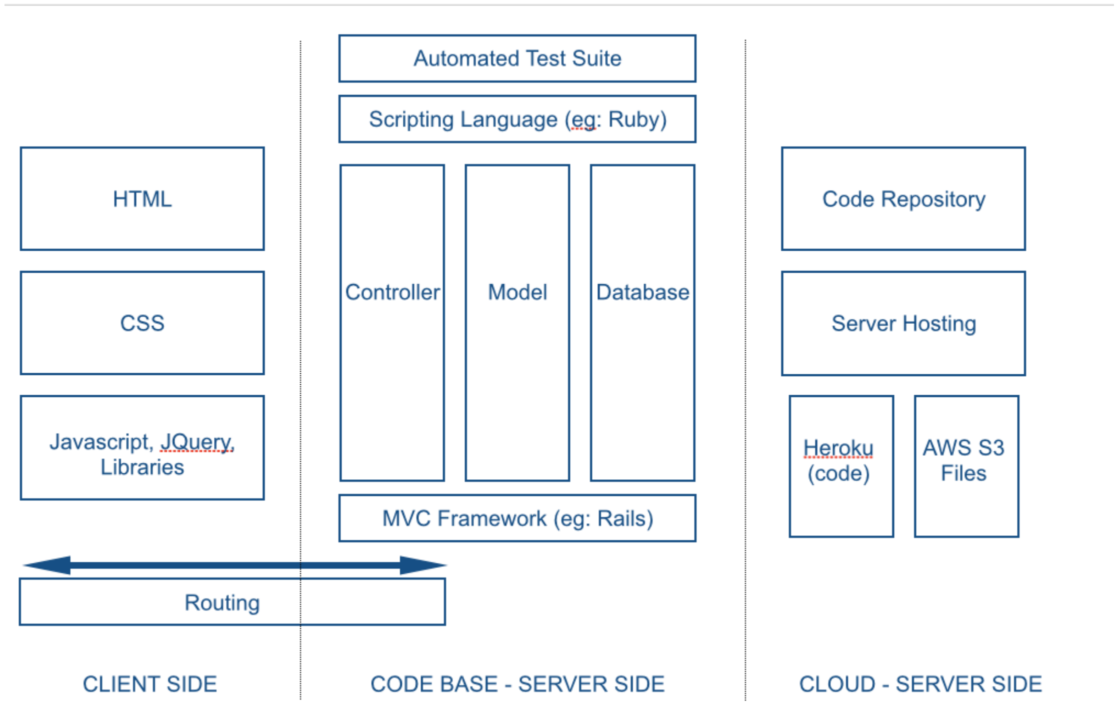
# CODING FOR PRODUCT MANAGERS

## INTRODUCTION

Web development brings together a varied of languages, frameworks, and technologies, each of which is a system in and of its own. When combined their complexity is compounded and the process can seem impenetrable.

In this course, we're going to break down each component from the outside in, building a blog app (https://product-school-blog.herokuapp.com/) that brings them all together in a functioning, live system. In the process, you'll get an understanding of both the parts and the whole. By the end of this course, our goal is that you'll be able to build on this base of learning to create more complex apps as well as collaborate with developers as a product manager to build products that far exceed what you could have created without an understanding of the development process (not to mention building them in a more frictionless, time-efficient manner!).

We'll tackle three core pillars of the web development process: front-end web development, backend web development, and cloud-side code deployment and hosting. Here's a snapshot of what we'll be learning:



In our front-end module, we'll start with HTML, CSS, and mobile-optimized responsive styling.

We'll then dive into Ruby, your first backend scripting language. We'll cover the basics of the language and how you can use it to build functionality into your system. Once we have a sense of Ruby, we'll dive into Rails, a Ruby-based Model-View-Controller framework that will allow you to quickly build out your blog app. In the process we'll also get familiar with Git, Github, versioning, and databases – which are a constant in any good web development process. We'll round out our backend module by diving into automated testing as a way of building reliability and efficiency as your codebase scales.

Finally, you'll learn how to push your application onto Heroku and integrate with Amazon's AWS S3 service to dynamically serve images that are uploaded by users of your app.

If we happen to have time, we can dive back into JavaScript, which is a client-side scripting language, and JQuery a JavaScript library, which will allow to you create interactivity in and additional functionality into your web page.

You can access all of the code for this course on the Product School github account at: https://github.com/product-school/ .

TIMELINE (tentative):

CLASS 1 – HTML, CSS & Responsive Web Styling

CLASS 2 – Git, Github & Ruby

CLASS 3 – Introduction to the Rails MVC framework – Database Structure, Models, Views, and Controllers

CLASS 4 – Rails Continued

CLASS 5 – Automated Testing

CLASS 6 – Hosting on Heroku & Amazon AWS S3. Overview of Javascript & JQuery and other relevant languages/frameworks.

## Unit 1 - HTML, CSS, & Responsive Styling

### 1.1 HTML

Introduction:

HTML (Hyper-text Markup Language) and CSS (Cascading Style Sheets) represent the skeleton and outer-facade of any page that you see on the Internet. Any frontend or full-stack developer will be working with these daily. The basic concepts within HTML and CSS can be learned quickly and you'll have time to deepen your knowledge of specific attributes, tricks, and resources over time.

