Introduction to Web Development, V2

Introduction to the Course-

Coding is closer as learning a language
Don't compare
No correct speed to learn
Never feel afraid to google anything (It is a requirement)
Don't be afraid to copy and paste
Don't need to learn everything right away.

Overview of learning in this Course-

Mostly Terminology (Words you need to search to get what you need)
Learning three Language = HTML, CSS and JAVASCRIPT
Learning about "Git"
Server "Node.js"-request response for a pizza place.

The House

HTML= Raw materials of the house (Bricks, wood etc)-[structure]
CSS= Blueprints of the house (Rules of the house- want this over here and there)-[blueprint]

Javascript=smart house of the house (alexa, lights on etc). You don't need a smart home, but most people these days use a smart home.-[frontend/ smart home)

Tools

If you build a house (HTML, CSS, JAVASCRIPT etc.) you need tools.
Choose tools that work for you.

Browser

Chrome, Firefox, Safari(Apple) will work fine.

The Editor

Tool you will be using to write code.
Recommendation: **Visual Studio Code** (Most people use it including professors)

The Terminal-

- Shell-You have the emulator and the emulator runs inside of it which is called the shell-use “Bash”
- Terminal Emulator-”iTerm2” or “cmder” for Windows- A console Emulator for windows- (VS code has built in Terminal Emulator)

Trusted Resources

- For anything to do with HTML, CSS, JAVASCRIPT, - **MDN** **https://developer.mozilla.org/en-US/**
- **CSS Tricks** **https://css-tricks.com/**

Learning HTML

Basic HTML

HTML- Hypertext Markup Language -(not important)
-Simply just language and pictures

Tags

Tags-building blocks (ex) <h1>, , <a>
-Every tag has a closing tag (ex) <h1></h1>, Hello
-closing tag always has a slash “/”. ex)

```
1 <h1>This is the title to my document</h1>
```

result

This is the title to my document

-<h1> “h” means “header”

-There are things called “Self closing Tags” or “Void Tags” that open and close themselves. ex) <input />

```
1 <input />
```

result

this is a text box

-An input tag creates an empty box for you to put inside the text. It wouldn’t make sense to put something inside the input tag.

-Self closing tags have nothing inside.

```
1 <h1>Hello</h1>
2 <h2>My</h2>
3 <h3>Name</h3>
4 <h4>Is</h4>
5 <h5>Chris</h5>
6 <h6>!</h6>
7 <p>sdfsdfd</p>
```

result

Hello

My

Name

Is

Chris

!

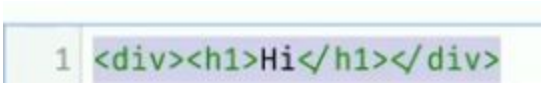
sdfsdfd

-It makes bolder closer to h1.

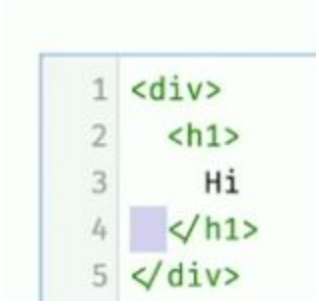
-Good to read later

-Tags are opened and closed in a specific order

Organization-



Works fine but hard to read and work with others.



The organized way.

HTML Elements

Types of Tag-

- **H1, h2, h3, h4, h5, h6**- Headings (Good for reading HTML)
- **P**- Paragraph. Only text goes into p tags. For example news articles.
- **A**- Anchor. To link somewhere.
  ex) <a href="https://www.frontendmasters.com>Click Here<a>

- **Div**- Short for division. Like a box that groups things. This goes well with CSS. **<u>USED A LOT</u>**
- **Span-** A container with small pieces of text. If div is like a box, span is like a Ziploc Bag.

Some text inside the div.
Some text inside the div.
Some text inside the div.
This is a span. This is a span. This is a span.

-You can still change display: inline , display: block;
**"Inline Elements" cannot have a width and a height.*

```
    <title>Div and Span</title>
    <link rel="stylesheet" type="text/css" href=
</head>
<body>

    <div><!-- block element -->
        Some text inside the div.
    </div>
    <div><!-- block element -->
        Some text inside the div.
    </div>
    <div><!-- block element -->
        Some text inside the div.
    </div>

    <span>
        This is a span.
    </span>
    <span>
        This is a span.
    </span>
    <span>
        This is a span.
```

-Span will display <u>in-line</u>.

- **Ol, ul, li** -
  Ol- Organized list ex) 1, 2, 3, 4, 5
  Ul- Unorganized list ex)

-
-
-

Li-listed elements. One individual entry inside of the ol or inside of the ul.

```
<ul><li>Bob</li><li>Eve</li><li>Alice</li></ul>
```

- **button** - a button. Can be used by JavaScript to respond to a user clicking it. It is like a doorbell of a door house. However, it will do nothing unless you connect it with a buzzer.

```
1 <button>Click me</button>
```

result

Click me

- **Img -** an image.
  - An image tag needs a source (where it comes from aka. “src”).
  - Alt is an alternative name for screen readers. (If the image broke or no longer exists, it is a backup for the readers to know.

```
1 <img src="http://www.placepuppy.net/100/100" alt="an adorable puppy" />
```

result

an adorable puppy

The above is correct.

  - ex)

```
1 <img src="http://www.placepuppy.net/100/100" alt="an adorable puppy" />
```

result

The above is correct.

- Inputs- ex) <inputs />

seen here: .

  - You can also utilize “input” by doing *numbers, dates, colors, checkboxes, radio buttons* etc.

- Textarea- ex) <textarea></textarea>
  - Bigger version of “inputs”.
  -For example, write an email to us.

```
1 <textarea></textarea>
```

result

ads;lfasd;lfa;ldskf

akldjfalk;sdfjlk;

-The circle above, you can make the “textarea” bigger or smaller.

**the text area does not have self closing tags because they are old.*

- Select and Option- ex)

<select>

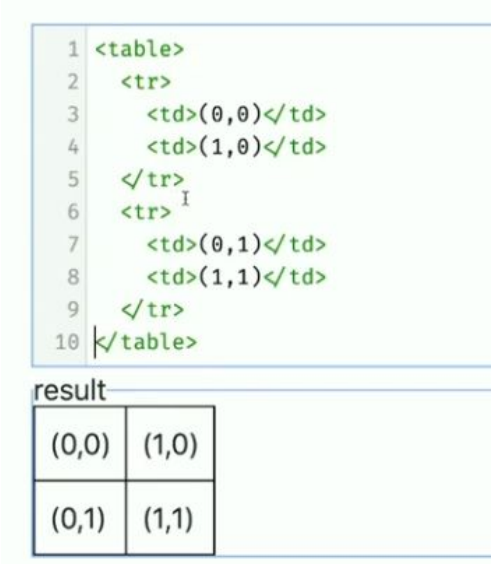
<option value=”seattle”>

Seattle

```
            </option>
            <option value=”portland>
                Portland
            </option>
            <option value=”san-francisco”>
                San Francisco
            </option>
        </select>
```

- Form -

- Tables- like making a table in Excel
    - Outdated
    - “Tr” means one row
    - “Td” means one cell
    - *Not too important...*



- Strong - ex) Hello **BOLD**
- Em- “emphasis” ex) Bye bye *Italic*

HTML Comments

-Coders always forget what they wrote after some time.

-”Comments” will help you remember the code of what you wrote.

<!-- this is a comment -->

```
<!-- this is a comment -->
<table>
  <tr>
    <td>(0,0)</td>
    <td>(1,0)</td>
    <td>(2,0)</td>
  </tr>
  <tr>
    <td>(0,1)</td>
    <td>(1,1)</td>
  </tr>
  <tr>
    <td>(0,2)</td>
    <td>(1,2)</td>
  </tr>
</table>
```

-comments do not show on actual result

-comments are “Notes”.

-However, don’t go overboard with comments.

-ex) `<h1>Title of the Article</h1><!-- the title -->`

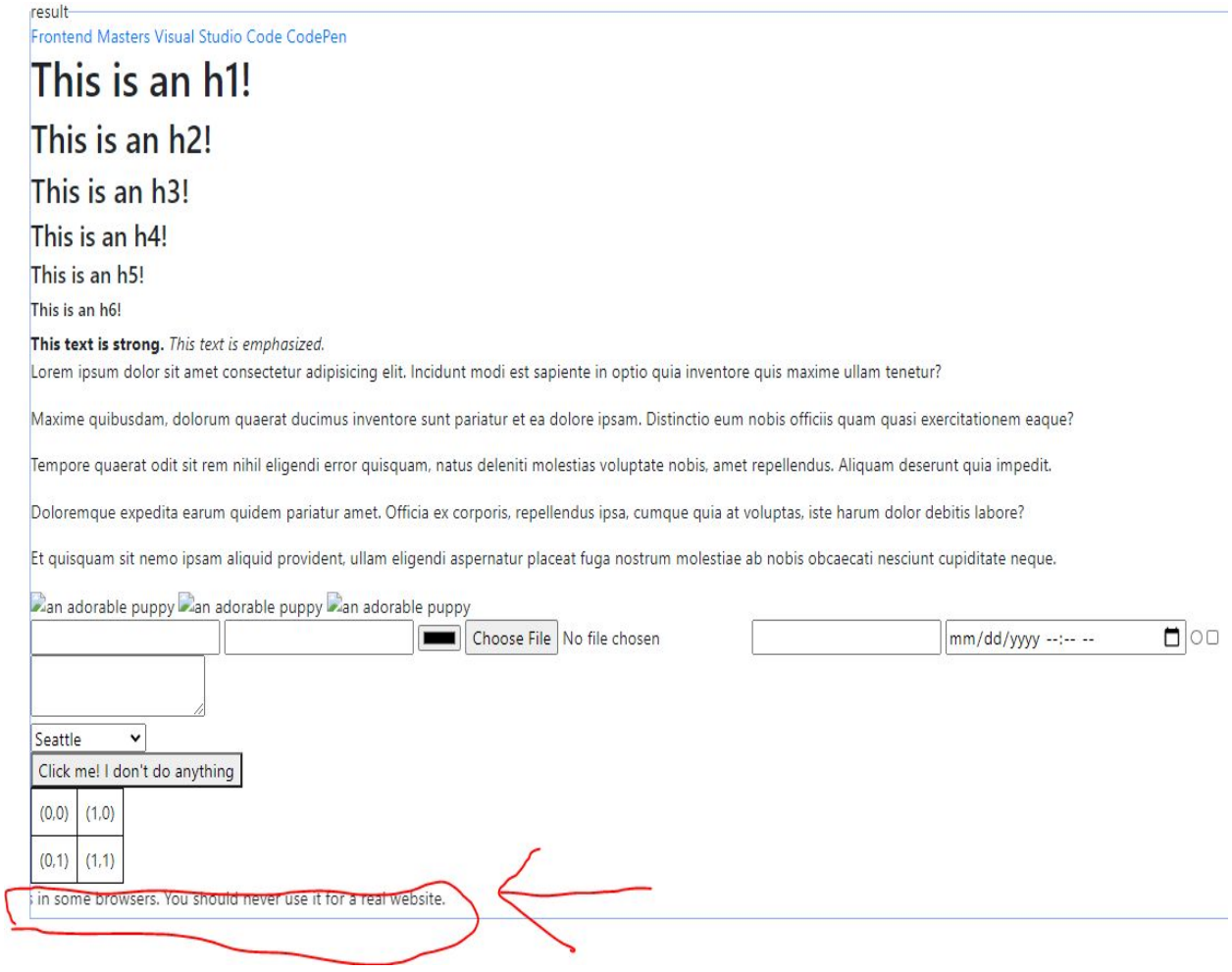
HTML Playground

- Lorem Ipsum Text -https://www.lipsum.com/

  -it looks like real text. Very useful for building text.

```html
<!-- links -->
<div>
  <a href="https://www.frontendmasters.com">Frontend Masters</a>
  <a href="https://aka.ms/visual-studio-code">Visual Studio Code</a>
  <a href="https://www.codepen.io">CodePen</a>
</div>

<!-- header -->
<div>
  <h1>This is an h1!</h1>
  <h2>This is an h2!</h2>
  <h3>This is an h3!</h3>
  <h4>This is an h4!</h4>
  <h5>This is an h5!</h5>
  <h6>This is an h6!</h6>
</div>

<!-- text empasis -->
<div>
  <strong>This text is strong.</strong>
  <em>This text is emphasized.</em>
</div>

<!-- paragraphs -->
<div>
  <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.
    Incidunt modi est sapiente in optio quia inventore quis maxime ullam tenetur?</p>
  <p>Maxime quibusdam, dolorum quaerat ducimus inventore sunt pariatur et ea dolore ipsam.
    Distinctio eum nobis officiis quam quasi exercitationem eaque?</p>
  <p>Tempore quaerat odit sit rem nihil eligendi error quisquam, natus deleniti molestias
    voluptate nobis, amet repellendus. Aliquam deserunt quia impedit.</p>
  <p>Doloremque expedita earum quidem pariatur amet. Officia ex corporis, repellendus ipsa,
    cumque quia at voluptas, iste harum dolor debitis labore?</p>
  <p>Et quisquam sit nemo ipsam aliquid provident, ullam eligendi aspernatur placeat fuga
    nostrum molestiae ab nobis obcaecati nesciunt cupiditate neque.</p>
</div>

<!-- images -->
<div>
  <img src="http://www.placepuppy.net/1p/100/100" alt="an adorable puppy" />
  <img src="http://www.placepuppy.net/2p/100/100" alt="an adorable puppy" />
  <img src="http://www.placepuppy.net/3p/100/100" alt="an adorable puppy" />
</div>

<!-- inputs -->
<div>
  <input /> <!-- with a trailing slash -->
  <input> <!-- without a trailing slash; it's the same thing -->
  <input type="color" />
  <input type="file" />
  <input type="number" />
  <input type="datetime-local" />
  <input type="radio" />
  <input type="checkbox" />
</div>
```

```
</div>

<!-- textarea -->
<div>
  <textarea></textarea>
</div>

<!-- select -->
<div>
  <select>
    <option value="seattle">Seattle</option>
    <option value="portland">Portland</option>
    <option value="san-francisco">San Francisco</option>
  </select>
</div>

<!-- buttons -->
<div>
  <button>Click me! I don't do anything</button>
</div>

<!-- table -->
<table>
  <tr>
    <td>(0,0)</td>
    <td>(1,0)</td>
  </tr>
  <tr>
    <td>(0,1)</td>
    <td>(1,1)</td>
  </tr>
</table>

<!-- for fun -->
<div>
  <marquee>This is a really old feature that only works in some browsers.
    You should never use it for a real website.</marquee>
</div>
```

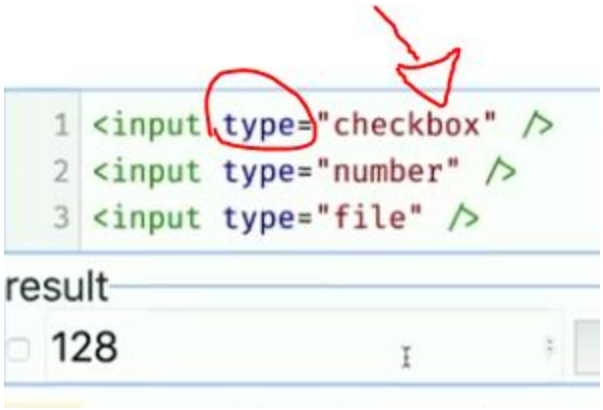


HTML ATTRIBUTES

The Circle below is an attribute.

-Input tag and type attribute (characteristic, type)

-Input tag (arrow) has to be one of the subset. Or else it will become default (input).



: Code :	Example
`<input value="This is a pre-filled value" />`	This is a pre-filled value
`<input placeholder="This is a placeholder" />`	This is a placeholder
`<input type="checkbox" checked />`	
`<input type="radio" checked />`	
`<input type="checkbox" disabled checked />`	
`<input type="color" value="#FF0000" />`	

HTML Class Attributes

Classes- Special attributes

-make sure to make good names.

-to distinct instructions for CSS

-VERY VERY COMMON

```
<div class="header">
  <h1 class="header-title">My Great Blog</h1>
</div>
<div class="blog-posts">
  <div class="post">
    <h1 class="post-title">When Not to Overextend House Metaphors</h1>
    <p class="post-text"> … </p>
  </div>
  <div class="post">
    <h1 class="post-title">Another Great Blog Post</h1>
    <p class="post-text"> … </p>
  </div>
</div>
```

IDs (HTML ID ATTRIBUTES)

-An ID is like a bigger class

-If ID is used, it should be only used **<u>once</u>** in your whole website. (some professors recommend not to use ID at all)

-It is just way too powerful (if classes are like normal hammers, IDs are like sledge hammers)

```
<div class="header">
  <h1 class="header-title">My Great Blog</h1>
</div>
</div>
<div class="blog-posts">
  <div id="house-metaphors-post" class="post">
    <h1 class="post-title">When Not to Overextend House Metaphors</h1>
    <p class="post-text"> … </p>
  </div>
  <div class="post">
    <h1 class="post-title">Another Great Blog Post</h1>
    <p class="post-text"> … </p>
  </div>
</div>
```

-As the ‘red circle’ and ‘blue circle’ below, you can use separate classes. (However, cannot use separate IDs)

```
<div class="header">
  <h1 class="header-title">My Great Blog</h1>
</div>
</div>
<div class="blog-posts">
  <div id="house-metaphors-post" class="post featured-post">
    <h1 class="post-title">When Not to Overextend House Metaphors</h1>
    <p class="post-text"> … </p>
  </div>
  <div class="post">
    <h1 class="post-title">Another Great Blog Post</h1>
    <p class="post-text"> … </p>
  </div>
</div>
```

-Don’t use CSS or JavaScript to tag IDs.
-ID are old.
-***<u>They are good to make a link and takes you directly to the ID on that page</u>***

Naming and Semantics

Naming and Tags to Use

- 75% of programmers and web-developer’s jobs are naming (classes). It is a good idea to think about what to name it.

-On this example above, you want to name the section “red-title”.

-However, designers change design <u>all the time</u>.

-If it’s still called “red-title” it will be very confusing to the programmers/designers later on.

-Should use the name like “post-title”, “blog-title” etc.

```
<meta http-equiv="X-UA-Compatible" content="IE-edge"> == $0
```

META HTML

-What we learned until now were the different parts of HTML.

-We will now learn the “Back Bone” of HTML.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>My amazing HTML Document</title>
  </head>
  <body>
    <h1>Check this out</h1>
    <!-- Your amazing HTML here -->
  </body>
</html>
```

-VS code (you can press “!” for a short cut below)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

</body>
</html>
```

`<!DOCTYPE html>` -Means that you are using HTML:5

-HTML:5 has been around for over 10 years.

