

WORKSHOP 1:

Front End Web Technologies

May 14, 2022

Ian McLean

www.linkedin.com/in/ian-bt-mclean/

Who am I???



- Studied biology (MSc at Carleton - 2011)
- Decided I wanted to learn about computers and software development
 - BCS program at UBC - 2nd degree program (2013 - 2017)
- Sr. Software Developer (originally QA developer co-op) at D2L (2015 - present)
 - I do whatever is needed (I was once labelled as “Human Shield” for 2-? months) ... currently investigating possible ML product opportunities
 - Currently working as a Lead Technical Trainer (a title I have made up)
- Co-creator/developer/teacher of this course at UBC (2018 - present)

THANK YOU!

Communication and Compassion



Not communicating with others and being disrespectful/impatient/non-compassionate :(



Making sure to communicate frequently, and being respectful and patient towards others, and treating others with compassion :)

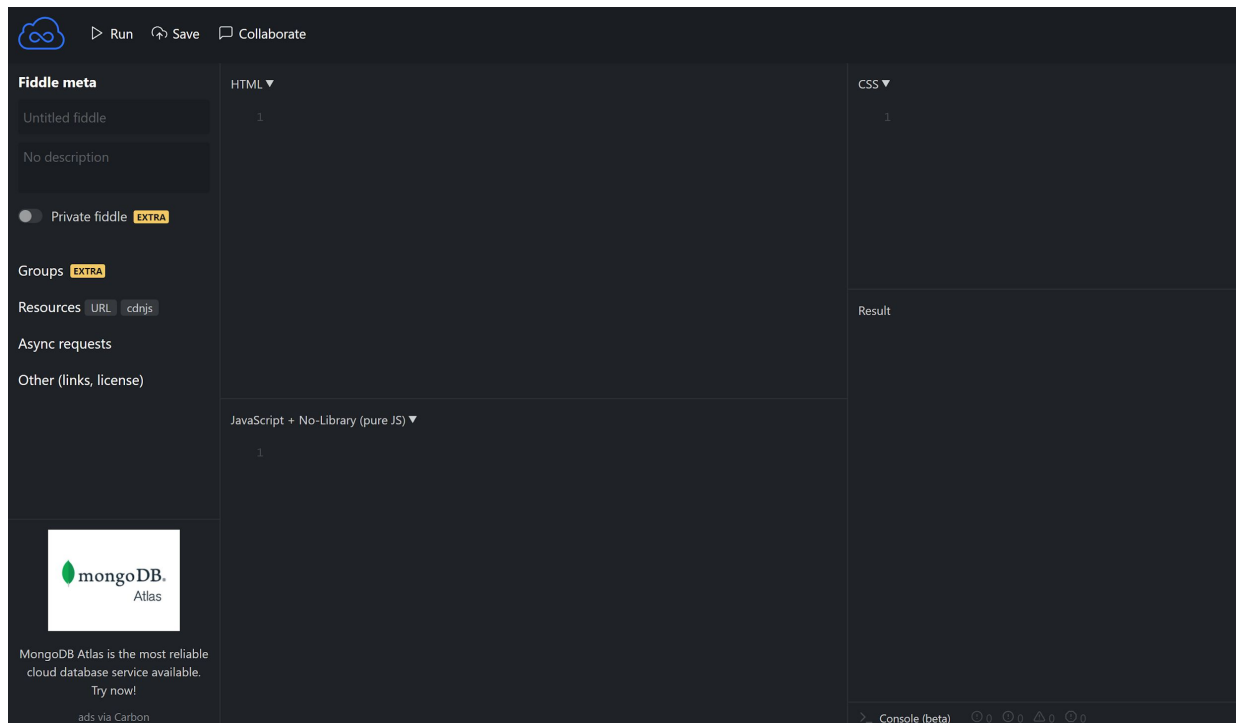


Image taken from https://en.wiktionary.org/wiki/tip_of_the_iceberg#/media/File:Iceberg.jpg on May 9, 2019

- I can't possibly teach you everything about HTML, CSS, JS in 2-3 hours (also, I don't know everything ... still learning myself)
- This will cover the tip of the tip of the iceberg (but hopefully enough to make you somewhat dangerous)
- I'm likely going to need to move quickly/skip some slides due to time (I apologize)
- We'll do our best to answer all your questions during breaks or throughout the course on Slack/in lab/at other workshops

Follow along!

- <https://jsfiddle.net/>



F12 - Dev tools

Canada

Advertising Business Privacy Terms Settings

Elements Console Sources Network Performance Memory Application Security Audits Redux aXe

```
<!DOCTYPE html>
<html itemscope itemtype="http://schema.org/WebPage" lang="en-CA">
  <head>...</head>
  <body class="hp vasq" onload="document.f&&document.f.q.focus();document.gbqf&&document.gbqf.q.focus();if(document.images)new Image().src="/images/nav_logo242.png" id="gsr">
    <div class="ctr-p" id="viewport">
      <div id="doc-info"></div>
      <div id="cst">...</div>
      <textarea name="csi" id="csi" style="display:none"></textarea>
      <style>@media only screen and (max-width:580px){#gb div{display:none}}</style>
      <div class="jhp big" id="searchform"></div>
      <div class="sfbgx"></div>
      <div id="gac_scont"></div>
      <dialog class="spch-dlg" id="spch-dlg">...</dialog>
      <div style="display:none" jsl="St t-orNZyHXTT74;$x 0;" class="r-iehFjklzl_o0"></div>
      <div class="content" id="main">
        <span class="ctr-p" id="body">
          <center>
            <div style="height:233px;margin-top:89px" id="lga">...</div> == $0
            <div style="height:118px"></div>
          </center>
        </span>
      </div>
    </div>
  </body>
</html>
```

html body#gsr.hp.vasq div#viewport.ctr-p div#main.content span#body.ctr-p center div#lga

Styles Computed Event Listeners DOM Breakpoints Properties Accessibility

Filter :hov .cls +

```
element.style {
  height: 233px;
  margin-top: 89px;
}
div {
  display: block;
}
Inherited from center
center {
  display: block;
  text-align: -webkit-center;
}
Inherited from body#gsr.hp.vasq
body, html {
  font-size: small;
}
```

HTTP CODES

200 - you win! Everything is a success!