Pre-Work:

- Download Sublime Text (http://www.sublimetext.com/2). You'll use this as your code editor for the rest of the program.
- Download Chrome (if you don't yet have it downloaded).
- Install Xcode on your laptop (via the app store). Open up Xcode, in the Xcode dropdown (upper left hand side of the screen) go to Preferences and click on the Download tab. Check "Command Line Tools" and then click the "Check and Install Now" button at the bottom left of the window. This will take a while, so feel free to let the download run in the background. Note: If you don't see the 'Command Line Tools' option, open the Utilities folder within Applications and then open the Terminal application. Then enter the command xcode-select –install . If still causes an error, hold off for help in class.
- Create a ProductSchool folder on your computer where you'll house all of your code
- Check out the blog you'll be building: https://product-school-blog.herokuapp.com/

In-Class:

i. HTML page structure, head & body

```
<! DOCTYPE Html>
<html>
        <head>
        <title>Page Title</title>
        </head>
    <body>
            [Page's HTML content]
    </body>
</html>
```

ii. HTML body tags (full list: http://www.w3schools.com/tags/)

Most Commonly Used Tags:

- Full Width HTML Tags
  - Headings: <h1> - <h6>

    ```
    <h1>Hello World</h1>
    ```
  - Paragraph:

    ```
    <p>Copy text <p>
    ```
  - Div (aka 'box'):

    ```
    <div> [content to be placed inside] </div>
    ```
  - Ordered (numbered) & Unordered (bulleted) Lists: <ul> & <ol>

    ```
       <ul>
    ```

```
        <li>Point one</li>
        <li>Point two</li>
        <li>Point three</li>
      </ul>
```

- Blockquote:

```
      <blockquote>Copy worth quoting</blockquote>
```

- Horizontal Rule (aka: horizontal divider line): <hr>

- Line break: <br>

- Nav: <nav> </nav>

```
      <nav>
           <a href="/html/">HTML</a> |
           <a href="/css/">CSS</a> |
           <a href="/js/">JavaScript</a> |
      </nav>
```

- Footer:

```
   <footer>
      <ul>
               <li><a href="/about/">About Us</a></li>
               <li><a href="/privacy/">Privacy</a></li>
               <li><a href="/contact/">Contact Us</a></li>
      </ul>
   </footer>
```

- Form:

<form>[1]

```
       <div>
         <label for="post_title">Title</label>
         <input required="required" type="text">
       </div>

       <div>
         <label for="post_image">Image</label>
         <input required="required" type="file">
       </div>

       <div>
         <label for="post_show_image">Show Photo?</label>
      <input type="checkbox" name="post_show_photo" value="1">
       </div>

       <div>
          <label for="post_category">Category</label>
          <select>
            <option value="1">Product School</option>
            <option value="2">Travel</option>
       </div>

       <div class="actions">
         <label for="post_submit">Submit</label>
         <input type="submit" value="Create Post">
       </div>
   </form>
```

Postgres1 Form code accessible @ https://github.com/product-school/html-css-exercises/blob/master/form.html

<u>Common form text field types:</u> text, email, password, search, URL

<u>Form field types without text field 'input':</u> checkbox, radio, number range

<u>Form buttons:</u> button, image, reset, submit

iii. In-Line HTML Tags

- Image: <img src="linked-file-path" alt="alternative title">
- Anchor (aka: 'link'): <a href="/link-url">Linked copy</a>

iv. Formatting HTML Tags

- Bold: <b>bolded text</b> ; alternate: <strong>bolded text</strong>
- Underline: <u>underlined text</u>
- Italics: <i>Italicized text<i>

1.1.2 In-line Styling

- <tag style="attribute: value;">
- Starter styles: width (px or %), height (px or %), text-align (left, center, right), color (hex), background-color, border (px color solid/dashed)

In-Class Exercises:

- Create the HTML structure for your blog index page[2]

1.1.3 Boxes: Positioning, Display Types & Floats

In-Class Reading:

- Read: http://learnlayout.com/position.html
- Read up on floats: http://css.maxdesign.com.au/floatutorial/introduction.htm

i. Box Positions

- Static:
  - Default, 'unstyled' position
- Relative:
  - Behaves the same as static unless you add some extra properties.
  - Setting right, left, top, bottom, will adjust the position of a relative box.
  - Other elements will not fill open gaps (eg: if the width of a relatively positioned box is only 40% of the line, the other 60% of the line will be empty).
- Fixed:
  - A fixed position element is positioned relative to the viewport and will stay in the same place in the browser as you scroll.
  - Does not leave a gap in the page where it would have otherwise been placed.
- Absolute
  - Behaves like a fixed position element, but is fixed relative to it's nearest parent rather than the viewport

ii. Box Display Types:

- <u>Inline Element</u>: has no line break before or after it, and it tolerates HTML elements next to it.
  - Respect left & right margins and padding, but **not** top & bottom
  - **Cannot** have a width and height set

[2] HTML + CSS rendering of the page for reference: http://product-school.github.io/blog-html-css/index.html

  - Allows other elements to sit to their left and right.

- Block Element: Owns the full width of the page
  - Respect top & bottom margins and padding
  - Respects height and width
  - Forces a line break after the block element

- Inline-block Element: is placed as an inline element (on the same line as adjacent content), but it behaves as a block element.
  - Allows other elements to sit to their left and right
  - Respects top & bottom margins and padding
  - Respects height and width

iii. Floats

- Floated boxes will move to the left or right until their outer edge touches the containing block edge or the outer edge of another float.

- If there isn't enough horizontal room on the current line for the floated box, it will move downward, line-by-line, until a line has room for it.

- Block level elements above a floated element will not be affected by it. However, elements below will wrap around the floated element.