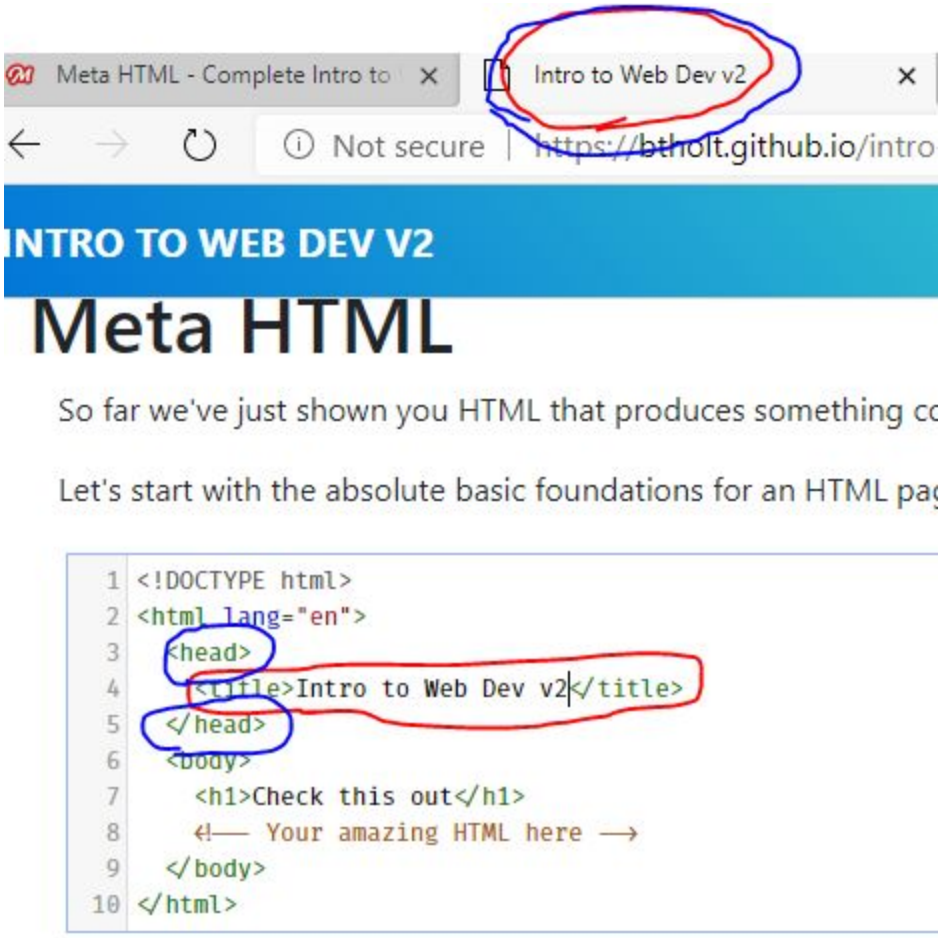
-Always has to be in the first line.

`<html lang="en"></html>` -”en” means “english”

-you will be receiving english documents.

`<head></head>`

-*"header" and "head" are different.*



-All the CSS links goes to "head" **[red]**
-"meta" inside "head" can be used to make the screen different compared to different screens (ex. Iphones, tablet, computer screen) **[blue]**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Calculator</title>
    <link rel = "stylesheet" href = "./calculator.css">
</head>
<body>
   <div class = "calc">
        <section class = "screen">
            0
        </section>
    <section class = "calc-buttons">
<div class = "calc-row">
```

`<body></body>` -Body

-From here, people are going to see (ex) div, class, img etc.)

[black]- link javascript

[red and blue] - link CSS.

[red only] - not usually a normal way to link CSS

CSS

-Introduction to CSS

CSS- **C**ascading(arranging) **S**tyle **S**heet

CSS is powerful: can change-

colors, sizes, order, positioning, hiding, showing, animation etc.

Like HTML, CSS is not a programming language (Presentation).

CSS is a list of rules that give the browser.

CSS's Frontend Masters learning path course -

https://frontendmasters.com/learn/css/

CSS Rules-

ex) <h1>This is an h1</h1>

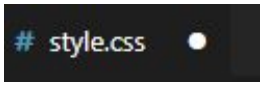
If you want to make it red.... This is an h1.

or ..

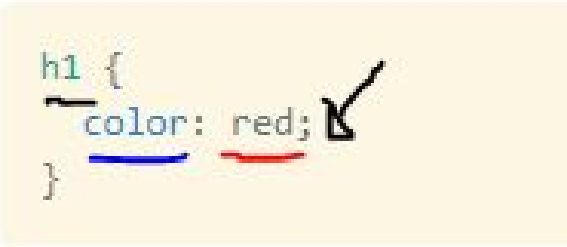
<head>

```
<link rel="stylesheet" href="./style.css">
```

</head>




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



- h1(black) - **Selector** - everything between “{}” will be applied.
- color(blue) - **Property** - there are about ~ 350 properties that you can use. You will mostly use ~50 of them (kind of like a skill lists)
  Link: https://meiert.com/en/indices/css-properties/
- red(red) - **value** - For colors, there are
  - Hexadecimals- #ff0000
  - RGB value- rgb(255, 0, 0)
  - HSL value- hsl(0, 100%, 50%)

  - *(Hexadecimals are mostly used)*
  - Named colors like “red” ~150 colors.
  - *Link for colors: https://www.colourlovers.com/colors*

**If you forget, google it.*

- ; (arrow) - **semicolon** - like a period of every sentence (CSS & Javascript)

Ex)

```
h1 {
  color: limegreen;
  font-size: 60px;
  font-weight: normal;
  text-decoration: underline;
  text-transform: uppercase;
  border: 3px solid pink;
}
```

- `font-size: 60px;`

  Px = pixels (fixed measures)- *ex) 60px looks the same on mobile and laptop screens.*

  Em = (relative unit) - measured according to the original font (changes according to parent)

```
.parent {
  font-size: 30px;
}
```

  - *If you change your font, your measurement will change relating to the font.*
  - *1em = 30px (1 x 30)*
  - *2em = 60px (2 x 30)*
  - *.5em = 15px (.5 x 30)*

  % = Percent - ex) p = 16px *(100% is around 16 px = "p")*

```
p{
  Font-size: 50%;   = 8px (.5 x 16)
```

}

Rem*= (relative to HTML tags)- root ems (m measured in a root level)*

1em = 16px
H1 ~= 36px

- `font-weight: normal;` - h1 is “Bold” by default. “Normal” makes the text normal. “Lighter” makes text thinner.
- `text-decoration: underline;`- This is how to underline the text.

```
text-decoration: line-through;
```

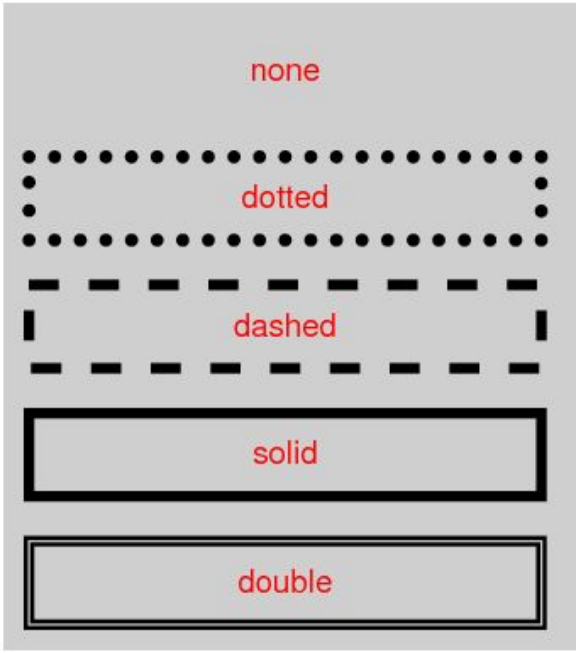
- `text-transform: uppercase;`- makes everything “Uppercase”

- `border: 3px solid pink;`

(Very common)

1. Border-width: 3px; (how thick)
2. Border-style: solid;

(border-style list: none, dotted, solid, double, dashed etc.)



3. Border-color: pink;

**(Order for "border" does not matter ex. Solid pink 3px)*

Parents and Children-

ex)

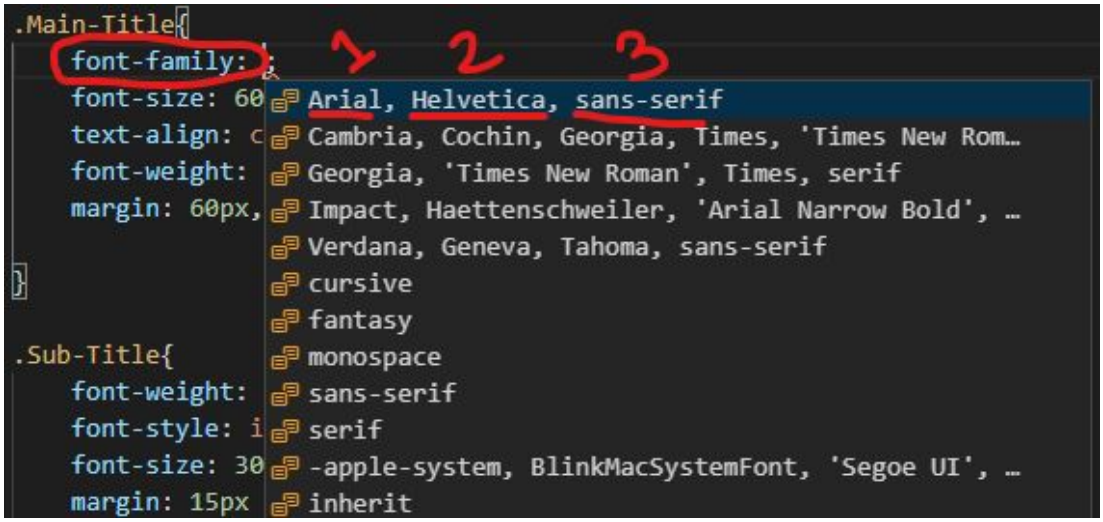
```
<body>
    <div>
        <h1>Hello, World</h1>
        <h2>Bye, World</h2>
    </div>
</body>
```

- "div" is the parent of "h1".
- "H1" is the child.
- Ancestor <body> & sibling <h2>

```
div {
  color: blue;
}
```

-”h1” will be blue.
-parent will affect the child.

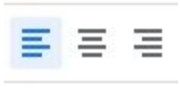
- Font-family: Arial, Helvetica, san-serif;



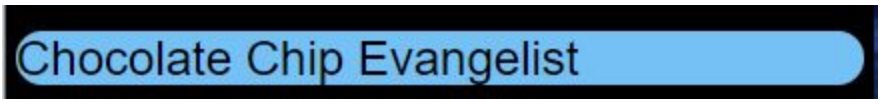
- Font-style: *italic*;

*Difference between “font-family” and “font-style” = “font-family” goes by order if the browser doesn’t support it.

(Helvetica (2) and sans-serif (3) are backups)

- Text-align: center, left, right 
- Background: black; or Background-color: black;

- Border-radius: 15px;



(Cookie Monster Project) -> Higher the number(px), the rounder the edge will be.

- List-style: upper-roman, lower-roman

i. List Item 1
ii. List Item 2
iii. List Item 3
iv. List Item 4
v. List Item 5

, square etc.

-must use for “lists” only ex) li, ol, ul etc.

CSS Playground/Exercise - https://codepen.io/btholt/pen/ELaxOB

CSS Selectors and the Cascade-

- Although it can be very effective, it can also be abused.

```
    <style>
        h1{
            color: black;
        }

        h2{
            color: green;
        }
    </style>


</head>
<body>
    <h1>Party Tips</h1>
    <h2>1. Party Tip: Crash Parties</h2>
    <!--<img src="partytips/penguin-party.jpg" alt="Penguin Party">-->
    <h2>2. Party Tip: Party Hard!</h2>
```

Result

How do you make 1. Red and 2. Green above?
Answer: Use Classes. (Important)

```
    <style>
        h1{
            color: black;
        }

        .first-tip{
            color: red;
        }

        .second-tip{
            color: green;
        }

    </style>
</head>
<body>
    <h1>Party Tips</h1>
    <h2 class="first-tip">1. Party Tip: Crash Parties</h2>
    <!--<img src="partytips/penguin-party.jpg" alt="Penguin Party">-->
    <h2 class="second-tip">2. Party Tip: Party Hard!</h2>
```

Result

* For Tags ex) h1, h2, p, **div**, span etc., put regular names to CSS

ex) h1 {
Color: red;
}

* For Classes ex) <h2 class =”first-tip”>, use **“.”** first to CSS
(“.” means class)

ex) .first-tip{
Color: red;
}

- **“.”** (Period) Means Class.
- “#” (Hashtag) Means ID.
- Professionals mostly use style(CSS) on “Classes”.

use CSS on tags if everything looks the same.

The Cascade-

- CSS can be unnecessarily complicated.
- Try to make CSS as simple as possible.

Which one wins?

Ex. 1)

```
<style>
  .title {
    color: red;
  }

  .title {
    color: green;
  }
</style>
<h1 class="title">Cool Title</h1>
```

result

Cool Title

Why? = If the property is equal, the bottom/last one wins.

Ex. 2)

```
<style>
  .main-brand-2 {
    border: 1px solid black;
    color: red;
  }

  .title-2 {
    color: green;
  }
</style>
<h1 class="title-2 main-brand-2">Branding here</h1>
```

result

Branding here

*one tag can have more than one class (blue).

*Two selectors have the same ***specificity**/value. It runs inside “list by list”.

Ex. 2.5)

```
    <style>
        h1{
            color: black;
        }

        h2{
            color: green;
            border: dashed 3px pink;
        }
        .first-tip{
            color: red;
        }



    </style>

</head>
<body>
    <h1>Party Tips</h1>
    <h2 class="first-tip">1. Party Tip: Crash Parties</h2>
    <!--<img src="partytips/penguin-party.jpg" alt="Penguin Party">-->
    <h2 class="second-tip">2. Party Tip: Party Hard!</h2>
```

Ex. 3)

```
1  <style>
2    .main-brand-3.title-3 {
3      color: red;
4      border: none;
5    }
6
7    .title-3 {
8      color: green;
9
10   }
11 </style>
12 <h1 class="title-3 main-brand-3">Branding here</h1>
```

result

Branding here

= Top wins (blue) because it is more “specific” (2 class vs. 1)

- Two Classes above (blue) MUST be shoved together or the function will break. (No space)

  ex) .main-brand-3.title-3 (right)

  .main-brand-3 .title-3 (wrong)

Ex. 4)

```
1  <style>
2    .title-4 {
3      color: orange;
4    }
5
6    h1 {
7      color: green;
8    }
9  </style>
10 <h1 class="title-4">Another h1</h1>
```

result

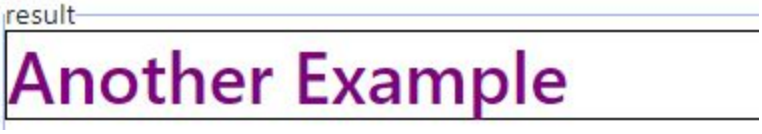
= Top wins because Class(.title-4) is more specific(important) than Tag(h1).

Class(10) > Tag(1)

Ex. 5)

```
<style>
  h1.main-brand-5 {
    color: red;
  }

  .main-brand-5.title-5 {
    color: purple;
  }

  .main-brand-5 {
    color: green;
  }

  h1{
  color: orange;
  }
</style>
<h1 class="title-5 main-brand-5">Another Example</h1>
```

result



=Purple wins because purple has two classes(main-brand-5, title-5). Top(11), Second(20), Third(5), Fourth(1).

- Not recommended to put tags and classes together.

IDs and !important-

- IDs and !important are like Sledge Hammers/Wrecking Ball when most problems require regular house hammers.
- Try not to use these tools unless you have a good reason.

Ex)

```
<style type="text/css">
  #site-brand {
    color: red;
  }

  h1.nav-head.nav-main.other-useful-class {
    /*
     * this class is way too specific; never have a class selector so long
     * it's ridiculous and just to illustrate a point
     */
    color: green;
  }
</style>
<h1 id="site-brand" class="nav-head nav-main other-useful-class">The Brand of my Website</h1>
```

result

-Not too great to use “id” in the first place.

- for ID, use “#”

ID (255) > Class(10) > Tag(1)

- <!--comments for HTML-->
- /* comments for CSS */

!important-

```
<style>
  #site-brand-2 {
    color: red;
    border: 1px solid red;
  }

  h1 {
    color: green !important;
    border: 1px solid green;
  }
</style>
<h1 id="site-brand-2" class="nav-head-2 nav-main-2 other-useful-class-2">The Brand of my Website</h1>
```

result

!important(1000’s) > ID (255) > Class(10) > Tag(1)

-Old, for lazy developers.

-You know the project is messed up if you see ID or !important.

Ex) *you can also repeat “same” classes.

```
<style>
  h1.main-brand-5 {
    color: red;
  }

  .main-brand-5.title-5 {
    color: orange;
  }

  .main-brand-5.main-brand-5.main-brand-5 {
    color: green;
  }
</style>
<h1 class="title-5 main-brand-5">Another Example</h1>
```

result

Another Example

Pseudo-Classes-

Pseudo Class Selectors are CSS selectors with a “**colon**” (**:**) preceding them.

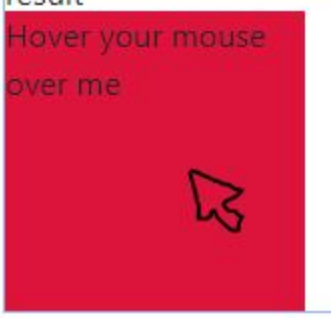
ex)

```
<style>
  .hover-example:hover {
    background-color: crimson;
    width: 150px;
    height: 150px;
  }
</style>
<div class="hover-example">Hover your mouse over me</div>
```

result

Hover your mouse over me

result



Another Example with “:hover”

```
1 <style>
2   .hover-example:hover{
3     text-decoration: underline;
4
5   }
6 </style>
7 <div class="hover-example">Hover your mouse over me</div>
```

result

Some examples for “Pseudo-Classes”

- :active - changes/activates when you “Press”
- :first-child - chooses the “first” from the list.
- :last-child - chooses the “last” from the list.

```
<style>
  .first-child-example {
    color: red;
  }
  .first-child-example:first-child {
    color: limegreen;
  }
</style>
<ol>
  <li class="first-child-example">First</li>
  <li class="first-child-example">Second</li>
  <li class="first-child-example">Third</li>
</ol>
```

result

1. First
2. Second
3. Third

- :nth-child(#) - chooses the “#” from the list
  -You can also use math for :nth-child
  ex) :nth-child(2n), :nth-child(2n+1) etc.

