green !important;           //this beats all even inline properties  
}
```

Css selectors [30 Css selectors](#) [MDN selectors](#)

combinator selectors

| | |
|---|--|
| #no1 | id |
| .group1 | class |
| .group1.group2 | select elements having both classes. |
| section[class='group1'] | every section where class='group1' |
| section.group1 | every section where class='group1' |
| section:hover div | select the section being hovered and apply properties on the descendant div tag. |
| section:first-of-type div:nth-of-type(2) | |
| h1,h2 | |
| li a | //descendant selector |
| ul+p | //right next to it, together . Applied to <p> |
| #container1>ul | //direct children |
| ul~p | //similar to ul+p,less strict,just have2 be followed by ul.Applied to <p> |
| x[title] e.g:-a[title] | // anchor tag with an attribute of title. |
| x[href='some-url'] | |
| x[href^='http'] | |
| x[href*='udemy'] | |
| x[href\$='.com'] | |
| x:checked e.g:- input[type=radio]:checked | |
| 15,18 | |
| x:not(selector) e.g div:not(#container1) | // all div except the one which has id #contianer. |
| X::firstline | |
| X::firstletter | |
| li:nth-child(3) | |
| li:nth-child(3n) | <!-- for (3,3*3,3*3*3,.....) |
| li:nth-last-child(3) | |
| X:nth-of-type(4) | // imagine if we have 5 ul, we wanna select the 3 ul. |
| X:nth-last-of-type(n) | |
| ol:first-child | // only the first child |
| ul:last-child | //only the last child |
| X:only-child | |
| X:only-of-type | |
| e.g | ul > li:only-of-type //target elements that do not have siblings |

within it's parent container,lets target all ul's which have only a single list item.

X:first-of-type

X:first-of-type //first sibling of its type,cuz it could be nested by a descendant selector.

PSEUDO CLASSES

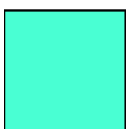
keyword added to a selector that specifies a special state of the selected element(s)

- :active
- :checked
- :first
- :first-child
- :hover
- :not()
- :nth-child()
- :nth-of-type()

PSEUDO ELEMENTS

Keyword added to a selector that lets you style a particular part of selected element(s)

- ::after
- ::before
- ::first-letter



Lorem ipsum dolor, sit amet consectetur adipisicing elit. Officiis dicta nam labore accusamus sunt temporibus!



--->What the user highlights

X:nth child

```
div:nth-child(2),li:nth-child(2){  
    background-color: green;  
}
```

X:nth-child

-----> It's applied for all separate

- My names are
1. kazim
2. sahil
3. kaju
4. kaju katri

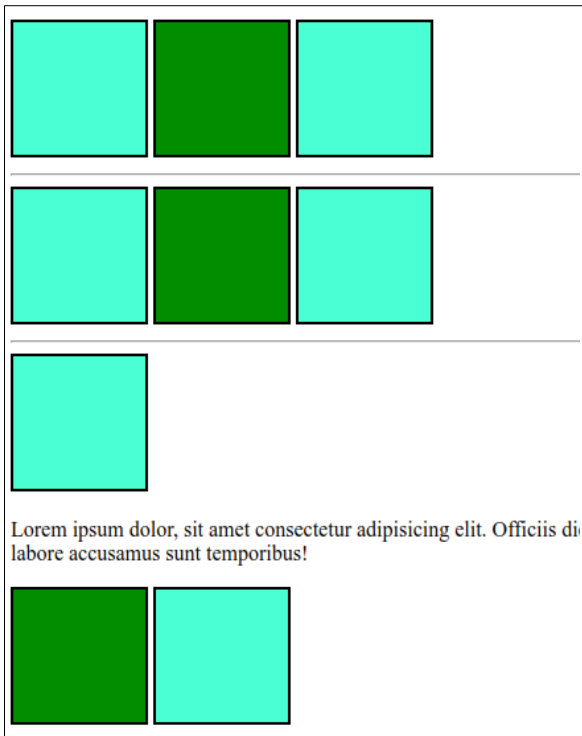
- I have my homes at
1. yari-road
2. malad

interesting.it works for only.

Another example

dnt work<---nested stack

X:nth-of-type



Here, it doesn't have to be a stack of consecutive elements of same type. Another example, nested works here too



- My names are
 1. kazim
 2. sahil
 3. kaju
 4. kaju katri
- I have my homes at
 1. yari-road
 2. malad

Css Box Model [Css box model](#)

Border:- also use for bringing attention to certain elements on the page. Like if i hover, they tell us that the element is clickable or selectable to interact.

Border properties: [border properties](#)

Border Properties (the important ones)		
BORDER-WIDTH	BORDER-COLOR	BORDER-STYLE
Controls the thickness of the border.	Controls the...color of the border	Controls the line style - dashed, solid, etc.

Lorem ipsum dolor sit, amet consectetur adipisicing elit. Impedit magni pariatur, odio voluptates esse veniam. Quos omnis, necessitatibus beatae architecto sint inventore dicta vero magni. Dicta rem ipsam accusamus atque.

Lorem ipsum dolor sit, amet consectetur adipisicing elit. Impedit magni pariatur, odio voluptates esse veniam. Quos omnis, necessitatibus beatae

Here the width is:
 $200\text{px} + 2 * \text{border-thickness}$.
To make sure the box object take the space specified i.e 200px.


Use property,
`box-sizing: border-box;`

`border-width: 5px;`

`border-color: green;`

`border-style: dashed groove dotted none` //different style to all 4 sides

`border-radius: 15% 20% 12% 10%` //rounds the corner

`border-width: 10px 15px`  `left and right`
`top & bottom`

Display Property (our first encounter)		
INLINE	BLOCK	INLINE - BLOCK
Width & Height are ignored. Margin & padding push elements away horizontally but not vertically.	Block elements break the flow of a document. Width, Height, Margin, & Padding are respected.	Behaved like an inline element except Width, Height, Margin, & Padding are respected

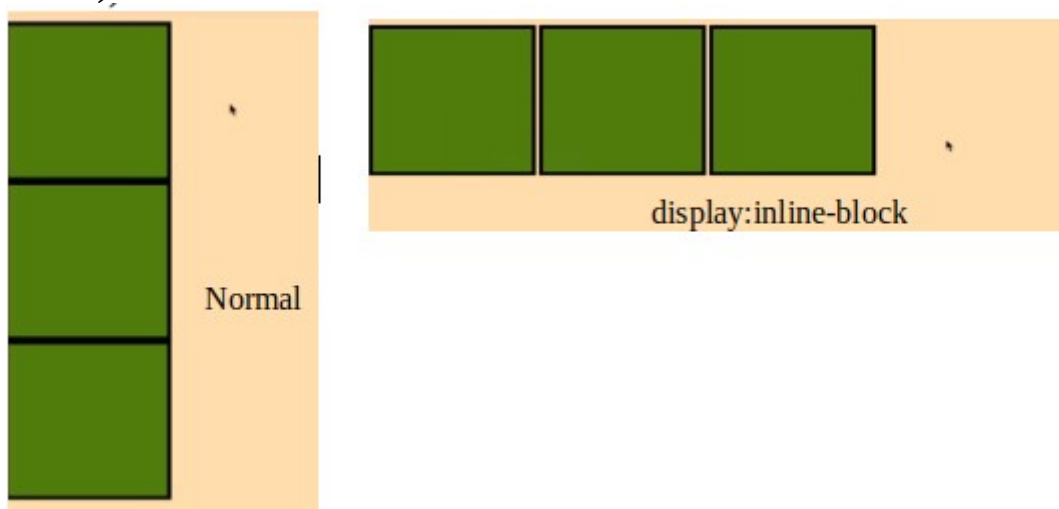
`span{display:inline; }` <!-- imagine `<p> </p>` which makes sense cuz its an inline element. And only dimension it can push(M+P) is horizontally.

`span{display:block}` <!-- turn an inline element into a block element.
 Inline element:fits the data.

Block elements:follows the dimension(Width+height)
 so if u have a three green block (50px width and 50px height)literally and visually.

If you use `display:inline` on this, it will ignored the (width+height).

So use `display:inline-block`, it will tell dress like you normally would (as a block) and make it inline.



`display:none`

Relative units [Relative unit](#)

percentages



PERCENTAGES ARE ALWAYS RELATIVE TO SOME OTHER VALUE

Sometimes, it's a value from the parent and other times it's a value from the element itself.

width: 50% - half the width of the parent

line-height: 50% ● half the font-size of the element itself

em



EM'S ARE RELATIVE UNITS

With font-size, 1em equals the font-size of the parent. 2em's is twice the font-size of the parent, etc.

With other properties, 1em is equal to the computed font-size of the element itself.

e.g