In-Class Exercises[3]:

- Add CSS to the html scaffolding of this blank page: https://github.com/product-school/html-css-exercises/blob/master/blank_face.html into a face[4]: http://product-school.github.io/html-css-exercises/face.html

1.2 CSS

Resources:
- For a full-list of css properties: http://www.w3.org/TR/CSS21/propidx.html .
- For more information on CSS selectors: https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_started/Selectors

i. File Structure & Syntax

- Linking your css file to your html file

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

- Syntax

```
h1 {
     property: value;
     color: red;
     font-size: 16px;
     text-align: center;
 }
```

ii. CSS Selector Types

- HTML Tags:

[3] Here's a list of common style properties to get you started: http://tech.journalism.cuny.edu/documentation/css-cheat-sheet/
[4] Feel free to use this triangle generator: http://apps.eky.hk/css-triangle-generator/

```
h1, p, div{
    color: red;
}
```

- Classes & Id's: HTML tag attributes used for CSS & JavaScript targeting

    - Classes: Assigned to one or more HTML elements on a page.

```
.my-class {
     text-align: center;
     margin: 0 auto;
}
```

- IDs: Assigned to only ONE HTML element on a page

```
#my-Id {
     color: green;
}
```

- Pseudo Selectors: styling that is assigned to a state of a page element or specific subset of elements

    - a:hover { color: red;} : Links will turn red when a mouse hovers over them
    - p:first-child { background-color: gray;}: The first paragraph element on a page will have a gray background color
    - div:nth-child(3) { text-align: center; }: Text in the 3rd div on the page will be centered

- Selectors based off of relationships

    - A E {}: Any E element that is a *descendant* of an A element (that is: a child, or a child of a child, *etc.*)
    - A > E {}: Any E element that is a child of an A element
    - E:first-child {}: Any E element that is the first child of its parent
    - B + E {}: Any E element that is the next *sibling* of a B element (that is: the next child of the same parent)

iii. Chrome developer tools

- Elements + CSS
- Console

## 1.3 Responsive Styling

Introduction:

Responsive styling enables you to customize how a page is displayed at different browser sizes. With media queries, you can set custom CSS for any browser with you'd like to target.

i. Typical Viewport Sizes

- Smartphones: 680 x 960 pixels
- Tablets: 768 x 1024 pixels
- Laptops: 1440 x 900 pixels
- Desktops: 1920 x 1080 pixels

ii. Responsive media queries

```
@media (max-width: 767px) and (min-width: 480px) {
     h2 {
          font-size: 12px;
```

```
        }
        nav {
                display: none;
        }
    }
```

iii. Head Meta-tags:

Add this to the head of your file to let your mobile browser know your site is optimized for mobile: <meta name="viewport" content="width=device-width, initial-scale=1">

Homework:

i. Finishing Up The HTML + CSS For Your Blog:

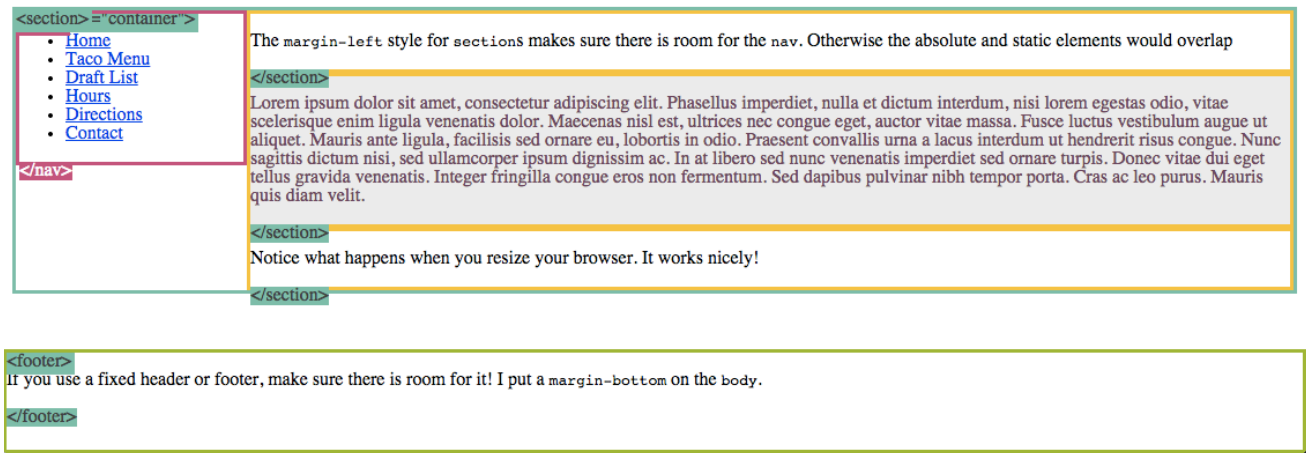
- Review the most common CSS style properties: http://tech.journalism.cuny.edu/documentation/css-cheat-sheet/
- Finish the HTML for the index and individual blog page of your blog.
- Add responsive CSS styling to your Product School blog index and post page. The end result should look something like this: http://product-school.github.io/blog-html-css-responsive/index.html . Here is a link to a final code version on Github, if your absolutely need it! https://github.com/product-school/blog-html-css

i. Prepping For Next Class

- Git & Github: Do all of the steps included in the Unit 2 pre-work
- Ruby:
  - Do CodeSchool's 15 minute Intro to Ruby exercise: http://tryruby.org/levels/1/challenges/0
  - Go the full way through this intro to Ruby tutorial: https://www.ruby-lang.org/en/documentation/quickstart/

Bonus Exercise (if you're bored and want to continue honing your skills)

- Recreate the image below using box positioning and inline styling: Live version here: http://product-school.github.io/html-css-exercises/box-positioning.html



<section>="container">

- Home
- Taco Menu
- Draft List
- Hours
- Directions
- Contact

</nav>

The `margin-left` style for `section`s makes sure there is room for the `nav`. Otherwise the absolute and static elements would overlap

</section>

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac. In at libero sed nunc venenatis imperdiet sed ornare turpis. Donec vitae dui eget tellus gravida venenatis. Integer fringilla congue eros non fermentum. Sed dapibus pulvinar nibh tempor porta. Cras ac leo purus. Mauris quis diam velit.