400s - client side error :(

- 400 - Bad request (happens a lot with API calls)
- 401 - Unauthorized (wrong password!)
- 403 - Forbidden (You aren't allowed to see this)
- 404 - Not found (Resource isn't available for any number of reasons)

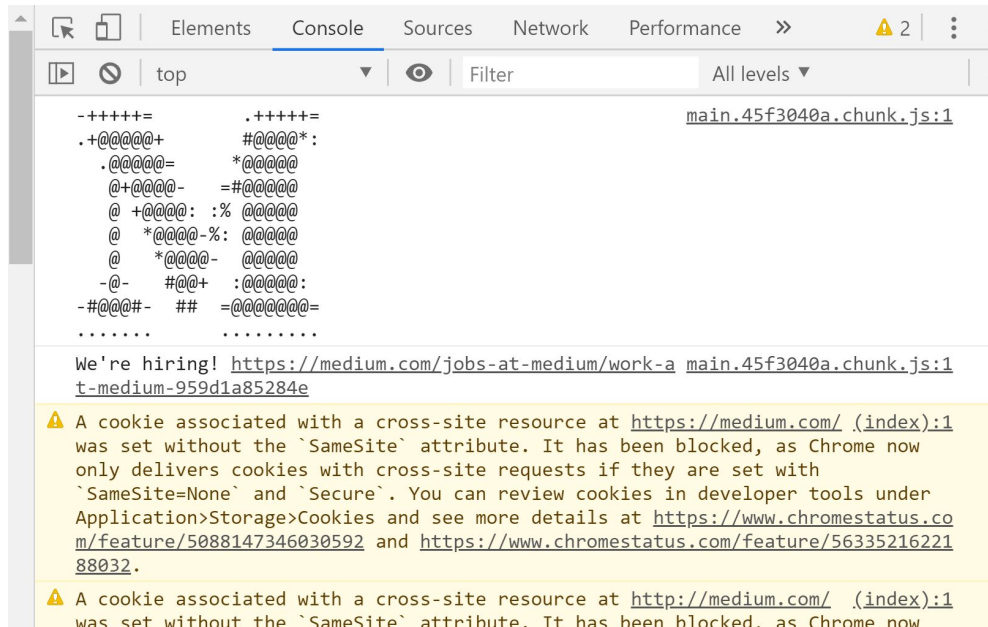
500s - server side error :(

- 500 - Internal server error (general error for something went wrong on server)

Quick exercise

Use dev tools on sites you use

- Fun ones to try
 - Medium
 - Facebook
 - Any others?
- Fun to try removing paywalls :)



Basic web technologies

HTML



CSS



JS



HTML



What is HTML?

Hypertext Markup Language

- It provides the content for a webpage
- It also describes the structure of the webpage

Let's break it down

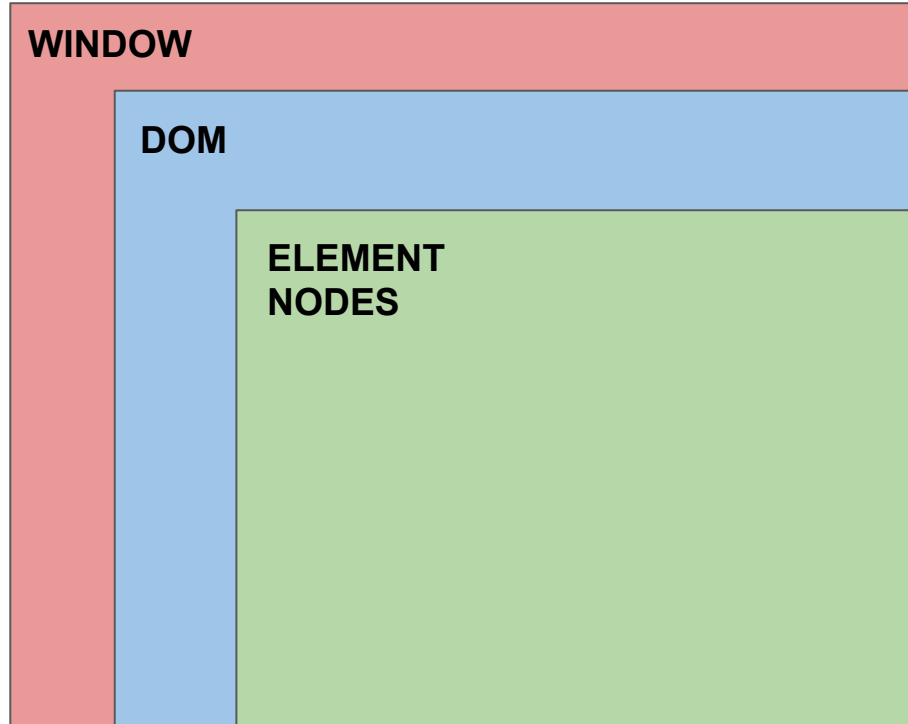
Hypertext = text with references to other text (hyperlinks)



Markup Language = a way to label a document, and provide extra information about the document and its content (semantic and presentational meaning)