```
<style>
  .first-child-example {
    color: red;
  }
  .first-child-example:nth-child(2n) {
    color: limegreen;
  }
</style>
<div>
  <p class="first-child-example">First</p>
  <p class="first-child-example">Second</p>
  <p class="first-child-example">Third</p>
  <p class="first-child-example">Fourth</p>
</div>
```

result

First

Second

Third

Fourth

```
<style>
  .first-child-example {
    color: red;
  }
  .first-child-example:nth-child(3) {
    color: limegreen;
  }
</style>
<ol>
  <li class="first-child-example">First</li>
  <li class="first-child-example">Second</li>
  <li class="first-child-example">Third</li>
  <li class="first-child-example">Fourth</li>
  <li class="first-child-example">Fifth</li>
  <li class="first-child-example">Sixth</li>
</ol>
```

result

1. First
2. Second
3. Third
4. Fourth
5. Fifth
6. Sixth

Alternative Example-

```
<style>
  .first-child-example {
    color: red;
  }
  .first-child-example:nth-child(2) {
    color: limegreen;
  }
</style>
<div>
  <p class="first-child-example">First</p>
  <p class="first-child-example">Second</p>
  <p class="first-child-example">Third</p>
</div>
```

result

First

Second

Third

Pseudo-Classes list=

https://css-tricks.com/pseudo-class-selectors/

Pseudo-Elements- ::before, ::after (will go this subject later)

1)-https://www.youtube.com/watch?v=OtBpgtqrjyo

2)-https://css-tricks.com/almanac/selectors/a/after-and-before/

WildCard Selector- The ultimate code- It will select EVERY element

```
* {
}
```

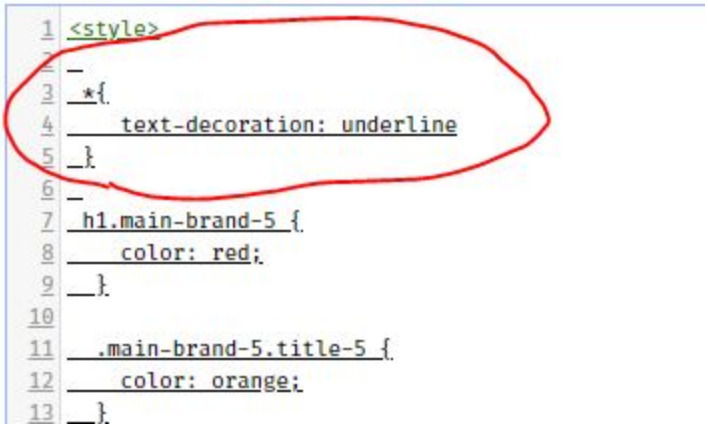
ex)

This is why it's useful to do *everything* using classes; you

A useful but imperfect way to think about this is to think

The bigger number wins. This is imperfect because one

Last one and we'll move on:



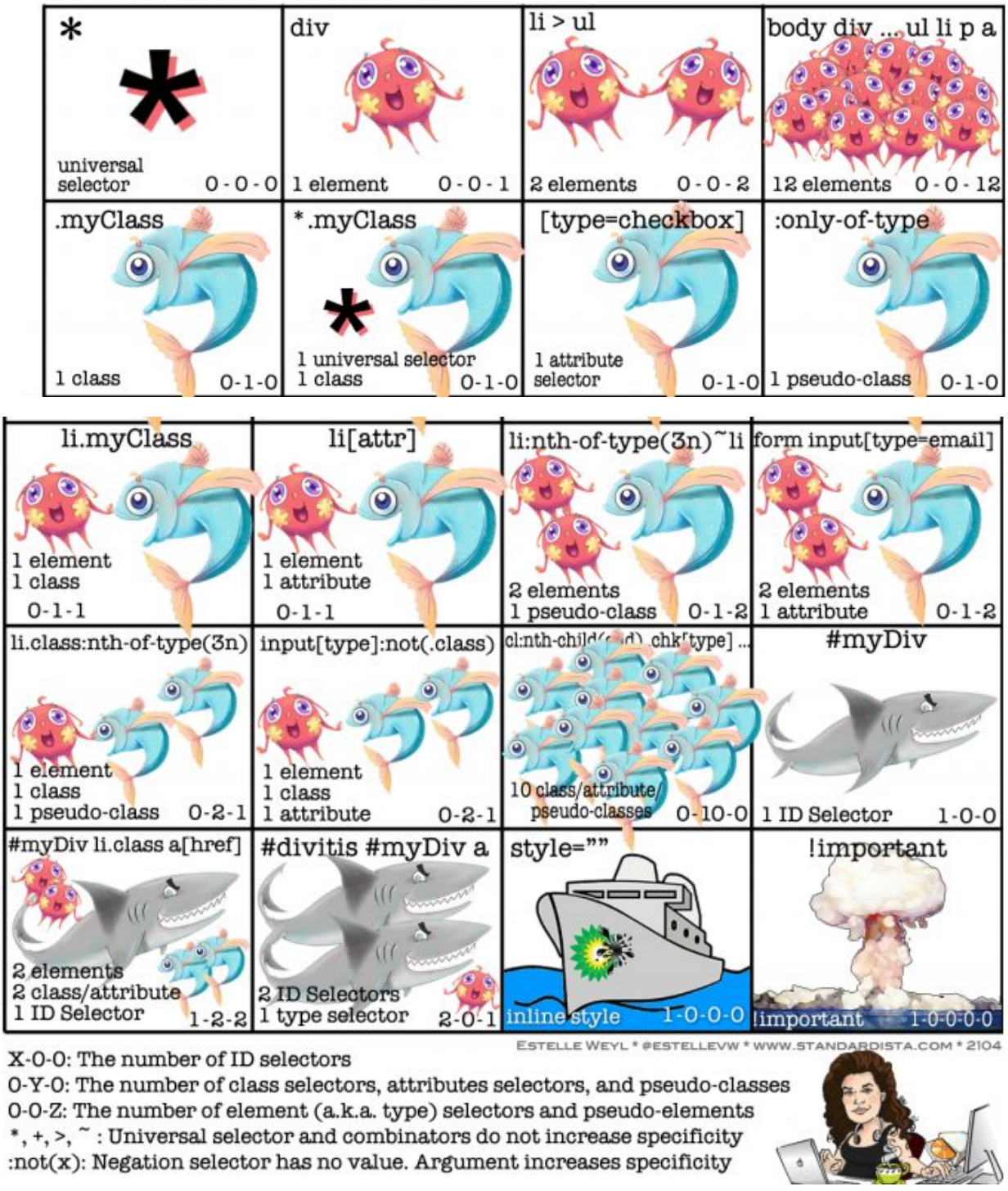
```
<style>

 *{
    text-decoration: underline
 }

 h1.main-brand-5 {
    color: red;
 }

 .main-brand-5.title-5 {
    color: orange;
 }
```

-Almost never use the "wild-card" except in one situation. (Border-Box)

CSS Specificity Guide- https://specifishity.com/

CSS SPECIFISHITY

WITH PLANKTON, FISH AND SHARKS



ESTELLE WEYL * @ESTELLEVW * WWW.STANDARDISTA.COM * 2104

X-0-0: The number of ID selectors
0-Y-0: The number of class selectors, attributes selectors, and pseudo-classes
0-0-Z: The number of element (a.k.a. type) selectors and pseudo-elements
*, +, >, ~ : Universal selector and combinators do not increase specificity
:not(x): Negation selector has no value. Argument increases specificity

How to Get a Job
http://getting-the-front-end-job.surge.sh/

How to Get a Job example website
https://adriangrimm.com/

https://tarunyadav.codes/#work

Youtube: traversy media

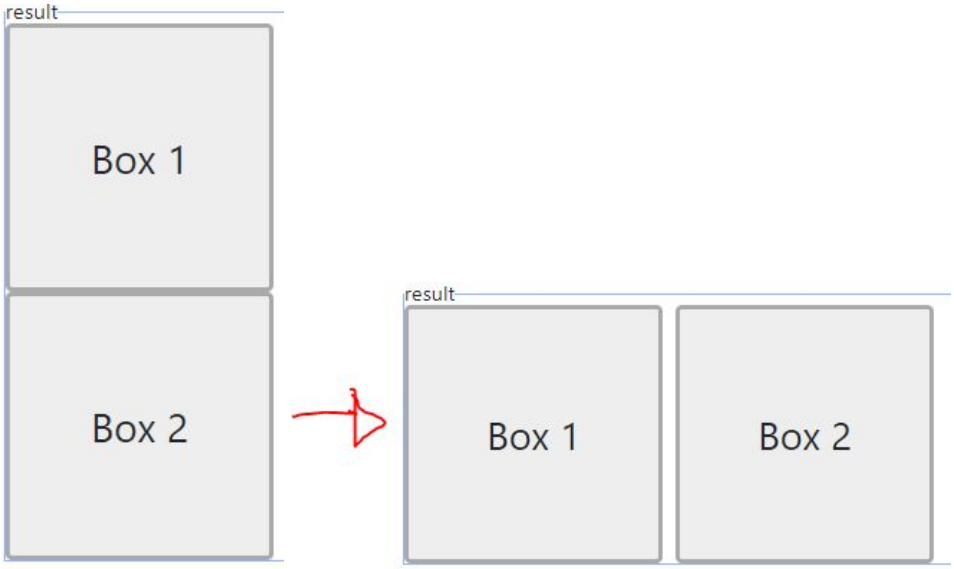
Codepen

Interviewing for Front-End Engineers
https://frontendmasters.com/courses/intehttps://tarunyadav.codes/#workrviewing-https://tarunyadav.codes/#workfrontend/

CSS Box Model-

Layout CSS- You can use CSS to “layout” your page.

CSS makes boxes(layouts) from left to right.



Float- The oldest version of doing “Layouts”(arrangements)

- Because “Float” is so old, they are almost never used anymore.

```
<style>
  .ex-box {
    border: 3px solid #aaa;
    background-color: #eee;
    height: 200px;
    width: 200px;
    display: flex;
    align-items: center;
    justify-content: center;
    float: left;
    margin-right: 10px;
    border-radius: 5px;
    font-size: 30px;
  }
  .ex-box:last-of-type {
    clear: right;
  }
</style>
<div class="ex-box">Box 1</div>
<div class="ex-box">Box 2</div>
```

”float-: left;” means “keep pushing to the left”.

```
<style>
  .ex-box {
    border: 3px solid #aaa;
    background-color: #eee;
    height: 200px;
    width: 200px;
    display: flex;
    align-items: center;
    justify-content: center;
    float: right;
    margin-right: 10px;
    border-radius: 5px;
    font-size: 30px;
  }
  .ex-box:last-of-type {
    clear: left;
  }
</style>
<div class="ex-box">Box 1</div>
<div class="ex-box">Box 2</div>
```

result

- There is no “float: center” in Float.
- Before, it was very difficult to center an image. Not anymore.

The Box Model-

- Important to know.
- It will help you if you learn the “Box Model” earlier than later.
- Confusing, but worth investing time into understanding.

Display-

- Inline (display: inline)- Inline is not a great practice for CSS. (messy and limited)

```
<!DOCTYPE html>
<html>
<head>
<style>
span.a {
  display: inline; /* the default for span */
  width: 100px;
  height: 100px;
  padding: 5px;
  border: 1px solid blue;
  background-color: yellow;
}
```

-Code (CSS)

Code (HTML)

```
<h2>display: inline</h2>
<div>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat scelerisque elit sit amet
consequat. Aliquam erat volutpat. <span class="a">Aliquam</span> <span class="a">venenatis</span> gravida nisl
sit amet facilisis. Nullam cursus fermentum velit sed laoreet. </div>
```

Result-

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat scelerisque elit sit amet consequat. Aliquam erat volutpat. Aliquam venenatis gravida nisl sit amet facilisis. Nullam cursus fermentum velit sed laoreet.

- ❏ Won't let you change any of the height, width, padding, margins etc.

- Block (display: block)- Will give you control of height, width, padding, margin etc.

```
span.c {
  display: block;
  width: 100px;
  height: 100px;
  padding: 5px;
  border: 1px solid blue;
  background-color: yellow;
}
```

-Code(CSS)

Code(HTML)

```
<h2>display: block</h2>
<div>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat scelerisque elit sit amet
consequat. Aliquam erat volutpat. <span class="c">Aliquam</span> <span class="c">venenatis</span> gravida nisl
sit amet facilisis. Nullam cursus fermentum velit sed laoreet. </div>
```

Result-

display: block

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat scelerisque elit sit amet consequat. Aliquam erat volutpat.

Aliquam

venenatis

gravida nisl sit amet facilisis. Nullam cursus fermentum velit sed laoreet.

- ❏ By default, “block” takes the whole line itself.
- ❏ “Div” and “p” default are “display-block”.

- Inline-block (display: inline-block)- hybrid between “Inline” and “block”

```
span.b {
  display: inline-block;
  width: 100px;
  height: 100px;
  padding: 5px;
  border: 1px solid blue;
  background-color: yellow;
}
```

-Code (CSS)

Code (HTML)

```
<h2>display: inline-block</h2>
<div>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat scelerisque elit sit amet
consequat. Aliquam erat volutpat. <span class="b">Aliquam</span> <span class="b">venenatis</span> gravida nisl
sit amet facilisis. Nullam cursus fermentum velit sed laoreet. </div>
```

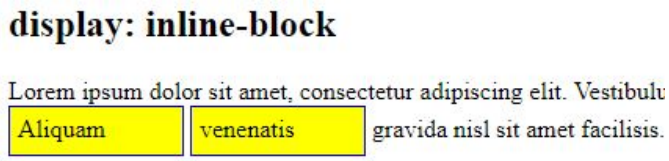
Result:

display: inline-block

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat scelerisque elit sit amet consequat. Aliquam erat volutpat.

Aliquam venenatis gravida nisl sit amet facilisis. Nullam cursus fermentum velit sed laoreet.

```
span.b {
  display: inline-block;
  width: 100px;
  height: 20px;
  padding: 5px;
  border: 1px solid blue;
  background-color: yellow;
}
```



❏ Only “inline-block” can do this out of the three examples.

- Flex & inline flex-

  Newer version of layouts- Was something not common before, but is used almost everywhere

  - For example, flex will handle in a certain section of a website to be laid out horizontally and vertically centered, spaced out in a particular way etc. in a nice fashion.

- Grid and inline-grid-

  Newest version and most advanced of layouts in CSS.

  - Can do certain things that flex-box cannot. (positioning elements in 2 dimensions)-[rows AND columns]

```css
.grid {
  display: grid;
  max-width: 800px;
  grid-template-columns: 1fr 2fr;
  grid-template-rows: 1fr 2fr 1fr;
}
```

- (Flex-box can only do rows or columns)
- Flex-box and Grid can be used together.

-Not many companies are using “grid” in production today since it is still pretty new.

- table-

Height, Width, Padding, Border and Margin-

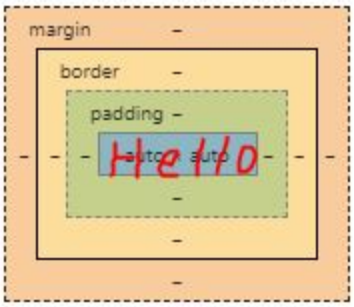
5 Properties that will make the complete element.

ex) Most people get confused between “Padding” and “Margin”.

```
.example {
  border: 3px solid red;
  padding: 5px;
  margin: 25px;
  background-color: white;
}
```

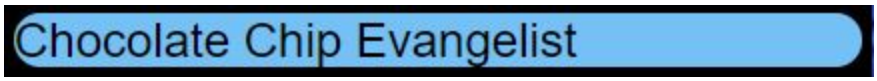
Margin(Red)- Space “Outside” the Border.
Border(Black)
Padding(Blue Circle)- Space “Inside” the Border.

Ex 2)



Original-

```
h2{
    background: #75c1f6;
    color: black;
    border-radius: 15px;

}
```

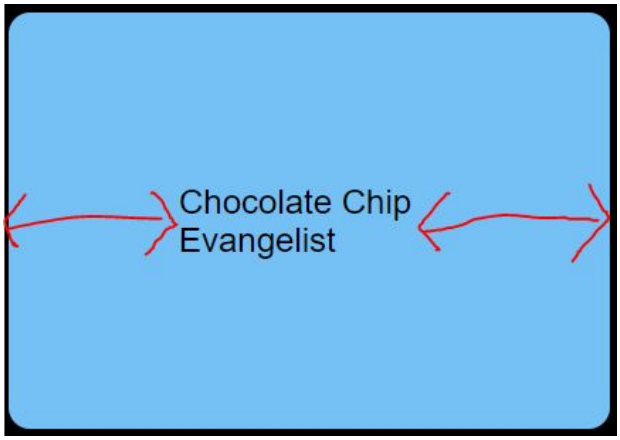


Margin-

```
h2{
    background: #75c1f6;
    color: black;
    border-radius: 15px;
    margin: 5em;

}
```

Padding-

```
h2{
    background: #75c1f6;
    color: black;
    border-radius: 15px;
    padding: 5em;

}
```



Question: What does for example “width: 200px” include below?

```
.example {
  border: 3px solid red;
  padding: 5px;
  margin: 25px;
  background-color: white;
}
```

+ width: 200px;

border-box

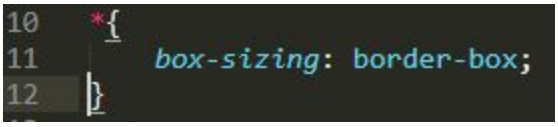
Answer:

It depends…It depends on what kind of box-sizing we are using. However, normal/default box-sizing is called, “Content Box”.

Content Box does not include “padding” or “border” (only “content”). This makes measuring **really really** difficult.

-It would be **much** more easier, if “width: 200px” included the “border”, “padding” and the “content”. That way it would be easier to measure.

- Box-sizing: border-box; will change it for you to include “border”, “padding” and “content” when sizing the box.



- **“Box-sizing: border-box” is the ONLY time you can use the *Wild-Card Selector*(*)-(page. 38)**
- This way, you don’t have to think about adding padding and borders together. (Very Common)

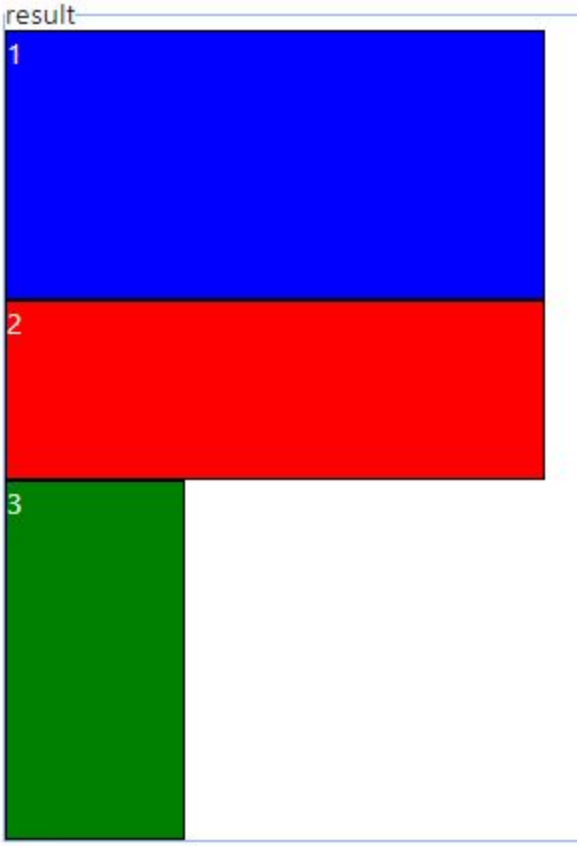
CSS Floats and Flexbox-

We will be using box-1, box-2 and box-3 for examples.

Website: https://btholt.github.io/intro-to-web-dev-v2/layout-css

```
<style>
  .box-1 {
    border: 1px solid black;
    color: white;
    background-color: blue;
    height: 150px;
    width: 300px;
  }
  .box-2 {
    border: 1px solid black;
    color: white;
    background-color: red;
    height: 100px;
    width: 300px;
  }
  .box-3 {
    border: 1px solid black;
    color: white;
    background-color: green;
    height: 200px;
    width: 100px;
  }
</style>
<div class="box-1">1</div>
<div class="box-2">2</div>
<div class="box-3">3</div>
```

result



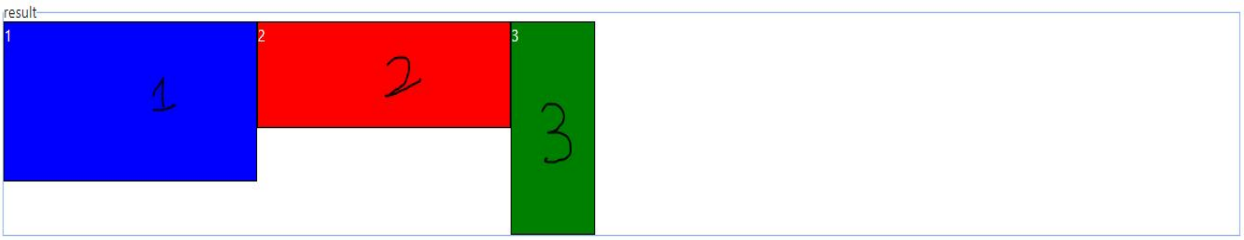
Floats-

Although it is an older version of layout than flex, Floats still can be useful.

ex 1)

```
<style>
  .floated div {
    float: left;
  }
</style>
<div class="floated">
  <div class="box-1">1</div>
  <div class="box-2">2</div>
  <div class="box-3">3</div>
</div>
```

- The ".floated div" means all the "divs" inside "class floated".

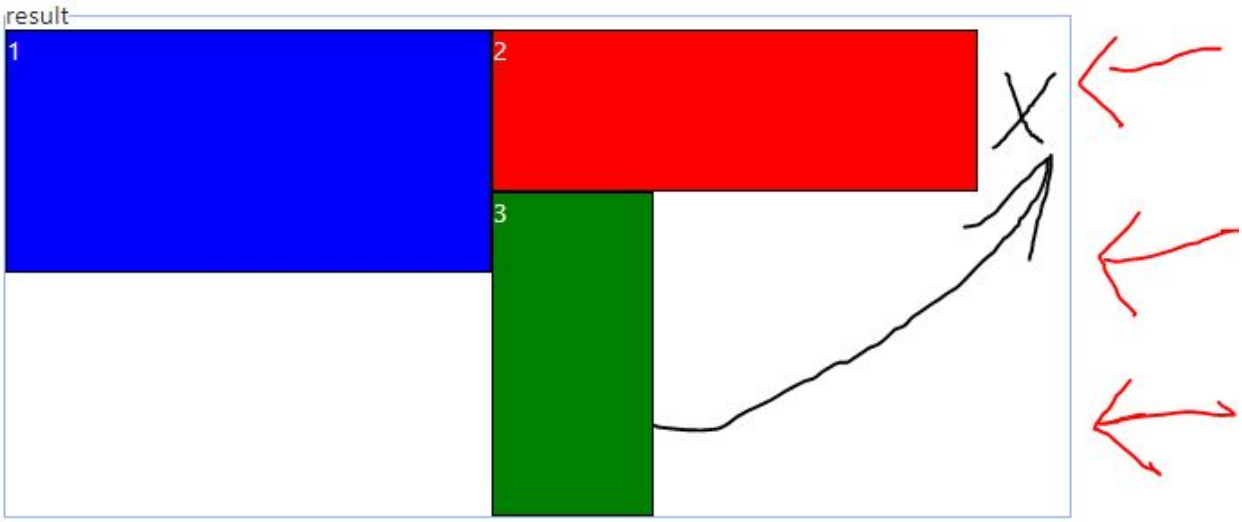


However, instead of “.floated div”, it is better to use “float” example as below:

