Front-End-Developer - Level 2

Opteamize, an IBM Coding School

Table of Contents

Introduction to Level 2, IBM Coding School

> **Embedded Video: https://www.youtube.com/watch?v=6MaOPdQPvow?rel=0**

Congratulations on completing Level 1! Before we introduce you to Level 2, we recommend that you start reading some good articles and blogs regularly.

We have compiled a list of blogs that describe the journey to becoming a good programmer. Do read them and it will help you make the most of your time at Opteamize and during your job search and after you join your new job! Happy reading!

- 7 reasons every programmer needs to learn JavaScript
- 4 reasons to learn JavaScript as your 1st programming language
- Why some JavaScript training sucks?
- 10 mistakes that JavaScript beginners make
- 7 JavaScript interview questions to weed out imposters

CSS Intermediate

These are some topics you are expected to learn and understand if you are in a CSS Intermediate class.

- Class and ID Selectors: Make your own selectors without the need for sticky-backed plastic!
- Grouping and Nesting: Properties assigned to multiple selectors or selectors within selectors.
- Pseudo Classes: Defining various states of a link selector.
- Shorthand Properties: Various properties, such as borders and margins that amalgamate other properties into one.
- Background Images: Guess. Specificity: How a browser will deal with conflicting CSS rules.
- Display: Specifying the characteristics of a box.
- Pseudo Elements: Styling first letters, first lines and placing content before and after elements.
- Page Layout: Floating and positioning boxes.

CSS Refresher

Watch videos 1-7 of Travis Neilson's CSS Basics where he introduces the following topics:

- What is CSS?

> **Embedded Video: https://www.youtube.com/watch?v=s7ONvlgOWdM?rel=0**

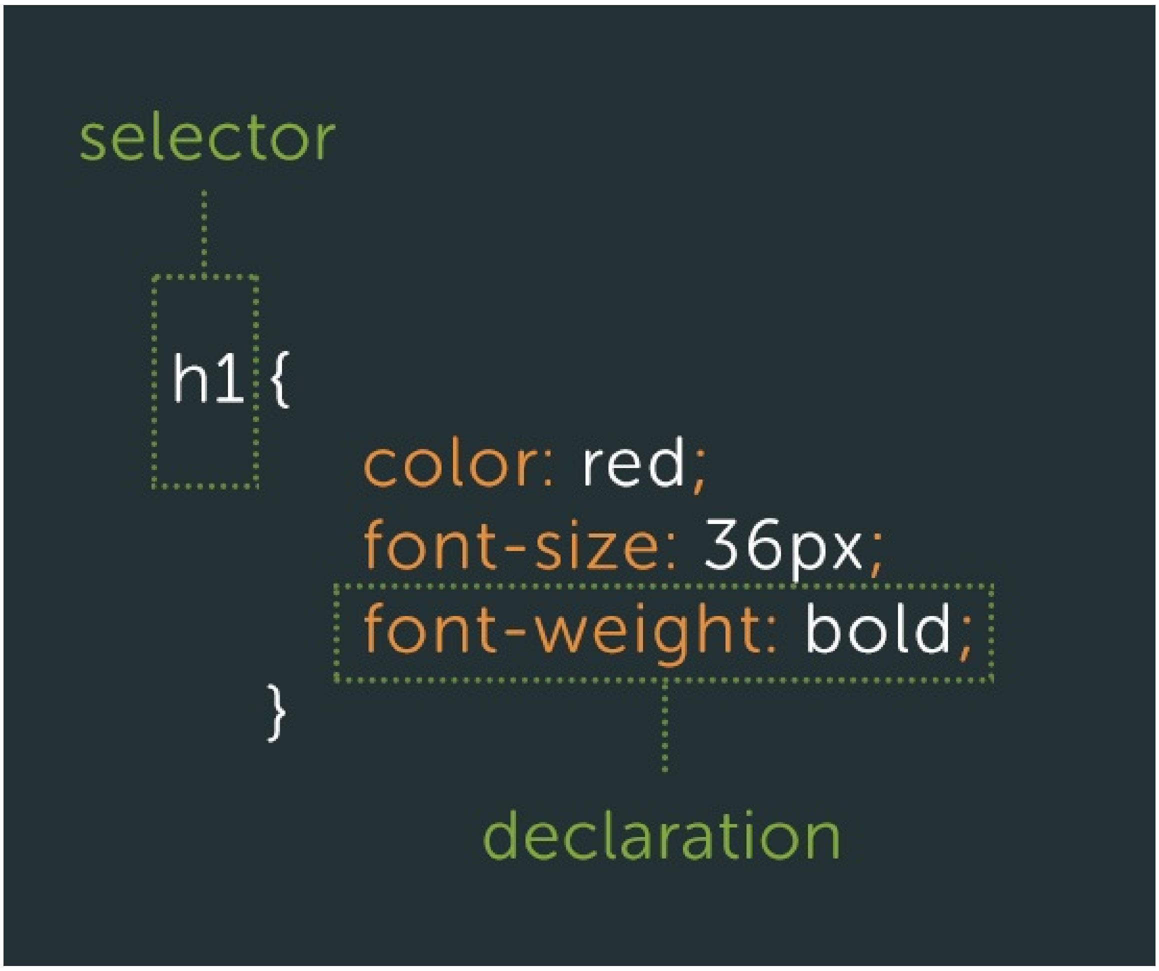
- Cascades
  > **Embedded Video: https://www.youtube.com/watch?v=tZhmjgLQgXU?rel=0**
- Selectors
  > **Embedded Video: https://www.youtube.com/watch?v=emMO3iCpvrc?rel=0**
- Property and values
  > **Embedded Video: https://www.youtube.com/watch?v=4LtwZQ5jxic?rel=0**
- Advanced Selectors
  > **Embedded Video: https://www.youtube.com/watch?v=oh2JLo2yxCM?rel=0**
- Specificity
  > **Embedded Video: https://www.youtube.com/watch?v=fy07HYm-geM?rel=0**
- CSS3
  > **Embedded Video: https://www.youtube.com/watch?v=Qkx3avfK28k?rel=0**

We have also compiled a list of concepts you should know from Level 1:

- The difference between block elements and inline elements
- How to style text
- How to style with classes
- How to organize with divs and spans
- The basics of laying out with floats
- The Box Model
- Cascading
- How to use Bootstrap
- How to create rows and columns using the grid system

Another look at CSS Selectors

Selectors allow us to target and style specific elements on a page.



This is a Type Selector, which matches HTML element names. *Image courtesy of* Tuts+

You're also familiar with `class` selectors, which target all elements with the same class (e.g. `<div class="content">`):

```
.content {
  padding: 0;
}
```

And you know how to target a single, specific element using id selectors (e.g. `<img src="img/baby_1" id="baby-photo-1"/>`):

```
#baby-photo-1 {
  max-width: 300px;
}
```

But we've only just scratched the surface. Have you wrapped your head around targetting parent elements and child elements yet? Maybe some cat visuals will help.

Familiarizing yourself with the multitude of selectors available to you (and how they work) will be crucial in the coming weeks as you practice writing good clean CSS.

For an overview of common selectors you should know starting out, please read this fantastic article by Chris Coyier.

Pseudo-Classes

You've already had a some practice with pseudo-classes (perhaps without being fully aware of it), but check out this basic refresher on what they are and how they work. You might also want to check out a few other use cases for them.

Codrops has an awesome reference list if you want to check that out as well.

Note: You do not need to know how to do every example from these resources, per se. The purpose of this particular lesson is to reinforce a basic familiarity with the many pseudo classes available to you -- it will come in handy in the coming days.

Summary

ia Treehouse:

Think of a pseudo-class as a keyword we add to a selector to style a special state of an element.

The :link pseudo-class targets links that have not been visited by the user:

```
a:link {
  color: orange;
}
```

The :visited pseudo-class targets links that have been visited — or clicked — by the user:

```
a:visited {
  color: lightblue;
}
```

The :hover pseudo-class targets an element when a user hovers over it:

```
a:hover {
  color: forestgreen;
}
```

The :active pseudo-class gets applied when an element is in an active state:

```
a:active {
  color: lightcoral;
}
```

The :focus pseudo-class is only applied to interactive elements like links, buttons and form elements. The styles are applied as soon as the element receives focus:

```
a:focus {
  color: white;
  background-color: orange;
}
```

Styling HTML Forms

Read the first half of Mozilla's tutorial on styling HTML forms. Follow along all the way down to the Example image of the postcard -- but no further. We'll get our hands dirty tomorrow.

True to Form

Start your day off by following along and completing the Example section of Mozilla's form-styling tutorial (see last night's homework). Do not copy and paste.

Nobody enjoys filling out forms, but developers and designers can help make users' experience a little more pleasant with some styling. Check out this and this. As you can see, filling out forms doesn't have to be a dreary experience. Take some notes! Your next task is to craft some forms of your own.

Start with a basic sign-up or contact form. Get creative with your CSS. Style your UI. Experiment with basic pseudo classes like a:hover. When you're done with that, create and style something a little more sophisticated, like this one. Make it look like something an actual client would use in production.

Tables

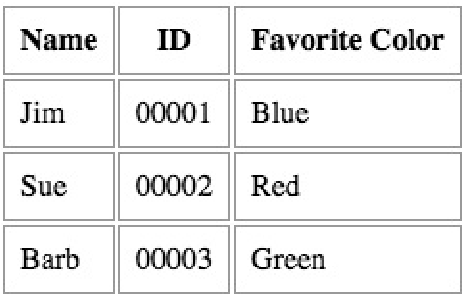
Out in the wild there'll be times when you want to create tables on the fly. Whether it's a sizing chart for a client's products, or a data analytics report your company wants to display on its website, the truth is that Bootstrap won't always be there to save you.

So take this opportunity to study Chris Coyier's Complete Guide to the Table Element. You'll be glad you did.

Summary

via CSS-Tricks:

Here is an example of a basic table and its corresponding code:



Name	ID	Favorite Color
Jim	00001	Blue
Sue	00002	Red
Barb	00003	Green

HTML:

```
<table>
 <thead>
   <tr>
     <th>Name</th>
     <th>ID</th>
     <th>Favorite Color</th>
   </tr>
 </thead>
 <tbody>
   <tr>
     <td>Jim</td>
     <td>00001</td>
     <td>Blue</td>
   </tr>
   <tr>
     <td>Sue</td>
     <td>00002</td>
     <td>Red</td>
   </tr>
   <tr>
     <td>Barb</td>
     <td>00003</td>
     <td>Green</td>
   </tr>
 </tbody>
</table>
```

CSS:

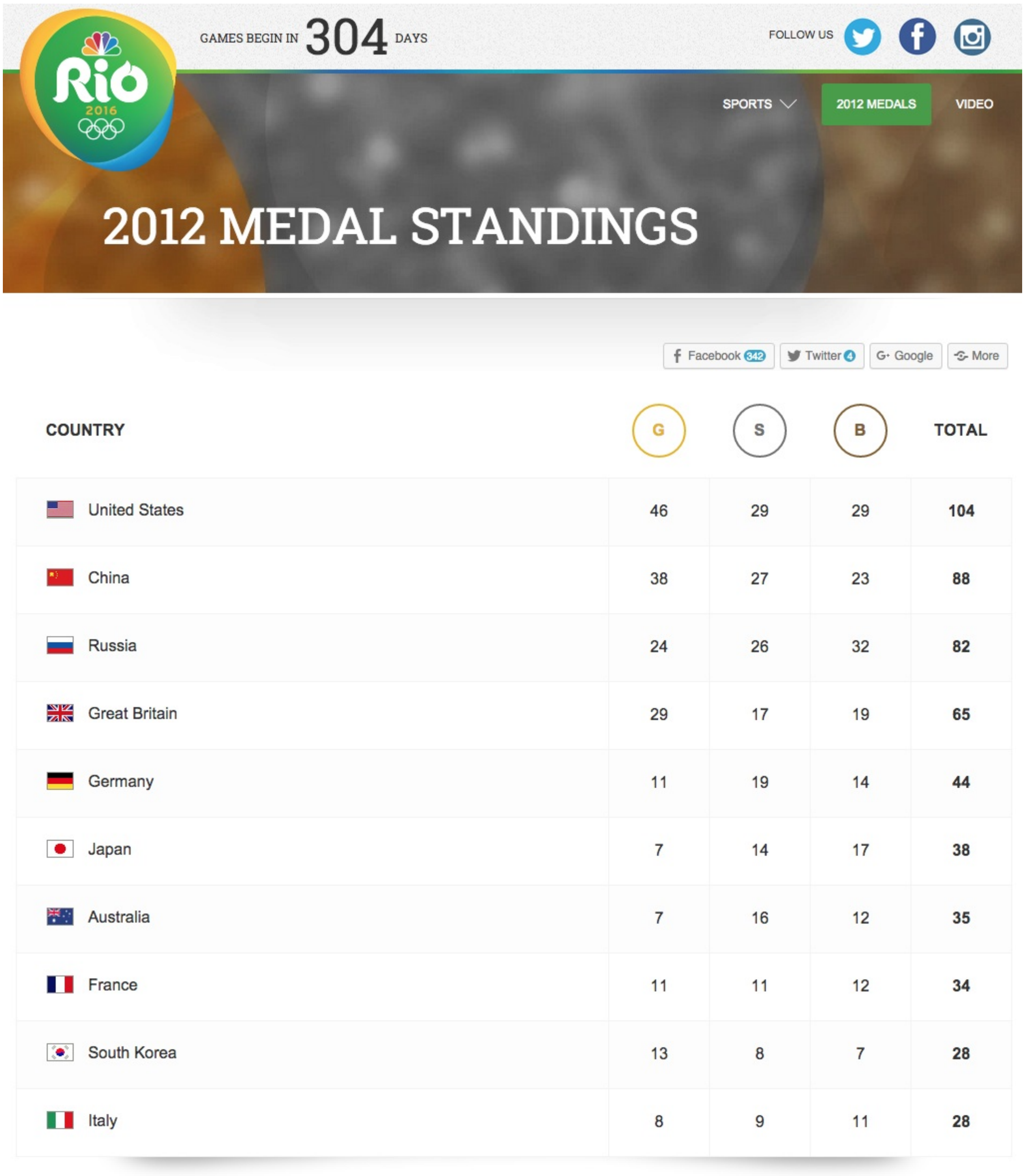
```
td, th {
 border: 1px solid #999;
 padding: 0.5rem;
}
```

Turning the Tables

Today you will practice building fully-fleshed out tables from scratch.

Friendly reminder: no Bootstrap!

Olympic Medal Count

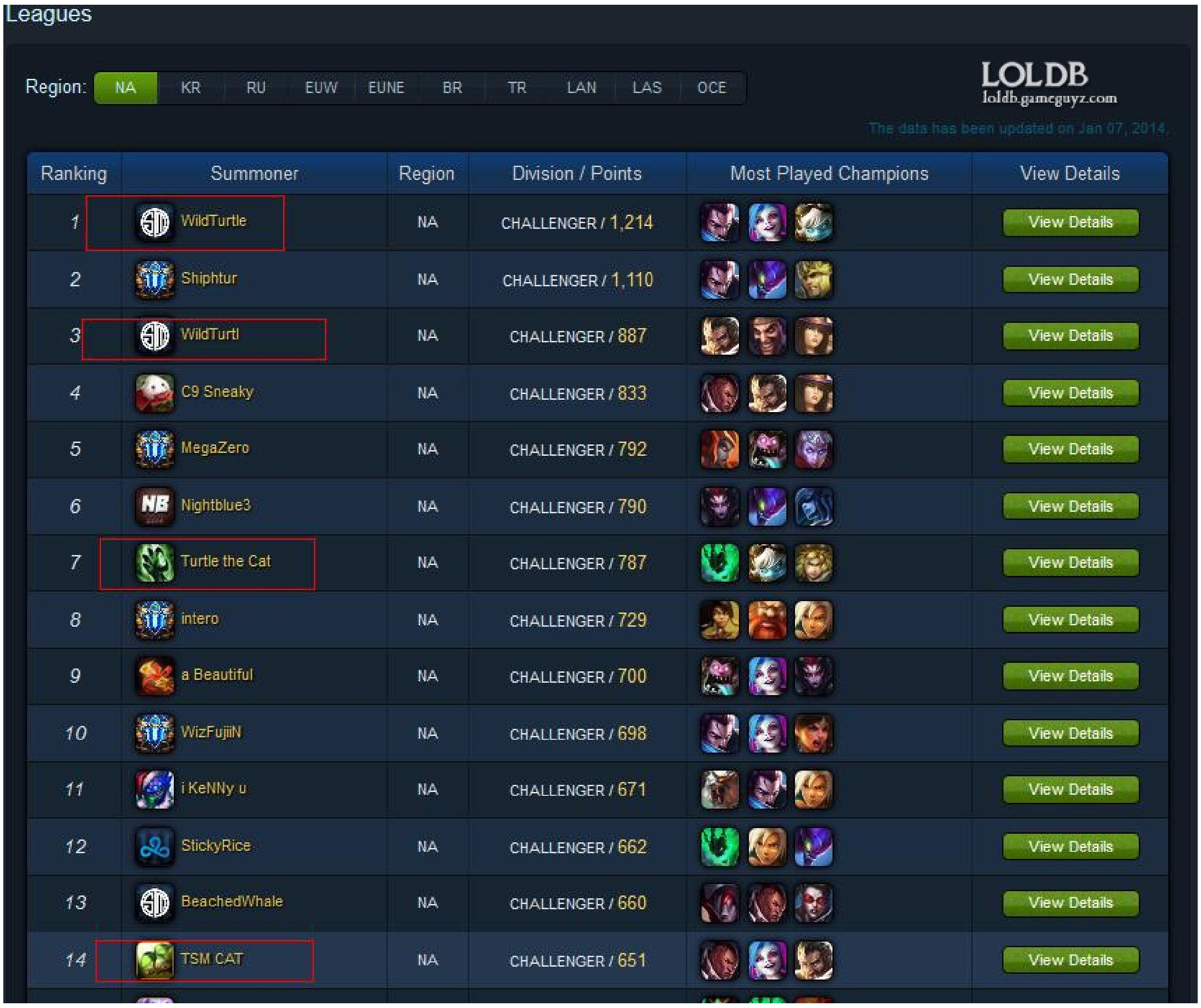


COUNTRY	G	S	B	TOTAL
United States	46	29	29	104
China	38	27	23	88
Russia	24	26	32	82
Great Britain	29	17	19	65
Germany	11	19	14	44
Japan	7	14	17	38
Australia	7	16	12	35
France	11	11	12	34
South Korea	13	8	7	28
Italy	8	9	11	28

Your challenge is to make a page as close in resemblance to the above as possible. There are a lot of cool elements to style. Be sure to prioritize the table itself before anything else.

Once you're finished, you'll have three options:

Option A: Gaming Leaderboards or Sports Standings

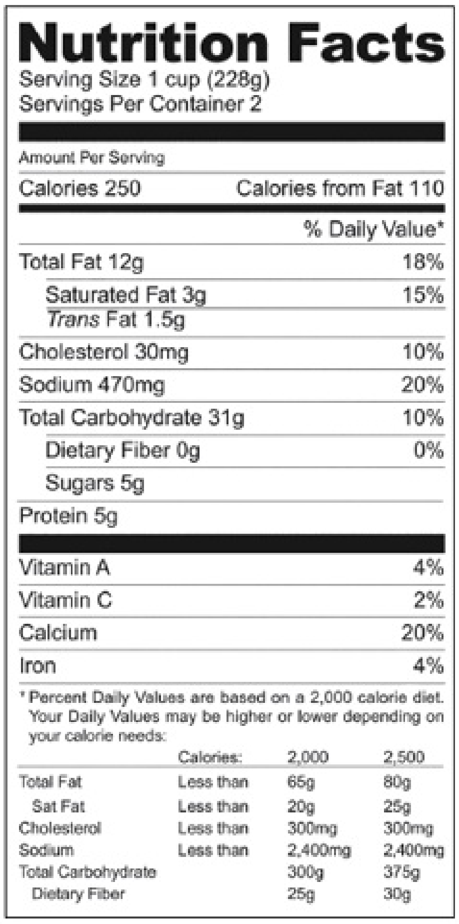


The data has been updated on Jan 07, 2014.

Ranking	Summoner	Region	Division / Points	Most Played Champions	View Details
1	WildTurtle	NA	CHALLENGER / 1,214		View Details
2	Shiphtur	NA	CHALLENGER / 1,110		View Details
3	WildTurtl	NA	CHALLENGER / 887		View Details
4	C9 Sneaky	NA	CHALLENGER / 833		View Details
5	MegaZero	NA	CHALLENGER / 792		View Details
6	Nightblue3	NA	CHALLENGER / 790		View Details
7	Turtle the Cat	NA	CHALLENGER / 787		View Details
8	intero	NA	CHALLENGER / 729		View Details
9	a Beautiful	NA	CHALLENGER / 700		View Details
10	WizFujiiN	NA	CHALLENGER / 698		View Details
11	i KeNNy u	NA	CHALLENGER / 671		View Details
12	StickyRice	NA	CHALLENGER / 662		View Details
13	BeachedWhale	NA	CHALLENGER / 660		View Details
14	TSM CAT	NA	CHALLENGER / 651		View Details

The above is just an example. Feel free to use different content.

Option B: Nutrition Facts



Nutrition Facts

Serving Size 1 cup (228g)
Servings Per Container 2

Amount Per Serving

Calories 250	Calories from Fat 110
	% Daily Value*
Total Fat 12g	18%
Saturated Fat 3g	15%
Trans Fat 1.5g	
Cholesterol 30mg	10%
Sodium 470mg	20%
Total Carbohydrate 31g	10%
Dietary Fiber 0g	0%
Sugars 5g	
Protein 5g	
Vitamin A	4%
Vitamin C	2%
Calcium	20%
Iron	4%

* Percent Daily Values are based on a 2,000 calorie diet. Your Daily Values may be higher or lower depending on your calorie needs:

	Calories:	2,000	2,500
Total Fat	Less than	65g	80g
Sat Fat	Less than	20g	25g
Cholesterol	Less than	300mg	300mg
Sodium	Less than	2,400mg	2,400mg
Total Carbohydrate		300g	375g
Dietary Fiber		25g	30g

This is a slightly trickier challenge than you think. See if you can make it pixel-perfect without resorting to online code examples!