Bonus ... Hypermedia = hypertext + videos/images/audio/etc. which can also lead to other items through hyperlinks

A quick browser window breakdown

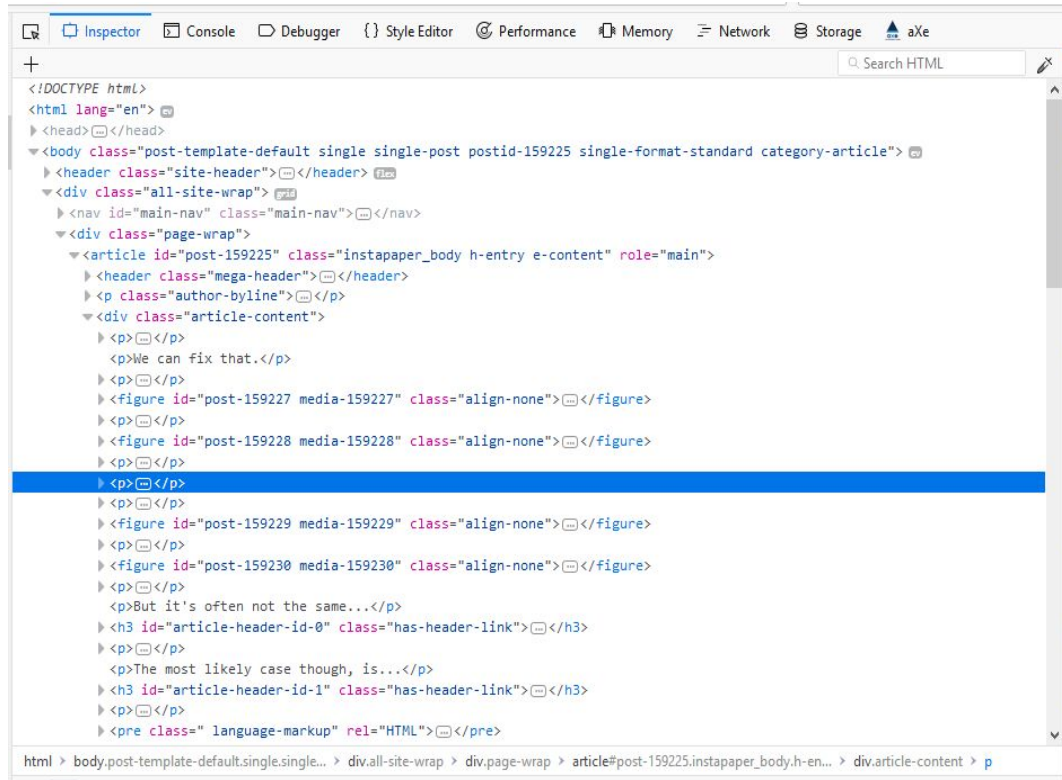


DOM

Document Object Model

- Your HTML is turned into the DOM (with CSS changing the appearance and JS allowing you to do stuff to it)
- Elements are turned into nodes

<https://css-tricks.com/dom/>



```
<!DOCTYPE html>
<html lang="en">
  <head>
  </head>
  <body class="post-template-default single single-post postid-159225 single-format-standard category-article">
    <header class="site-header">
    </header>
    <div class="all-site-wrap">
      <nav id="main-nav" class="main-nav">
      </nav>
      <div class="page-wrap">
        <article id="post-159225" class="instapaper_body h-entry e-content" role="main">
          <header class="mega-header">
          </header>
          <p class="author-byline">
          </p>
          <div class="article-content">
            <p>
            </p>
            <p>We can fix that.</p>
            <p>
            </p>
            <figure id="post-159227 media-159227" class="align-none">
            </figure>
            <p>
            </p>
            <figure id="post-159228 media-159228" class="align-none">
            </figure>
            <p>
            </p>
            <p>
            </p>
            <figure id="post-159229 media-159229" class="align-none">
            </figure>
            <p>
            </p>
            <figure id="post-159230 media-159230" class="align-none">
            </figure>
            <p>
            </p>
            <p>But it's often not the same...</p>
            <h3 id="article-header-id-0" class="has-header-link">
            </h3>
            <p>
            </p>
            <p>The most likely case though, is...</p>
            <h3 id="article-header-id-1" class="has-header-link">
            </h3>
            <p>
            </p>
            <pre class=" language-markup" rel="HTML">
            </pre>
          </div>
        </article>
      </div>
    </div>
  </body>
</html>
```

Elements and Tags

Elements are how you break up content within an HTML document

- A paragraph, a button, a heading ... all of them are individual elements

Tags are how you create an element (open and usually a closing tag)

- For example, for a paragraph element:

```
<p> I am a paragraph! </p>
```

- Sometimes, you don't need that closing tag (if there isn't content between the tags)

Attributes

Provide more information for a tag:

```

```

There are global attributes (usable by every html element):

- id, class, tabIndex, dir, etc.

There are more specific attributes usable by specific elements:

- src, href, width, height, etc.

There are event attributes (I think usable by every element, with some modification):

- onclick, onfocus, onblur, etc.
- Div can't receive focus, unless you set a tabIndex

Comments

```
<!-- I AM A COMMENT -->
```

A basic HTML page

```
1  <!DOCTYPE html>
2  <html>
3      <head></head>
4      <body></body>
5  </html>
```

Doctype = lets the browser know it's an html document, and what version (HTML5)

Html tag = houses the entire html document

Head tag = a place to put document information and important resources (e.g. title, styles, links (to resources), etc.)