```
<style>
  .floated .box {
    float: left;
  }
</style>
<div class="floated">
  <div class="box box-1">1</div>
  <div class="box box-2">2</div>
  <div class="box box-3">3</div>
</div>
```

-All “class box” inside “class floated”.

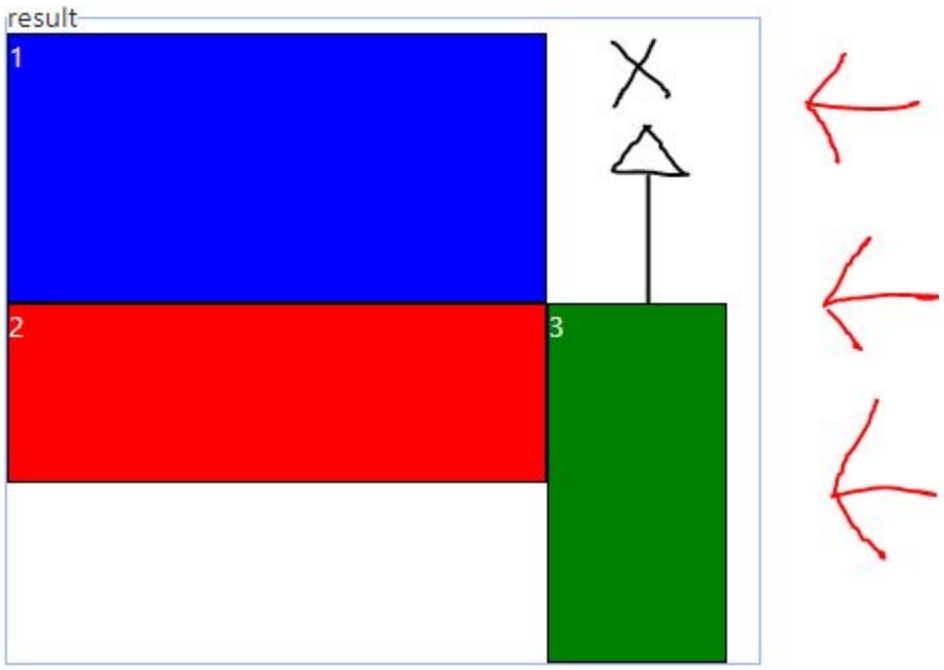
ex 2)



-As the screen becomes smaller (red), the result box becomes smaller as well. Because “box-3” does not have enough space on the right, it goes under for “float: left”.

The next example can make floats confusing. Which is one of the negative points of Float.

ex 3)



-As we make the screen <u>smaller</u>, “box-3” goes next to “box-2” instead of “box-1”.

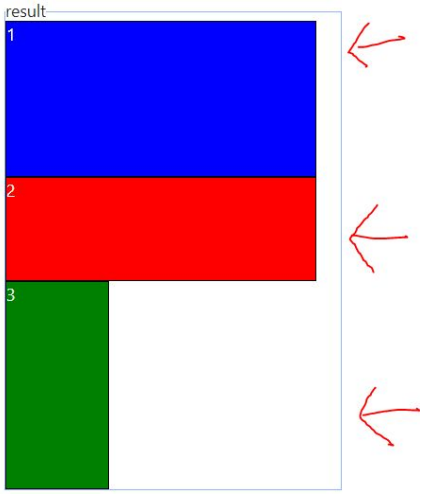
-This is because “floats” never push up higher that comes before.

-”box-3” will <u>never</u> be higher than “box-2” or “box-1”.

-Why? It’s just the way it is.

If you make the screen even smaller…

ex 4)



-You can also float “right” (float: right;). However, no “center” (float: center;).

Flex-

If you set “display: flex” you are controlling all the “children” of it. You are controlling everything “inside” of it which is different from float.

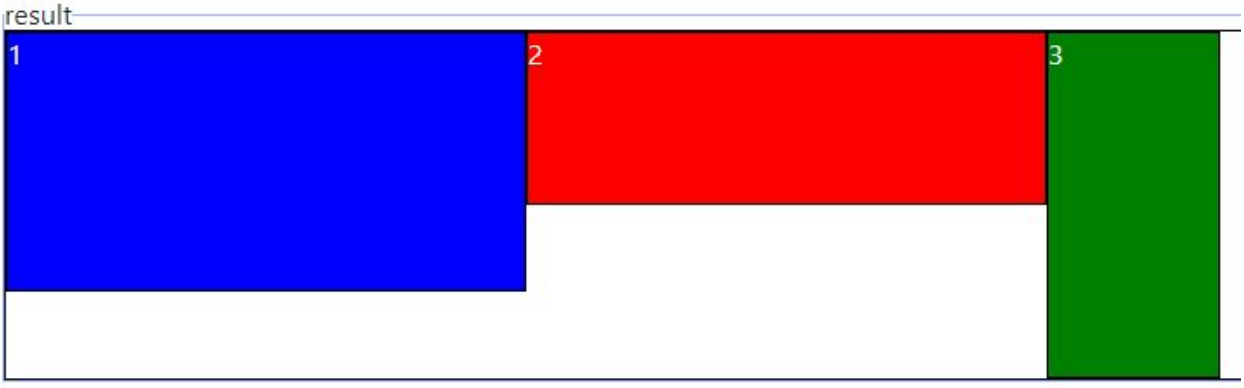
Therefore, you put the “display: flex” on the “parent” container.

Flex-Froggy: https://flexboxfroggy.com/

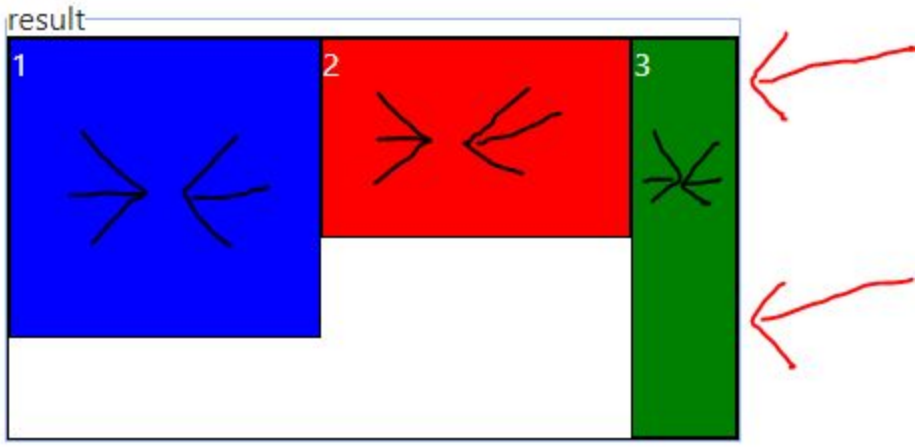
Ex 1)

```
<style>
  .flex-container {
    display: flex;
    width: 100%;
    border: 1px solid black;
  }
</style>
<div class="flex-container">
  <div class="box-1">1</div>
  <div class="box-2">2</div>
  <div class="box-3">3</div>
</div>
```

result



However, unlike “float”, if you make the screen smaller,

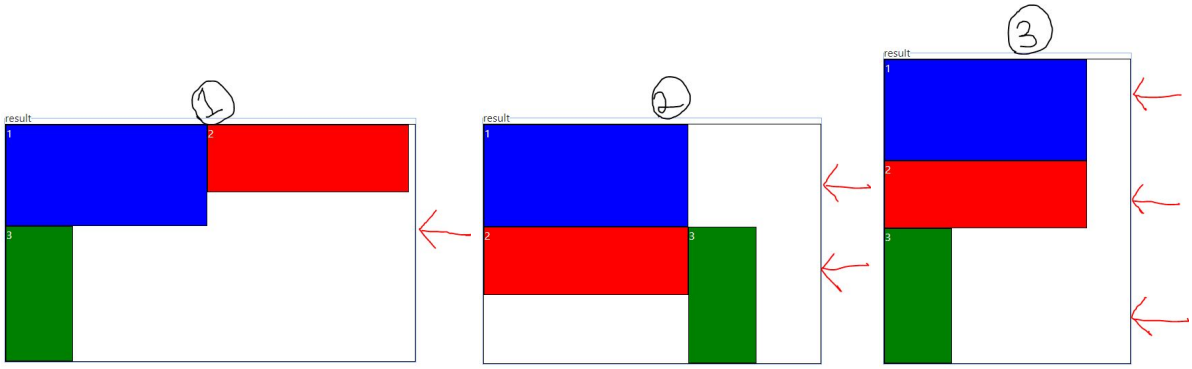


Instead of “boxes” going bottom each other, the box will squish.

Why? Because we did not apply “flex-wrap” to our flex.

```
<style>
  .flex-container {
    display: flex;
    width: 100%;
    border: 1px solid black;
    flex-wrap: wrap;
  }
</style>
<div class="flex-container">
  <div class="box-1">1</div>
  <div class="box-2">2</div>
  <div class="box-3">3</div>
</div>
```

“flex-wrap: wrap;”



Three things most useful for “Flex”

1. Flex-direction
2. Justify-content
3. Align-items

- **When you use “flex” properties, you <u>MUST</u> put “display: flex” <u>First</u>** (Important)

```
.flex-container {
  display: flex;
  width: 100%;
  border: 1px none black;
  flex-direction: row;
}
```

More Into depth for flex-box (Professor Jen Kramer):
https://frontendmasters.com/courses/css-grids-flexbox/

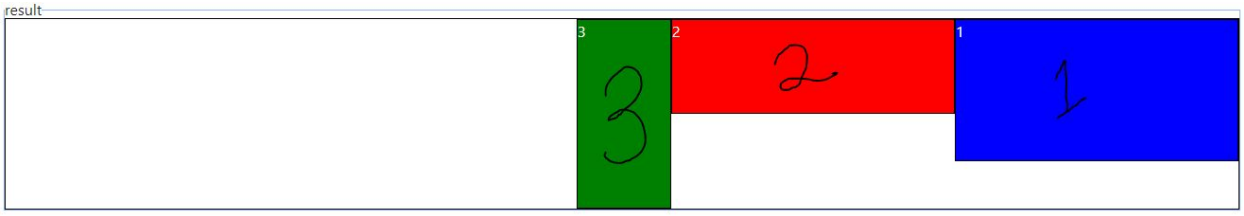
Flex-direction-

- “Flex-direction” has 4 properties-[**Direction** properties]
  <u>-(row or column)</u>

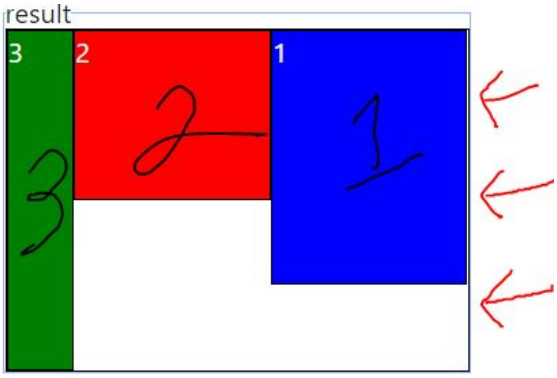
- **row**: Items are placed the same as the text direction.
- **row-reverse**: Items are placed opposite to the text direction.
- **column**: Items are placed top to bottom.
- **column-reverse**: Items are placed bottom to top.

Flex-direction: row-reverse;

```
<style>
  .reverse {
    flex-direction: row-reverse;
  }
</style>
<div class="flex-container reverse">
  <div class="box-1">1</div>
  <div class="box-2">2</div>
  <div class="box-3">3</div>
</div>
```

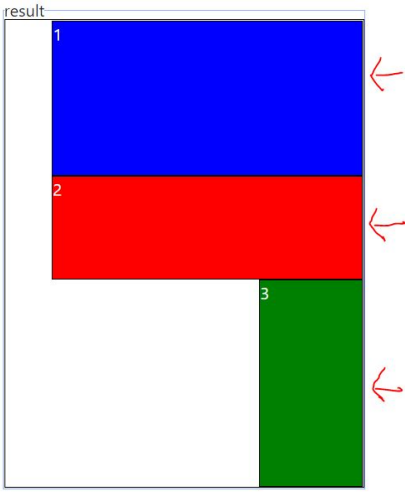


-Just the opposite of “display: flex”



-”flex-direction: row-reverse” smaller screen.

```
<style>
  .reverse {
    flex-direction: row-reverse;
    flex-wrap: wrap;
  }
</style>
<div class="flex-container reverse">
  <div class="box-1">1</div>
  <div class="box-2">2</div>
  <div class="box-3">3</div>
</div>
```

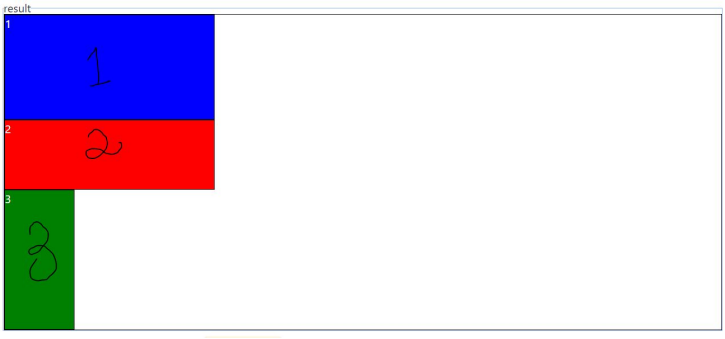


Even if we put “flex-wrap: wrap;”, it is exactly opposite.

- Reminder: for anything related to “flex”, you are using the “parent”, <u>not</u> the “child”.

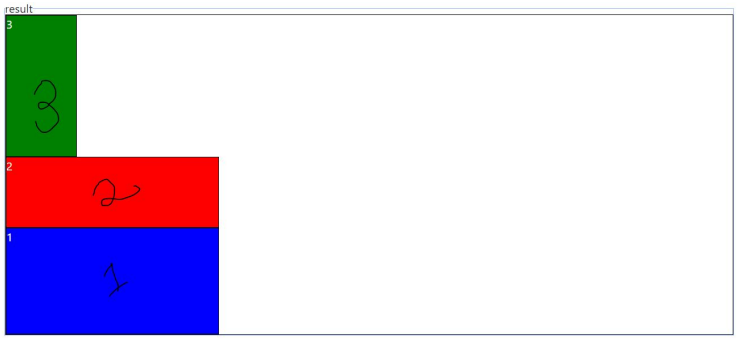
Flex-direction: column;

```
<style>
  .column {
    flex-direction: column;
  }
</style>
<div class="flex-container column">
  <div class="box-1">1</div>
  <div class="box-2">2</div>
  <div class="box-3">3</div>
</div>
```



Flex-direction: column-reverse;

```
<style>
  .column {
   flex-direction: column-reverse;
   }
</style>
<div class="flex-container column">
  <div class="box-1">1</div>
  <div class="box-2">2</div>
  <div class="box-3">3</div>
</div>
```



Justify-content-

- Justify-content: has 6 properties-[**Vertical** properties]
  -(left & right)

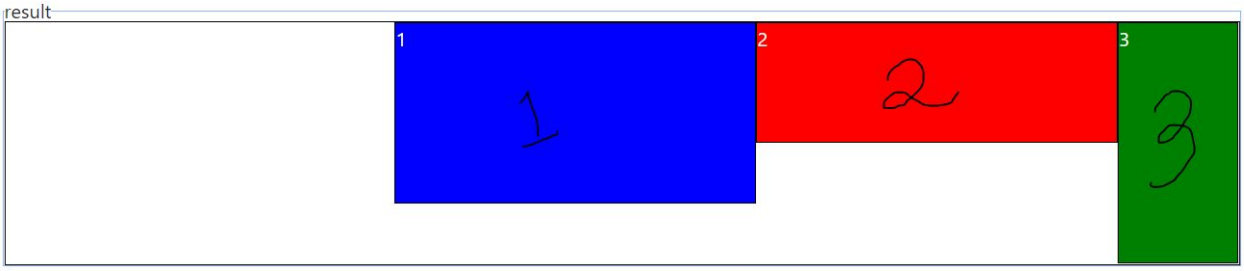
- **flex-start**: Items align to the left side of the container.
- **flex-end**: Items align to the right side of the container.
- **center**: Items align at the center of the container.
- **space-between**: Items display with equal spacing between them.
- **space-around**: Items display with equal spacing around them.

- Space-evenly: Items display with exact spacing around them.

Justify-content: flex-end;

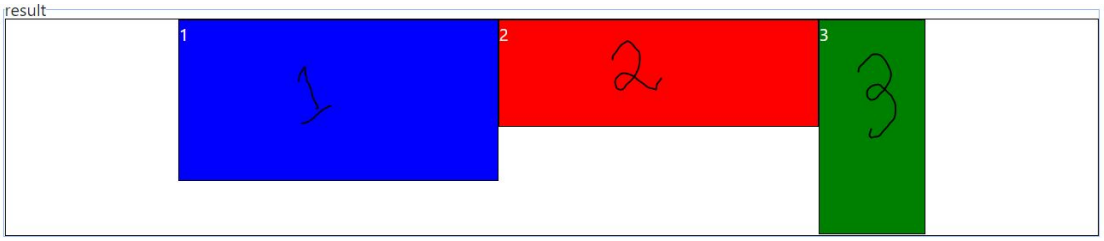
```
<style>
  .jc-right {
   justify-content: flex-end;
  }
</style>
<div class="flex-container jc-right">
  <div class="box-1">1</div>
  <div class="box-2">2</div>
  <div class="box-3">3</div>
</div>
```

result



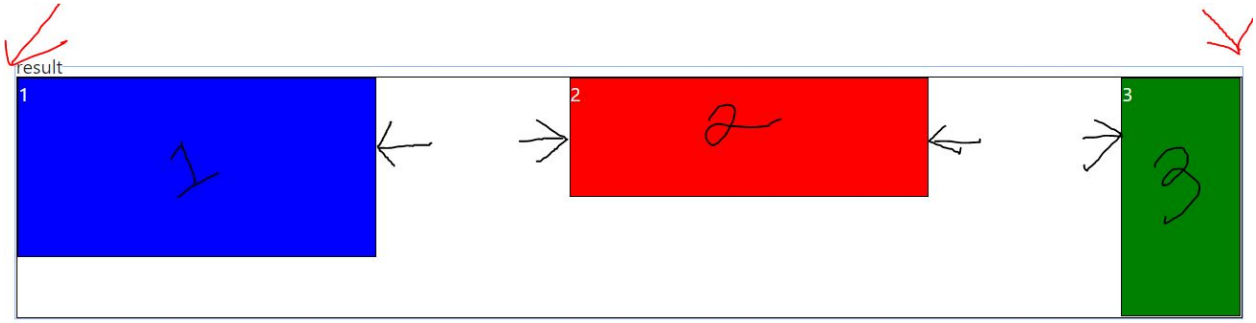
- By default, it’s always “flex-start”.