```
h2{
```

```
font-size:2em;
```

```
margin-left:1em;           //i.e 1em=1* 2em
```

```
}
```

e.g

```
body{font-size:12px;}
```

```
article{font-size:14px;}
```

```
p{font-size:1.5em;} <!-- that is 1.5*14px
```

```
<article>><p>
```

e.g

```
section div{
    background-color:magenta;
    font-size:1.5em;
    border-radius:0.5em;
}
```

if u wanna preserve the shape when u scale up and down,use relative unit for all shape-properties.

Rem

ROOT EMS

Relative to the **root html element's** font-size. Often easier to work with.

If the root font-size is 20px, 1 rem is always 20px, 2rem is always 40px, etc.

Often you'll see people mix, people mixing rems and em together. Cuz there are times when you wanna be situationally dependent.

Click Me

Maybe the font-size is based on rem, but the border radius or padding should change depending on the actual font-size computed to be.

Opacity and alpha channel

rgba(255,255,255,0.6)

|△|
alpha

0 -completely transparent <!-- consider not there is 0

1 -not transparent at all <!-- like a wall there is 1

background color:white is same as rgba(255,255,255,1)

opacity:0 <!-- it range from 0 to 1

background:#ffffff00 <!-- that is white transparent

rgba(0, 209, 112, 0.5)

red green blue alpha

alpha:1 is not transparent

alpha:0 is completely transparent

Above you have alpha 0.5 imagine looking through a mountain dew bottle.

Opacity:

```
width: 50%;  
height: 50%;  
background-color: yellow;  
opacity: 0.5;
```

opacity on a div make it how you imagine,

rgba is a property on either background-color or color

```
background-color: #00cca0FF;
```

Above, for hexadecimal: alpha channel ranges from 00 to FF.

Position [position video](#)

In relative, why left:30px moves to the right?

In my brain, I think position as u normally do and then offset by 30px to the left

now, I have to make an assumption,

like, consider the top-left vertex as 0,0 and bottom left vertex as 0,x of our squares

meaning you are in the 4th quadrant while plotting and the left here tells you the axis are to the left(the x and y).

and similarly for right.

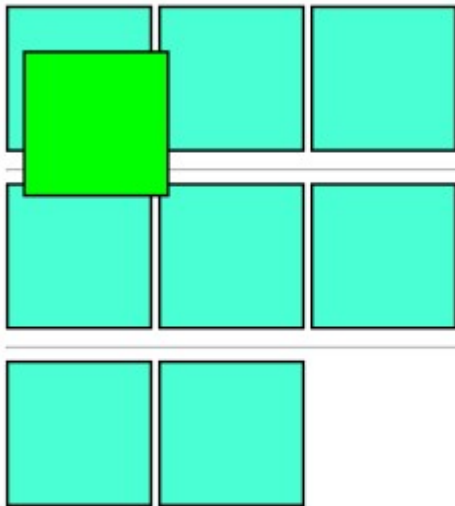
Is it me just feeling this way, also am I interpreting correctly?

it adds 30 px of space to the left of the object, so as a result the object moves to the right

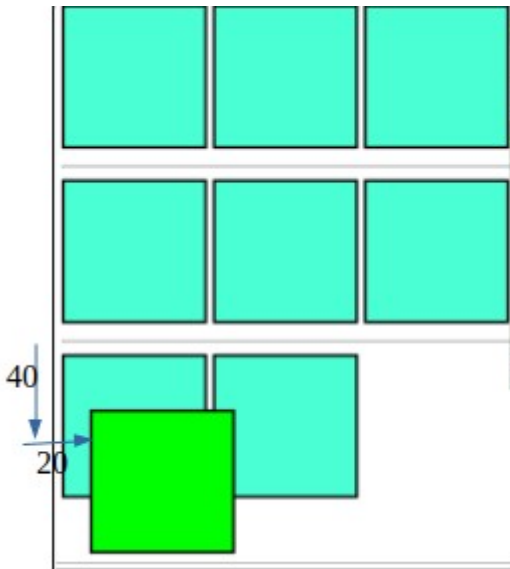
The element is removed from the normal document flow, and no space is created for the element in the page layout. It is positioned relative to its closest positioned ancestor, if any; otherwise, it is placed relative to the initial **containing block**. Its final position is determined by the values of **top**, **right**, **bottom**, and **left**.

This value creates a new **stacking context** when the value of **z-index** is not **auto**. The margins of absolutely positioned boxes do not **collapse** with other margins.

Here, positioned ancestor refers to a element that is postioned anything but static. Else, w.r.t initial containing block i.e the body.