Body tag = where all the visible content lives

A few useful tags ... of many

<p> = paragraph

 = line break

 = important text (bolds the text)

 = emphasized text (italicizes the text)

 = unordered list

 = ordered list

 = list items

<h1> ... <h6> = headings

Block vs. Inline

Default display value for most elements is either block or inline

Block:

- Starts on new line and takes the full width



Inline:

- Stays on same line, and takes minimum width needed



Extra - Hyperlinks

The anchor tag allows you to connect to other resources using the hypertext reference (href) attribute:

```
<a href="https://www.google.ca">  
  This link will take you to Bing search engine  
</a>
```

You can also:

- Wrap other elements with it (images, video, etc.)
- Use the href to find another place on the page (element id in the href)
- Use relative or absolute URLs (for going to other sites or other pages within your site)

Extra - Images

The image tag allows you to embed an image into the webpage:

```

```

- src and alt are required (alt can be empty)

There's also a video tag ... I don't think I've ever used it, but it works similarly to the img tag, but for videos :)

Extra - Div vs. Span

Div:

- Block element that is generally used as a container for other elements

Span:

- Inline element that is generally used to wrap text, or other inline elements

```
<div>  
  <p>Hello, I am a test <span> case </span></p>  
</div>
```

Extra - Form + Input (sort of a quick catch all)

Form tags create a form on the screen, and help to group together input tags

- Attach input tags with a certain type (text, button, checkbox, radio, etc.)
- Attach a submit button or input to send the data
- Make sure to check out the details around this, as there are more rules

```
<form onsubmit="add(this.text.value); return false;">
  <label>
    Text:
    <input name="text" type="text" />
  </label>
  <br />
  <input type="submit" value="Add" />
</form>
```

Extra - Iframe (sort of a quick catch all)

Iframes act as separated areas on the page the you embed other webpages (can be from the same domain as your webpage or from a different domain)

- Depending on content and security features, you could have trouble displaying the page
- You can't do anything to iframe content that is from a different domain

```
<iframe src="http://blog.linkageinc.com"></iframe>
```

CSS




What is CSS?

Cascading Style Sheets

- If HTML provides content/page structure, CSS decides how it is going to look/how it will be displayed on the page
- Style deathmatch! (according to a Stackoverflow post)
 - Called cascading because style selection will fall or “cascade” from one rule to the next, picking up styling that applies/isn’t covered by a previous rule

Inline CSS

Inline:

- You can add it directly to an element using a style attribute ... (usually) DON'T! 
- Dynamic styling where you don't know what the value might be (provided by user)

```
<p style="color: red;">I'm a red paragraph</p>
```

Internal CSS

Internal:

- You can add it to an HTML file in the head tag using a style tag ... (usually) DON'T!
- OK for small amount of CSS/single page ... still not great ... just use comments to create a section in a CSS file



```
<head>  
  <style> p { color: red;} </style>  
</head>
```

External CSS

External:

- You can import a CSS file in the head tag using a link tag ... DO IT!



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


Comments

```
/* I AM A COMMENT */
```

ID and Class

These are HTML attributes that get used with CSS to target certain elements with certain rules

ID:

- Each element **should** only have one unique id (don't need to have one)
- Targeted in CSS using “#”

```
#element-id {}
```

Class:

- Each element can have many classes, and classes are meant to be re-used with similar elements
- Targeted in CSS using “.”

```
.element-class {}
```

Selectors + Rules (Property and Value)

The diagram illustrates the components of a CSS rule. It shows a CSS rule for the 'p' selector. An arrow labeled 'Selector' points to the 'p' character. An arrow labeled 'Property' points to the 'text-align' property. An arrow labeled 'Value' points to the 'center' value. The rule is enclosed in curly braces, and the property and value are separated by a colon.

```
p {  
    color: red;  
    font-size: 20px;  
    font-family: Arial, Helvetica, sans-serif;  
    margin: 10px 20px 10px 20px;  
    text-align: center;  
}
```

Selector

Property

Value

Sooooooooo many selectors!

https://www.w3schools.com/cssref/css_selectors.asp

A trick to checking a selector

```
<p>A CSS rule-set consists of a selector and a declaration block:</p>
▶<p>...</p>
<p>The selector points to the HTML element you want to style.</p>
▶<p>...</p>
▶<p>...</p>
▶<p>...</p>
▶<p>...</p>
▼<div class="w3-example">
  <h3>Example</h3>
  ▶<div class="w3-code notranslate cssHigh">...</div>
... <a target="_blank" href="tryit.asp?filename=trycss_syntax1" class="w3-btn w3-margin-bottom">Try it Yourself </a> == $0
</div>
<hr>
<h2>CSS Selectors</h2>
▶<p>...</p>
```

html body div#belowtopnav.w3-main.w3-light-grey div.w3-row.w3-white div#main.w3-col.l10.m12 div.w3-example a.w3-btn.w3-margin-bottom

.w3-example h3

1 of 8

Animations Console What's New Network conditions

Specificity

Inline Style

ID

Class/pseudo-class/attribute

Element



Decreasing specificity

If you have equal specificity, the one closer to the bottom of the page wins!

!important ... is kind of the devil

```
color: red !important;
```

This wins ... beats everything! (Ideally, it's used only in cases where you know that a certain styling needs to stay the way it is)

BUT ... it's difficult to then beat, without also using an !important ... which is gross, and really messy

I've seen it misused more than I've seen it used right ... and I've seen it used lazily ... so, avoid using unless you absolutely have to (e.g. overriding a style that you don't control)

Box model

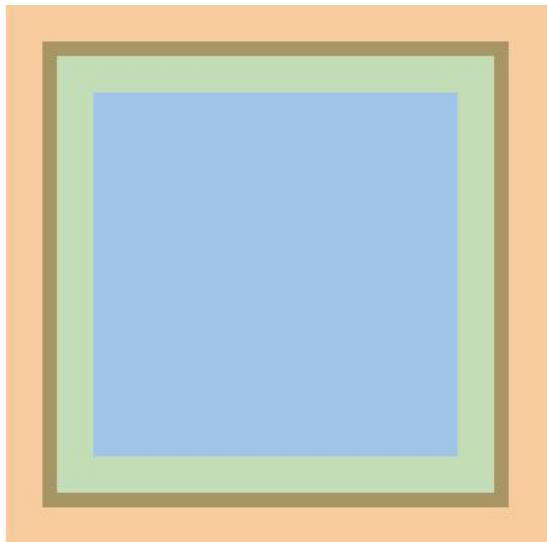
A magical box around every element!