```
<style>
  .jc-center {
   justify-content: center;
  }
</style>
<div class="flex-container jc-center">
  <div class="box-1">1</div>
  <div class="box-2">2</div>
  <div class="box-3">3</div>
</div>
```



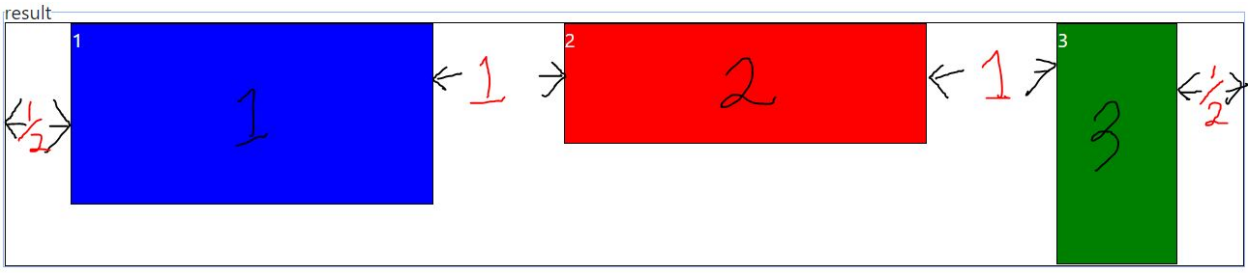
- Before flex, it was very difficult to “center” boxes.

```
<style>
  .jc-sb {
    justify-content: space-between;
  }
</style>
<div class="flex-container jc-sb">
  <div class="box-1">1</div>
  <div class="box-2">2</div>
  <div class="box-3">3</div>
</div>
```



- Space-between: to make space things “evenly” where the left and right are each at the end.

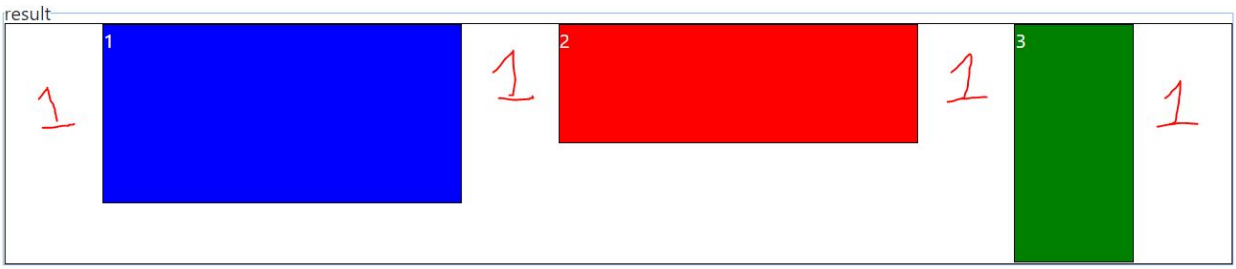
```
<style>
  .jc-sa {
    justify-content: space-around;
  }
</style>
<div class="flex-container jc-sa">
  <div class="box-1">1</div>
  <div class="box-2">2</div>
  <div class="box-3">3</div>
</div>
```



If you want all equally spaced,

```
1  <style>
2    .jc-sa {
3      justify-content: space-evenly;
4    }
5  </style>
6  <div class="flex-container jc-sa">
7    <div class="box-1">1</div>
8    <div class="box-2">2</div>
9    <div class="box-3">3</div>
10 </div>
```

space-evenly;



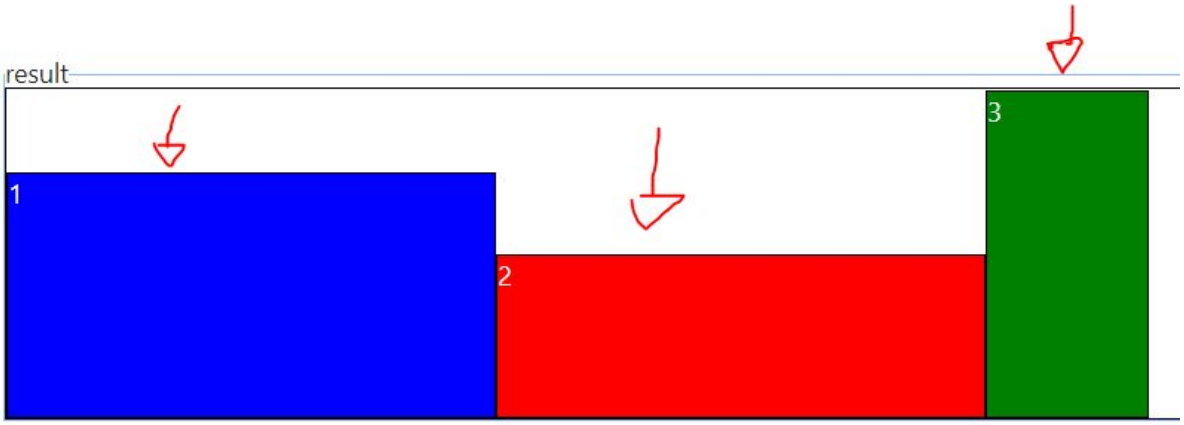
Align-items-

- align-items: has 5 properties-[**Horizontal** properties]

-(up & down)

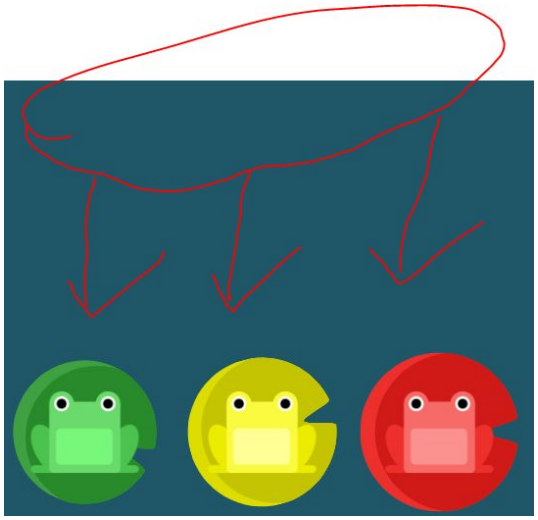
- **flex-start**: Items align to the top of the container.
- **flex-end**: Items align to the bottom of the container.
- **center**: Items align at the vertical center of the container.
- **baseline**: Items display at the baseline of the container.
- **stretch**: Items are stretched to fit the container.

```
<style>
  .ai-fe {
    align-items: flex-end;
  }
</style>
<div class="flex-container ai-fe">
  <div class="box-1">1</div>
  <div class="box-2">2</div>
  <div class="box-3">3</div>
</div>
```



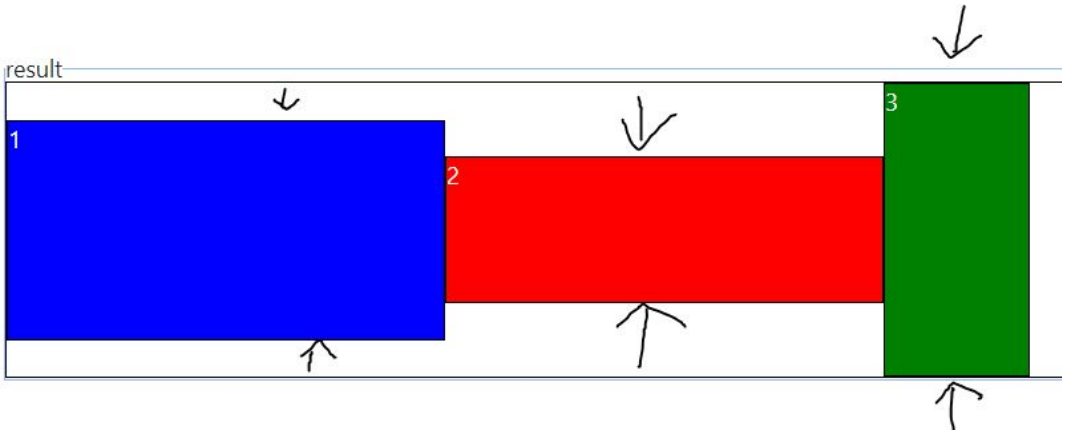
Flex-box froggy example-

```
#pond {
  display: flex;
  align-items: flex-end;
}
```



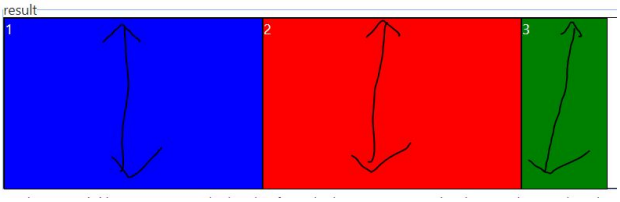
Align-items: center;

```
<style>
  .ai-center {
    align-items: center;
  }
</style>
<div class="flex-container ai-center">
  <div class="box-1">1</div>
  <div class="box-2">2</div>
  <div class="box-3">3</div>
</div>
```



The blocks were centered horizontally.

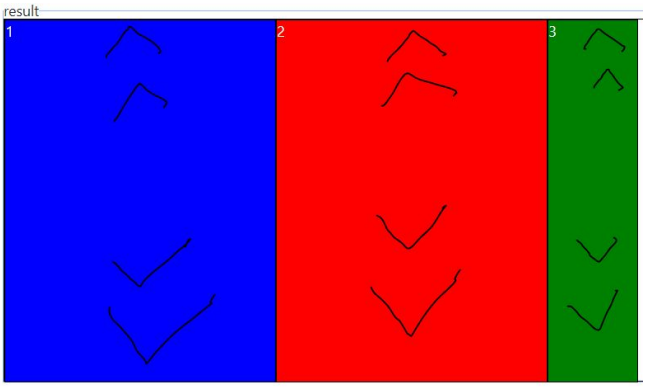
```
<style>
  .ai-stretch {
    align-items: stretch;
    height: 200px;
  }

  /* remove the height from the three boxes */
  .no-height {
    height: inherit;
  }
</style>
<div class="flex-container ai-stretch">
  <div class="box-1 no-height">1</div>
  <div class="box-2 no-height">2</div>
  <div class="box-3 no-height">3</div>
</div>
```



Will stretch (horizontally) until it fits the div.

```
<style>
  .ai-stretch {
    align-items: stretch;
    height: 400px;
  }

  /* remove the height from the three boxes */
  .no-height {
    height: inherit;
  }
</style>
<div class="flex-container ai-stretch">
  <div class="box-1 no-height">1</div>
  <div class="box-2 no-height">2</div>
  <div class="box-3 no-height">3</div>
</div>
```

If you change height to 400px,



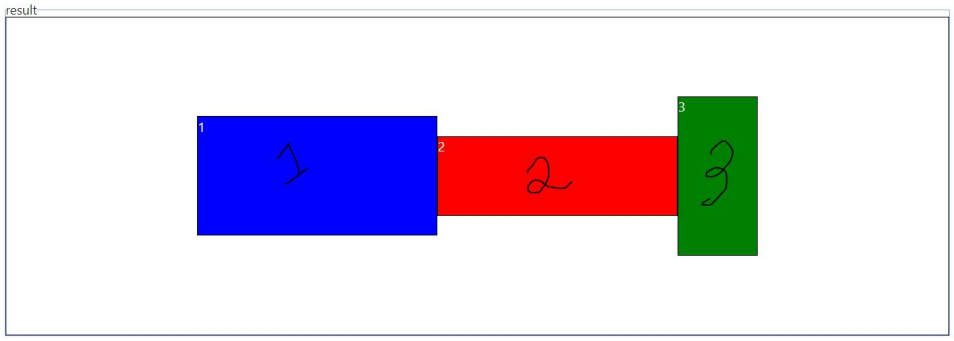
will stretch further.

You can also mix between the three properties (“flex-direction”, “justify-content” and “align-items”)

```
<style>
  .ai-center {
    display: flex;
    height: 400px;
    align-items: center;
    justify-content: center;



  }
</style>
<div class="flex-container ai-center">
  <div class="box-1">1</div>
  <div class="box-2">2</div>
  <div class="box-3">3</div>
</div>
```

ex)

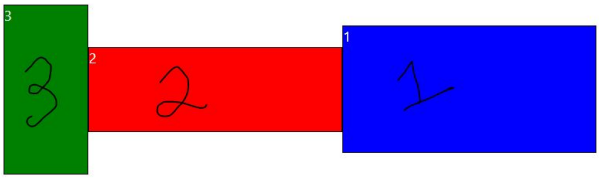
result




```
<style>
  .ai-center {
    display: flex;
    height: 400px;
    align-items: center;
    justify-content: center;
    flex-direction: row-reverse;



  }
</style>
<div class="flex-container ai-center">
  <div class="box-1">1</div>
  <div class="box-2">2</div>
  <div class="box-3">3</div>
</div>
```

-adding “flex-direction”

result



- All the “Flex” examples above, you use it through “Parent”. However, there are properties that you can use with “Children” as well. (Individual)
- “Order” and “align-self”

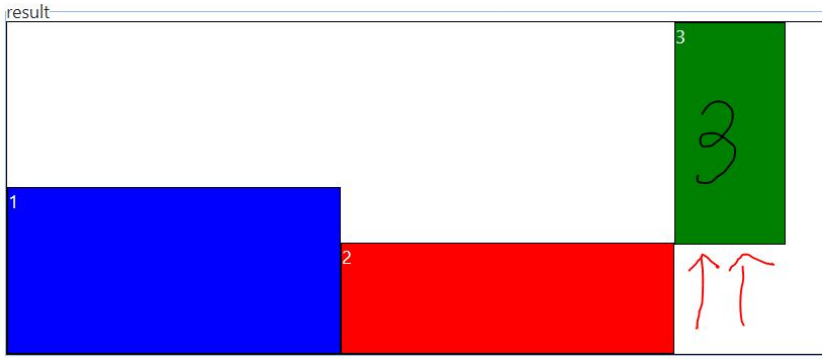
order with **align-self** to help the frogs to their destinations.

- “Align-self” accepts the same property as “align-items”

- **flex-start**: Items align to the top of the container.
- **flex-end**: Items align to the bottom of the container.
- **center**: Items align at the vertical center of the container.
- **baseline**: Items display at the baseline of the container.
- **stretch**: Items are stretched to fit the container.

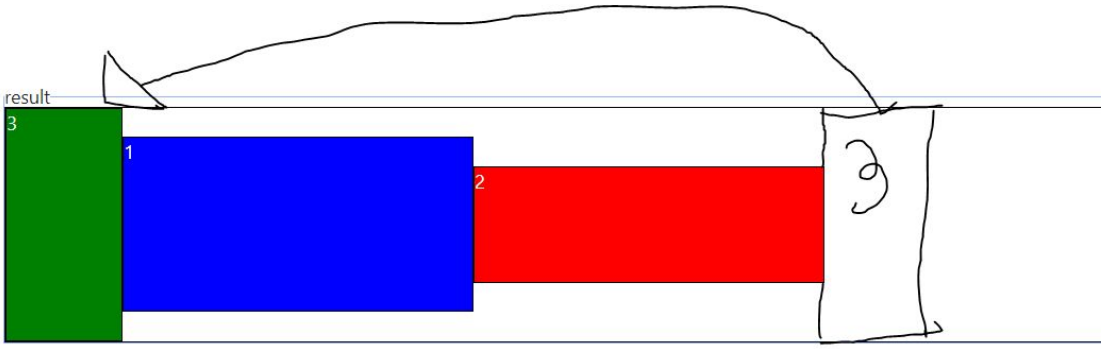
```
<style>
  .ai-fe {
    align-items: flex-end;
    height: 300px;
  }

  .box-3 {

    align-self: flex-start;
  }
</style>
<div class="flex-container ai-fe">
  <div class="box-1">1</div>
  <div class="box-2">2</div>
  <div class="box-3">3</div>
</div>
```

ex)



Order-
ex)

```
<style>
  .ai-center {
    align-items: center;
  }

  .box-3{
    order: -1;
  }
</style>
<div class="flex-container ai-center">
  <div class="box-1">1</div>
  <div class="box-2">2</div>
  <div class="box-3">3</div>
</div>
```



- The higher the order number, the righter it will go.
- The lower the order number, the lefter it will go.
- If you have many boxes and want a certain box to go to a certain location, order will do the trick.

Because there are too much and will be almost impossible to remember everything about flex-box, Here is the website below for all about flex-box properties:

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

Definition of <Nav>

-The HTML <**nav**> element represents a section of a page whose purpose is to provide navigation links, either within the current document or to other documents. Common examples of navigation sections are *menus, tables of contents, and indexes*.

```
<nav>
    <ul class="main-flex">
        <p class="list-1"><img class="logo" src="img/1.jpg" alt="logo">Home</p>
        <li class="list-2 list">About <span>Us</span></li>
        <li class="list-3 list">Menu</li>
        <li class="list-4 list">Recommendations</li>
        <li class="list-5 list">Contact Us</li>
    </ul>
</nav>
```

op/Beer%20Website/brewery_website-master/brewery_website-master/ir

Effective Patterns for Coding CSS

-Main Goal is to put HTML and CSS together.

Ways to connect HTML and CSS

- Link tag
- Style tag

ex) for Style tags

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>My amazing HTML Document</title>
  <style>
    .my-brand {
      color: red;
    }
  </style>
</head>
<body>
  <h1 class="my-brand">Check this out</h1>
  <!-- Your amazing HTML here -->
</body>
</html>
```

-Anything inside Style tag will be read as CSS.

- Good for testing something out quickly
- Never use “Style” tags in a real project.f

Use Link tags when using CSS.

```
<html lang="en">
<head>
  <title>My amazing HTML Document</title>
  <link rel="stylesheet" href="./style.css" />
</head>
<body>
  <h1 class="my-brand">Check this out</h1>
  <!-- Your amazing HTML here -->
</body>
</html>
```

-rel=”stylesheet” is letting you know that it is a stylesheet=CSS.
-”href” means where the stylesheet/CSS is.
-”.” means a file before.

“Style.css” file

```
.my-brand {
  color: red;
}
```

- Better for organization.

Examples to when to Actually Use the “Cascade”-

(An algorithm that defines how to combine property values originating from different sources)

```
<style>
  .ex-btn {
    background-color: whitesmoke;
    border: 2px solid pink;
    padding: 4px 15px;
    border-radius: 5px;
    font-weight: bold;
    font-size: 17px;
    cursor: pointer; /* changes the mouse when you hover the button */
  }

  .ex-btn-warn {
    color: white;
    background-color: crimson;
    border-color: darkred;
  }

  .ex-btn-success {
    color: white;
    background-color: limegreen;
    border-color: blue;
  }
</style>

<button class="ex-btn">Default Button</button>
<button class="ex-btn ex-btn-warn">Warn Button</button>
<button class="ex-btn ex-btn-success">Success Button</button>
```

result

-”padding: 4px 15px” means

-top, bottom padding = 4px

-left, right padding = 15px

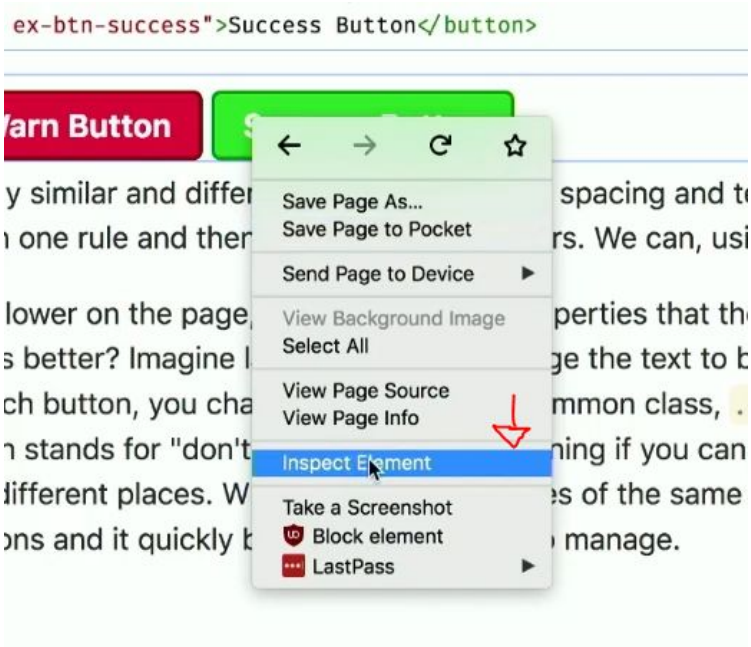
- Padding order goes by 1. Top 2. Right 3. Bottom 4. Left

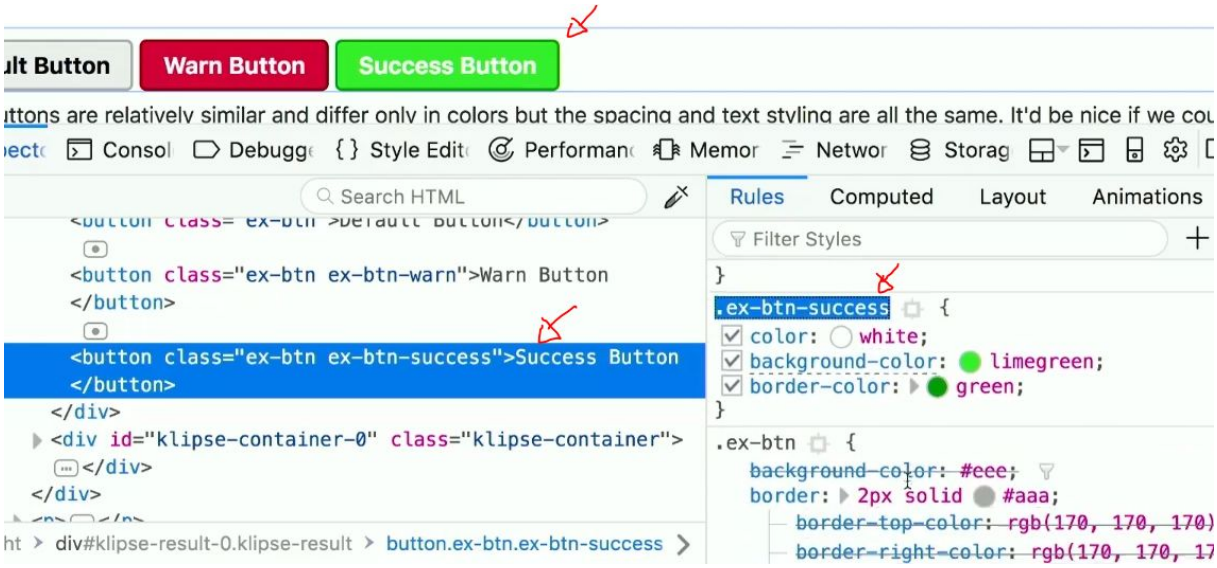
-Because the basic structure of all three buttons are the same, you can change the font by changing “font-weight: bold” in the “.ex-btn” section instead of changing all three buttons individually.

