**Class Notes**

**TO DO**

* Review notes from last 3 classes
* Finish rock, paper, scissor coding
* Coding car game to not work until tune up is complete
  + this.mileage vs. car.mileage
* Color Corrector Game
* Research objects, how they’re used, what they do, etc.

**Resources to Review**

* Git review
  + <https://www.youtube.com/watch?v=Y9XZQO1n_7c&feature=youtu.be>
* JavaScript review
  + <https://spin.atomicobject.com/2014/10/20/javascript-scope-closures/>
  + <https://www.rithmschool.com/courses/javascript>

**Questions**

<https://diegopiovesan.files.wordpress.com/2010/07/livro_-_the_non-designers_desi.pdf>

**Class 20: Wednesday January 16**

**Class 19: Monday January 14**

Must include the following 4 terms in my config to sync my database with my page

apiKey: "AIzaSyAOHfYwAcZJ2fRBRHqbdBUSU--GyF8aNCw",

authDomain: "fir-demo-a5be8.firebaseapp.com",

databaseURL: "https://fir-demo-a5be8.firebaseio.com",

storageBucket: "fir-demo-a5be8.appspot.com",

**Class 18: Saturday January 12, 2019**

* Data persistence = state, where data is being stored
  + Cookies, local storage, session storage
* Array Methods:
  + .slice()
  + .splice()
  + .substring()
  + .split()
  + .jsonstringify()
  + .parse()
* How to assign breakpoints, evaluate values mid-execution

Three types of data storage

* Cookies
  + older way of data persistence
  + store up to 4K worth of data
  + only one of the 3 that allows us to interface with the server
  + always dealing with entire content of cookies as one string, not able to parse out individual elements
* Session Storage
  + Expires once you close the browser
  + 1:38:00
* Local Storage
  + Data persists between sessions (loading of tab/browser)

**Class 17: Wednesday January 9, 2019**

**Class 16: Monday January 7**

API Recap

Check the slides provided in 6.2, watch the HueCraft video

JSON

Difference between JSON and JavaScript Objects: keys are wrapped in quotes

**Class 13: Saturday December 15 – Review**

* Ways to check code: debugger, breakpoints, console.log() // better than alerts
* Don’t use IDs in CSS (whenever possible)

**Class 12: Wednesday December 12**

Objects: “key-value pairs”; it has a property and a value; declared with var objectName = {key/property, “value”};

An object is a variable that contains multiple values

e.g. var champ = warriors [creates a variable called “champ”, the value of which is “warriors”]

e.g. var champ = {team: warriors, MVP: Durant, coach: Kerr}

this creates an OBJECT called “champ”, which has multiple values (warriors, Durant, Kerr). Those values can be specifically called using the keys (team, MVP, coach)

Objects can have methods – these methods are actions that can be performed on the object (and only on the object). These methods are stored as functions in the key/value of the object.