```
div {
  width:100px;
  height:100px;
  background-color: aquamarine;
  border:2px solid black;
  display: inline-block;
}
#id3 div:nth-child(2){
  position: absolute;
  top:40px;
  left:20px;
  background-color: chartreuse;
}
```



```
#id3 {
  position: relative;
}
#id3 div:nth-child(2){
  position: absolute;
  top:40px;
  left:20px;
  background-color: chartreuse;
}
```

position:fixed

The element is removed from the normal document flow, and no space is created for the element in the page layout. It is positioned relative to the initial containing block

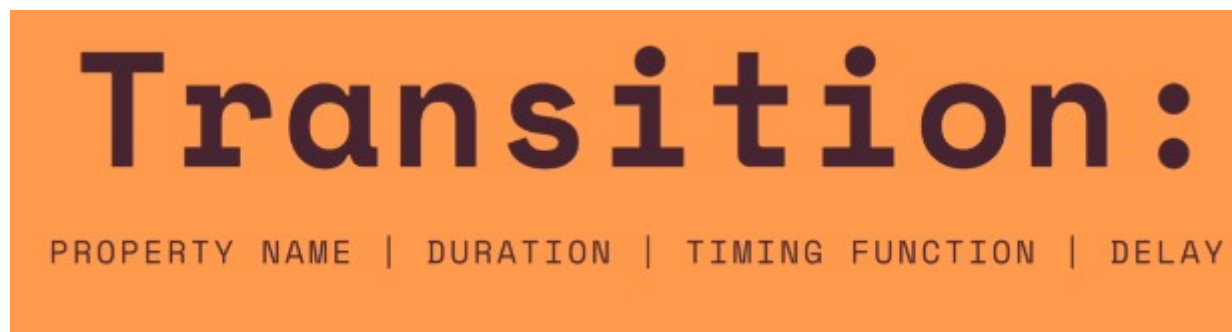
position:sticky

It stay at place when scroll, when the scrollbar hits the top, it moves with the scrollbar. That is, start being not-fixed and get fixed when the scrollbar hit the top.

Transitions [transition](#) [video](#) [myhtml](#)



Transition helps you animate the transition of one property into another property value.

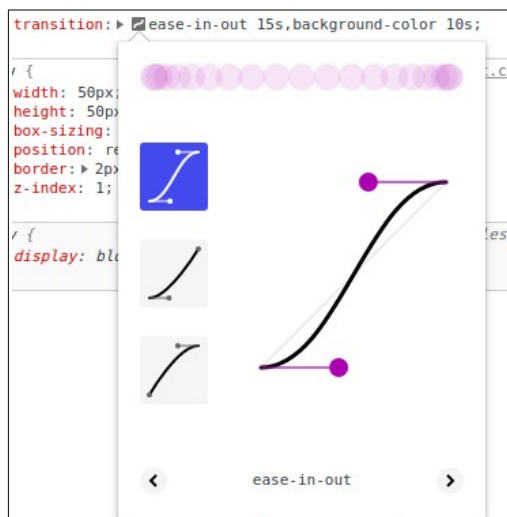


To consider all properties: use **all** keyword

e.g:

transition:background-color 10s,border-radius 2s;

You'll see control when you use ease-in,ease-out



Transform transform-functions

status:-pending

Background [background](#) [video](#)
status:pending

Responsive Css & Flexbox

what is flexbox? >>> [flexbox](#) [flexbox](#)

display property is used on the container.

& evrything below is done to the container rather than individual div's.

display:flex <!-- kinda like inline but with more controls