Content = blue

Padding = green

Border = black

Margin = orange



Position

```
position: relative;  
top: 100px;  
left: 100px;
```

STATIC



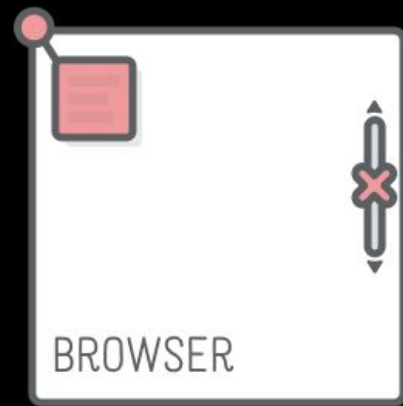
RELATIVE



ABSOLUTE



FIXED



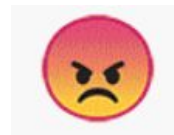
And also sticky (which is a cross between relative and fixed ... relative until you scroll to a point and then fixed)

<https://css-tricks.com/almanac/properties/p/position/>

Collapsing Margins

Margins are weird ... they should keep distance between objects ... but sometimes that distance isn't what is expected

Horizontal margins are fine ... but vertical margins create problems (with the larger margin winning out)



Responsive web design

Your website should look good everywhere (what happens on a cell phone)

```
.element {  
  responsive: yes;  
}
```

= Not a real thing ... sorry



There are different things you can do, like:

- Max and min height/width
- Percentages and rem/em
- Flexbox (probably Grid)
- Media queries

```
@media screen and (max-width: 600px) {  
  .element {  
    width: 100%;  
  }  
}
```

Flexbox

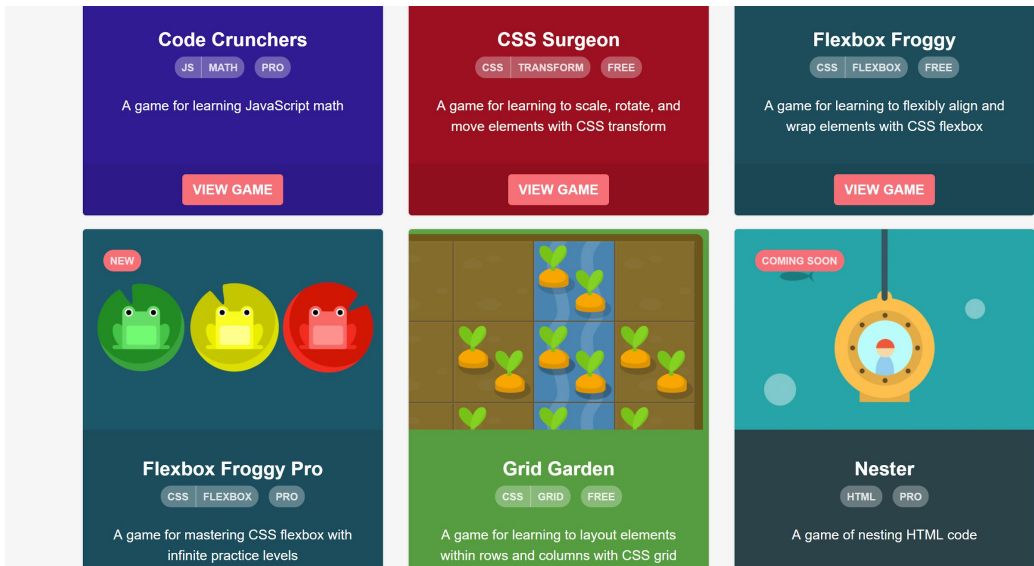
“aims at providing a more efficient way to lay out, align and distribute space among items in a container, even when their size is unknown and/or dynamic”

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

Super duper important stuff!

That I don't have time/enough experience to teach you!

- CSS flexbox
- CSS grid
- CSS animations and transitions
- <https://codepip.com/games/>



SASS, LESS, Bootstrap

SASS and LESS are stylesheet languages that get turned into CSS

- Have some fun things that make writing CSS a bit easier (e.g. import, nesting, variables, etc.)
- Looks like CSS for the most part ... just a bit more flexible

Bootstrap

- Frontend framework that provides html templates, stylesheets, and JS plugins
- Provides a consistent design experience across elements

CSS Activity

```
#container-div {  
  font-family: Arial, sans-serif;  
}
```

```
span {  
  background-color: #98FB98;  
  font-size: 3rem;  
}
```

```
img {
```

```
}
```

```
#vertical {  
  background-color: blue;  
  color: #FFFFFF;  
}
```

```
#horizontal {  
  /* color: white; ... a fun lesson in web accessibility and contrast */  
  background-color: yellow;  
}
```

```
.button_stuff {
```

```
}
```

Extra - Pseudo-class and Pseudo-element

Pseudo-class:

- Represents a special state of an element
- Uses one colon

```
#element-id:hover {}
```

Pseudo-element:

- Represents a part of an element to style
- Uses two colons

```
#element-id::first-letter {}
```