**Class 11: Monday December 10 – JS and jQuery Jubilee**

<https://spin.atomicobject.com/2014/10/20/javascript-scope-closures/>

<https://www.rithmschool.com/courses/javascript>

**Class 10: Saturday December 8 – Jquery $**

DOM

Document Object Model 🡪 objects are things like elements

**Class 9: Wednesday December 5 – JavaScript Juggernauts**

JavaScript Resource

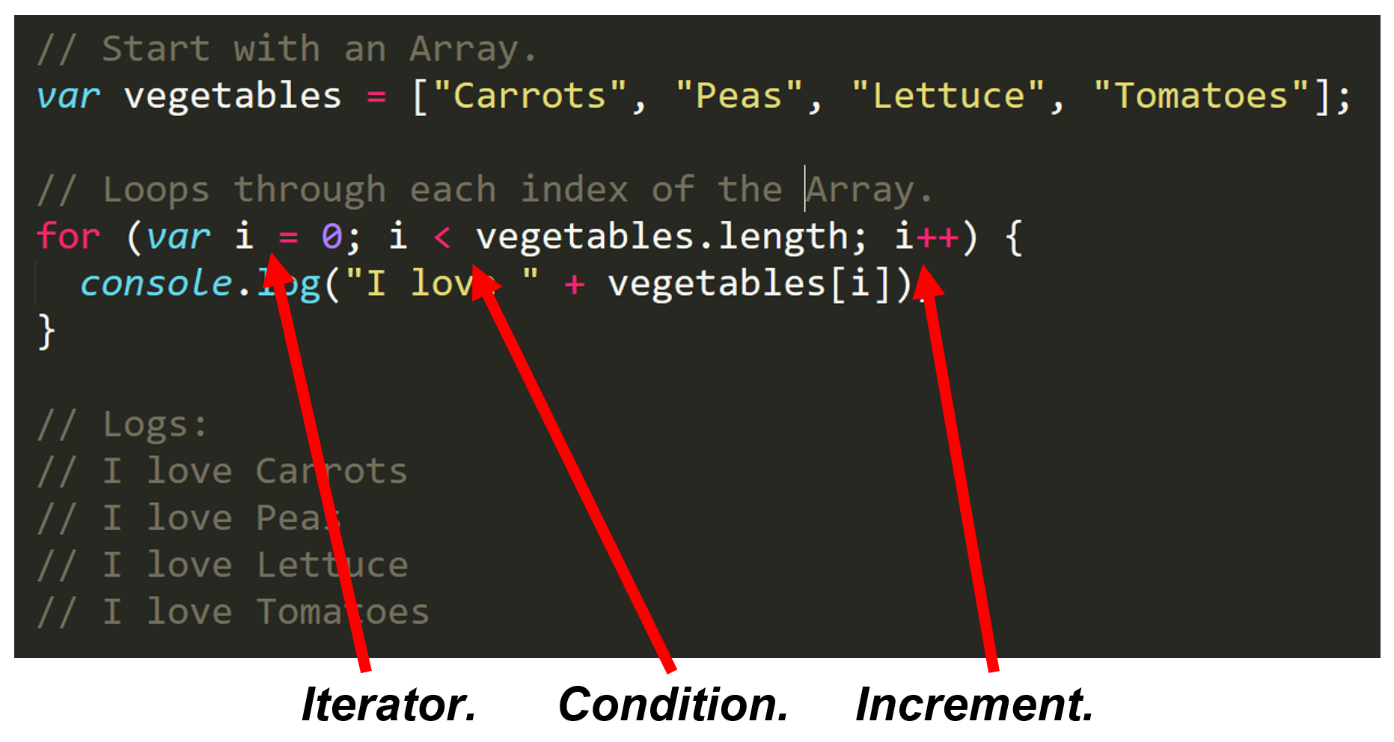
<https://javascript-workbook.netlify.com/>

**Class 8: Monday December 3 – Jumping for JS**

Basic Variables

* console.log(
  + logs data to the console (discreet, user doesn’t see unless they look in console)
* alert(
  + displays pop-up message to user
* confirm(
  + displays true/false (Boolean) popup
* prompt(
  + pop-up that requires text entry from user

For Loops

It can be tedious, inefficient, time-consuming to retype code for each index of an array. We can use For Loops to type code once and have it cycle through an entire array.

**Rock Paper Scissors**

Array of variables [rock, paper, scissors]

1. Player A picks r/p/s
2. Player B picks r/p/s
3. Compare Player A vs. Player B
4. Choose a winner
   1. rock vs. scissors: rock wins
   2. scissors vs. paper: scissors wins
   3. paper vs. rock: paper wins

**Class 7: Saturday December 1 – JavaScript**

* HTML: markup language; FOR: search engines
* CSS: presentation language; FOR: user
* JavaScript: programming language FOR interactivity, dynamism
  + Output changes depending on the input