flex-direction:row <!-- default

flex-direction:row-reverse,column,column-reverse

<!-- note if there is a overflow, flex will change the dimensions to fit in the container -->

justify-content:flex-start <!-- default

justify-content:flex-end,center,space-between,space-around,space-evenly

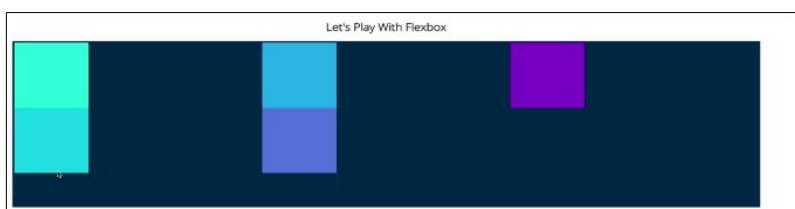
flex-wrap:wrap,wrap-reverse



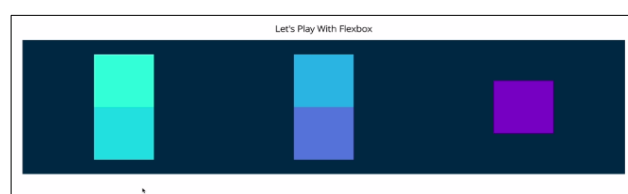
<!-- if there is a overflow, By using flex-wrap ,it will wrap which is based on justify-content

&

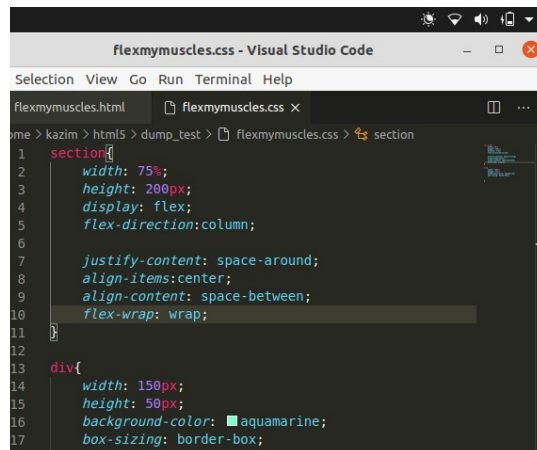
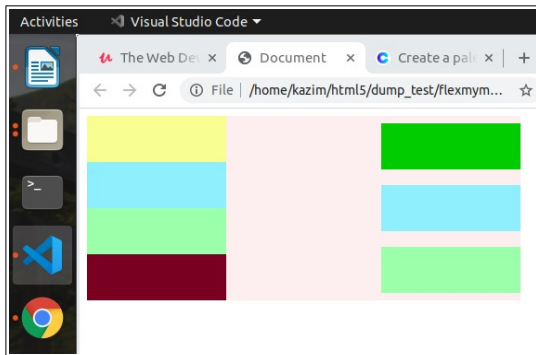
wrap-reverse is like a reflection of wrap where line of reflection is known by flex-direction i.e row or column



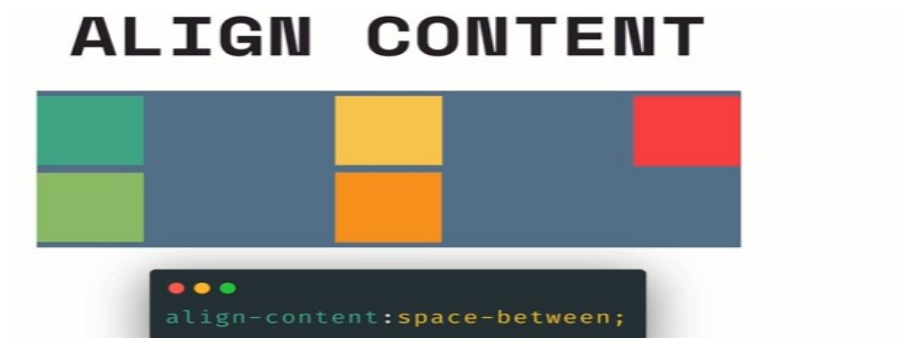
Align-items:baseline,flex-start,flex-end
challenge E.g for **Align items**



justify-content changes w.r.t row and column.



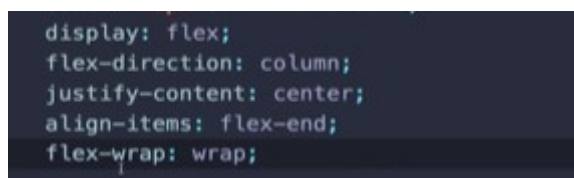
align-self: flex-start <!-- inline css.

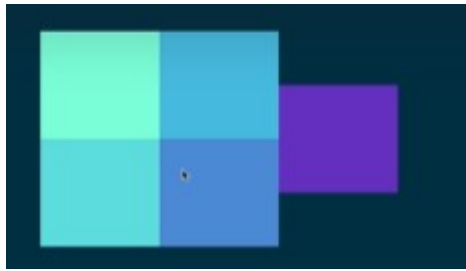


We have align-content use to distribute space between cross axis, but only when we have multiple rows or columns depending on if you r on row or column based layout, basically if we are in columns, align columns controls the space between those columns.

If we are in a row, we are controlling the space between rows.

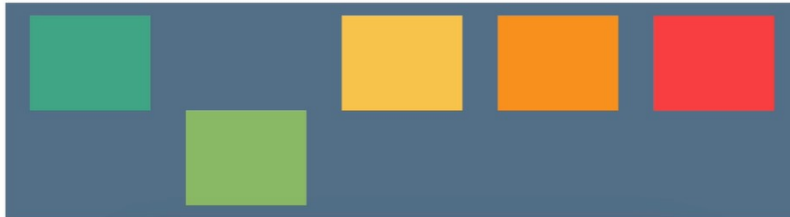
Align -content: space-around, space-between, flex-start, flex-end, center





align-content:center;

ALIGN SELF