Option C: Whatever table you want!

Examples include: sports standings, gaming leaderboards, mobile phone pricing plans, etc.

Remember, details matter.

Using CSS Image Sprites

An image sprite is a collection of images placed onto a single image file. This allows for us to upload a single image to our site (instead of, say, 10) and use CSS wizardry to display different sections of the sprite according to our needs. Less images means fewer HTTP requests and thus better performance overall. It's not something we really worry about too much in class, but it's a very common optimization trick you'll use in real-world apps. Every second of page-load really does matter, after all.

Read (and personally bookmark) this CSS-Tricks lowdown on how to use sprites. As mentioned in the article, there are many different ways to implement them. For class we'll use SpritePad; this simple web tool expedites the process of combining images into a single file, as well as generating the appropriate CSS for us all at once.

Basic layout objectives

This week we’ll practice concepts that still commonly puzzle new developers:

- absolute vs. relative positioning
- floats
- clearfixes
- basic media queries

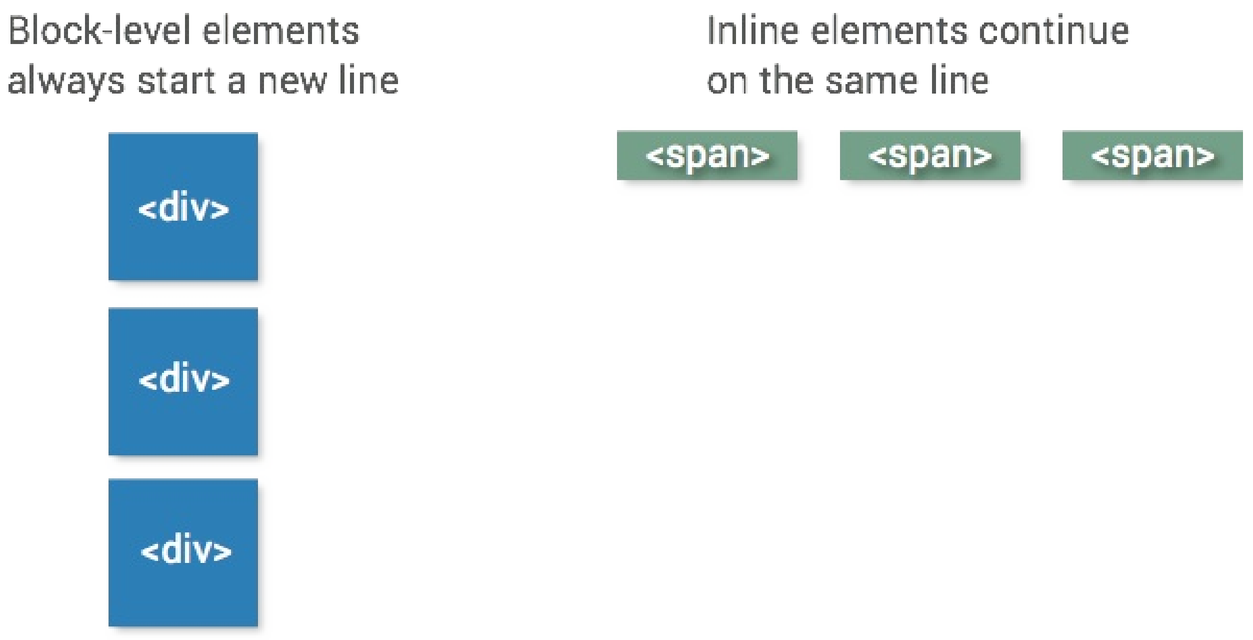
You’ve no doubt had plenty of exposure to them thus far, but let’s take this time to wrap our head around them once and for all.

In addition, we'll have a field day with some major front-end frameworks including Twitter's Bootstrap and Foundation by ZURB. Their built-in grid systems and plug-n-play components are among the most common and well-supported in the industry; thus, becoming proficient with them will make you much more marketable on the job front.

Block vs. Inline Elements

Make sure you understand the difference between block-level and inline elements by watching Treehouse’s overview. A good grasp of this distinction will be important this week as our layouts become more and more complex.

Summary



Block-level elements always start a new line

Inline elements continue on the same line

Other examples of block-level elements:

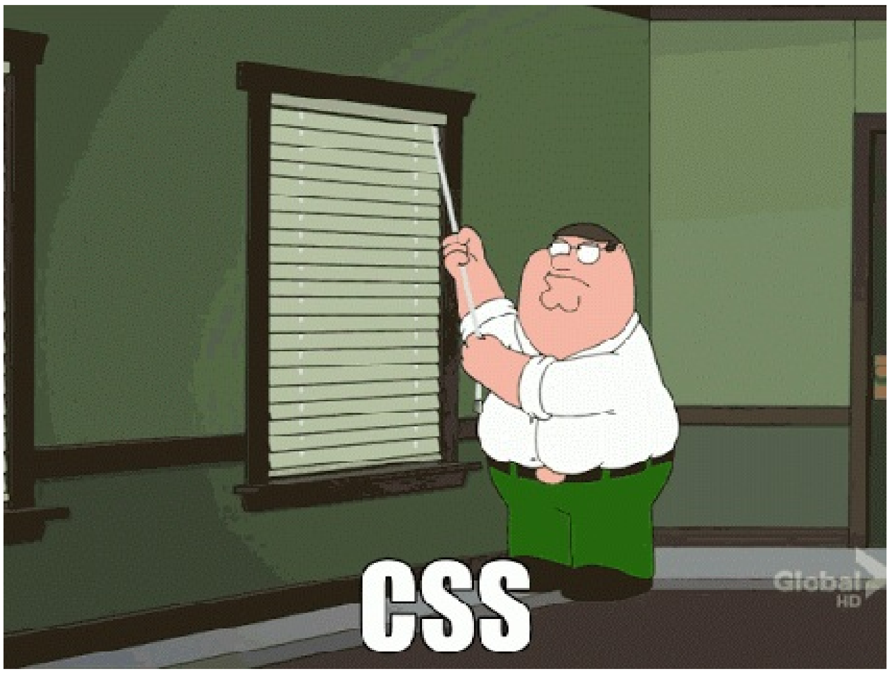
- <h1> - <h6>
- <section>
- <form>
- <ul>
- <p>

Other examples of inline elements:

- <a>
- <b>
- <img>

Positioning

Some coders cringe at the question of absolute vs relative positioning. Definitions are simple enough, but it may be difficult sometimes for beginners to wrap their head around them. Take some time to study the concept, as it'll save you many hours of frustration in the long run.



Also, check out this awesome cheat sheet.

Precise Positioning

Today we are going to “flex our newfound positioning muscle”.

Endangered Species Map Infographic

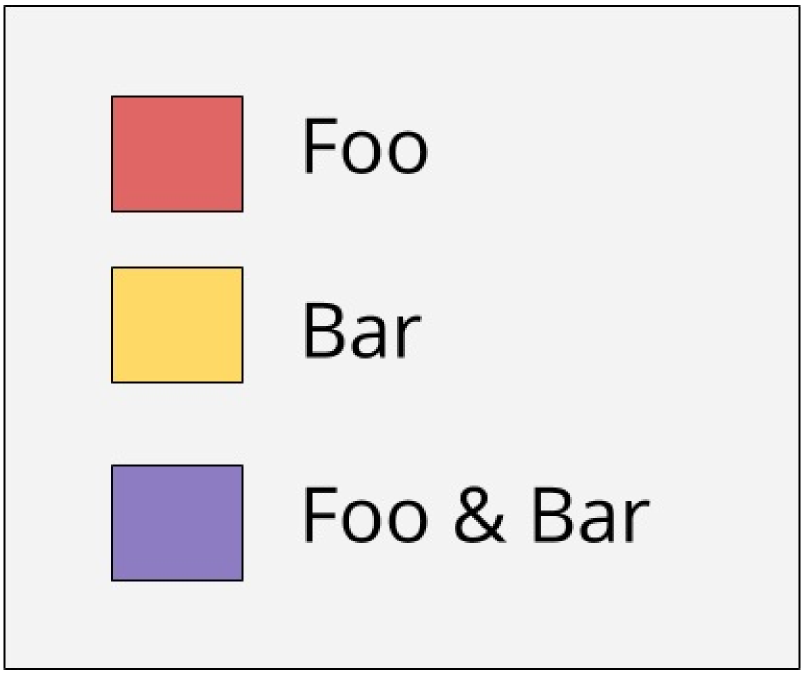
You will be creating a map displaying overlaid images of hand-picked endangered species along with their details.

1. Choose at least 10 species total. Try to include every continent. This site should prove helpful.
2. On a new webpage, set up a div with a class of “map” and place a blank world map image into it. I recommend using this world map example, but you’re welcome to find another.
3. You will also need a fixed masthead or overlayed header so users know what they’re looking at.
4. Using Google, search for images of the endangered species you’ve chosen. I recommend using .JPEG files for performance, but for higher resolution photos you’ll want to use .PNG. As far as aesthetics goes, one approach would be to take an image and add a border and some box-shadow for a cleaner look. Needless to say, resizing may be necessary. But tread carefully! It is worth noting that using CSS to resize large photos often take longer to download than necessary. One quick workaround would be to download and resize photos using an image editing tool like Pic Monkey, then adding the image into your project folder.
5. Place each image into a div and position it on the map according to the region of the world they are found in. Remember, they must not move out of place if the window is resized. Troubleshooting may be required. ;)
6. When you’re finished, add labels to each species. Include their common name (e.g. “Siberian Tiger”) as well as their scientific name (e.g. Panthera tigris altaica). You can optionally include some statistics.
7. Style up the details! Make it look pretty :)

Top 5 World Destinations

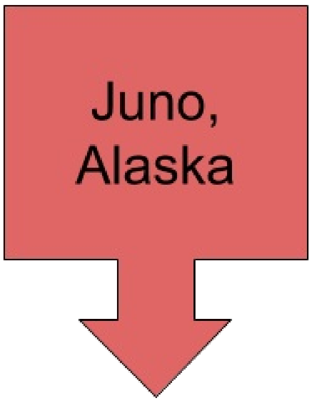
1. Start by making a list of you and your partner’s top 5 destinations in the world you’d each wish to visit. In other words, two separate top 5 lists. It’s fine if some choices are the same.

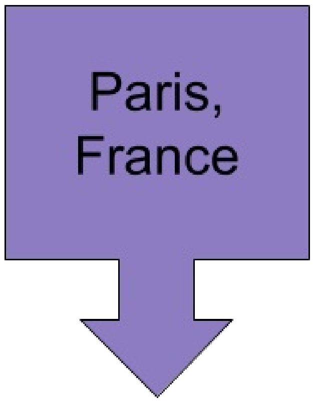
2. On a new webpage, set up a div with a class of “map” and place a blank world map image into it. I recommend using this world map example, but you’re welcome to find another.
3. Create a simple key that designates a color code for each of you. Include a third color that represents cities you and your partner both wish to visit, if any. If none exist, exclude it. Lay the key over the map somewhere so that it stays put no matter where users scroll. Have it look something like this (using the same colors, for now):

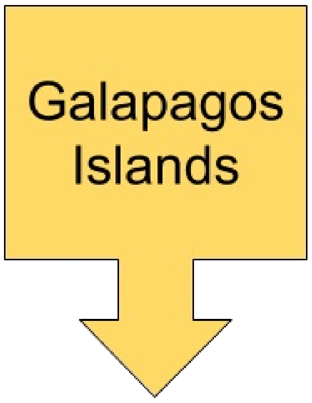


4. Using the positioning skills you’ve learned from yesterday’s homework, place markers on the map for each of your respective dream destinations. They must stay in position even if the window is resized. I’ve provided stock marker images for you to use if you wish (you’re welcome to customize your own using icons you find online), but you’ll still need to place text over them yourself. Here are the links (Right-click + Copy Link Address), followed by examples:

   - red
   - purple
   - yellow







When you’re finished precisely positioning all your markers, go ahead and style the rest of the webpage. Add a fixed masthead or overlay a heading so as to let people know what the map is for.

Floats - what's the big deal?

As you may have noticed, extended use of `float` requires a lot of hacking around and can be quite cumbersome. There are newer, better tools to meet our layout needs (which we'll get to later), but considering their prevalence, developers are still expected to have a strong grasp of floats – despite their on-going disdain for them.

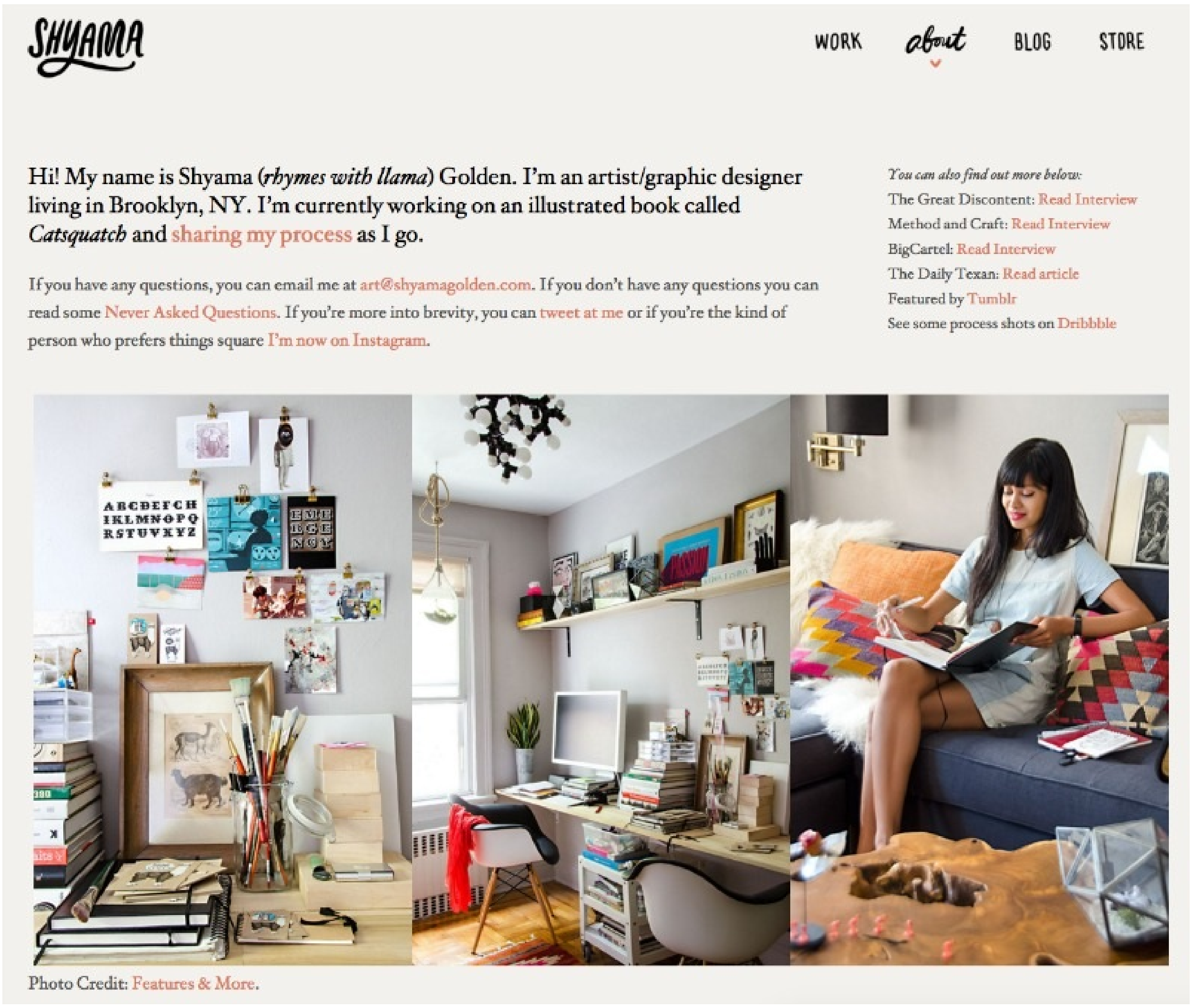
Not sure what the big deal is? Take some time tonight to read this. Also compliment the article with this fantastic (and quick) overview of clearfixes – the “hack” developers use to get around the collapsing of elements caused by floats.

About me and floats

Today you are going to use only float for layout. No Bootstrap columns, no display: inline-block. It will feel clumsy at times but, interestingly, websites have been (and in many cases, continue to be) made this way for years. Consider this a history lesson.

Check out the following awesome About Me pages:

Shyama Golden



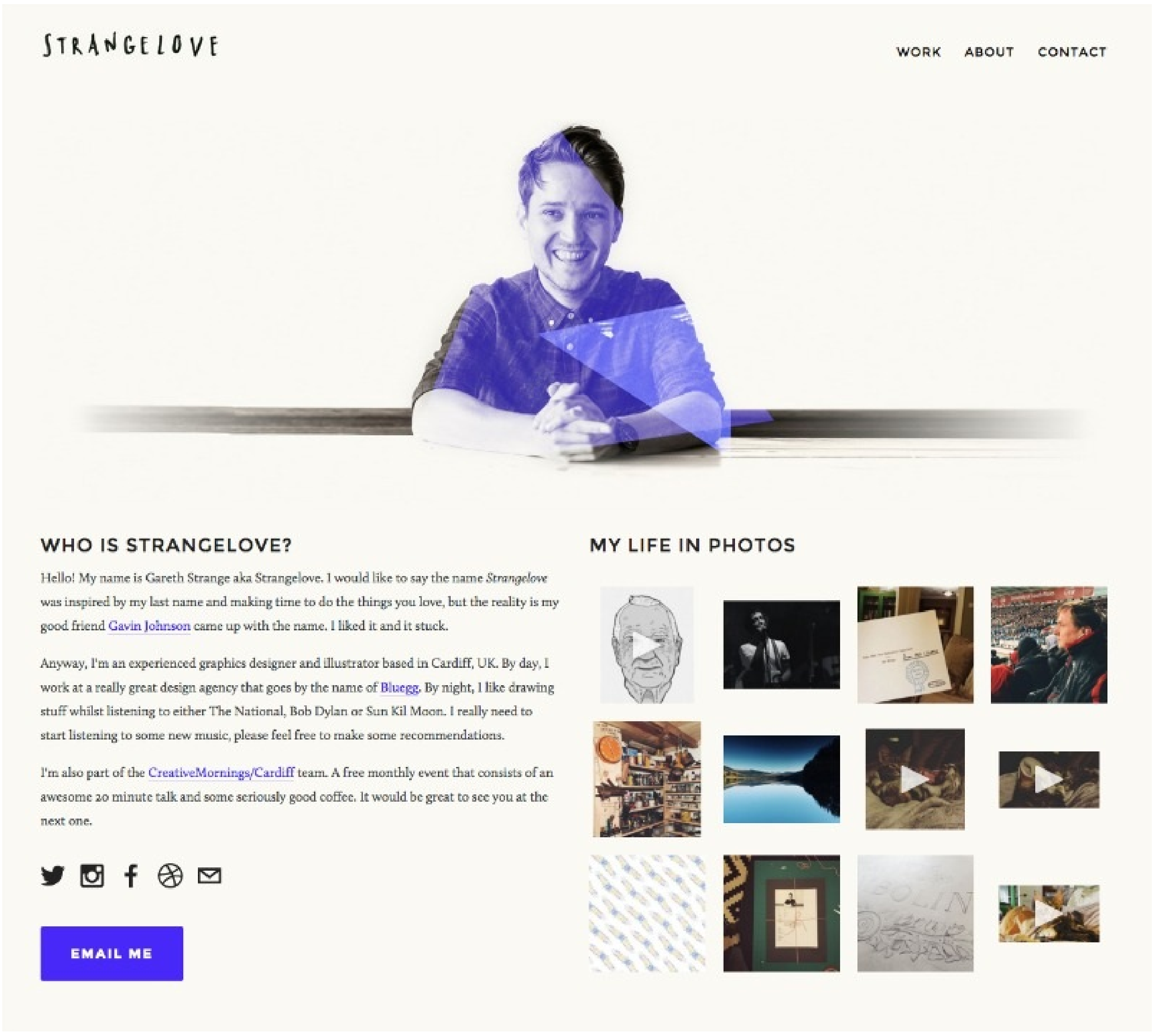
Hi! My name is Shyama (*rhymes with llama*) Golden. I'm an artist/graphic designer living in Brooklyn, NY. I'm currently working on an illustrated book called *Catsquatch* and sharing my process as I go.

If you have any questions, you can email me at art@shyamagolden.com. If you don't have any questions you can read some Never Asked Questions. If you're more into brevity, you can tweet at me or if you're the kind of person who prefers things square I'm now on Instagram.

You can also find out more below:
The Great Discontent: Read Interview
Method and Craft: Read Interview
BigCartel: Read Interview
The Daily Texan: Read article
Featured by Tumblr
See some process shots on Dribbble

Photo Credit: Features & More.

Gareth Strange



WHO IS STRANGELOVE?

Hello! My name is Gareth Strange aka Strangelove. I would like to say the name *Strangelove* was inspired by my last name and making time to do the things you love, but the reality is my good friend Gavin Johnson came up with the name. I liked it and it stuck.

Anyway, I'm an experienced graphics designer and illustrator based in Cardiff, UK. By day, I work at a really great design agency that goes by the name of Bluegg. By night, I like drawing stuff whilst listening to either The National, Bob Dylan or Sun Kil Moon. I really need to start listening to some new music, please feel free to make some recommendations.