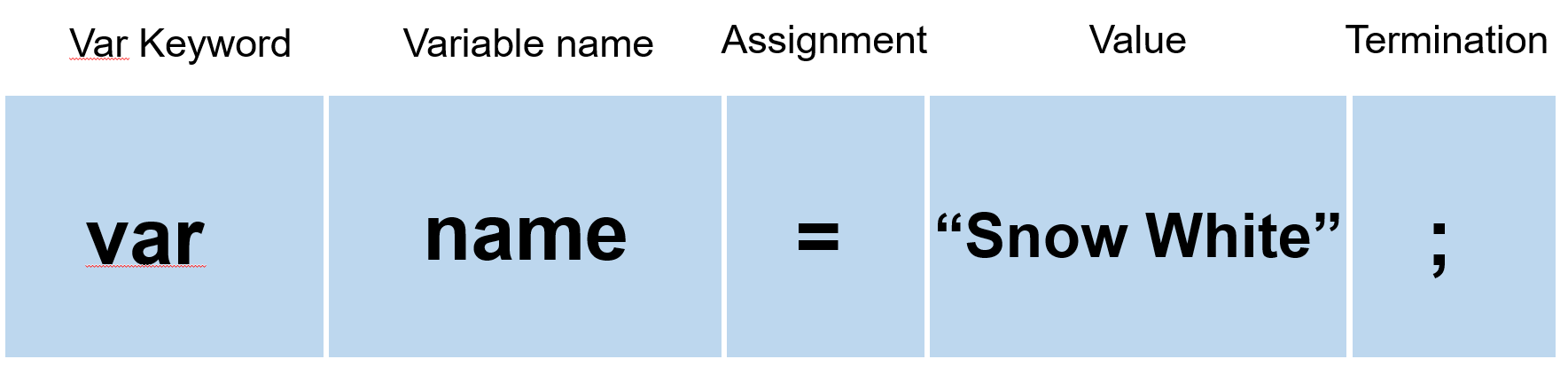
Interpreted vs. compiled languages

* Compiled languages 🡪 executables

JavaScript: It was initially written/developed in 10 days

Variables

* A way to store data in memory
* coded as “var”
* composed of: variable names and values
* Types of variables:
  + Numbers
  + “strings” (of characters)
  + Boolean (true/false)
* Syntax



Where to place JavaScript in the document?

* If the script is manipulating objects in the DOM, better to place script at the end of the body so that all elements of the DOM are rendered before script runs

Arrays: a list of values that falls into a category (array = the name of the parent category)

CSS Specificity

Weights for conflicting styles:

* 1 – element selectors
* 10 – classes
* 100 – ids
* 1000 – inline styles
* ∞ - !important (hack/kluge)

**Class 6: Wednesday November 28th**

Media Queries

**Class 5: Monday November 26th**

Resources for Future Use

* Jon Duckett books
* Eloquent JavaScript
* Code School

Typography

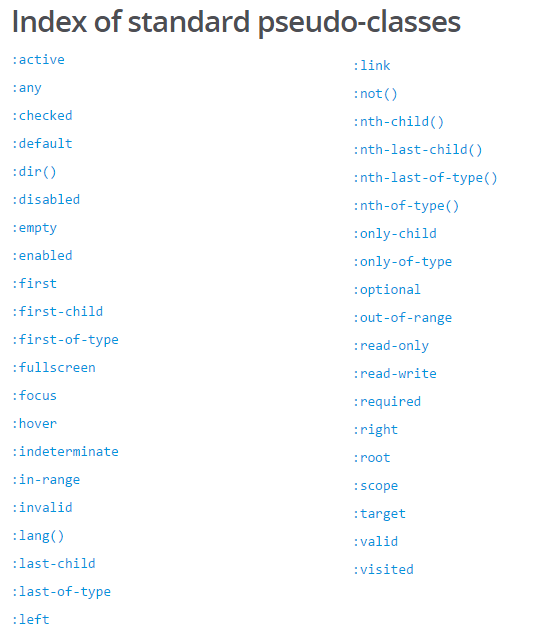
Properties

* Line height
* font size
  + px
  + em (relative to parent container)
  + rem (relative to document size)
  + vwh (responsive to browser size)
* line length
* letter spacing
* sans serif vs. serif