```
align-self: flex-end;
```

align self is applied to single element not the flex container.

```
div:nth-of-type(3){  
  align-self: center;  
}
```



Flex basis, grow & shrink

FLEX-BASIS

Defines the initial size of an element before additional space is distributed.

FLEX-GROW

Controls the amount of available space an element should take up.
Accepts a unit-less number value.

FLEX-SHRINK

If items are larger than the container, they shrink according to flex-shrink.

`flex-basis:200px`

is the initial size that is given and it can shrink or grow from there.

if a div in a flex-container, is given width and height.

Flow direction:row ----> width=flex-basis:200px

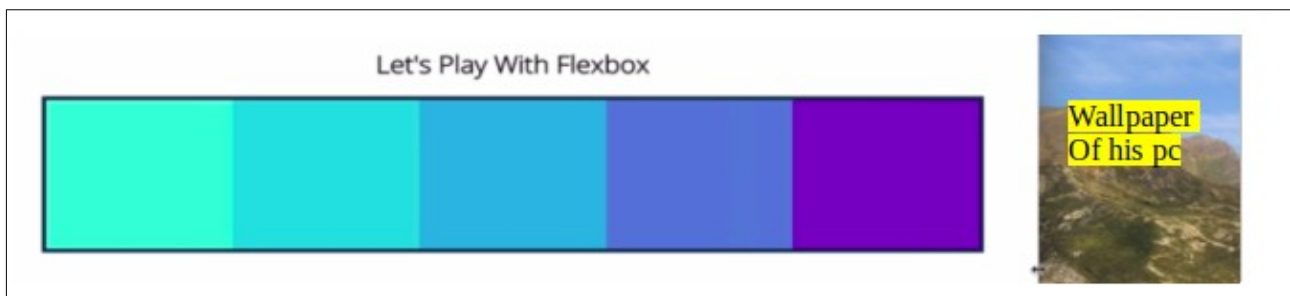
Flow direction:column -----> height=flex-basis:200px

for e.g, if container height isn't specified, it'll take as much height is needed for the div to fit-in.

If specified then, if there isn't enough space it'll overflow.

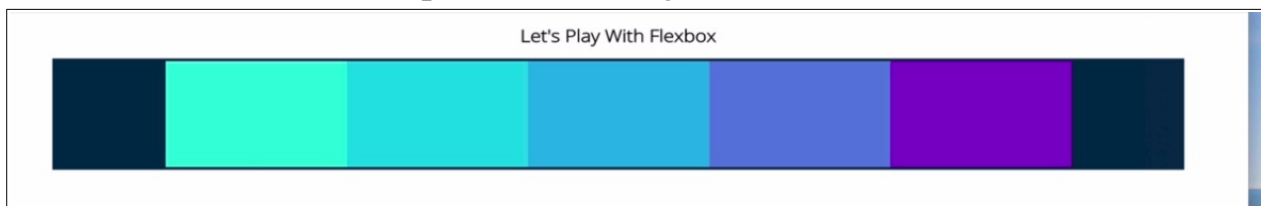
`min-width:200` <!-- Min-width i.e given to div to specify minimum width,
no matter what

`max-width:200` <!-- max width i.e given to div to specify maximum width,
no matter what

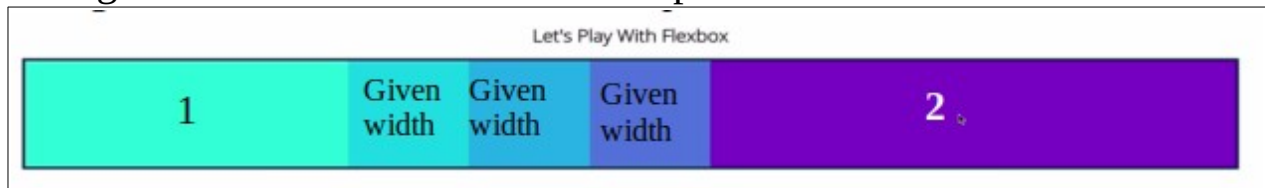


if max width is used. e.g max-width:300px

So when u maximize the window screen using cursor, the div will grow and when it reaches 300px width it stays there.



Flex-grow is used when there is extra space



Here,

whatever space is left

1/3 is given to div-1

2/3 is given to div-5

flex-shrink:1 <!--default given to all

flex-shrink:0 <!-- to maintain it's given dimension

flex-shrink:2,3.. <!-- smaller than others, if there isn't enough space , this1 is the one that will shrink. We aren't telling how many px to shrink, we are saying with relative to others to shrink. for example

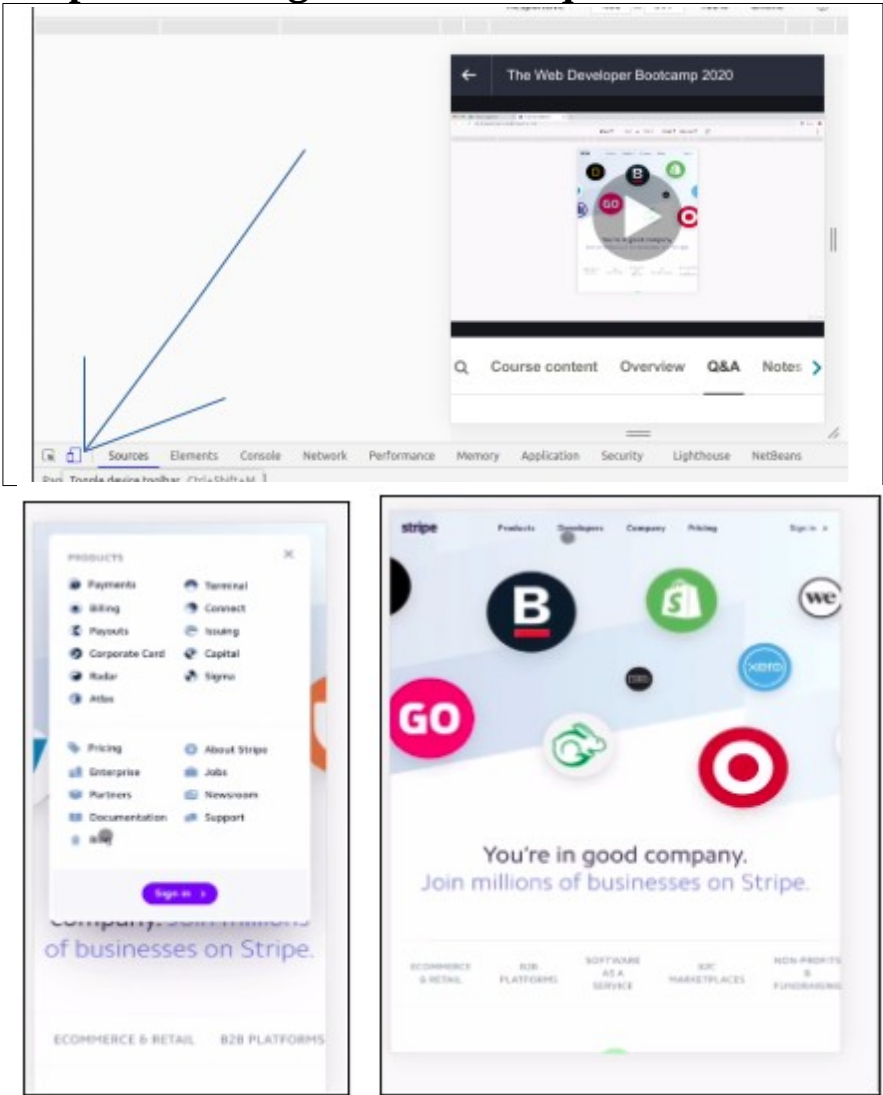


flex shorthand

flex: flex-grow flex-shrink flex-basis

e.g flex:1 0 200px

Responsive design and Media queries Intro



Media queries [Media @media](#)

```
@media (max-width: 800px) {  
  .sidebar {  
    display: none;  
  }  
  
  .main {  
    width: 80%;  
  }  
}  
  
@media (min-width: 30em) and (orientation: landscape) {  
  #container {  
    flex-direction: column;  
    justify-content: center;  
  }  
}
```