- This is the occasion where you want to use the “Cascade” lightly/minimal.
- If you are “Cascading” make sure they live in the same file next to each other. Or else, they will be unorganized and confusing later on.

DevTools- Tools that are built within the internet engine (Chrome, FireFox etc). Great for future purposes for example, network speed, etc).

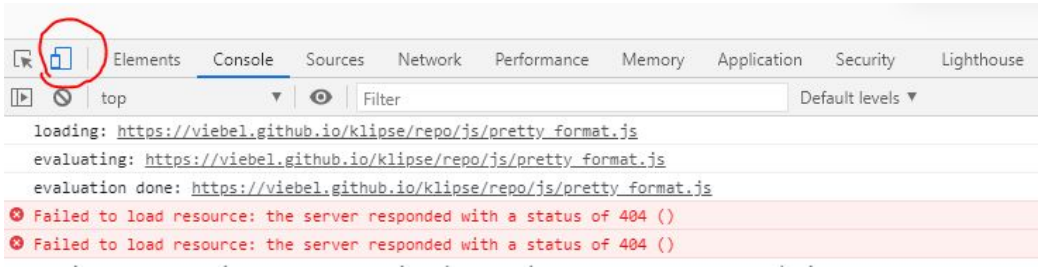
1) Right Click something inside the internet





-Takes you directly to the button.

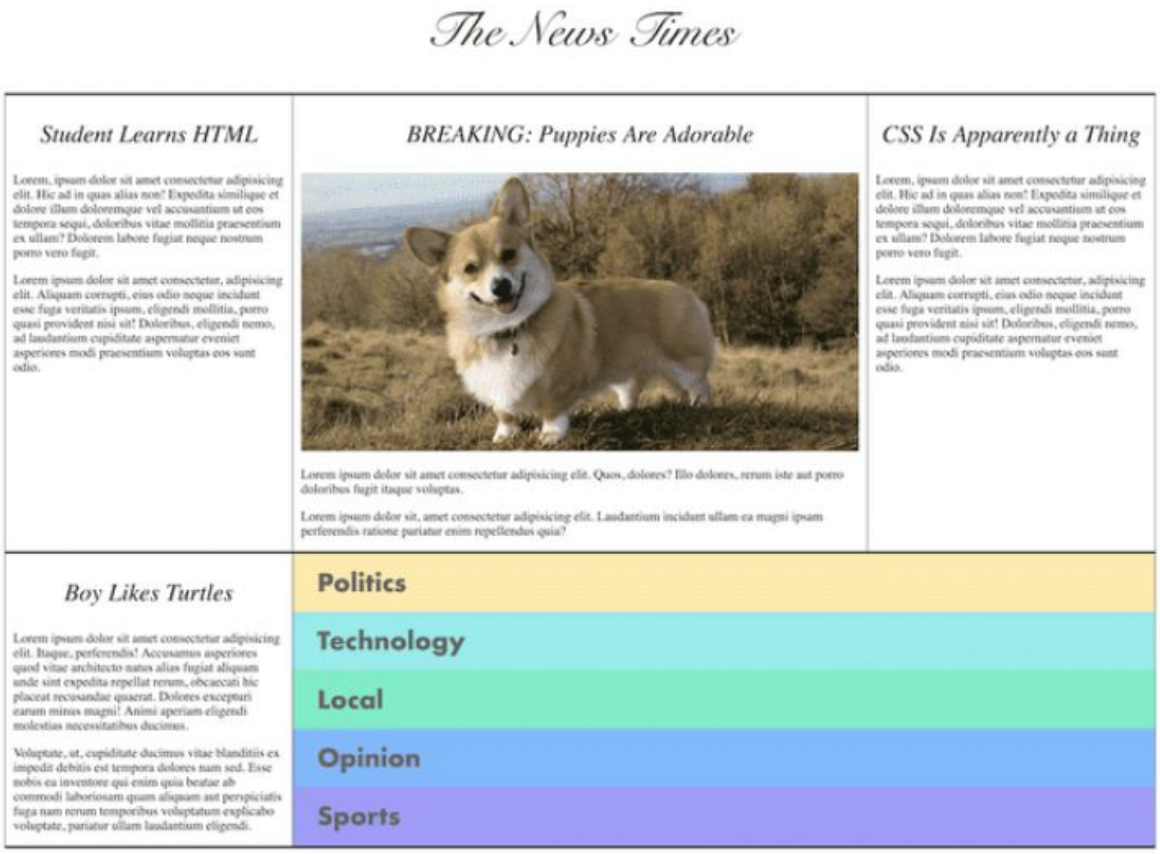
- You can experiment around with it to figure out what you want to do first. Then copy everything and apply it to the editor (VS code).



-You can see versions of different mobile devices. (Great for debugging what phone version looks like)

HTML & CSS Project Exercise-

We will be making this layout with HTML and CSS



Exercise Link- https://btholt.github.io/intro-to-web-dev-v2/project-html-css

Usually, a designer will hand you out something like this and will say, “Hey, I made this through Photoshop, InDesign etc. Can you create a page something like this?”

1) Open VS code, create a new file inside the new folder “fem”. Save file as “index.html”

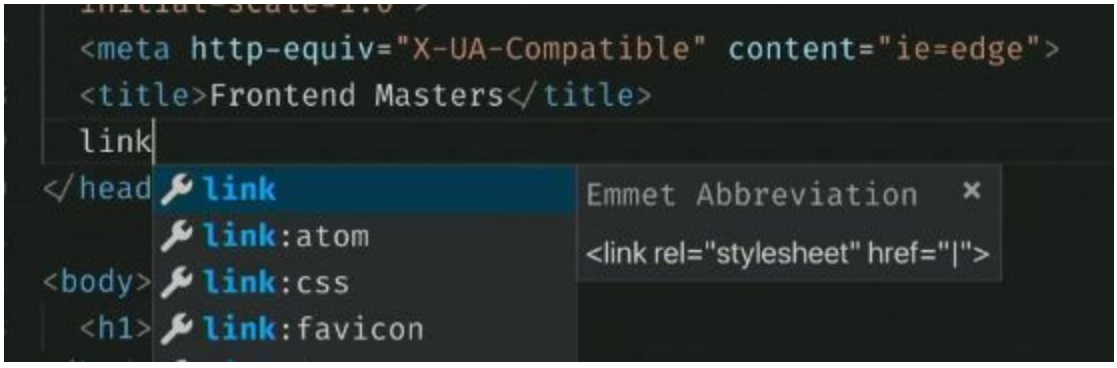
Tips: After you saved new file as “index.html”, If you press “html:5” and tab,

It generates the backbone of html:5.

result:

(You can also type “!”)

2) Create “link” for CSS

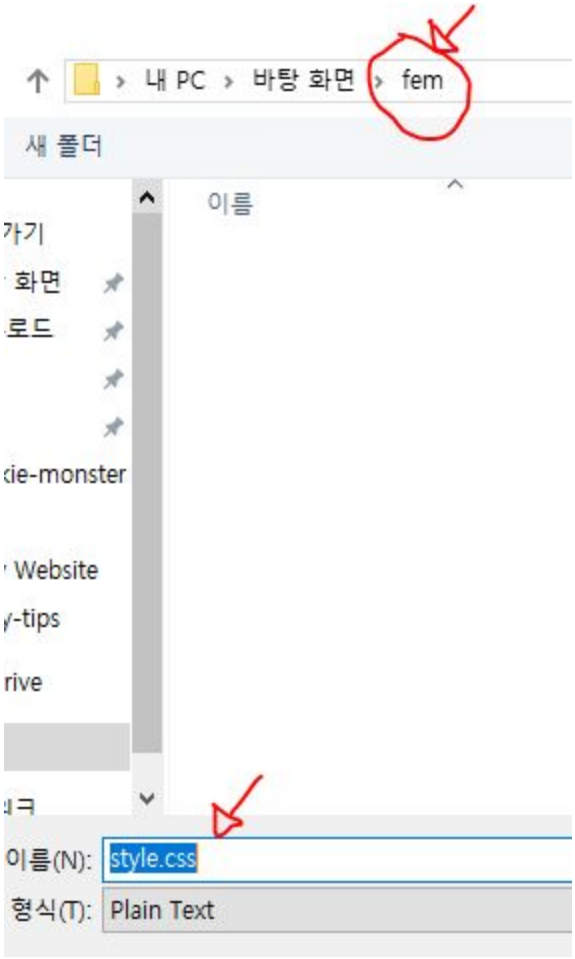


-if you type “link” and tab, it will generate

for your style sheet.

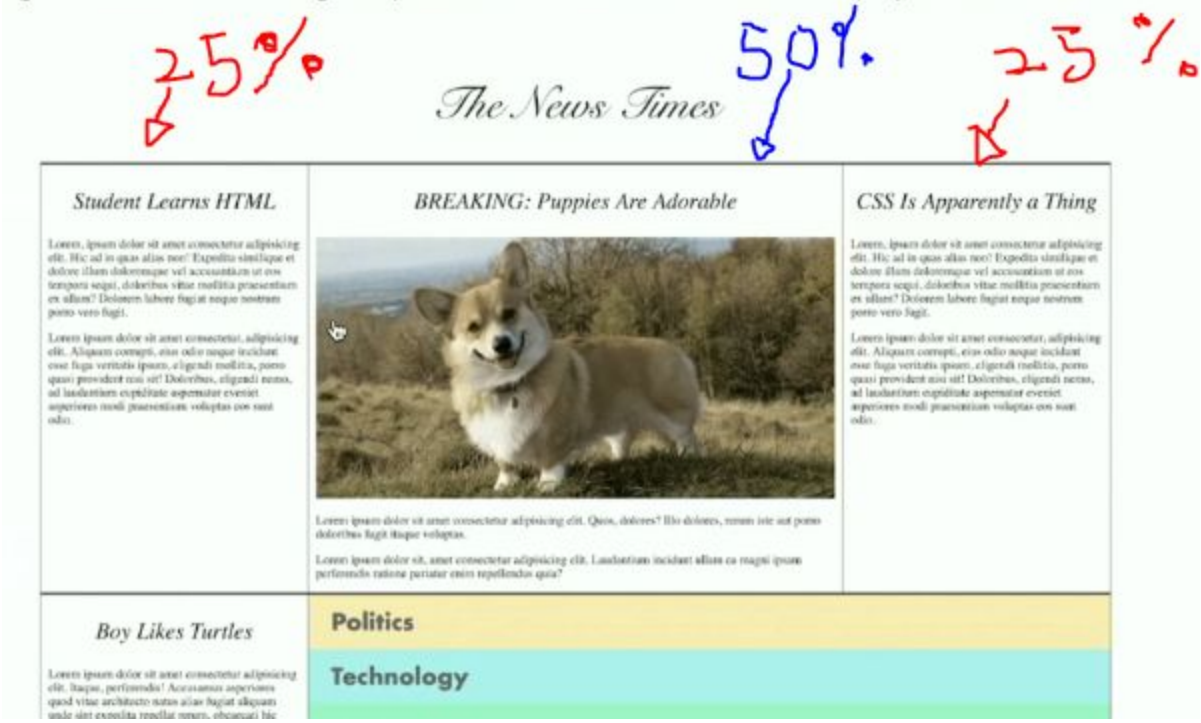
```
<link rel="stylesheet" href="./style.css">
```

Create a new file and save as “style.css” inside the “fem” folder.

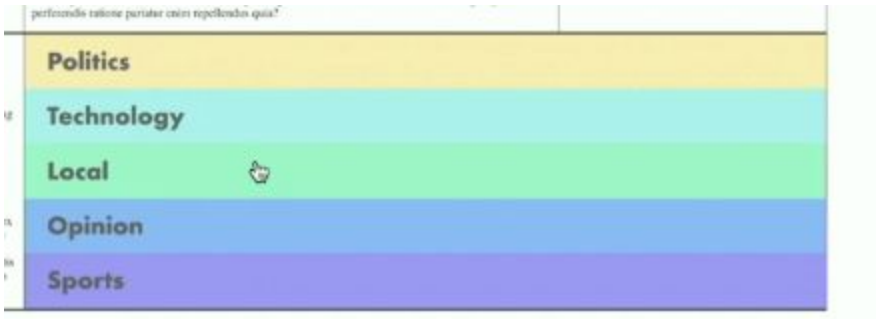


3) Name the <title>

ng to make a news site together, similar to The New York Times. Your project will look like this:



-Finish the top row first before going into the second row.



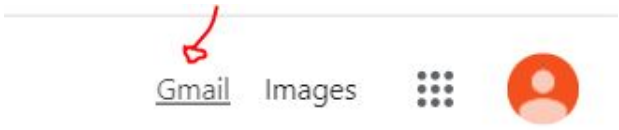
-The five lists are equally spaced. (flex-direction: column;)
-Google EVERYTHING.
-The 5 listed above are the hardest part. You put a nested display: flex inside a display flex.

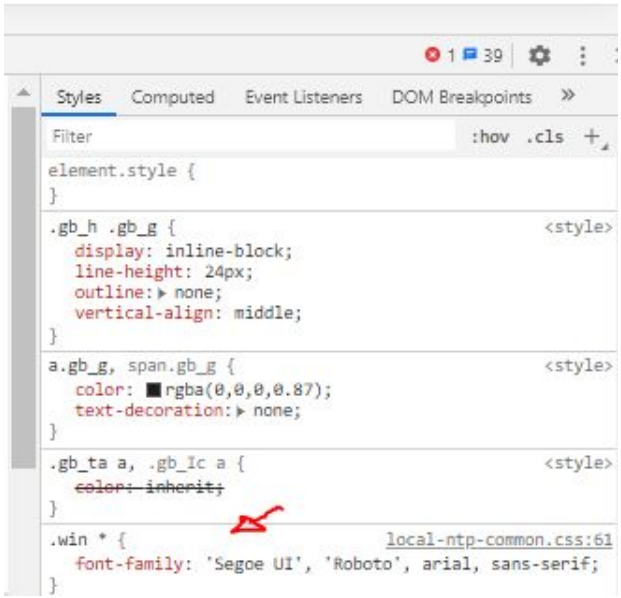
Random Dog Pictures Website: http://placecorgi.com/

-The way to fix the image size, put
http://placecorgi.com/ `<width>` / `<height>`
ex) http://placecorgi.com/500/300

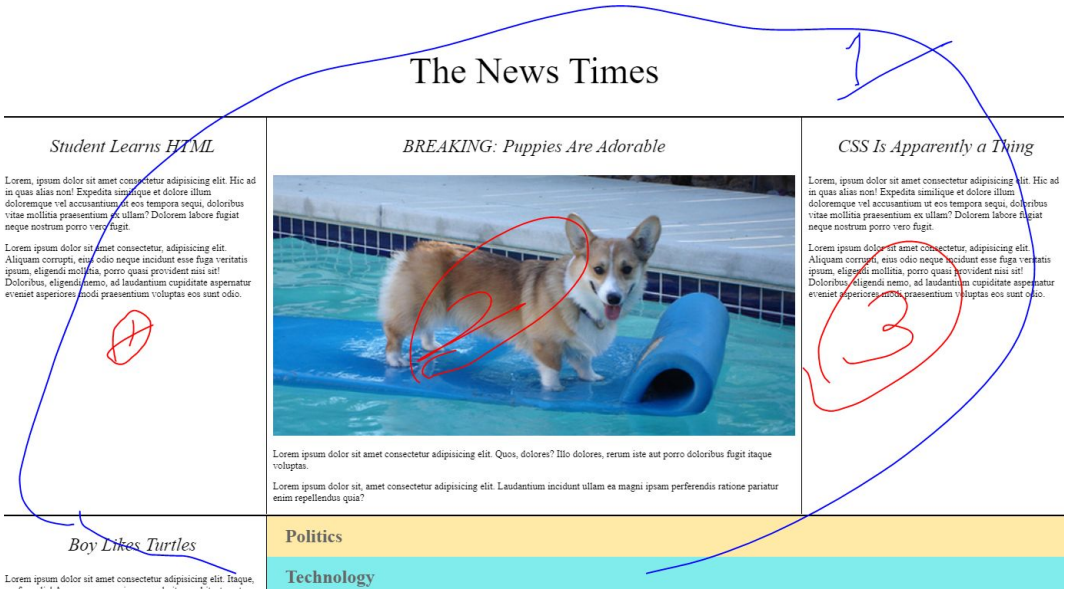
Completed Project Link:
https://btholt.github.io/intro-to-web-dev-v2/news.html

-There is no “correct” answer. There are many different ways to solve it.





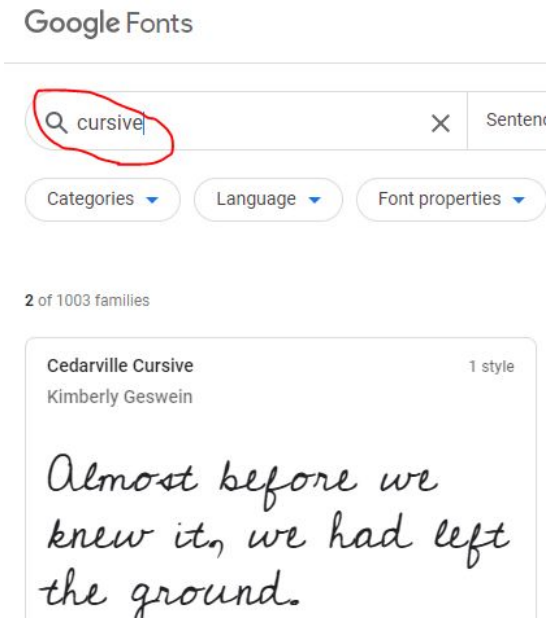
-You can check the website’s “font-family”.



-It is better to divide the problem into smaller parts instead of seeing the problem as a whole.

Google Fonts-

1. Link: https://fonts.google.com/
2. Find or Search a Font



3. Select Style

4.