Extra - Inheritance

In some cases, if you don't set a value for a property, it will just inherit from its parent or an ancestor

```
<div class="test" style="color: red;">  
  <p>This is about inheritance ... I inherited red from a parent</p>  
  <div><p>I also inherited red ... from an ancestor</p></div>  
</div>
```

You can also set some property values to be “inherit”, if it doesn't already inherit, or if you need to override an existing rule

Extra - Color

```
.element-color: {  
  color: #FFFFFF;  
  
  color: white;  
  
  color: rgb(255, 255, 255)  
}
```

Extra - Size

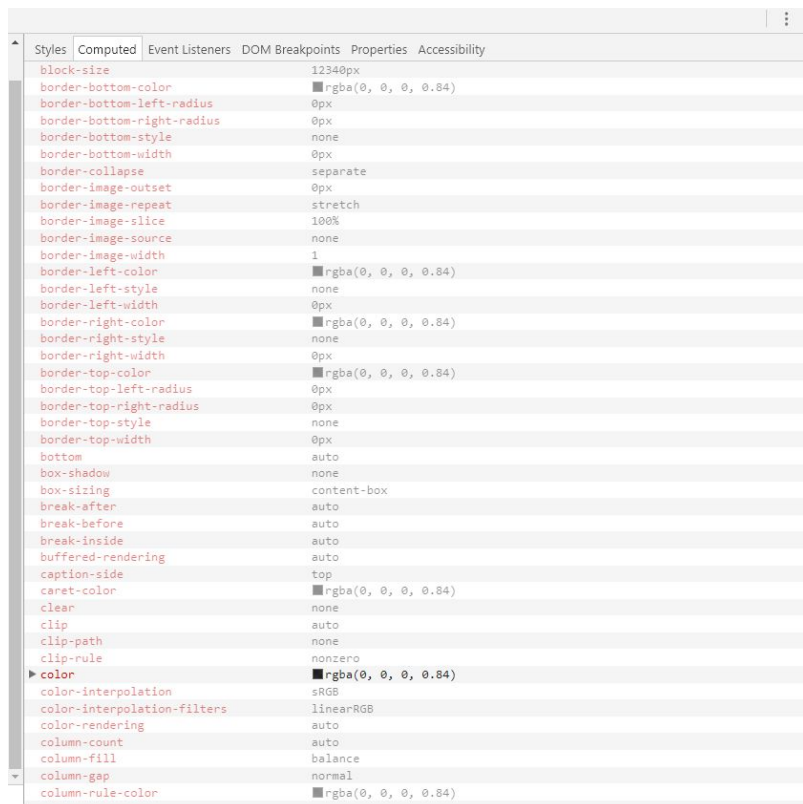
There are a number of units for size (only a few that matter):

```
height: 1px;  
height: 1rem; /* Font size of root element (<html>) ... usually 16px */  
height: 1em; /* Closest ancestor that has a set font size */  
height: 1vh; /* Percentage of the viewport height */  
width: 1vw; /* Percentage of viewport width */  
height: 1%; /* Percentage of parent element size for that property */
```

It's all pretty confusing ... generally use % and rem (px for smaller sizes is usually ok) ... use px if something always has to be a certain size

Also, the calc() function can come in handy for more “dynamic” sizing

Extra - Some other properties ... so many properties



Styles	Computed	Event Listeners	DOM Breakpoints	Properties	Accessibility
block-size				12340px	
border-bottom-color				■ rgba(0, 0, 0, 0.84)	
border-bottom-left-radius				0px	
border-bottom-right-radius				0px	
border-bottom-style				none	
border-bottom-width				0px	
border-collapse				separate	
border-image-outset				0px	
border-image-repeat				stretch	
border-image-slice				100%	
border-image-source				none	
border-image-width				1	
border-left-color				■ rgba(0, 0, 0, 0.84)	
border-left-style				none	
border-left-width				0px	
border-right-color				■ rgba(0, 0, 0, 0.84)	
border-right-style				none	
border-right-width				0px	
border-top-color				■ rgba(0, 0, 0, 0.84)	
border-top-left-radius				0px	
border-top-right-radius				0px	
border-top-style				none	
border-top-width				0px	
bottom				auto	
box-shadow				none	
box-sizing				content-box	
break-after				auto	
break-before				auto	
break-inside				auto	
buffered-rendering				auto	
caption-side				top	
caret-color				■ rgba(0, 0, 0, 0.84)	
clear				none	
clip				auto	
clip-path				none	
clip-rule				nonzero	
color				■ rgba(0, 0, 0, 0.84)	
color-interpolation				sRGB	
color-interpolation-filters				linearRGB	
color-rendering				auto	
column-count				auto	
column-fill				balance	
column-gap				normal	
column-rule-color				■ rgba(0, 0, 0, 0.84)	

Extra - Webkit and Moz

You might see:

-webkit-<insert property name>

Or

-moz-<insert property name>

These are vendor specific property versions that are meant to accommodate experimental properties before they become standard ... they're starting to go away

Don't worry too much about this :)

JS



How do I get it into my webpage?

```
<head>  
  <script type="text/javascript">  
    //Insert your javascript code here  
  </script>  
</head>
```

```
<head>  
  <script type="text/javascript" src="./demo.js"></script>  
</head>
```



- Minified code ... you'll know it if you see: <file-name>.min.js

What is Javascript?

It is a programming language that brings web pages to life!

All browsers have an engine that run it, and it has readily available commands that allow it to manipulate the DOM and other data

Originally only frontend, but now everywhere!!!