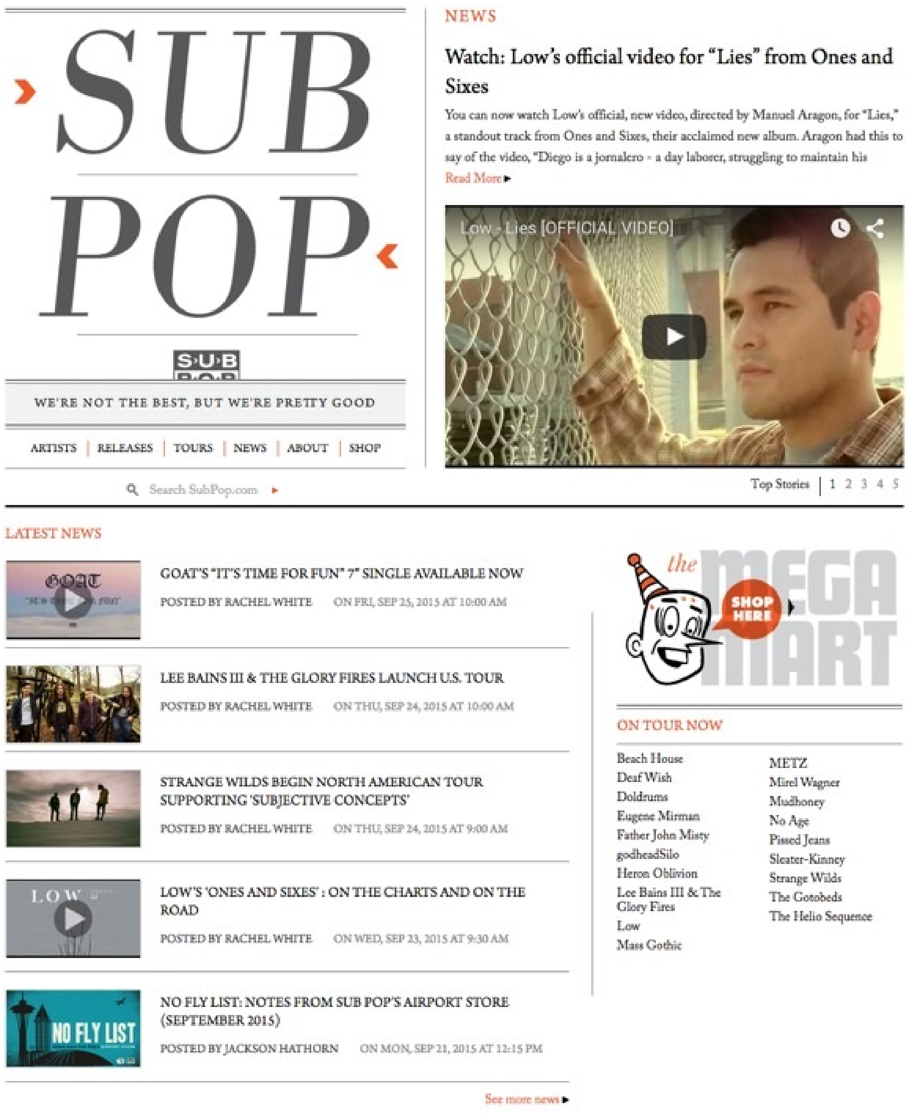
I'm also part of the CreativeMornings/Cardiff team. A free monthly event that consists of an awesome 20 minute talk and some seriously good coffee. It would be great to see you at the next one.

Choose one and make a page of your own that looks as close to the example as possible.

Needless to say, use or find your own assets (images, icons, etc.)! Get creative!

If you find yourself finishing early, try taking the following on for size:

Sub Pop Records


SUB POP
WE'RE NOT THE BEST, BUT WE'RE PRETTY GOOD
ARTISTS | RELEASES | TOURS | NEWS | ABOUT | SHOP
Search SubPop.com
Low - Lies [OFFICIAL VIDEO]
Top Stories | 1 2 3 4 5
the MEGA MART
SHOP HERE

NEWS

Watch: Low's official video for "Lies" from Ones and Sixes

You can now watch Low's official, new video, directed by Manuel Aragon, for "Lies," a standout track from Ones and Sixes, their acclaimed new album. Aragon had this to say of the video, "Diego is a jornalero - a day laborer, struggling to maintain his

Read More ▸

LATEST NEWS

GOAT'S "IT'S TIME FOR FUN" 7" SINGLE AVAILABLE NOW
POSTED BY RACHEL WHITE ON FRI, SEP 25, 2015 AT 10:00 AM

LEE BAINS III & THE GLORY FIRES LAUNCH U.S. TOUR
POSTED BY RACHEL WHITE ON THU, SEP 24, 2015 AT 10:00 AM

STRANGE WILDS BEGIN NORTH AMERICAN TOUR SUPPORTING 'SUBJECTIVE CONCEPTS'
POSTED BY RACHEL WHITE ON THU, SEP 24, 2015 AT 9:00 AM

LOW'S 'ONES AND SIXES' : ON THE CHARTS AND ON THE ROAD
POSTED BY RACHEL WHITE ON WED, SEP 23, 2015 AT 9:30 AM

NO FLY LIST: NOTES FROM SUB POP'S AIRPORT STORE (SEPTEMBER 2015)
POSTED BY JACKSON HATHORN ON MON, SEP 21, 2015 AT 12:15 PM

See more news ▸

ON TOUR NOW

Beach House
Deaf Wish
Doldrums
Eugene Mirman
Father John Misty
godheadSilo
Heron Oblivion
Lee Bains III & The Glory Fires
Low
Mass Gothic
METZ
Mirel Wagner
Mudhoney
No Age
Pissed Jeans
Sleater-Kinney
Strange Wilds
The Gotobeds
The Helio Sequence

Adjusting layouts with media queries

So far we have only been building sites optimized for large screens. In today's world, however, smartphones have become the most popular way to browse the internet. As such, you will need to learn how to build for smaller screens as well as big ones, and everything in between. Media queries are a good place to start. Watch this video and this one to learn how to add media queries to your project.

Additional Resources: http://htmldog.com/guides/css/advanced/mediaqueries/

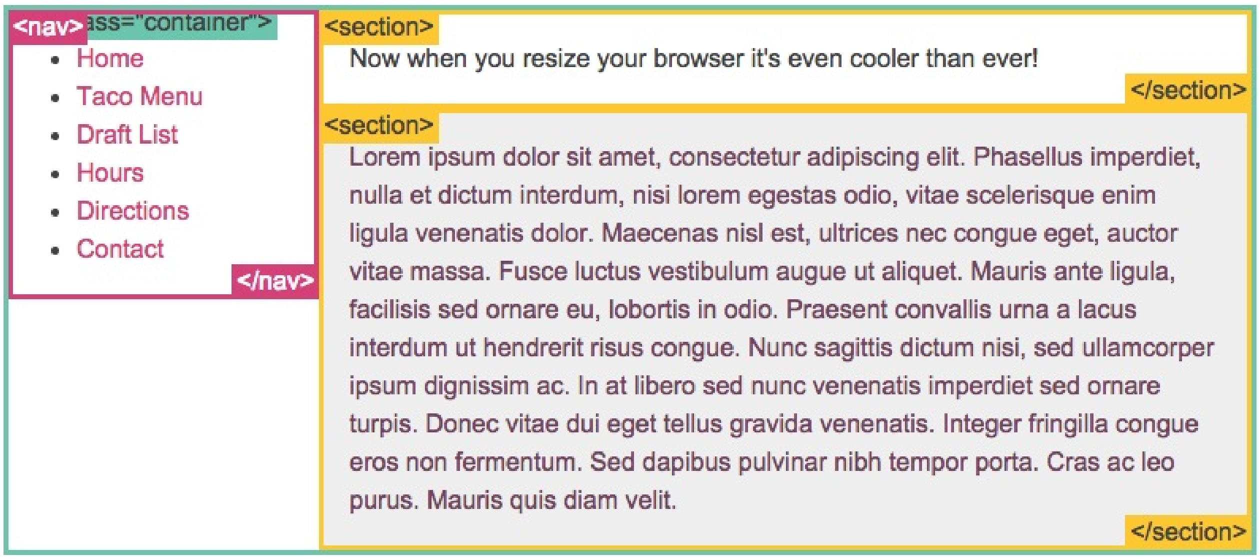
Summary

Media queries in action:

```
/***************************
Exhibit A
***************************/
@media screen and (min-width:600px) {
  nav {
    float: left;
    width: 25%;
  }
  section {
    margin-left: 25%;
  }
}

/***************************
Exhibit B
***************************/
@media screen and (max-width:599px) {
  nav li {
    display: inline;
  }
}
```

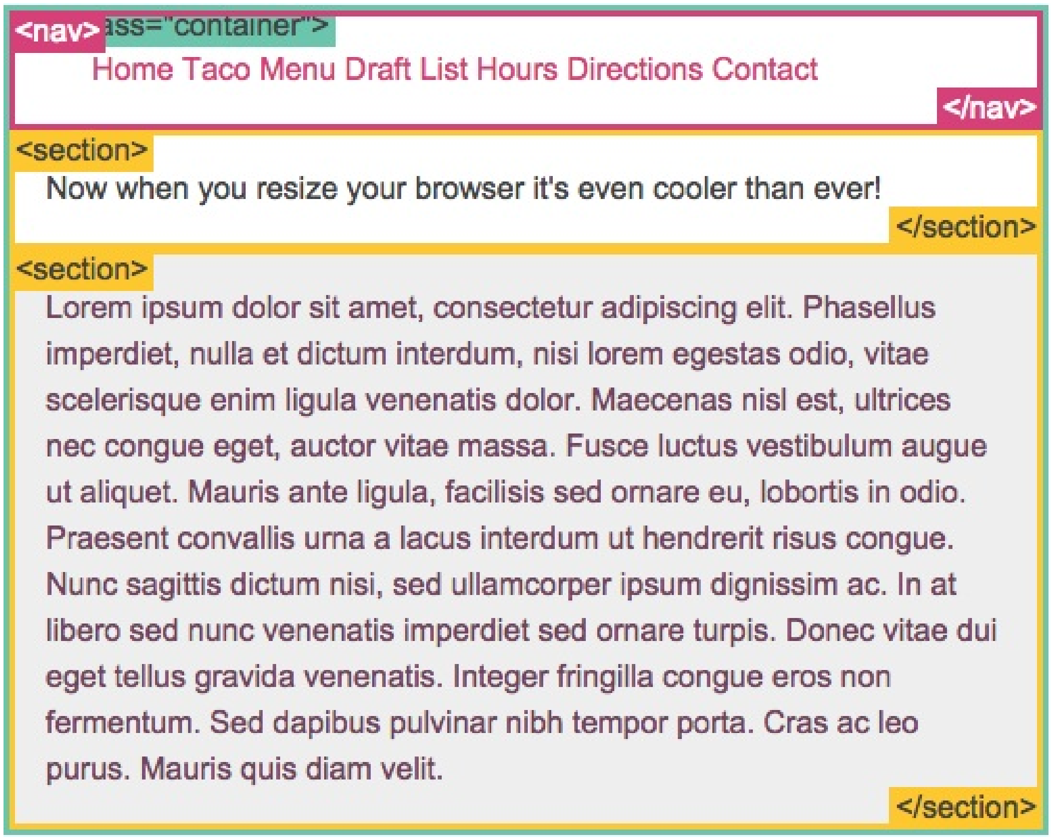
Exhibit A:



Now when you resize your browser it's even cooler than ever!

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.

Exhibit B:



Now when you resize your browser it's even cooler than ever!

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.

Going responsive

Today you are going to be adding media queries to the projects you worked on yesterday. You are encouraged to partner with the same person, but it’s not required. This project will definitely take some problem-solving. Refer to mediaqueri.es for some great examples of page structure and responsiveness.

Also, you can use Chrome’s Device Mode & Mobile Emulation to preview what your page will look like on different screens. You can access it quickly by typing `CMD` + `Shift` + `i` and then clicking the Toggle device mode *toggle device mode* icon at the top left.

Once you’ve completed at least two full websites, watch this video on the mobile first approach. You’ll have a couple options for what to do next: either refactor one of the websites you just added media queries to, or move on and create a *different* website with the **mobile first** approach. The choice is yours.

Using pre-built grid systems and frameworks

As we have said before, developers are lazy. Thankfully there are tools out there that do all the floating, clearfixing, and media querying for us. In addition, **grid systems** help us structurally lay elements out on a page based on simple math. One framework that does all of the above and more is Twitter's Bootstrap -- which you should already be familiar with. If you're feeling a little rusty on how to get started with it, you should go back and learn the basics. Additionally, take extra time tonight to review how Bootstrap's grid system works. The most important takeaway for this lesson is how to use pre-built grids.

As an aside, keep in mind that Bootstrap, like most other frameworks, does have its pros and cons. While its grid system is powerful and responsive, the framework itself comes packaged with a lot of fluff -- pre-styled components like navigation bars, menus, buttons, and more that you may or may not need in your project folder. While this is welcome in some cases where you're actually using these components, it is important that front-end developers know how to style elements on their own -- which is why, for most of our assignments in this course, Bootstrap is prohibited.

In any case, feel free to ignore (or customize) Bootstrap's plug-n-play components tomorrow.

Summary

Example of Bootstrap's 12-column grid system in action:

```
<div class="container">
  <div class="row">
    <div class="col-md-4">
      I'm a column! Perhaps I'm a right sidebar.
    </div>
    <div class="col-md-8">
      I'm another column! Perhaps I'm the main body text.
    </div>
  </div>
</div>
```

Framework field day

Today we are going to dedicate some in-depth practice to using Bootstrap. After that, we are going to take what we already know about grid systems and experiment with other popular front-end frameworks.

Now if you're wondering, "why should I care about learning other frameworks if I already know Bootstrap?", the answer is simple. Bootstrap may be the most popular framework, but there are many more out there that are just as powerful. It really just depends on the job and the dev team.

Whatever the case, it's important that you know how to transfer over your Bootstrap skills if and when the time comes you're asked to use something else.

Framework field day

Using Bootstrap, create a page replica of gliffy and/or dealerspike.com. Today, make it a point to walkthrough all the motions with your partner: how to lay out with columns and rows for various screen sizes, how to use navigation bars and carousels, experiment with various components, etc. Make sure you are taking turns at even intervals.

When you're done, you can either build another site that uses Bootstrap, or, if you're feeling adventurous, you can try your hand at a different framework below:

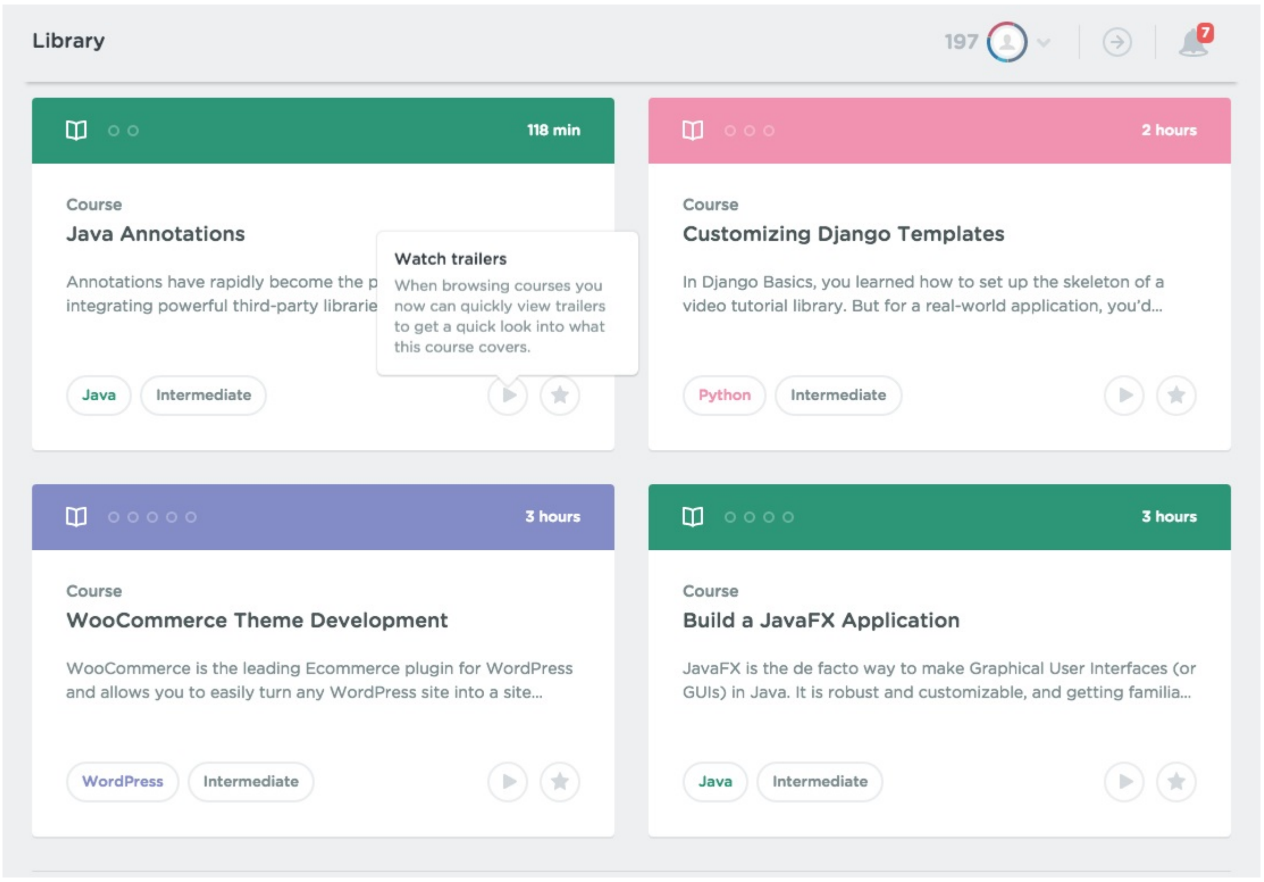
- Foundation - A popular and equally (if not more) powerful alternative to Bootstrap. Check out who's using it.
- Pure - Yahoo's take on a fast, minimal UI framework.
- Materialize - Materialize is based on Google's new material design guidelines.
- Skeleton - It's not so much a framework -- it's a "boilerplate". Skeleton encourages you to customize its components so they look unique to every project.

Basic Layout Code Review

Your Project

Treehouse Library

Style a page that looks something like the following:



118 min

Course
Java Annotations

Annotations have rapidly become the p
integrating powerful third-party librarie

Watch trailers
When browsing courses you now can quickly view trailers to get a quick look into what this course covers.

2 hours

Course
Customizing Django Templates

In Django Basics, you learned how to set up the skeleton of a video tutorial library. But for a real-world application, you'd...

3 hours

Course
WooCommerce Theme Development

WooCommerce is the leading Ecommerce plugin for WordPress and allows you to easily turn any WordPress site into a site...

3 hours

Course
Build a JavaFX Application

JavaFX is the de facto way to make Graphical User Interfaces (or GUIs) in Java. It is robust and customizable, and getting familia...

Important: the ‘Watch Trailers’ box hanging over the Java Annotations box must be included on your page.

You can use this:

(Right-click, Save Image As...)

Be sure the following objectives are met:

- Use `float` to lay out nav elements, Courses, and appropriate children elements.
- ‘Watch Trailers’ element is properly positioned using `absolute` & `relative` positioning.
- Feel free to use Google Search for icons and other assets. They don't need to be exact.

When you have reviewed your code for all of the objectives, submit a link to your GitHub repository under the CSS: Layouts, Pt. 1 Code Review on Epicenter.

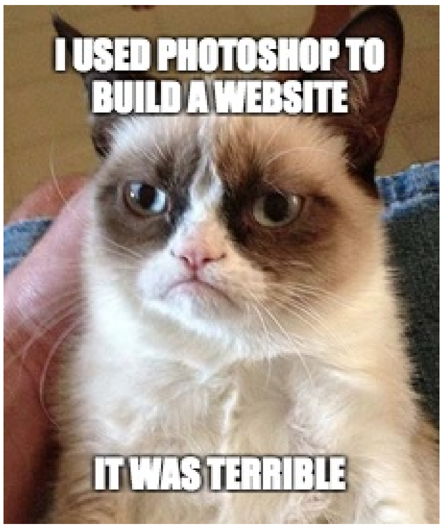
Grumpy Cat, Landing Pages, Event Flyer

Today we’re going to be diving right in.

Your objective is to practice the many ways to work with color, text, images, the box model, and basic positioning. Attention to detail will be key.

Make Your Own Meme

Create a fun "meme" from scratch.

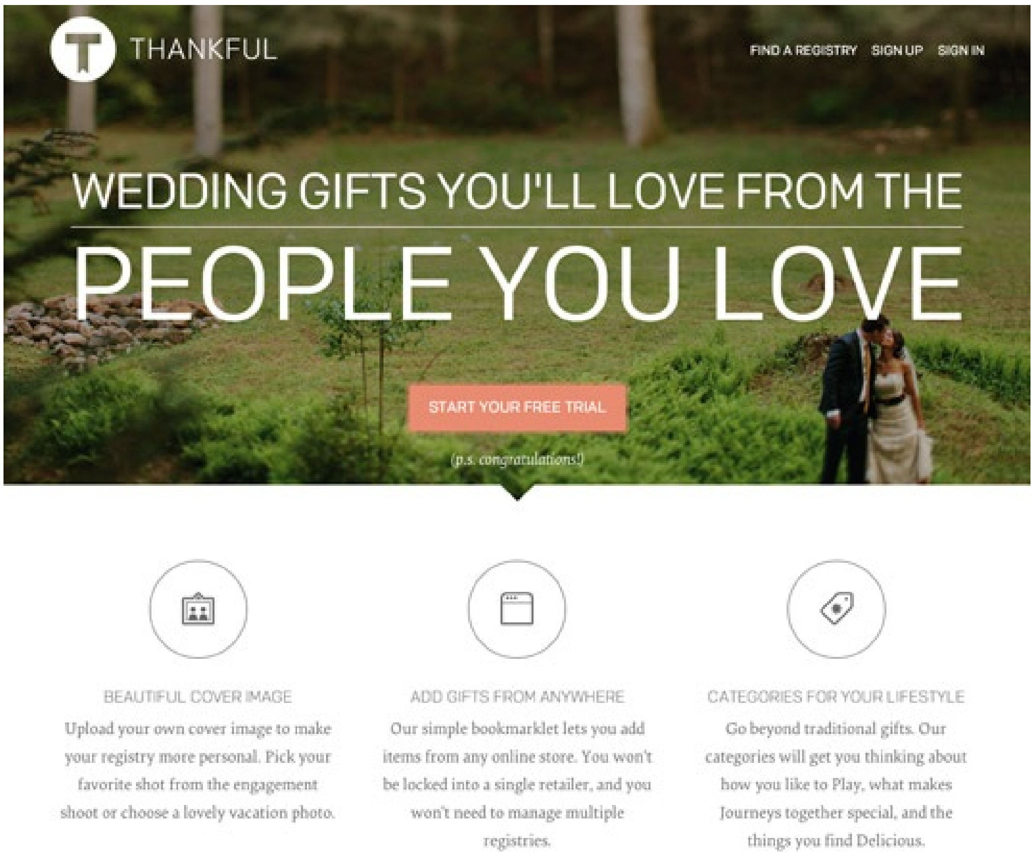


Instructions/Tips:

- Mind your spacing and positioning. Text at top and bottom should be centered.
- You will need to find out what fonts to implement, and how.
- Experiment with the text-shadow property. Make it look legit!