```
h1{color:red
```

```
}
```

```
@media (min-width:300px){
```

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

```
}
```

```
@media(min-width:500 and orientation:landscape){
```

```
  h1{color:orange}
```

```
}
```

--pricing panel code along

CSS Reset [CSS Reset](#)

The goal is to normalize how browser styles works across the browser. They give u a clean slate.

Flex- Repeat

```
#container div {  
  width: 200px;  
  height: 200px;  
  text-align:center;  
  flex-basis: 400px;  
}
```



You'll see all div are set to 400px, is along the main-axis. Flex-basis is the initial size that is given, it might be a width or height depending on the flex-direction.

FLEX-GROW

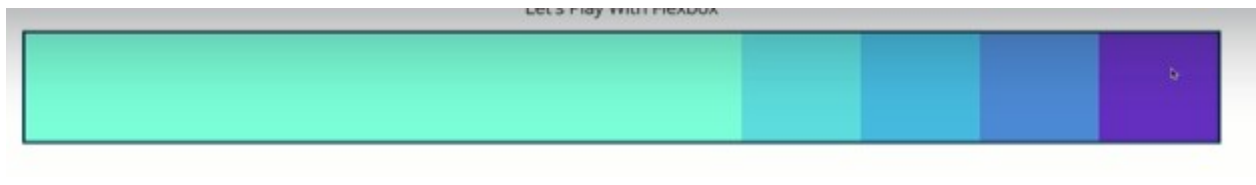
Controls the amount of available space an element should take up.
Accepts a unit-less number value.



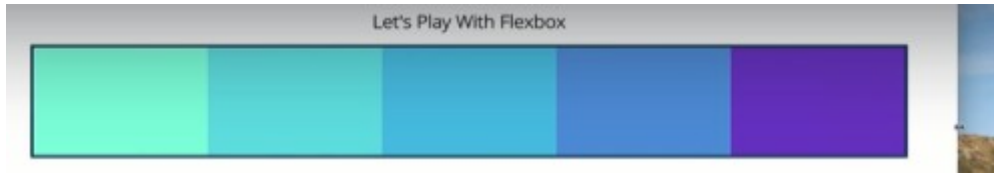
I can assign flex-grow property to any element.

Flex-grow:1

<!-- takes all the available space left-->



`div {flex-grow:1,width:120px}`



They share space equally and responsively.

Above, width is ignored.

But as we spill over



Note:

You can use min-width and max-width to control how big they scale.

Examples:

...

`div:nth-child(1){ flex-grow:1}`

`div:nth-child(5){flex-grow:2}`

They both will eat up space, but the 5th div will take twice as much space to 1st div.

Understand the maths

all div take 100px

remaining space left was 150

`div-5 extra-width=2*(150/3)`

`<!-- divide by 3 cuz 3 slice -->`

`div-5 extra-width=100`

`final div-5 width=100+100=200`

`div-1 extra-width=1*(150/3)`

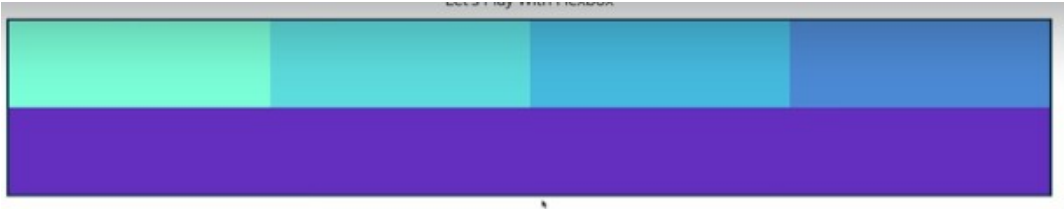
`div-1 extra-width=50`

`final div-1 width=150`

another e.g

```
#container div {  
  width: 200px;  
  height: 200px;  
  /* max-width: 300px; */  
  text-align:center;  
  flex-basis: 500px;  
}
```

if flex-wrap:wrap



Above, violet flex-grow:

flex-shrink:0 <!-- means they not gonna shrink
others have default value:1 <!-- they all shrink at same rate.

```
section div{  
  background-color: cyan;  
  height: 50px;  
  width: 50px;  
  flex-basis: 400px;  
  border: 1px solid black;  
}  
  
section div:nth-child(2){  
  background-color: greenyellow;  
  flex-shrink:2;  
}  
  
section div:nth-child(5){  
  background-color: rgb(255, 21, 165);  
  flex-shrink:0;  
}
```



Flex-shrink governs the rate at which elements shrink when there isn't enough space in the container. Above cyan block width are 50px but it shrink to 25.33, where as pink block maintained his flex-basis width.



all elements are default at flex-shrink:1

another example

```
section div:nth-child(2){
  background-color: greenyellow;
  flex-shrink:1;
}
```



Flex-shorthand

```
/* One value, unitless number: flex-grow */
flex: 2;

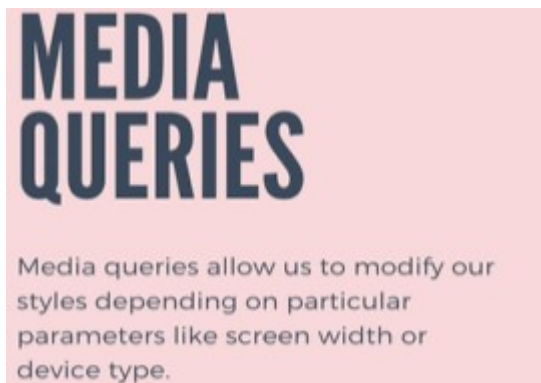
/* One value, width/height: flex-basis */
flex: 10em;
flex: 30%;
flex: min-content;

/* Two values: flex-grow | flex-basis */
flex: 1 30px;

/* Two values: flex-grow | flex-shrink */
flex: 2 2;