5. Put “1” in the HTML file and “2” inside CSS.

```
<link href="https://fonts.googleapis.com/css2?family=Cedarville+Cursive&display=swap" rel="stylesheet">
```

```
.Main-Title{
    font-family: 'Cedarville Cursive', cursive;
    font-size: 60px;
    text-align: center;
    font-weight: normal;
    margin: 60px, 0;
```

6. Result

The News Times

```
.main-brand {
  font-size: 60px;
  text-align: center;
  font-weight: normal;
  font-family: "Great Vibes", cursive;
  margin-top: 60px;
  margin-bottom: 60px;
}
```

Or

```
margin: 60px 0;
```

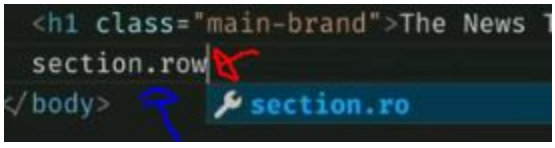
instead of

```
margin-top: 60px;
margin-bottom: 60px;
```

First Part-

* <Div> and <Section> are the same. <Section> may describe a bit better when coding “sections”.

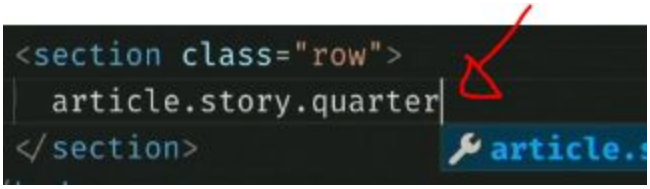
```
<body>
  <h1 class="main-brand">The News Times</h1>
  <section class="row"></section>
</body>
```

Shortcut Keys- 

+ *tab*

- Picture above means “section” “class” name equals “row”.

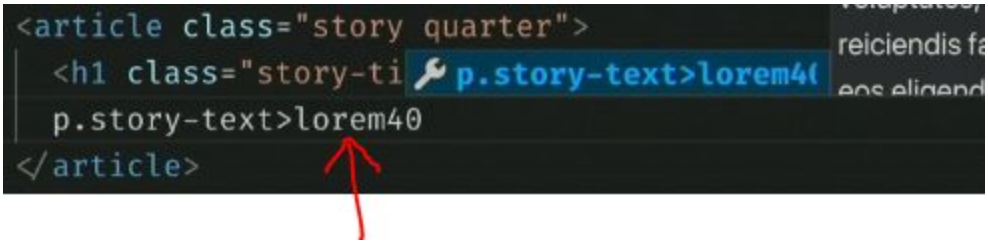
```
<section class="row">
  <article class="story quarter"></article>
</section>
```

Shortcut Keys- 

+ *tab*

- Inside “article”, there are classes named “Story” and “Quarter”.

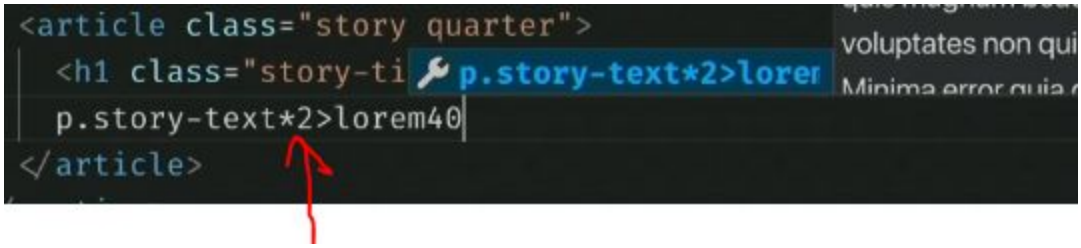
```
<article class="story quarter">
  <h1 class="story-title">Student Learns CSS!</h1>
  <p class="story-text">Lorem, ipsum dolor sit amet consectetur
  adipisicing elit. Hic, non. Accusantium quaerat eaque harum
  iusto repellat pariatur consequuntur, officiis tenetur, ex ad
  sint. Ea, optio rem iusto inventore odit voluptates, possimus
  quod reiciendis facilis sunt nesciunt eos eligendi harum iste?
  </p>
</article>
```



Shortcut Keys-
+*tab*

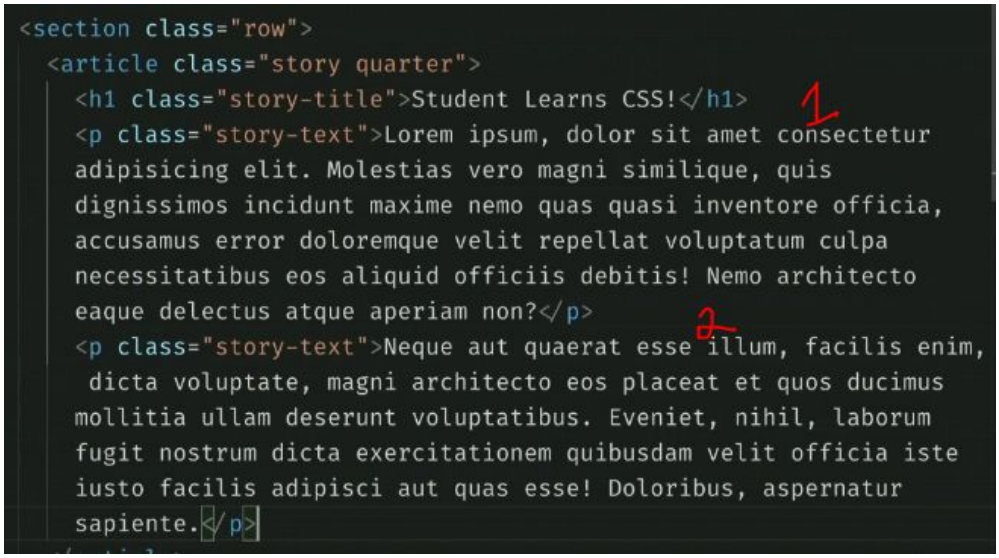
- Paragraph(“p”) has a class name “story-text” that has a “lorem” of 40 words.

If you want two of those,

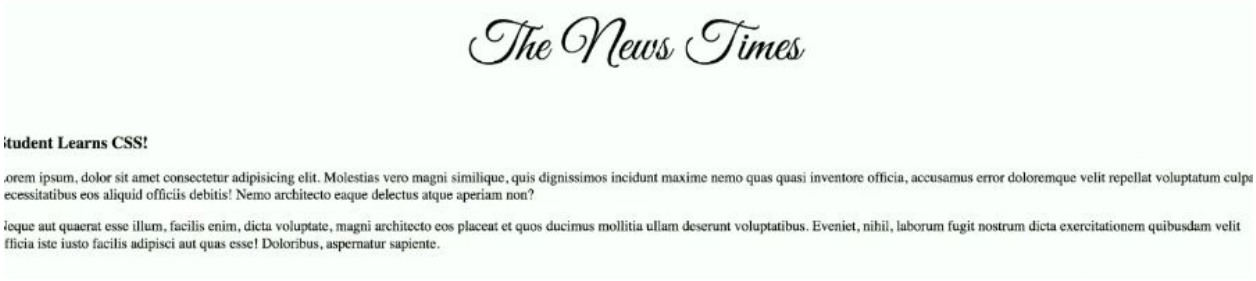


+ *tab*

Result-



Result-



Now we will write the first row and HTML First and then do CSS to avoid confusion.

Half Part-

```
<article class="story half">
  <h1 class="story-title">Dogs are Adorable</h1>
  <img src="http://placecorgi.com/500/300" alt="adorable corgi
  dog" />
  <p class="story-text">Lorem ipsum dolor sit amet consectetur
  adipisicing elit. Beatae dolore ab aliquid distinctio fugiat a
  rem, dolores
    iusto mollitia incidunt omnis, voluptatum molestias debitis.
    Veniam error labore quo voluptate aspernatur eligendi,
    sequi accusantium consequatur esse sunt. Molestias labore hic
    quibusdam?</p>
</article>
```

Result-

Third Part- Exactly Same as the First Part Section

(Copy and Paste First Part)

```
<article class="story quarter">
  <h1 class="story-title">Student Learns HTML!</h1>
  <p class="story-tex
  adipisicing elit. M
  dignissimos incidun
    maxime nemo quas quasi inventore officia
    doloremque velit repellat voluptatum culpa necessitatibus
    eos aliquid officiis debitis! Nemo architecto eaque delectus
    atque aperiam non?</p>
  <p class="story-text">Neque aut quaerat esse illum, facilis enim,
   dicta voluptate, magni architecto eos placeat et quos ducimus
  mollitia
    ullam deserunt voluptatibus. Eveniet, nihil, laborum fugit
    nostrum dicta exercitationem quibusdam velit officia iste
    iusto facilis adipisci aut quas esse! Doloribus, aspernatur
    sapiente.</p>
</article>
```

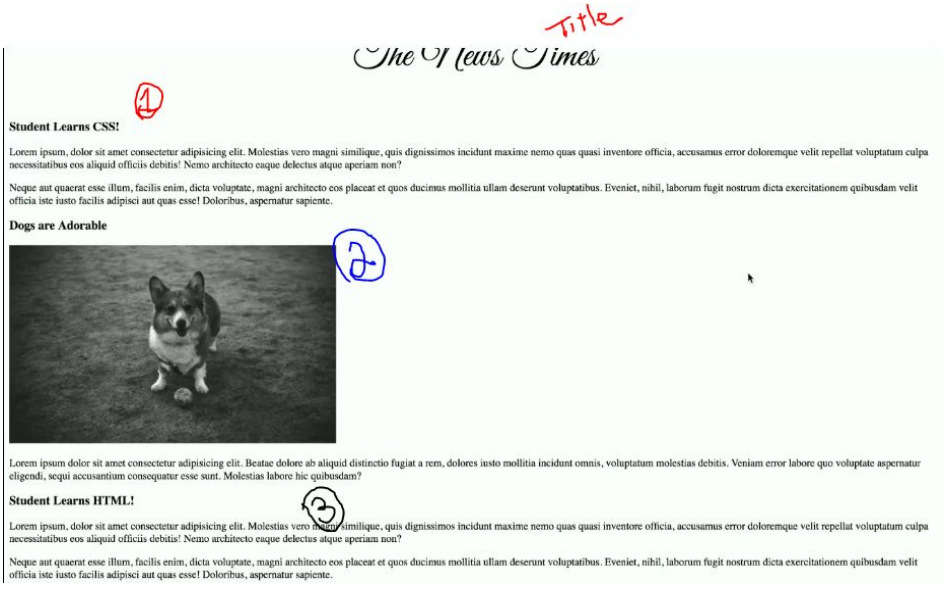
Emmet Abbreviation

<HTML>|</HTML>

- HTML
- HTML:5
- HTML:xml
- html

-Just change the Third Part Article to “story quarter”.

Result-



CSS Styling (HTML & CSS Project Solution)-


```
.row {
  display: flex;
  align-items: stretch;
}

.story {
  border: 1px solid #333;
  padding: 5px;
}
```

-align-items: stretch because three sections all have the same “height”.

```
.story-title {
  font-weight: normal;
  font-style: italic;
  font-size: 30px;
  margin: 15px 0;
  text-align: center;
}
```

```
.story-text {
  line-height: 1.5;
}
```

-line-height- space between the texts

result:


```
31  .quarter {
32    width: 25%;
33  }
34
35  .half {
36    width: 50%;
37  }
38
```

result:

To make the image centered:

```
<article class="story half">
  <h1 class="story-title">Dogs are Adorable</h1>
  <img class="story-image" src="http://placecorgi.com/500/300"
  alt="adorable corgi dog" />
```

- Make a class name for the image

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

Result:

The image will “stretch” to a 100% inside the border-box to fill in the width.

```
.story-text {
  line-height: 1.5;
  font-size: 18px;
}
```

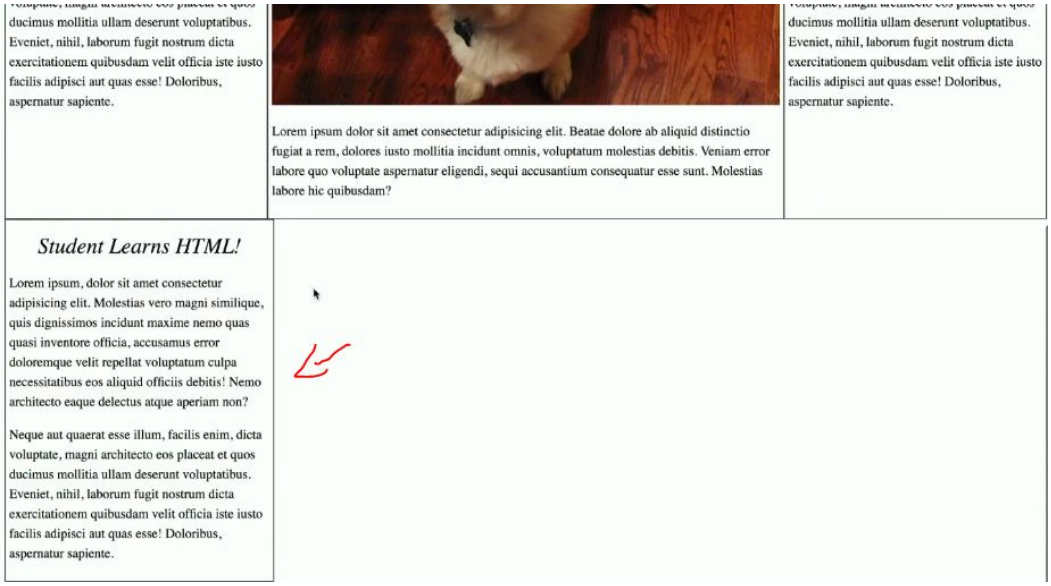
-To make the font-size a little bit bigger.

The Bottom Row- Copy and Paste First part of Row

HTML-

```
<section class="row">
  <article class="story quarter">
    <h1 class="story-title">Student Learns HTML!</h1>
    <p class="story-text">Lorem ipsum, dolor sit amet consectetur
    adipisicing elit. Molestias vero magni similique, quis
    dignissimos incidunt
      maxime nemo quas quasi inventore officia, accusamus error
      doloremque velit repellat voluptatum culpa necessitatibus
      eos aliquid officiis debitis! Nemo architecto eaque delectus
      atque aperiam non?</p>
    <p class="story-text">Neque aut quaerat esse illum, facilis enim,
     dicta voluptate, magni architecto eos placeat et quos ducimus
    mollitia
      ullam deserunt voluptatibus. Eveniet, nihil, laborum fugit
      nostrum dicta exercitationem quibusdam velit officia iste
      iusto facilis adipisci aut quas esse! Doloribus, aspernatur
      sapiente.</p>
  </article>
</section>
```

Result-



Nav items - Good for lists that navigate

```
52      <ul class="nav-list">
53        li.nav-item*5
54      </ul>
55    </section>
56  </body>
57
58  </html>
```

li.nav-item*5

Emmet Abbreviation

```
<li class="nav-item">|</li>
<li class="nav-item">|</li>
<li class="nav-item">|</li>
<li class="nav-item">|</li>
<li class="nav-item">|</li>
```

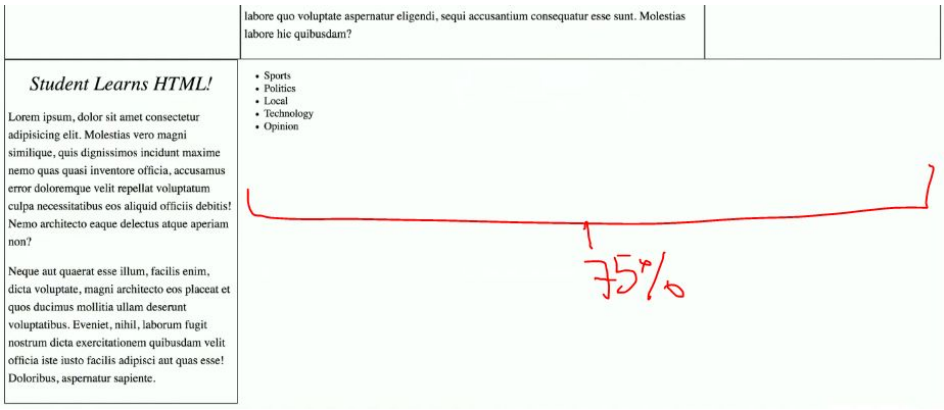
+tab

```
<ul class="nav-list three-quarter">
  <li class="nav-item">Sports</li>
  <li class="nav-item">Politics</li>
  <li class="nav-item">Local</li>
  <li class="nav-item">Technology</li>
  <li class="nav-item">Opinion</li>
</ul>
```

CSS-

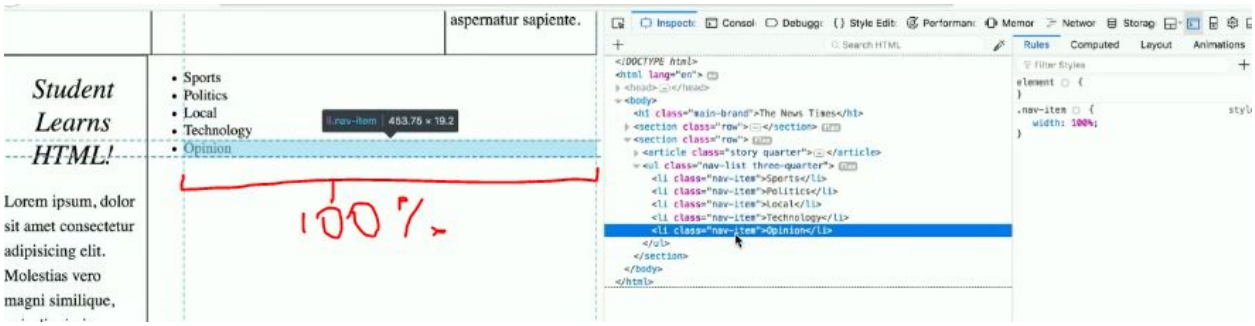
```
44  .three-quarter {
45    width: 75%;
46  }
47
```

Result-



Next- make each list 100%

```
48  .nav-list {
49    display: flex;
50    flex-direction: column;
51  }
52
53  .nav-item {
54    width: 100%;
55  }
```



How to fill up the whole section in a border-box:

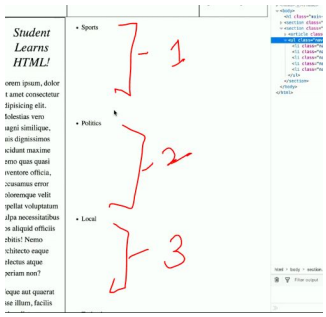
flex: 1;

What is flex: 1?

Each “1” is one item, and I want to take up as much space as possible.

```
53  .nav-item {
54    width: 100%;
55    flex: 1;
56  }
```

-CSS

Result- 

-Because There are 5 in the “.nav-item”, each will take 20% of space inside the border-box.