</section>

Notice what happens when you resize your browser. It works nicely!

</section>

<footer>

If you use a fixed header or footer, make sure there is room for it! I put a `margin-bottom` on the `body`.

</footer>

## UNIT 2 – GIT & GITHUB

Pre-work:

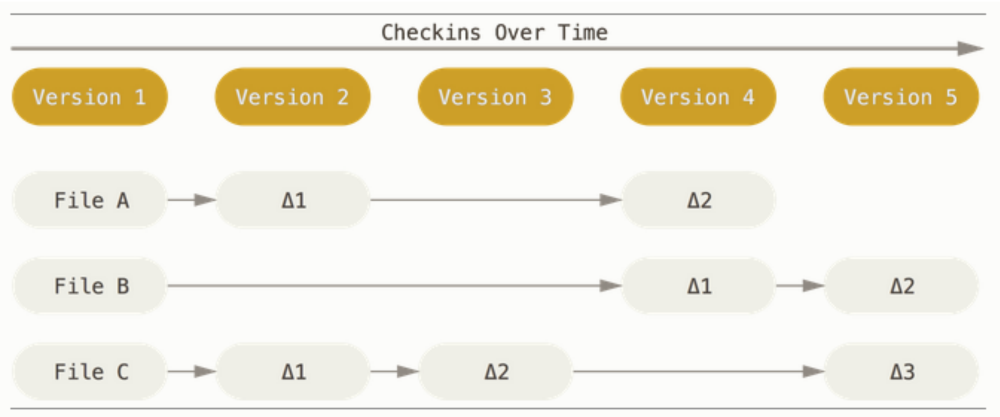
- Create a Github account
- Download and Install Git via http://git-scm.com/ . Follow the instructions post-download to complete the installation on your computer. Once git is installed, follow these instructions to configure git on your computer: https://help.github.com/articles/set-up-git/#platform-mac.
    - PC Users: When you install Git, it also will include a program called GitBash. Later on, when we refer to the "Terminal", that will be equivalent to GitBash on your computer
- Set up Git on your computer: Follow the instructions on this site. NOTE: Under the "Set Up Git" header, there are small links for set up based off of your computer type. Make sure you're using the set-up instructions for your computer type. https://help.github.com/articles/set-up-git/
- Create a repository to host your Product School blog at https://github.com/new
- Extra Credit: Read up on the basics of git: http://git-scm.com/book/en/v2/Getting-Started-Git-Basics

Introduction:

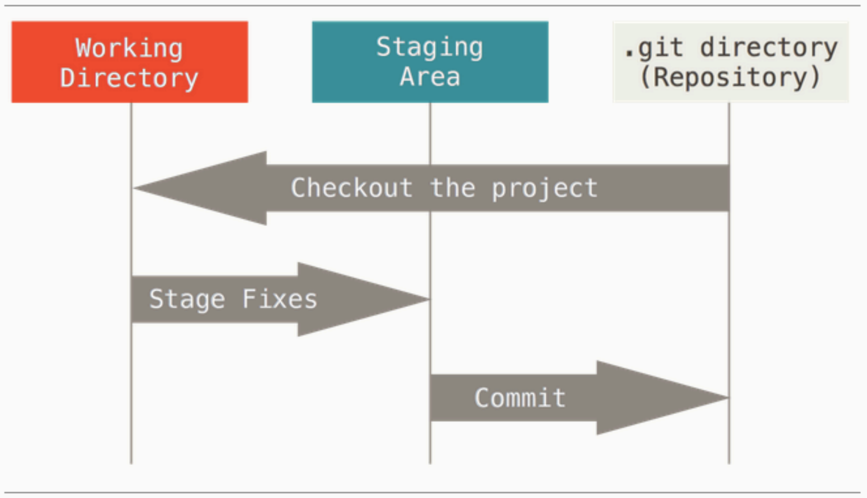
The combination of Git (on your local computer) and Github (online) is a powerful toolset that enables developers to have a record and access to any versions of the code they create. Github allows developers to have a single repository online where they can share code and review new code before it is merged into their production code base. We'll cover the basics of Git, Github, and how teams of developers use these tools to collaborate on their projects.

### 2.1. Introduction to Git

The basic functionality of git allows developers to track specific versions of their code-base, and revert to a previous version at any time (say, their most recent changes broke their system and they wanted to return to the previous version of their code). Git also enables developers to interact with online platforms like Github and Heroku where they can share and host their codebase.



*Storing data as changes to a base version of each file.*



*Working directory, staging area, and Git directory.*

i. Navigating your file structure using your Terminal[5]

- Traversing your file structure using your terminal
  - Visiting your nearest parent folder: $ cd ..
  - Traversing downwards: $ cd folder_name/child_folder_name
  - Going back to your roots: cd ~

- Orienting yourself within your file structure
  - Where am I?: $ pwd
  - What's inside this folder: $ ls

- Shortcuts
  - Auto complete (file/folder name): tab
  - Go to beginning of line: control + a
  - Go to end of line: control + e
  - Erase line: control + k

ii. Getting started with Git

- Within your terminal, navigate to your blog folder
- Create a git repository: $ git init
- Check out the status of your git repository: $ git status
- Stage all of the files in your folder to be included in your first commit: $ git add -A (to add individual files: $ git add file_name)
- Make your first commit: $ git commit -m "Some notes on your first commit

iii. Reverting to past commits

- Discarding any uncommitted changes: $ git reset --hard HEAD
- Revert to past commit state:
  - See a list of past commits: $ git log
  - Revert to a past commit: $ git reset --hard [#shah ###]

[5] For Windows users, the Windows equivalent of Apple's Terminal is the Command Prompt. To open a Command Prompt session, click the: "Start → Program Files → Accessories → Command Prompt".

iv. Creating and Merging Branches

- Creating a new branch: $ git checkout –b [branch-name]
- Moving between branches: $ git checkout [branch-name]
- Merging a new branch with your master code:
  - Navigate into your master branch: $ git checkout master
  - Merge your newly committed changes from your branch into master: $ git merge [branch-name]

## 2.2. Github

<u>Introduction</u>:

Github is a platform that enables developers to store their codebase online, review developers' code branches, and share feedback and error shooting before it's integrated in the full code base. Similar to Git, it stores a full-history of any changes made over the codebase life cycle, creating a safeguard as the number of developers working with the codebase increases over time and your codebase grows in complexity.