/* Three values: flex-grow | flex-shrink | flex-basis */
flex: 2 2 10%;
```

Responsive design and media queries



depending on screen-width, device type, device orientation, landscapevs portrait.

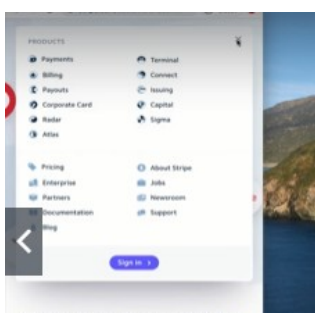
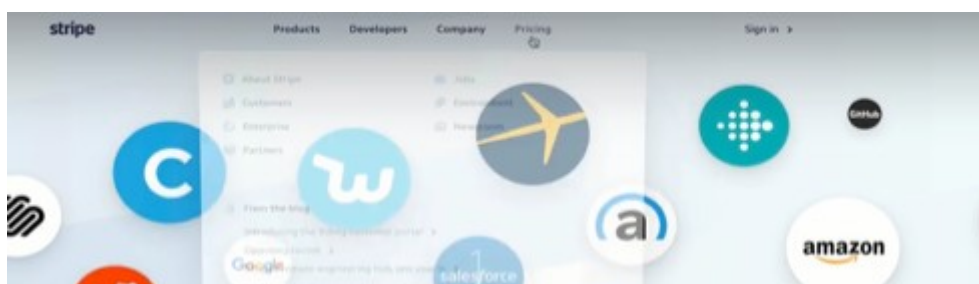
If it's on landscape mode or portrait mode do this, like show it on landscape and hide it on portrait. If we are in super tiny screen, collapse the navbar so it doesn't take too much space.

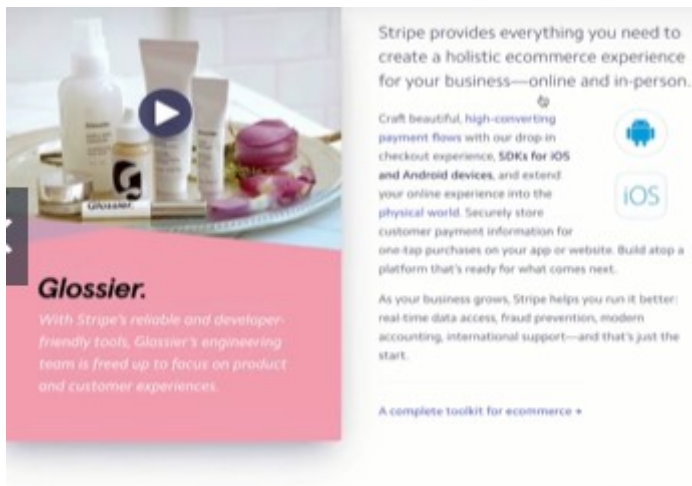


It's pretty large up there

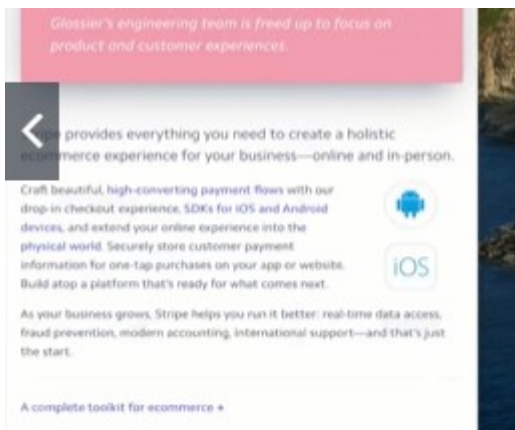


At small it becomes scrollable.





V
V
V



It stacked

Media queries

```
@media (min-width: 600px) and (max-width: 800px){
  h1 {
    color: purple;
  }
}
```

```

@media (max-width: 500px){
  h1 {
    color: red;
  }
}

@media (max-width: 1000px){
  h1 {
    color: orange;
  }
}

```

[Home](#) [Learn More](#) [About](#) [Contact](#) [Sign Up](#)

Media Queries



we are constantly orange why?

We are overwriting

1st media queries say before 500px we gonna be red.

2nd media queries say before 1000px we gonna be orange.

```

@media (max-width: 1000px){
  h1 {
    color: orange;
  }
}

@media (max-width: 500px){
  h1 {
    color: red;
  }
}

```

This is gonna work

Basically, max width overlaps and cause problem. So above way works if we arranged properly.

Use min-width to create a starting point and above.

```

h1 {
  color: red;
}

@media (min-width: 500px){
  h1 {
    color: orange;
  }
}

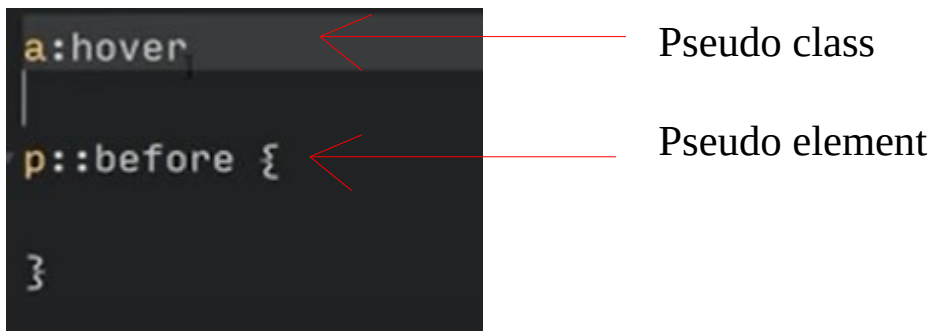
@media (min-width: 1000px){
  h1 {
    color: yellow;
  }
}

```

Before and after psuedo elements are really good ways to add extra style to your website without adding extra markup u dont really need.

::before & ::after

- 1) What they are and how they work
- 2) The misunderstood content property
- 3) Using them as design elements



```
p::before {  
  background: red;  
  display: block;  
  position: absolute;  
  top: 0;  
  bottom: 0;  
  left: 0;  
  right: 0;  
}
```

Here is some generic content

Cuz ur psuedo element is not their.

For your pseudo element to exist, do This
Weird

```
content: '';
```

```
p::before {
  content: '';
  background: red;
  display: block;
  position: absolute;
  top: 0;
  bottom: 0;
  left: 0;
  right: 0;
}
```



```
p::before {
  content: '';
  background: red;
  display: block;
  width: 10px;
  height: 10px;
}
```

Here is some generic content

content tell there should be some content if we dont add this, it default to none.