If you want to color each “nav-item” into different colors,

```
.nav-item:nth-child(1) { background-color: #ffeaa7 }
```

So

```
.nav-item:nth-child(1) { background-color: #ffeaa7 }
.nav-item:nth-child(2) { background-color: #81ecec }
.nav-item:nth-child(3) { background-color: #55efc4 }
.nav-item:nth-child(4) { background-color: #74b9ff }
.nav-item:nth-child(5) { background-color: #a29bfe }
```

- Alternative way is to name each lists a name and fill it below:

HTML	CSS
<li class=”list-1”>	.list-1{background-color: #ffeaa7}
<li class=”list-2”>	.list-2{background-color: #81ecec}
<li class=”list-3”>	.list-3{background-color: #55efc4}
<li class=”list-4”>	.list-4{background-color: #74b9ff}
<li class=”list-5”>	.list-5{background-color: #a29bfe}

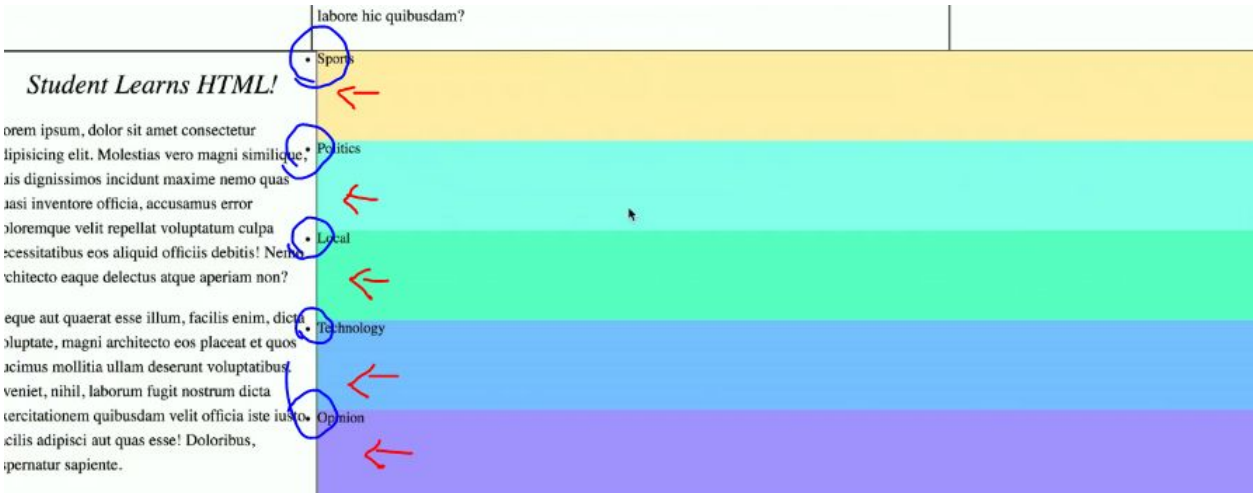
Result-

The last part still has an annoying empty space above.

-Need to make both margin and padding to 0.

```css
.nav-list {
  display: flex;
  flex-direction: column;
  margin: 0;
  padding: 0;
}
```
-CSS

Result-



Next- Take out bullets above (blue)

```css
.nav-list {
  display: flex;
  flex-direction: column;
  margin: 0;
  padding: 0;
  list-style: none;
}
```

Next- We will put the texts from “nav-list” left-center.

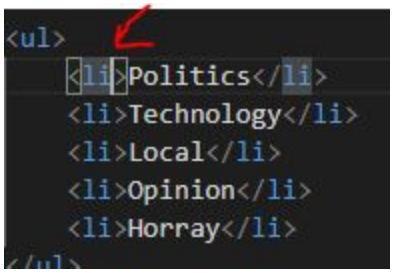
```
.nav-item {
  width: 100%;
  flex: 1;
  display: flex;
  align-items: center;
  padding-left: 25px;
  font-size: 30px;
  color: #555;
  font-weight: bold;
  font-family: Arial, Helvetica, sans-serif
}
```

-Because flex only works on “parent”, “nav-item” is the parent and “Sports”, “Politics”, “Local”, “Technology” and “Option” is the child.

-Padding-left: the left gap between the words and the box
-Font-size: Makes font bigger.
-color: text color change.
-font-weight: how thick is the font
-font-family: a list of font styles. (Will go to the next style text if the previous isn’t available)

Result-

Shortcut Keys-



1. Click
2. Hold “Alt” Keys
3. Click the place you want to write together.
4. It will type everything at once.

Next- “anchor/link” the lists.

-<a href=”#”>Opinion<a>

*(href=”#” means to link the top of the current page)

Result-

Next- take out the Blue Underline above.

or

* Just “a” works as well. It depends on how specific that you want it to be.

Result-

-The “cursor” will change once you point at the list.

Next- Make the list “hover/underlined” once you point it with your cursor.

CSS-

Result-

Finish

Learning Javascript

Programming Fundamentals-

Try to write code that is readable/maintainable for you.
Try not to be clever.
Code is communication.

Javascript is *Single Threaded* =

It means that only <u>one</u> thing is happening at a time.

ex)

-Will be read in order.

-monthlyRent is set as a variable of “500”.

ex)

-Because you already put “monthlyRent” as 500, you can’t change the variable to 600.

*”Const.log” means that you will write down the result of “(___)”
* “*” star means multiplication, not “x”.

- There can’t be any spaces in any variable names.

-This will be too hard to read later on.

-Technically, you can underscore the name.
(This is how “Python” looks for example)
It is valid. However, it’s not how we do it in JavaScript.

-This is called “Camel Casing”in Javascript.
Everything is done in Camel Casing in Javascript.

-Do *NOT* put ”commas” between numbers.

-”Semicolons” are used in JavaScript. However, it will still work without the semicolons.

-These days, programs will put the semicolons for you when you “save”.

Variables- -Stores Memory

-Variables can be named almost anything.

-However, they **can’t** be named as keywords.

ex)

-”const” cannot be named as a “variable” because it “const” is already an existing “Keyword”.

-The programming tools (VS code etc) will tell you what you can’t use.

ex)

You can directly go to “Inspect” and “Console” on your webpage (Chrome, Fire-fox etc.) and type in JavaScript

You can mess around inside the “Console”.
-When you write “JavaScript” code, the result will be inside “Console”.

-When you make a Javascript file, put an extension as “.js” in the file.

ex)

.js

Result-

-”6000” will be shown in the “Console” section.

Numbers, Strings and Booleans-

Strings- String of Characters put together-(Words, Characters etc.)

ex) -”Brian Holt” is a “String”

- You can also use the back-tick**(`)** for string as well

ex) -Still works.

- We can connect strings together using “plus” (+) signs.
  - It is almost like math with Strings.

ex)

-arrows above (red) are spaces which you will need to do manually.

There is a simpler Math Strings which is called a “**Template String**”.
ex)

-Template Strings MUST use “back-ticks” (`) for the sentence.

-With Template Strings, you can use space as much as you want.

- ${variable} - You can put as much “Curly Braces” as you want.

ex)

-Template Features is one of the new features in JavaScript.

- You can also use numbers inside the string. ex) 5
- Once a number is inside a string, “5” inside will be considered as a string.

ex)

- <!--comments for HTML-->
- /* comments for CSS */
- /* comments for JS */

Or

- // comments for JS

  -// for JS is only one line

Ex)

Booleans- “True” or “False”.

ex) If there is a smart technology for example and you want to check if the lights are “on” or “off”, you would use “Boolean”.

```
const lightIsOn = true;
```

-Booleans are useful and you will see them everywhere.

Numbers-

Different languages separate between different forms of numbers.
ex) Whole Numbers- 1, 2, 3, 5, 23, 500, 1000.
Floats-(1.2, 3.141592.., 14.01 etc).
However, for JavaScript, every number type is called “Numbers”

ex) JavaScript- (1, 2, 3.141592.., 6.71) = “Numbers”.
-Because JavaScript has only one type of “Number” it may be a bit difficult to do “precise” math with JavaScript.

Control Flow- (“If” Statements)

-Because it is “true”, only the first block in runned. (Not the else block)

-If the “skyIsBlue” is false, it only runs the else block. It doesn’t touch any of “The Sky is blue!”.

Another Example-

-Red arrow means “Triple Equals”

(===)

- This is not a good example because it can deal with only one temperature.

ex) -Red arrow means “Greater or Equal than” (<=)

-Temperature today is 86 degrees. If the temperature is 80 or higher, “it is too damn hot”. Else, “It is fine. Maybe…”. Because the temperature today is 80 or higher (86), “It is too damn hot”.

-You can use comparison here.

Difference between “=”, “==”, “===”

- A single equals “=” means that it is “assigning” values.

  ex)

- A double equals “==” will try coercion. “Coercion”- If you try to compare between a number and a string, it will “*try*” to make it work.

ex)

-The top 86 is a Number and the below “86” is a String. The Arrow is Double Equals (“==”).
-(Double Equals “==” will **also** work with the SAME type.
ex) Numbers & Numbers, String & String.

- A Triple equals “===” will **only** work with the SAME type.
  ex) Numbers & Numbers, String & String.
  -If you do triple equals “===” and if they are not the same type, it is NOT equals.

-Using triple equals “===” instead of double equals “==” can save a lot of time later on.

ex)

ex) You can also use “**!**” before the “==”equal sign for the opposite effect.

-If 2 + 2 does not equal 4 (!==), then "uh oh…".

-(!===) triple equals doesn't work for "!" opposite. (!==) Double equal works.

-You can also add more than one question as well.

ex)

-"else if" is to ask another question.

-Can be as specific/put as many “else if” as you want to.

- What if more people starts to show up at the party

ex)
-Notice that the example above is “let” instead of “Const”
-If you think the variable (friendsAtYourParty) is going to change along the road, you use “**let**” instead of “**const**”.
-Variable inside ”**const**” does not have the power to change.
-If you know you used “**const**” in variable “friendsAtYourParty” before, you can at least know later that it is the same variable as before, which can help you later at work.

-Whatever “friendsAtYourParty” was assigned, the “friendsAtYourParty” will be reassigned as +1.

-Right side of “friendsAtYourParty” will be read first, then the left side of “friendsAtYourParty” will be read second.

It would be nice if there was a better way to do this so that you don’t “copy and paste” everytime you add a person to your party.

Loops- function that allows us to execute something multiple times.

“While” Loop-

ex)

- This function above is called a “While Loop”.

- The “While Loop” looks similar to a “If” Statement.

1. Start with 0 people.
2. As long as inside “()” is ***true***, it will keep doing what is inside “{}”
3. Eventually,

   will be bigger than 10, and when it becomes ***false***, it will break-out of it and go to
4.

ex) -Because it is bigger than 10, it doesn’t go to the while loop and goes directly to “console.log”

ex) -Starts with -100 and goes all the way until it becomes “10”.

What if you make a loop example as below-

This is called an “Infinite Loop” because it does not end.
result-

-If the browser gets stuck using JavaScript, the “Infinite Loop” can be a possibility.

FYI

-1, 2, 3, 4 above means all the same. (friendsAtYourParty +1)

-3 is most common to use (For loops)

-4 is almost never used.

-For 2,

means “friendsAtYourParty + 5.

-Multiplication

-Division

-Squared 2^5

“For” Loop-

ex)

- Most common loop (for loop)- important.
- There are 3 parts of a “for loop”
    1. The first part is, with a “control variable” (“i”).

-For a “for loop”, the control variable has always been “i”. It doesn’t have to be i but everyone uses as i.

- We do not use “const” in a “for loop” because the variable changes.

ex)

2. Second Part is the “condition”

-Just like the while loop, it shows that as long as i<=10 is true, it’s going to keep doing it. As soon as it is false, it breaks out the loop.

3. Third Part
-What do you do at the end of every loop.
-”i++” means that it will increment/add by 1.

-Infinite Loop/crashed

Another example
1)

-Ash attack starts with 0 damage.
- For each lvl, Ash gains +3 atk.
-(until lvl 5)
-lvl 1 = 3 atk
-lvl 2 = 6 atk
-lvl 3 = 9 atk
-lvl 4 = 12 atk
- lvl 5 = <u>15 atk</u>

-Because the “for loop” only wants to count up to lvl 5, it exists to “console.log(AshAttack)”. There is no lvl 6.

Result-

Difference between “var” and “let”

- “Var” has function level scope.
- “Let” has block level scope.

2)

Exercise)

```
/*
  Write some code that declares two variables, character and timesToRepeat.
  Using a loop, repeat that character that many times and then console.log it.
  Example, if I had character = 'f' and timesToRepeat = 5, it'd console.log
  'fffff'
*/
```

link : https://codepen.io/btholt/pen/ELozBz

JS

It is a string of “nothing”. Needs to start from a starting point.

- tip

+= has to be in order.

-Alternative answer.

Chrome Result)

-If you want to see how many “f”s you put..

-just add “.length”

Result-

Functions and Scope

ex)

Functions-Bit of Reusable code. Just how we like to re-use CSS classes, we love to reuse code.

ex)

Make a folder for VS code for JS to practice

-alternative example (Same as above)

-note that the variables above are different from the first example. However, the results are the same

-The bottom part function is reused. It still works

- Once a function name is made, you can use the same function all over the place.

Once the function name is made, you don’t have to make the string all over again.

ex)

-Notice you can use the same function multiple times.

- Note that (“”) does not work.

1. **Function**- You are making a new “function”
2. **Name of the Function**-Try to give the name of the function a ‘descriptive’ name. ex) bestCharacters, addTwo etc.
3. **Calling a Function**- you are basically “summoning” a function

ex) even if there is nothing inside the parenthesis “()”, it still “summons” out a result.

-with No parenthesis. The function name “finalAnswer” is “addTwo” itself.

ex)

result-

If you want to change the results,

Result-

You can put the representation of the string (firstBuild) or Strings directly (“Knight’s Vow”) inside the parenthesis. Because ex) (firstBuild) represents a string anyway.

Alternative-

Result-

Scope-If you have “scope” problems, it can be big problems.

-hard concept for newbie developers and even intermediate developers.

Two types of scope- Global Scope and Local Scope

result-

Result-

Because goes outside of the function. Therefore, const “someVariable” does not exist anymore.

- Seems like if one doesn’t work inside the “Scope”, everything does not work.

What if goes “inside” the function.

ex)
Result =

- One thing/tip to know what is scope- Look for the Parenthesis”{ }”. (does not always apply).

Then what if the “console.log(someVariable) is “inside” and “outside” the scope?
ex)

Result-

- Once the scope ends, it expires everything that was declared inside of it.

What if-

Result- - It works.

Because we are still inside of that function scope. The scope is in a higher scope. Scope inside a Scope.

What if

-const otherThing is outside of the Scope function completely.
-The Result, it works.

This is called the **Global Scope- The highest level scope.**

-It is not inside of anything.

- Don’t put too much stuff from the global scope just because it works. Everything can change the global scope which can lead to some terrible bugs.

-once the “i” from the loop is over, “i” dies and cannot be reused anymore.
(After +10 of “i”, it is over for the “i”)

If a global scope exists and the same scope is inside a local scope, the scope still survives.

- If you see a “Reference Error” in your javascript file, you can suspect that it is a “Scope Error”.

Scope Exercise-

Result-

Since these two are “Global Scope”, it will survive.

E is also in the global scope.

console.log(G); // does not work, declared inside the while loop and it's over

Built ins-

- JavaScript already has a bunch of stuff pre-built for you.

As well as uppercase.

Objects- an Object is a collection of properties

- So far, we've been talking about numbers, strings, booleans and functions. 4 things.
- Objects and Arrays are very important concepts in JS as well.
- Objects and Arrays are collection of numbers, strings, boolean and functions

In real life, a car is an **object.**

All cars have the same **Properties**, byt the property **values** differ from car to car.

- An Object can be Chris or SJ

ex)

-Commas are required when writing an object.

Alternative

Ex 2)

Alternative-

Result-

- You can have “Objects” within “Objects”

Why do you want to use “Objects”?

-Objects are extremely useful in JavaScript because they’re how you will group together like-information so that they can be used together.

ex)

*Age Range has to be “exact”

- As you can see, you can combine using different objects, functions or statements to work together.

- **Objects can be even more useful when integrating with servers and APIs.**

- Objects can even have their functions!

=

Objects can have “nested” objects inside of them.

-Similar as one of the examples above.

    - Nesting inside an object can go on and on.

- “Keys” in an Object have to be “Unique” while “Values” do not.

- Do NOT use “=” inside objects

ex)

Context- It would be nice to utilize the above object to print out nicely where the person was from for example, “Shipping” address from the mail.

-This is our first time seeing the weird “this” keyword. This is a strange, complicated and difficult concept in JavaScript known as context and can confuse all kinds of people. It is new and old to the language.

-If you decide to pursue a career as a developer, interviewers will often ask questions about context in JavaScript. It's worth investing time to understand how it works.

-Context gets hard and gets harder.

Arrays-

If "Objects" are un-ordered collections of datas using "keys" and "Values", Arrays are **ordered collection of data**. If you put something in an array, it has an order(순서).

ex) this is the first thing, second thing, third thing etc)...

- Arrays starts the order number from "0" Zero.
- Therefore index 1, means the 2nd one.

- If you see a square bracket, you can tell that it is an Array
    - Array uses Square Brackets “[ ]”

Arrays also have a bunch of “built ins”.

ex)

- “Join” puts the array into 1 “String”.

What if you want to “Add” an element to an “Array”

-There is a built in called “Push”

ex)

ex-2)

Result-

-Notice the course is changed to v4 from v3.

Alternative from above function. May make more sense.

Also can work utilize w/ loops.

Ex 1)

Result-

You will see these examples a lot.

Ex 2)

The “DOM”- We will now start using JavaScript to interact w/ your webpage.

- The DOM is the way that JavaScript interacts w/ HTML & CSS.
- (Like a bridge between the two)

We will discuss between a normal code and internet browsers

1. You right code in your editor (VS Code)
2. At some point, puts the code out of the server [ex) Bluehost, Hostgater, GoDaddy, SiteGround etc]
3. ex) chriseun.com- Server sends a copy of HTML and the browser reads the HTML
4. It sees that it has a script server ex) script.**js**.
5. It makes another request back to the server and asks for “script.js”
6. It starts reading and executing javascript (Same process as CSS).

- When you use VS code at home by yourself, your computer is acting as if it is a server to itself.-It is pretending that it is a server out there.

Query Selector-

- Query selector is more like a game converter? (ps4 cd to xbox console)
- It reads most CSS.
- It gets the first instance on that page-(If no matches are found, “null” is returned)

ex-1)

ex-2)

result-

- document.querySelector
- querySelectorAll - Will select everyone on the page.

CSS-

JavaScript- (camelCased)

- You can do a lot more with it
    - Where it is positioned on the page, changing style, inner HTML etc).

What if you wanted to change more than 1?

ex)

result-

Event Listener-

- addEventListener - I’m waiting for this event to happen
- Whenever a ‘click’ happens on this “button”, run this “function”.
- Inside the function, there is an ‘alert’ saying “Hey there!”
- We grab the button via `querySelector` and store it in the JavaScript variable `button`.
- We then call the `addEventListener` method on the button. This takes two parameters (no need to memorize this, you can always look it up): the name of the event you want to respond to, which in this case is the `click` event, and a function that is called whenever that event happens. This function is often called a callback because it gets called back whenever the event happens.

- We then call a function called `alert`. `alert` is a super, super annoying function that pops up a dialog window with whatever you call it with.
- People often get confused seeing `});` on the last line. The first `}` is closing the function, the second `)` is closing the function call of `addEventListener`, and the `;` ends the statement.

- Key Up is whenever you lift up your finger, whatever you were writing, it will copy.
- There is a word called “keydown”. It is annoying because it always types in one key slower.

- We're reference `input.value`. The value property of an input reflects whatever the user has typed into the input.
- We're taking whatever is in `input.value` and passing that directly into the `paragraph.innerText`. Since that function is called every time a user types into the input, it keeps the two in sync!

ex)

- Similar to above. The key difference here is that we're listening for `change` events.

- `change` events happen whenever a user types something in the input and then unfocuses the input by clicking somewhere else or hitting tab to change the focus.
- Try typing "red" and then clicking somewhere else. Also, try something that isn't a color.
- Notice that if you give it an invalid color it just doesn't change anything.

-hexadecimals works as well

Event Delegation

-Pretty important to Junior Developer Questions

- If you have a bunch of elements that you need to listen for events on, you could attach an event listener to each but that's a bit tedious to do.
- Instead what is sometimes easier to do is to use what's called event bubbling.
- When an event fires on an element, after that "bubbles" up to its parent, and then its parent, and its parent, etc. until it's at the root element.

NODE

*Bash is case sensitive

cd - change directory
Cd .. - one file backwards
Ls - what is in the directory
Cd ~ -go to home directory
Ls - a - find hidden files
Ls -lah - show it in list style
Clear- clear list
mkdir some-cool thing
Ls = creates a file some-cool-thing
Touch index.html - makes an index file
Echo “<h1>Hello, this is cool</h1> > index.html - you can create inside bash
Code index.html - goes inside visual studio what you wrote above.
Vi index.html- looks like the whole html file that you can code inside bash.
PRESS “i” to modify the “vi index.html”.

Ctrl + c - exit

After you press the esc button, you will escape the modifying mode.
Then type “:q” to exit
Two choices:
:w will save file
:q! Means still going to quit

Inside visual studio

Req -means request
Res- response (sending back to the user)
Res.end -ending user