Scope

Where are variables accessible from:

- Global scope (outside of any functions) ... gets attached to global window object
- Function scope (inside of function) ... gets attached to that function
 - Functions declared inside of functions can always reach up to look for a variable, but can't reach down
- Block scope (inside of a block) ... gets attached to a block of code (e.g. if statement, for loop, etc.)

```
Global
var a //accessible at this level

Function 1 {
  //a access
  var b //only accessible function 1
}

Function 2 {
  //a access
  var c //accessible at function 2

  if() {
    let d //only accessible in if
    const e //only accessible in if
    var f //accessible at function 2
  }

  Function 3 {
    //a access
    //c access
    //f access
  }
}
```

Declaring a variable

```
var first = 'first' // function scoped and can be re-assigned  
  
const second = 'second' //block scoped and can't be re-assigned  
  
let third = 'third' //block scoped and can be re-assigned
```

- Block scoped = it exists within the set of curly braces where it was declared (could be a function)
- Function scoped = it's available anywhere in the function
- <https://www.freecodecamp.org/news/var-let-and-const-whats-the-difference/>
- SUMMARY: USE LET AND CONST

Dynamic vs. Static Typing

Static typing:

- We know the type at compile time
- int, bool, string, etc.

Dynamic typing:

- JS
- Type is figured out at run time
- var/let/const ... no need to tell it the type ... it will figure it out

Comments

```
// I AM A COMMENT!
```

```
/* I AM ALSO A COMMENT, AND CAN COVER MULTIPLE  
LINES */
```

Primitive Types

There are ~~6~~ 7 primitive types (immutable):

```
var boolean = true; //true or false
```

```
var string = 'hello'; //single or double quotes will work ... usually single
```

```
var number = 1; //all the number types
```

```
var nullVariable = null; //intentionally nothing
```

```
var undefinedVariable = undefined; // the initial state of a declared variable that hasn't been set
```

- Symbol (this is new-ish, and I don't think you need to worry about it for right now)
- BigInt (numbers larger/smaller than +/- 2^{53})

Undefined

If you declare a variable, and you don't assign anything to it, it is undefined

```
var first; //This is set as undefined
```

```
var second = undefined; //This is also a thing you can do
```

Don't do it!



Coercion ... this is where things get CRAZY!

Coercion is when, depending on the operator, one or more of your values will be type cast

Javascript is trying to help you, and is trying to anticipate what you want to do:

```
1 + 'hello' = '1hello'
```

Sometimes it can't really do anything (which is what you'd expect):

```
1 - 'hello' = NaN
```

Other times ... you get strangeness:

```
4 * true = 4
```

Coercion ... still CRAZY!

- 1) Watch out for operator precedence and associativity:
 - Precedence (e.g. multiplication comes before addition)
 - $4 + 3 * 2 = 10$
 - Associativity (addition will go left to right ... right to left for assignment)

```
4 + 3 + 'hello' = '7hello'
```

```
> a=b=2
```

```
< 2
```

```
> 2=b=a
```

✖ Uncaught ReferenceError: Invalid left-hand side in assignment

2) ALWAYS: `===/!==` (strict equality operator) instead of `==/!=` (equality operator)

Coercion ... even CRAZIER!

3) Watch out for turning things into numbers or Boolean (e.g. Boolean values or null or undefined)

```
Number(null)
```

```
0
```

```
Number(false)
```

```
0
```

```
Number(true)
```

```
1
```

```
Number(undefined)
```

```
NaN
```

```
Boolean(undefined)
```

```
false
```

```
Boolean(null)
```

```
false
```

```
Boolean(0)
```

```
false
```

Please go to the Chrome console and type the following:



```
isNaN(null)
```

Objects

Everything is an object (except for primitives)

Object literals are the easiest way to create an object (so do this):

```
var dog = {}
```

Objects can have properties:

- Properties: objects can have keys that store a value

```
var dog = {name: 'Darwin', breed: 'Beagle', age: 5, collar: true}
```

Objects

Objects can have methods (functions attached to an object):

```
var dog = {name: 'Darwin', dogBreed: function() {return 'Beagle';}}  
dog.dogBreed() //returns 'Beagle'
```

Objects are mutable:

- Read or set an object using '.' or with '[']'

```
dog.name = 'Huxley'  
dog['dogBreed'] = function() {return 'Bulldog';}
```

Arrays

A special type of object:

```
var animals = [];  
var animals = ['dog', 'cat', 'turtle']; //zero-indexed  
animals[2] //'turtle'
```

You can put anything in an array ... in any array:

```
var assorted = ['a', 1, true, null, {test: 'test'}, function() { console.log('test')}];  
assorted.pop(); //removes the function at the end  
assorted.push('turtle'); //adds turtle to the end
```

Lots of array specific properties/methods: length, sort(), indexOf(), splice(), etc.

Two important array functions

Array.map()

- Applies a function to each value of the array, and returns an array with modified values
- Takes a function that takes the current value, and does stuff to it
- Returns a new array with updated values

```
let even = [2, 4, 6, 8, 10];  
let odd = even.map(currValue => currValue - 1);  
console.log(odd); //[1, 3, 5, 7, 9]
```

Two important array functions

Array.reduce()

- Applies a function to each value of the array, and condenses all the values into one value
- Takes a function that takes an accumulator (total) value, as well as the current value, and does stuff to it
- Returns a single value

```
let numbers = [1, 2, 3, 4, 5];  
let total = numbers.reduce((accumulator, currValue) => accumulator + currValue);  
console.log(total); //15
```

Functions

```
function doSomething(parameter) {  
  console.log('I did ' + parameter + '!!!!!!');  
}  
var argument = 'something';  
doSomething(argument); //you just invoked the function ... congratulations!!!!
```

Functions in JS are first-class ... you can treat like any other variable

- Pass as argument into function (callback)
- Set a variable equal to the function (function expression)
- Set as an object property (method)