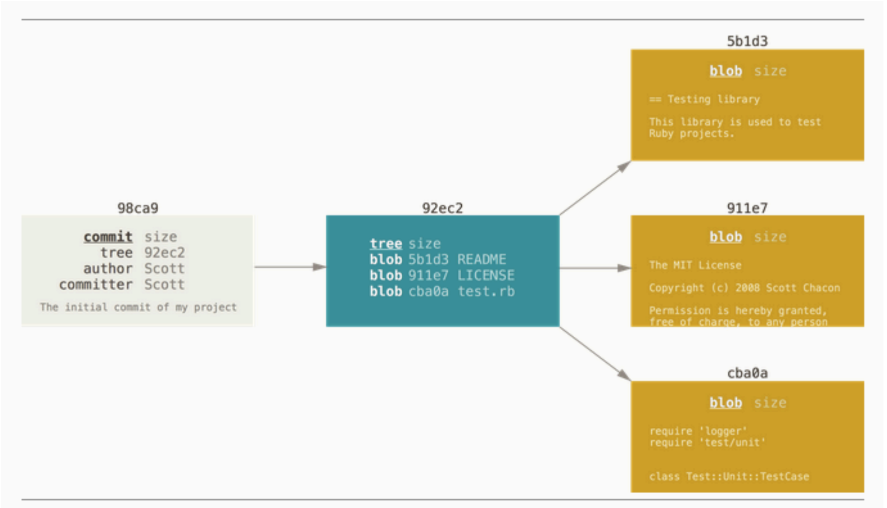
i. Linking your local Git repository with your Github repository

- Visit your Github repository and copy the "HTTPS Clone URL". Enter $ git clone [git clone url]
- Go to your terminal, navigate to your blog file and link your git repo with your github repository: $ git remote add origin [your HTTPS Cloned URL]
- Now that your repos are linked, push your first commit to github: $ git push -u origin master

ii. Creating viewable, sharable Github Pages

- Now that you're code is on github, let's make the code render in your browser through Gh-Pages

**Git's Branch Structure**



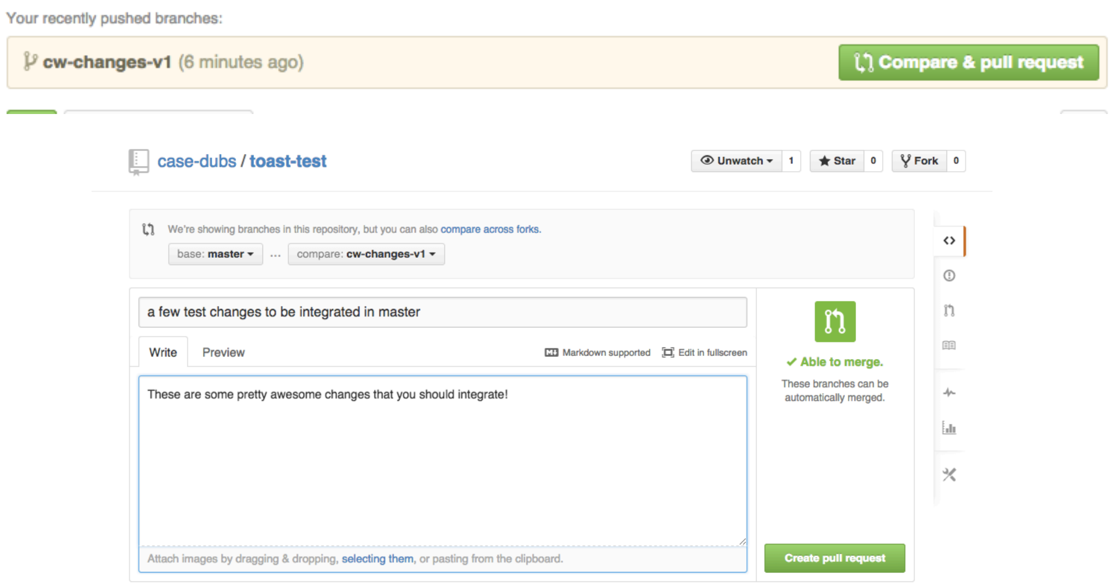
*A commit and its tree*

- Create a new gh-pages branch in your terminal's blog git repo: $ git checkout -b gh-pages

- Push your blog's git repo to the gh-pages branch of your github repository: $ git push -u origin ph-pages
- View your code at: http://[your-github-username].github.io/[your-blog-file-name]/

iii. Pull Requests: Screening your developments before releasing your code

- Let's create a new branch where we'll make some new changes that we'll review before integrating into our master code base: $ git checkout -b [your-branch-name]
- Go ahead, make some changes to your code
- Stage your changes for your commit: $ git add -A
- Commit your changes: $ git commit -m "A message about your changes"
- Push your branch and new commit to github: $ git push -u origin [your-branch-name]
- Visit your github repo. You should see that your Github registered your pushing a new branch. Go ahead and click 'Compare & pull request' to submit a request to merge your code changes into your master code base. Don't forget to include a message so that others know why they should include your changes.