Landing Pages

Choose two of the following three page examples and translate them into your own HTML & CSS (do not use Bootstrap):



WEDDING GIFTS YOU'LL LOVE FROM THE

PEOPLE YOU LOVE

(p.s. congratulations!)

BEAUTIFUL COVER IMAGE

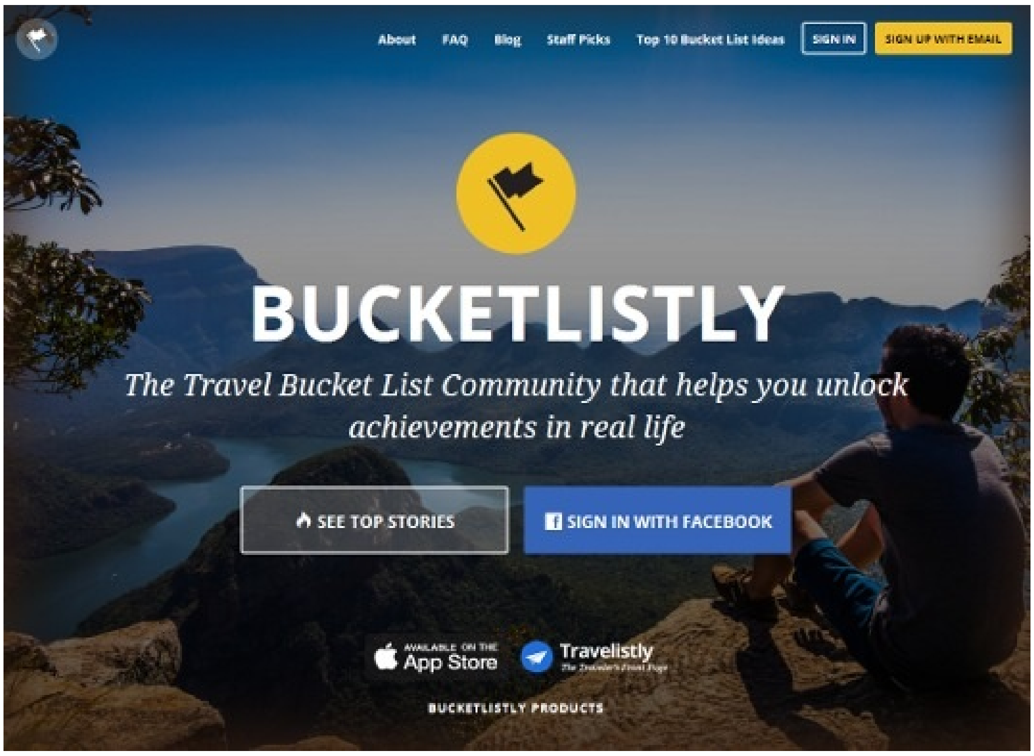
Upload your own cover image to make your registry more personal. Pick your favorite shot from the engagement shoot or choose a lovely vacation photo.

ADD GIFTS FROM ANYWHERE

Our simple bookmarklet lets you add items from any online store. You won't be locked into a single retailer, and you won't need to manage multiple registries.

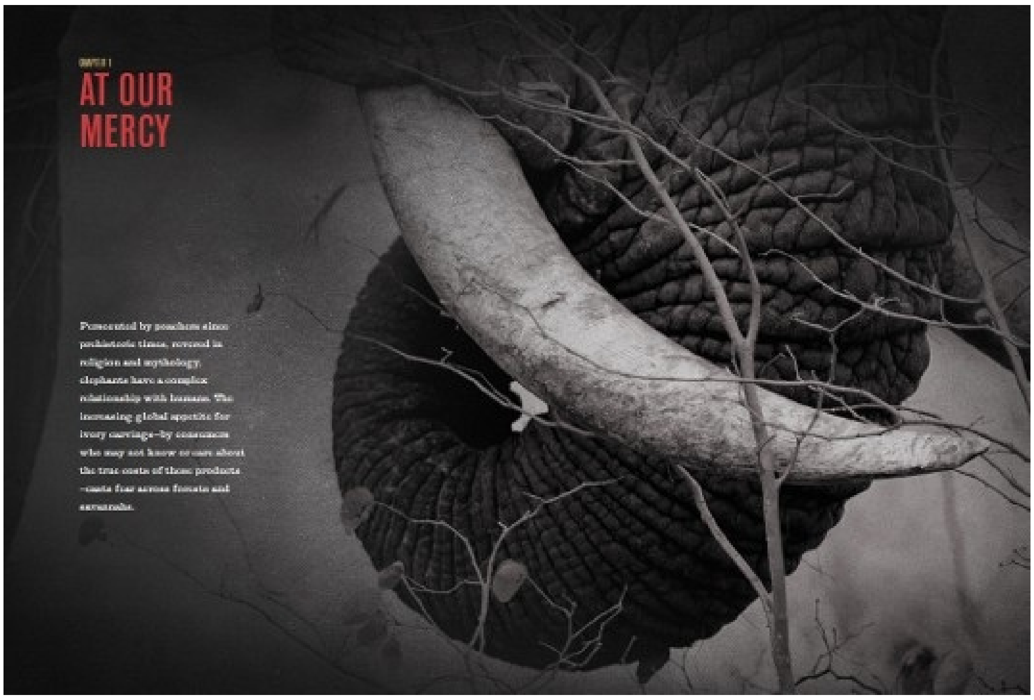
CATEGORIES FOR YOUR LIFESTYLE

Go beyond traditional gifts. Our categories will get you thinking about how you like to Play, what makes Journeys together special, and the things you find Delicious.



BUCKETLISTLY

The Travel Bucket List Community that helps you unlock achievements in real life



Persecuted by poachers since prehistoric times, revered in religion and mythology, elephants have a complex relationship with humans. The increasing global appetite for ivory carvings—by consumers who may not know or care about the true costs of those products—casts fear across forests and savannahs.

Instructions/Tips:

- For your links, use “#” instead of URLs for your HTML mark-up. (i.e. `<a href=”#”>Dummy Link</a>`)
- I recommend checking out StockSnap.io if you’re having trouble finding good free images of your own. Mind your HTML! Is it semantic?
- Don’t worry about locating the exact fonts, styles, and colors. Feel free to get creative.
- This isn’t a race. Challenge yourself to be as true to detail as can be. Translating page mock-ups into code is one of the primary duties of real front-end developers.
- Make sure both you and your partner are taking turns at fair intervals.

Event Flyer

Create your own event flyer. See this and this if you need some inspiration.

Who hires Epicodus graduates?

As you learned from the homework, It is common to use image sprites in web apps wherever multiple icons or logos are used. Packaging icons into a single image file is much more efficient than including a separate .png for each individual icon.

Today we are going to take a very common use case and put our theory into practice.

- Build a copy of the Who Hires Epicodus Graduates section on the Epicodus website. Refer to the homework!

- Use SpritePad. (Hint: save the images locally by right-clicking each and selecting `Save Image as...', then simply drag-and-drop into the web app.)
- Bootstrap is allowed today.

When you're finished, here are a few ideas to continue practicing:

- Build the rest of the page, including header/footer, mid sections, etc. Use a sprite for those social media links!
- Implement sprites into your tables from yesterday's classwork
- Brainstorm with your partner on some other cases where image sprites are useful, and then build it. Feel free to explore a bit.
- Consider incorporating this technique into past projects, or your portfolios.

JavaScript Intermediate

JavaScript BDD Objectives

In this section, we will be practicing behavior driven development in JavaScript using Mocha and Chai as our testing tools. At the end of the week, you should be able to:

- write thorough specs that test the behaviors you have identified
- build Javascript functions that use branching and looping
- create HTML pages that use jQuery to manipulate and traverse the DOM

The code review at the end of the week will be reviewed for the following objectives:

- Complete spec coverage
- All specs are passing (indicating that your Javascript function(s) * are working as designed if you have thorough specs in place)
- Program logic is separate from user interface logic
- Variable names are descriptive and use lower camel case (likeThis)
- Code has proper indentation and spacing

Behavior-driven development (BDD)

> **Embedded Video: https://www.youtube.com/watch?v=sM2wPoxSmhk?rel=0**

One of the most difficult challenges facing us as developers is taking a problem we want to solve or a concept we want to realize and turning it into a set of specific programming tasks.

A common process to do this is called **Behavior-Driven Development** (or BDD) and is used by developers across coding languages. In BDD, rather than thinking about the code first, the focus begins on the behaviors that we want to see in our final application. We identify what the program should do before determining how to make it do it.

To demonstrate developing an application using BDD, we'll imagine that we have been hired by a person born on February 29th. She would like to determine if any given year is a leap year (meaning it's a birthday year for her!). Here's a finished example of what she'd like: Leap year detector.

Before we think about the programmatic elements, what should a leap year program do? At its most basic, it will need to be able to take a year from the user and answer true or false to the question: is this year a leap year? Our program will only be able to provide an answer once it successfully evaluates what the user provides as a year. Let's think of all of the possibilities we might get from a user and what the correct response should be for them.

Specifications

Let's consider how we can evaluate if a leap year is a leap year. Timeanddate.com offers us the 3 criteria that must be considered to determine if a year is a leap year:

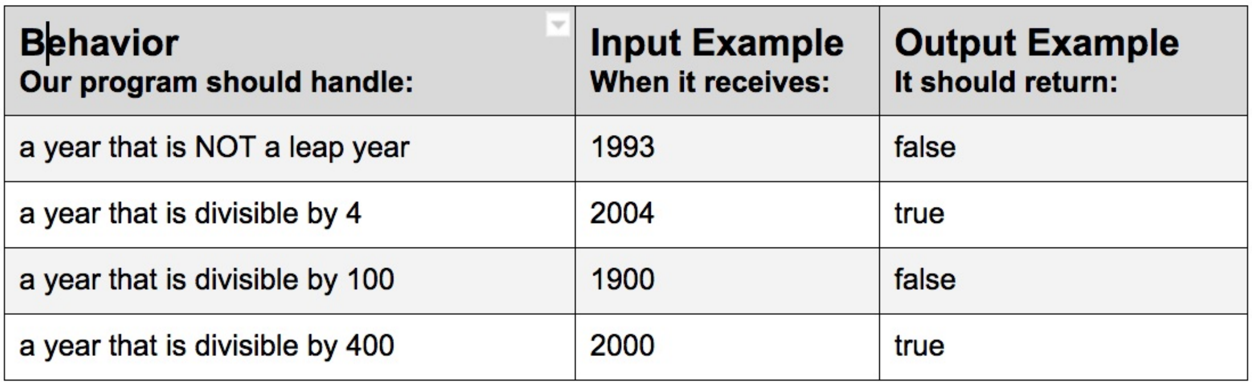
- The year is evenly divisible by 4;
- If the year can be evenly divided by 100, it is NOT a leap year, unless;
- The year is also evenly divisible by 400. Then it is a leap year.

Therefore, each time a user offers a year to evaluate, we will ultimately need to test the value against each of the leap year rules.

In BDD, our next step is to generate examples of these rules one-by-one. These examples are also known as **specifications** or **specs**. We can create a table that helps us sort out the details of the specifications for each rule using the following pieces of information:

- the behavior that we'll need to write code to handle
- a sample of input that would demonstrate the behavior

- the expected output we'd get when the code is working correctly



Behavior **Our program should handle:**	**Input Example** **When it receives:**	**Output Example** **It should return:**
a year that is NOT a leap year	1993	false
a year that is divisible by 4	2004	true
a year that is divisible by 100	1900	false
a year that is divisible by 400	2000	true

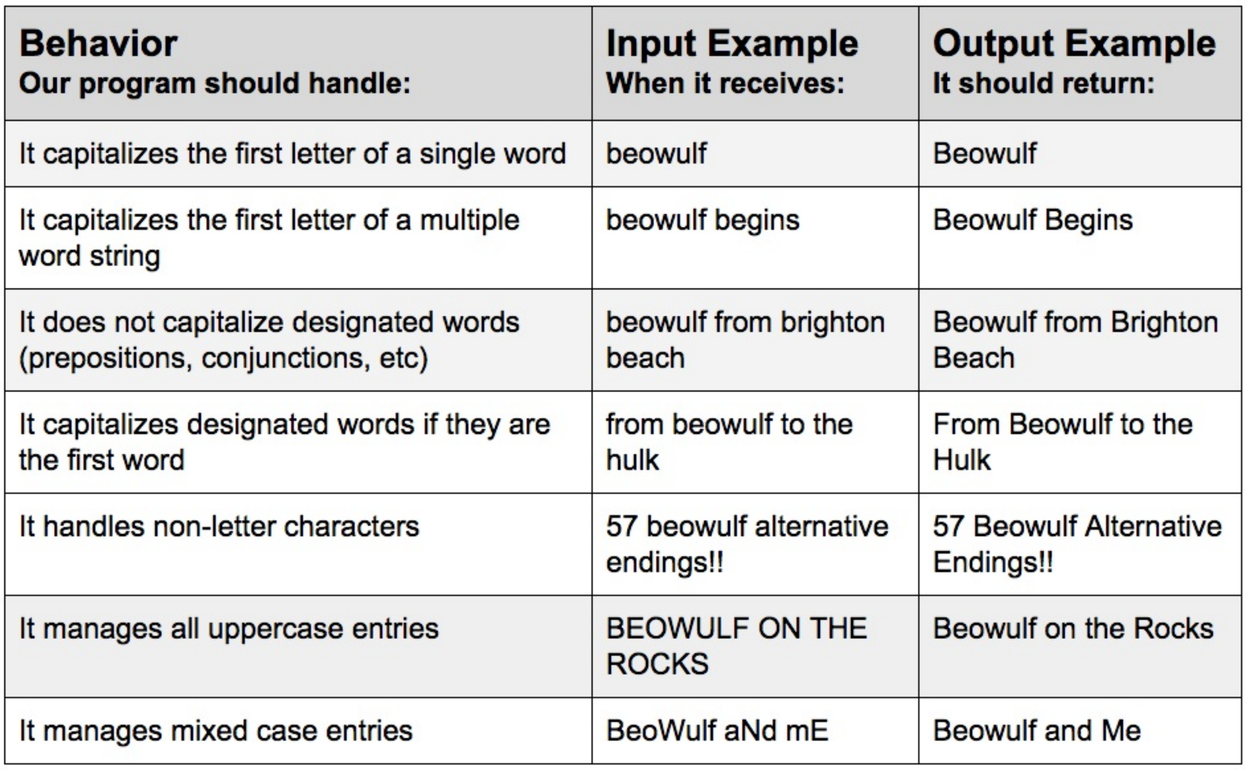
When we have finished our first round of brainstorming for the behaviors our program should incorporate, we are ready to determine how to write our first coded specification test using the BDD tools of the language our program will be written in.

Although there are many other considerations for our final application (display, user interaction, form building, what if someone enters a letter?, etc), we will not worry about those until we have the core functionality in place. If we think of any additional functionality we need, we can add behaviors to our specification list as we go. When you code using BDD, it is good to get into the habit of making a note of all behaviors as you think of them, but staying focused on one test and one task at a time.

Let's look at another example of specs organized on a table showing behavior, input and output:

Title Case

In this example, we want to build an application that will take a user's string of words and convert them to title case - capitalizing letters like we'd find in a book title. There are a few more rules for creating title cased words from strings than Leap Year. Let's brainstorm the first several:

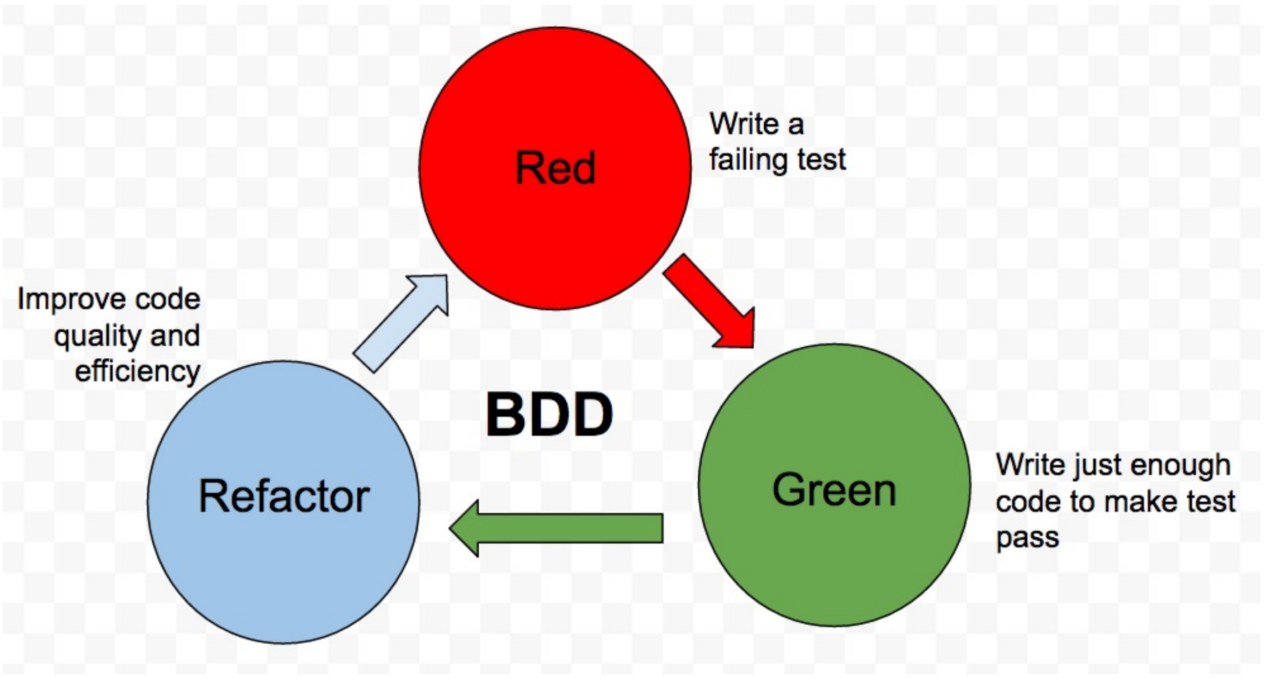


Behavior **Our program should handle:**	**Input Example** **When it receives:**	**Output Example** **It should return:**
It capitalizes the first letter of a single word	beowulf	Beowulf
It capitalizes the first letter of a multiple word string	beowulf begins	Beowulf Begins
It does not capitalize designated words (prepositions, conjunctions, etc)	beowulf from brighton beach	Beowulf from Brighton Beach
It capitalizes designated words if they are the first word	from beowulf to the hulk	From Beowulf to the Hulk
It handles non-letter characters	57 beowulf alternative endings!!	57 Beowulf Alternative Endings!!
It manages all uppercase entries	BEOWULF ON THE ROCKS	Beowulf on the Rocks
It manages mixed case entries	BeoWulf aNd mE	Beowulf and Me

As we did with the leap year application, we choose the simplest first - one word gets capitalized - `beowulf` becomes `Beowulf` and go to the most complex. And as always, we may think of additional behaviors along the way: what if a user enters nothing? what about iPod or McDuff? Remember, let your brain keep brainstorming by adding new behaviors to your list but always stay focused on one specification at a time.

Red-Green-Refactor

After the initial brainstorming of behaviors, we're ready to get to coding. Every programming language has testing tools for BDD. In fact, most have more than one flavor to choose from. Testing tools act like a user by running our code for us the same way that we run our code manually to ensure that it is working. With testing tools, if the behavior we are testing doesn't result in the expected outcome, the tools will offer us insight as to what we may need to fix. In this lesson, we are going to explore the process that is followed for BDD testing, coding and improving that is the same in every language. We call this the **red-green-refactor** cycle.


Red
Write a failing test
Green
Write just enough code to make test pass
Refactor
Improve code quality and efficiency
BDD

In BDD, the first code written is always the test code. This is the **red** phase because when we run our test, we will get a **red**, meaning the test has failed. Since we have not written ANY code to make it pass, we expect the **red** or failing indication each time we write a new specification.

Let's use our leap year table to translate some behaviors into coded specifications. How do we decide which example should be written first? This can be tricky for beginning developers. Good advice is to think of the simplest behavior that our application must have. In the leap year application, we are going to have all kinds of conditions which will determine the outcome (divisible by 4, divisible by 100, divisible by 400). If a year is entered that doesn't meet ANY of these conditions, `false` will be returned. That is the most simple behavior requiring the least analysis. In this case like many you'll find, the simplest behavior is what will ultimately be our `else` statement after all of the conditions are NOT met

Looking at the behavior table, we chose `1993` as an example of a year that doesn't meet ANY of the leap year criteria. It's important to choose a value in an example that will ALWAYS work as the program evolves. Even after we add each condition, `1993` will always be `false`.

Red

The way a BDD test is coded will look different based on the language and the tool that is used but will always include:

- a description of the behavior ("Given")
- how it is triggered ("When")
- what the expected result should be ("Then").

In a non-language specific example (see the cheat sheet for this lesson for language-specific tests written in Java, PHP, Ruby and C#), a BDD test looks like this:

Given the year 1993 **When** the leap_year program is executed **Then** false will be expected

This spec will test that when our program runs, an input of 1993 should result in an output of *false*.

A specification when run in any language will actually execute the code it is instructed to. This would run the `leap_year` program with the input "1993" to analyze. It will then take whatever response it receives and compare it to what is expected. In this case, the test will only pass (**green**) if the returned value is false. For any other response, the test will fail(**red**).

Our `leap_year` code does not exist at this point, so when the spec is executed, we will see it fail(**red**) with an error message indicating that the code we tried to call does not exist.