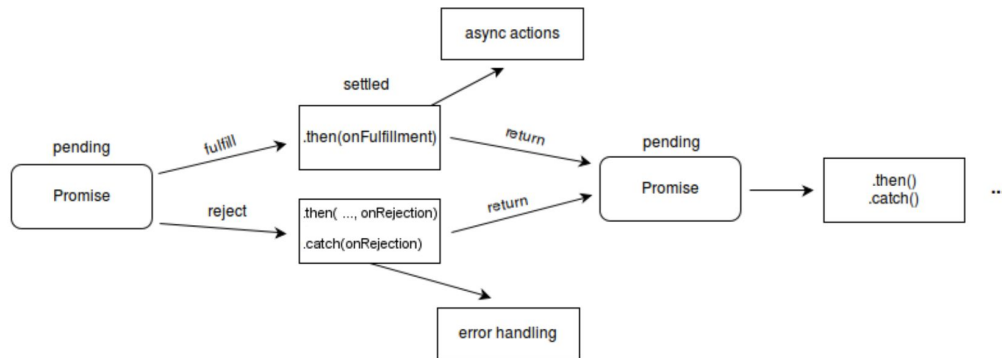
```
var doSomething =  
( ) => {console.log('Im doing something!')};
```

Functions

- Named function (function declaration):
 - `function name() { ... }`
- Anonymous function (function expression):
 - `function () { ... }`
- Arrow function (function expression ... but easier to write):
 - `() => {...}`

Promises

- JS is synchronous ... but a lot of what we do is asynchronous
 - Fetch api
- Promises provide a way to deal with asynchronous actions
- ASYNC/AWAIT (relatively new ... look them up!)
- <https://web.dev/promises/>



Promises (a conversation in a grocery store)

ME: Do you have flour? (**request**)

EMPLOYEE: I'll go check our inventory system. I promise I'll come back with an answer.
(**promise**)

ME: <does other stuff while waiting>

EMPLOYEE: The system says we have flour (**resolve**). I'll go to the back to get it, and I promise to come back. (**then** with a new **promise**)

ME: <does other stuff while waiting>

EMPLOYEE: Here's your bag of flour (**resolve**). Have a good day! (**then**, with a **resolution**)

ME: Thanks! (not part of the program ... but just nice to say :))

JSON

Javascript Object Notation

It looks like a JS object ... minor differences (keys need to be strings, functions/undefined/dates can't be used) ... within JS they work similarly

But it's easy to read, and cheaper to send across the internet than XML (which is kind of gross)

Serialization: `JSON.stringify()` ... turn an object into a JSON string

- Not everything gets converted to a JSON string ... no functions, no undefined

Deserialization: `JSON.parse()` ... turn an object into a JS object

JS and the DOM

There are a number of methods that allow you to manipulate the DOM (add elements, remove them, update attributes, etc.):

```
var testElement = document.getElementById(); //gives you a reference to a DOM element based on the ID
testElement.setAttribute('class', 'red_class');
```

Interacting with the DOM also creates events (e.g. click event, move event, submit event) that you can listen for and launch JS with (e.g. onclick, onsubmit, etc.):

```
<button id="vertical" class="button_stuff" onclick="moveVertical();">
Vertical Cat
</button>
```

JS Activity

```
function setSize() {
    var size = 0;
    var randomNum = Math.floor(Math.random()*10) ;
    for (var i = 0; i < randomNum; i++) {
        size += 100;
    }
    return size;
}

function moveVertical() {
    var catPicture = document.getElementById('picture');
    if (catPicture) {
        catPicture.style.top = setSize() + 'px';
    }
}

function moveHorizontal() {
    var catPicture = document.getElementById('picture');
    if (catPicture) {
        catPicture.style.left = setSize() + 'px';
    }
}
```

Extra - NaN

This frightening symbol means “Not a Number”

There's also a function: `isNaN()` ... which will check if something is a number

It can behave strangely (you'll see in a bit)

Extra - Operators ... so many operators

https://www.w3schools.com/jsref/jsref_operators.asp

Extra - If Statements

```
if (/*condition*/) {  
    //do stuff  
} else if (/* other condition*/) {  
    // do other stuff  
} else {  
    //do other stuff  
}
```

Fun fact: This is a great place to take advantage of coercion ...

- 0
- NaN
- null
- undefined
- ""

... all coerce to false ... which makes it really easy to check if something exists before you do things to it

Extra - Loops

for loop (good for arrays):

```
for(var i = 0; i < 5; i++) {  
    //do stuff with i  
}
```

for-in loop (good for objects ... not good for arrays):

```
for(var property in object) {  
    //do stuff with property  
}
```

while loop:

```
while (/*condition*/) {  
    //do stuff  
}
```

do/while loop:

```
do {  
    //do stuff  
} while (/*condition/);
```

Extra - Useful functions (SO MANY FUNCTIONS!!!)

`document.getElementById()`

`document.querySelector()`

`element.setAttribute()`

`element.getAttribute()`

`array.map()`

`array.splice()`

`Math`

There are a lot of functions that are available for different objects and types

TOO many to put here!

Look them up as you go :)

THE END :)

D2L

<https://www.d2l.com/>

Things to think about:

- Don't worry ... you're learning how to talk to machines ... that's hard
- Lots of different roles are possible
- What are your strengths? What are your interests? ... Try different things!
- Manage your manager
- Own your mistakes
- Don't ask for help ... but definitely ask for help
- Look at how to make the team better
- Say yes more
- Take on the harder tasks
- Keep an eye out for opportunity ... consider being an intrapreneur
- STAY HEALTHY!!! (Physical, mental, emotional)