Your recently pushed branches:

cw-changes-v1 (6 minutes ago)

case-dubs / toast-test

We're showing branches in this repository, but you can also compare across forks.

base: master ... compare: cw-changes-v1

a few test changes to be integrated in master

These are some pretty awesome changes that you should integrate!

Attach images by dragging & dropping, selecting them, or pasting from the clipboard.

✓ Able to merge.

These branches can be automatically merged.

- Once your pull request is open, go ahead and merge your code to the master code base.
- Problem, when we visit gh-pages, we still see the old version of your code. To update gh-pages, in the terminal, lets return to our master branch ($ git checkout master), pull the latest master version from github to our local repository ($ git pull), checkout out gh-pages branch ($ git checkout gh-pages), merge the updated master code-base into our gh-pages branch ($ git merge master), and, finally, push that updated gh-pages branch commit to github ($ git push)
- Visit your github gh-pages vanity url and you should see those newly made changes

## UNIT 3 – RUBY

In-Class Setup:

1. MAC Install Ruby & Ruby's Version Manager:
   - Check what version of ruby you have: Ruby comes pre-installed on Macs. We'll want to use a version of ruby that is 2.0.0 or above. Enter `ruby –v` in your terminal to see which version of Ruby you have.
   - Install RVM: RVM (Ruby Version Manager) allows you to separate your ruby projects. In your terminal, install ruby by entering: `\curl -sSL https://get.rvm.io | bash -s stable --ruby` . If you see an error, make sure you've installed mpais public key by entering `gpg --keyserver hkp://keys.gnupg.net --recv-keys D39DC0E3`. Note: If you run those two commands and still see error, please keep your terminal open with the error message and get help in class.
   - Install Ruby version 2.0.0: Open up a new terminal window, then type `rvm install 2.0.0` .
   - Make Ruby 2.0.0 Your Default Ruby Version: In that new terminal window, make ruby version 2.0.0 your default version by typing `rvm --default use 2.0.0`. Verify that you have ruby installed and are using version 2.0.0 by entering `ruby –v` in your terminal.
   - Update Gem Manager and Default Gems: Gems are libraries or plug-ins that enable your Ruby on Rails application to have access to extra functionality without you having code it yourself. To update your default gem manager, type `gem update --system` . If you're default gem manager is already up to date, you'll receive a message saying "Latest version currently installed. Aborting." Otherwise, you should see that your gem manager has updated. Now that your gem manager is up to date, lets update the gems that you have access to by default in the current ruby version and your global ruby version by entering these four commands in your terminal: 1. `gem update` , 2. `rvm gemset use global` , 3. `gem update`, 3. `rvm gemset use default` to return to your default gemset.
2. PC:
   - Download Ruby and Rails: Follow the instructions for windows download and setup http://railsinstaller.org/en

Introduction[6]:

Ruby is a backend scripting language. Many of the JavaScript constructs that you learned, such as functions, variables, objects, iterators, and conditionals, are very similar to those used in Ruby. The main difference between Ruby and JavaScript is that JavaScript is asynchronous (multiple tasks can be running at one time), while Ruby is synchronous (tasks must be run sequentially). One of the benefits of Ruby is that it includes a huge library of methods and built in functionality[7], which makes your life a lot easier. Another benefit of Ruby, is that Rails, a Model-View-Controller (MVC) framework with an abundance of open-source functionality plug-ins that you can integrate into your app, is built on top of it enabling you to combine the two to quickly build awesome websites.

*Note: Open your terminal and type irb to open a Ruby console. As you learn each step below, please follow along by practicing what you're learning in your terminal.

3.1 What is an Object? Everything in ruby is an object. Whether that is a string, complicated data structure, or functions. A few common objects include:

- Integer: whole number (eg: 1)
- Float: numbers that include decimal points (eg: 1.3)
- Strings: a piece of text. (eg: "hello world")
- Booleans: true / false
- Arrays: a series of values. (eg: [1, 2, 3, 4])
- Hashes: a collection of key value pairs. (eg: casey = {"age" => 31, "height" => 5.8, "occupation" => "web developer"})
- Methods: Bundle functionality. Analogous to functions in JavaScript.
- Classes: Serve as 'factories' or 'molds' for creating new objects with a particular specifications and functionalities.

[6] For more practice with Ruby, check out: http://rubymonk.com/learning/books/1-ruby-primer
[7] Ruby documentation with a full list of built in functionality: http://www.ruby-doc.org/core-2.2.0/

3.2 Data Types:

- Integers: Integers are whole numbers. Any arithmetic done on an integer will always produce another integer. For example, 1 / 2 will produce 0.
  - In order to deal with fractions, you have to use floats. You can do this by using Ruby's built in .to_f method. For example: (1.to_float)/2 → .5.

- Floats: Stands for floating point. It allows you to work with numbers that have decimal places.

  In-Class Exercise: Use floats and integers to divide two objects and return a fraction. Then, convert that fraction into an integer.

- Strings & String Interpolation: Strings represent text. For example "Hello world" or "34 is my favorite number".

  - String Interpolation: In ruby allows you to combine strings by adding them together. Eg: "Hello" + " World" → "Hello World".

  - Including variables in strings: In ruby, you can include a variable in a double quoted string (a string wrapped in " " instead of ' ') by surrounding the variable with #{variable}. For example, if have a variable, myName = "Min". Then "Hello #{myName}!" → "Hello Min!"

  In-Class Exercise: Create a string of text that includes three variables.