Google Fonts

* load times are an important consideration for UX
* Chrome extension: whatfont

Pseudo Classes



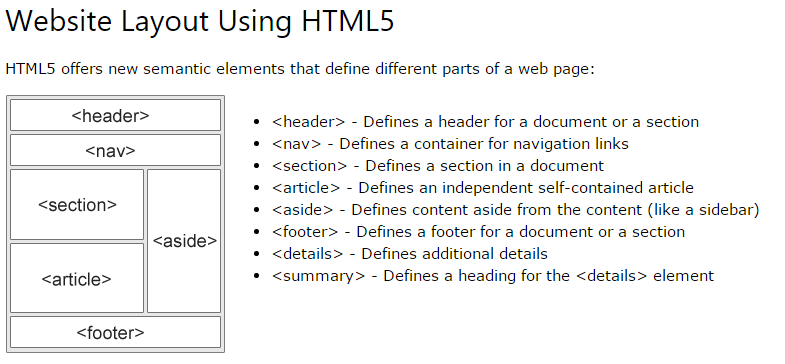
* hover
  + use transition property to make changes more visible (see the change in hover state as opposed to jumping immediately)
* active: element engaged (e.g. being clicked on, typed in, etc.)
* focus: doesn’t need to be active, tabbing through elements one way to stimulate focus without being active

Bootstrap

**Class 4: Monday November 19th**

HTML semantics

* Using specific tags makes it easy for machines to digest the importance of the content from the human perspective



* Difference between classes & IDs
  + IDs are one-time use (and can be used to link directly to the ID in the page)
  + Classes are used for multiple HTML elements

Multiple CSS files

* Browser will read multiple files in chronological order as one giant .css
* loading order MATTERS, CSS reads top-to-bottom, don’t want to conflict and end up with undesirable result

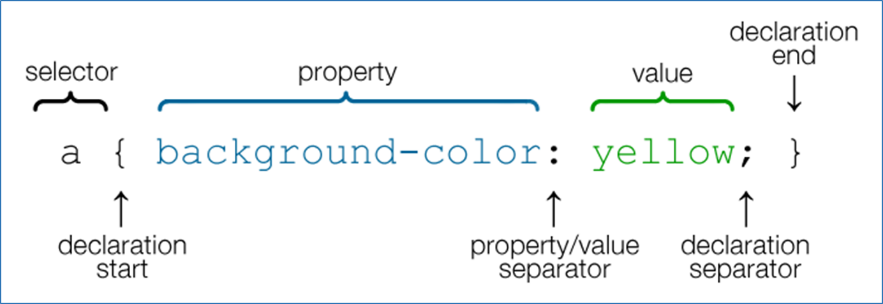
CSS Resets

* What is it? A file that returns every CSS style to “0”, removes all formatting
* Why use one? It is a tool for cross-browser functionality
  + *guessing we reset to zero BEFORE adding our styles to remove any browser-specific formatting?*