Green

The next step is to write the least amount of code to make the test pass. This keeps our coding focused on this one, small, narrow programming task.

To make this test pass, we don't need to write any of the conditional code yet. We can just return false when the program runs. That is the least amount of code to make our test pass. The program gets `1993` and returns `false`. Done. When the test passes, we have completed the **green** phase of BDD. Our code works.

This may seem odd. What about the other examples we identified? We can't return `false` for every input. Remember, we are only testing and coding one example of a year that does NOT meet leap year criteria. And we write the simplest code to make it pass which is to do no analysis and only return `false`. It may seem illogical at first, but make a habit of only coding the least necessary code to make your test pass. You will have additional tests that handle all of the other possibilities as you add functionality to your application.

Refactor

Before going back to red by creating a new failing test for the next rule/condition we want to add, we need to do a final review of our code. Is it clear? concise? efficient? free of redundancies? If improvements can be made, we **refactor** it. Just because the code is working isn't the only consideration. We want to leave our code in its cleanest, most efficient, passing form so that anyone (including our future selves!) coming back to our code can easily add to it or modify it.

Repeat

After we have refactored our code and confirmed our first test still passes, we are ready to repeat the process by creating a new failing test.

We follow the red-green-refactor process for each specification that we identified in our grid. Each time, we write the failing test (red), add the minimal amount of code to make it pass (green), and ensure that our code is the best it can be (refactor).

By doing one test at a time, we are able to stay focused on a single effort without being overwhelmed by the requirements of the whole application. With each test, we are also creating a series of automated tests that means we will not have to manually test our application over and over again as it grows. We can run all of our specs at any time to determine if our application is working as expected. Sometimes as an application grows, new code may break functionality that has already been successfully tested. When we run our series of tests, we can spot those kinds of errors immediately. Having automated testing speeds the development process along and frees the developer from going back and having to re-confirm functionality.

Refactoring- Functions Calling Functions

> **Embedded Video: https://www.youtube.com/watch?v=hhbkeOUS8GI?rel=0**

After we've finished writing our code, it's always a good idea to review it for readability and efficiency. Just because a function works does not necessarily mean it is written as well as it could be. Remember, refactoring is the final step of the BDD process after we've made our first set of specs pass.

Consider refactoring when your code exhibits:

Repetition/Redundancy

If you find yourself copying and pasting the code from one function to the next, you may be repeating functionality that may best be written in a single function (Remember DRY - Don't Repeat Yourself). There also may be redundancies when multiple developers have contributed code and have recreated some of the same functionality in their work.

Unnecessary Complexity

If you have trouble following a function or conditional statement, think about what tasks are being evaluated or performed. If it is difficult to say in words what the condition or function does without a series of "and…" statements, it may be time to break the work into more functional pieces.

Difficult Readability

If the logic is sound but difficult to follow, it will be a challenge for other developers or even yourself in the future to be able to come back and easily contribute or update the code. Being able to scan through code and easily pinpoint functionality will make it easy to maintain and evolve.

Inefficiency

We don't focus much on performance in this class but it is good to be aware that in the future, you may also need to analyze your working code for efficiency. See if you can figure out ways to make your code run faster, such as by limiting the number of times a loop runs.

JavaScript BDD code review: Level 2 Placement Test

For acceptance into Level 2 courses at Epicodus, each student is expected to be independently proficient in these areas of knowledge:

- Markdown
- HTML and CSS
- Bootstrap
- Command line
- Git
- GitHub Pages
- Behavior Driven Development
- JavaScript
- jQuery

The Level 2 Placement Test provides students an opportunity to demonstrate the necessary proficiency to begin Level 2 coursework. You may use documentation resources for the languages and concepts you use, but you should not get help from other people or look for solutions others have come up with before.

If you feel you are not yet able to independently complete all of the areas in the challenge, we encourage you to work through the Epicodus Intro to Programming coursework or enroll in our Intro to Programming course.

To proceed with the test, carefully read all of the requirements for coding and submission. You will only have the opportunity to submit your work for placement once.

Placement Test Instructions

Create a web application that takes a positive number from a user and returns a range of numbers from 1 to the chosen number with the following exceptions:

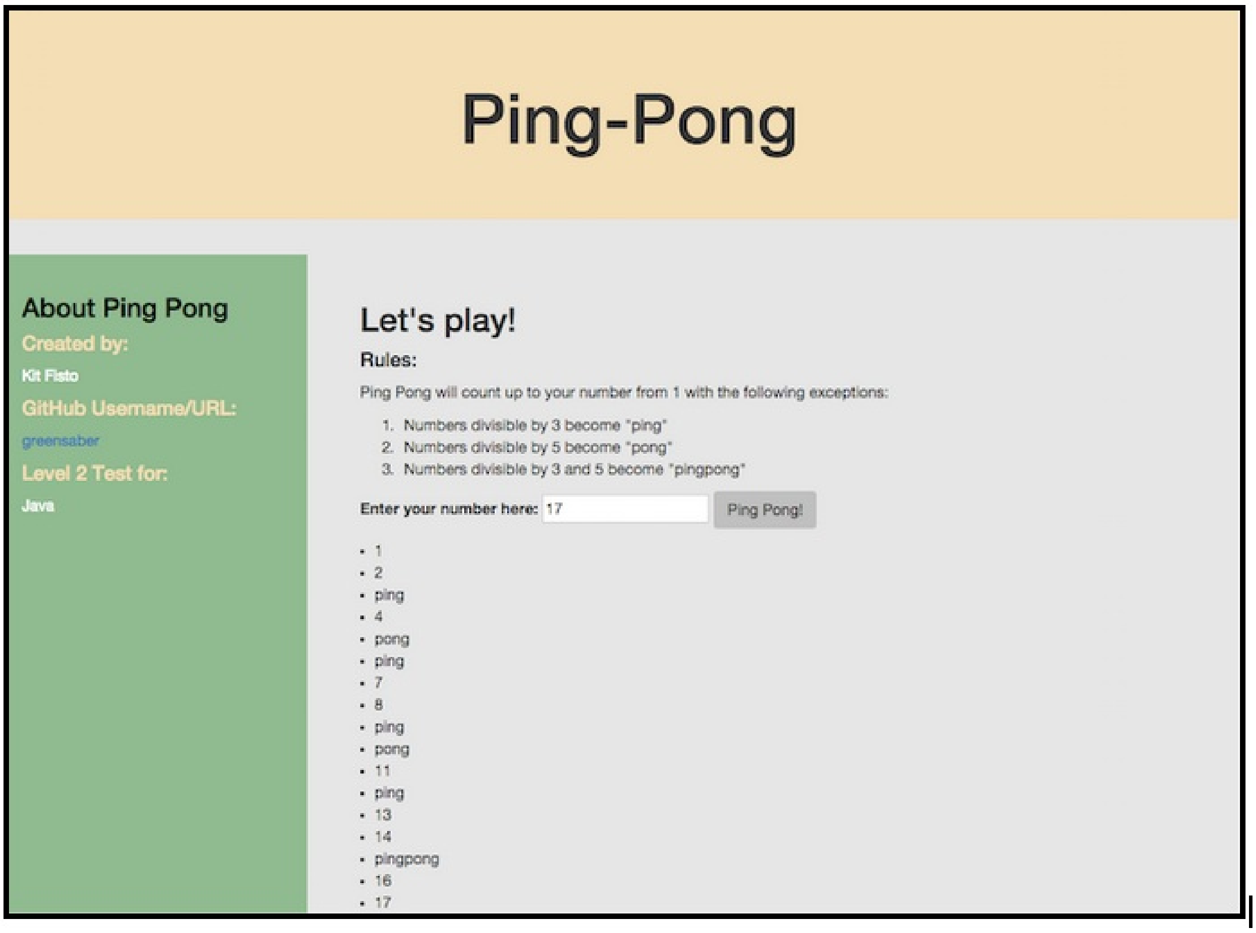
- Numbers divisible by 3 are replaced with "ping"
- Numbers divisible by 5 are replaced with "pong"
- Numbers divisible by 15 are replaced with "pingpong"

Watch the video to see a live version of the expected application. Your final work should look like the examples below including the following information in the sidebar:

- your full name
- a link to your GitHub repository for this test
- the name of the course you want to be placed into

Your page and README should mirror these examples. Slight variances in color and font will be accepted but should be matched as closely as possible.

Layout and styling example



Ping-Pong

About Ping Pong
Created by:
Kit Fisto
GitHub Username/URL:
greensaber
Level 2 Test for:
Java

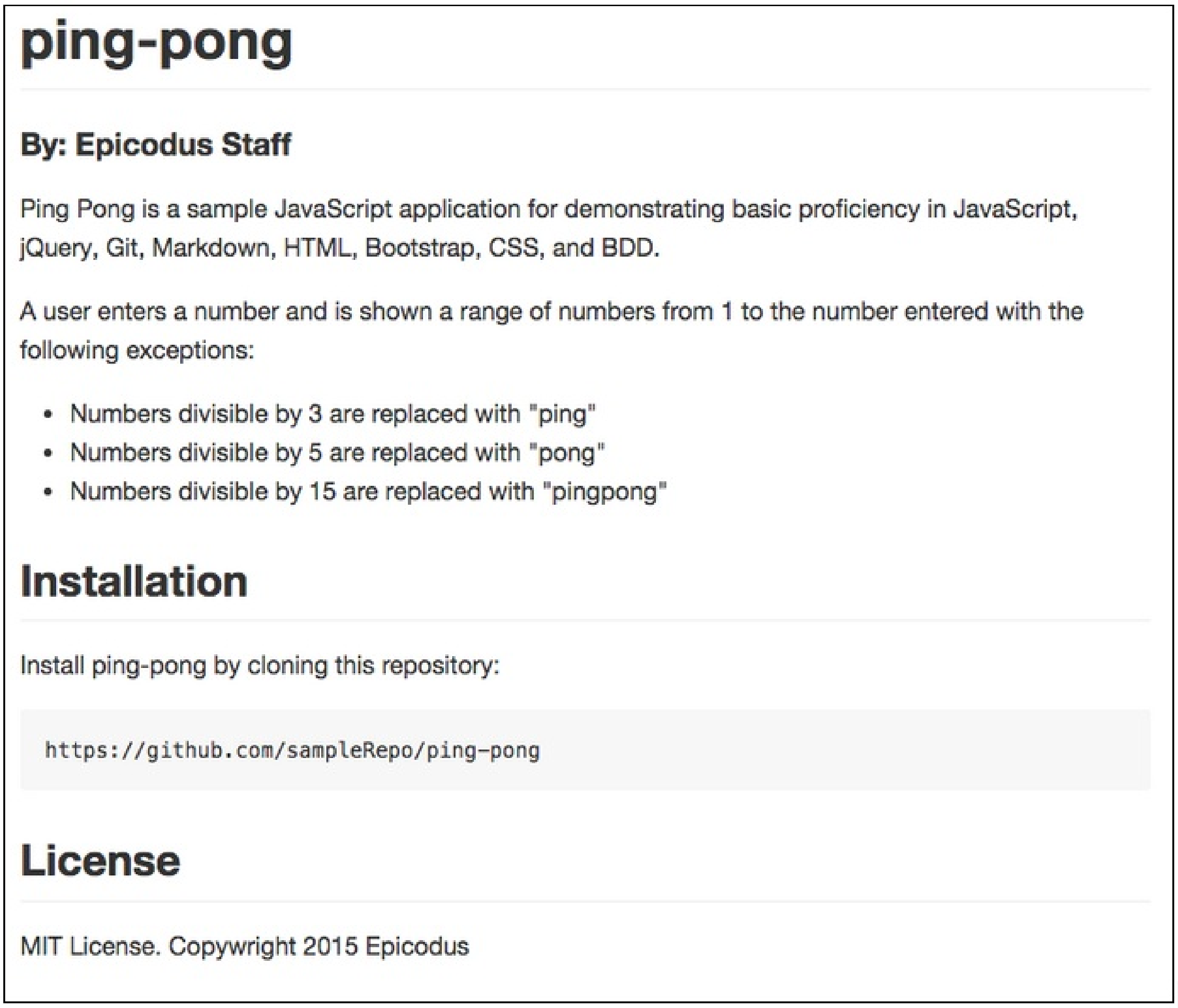
Let's play!
Rules:
Ping Pong will count up to your number from 1 with the following exceptions:

1. Numbers divisible by 3 become "ping"
2. Numbers divisible by 5 become "pong"
3. Numbers divisible by 3 and 5 become "pingpong"

Enter your number here: 17 Ping Pong!

- 1
- 2
- ping
- 4
- pong
- ping
- 7
- 8
- ping
- pong
- 11
- ping
- 13
- 14
- pingpong
- 16
- 17

README example



ping-pong

By: Epicodus Staff

Ping Pong is a sample JavaScript application for demonstrating basic proficiency in JavaScript, jQuery, Git, Markdown, HTML, Bootstrap, CSS, and BDD.

A user enters a number and is shown a range of numbers from 1 to the number entered with the following exceptions:

- Numbers divisible by 3 are replaced with "ping"
- Numbers divisible by 5 are replaced with "pong"
- Numbers divisible by 15 are replaced with "pingpong"

Installation

Install ping-pong by cloning this repository:

```
https://github.com/sampleRepo/ping-pong
```

License

MIT License. Copywright 2015 Epicodus

Requirements

Please do not add additional features or user interface functionality beyond the scope of what is requested in the instructions. Before submission, review your code to ensure it meets the following objectives. If you are unclear what is expected in an objective, go to the level 1 lessons that cover that content.

- Code has proper indentation and spacing
- Bootstrap, HTML and CSS are used to re-create the layout and styling example
- JavaScript variable names are clear, descriptive, and follow naming conventions
- JavaScript specs have been written using the Mocha spec framework and Chai assertion library to cover all user input possibilities
- All JavaScript specs pass when run
- JavaScript function(s) have been written to process user input
- jQuery is used to update the DOM after user input is processed
- Working code is stored in a GitHub repository with regular commits (Commits should

follow the standard tense completing the sentence "This commit will…")
- Markdown is used in a README file with styles and sections like the README example
- The site is successfully deployed using GitHub's GitHub Pages option Opteamize instructors will review your code and provide one of the

following responses for each of the above objectives:

- Code meets the standard ALL of the time.
- Code meets the standard MOST of the time.
- Code does NOT meet the standard.

IMPORTANT NOTE: All objectives must meet the standard ALL or MOST of the time to enroll in a Level 2 course. If an objective is NOT met, the student is enrolled in our Intro to Programming course. Additional feedback regarding performance on the Code Challenge is not provided. Because our teachers have limited time to grade these tests, no resubmissions will be allowed.

Submission

When you are confident that your code meets the objectives of the challenge:

1. Login to epicenter.epicodus.com and navigate to the Placement Test Code Review.
2. Select the Level 2 Placement Test.
3. Enter the URL for your deployed test on your GitHub Pages (e.g. "https://greensaber.github.io/ping-pong").
4. Press Submit.

Results

Your code will be reviewed within two weeks of submission. You will be notified by email with your results.

Javascript Assignments

Leap Year, Triangle Tracker, Factorial, Cryptosquare

Goal for the day: The goal for today is to get familiar with using BDD to drive your JavaScript development. Practice writing functions with clear logic, good syntax and easy-to-read code. Focus on understanding more than the number of exercises you complete.

Leap year

Here's your first assignment for JavaScript:

- Follow along with the leap year lesson, building the app with the guidance of the lesson.
- There's one final rule for leap years: if a year is divisible by 400, it is a leap year. Write a test, make it pass, then refactor your code.
- There's a bug in the code! If you enter a year that isn't a leap year, and then enter a year that is, it incorrectly tells you that that the year is not a leap year. Fix this bug.

Triangle Tracker

Make a web page that lets the user input the lengths of the sides of a triangle, and returns whether they form an equilateral, isosceles, or scalene triangle. Sound familiar? To refresh, here are the details for what makes each type of triangle:

- equilateral: all sides are equal;
- isosceles: exactly 2 sides are equal;
- scalene: no sides are equal.

Keep in mind that not all combinations of three numbers make valid triangles. For example, (2, 2, 8) does not make a triangle. The general rule is that if one side is the same or LARGER than the sum of the other two, the lengths do not form a triangle.

Factorial

A factorial is an operator that multiplies a number by all of the positive integers less than that number. For example, 5! = 5 *4* 3 *2* 1 = 120.

By definition, 0! = 1.

Make a web page to compute factorials.

Bonus points: refactor your code using recursion.

Recursion is when a method calls itself. For an additional challenge, try refactoring your function using recursion. Here's an example of how to use recursion to write the lyrics to 99 Bottles of Beer to the console:

```
var bottlesOfBeer = function(number) {
  console.log(number + " bottles of beer on the wall...");
  if (number > 0) {
    bottlesOfBeer(number - 1);
  }
};

bottlesOfCoke(99);
```

Cryptosquare

A classic method for composing secret messages is called a *square* code.

The spaces and punctuation are removed from the English text and the characters are written into a square (or rectangle) and the entire message is downcased. For example, the sentence "don't compare yourself to others, compare yourself to the person you were yesterday" is 69 characters long, so it is written into a rectangle with 8 rows and 9 columns.

```
dontcompa
reyoursel
ftoothers
compareyo
urselftot
hepersony
ouwereyes
terday
```

The coded message is obtained by reading down the columns going left to right. For example, the message above is coded as:

```
drfcu hotoe toreu enyom
spwrt oopee edcut alrra
mrhrf seyms eetoy peryo
neals otys
```

Write a program that, given an English text, outputs the encoded version of that text.

The size of the square (number of columns) should be decided by the length of the message. If the message is a length that creates a perfect square (e.g. 4, 9, 16, 25, 36, etc), use that number of columns. If the message doesn't fit neatly into a square, choose the number of columns that corresponds to the smallest square that is larger than the number of characters in the message.

For example, a message 4 characters long should use a 2 x 2 square. A message that is 81 characters long would use a square that is 9 columns wide. A message between 5 and 8 characters long should use a rectangle 3 characters wide.

Output the encoded text in groups of five letters. For example:

```
encrypt("Have a nice day. Feed the dog & chill out!");
# => "hifei acedl v..."
```

Pig Latin, Palindromes, Prime Sifting

Goal for the day: Continue practicing your development of JavaScript applications using the BDD process: red, green, REFACTOR.

Pig latin

Write a Pig Latin translator or should we say an "igPay atinLay anslatorTray".

Here are the rules of Pig Latin that you should use:

- For words that start with a vowel, add "ay" to the end.
- For words that start with one or more consonants, move all of the first consecutive consonants to the end and add "ay". (If the first consonants include "qu", move the "u" along with the "q". Don't forget about words like "squeal" where the "qu" doesn't come first!)
- For words that start with "y", treat the "y" as a consonant.

Remember to break this down into small tests and do testing and coding one behavior at a time. When you get to consonants, don't try to solve it all at once. Instead, start with an example of a word that only has one consonant; then a word with two consonants; then a word with three; and then tackle "qu" and "y".

The `.slice()` method for strings may come in handy.

Once you have it working for one word, get it working for sentences.

For further exploration: Refactor your code using regular expressions. A handy place to try using regular expressions before implementing them is at Rubular.

Palindromes

A palindrome is any word, phrase, number, or other sequence of characters which reads the same backward or forward. Create a function to identify if a word is a palindrome. It should return true if the input word is a palindrome and `false` if it is not.

Bonus points: Make your function check a string of words and also an integer. For example: "Hello olleH" is a palindrome by our definition. An integer palindrome would be: 1002001.

You may want to browse some of the built-in Javascript functionality available for strings and arrays at MDN. Look under References: Built-in objects.

Prime sifting

Given a number, write a method that returns all of the prime numbers less than that number.

This is a tricky problem, and I want you to use the Sieve of Eratosthenes to solve it. Here's how the Sieve of Eratosthenes works to find a number up to a given `number`:

- Create a list of numbers from 2 through n: 2, 3, 4, ..., `number`.
- Initially, let `prime` equal 2, the first prime number.
- Starting from `prime`, remove all multiples of `prime` from the list.
- Increment `prime` by 1.
- When you reach `number`, all the remaining numbers in the list are primes.

Now, implement this in JavaScript, using BDD to guide you. If you have time before class is over, make a website for a user to enter a number and return all of the prime numbers less than that number.

Black Jack or Meetup Dates

Goal for the day: Choose one of the following projects to work on today. Both offer you added challenges for implementing all of the JavaScript concepts that you have practiced this week. The goal is to stretch your skills and understanding whether or not you complete your chosen project today (and you may not! No worries.).

Blackjack

A popular card game at most casinos is Blackjack. The goal of the game is to beat the dealer by either reaching "21" with the cards you are dealt or being closer to "21" than the dealer. If your cards add up to more than "21", you lose. If you are unfamiliar with the game or need a refresher, play a couple of hands at: 24-7 Blackjack.

For the specific rules of the game, check out Wikipedia's entry for Blackjack.

To get started, don't worry about any of the special plays that can be made like splitting or doubling down. Remember to start with the simplest behaviors and build from there.

Meetup dates

Meetups often happen on the same day every month: for example, New Relic's Future Talks are every second Monday of the month, the Portland Java User's Group meets every second Tuesday of the month. Write a program that calculates the dates of a meetup based on its recurring schedule. Find the details for this exercise at exercismio.com. Continue to use Mocha and Chai for testing (not jasmine as in the exercise).

99 bottles of coke, Roman numerals, Word Order, Bases

Goal for the day: Practice looping in your JavaScript applications. Follow good BDD practice - red, green, REFACTOR.

99 bottles of coke

Create a program which takes input from a user and writes out the lyrics for the song "99 bottles of coke" replacing the starting point of 99 with the user's chosen number.

If you are unfamiliar with the song, it follows these lyrics from 99 down to the final 1 bottle of coke:

"99 bottles of coke on the wall, 99 bottles of coke... take one down, pass it around, 98 bottles of coke on the wall. 98 bottles of coke on the wall, 98 bottles of coke... take one down, pass it around, 97 bottles of coke on the wall."

And on it goes, until at last:

"1 bottle of coke on the wall, 1 bottle of coke... take it down, pass it around, no more bottles of coke on the wall!"

To which everyone sighs a great relief.