- Booleans: True or false. In Ruby, you might have a method that returns true/false, or you might use a statement that evaluates to a Boolean for an if…else statement. For example, if (2 + 2 == 4)….else…end In this case, (2+2 == 4) evaluates to true or false. If true, then it will perform the subsequent operation, otherwise, it will perform the operation detailed in the else statement.

In-Class Exercise: Write 5 statements. 2 that evaluate to true. 3 that evaluate to false. Make each one as unique as possible.

- Transforming Data Types: Ruby has a number of built in methods to transform one data type to another.

  - To Float - .to_f : Transforms a string or integer into a float. "1".to_f → 1.0 ; 1.to_f → 1.0

  - To Integer - .to_i: Transforms a string (eg: "5") or a float into an integer. "5".to_i → 5; This can be really helpful when you have a string representation of a number that you want to perform numeric functions on.

  - To String - .to_s: Transforms an integer or float into a string. 1.to_s → "1". Similarly with to_i, you'll find this helpful at times when you want to use numbers for displays.

- Arrays: An array is a combination of objects separated, each separated by a comma. Eg: [1, "two", three, {hello: world, weather_today: "sunny"}]. Arrays are an incredibly important building block as we start building out full backend applications. For example, we'll query and work with all of the blog posts, categories, and users via arrays of objects. Your ability to access the array data that you want and use common array methods will make produce dividends as a coder.

  - Accessing Array Data: Imagine we have an array with values between 1 and 6. ourArray = [1,2,3,4,5,6]

    - Accessing A Single Array Object: ourArray[object_index]. Arrays start with a 0 index, meaning that ourArray[0] → 1. ourArray[5] → 6. ourArray[100] → nil.
    - Accessing a single array object counting down from the end: To access the last array object, counting backwards, you use negative numbers, starting with -1. ourArray[-2] → 6.
    - Accessing a specific number of objects, from a index onwards: if you wanted to access 3 array objects, starting with the 2nd object in the array: ourArray[1,3] → [2,3,4]

  - Accessing all objects between to objects: ourArray[1..4] → [2,3,4]. This is inclusive, in that it includes the first object, last object, and everything in between.

    In-Class Exercise: Create an array with 8 array items. 1. Print out the third item in the array. 2. Print out the second to last array item. 3. Print out the first 3 items in the array. 4. Print out all array items between the 2nd and 5th array item.

- Commonly used Ruby array methods:

Practice Array: ourNames = ["Min", "Gaurev", "Adam", "Anastassia", "Constantin", "Casey"]

  - Length - .length: ourArray.length → 6. The number of objects in an array.
    In-Class Exercise: How long is our array?
  - Count - .count: ourArray.count → 6. The number of objects in an array.
    In-Class Exercise: How many items are in our array?
  - Each – .each: Allows you to loop through each object in an array to perform a specific operation. (More on this in the iterators section). Eg: ourArray.each do |item| puts item end
    In-Class Exercise: For each member of our class, print our "Hello, personsName!". Eg: "Hello, Min!"
  - First - .first: Returns the first element in an array.
    In-Class Exercise: Return the name of the first name in ourNames array.
  - Last - .last: Returns the last element in an array.
    In-Class Exercise: Return the name of the first name in ourNames array.
  - Pop - .pop: Removes the last element of an array. ourArray.pop → 6 . ourArray is now [1,2,3,4,5]
    In-Class Exercise: Return the last array name in ourNames array. What is the new length of your array?
  - Push - .push: Inserts new objects into the end of an array. ourArray.push(7) → [1,2,3,4,5,6,7] ; ourArray.push(8,9) → [1,2,3,4,5,6,8,9]
    In-Class Exercise: Add the name "Carlos" to your array. What is the new length of your array?
  - Reverse - .reverse: Returns a new array in the reverse order. This is only temporary. It doesn't actually transform the array object. ourArray.reverse → [6,5,4,3,2,1] ; ourArray is still [1,2,3,4,5,6]
  - Index – index: Returns the index of that item in the array. ourArray.index(3) → 2.
  - Unique – uniq: Returns a new that only includes the unique values in array. Eg. [1,2,2].uniq → [1,2]
  - More on arrays and array methods: http://www.ruby-doc.org/core-2.2.0/Array.html

• Hashes: Hashes are a combination of key: value pairs. Much of the data that we'll be working with in ruby is stored in the form of a hash. For example, if carlos were are user, we might store his information as such: carlos = {height: 70, weight: 170, age: 29, profession: "entrepreneur"}.

- Accessing hash data: You can access a particular key-value pair using the syntax: variable_name[:key]. Eg: carlos[:age] → 29

- Hash syntax options: There are three different ways of defining a hash. The two syntax types that include a => are known as 'hash rockets'.
  - {key: value}

    - {“key” => value}
    - {:key => value}.

- More on hashes and hash methods: http://www.ruby-doc.org/core-2.2.0/Hash.html

3.3 Operators:

i Arithmatic Operators:

- Plus: a + b
- Minus:  a - b
- Times: a * b
- Divided by: a / b
- Modulo: Returns the division remainder of two numbers. Eg: 5 % 3 → 2. 6 % 3 → 0.
- Exponent: a ** b

ii Comparison Operators:

- == : Checks whether the value of two operands is equal. If yes, evaluates to true, if no, evaluates to false
- != Checks whether the value of two operands ARE NOT equal. If yes, evaluates to true, if no, evaluates to false
- >: Checks if operand on left is greater than operand on right. If yes, evaluates to true, if no, evaluates to false
- <: Less than
- <=: Less than or equal to
- >=: Greater than or equal to
- <==>: Combined comparison. Returns 0 if both operands are equal. 1 if left operand is greater. -1 left is lesser.

iii Assignment Operators: Permanently change the value of an operand.

