This class works differently from (probably) any class you've ever taken before.

- Class time is for learning the big, important ideas. You'll be reading a lot of code.
- Homework time is for learning the details. You'll be writing a lot of code.

- Notes will be provided so you don't have to take notes during class.
- These notes are sufficient for preparing for homework, the midterm, and your final project.
- When I'm demonstrating something, don't take more notes. **Your job is to think of questions.**
- You are free to jot down extra notes during the labs, during break times, and after class.
- Lecture slides (rare) are considered to be part of these notes. Get them online.
- You *will* need to look up more details online. I provide helpful links in Piazza. Oh and there's a thing now called Google.

- The code that I write in class is always available for review.
- **99% of what you'll need to solve the homework problems will be in my code.**

Consider The Following…

Professional developers make their money by knowing how to invent their own solutions through research, not by memorizing specific solutions.

They have learned how to think. Not how to copy.

- When you Google: try to use specific, uncommon words; or, paste an entire error message.
- Filter results to only the past year or shorter (click Search Tools on the right side).
- You can trust answers from rubyonrails.org, ruby-lang.org, and stackoverflow.com.
- Read error messages.
- Watch out for *singular vs. plural* words: movie vs. movies, user vs. users, etc.
- Make sure that blog or tutorial you're reading is for Rails 4+ and Ruby 2+
- **Don't believe online tutorials** (using code generators, etc.) until after this course.
- Just Try It™. You will learn more by experimenting than by thinking or searching or reading.
- Write as much code as you can. Make up tiny problems for yourself and try to solve them.
- Watch your Rails server log.
- Read error messages.
- Always look for cause and effect. There is a reason for everything.
- Got code that works but not sure why? Intentionally break the code, and see what changes.
- Be curious about everything.
- Stuck for more than five minutes? Ask the ducky™.
- What's fuzzy? What's crystal-clear?

Consider The Following…

If you want to become a professional developer, this course is a great start, but you'll have to keep learning more, after this course is over.

What should you learn now so that you can keep learning after it's over?

- Use Google Chrome.
- Learn how to view the source code of every page you browse.
- Learn how to activate the web inspector.
- HTML uses **<** and **>** to separate *elements* from *data*.
- HTML elements have a general structure consisting of *tags*, *attributes*, and *content*.
- HTML is for specifying content, not visual appearance. Browsers have a default opinion of how to draw each type of content, making people think that twiddling with the HTML is the way to control the visual appearance, but *that's not correct*.
- Block-level elements will always be displayed on their own line.
- Inline elements are streamed inline with the content that surrounds it.
- Every HTML element becomes a rectangle on the screen. Read that again.
- CSS is a language that lets you define visual appearance: colors, fonts, spacing, layout, borders.
- CSS works by letting you define a series of *rules* that select content on the page and apply certain visual features.
- CSS rules intentionally intertwine with each other, so that you can combine a small number of rules in lots of different ways.

```
<a href="http://www.google.com">Search</a>
```

```
a { text-decoration: none; color: purple }
```

Consider The Following…

Your browser draws ("renders") a page by displaying the HTML content and painting it by following CSS rules.

If you don't provide any CSS rules, what rules should it use?

- Get your Rails environment setup this week!
- HTTP is a request-response protocol
- HTTP packets consist of a header (list of key-value pairs), plus optional body content
- The Chrome Inspector is a great place to learn about HTTP, HTML, and CSS
- Ruby is a dynamically-typed, strongly-typed, object-oriented language
- You should get familiar with Ruby strings, symbols, arrays, and hashes
- Ruby uses **do**...**end** blocks to streamline the most common design patterns
- You should learn how to to loop over an array with the **each...do...end** pattern
- Ruby uses the **require** statement to activate extra functionality
- Ruby has built-in support for JSON parsing and HTTP access
- Ruby classes can contain *instance* methods and *class* methods
- Common doc notation: #instance_method ::class_method
- Lots of ways to generate random data: rand(), Array#sample, etc.
- Output to the screen with **puts**
- Use **irb** to use Ruby interactively, and to learn everything you need to know.
- Helpful for detective work: **.class** and **.inspect**

```
favorites = ["hockey", "cookies", "purple"]

favorites.each do |word|
  puts "I like #{word}"
end
```

Consider The Following...

Think of an app you use a lot.
Does it rely on hardcoded data?
If not, where does the data come from?
What happens to the data when you close the app?