Remember that the last two lines of the song have to be handled a little differently than the rest of the lyrics since the song is down to a single bottle.

Roman numerals

Write a method to convert numbers into Roman numerals.

For some background, Roman numerals are based on seven symbols:

```
Symbol  Value
I       1
V       5
X       10
L       50
C       100
D       500
M       1,000
```

The most basic rule is that you add the value of all the symbols: so II is 2, LXVI is 66, etc.

The exception is that you can't have more than three of the same character is a row: instead, you switch to subtraction. So instead of writing IIII for 4, you write IV (for 5 minus 1); and instead of writing LXXXX for 90, you write XC.

You also have to separate ones, tens, hundreds, and thousands. In other words, 99 is XCIX, not IC. With Roman numerals, you can't count higher than 3,999 in Roman numerals.

Draft some input-output specs to brainstorm the behaviors that you'll want to capture in your application. Use BDD to tackle them one-by-one, worrying only about the future specs once the current spec is passing.

Bases

Binary

Write a method to convert numbers from binary to decimal. The input should be a string, and the output an integer.

Decimal is the normal system we use for counting. We start at 0, increment until we reach 9, and then reset back to 0 and add another number to the left.

In binary, we also start at zero, but we only increment until we reach 1. Then we reset back to zero and add another number to the left.

Here are some example of numbers in decimal and binary:

```
Decimal  Binary
  0        0
  1        1
  2       10
  3       11
  4      100
...      ...
```

Trinary

You get where I'm going, right?

Hexadecimal

Here's what happens after 9:

```
Decimal  Hexadecimal
...      ...
 9        9
10        a
11        b
12        c
13        d
14        e
15        f
16       10
17       11
...     ...
```

If you get this far, write a method that takes two arguments: the number to be evaluated and the base you would like it to be evaluated in. Make sure to use BDD as you go.

AngularJS - Introduction Video

> **Embedded Video: https://www.youtube.com/watch?v=fzWtSdgwgzU?rel=0**

This video is by Dan Wahlin.In the video you'll learn how to get started with the AngularJS framework and some of the key features it provides that simplify SPA development. You'll see how to use directives, filters and data binding techniques to capture and display data. Next up is views, controllers and scope and the role they play followed by a discussion of modules, routes and factories/services. At the end of the video a sample application built using AngularJS is shown. Hope you enjoy and thank Dan for this wonderful video and that it helps jumpstart your AngularJS learning process!

AngularJS objectives

In this section we will be using AngularJS to develop interactive front end without all of the jQuery. We will start by integrating AngularJS into our apps to add to a list and display that list. We will learn about controllers and directives. Then we will learn how to use multiple controllers in our applications and how to create factories to share data between controllers. Next we will look at using AngularUI Router to create nested states and routes in our application so that our website can have multiple pages. Finally, we're going to learn how to write reusable custom directives, to make our pages simple to write, and extremely flexible.

Getting started with AngularJS

> **Embedded Video: https://www.youtube.com/watch?v=__e05i4vIU4?rel=0**

As you've increasingly used JavaScript and jQuery in your websites, you might be itching to take things a step further and build more complex user interfaces. For example, check out Virgin America's website. Notice how much interaction they have - when you click a destination city, new options are presented in place. Everything is super snappy and interactive. Take a moment to think about how difficult this would be to implement with just jQuery.

To build complex user interfaces like these, a new set of tools has evolved called **client-side MVCs**. MVC stands for model-view-controller, which is a way of setting up user interfaces that separates out the code into three interconnected parts: model, view, controller. A client-side MVC provides structure for your JavaScript code and automates common tasks. AngularJS is one of the most popular client-side MVCs. To start learning how to use it, we're going to build a simple app to manage all the students in a class. With Angular, we can add, edit and delete students as well as filter them on a single page, with relatively little work.

To get started, let's download Angular from the AngularJS homepage. Click the download button, and then a pop up window will present you with some options. Leave the selected branch at the number marked "stable". Then choose the uncompressed version which offers better debugging and development capabilities than the minified version. Then right click on the "Download" button and select "Save linked file as…", naming it `angular.js` . This is the core angular code that your application will depend on to run.

We'll start by creating a project directory called ‘student-roster’. Inside that folder, we can create folders called `lib` , `js` and `css` . Let's put the file `angular.js` inside of `lib` . I’m using Bootstrap, so I’ll put the `bootstrap.min.css` file in the `css` folder.

To create an application with Angular, we need to start off by creating a file called `app.js` in the top level of the project directory. For now, it only needs this one line of code:

student-roster/app.js

```
var studentRoster = angular.module('studentRoster', []);
```

The first argument to `angular.module` is the name of our app, which we are calling `studentRoster` . The second argument is any dependencies that the app might have; since our app has none, we will leave that array empty for now.

Just so we can make sure the app is actually loading, let's create a skeleton `index.html` file, also in the top level of our project directory, and then we'll open it in the browser:

student-roster/index.html

```
<!doctype html>
<html>
<head>
  <title>Student App</title>
  <script src="lib/angular.js"></script>
  <script src="app.js"></script>
  <link rel="stylesheet" href="css/bootstrap.min.css">
</head>
<body>
</body>
</html>
```

First, we include the usual tags for doctype, html, head, and body. Then inside of the head we set the title of our app, load the `lib/angular.js` file, followed by our `app.js` file, and lastly a link to the `bootstrap.min.css` file in our css folder. Be sure to notice the load order - we have to load `angular.js` first, and then the `app.js` file.

At this point, we'll see a blank page. But if we open the JavaScript console and type `studentRoster`, we should get an object returned. This object is actually our new Angular app. Since we haven't written any code yet, there's not much to see there besides the `name` property. Let's write some code to display a list of current students.

In Angular, we use `controllers` to handle the logic of the app. Let's create our first controller and populate it with some sample data so that we have students to list. We will get to dynamically adding students with a form later. Create a file called `StudentsController.js` inside of your `js` folder, and then add this code to it:

student-roster/js/StudentsController.js

```
studentRoster.controller('StudentsCtrl', function StudentsCtrl($scope) {
  $scope.students = [
    { name: "Sam Schmidt" },
    { name: "Jessica Martin" },
    { name: "Sandy Smith" },
    { name: "Ryan Samuels" },
    { name: "Brentwood Davis" }
  ]
});
```

In this file, we are calling the `controller()` method on our app `studentRoster`, which creates a controller. As arguments, we're passing in the name of the controller - `StudentsCtrl` - and a function that holds the controller's logic. This is similar to the way we created our app object by passing the name 'studentRoster' to the `angular.module()` method in our first file. We pass in a name as the first argument, and any other information the method needs as the second argument. The controller needs a function as that second argument.

Notice too that our controller function has a parameter called `$scope`. We've added an array of student data as a property of that `$scope`, and we'll use this data to populate our page. In Angular apps, data that is used to populate the page is called a **model**.

Now, let's display our sample student data. Make your `index.html` file look like this:

student-roster/index.html

```
<!doctype html>
<html ng-app="studentRoster">
<head>
  <title>Student App</title>
  <script src="lib/angular.js"></script>
  <script src="app.js"></script>
  <script src="js/StudentsController.js"></script>
  <link rel="stylesheet" href="css/bootstrap.min.css">
</head>
<body>
  <div class="container" ng-controller="StudentsCtrl">
    <div class="row">
      <div class="col-md-12">
        <h2>Student List</h2>
        <ul>
          <li ng-repeat="item in students">
            {{item.name}}
          </li>
        </ul>
      </div>
    </div>
  </div>
</body>
</html>
```

If we refresh the browser, we should see a list of the sample students!

Let's look through the changes we've made. First, we have added `<script src="js/StudentsController.js"></script>` into the `<head>` tag, to load our controller. Again, be sure to notice the load order. We have to load the controller file after the app.js file, and both of them have to be loaded after `angular.js`.

The `ng-` attributes we've added to our HTML tags are called **directives** in Angular (`ng-app` , `ng-controller` , and `ng-repeat`). They are used to extend the HTML with dynamic content and special behaviors. When your Angular app boots, it looks at these attributes, attaches the specified behavior to each element, and updates the DOM if necessary. The final version of what's rendered in the browser is called a **view** in Angular.

The first directive we have is `ng-app="studentRoster"` that we added to the `<html>` tag at the top of the file. This tells Angular that our entire web page will be controlled by Angular. If we only wanted a part of it to be controlled, we could add that directive to another element in our page, such as `<div ng-app="studentRoster">` .

Next we have the `ng-controller` directive. As you might expect, this designates the controller for the element where it is placed. By adding it to the `<div>` , we're specifying that any directives inside of that `<div>` should refer to the StudentsCtrl controller, and any data that it pulls should come from the `$scope` in that controller.

Lastly, the `ng-repeat` directive is used to iterate through data that is stored in the `$scope` via the controller. `item in students` says to take each element in the `students` array and refer to it as `item` . Since we defined `ng-controller` in a parent `<div>` , Angular knows that to find that `students` array it must look in the `StudentsCtrl` controller. We then use the double curly braces to display the name property of each `item` . Just so you know, the word item isn't special - this could be called `"student in students"` or anything else just as well. Angular developers use `item` a lot, so we'll stick with that for clarity.

It's important to remember that in order for a given directive to have access to a specific controller, the directive must be nested inside of an element with that controller defined via `ng-controller` .

That was a lot to take in! As you can see, directives serve a variety of different purposes. But they all share the common job of dealing with display logic and have direct access to `$scope` . Check out the AngularJS Documentation on directives and the [AngularJS API Docs}(https://docs.angularjs.org/api) for directives you can use.

Summary

Download a non-minified version of AngularJS.

Create an `app.js` file to start the Angular app:

app.js

```
var studentRoster = angular.module('studentRoster', []);
```

Use the `ng-app` directive in the `<html>` tag at the top of the `index.html` page, include the controller in a `<script>` tag, and use the directive `ng-controller` to set the controller for a particular section of the page:

student-roster/index.html

```
<!doctype html> <html lang="en" ng-app="studentRoster"> <head> <meta charset="UTF-8"> <ti
```

The directive `ng-repeat` will loop through an array and you can display any information about the element in that array by using `{{ }}` double curly brackets and using whatever you named each item in the `ng-repeat` directive.

student-roster/js/StudentsController.js

```
studentRoster.controller('StudentsCtrl', function StudentsCtrl($scope) {
  $scope.students = [
    { name: "Sam Schmidt" },
    { name: "Jessica Martin" },
    { name: "Sandy Smith" },
    { name: "Ryan Samuels" },
    { name: "Brentwood Davis" }
  ]
});
```

Make sure to include `$scope` when you declare the controller function. This `$scope` is scoped to within this controller and you need to use it to create properties or add methods to this controller.

Adding methods and forms

> **Embedded Video: https://www.youtube.com/watch?v=2w-WVKvXw1g?rel=0**

Next, let's make it possible to add new students dynamically so that we don't have to rely on sample data. We'll switch the order up and start with our HTML and directives this time, so that we have an idea of how we want the page to look:

student-roster/index.html

```
<!doctype html>
<html ng-app="studentRoster">
<head>
  <title>Student App</title>
  <script src="lib/angular.js"></script>
  <script src="app.js"></script>
  <script src="js/StudentsController.js"></script>
  <link rel="stylesheet" href="css/bootstrap.min.css">
</head>
<body>
  <div class="container" ng-controller="StudentsCtrl">
    <div class="row">
      <div class="col-md-12">
        <h2>Student List</h2>
        <ul>
          <li ng-repeat="item in students">
            {{item.name}}
          </li>
        </ul>
      </div>
      <div class="col-md-4">
        <h2>New Student</h2>
        <form ng-submit="addStudent()" class="form-inline" role="form">
          <div class="form-group">
            <input type="text" ng-model="studentName" class="form-control" placeholder="E
          </div>
          <button type="submit" class="btn btn-default">Submit</button>
        </form>
      </div>
    </div>
  </div>
</body>
</html>
```

We've added a new column below where we are listing out the students. It contains a form to add a new student. The `ng-submit` directive will trigger the `addStudent` method when that form is submitted. The directive `ng-model` creates a new property on `$scope` in our `StudentsCtrl` controller called `studentName` and sets it to the value of the input. Here, `studentName` is the model, just like `students` is the model in the part of the app that lists the students out.

Now, let's create the `addStudent` method to our `StudentsCtrl` controller in order to make this form work:

student-roster/js/StudentsController.js

```
studentRoster.controller('StudentsCtrl', function StudentsCtrl($scope) {
  $scope.students = [];
  $scope.addStudent = function() {
    $scope.students.push({ name: $scope.studentName });
    $scope.studentName = null;
  };
});
```

We've removed the sample data from the `$scope.students` array because now we will be pushing a new student object into that array when we use the `addStudent` method. This method will create a new student object based on the user's input and push it into the `students` array. It then resets the `studentName` model to `null` so that the field in the HTML is cleared.

We can now add our own students to the page!

Next, let's add the ability to delete students. Again, we'll start with the HTML and directives. Change the `<div>` holding the student list so that it includes the following `<a>` tag:

student-roster/index.html

```
...
<div class="col-md-12">
  <h2>Student List</h2>
  <ul>
    <li ng-repeat="item in students">
      {{item.name}} <a ng-click="deleteStudent(item)">Delete</a>
    </li>
  </ul>
</div>
...
```

We have added the line `<a ng-click="deleteStudent(item)">Delete</a>` inside the `ng-repeat` directive from the last lesson. `ng-click` is very similar to `ng-submit` - except, as you might have guessed, it gets triggered on a click instead of a submit action. This is similar to attaching a click handler to an HTML element in jQuery, except here instead of using a jQuery selector with the dollar sign, we are using a directive on an HTML element. So when the word "Delete" is clicked, the app will run the `deleteStudent` method. We have passed `item` into the method so it knows which particular object to delete. Here's the `deleteStudent` method in our controller. Add this to `js/StudentsController.js` right after the declaration of the `addStudent` method, but before the closing brackets of the `studentRoster.controller` method.

student-roster/js/StudentsController.js

```
...
$scope.deleteStudent = function(student) {
  var index = $scope.students.indexOf(student);
  $scope.students.splice(index, 1);
};
...
```

We pass in the student as an argument to the `deleteStudent` method, use the built-in JavaScript method `indexOf` to find the index of the student in our `$scope.students` array, and call the JavaScript `splice` method on it. The `splice` method removes elements from an array. Its first argument is an integer, which is the index where it should start removing array elements. The second argument is the number of elements we want to delete. So in this case we are starting our removal at the index of the student and only deleting that one element.

Summary

An Angular form:

```
<form ng-submit="addStudent()" class="form-inline" role="form">
  <div class="form-group">
    <input type="text" ng-model="studentName" class="form-control" placeholder="Enter nam
  </div>
  <button type="submit" class="btn btn-default">Submit</button>
</form>
```

- Add the `ng-submit` directive to run a particular function when the form is submitted.
- Use `ng-model` to create a new property on the `$scope` of that particular controller and

set it equal to the name passed into that directive.

The directive `ng-click` can be used to call a method or set a property when not using a form.

Two-way binding

Alright, I think we're getting comfortable with the basics. Let's learn about a really handy feature of Angular called **two-way data binding**. To help us understand this concept, we'll first implement the ability to edit a student on our roster. I'd like to make it so that the user doesn't have to leave the page to edit the student's name.

Amazingly, we won't need to write any code in the controller to accomplish this. Instead, we're just going to change some things in the student list section of our `index.html` file:

student-roster/index.html

```
<h2>Student List</h2>
<ul>
  <li ng-repeat="item in students">
    <span ng-click="editing = true" ng-hide="editing">
      {{item.name}} <a ng-click="deleteStudent(item)">Delete</a>
    </span>
    <span class="form-group" ng-show="editing" ng-submit="editing = false">
      <form class="form-inline" role="form">
        <input type="text" class="form-control input-sm" ng-model="item.name" placeholder
        <button class="btn btn-default btn-sm" type="submit">Save</button>
      </form>
    </span>
  </li>
</ul>
```

Let's see how all of this is working. First, we use the `ng-click` directive to set a new property called `editing` equal to `true` when the name of a student in our list is clicked. We've also attached the `ng-hide` directive to that same span, which will automatically hide the given element when the expression provided evaluates to `true`. Since `editing` gets set to true by being clicked, the `<span>` gets hidden when it is clicked.

In the next `<span>`, we use the `ng-show` directive (which works exactly opposite to `ng-hide`) to reveal our new form. So when the first is clicked and gets hidden, this next `<span>` is revealed, since editing is now true.

Finally, we've also specified via `ng-submit` that when that form gets submitted, our `editing` property will be set to `false`, switching which `<span>` is hidden and which is shown.

Now for the magic. We've set `ng-model="item.name", but how does Angular save the new value we provide after editing the form? That is the two-way data binding I mentioned earlier. Two-way data binding in Angular apps is the automatic synchronization of data between the model and view components. Remember that the view simply refers to the rendered HTML in Angular. So when the model changes, the view reflects that change; when the view changes, the model reflects that change.

The model for our input field in the HTML is the name property of the current item object. Remember that the word `item` came from using the `ng-repeat` directive to loop through all of the students in our list, placing each one of them into a variable called 'item'. By doing this we were able to display the name of the student in our view by saying . By also setting the `ng-model` directive to `item.name` we are able to update it using this simple inline form.

For a slightly silly metaphor, pretend Angular is in charge of your kitchenpantry. If you ask Angular to display how much coffee is left, that would be like putting into your view. Angular will look in the correct place in the pantry and tell you how much coffee you have left. Angular knows which jar has the coffee in it, and keeps "live" track of the inventory. If you buy more coffee Angular will be smart enough to keep track of the increase in the overall amount of coffee and report that change to us in real time. This is like updating the student's name in our model. It's important to understand that Angular doesn't have a separate place where these values are stored, they're always live, and the view is always a direct window to the model.

This means that while you are editing, your changes are continuously being saved to the model as they are being made. The view is a projection of the model at all times. So there is no specific action required when you submit the changes, since they have already been synced.

Summary

Two-way data binding is the automatic synchronization of data between the model and view components. So when the model changes, the view reflects that change; when the view changes, the model reflects that change.

Example of how to change a form field inside the HTML using two-way data binding:

```
<h2>Student List</h2>
<ul>
  <li ng-repeat="item in students">
    <span ng-click="editing = true" ng-hide="editing">
      {{item.name}} <a ng-click="deleteStudent(item)">Delete</a>
    </span>
    <span class="form-group" ng-show="editing" ng-submit="editing = false">
      <form class="form-inline" role="form">
        <input type="text" class="form-control input-sm" ng-model="item.name" placeholder
        <button class="btn btn-default btn-sm" type="submit">Save</button>
      </form>
    </span>
  </li>
</ul>
```

Filtering in Angular

Let's move along. Say we've added a bunch of students to our class and we need to search or filter the list of students. Angular provides a really simple way to accomplish this. Here's the new code:

student-roster/index.html

```
<body>
  <div class="container" ng-controller="StudentsCtrl">
    <div class="row">
      <div class="col-md-12">
        <h4>Search Students</h4>
        <form class="form-inline" role="form">
          <div class="form-group">
            <input ng-model="query" type="text" class="form-control" placeholder="Search"
          </div>
        </form>
      </div>
    </div>
    <div class="row">
      <div class="col-md-12">
        <h2>Student List</h2>
      </div>
      <div class="col-md-12">
        <ul>
          <li ng-repeat="item in students | filter:query">
            <span ng-click="editing = true" ng-hide="editing">
              {{item.name}} <a ng-click="deleteStudent(item)">Delete</a>
            </span>
            <span class="form-group" ng-show="editing" ng-submit="editing = false">
              <form class="form-inline" role="form">
                <input type="text" class="form-control input-sm" ng-model="item.name" pla
                <button class="btn btn-default btn-sm" type="submit">Save</button>
              </form>
            </span>
          </li>
        </ul>
      </div>
    </div>
...
  </div>
</body>
```

Now if we start typing in the search field, it will automatically filter the results!

Let’s take a look at how this was done. First, we are adding a new form at the top of the page with a text field where we can enter a student's name to search for. In the input tag of our new form we use the `ng-model` directive to set a new property called `query`. Thanks to the two-way binding, this model will constantly remain updated as the user is typing.

We then use the `filter` filter (`filter` is only one type of filter in Angular) and pass it `query` as the expression. The filter will generate a new array of students based on the expression passed. Again, thanks to the two-way data binding, `filter` will automatically narrow the results displayed based on the changing value of our `query` model.

To make this page look a little cleaner, let's hide the search `<div>` and the Student List if no students have been added yet. Here’s the code:

student-roster/index.html

```
<body>
  <div class="container" ng-controller="StudentsCtrl">
    <div class="row" ng-show="students.length">
      <div class="col-md-12">
        <h4>Search Students</h4>
        <form class="form-inline" role="form">
          <div class="form-group">
            <input ng-model="query" type="text" class="form-control" id="student-name" pl
          </div>
        </form>
      </div>
    </div>
    <div class="row">
      <div class="col-md-12">
        <h2 ng-show="students.length && filtered.length">Student List</h2>
        <h2 ng-show="students.length && !filtered.length">No Matches</h2>
        <ul>
          <li ng-repeat="item in filtered = (students | filter:query)">
            <span ng-click="editing = true" ng-hide="editing">
              {{item.name}} <a ng-click="deleteStudent(item)">Delete</a>
            </span>
            <span class="form-group" ng-show="editing" ng-submit="editing = false">
              <form class="form-inline" role="form">
                <input type="text" class="form-control input-sm" ng-model="item.name" pla
                <button class="btn btn-default btn-sm" type="submit">Save</button>
              </form>
            </span>
          </li>
        </ul>
      </div>
      <div class="col-md-4">
        <h2>New Student</h2>
        <form ng-submit="addStudent()" class="form-inline" role="form">
          <div class="form-group">
            <input type="text" ng-model="studentName" class="form-control" placeholder="E
          </div>
          <button type="submit" class="btn btn-default">Submit</button>
        </form>
      </div>
    </div>
  </div>
</body>
```

We've added `ng-show="students.length"` to the search `<div>`. If there are no students, then students.length will evaluate to 0, and `0` is falsy in JavaScript, so this will make the `ng-show` directive hide the search form.