- = : Assigns one value to equal another value. Usually used for assigning variable values. Eg: myVariable = 2.
- And assignment operator +=: Adds value to operand. Two = 2. two += 1 → 3
- Subtract And assignment operator -=. two -= 1 → 1
- Multiply and assigment operator *=. two *=2 → 4.
- Divide and assigment operator  /=. two /= 2 → 1
- Module and assigment operator two %= 2 → 0
- Exponent and assignment operator. **=

3.4 Variables: Unlike JavaScript, in ruby there’s no need to include a ‘var’ or end a line with a ‘;’

```
myNumber = 5
myString = “Hello Friends”
```

3.5 Methods: One ruby object talks with another one via methods.

i. Built in Ruby Methods: Ruby includes a host of built in methods. Which methods are available to which object depends on the type of object. For example, numbers, have access to methods like even?, odd?, and integer?, while strings have access to methods like length

```
myNumber.even? → true

myString.length → 13
```

For a full list of ruby methods: http://www.tutorialspoint.com/ruby/ruby_builtin_functions.htm

ii. Returning values from your method:

- Puts prints out a value
- Print prints out a value
- Return returns the value

```
def sayHello(name)
    puts name
end

def firstUser
   return User.first
end
```

iii. Creating Your Own Methods:

```
def method_name(parameters)  ← starts with 'def' + the method name + parameters if required
          ## your functionality
          ## method will automatically return the value of the last line of the method
end                                        ← close out the method with 'end'

def times_two(number)
          puts number*2
end

times_two(122) → 244

def hello_old_friend(name, yearsSinceIveSeenYou)
          puts "Hello #{name.capitalize}. It's great to see you again after #{yearsSinceIveSeenYou} years!"
end

hello_old_friend ("Carlos", 5) → "Hello Carlos. It's great to see you again after 7 years!"
```

iv. Rules for Defining Methods:

- Name should start with a lowercase letter.
- Use camel-case to chain words together (eg: helloOldFriend)
- Try to make the name as descriptive so that their function is clear.
- Must define method before calling it, otherwise you'll see an error.

3.6 Conditional – If…Else – Statements:

```
if condition
          #action to be run, if condition is true
elsif condition
          #action to be run, if condition is true and previous condition isn't true
else
          #action to be run if the above conditions are not true
```

```
end
```

In-Class Exercise:

Build your own FizzBuzzer. Create a method that takes an integer input. If the integer is divisible by 3, return ‘Fizz’. If it’s divisible by 5, return ‘Buzz’. If it’s divisible by 3 and 5, return ‘FizzBuzz’. And, if it’s not divisible by 3 or 5, simple return the number.

3.7 Iterators

i. Each – Iterating Over an Array: The most common iterator you’ll use in ruby is the .each method. It loops through each element in an array, executing a block of code each time. Let’s use the example of printing out the names of all of the users of our app:

```
userNames = [“John Doe”, “Jane Austin”, “Frank Copula”, “Carlos Gonzales”, “Sofia Garcia”]

userNames.each do |user_name|  ← pattern: array_name do |array_item|
        puts user_name
end
```

In-Class Exercise: Create a fun iterator that prints a welcome message for each user in the userNames array. Include the index of that user in your message.

ii. Map: Iterates over an array and returns a new array containing the values returned by the specified operation:

```
users = [{name: “John Doe”, age: 30, profession: “developer”}, {“Jane Austin”, age: 35, profession: “writter”},  {“Frank Copula”, age: 87, profession: “film producer”}, {“Carlos Gonzales, age: 28, profession: “entrepreneur”}]


hello_users = users.map do |user|
        user + “, Hello!!!”
end

hello_users → [“John Doe, Hello!”, “Jane Austin, Hello!”, “Frank Copula, Hello!”, “Carlos Gonzales, Hello!”, “Sofia Garcia, Hello!”]
```

In-Class Exercise: Print out the name and profession of each of your users.

iii. Times: You can also use the .times iterator to run a block of code a specific number of times. Let’s use this to print out numbers between 1 and 5.

```
counter = 0

5.times do
        counter += 1
        puts counter
end
```

In-Class Exercise: Using just your userNames array, times, a counter, an And Assignment Operator, and a puts statement, print the numbers between 1 and 5.

3.8 Classes

i. What is a Class? A class enables you to group a number of methods and variables together. Classes act as re-usable factory like structure that you can use to create custom objects.

```
class BestFriend
```

```
        def initialize(name) ← The initialize method is what's called when you use the class factory to
                               create a new object. Eg: carlos = BestFriend.new("Carlos")
          @name = name ← variables that start with an '@' are instance varibles, which are accessible
                         anywhere within the class
        end

        def say_kind_things
                puts "#{@name}, you're my favorite person on earth. Don't forget that!"
        end

        def invite_me_to_dinner
                puts "Hey, #{@name}. Want to grab dinner tonight?"
        end
end
```

ii. Using Classes to Generate New Objects: As you'll remember from before, classes operate as a factory that builds individual objects, but are not objects themselves. If you want to use a class to create an object, you'll need to create a new instance of the class:

bestie = BestFriend.new("Carlos") ← pattern: ClassName + .new + (initialize_method_paramenters)

bestie.say_kind_things → Carlos, you're my favorite person on earth. Don't forget that!

besite.invite_me_to_dinner → Hey, Carlos. Want to grab dinner tonight?

In-Class Exercise: Create your own new class that includes at least 2 methods. Once you're done, share it with our "class" on our projector screen.

HOMEWORK:

1. Ruby: Complete Ruby Monk's Ruby Primer: https://rubymonk.com/learning/books/1-ruby-primer

2. Responsive Styling: Add responsive styling to your blog.