```
<p>
  ::before
  Here is some generic content
```

In Dev tools, u'll see there is a b4 inside of it.

Alot of ppl misconception, if they use before they think its before p, but thats not true, it get inserted before the content of p.



Similary with after,

```
<p>
  ::before
  Here is some generic content
  ::after
</p>
```

For images, the image source is the content, short story is ::before and ::after doesnt work with images.

Example

```
p::before {  
  content: 'hello';  
  background: red;  
}
```

helloHere is some generic content

helloHere is some generic content
goodbye

Notes: you can easily see this as a design element instead of putting it as text.

It lets u add things without adding extra markup.

For e.g u dont want a empty div for just for design elements, i dont have to do that i could use my before and after to do that

Content property:

-The pseudo element, content is not selectable.

```
2 .intro::before {  
3   content: url(//unsplash);  
4 }
```

```
blockquote::before {  
  content: open-quote;  
}  
  
blockquote::after {  
  content: close-quote;  
}
```

"This is a blockquote, where I could have a testimonial talking about how great my services are"

example

so if we have a link if i hover over, i want a little extra info to show up.

Something different

```
ipsum <a href="#" data-tool-tip="a cool tool tip">do
```

this were added to create to customized as how we want, mostly used for js purposes.

```

a[data-tool-tip] {
  position: relative;
}

a[data-tool-tip]::after {
  content: attr(data-tool-tip)
}

```

It's taking data tool tip attribute value, and its putting into there.

psum dolor sit ameta cool tool tip, consectetur
nisi in faucibus ore

See video at time 7:52

```

a[data-tool-tip]::after {
  content: attr(data-tool-tip);
  display: block;
  position: absolute;
  background-color: $clr-gray;
  padding: 1em 3em;
  color: white;
  border-radius: 5px;
  font-size: .8em;
  bottom: 100%;
  left: 0;
  white-space: nowrap;
}

```

final piece

```
a[data-tool-tip]::after {
  content: attr(data-tool-tip);
  display: block;
  position: absolute;
  background-color: $clr-gray;
  padding: 1em 3em;
  color: white;
  border-radius: 5px;
  font-size: .8em;
  bottom: 100%;
  left: 0;
  white-space: nowrap;
  transform: scale(0);
}

a[data-tool-tip]:hover::after {
  transform: scale(1);
}
```



```
-----
bottom: 0;
left: 0;
white-space: nowrap;
transform: scale(0);
transition:
  transform ease-out 150ms,
  bottom ease-out 150ms;
}

a[data-tool-tip]:hover::after {
  transform: scale(1);
  bottom: 100%;
}
```

we want the info to look like it's coming from the element, not from above.

Ideas

1)

```
.section-with-deco::before {  
  content: url(//unsplash.it/400/10);  
  display: block;  
}
```



2) At time 13:00, i am confused a bit

```
a[href$=".pdf"]::after {  
  font-family: 'Font Awesome 5 Free';  
  content: ' \f1c1'  
}  
  
a[href^="http"]::after {  
  font-family: 'Font Awesome 5 Free';  
  content: ' \f35d';  
  font-weight: 900;  
}
```

>>>



`font-size: .8em;` , < do this, if you find icon a little too big.

Counter-Reset [Counter-Reset](#)

```
.counters {  
  background: lightgray;  
  text-align: left;  
  padding: 5em 8em;  
  margin-top: 7em;  
  
  counter-reset: counter-name;  
}
```

The way is counter-reset is, you need to give your counter a name, why it's counter reset is : everytime it gets to a class of counter it resets itself, here that is counter_name.

```
.section::before {  
  counter-increment: counter-name;  
  content: counter(counter-name) '. ';  
}
```

Imagine ur div, getting drawn, whenever it gets to the div with class “counter” it resets a variable whatever manner.

```
.section { position: relative; }  
.section::before {  
  counter-increment: counter-name;  
  content: counter(counter-name);  
  position: absolute;  
  left: -2.5em;  
  top: -.5em;  
  background: white;  
  width: 2em;  
  height: 2em;  
  border-radius: 50%;  
  display: flex;  
  justify-content: center;  
  align-items: center;  
  border: 3px solid $clr-gray;  
  box-sizing: border-box;  
}
```



Hover effect ideas: [hover effect ideas](#)

Video: [video](#)

```
$ff-ss: 'Unica One';  
$ff-s: 'Vollkorn';  
  
$clr-red: #c31;  
$clr-black: rgba(black, 1);  
$clr-gray: rgba(black, .7);
```

```
*, *::before, *::after { box-sizing: border-box }
```

if u use ::before and ::after psuedo element it wont be a subset of universal selector, you need to explicitly add *::before and *::after

```
.intro {  
  position: relative;  
  display: inline-block;  
  
  &::before,  
  &::after {  
  
  }  
}  
  
.intro::before {}
```

Same

```
.intro {  
  position: relative;  
  display: inline-block;
```

HERE IS A GENERIC HEADING

```
&::after {  
  background: $clr-red;  
  width: 100vw;  
}  
}
```

HERE IS A GENERIC HEADING

```
&::after {  
  background: $clr-red;  
  width: 100vw;  
  left: 50%;  
}
```

HERE IS A GENERIC HEADING

it's gonna put right into the middle, cuz the intro-text is center-align.

```
transform: translateX(-50%);
```

HERE IS A GENERIC HEADING

Below .intro has z-index:1 that's why red and white background is -2 and -1

So code final

```
.intro {  
  position: relative;  
  display: inline-block;  
  
  &::before,  
  &::after {  
    content: '';  
    display: block;  
    position: absolute;  
  }  
  
  &::before {  
    background: white;  
    z-index: -1;  
    height: 101%;  
    left: -20px;  
    right: -20px;  
  }  
  
  &::after {  
    background: $clr-red;  
    width: 100vw;  
    left: 50%;  
    top: 0;  
    transform: translateX(-50%);  
    z-index: -2;  
    height: 100%;  
  }  
}
```

A GENERIC HEADING

The thing is it's gonna grow and and shrink with the content.



Flaw: if it breaks into two lines, the background thing gonna disappear because the way inline block is working and couple of other things.

It's too large now, u can work with media queries and play around with it. The white padding(left and right) is gone too big.