Next, we only want to show the `<h2>` "Student List" if there are currently students in the roster, and students matching the search criteria. We do this with ng-show by telling it "Only show this element if both students.length and filtered.length are true".

We also create a second `<h2>` tag to display the words "No Matches" when there are students in the roster, but none of them match the student we are searching for. To do this we use `ng-show` again and tell it "only show this element when students.length is true, and when filtered.length is false."

Lastly, let's look at this line: `ng-repeat="item in filtered = (students | filter:query)"`. It used to say `ng-repeat="item in students"`, which allowed us to view all students. Here we are making a new variable called `filtered` to iterate through. Then we set it to `students | filter:query`. Breaking this down, `filter:query` refers to the filtered array of students that we made in our query model, and students still refers to all the students in the roster. The | symbol basically means `or`. This way, if the query returns anything (we found some student matches) then show the search results. If there has not been a query made, then display all students instead.

Check out other ways to filter data in the AngularJS API docs.

Alright, that's it for the basics of Angular!

Multiple controllers and factories

At this point, we are able to make a webpage with a controller where we can create new objects and display those objects in our `index.html` view. Let's say that the teacher using our `student-roster` application wants to be able to keep a separate list on the same page to keep track of which students have turned in their field trip permission slip, so that a phone call can be made to the parents of those who haven't.

At this point we're adding new functionality, so it's appropriate to add another controller. We want to build each of our controllers to handle one specific functionality. This prevents our controllers from becoming too large to understand as we develop more complex apps.

Let's create a new controller to handle the permission slip functionality. We'll call it `FieldTripsCtrl`. Let's also change the `js` folder to be named `controllers` to be clearer about what that folder contains. Don't forget to change the `<script>` tag in the `index.html` page too.

Let's add our `FieldTripsCtrl` now:

controllers/FieldTripsController.js

```
studentRoster.controller('FieldTripsCtrl', function FieldTripsCtrl($scope) {
  $scope.studentsWithPermission = [{ name: "Jane Doe" }, { name: "John Doe" }];
});
```

We've added some sample students who have turned in their permission slips. Now we'll need to update our index.html to display this list.

index.html

```
//This code can go below the form to enter a student
 <div class="row permission">
    <div class="col-md-6">
      <h2>Permission Slips</h2>
      <ul>
        <li ng-repeat="item in studentsWithPermission">
          {{item.name}}
        </li>
      </ul>
    </div>
  </div>
```

Now let's refresh and give it a whirl. Uh-oh, you'll notice it successfully loads our page with the `<h2>`, but there are no students showing! There are also no errors in the console. What's going on?

Right now this `<div class="row permission">` section is still under the power of the `StudentsCtrl`, so it can't access the `studentsWithPermission` property in the `FieldTripsCtrl` controller.

Let's fix that. Add in `ng-controller = "FieldTripsCtrl"` into the `<div class="row permission">`. Now everything contained in that `<div>` will be under the control of the `FieldTripsCtrl`. Let's move the `ng-controller = "StudentsCtrl"` to a new `<div>` that wraps the old part of the page:

index.html

```
<body>
  <div class="container">
    <div  ng-controller="StudentsCtrl">
      <div class="row" ng-show="students.length">
        <div class="col-md-12">
          <h4>Search Students</h4>
          <form class="form-inline" role="form">
            <div class="form-group">
              <input ng-model="query" type="text" class="form-control" id="student-name"
            </div>
          </form>
        </div>
      </div>
      <div class="row">
        <div class="col-md-6">
          <h2 ng-show="students.length && filtered.length">Student List</h2>
          <h2 ng-show="students.length && !filtered.length">No Matches</h2>
          <ul>
            <li ng-repeat="item in filtered = (students | filter:query)">
              <span ng-click="editing = true" ng-hide="editing">
                {{item.name}} <a ng-click="StudentsFactory.deleteStudent(item)">Delete</a
              </span>
              <span class="form-group" ng-show="editing" ng-submit="editing = false">
                <form class="form-inline" role="form">
                  <input type="text" class="form-control input-sm" ng-model="item.name" p
                  <button class="btn btn-default btn-sm" type="submit">Save</button>
                </form>
              </span>
            </li>
          </ul>
        </div>
        <div class="col-md-6">
          <h2>New Student</h2>
          <form ng-submit="StudentsFactory.addStudent()" class="form-inline" role="form">
            <div class="form-group">
```

```
                <input type="text" ng-model="StudentsFactory.studentName" class="form-contr
              </div>
              <button type="submit" class="btn btn-default">Submit</button>
            </form>
          </div>
        </div>
      </div>
     <div class="row permission" ng-controller="FieldTripsCtrl">
        <div class="col-md-6">
          <h2>Permission Slips</h2>
          <ul>
            <li ng-repeat="item in studentsWithPermission">
              {{item.name}}
            </li>
          </ul>
        </div>
      </div>
    </div>
  </body>
```

Now if you refresh the page, it should load our pre-made list of students who have turned in their permission slips.

Our pre-made list of students is great, but it isn't linked to our list of actual students. What we really need is a single place to store information about the students. In this case, that will include their name, and well as whether they've turned in their permission slip or not.

You might think that this code would work:

controllers/FieldTripsController.js

```
studentRoster.controller('FieldTripsCtrl', function FieldTripsCtrl($scope) {
  $scope.addStudentWithPermissionSlip = function(student) {
    student.permissionSlip = true;
  };
});
```

The problem is that this `FieldTripsCtrl` controller can't access the `$scope.students` array that currently resides in the `StudentsCtrl` controller. The `$scope` variable is scoped to its controller.

Because any one `<div>` can only have one `ng-controller` directive active at a time we need a place where any controller can access data.

Angular has us covered. In order to allow controllers to share data, we need to use a **factory**. A factory is one of many **services** that Angular offers to organize and share code across your app. A factory gets called once at the start of the application and is available for use by any controller throughout the session. It's the well from which any controller can draw water.

Let's create a folder called `services` and put our factory inside of it:

services/StudentsFactory.js

```
studentRoster.factory('StudentsFactory', function StudentsFactory() {
  var factory = {};
  factory.students = [];

  factory.addStudent = function() {
    var student = { name: factory.studentName, permissionSlip: false };
    factory.students.push(student);
    factory.studentName = null;
  };

  factory.deleteStudent = function(student) {
    var index = factory.students.indexOf(student)
    factory.students.splice(index, 1);
  };
  return factory;
});
```

We've taken all the properties and methods that used to be in the `StudentsCtrl` and moved them to the factory, so that multiple controllers can have access to them. We have also changed our `addStudent` method so that when we create a new student it has `permissionSlip` set to `false` by default. Here's what our StudentsCtrl should look like now:

controllers/StudentsController.js

```
studentRoster.controller('StudentsCtrl', function StudentsCtrl($scope, StudentsFactory) {
  $scope.students = StudentsFactory.students;
  $scope.StudentsFactory = StudentsFactory;
});
```

Notice we need to pass in `StudentsFactory` as an argument to our controller in order to use it. This tells our controller where to find data. We've also created a `$scope.students` property. That's a shortcut for the array of students, so that we can simply call `students` instead of `StudentsFactory.students` every time.

Lastly, we need to update the `index.html` file by requiring the `StudentsFactory` script and reflecting our changes in the body:

index.html

```
...
<div class="row">
  <div class="col-md-6">
    <h2 ng-show="students.length && filtered.length">Student List</h2>
    <h2 ng-show="students.length && !filtered.length">No Matches</h2>
    <ul>
      <li ng-repeat="item in filtered = (students | filter:query)">
        <span ng-click="editing = true" ng-hide="editing">
          {{item.name}} <a ng-click="StudentsFactory.deleteStudent(item)">Delete</a>
        </span>
        <span class="form-group" ng-show="editing" ng-submit="editing = false">
          <form class="form-inline" role="form">
            <input type="text" class="form-control input-sm" ng-model="item.name" placeho
            <button class="btn btn-default btn-sm" type="submit">Save</button>
          </form>
        </span>
      </li>
    </ul>
  </div>
  <div class="col-md-6">
    <h2>New Student</h2>
    <form ng-submit="StudentsFactory.addStudent()" class="form-inline" role="form">
      <div class="form-group">
        <input type="text" ng-model="StudentsFactory.studentName" class="form-control" id
      </div>
      <button type="submit" class="btn btn-default">Submit</button>
    </form>
  </div>
</div>
...
```

All we've done here is prepend 'StudentsFactory' to the methods that we're calling from the factory, and to the `studentName` model, since student names are now being saved in the `StudentsFactory` and not the `StudentsCtrl` .

Now we need to tell our `FieldTripsCtrl` how to find `StudentsFactory` and how to change data inside of it. Here's what it should look like:

controllers/FieldTripsController.js

```
studentRoster.controller('FieldTripsCtrl', function FieldTripsCtrl($scope, StudentsFactor
  $scope.students = StudentsFactory.students;
  $scope.addStudentWithPermissionSlip = function(student) {
    student.permissionSlip = true;
  };
});
```

Finally we need to update the `FieldTripsCtrl` section of our HTML to use this method and show the correct lists of students.

index.html

```
<div class="row" ng-controller="FieldTripsCtrl">
  <div class="col-md-6" ng-show="studentsWithoutPermission.length">
    <h3>Without Permission Slips</h3>
    <ul>
      <li ng-repeat="item in studentsWithoutPermission = (students | filter:{permissionSl
        {{item.name}} <a ng-click="addStudentWithPermissionSlip(item)"> - received permis
      </li>
    </ul>
  </div>
  <div class="col-md-6" ng-show="studentsWithPermission.length">
    <h3>With Permission Slips</h3>
    <ul>
      <li ng-repeat="item in studentsWithPermission = (students | filter:{permissionSlip:
        {{item.name}}
      </li>
    </ul>
  </div>
</div>
```

By adding the directive `ng-show="studentsWithoutPermission.length"` we are only showing the permission slip information if there are actually students in the `studentsWithoutPermission` array. You might be wondering when we actually created the `studentsWithoutPermission` array. We do just below with the line `<li ng-repeat="item in studentsWithoutPermission = (students | filter:{permissionSlip: false})">`. This sets the `studentsWithoutPermission` array equal to the students array with the filter `permissionSlip: false`.

When the user clicks on the - received permission slip - link in that list, it will set the `permissionSlip` status to `true`, and then the student's name will be displayed in the `studentsWithPermission` array below.

We have two kinds of functionality on our page now, but only one type of object - a student - which is stored in the `StudentsFactory`. We also have two controllers updating different properties of that object type. And we know how to use multiple controllers and create factories in our Angular apps.

Summary

index.html

```
<!doctype html>
<html lang="en" ng-app="studentRoster">
<head>
  <meta charset="UTF-8">
  <title>Student App</title>
  <script src="lib/angular.js"></script>
  <script src="app.js"></script>
  <script src="controllers/StudentsController.js"></script>
  <script src="controllers/FieldTripsController.js"></script>
  <script src="services/StudentsFactory.js"></script>
  <link rel="stylesheet" href="css/bootstrap.min.css">
</head>
<body>
  <div class="container">
    <div ng-controller="StudentsCtrl">
      <div class="row" ng-show="students.length">
        <div class="col-md-12">
          <h4>Search Students</h4>
          <form class="form-inline" role="form">
            <div class="form-group">
              <input ng-model="query" type="text" id="student-name" class="form-control"
            </div>
          </form>
        </div>
      </div>
      <div class="row">
        <div class="col-md-12">
          <h2 ng-show="students.length && filtered.length">Student List</h2>
          <h2 ng-show="students.length && !filtered.length">No Matches</h2>
          <ul>
            <li ng-repeat="item in filtered = (students | filter:query)">
              <span ng-click="editing = true" ng-hide="editing">
                {{item.name}} <a ng-click="StudentsFactory.deleteStudent(item)">Delete</a
              </span>
              <span class="form-group" ng-show="editing" ng-submit="editing = false">
                <form class="form-inline" role="form">
                  <input type="text" class="form-control input-sm" ng-model="item.name" p
                  <button class="btn btn-default btn-small" type="submit">Save</button>
                </form>
              </span>
            </li>
          </ul>
        </div>

        <div class="col-md-4">
          <h2>New Student</h2>
          <form ng-submit="StudentsFactory.addStudent()" class="form-inline" role="form">
            <div class="form-group">
              <input type="text" ng-model="StudentsFactory.studentName" class="form-contr
            </div>
            <button type="submit" class="btn btn-default">Submit</button>
          </form>
        </div>
      </div>
    </div>

      <div class="row" ng-controller="FieldTripsCtrl">
        <div class="col-md-12" ng-show="studentsWithoutPermission.length">
          <h3>Without Permission Slips</h3>
          <ul>
            <li ng-repeat="item in studentsWithoutPermission = (students | filter:{permis
              {{item.name}} <a ng-click="addStudentWithPermissionSlip(item)"> - received
            </li>
          </ul>
        </div>
        <div class="col-md-12" ng-show="studentsWithPermission.length">
          <h3>With Permission Slips</h3>
          <ul>
            <li ng-repeat="item in studentsWithPermission = (students | filter:{permissio
              {{item.name}}
            </li>
          </ul>
        </div>
      </div>

  </div>
</body>
</html>
```

StudentsFactory.js

```
studentRoster.factory('StudentsFactory', function StudentsFactory() {
  var factory = {};
  factory.students = [];

  factory.addStudent = function() {
    var student = { name: factory.studentName, permissionSlip: false };
    factory.students.push(student);
    factory.studentName = null;
  };

  factory.deleteStudent = function(student) {
    var index = factory.students.indexOf(student);
    factory.students.splice(index, 1);
  };
  return factory;
});
```

StudentsController.js

```
studentRoster.controller('StudentsCtrl', function StudentsCtrl($scope, StudentsFactory) {
  $scope.students = StudentsFactory.students;
  $scope.StudentsFactory = StudentsFactory;
});
FieldTripsController.js
studentRoster.controller('FieldTripsCtrl', function FieldTripsCtrl($scope, StudentsFactor
  $scope.students = StudentsFactory.students;
  $scope.addStudentWithPermissionSlip = function(student) {
    student.permissionSlip = true;
  };
});
```

Routing and nested states

So far we have learned how to create a basic Angular app, use multiple controllers and add in factories to share properties, data and methods across controllers. Now, what if we want our app to have multiple courses, each with their own rosters? Let's redo the `studentRoster` app from the beginning with this new requirement.

To build this more complex Angular app, we're going to use AngularUI Router, which is organized around **states**. States describe how the user interface looks and behaves in a particular part of your application. Think of the phrase "a state of being" - this is just like the states we create using the UIRouter..

Let's get our files set up. Create a main project folder called `course-roster` and inside that folder we will create `lib` , `controllers` , `services` and `css` folders. Put `angular.js` in the `lib` folder and `bootstrap.css` in the `css` folder.

Now, we'll need to download and include the latest version of AngularUI Router. You can get that at AngularUI Router's Github page. Make sure you download the non-minified version and save it into the lib folder.

Let's create our `app.js` file with the following code:

app.js

```
var courseRoster = angular.module('courseRoster', ['ui.router']);
```

We pass in `'ui.router'` as a dependency to our `angular` module. Now we need to add it to our `index.html` file. Also, in previous lessons we placed all of our markup in our `index.html` file, but now we will have a couple different pages, so here is what our `index.html` file will look like:

index.html

```
<!doctype html>
<html lang="en" ng-app="courseRoster">
<head>
  <meta charset="UTF-8">
  <title>Course Roster</title>
  <script src="lib/angular.js"></script>
  <script src="lib/angular-ui-router.js"></script>
  <script src="app.js"></script>
  <link rel="stylesheet" href="css/bootstrap.css">
</head>
<body>
  <div class="container">
    <div ui-view></div>
  </div>
</body>
</html>
```

The directive `ui-view` tells the AngularUI Router that every view should be rendered into this `<div>` .

Now we can go ahead and set up our first state in `app.js` :

app.js

```
var courseRoster = angular.module('courseRoster', ['ui.router']);

courseRoster.config(function($stateProvider, $urlRouterProvider) {
  $stateProvider.state('home', {
    url: "",
    templateUrl: "partials/home.html"
  });
});
```

In this code, we have called the `config` method on our application and setup parameters for `$stateProvider` and `$urlRouterProvider` .

`$stateProvider` is what configures our application's states. We pass in the name of the state - `'home'` - and the configuration of the state - `{ url: "", templateUrl: "partials/home.html" }` - as arguments to the `state` method. Each page of our app is now going to be split up in a **template** that contains the HTML for the page.

Let's create a homepage where a user will go when they first navigate to our website. All of our templates will be stored in a `partials` folder. Let's make that folder now. Here is what the template will look like:

partials/home.html

```
<h1>Welcome to the Course Roster Application</h1>

<h3><a ui-sref="courses">Click here to view all courses</a></h3>
```

We have added a link with the directive `ui-sref` instead of `href` because it allows us to name the state that we would like displayed when the link is clicked, rather than the route of that template.

Let's check to see that our `home.html` template is rendered when we open the `index.html` page in our browser. Oops! There is nothing there and if we open up the JavaScript Console, we can see the error `XMLHttpRequest cannot load...Cross origin requests are only supported for protocol schemes: http, data, chrome-extension, https, chrome-extension-resource`. We are getting this error because AngularUI Router is trying to load our templates asynchronously, using JavaScript to request the file after the page has already loaded. Asynchronous JavaScript requests are not allowed under the `file://` protocol, but they are allowed over the `http://` protocol. The easiest way to deal with this problem is to serve our app from a web server.

On Macs, it's really easy to start a server. In your project folder in the terminal, enter `$ python -m SimpleHTTPServer` and then navigate to localhost:8000.

On Windows, you can download Mongoose, copy it to your project folder, and double-click it to launch a server and open your page in the browser.

And now you should see our home page! We can see that the `home` template is rendered into the `<div ui-view>` by the router.

Let's make the `courses` link work by creating a route in our `app.js` file, a new controller called `CoursesCtrl`, and a template for `courses.html` in our partials folder.

app.js

```
var courseRoster = angular.module('courseRoster', ['ui.router']);

courseRoster.config(function($stateProvider, $urlRouterProvider) {
  $stateProvider.state('home', {
    url: "",
    templateUrl: "partials/home.html"
  });

  $stateProvider.state('courses', {
    url: "/courses",
    templateUrl: "partials/courses.html",
    controller: 'CoursesCtrl'
  });
});
```

In the `courses` state, we'll add a controller to handle the logic and data for this template. States don't need to have controllers, like we saw with our `home` state, but often they will have one. Thinking ahead in our app, we are going to need to share the array of courses with a `StudentsCtrl` controller so that we can select the correct course to add a student to. So instead of just creating a `CoursesCtrl` controller, let's make a `CoursesFactory` and a `CoursesCtrl` controller. Here is the factory:

services/CoursesFactory.js

```
courseRoster.factory('CoursesFactory', function CoursesFactory() {
  var factory = {};
  factory.courses = [];
  factory.addCourse = function() {
    factory.courses.push({ name: factory.courseName, id: factory.courses.length + 1, stud
    factory.courseName = null;
  };

  return factory;
});
```

In this factory, when we create a new `course` object, we are including an `id` and an empty array called `students`. We'll use the ID to find a course after it's added, and we'll use the `students` array to store the students in the course. And let's make our `CoursesCtrl` controller here:

controllers/CoursesController.js

```
courseRoster.controller('CoursesCtrl', function CoursesCtrl($scope, CoursesFactory) {
  $scope.courses = CoursesFactory.courses;
  $scope.CoursesFactory = CoursesFactory;
});
```

Don't forget to add `<script src="controllers/CoursesController.js"></script>` and a `` `<script src="services/CoursesFactory.js"></script> `` to your `index.html` file.

Now let's make our `courses.html` file to display all of the courses.

partials/courses.html

```
<div ng-show="courses.length">
  <h1>Here are all of your courses:</h1>
  <ul>
    <li ng-repeat="item in courses">
      <a ui-sref="courses.students({ courseId: item.id })">{{ item.name }}</a>
    </li>
  </ul>
</div>

<div class="col-md-4">
  <h3>Add a new course</h3>
  <form ng-submit="CoursesFactory.addCourse()" class="form-inline" role="form">
    <div class="form-group">
      <input type="text" ng-model="CoursesFactory.courseName" class="form-control" placeh
    </div>
    <button type="submit" class="btn btn-default">Submit</button>
  </form>
</div>

<div ui-view> </div>
```

Here is what is going on with this code:

- We have created a section to display the courses that have been added, but it won't display unless there is at least one course in the `$scope.courses` array.
- In the `ui-sref` directive, we are linking to `courses.students`. The first part of that link - `courses` - is the state that renders this template. The second part - `.students` - indicates that the next state will be nested inside of this state. At the end of the page, we have included `<div ui-view></div>`. This is the nested template that will be displayed.
- So that we know which particular course we should show the details for in the `courses.students` state, we pass the course ID parameter as an argument to `courses.students({ courseId: item.id }).
- We have added a form so a user can enter the name of a new course.

Now it is time to add students to our courses. When we select the course, we want to display the name of the course, any students in that course and a form to add a student to that course. Let's start with the `courses.students.html` file first:

partials/courses.students.html

```
<div ng-show="course.students.length">
  <h2>Student List for {{ course.name }}</h2>
  <ul>
    <li ng-repeat="item in course.students">
      {{ item.name }}
    </li>
  </ul>
</div>

<div>
  <h2>Add a student to {{ course.name }}</h2>
  <form ng-submit="addStudent()" class="form-inline" role="form">
    <div class="form-group">
      <input type="text" ng-model="studentName" class="form-control" placeholder="Enter t
    </div>
    <button type="submit" class="btn btn-default">Submit</button>
  </form>
</div>
```

Now let's update our `app.js` file to include the `courses.students` state.

app.js

```
…
  $stateProvider.state('courses', {
    url: "/courses",
    templateUrl: "partials/courses.html",
    controller: 'CoursesCtrl'
  });

  $stateProvider.state('courses.students', {
    url: "/:courseId",
    templateUrl: "partials/courses.students.html",
    controller: 'StudentsCtrl'
  });
```

Because we have named this state `courses.students`, we are telling `ui-router` that we want this state to be nested within the `courses` state. Since the `courses` URL is `/courses`, and the `courses.students` URL is `/:courseId`, the nesting means that the full URL for `courses.students` will actually be `/courses/:courseId`.