Deploying Sites

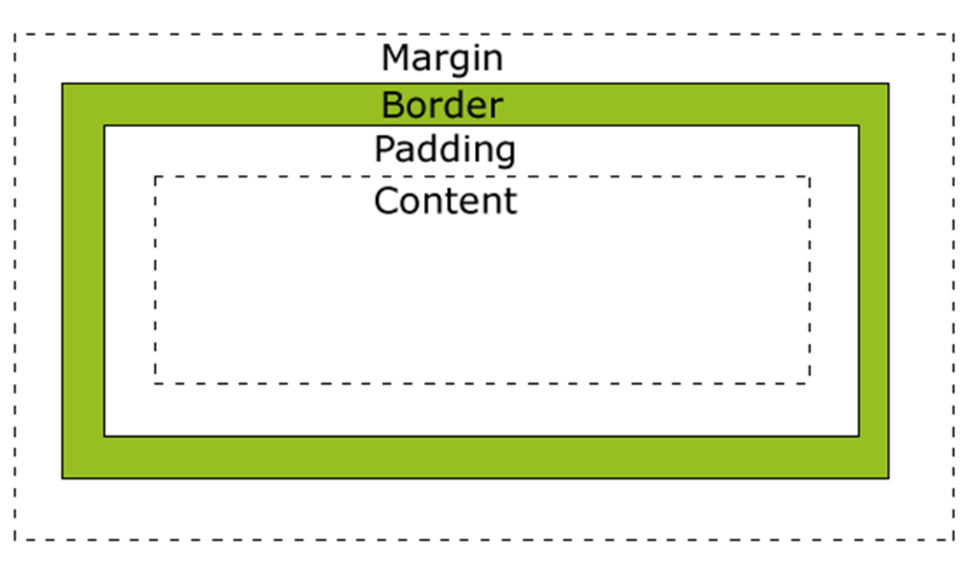
**Class 3: Saturday November 17th**

How to style HTML? CSS of course!

* Elements (called using element name: p { applies to all <p> tags )
* Classes (called using a period: .container { )
* IDs (called using a hashtag: #id { )

Relative vs. Absolute file paths

* Relative paths direct the browser to files within the working directory
* Absolute file paths direct the browser to a specific file
  + This is okay for files hosted online (e.g. URLs to images), but do not use absolute file paths to link to files/images on my machine



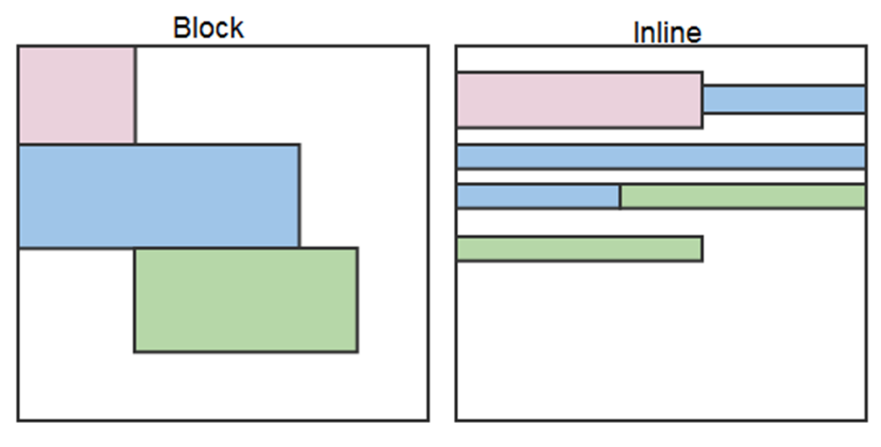
Box Model

* Elements in CSS work using a “box model” with variables like margin, padding and border
* box-sizing: makes it easier to make a box model the size you actually want it to be
  + e.g. make my box 400px x 400px, and factor in the margins/padding/borders from there to end with a 400px x 400px
  + benefits of this: if I need to create a page with specific-sized boxes, I can use box-sizing instead of having to calculate heights/widths including padding/margins/borders
  + <https://www.paulirish.com/2012/box-sizing-border-box-ftw/>

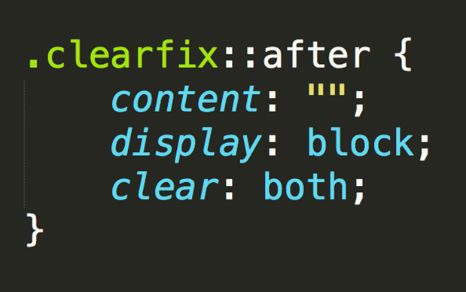
Floats

* Floats are a great tool to create “flow” on your webpage
* Functionally like text wrapping, turns block elements into inline elements (see below)