Let's talk a bit more about the `:courseId` bit. Because it starts with a `:`, the router treats it as a variable. In other words, it will handle the URL `/courses/5` as well as `/courses/kitten`. When we created the `ui-sref` link on the `courses.html` page, we passed in the ID of the course as a parameter called `courseId`. We can now use that value in our state and use `:courseId` to insert the value of that ID into the URL.

Alright, there is one more thing we need to do before we can create our `StudentsCtrl` controller. When we actually write that controller, we are going to have access to the array of courses from the `CoursesFactory`, and we will have access to the ID of the particular course we want to push new students onto because of the parameter that we passed into the `courses.students` state.

The problem is that we don't have a good method for grabbing the correct course out of that `courses` array. JavaScript doesn't have a `findById` method already built, so we will need to build one ourselves. Because we might want to expand our app later to use this method in many different controllers, let's create a new factory called `UtilitiesFactory` to hold a `findById` method.

services/UtilitiesFactory.js

```
courseRoster.factory('UtilitiesFactory', function() {
  return {
    findById: function(collection, id) {
      for (var i = 0; i < collection.length; i++) {
        if (collection[i].id == id) {
          return collection[i];
        }
      }
      return null;
    }
  };
});
```

In this factory our `findById` method accepts a collection of things and an ID as arguments and then loops through the collection and if the ID of the current item in the collection matches the ID that we passed in, then it will return that object from the collection. If there are no matches, it returns `null`. Later, if we wanted to add another function that needed to be used in multiple controllers, we could add it to this `UtilitiesFactory`.

Don't forget to include a `<script>` tag for this new factory in the `index.html` page.

Now that we have the method built, we can finally create our `StudentsCtrl` controller and use `$stateParams`, `CoursesFactory`, and `UtilitiesFactory` as a dependency.

controllers/StudentsController.js

```
courseRoster.controller('StudentsCtrl', function StudentsCtrl($scope, $stateParams, Cours
  $scope.course = UtilitiesFactory.findById(CoursesFactory.courses, $stateParams.courseId
  $scope.addStudent = function() {
    $scope.course.students.push({ name: $scope.studentName });
    $scope.studentName = null;
  }
});
```

In order to find the correct course, we call the `findById` method that we just created and pass in the `CoursesFactory.courses` array and `$stateParams.courseId. $stateParams` gives us access to the `courseId` variable from the `url: "/:courseId"` part of the state. In other words, if the URL is `/courses/5`, then `$stateParams.courseId` will return `5`. Finally, we will push a new `student` object into the students array in that particular `course`.

Now if we fire up our application, we can see that everything works and we are adding our students onto the correct course!

Sass

> **Embedded Video: https://www.youtube.com/watch?v=1XmUUa_pWw8?rel=0**

What is Sass?

Sass stands for Syntactically Awesome Stylesheets. You’ve probably heard a lot about Sass, but you might still be a little hazy on what it actually is. Basically, Sass is one of a class of tools called CSS preprocessors. Preprocessors help us code more efficiently by adding special features into normal CSS, such as variables, nested rules, mixins, and more.

Why use it?

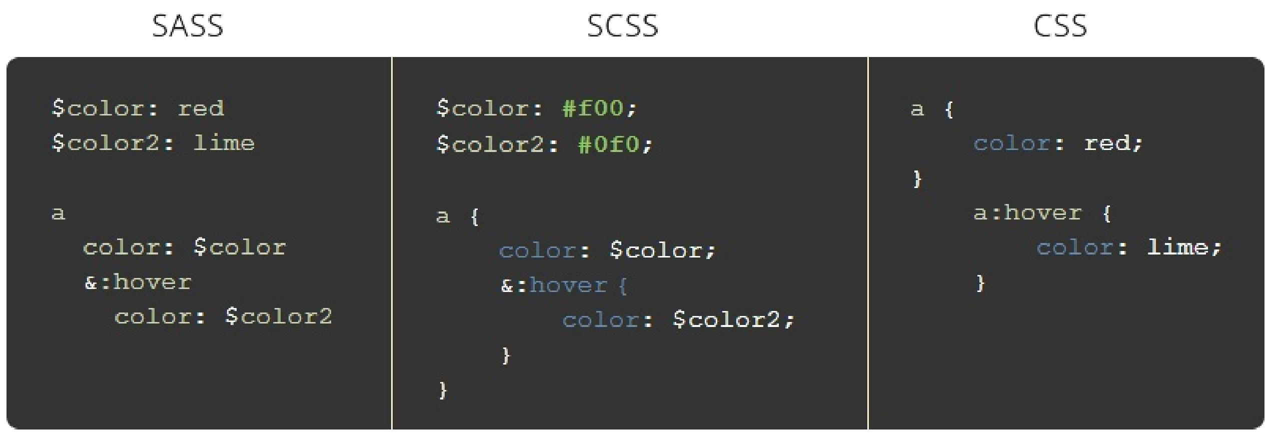
Have you ever had to use Find-and-Replace to make edits to colors or fonts across your website? Have you had to copy blocks of code over to multiple other elements in order for them to have the same custom styles? Do your stylesheets get a little repetitive or hairy sometimes? These are all common predicaments that bother every serious developer. Our cardinal rule, after all, is to keep our code DRY.

Sass helps us do just that.

Sure, a few repeated lines of CSS may not seem like a big deal for our class projects, but when you’re managing thousands of lines of production code in the real world, whether for a small startup or a big corporation, things can get pretty messy if you’re careless. CSS preprocessors like Sass help us alleviate that risk, and are thus a common requirement in many of today’s job postings.

Sass vs. SCSS

There are actually two kinds of syntax out there for Sass. Check 'em out:



SASS

```
$color: red
$color2: lime

a
  color: $color
  &:hover
    color: $color2
```

SCSS

```
$color: #f00;
$color2: #0f0;

a {
    color: $color;
    &:hover {
        color: $color2;
    }
}
```

CSS

```
a {
    color: red;
}
    a:hover {
        color: lime;
    }
```

As you can see, `.sass` syntax removes semicolons and curly brackets, and is arguably easier to type. On the other hand, many find the newer `.scss` syntax easier to read. In addition, SCSS is fully compliant with CSS, meaning you can rename your `.css` file to `.scss` and all your old code will still work. SCSS acts and looks more like your usual CSS, afterall, except with more tricks up its sleeve. If you're interested in reading more, check out this in-depth comparison.

In any case, you are welcome to code using whichever syntax you prefer, but for the most part we will be using SCSS in class.

Learning Sass

You will learn:

- All about Sass basic concepts
- How to manage media queries w/ Sass
- How to structure your Sass projects
- How to use Sass libraries like Bourbon

Sass does a pretty good job of outlining its basic concepts, but sometimes interactive, step-by-step examples can be a little more instructive.

- Getting Started with Sass (excluding Installing and Using Sass)
- Variables, Mixins, and Extending Selectors
- Speeding up Workflow with Sass (excluding Working with Sass Libraries)

For now, leave Advanced Sass Concepts alone.

Installing and using Sass

Installation

Follow the *Command Line* instructions on the official website.

There are several ways you can compile Sass:

- The original Ruby Sass binary. Install it with gem install sass, and compile it by running sassc myfile.scss myfile.css.
- A GUI app such as Hammer, CodeKit, or Compass
- My personal favorite libsass, which is a blazing fast Sass compiler written in C.
- You can also install libsass via NPM with node-sass (npm install node-sass).

Using Sass

Sass *compiles* into CSS. You – the developer/designer – code all of your styles into a `.sass` / `.scss` file. It is then compiled into normal CSS to be read and interpreted by browsers. You never have to touch the CSS file itself. This wizardry happens in a couple different ways.

You can either make it compile at will, using

```
sass filename-of-input.scss filename-of-output.css
```

or you can tell Sass to watch a file and update the CSS every time a change is detected:

```
sass --watch input.scss:output.css
```

The above tells Sass to sit and monitor all your .scss files in your css folder, and then automatically compile them into .css in the same folder. This command will be fitting for the majority of your Sass projects. It allows us to “set it and forget it”, giving us more time to focus on our actual Sass/CSS and site design.

Get up and running

All you really need to get started on a Sass project are the following:

- index.html
- output css file (e.g. main.css)
- input sass/scss file, which is all you'll be coding in (e.g. app.scss)

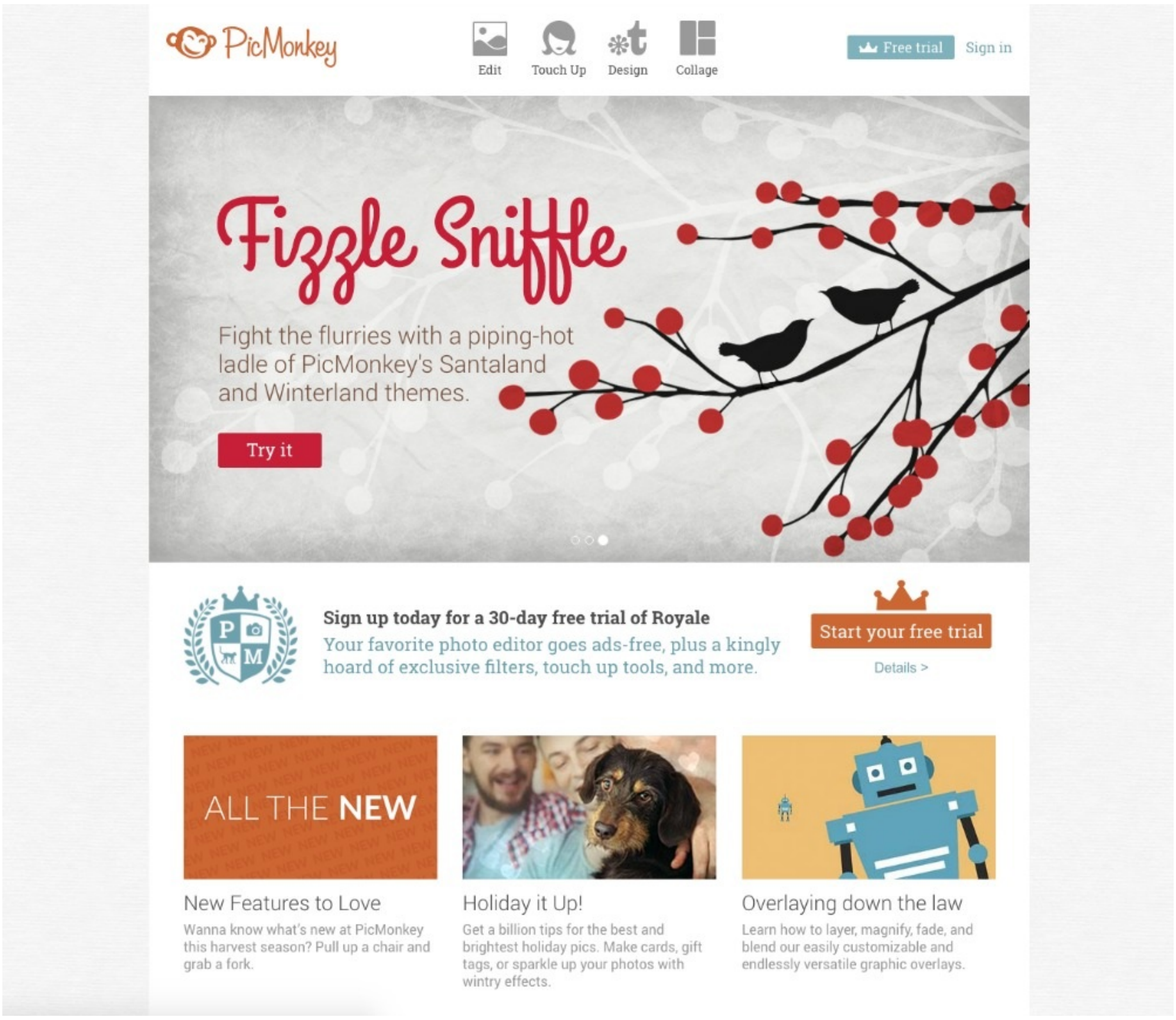
So, bringing it all together, all you have to do is open up your project folder in Terminal, type in `sass --watch application.scss:raw.css`, and get to work!

Creating your own grid layout using Sass

If you've used front-end frameworks like Bootstrap or Foundation before, you know how to make multi-column layouts using grids. They take the headache out of using floats for layouts by providing them built-in, clearfixes and all. Now, there's already a ton of well-polished grids out there for you to use in the real world (which is why we won't cover them much in our class), but creating your own is an excellent way for beginners to practice Sass fundamentals.

So get your Sass project set up (refer to the homework) and follow along with this awesome tutorial! Don't just blindly copy and paste, either. Try to analyze and understand what's going on with each step. As always, don't be afraid to pause and discuss concepts with your partner.

When you're done, try building a new webpage using your new custom grid! Let's see how many Sass features you can incorporate into your styling. If you're low on ideas, consider replicating the following:



Fizzle Sniffle

Fight the flurries with a piping-hot ladle of PicMonkey's Santaland and Winterland themes.

Sign up today for a 30-day free trial of Royale

Your favorite photo editor goes ads-free, plus a kingly hoard of exclusive filters, touch up tools, and more.

New Features to Love

Wanna know what's new at PicMonkey this harvest season? Pull up a chair and grab a fork.

Holiday it Up!

Get a billion tips for the best and brightest holiday pics. Make cards, gift tags, or sparkle up your photos with wintry effects.

Overlaying down the law

Learn how to layer, magnify, fade, and blend our easily customizable and endlessly versatile graphic overlays.

Sass Assignment #1 Lake Tahoe, Hack Oregon

Today, good ol' Guil is going to be walking you through how to convert a normal CSS project to Sass. Start with this video, and move onward.

Lake Tahoe

Complete the first two sections of the CSS to Sass course: *Installing Sass and Setting up the Project and Refactoring with Sass*. Both utilize Treehouse's **lake-tahoe** project. You are to download the project files onto your computer as instructed. You can find them in the `Downloads` section below the video. Follow along with Guil to completion.

Ignore the Workspaces and the questions, as well as the *Moving Forward With Sass* video at the end.

Hack Oregon

Create a static replica of the Hack Oregon website. Incorporate the grid you made yesterday! Also, I recommend using a similar file tree architecture as in the Treehouse CSS to Sass videos.

Sass + Bourbon

Watch Treehouse's Moving Forward with Sass video for an overview of some great Sass tools and resources. CSS preprocessors may be relatively easy to learn, but they take lots of practice to master. Since we won't get to cover every advanced feature of Sass in class, I suggest taking note of Guil's recommendations and exploring them at your leisure in the future.

For our class purposes, we are going to be using Bourbon with Sass. Bourbon is a nifty little library of mixins and premade tools that will help speed up our workflow. It's basically grab-n-go Sass.

Hampton Catlin explains it pretty well here (skip ahead to 3:05) and sets you right up. When you're done, check out this video as well for more neat examples of Bourbon use.

Sass Assignment #2 The Guardian, Facebook, Periodic Table

Over the next couple days, you and three other people (2 pairs total) are to build something awesome and portfolio-worthy with Sass.

Start out by making sure Bourbon is installed on your computer. Follow the “non-Rails app” instructions from the website. I recommend perusing its docs to get a feel for what tools are available to you. Remember, all Bourbon is is a collection of mixins. It doesn’t really do anything for you. It’s just there for you to get going!

With that being said, you have some flexibility for this project. You can:

a) Build a website that uses modular, dynamic components -- something like The Guardian or a Facebook page. Note how, for The Guardian, you have news 'cards' that are small with no pictures, medium cards with photos, and larger cards with videos. When resized, each type of news card is responsive in its own way. This is an excellent use case for CSS preprocessors like Sass. As per the usual, dummy HTML links are fine. Feel free to use a responsive grid system to help with layout, unless you want to `float` as you go. Bonus points if you implement Neat, Bourbon’s grid extension.

and/or

b) Build something totally unique, like this epic Periodic Table of Elements.

and/or

c) The choice is yours!

Just keep in mind the following:

- This is not only an exercise in Sass, but also Git workflow, teamwork, and project management. Be a good team player and don’t be afraid to share/challenge ideas. Use this opportunity wisely – group projects are a common talking point during interviews.
- Your objective is to explore Sass’ potential. Try to cover as many bases as you can (e.g. nesting, imports, mixins, etc.). Make sure everyone is getting enough Sass practice.
- Keep your project organized using partials. How you structure your project is really up to you. If you'd like to read detailed suggestions on how to structure your project, check out this article. Otherwise use the same architecture as the Treehouse videos.
- You are allowed to incorporate any other tools you wish (e.g. JavaScript/jQuery, Font Awesome, etc.), as long as Sass is the driving force of your stylesheets.

Sass Resources

- The Sass Way – A phenomenal source of Sass tutorials.
- Hugo Giraudel – An amazing tech writer & Sass wizard with a keen focus on Sass and author of this Sass style-guide.
- @SassNews – A twitter account managed by Stuart Robson.

Angular Assignments

Goal: Practice getting familiar with AngularJS, and in particular adding properties, using methods and Angular filters.

While working on these projects, play around again with what you can do with Angular. Since Angular provides a lot of power to making your application look awesome, make yours look awesome too. You can look up websites made with Angular to see how to manipulate visual elements here.

Angular Assignment #1 - Common for all - Coding School Attendance App

Create a site for allowing students to sign in to class for the day (similar to how Epicenter works). We want to navigate to three different views:

View 1: *A welcome screen, where I can navigate into different views.

View 2: * As a student I want to click a button next to my name so I can sign in for the day. I also want to click a button to sign out so everyone knows I've gone home for the day. NOTE: You may hard code some students if you choose.

View 3: * As a teacher, I want to go to a page to see a "Who's Here?" list of all the students who are here and not here, so I can keep track of my students.

Here are the criteria for your code.

- Use common directives, like ng-click & ng-repeat.
- Use ng-hide and ng-show to display information dynamically.
- Use a factory to share information among multiple controllers.
- Use ui-router to change states in your application.
- Use an Angular filter.
- We'll also be looking at some familiar criteria:

- Were individual partials used for all views?

- Is your logic easy to understand?
- Did you use descriptive variable names?
- Does your code have proper indentation and spacing?
- Did you include a README with a description of the program, setup instructions, a copyright, a license, and your name?
- Is the project tracked in Git, and did you regularly make commits with clear messages that finish the phrase "This commit will…"?
- We've also included some screenshots of how your site might look (these are suggestions, make it look how you want):

Angular Assignment #2 - Best Restaurants in Town

You've been hired to create a website using AngularJS to list out the best restaurants in town. Add various descriptive properties to the restaurants, like type of food, location, and price range. Play around with the filter feature of Angular to sort the restaurants by one of those properties. Also read the AngularJS API documentation and play around with adding checkboxes to your form, different types of ng-directives, etc. Really get familiar with what you can do in Angular.

Angular Assignment #3 - Online Shopper

Create a website using AngularJS for an online store. They can sell anything you and your partner choose, from apples to zebras. Add properties to the products that you are selling and then add a method that gives a 10% discount on purchases over $25. Don't worry about a login function or a checkout, but do create a way for users to add products to their shopping cart and see their total.

Don't forget to including a shipping estimator based on zip code. Since zip codes generally start on the east coast of the United States and increase in number as you move west, the difference between any two zip codes is also relative to their geographical distance. Before you code, answer the question; how might you use real-world zip-code information to build an Angular function? What other information would you need? Finally, set your cart to give your customers free shipping on any order over $50.

Angular Assignment #4 - Field trip permission

Goal: Your goal is to practice using multiple controllers on a single web page and to create and use a factory service to share data across controllers.

Field trip permission

Follow along with the video and recreate the Student Attendance List app and add in a second controller and ultimately a factory.

Angular Assignment #5 - Surveybot

Create a survey website where your classmates can answer two simple questions of your choice, perhaps about food preferences, or favorite colors to gather some random data. Use different controllers for asking questions and displaying results, but have a single factory that collects the responses. Design your page to prevent you from seeing results until you have at least five people take the survey, then have some classmates take the survey.

Remember to route your site correctly, allowing you to navigate without refreshing. If you have to refresh the webpage to return to the start of the survey you're also going to reset your model, and you'll lose all of your data.

Angular Assignment #6 - Vacation Tracker

Lots of sites have "expert picks" for the best places to eat, or see a show in a particular city. Let's build one that crowd sources that information, and collects data from users.

Make a website where you and your partner can independently enter the names of cities you have visited. At the end have the website produce a list of matching places, which you have both visited and ask you a question about it. For instance if you've both been to San Francisco, the site would ask "Since you've both been to San Francisco, where is the best place to eat?" or "Since you've both been to San Francisco, where is the best place to go hiking?" Come up with a response in case you haven't visited any of the same places. Don't forget to use different controllers for each of these functions.

Angular Assignment #7 - Courses Roster

Follow along with the Courses roster videos and create a courses roster website. This is good practice for getting familiar with how UI Router works and how to create states and routes in your application.

Angular Assignment #8 - Ask the experts

redbeacon/experts is a site written in AngularJS where homeowners and other do-it-yourselfers can post questions and get expert advice about problems they might be having. Build a similar website to answer questions about subjects you and your partner are 'experts' on. Feel free to inspect the code written on redbeacon to get an idea how they've written their site.

Don't worry about logging on, or the difference between users and experts. We'll assume that can be taken care of later.

Here are some things to include:

- As a user, I want to be able to post a question.
- As an expert I want to be able to answer a question.
- As an expert I want to be able to filter out questions that have already been answered.
- As a user, I want to approve answers, and give a rating based on how well my question was answered.
- As a user I want to be able to "upvote" the best answers I read.
- As a user, I want the links with the most upvotes to be listed first, so that I can see the best answers that have been submitted.
- As a user I want a search function, so I don't post a question someone has already asked.
- As a user, I want to comment on a link, so that I can have a discussion about the answer that an expert submitted.
- Bonus points for making it look good!

More AngularJS Resources

Here are some more resources to help you explore the great new world of AngularJS!

Videos

https://thinkster.io/topics/angular

https://www.youtube.com/playlist?list=PLP6DbQBkn9ymGQh2qpk9ImLHdSH5T7yw7

Links

https://github.com/jmcunningham/AngularJS-Learning