Block elements vs. inline elements

* most elements are Block Elements, which take up an entire line of space (unless otherwise specified)
* inline elements flow one after another

Clearing the Float

* using floats can mess with the layout in ways we don’t want, especially re: the height of our containers
* To fix this, we can use the clearfix “hack”:

[All About Floats by Chris Coyer, CSS-Tricks](https://css-tricks.com/all-about-floats/)

* Four float values:
  + left (float left)
  + right (float right)
  + none (element doesn’t float)
  + inherit (assumes float value of parent element)
* clear is the sister property of float, also has four values
  + both (clears floats coming from both directions)
  + left (clears floats from the left)
  + right (clears floats from the right)
  + none (doesn’t clear at all)

CSS positioning

Position types

* Static (default)
* Relative (positioned relative to other elements in the container, NOT taken out of the flow of the page)
* Absolute (taken out of the flow, specific positions relative to nearest positioned ancestor)
* Fixed (positioned on specific coordinates in the browser window)

Layering

* z-index: use large gaps to give yourself room to add elements in between
  + e.g. use 100 and 200 instead of 1 and 2, if I need to add something else, can use 150 instead of having to rename every single layer

Display

* Display: none will hide elements

Expand on overflow

Flex Box

<http://jonibologna.com/content/images/flexboxsheet.pdf>

<https://www.youtube.com/watch?v=jV8B24rSN5o&feature=youtu.be>

<https://www.youtube.com/watch?v=jV8B24rSN5o&feature=youtu.be>

**Class 2: Wednesday November 14th**

Working with GitHub

What is GitHub?

* A web-based platform that stores code online
* Serves as a backup to local machines
* Useful tool for collaborative coding
* Users pull (download) or push (upload) code to/from a GitHub repository
* Major benefit of working through GitHub: version control

Version Control

What is it?

* Developing code in installments (think of it like saving a document in multiple versions as you work on it) 🡪 Version 1, Version 2, etc.

Why is it helpful?

* Version control makes it a lot easier to work collaboratively
  + Easy to track who has worked on what
  + Easy to ensure you’re working with the latest code
* Segmenting your code makes it clearer which portions of code are creating problems
  + E.g. if Version 3 worked fine but Version 4 has problems, the coding errors reside in what was added in Version 4

Basic Git commands

* git clone = download a repository from GitHub to your local directory
* git add = first step, prepares a file to be committed to the GitHub repository
* git commit = commits a file to GitHub repository (use “-m” to write a message annotating the change)
* git push = sends changes to the online GitHub version
* git pull = downloads the latest version of a GitHub repository

HTML

* Headings factor into SEO
* HTML tags resource: <http://www.w3schools.com/tags/>

CSS

* when modifying elements using CSS, use a period for classes and a hashtag for IDs
  + e.g. if a class is called ‘container’, we’d refer to it as .container in CSS
  + e.g. if an ID is called ‘main\_bio’, we’d refer to it as #main\_bio in CSS

**Class 1: Monday November 12th**

# Bash / Terminal Commands

These are the basic commands for navigating directories in a Terminal or Git Bash window

*Moving in directories*

* Change Directory = cd
* Change to Home Directory = cd ~/
* Move to One Directory Up = cd ..
* View Folders and Files in the Directory = ls
* Show the current Directory = pwd
* Autocomplete a File Name in the Current Directory = Press `tab` key once to autocomplete once you have typed a unique portion of a file name

*File Manipulations*

* Make New File = touch [name of file to create]
* Make New Folder = mkdir [name of directory to create]
* Delete file = rm [name of file to remove]
* Delete folder = rm -r [name of directory to remove]
* Copying File = cp [filename1] [filename2]
* Move/Rename File = mv [filename1] [filename2]
* Open file or folder (PC only) = explorer [name of file]
* Open all files and folder in current directory (PC Only) = explorer .