Intro to HTML

Entry Ticket

To start this, make sure you're familiar with creating files and directories in terminal. Also make sure you know how to open files on your hard drive from terminal, with the `open` command, as well as how to open files in Sublime Text 3 with the `subl` command.

Materials

For this learning experience, you'll need to have the following installed:

- Sublime Text 3
- Chrome

It's also important to have the Chrome set as your default browser, and to have the `subl` command set up by following the setup instructions here.

Objectives

By the end of this lesson you should be able to:

- Create HTML pages from scratch
- Style them with CSS
- Link them to other documents, images, or other types of files

In other words... create a document you can put on the web and share with others!

Coursework

Introduction

What is HTML?

HTML is a subset of a language called XML. Anytime you see a webpage in a browser, HTML is the language telling the browser what content to put on the screen.

- HTML *describes* and applies *structure* to a page; it's the skeleton.
- Browsers *parse* and then *render* the HTML so that it's human-readable.

Syntax

Tags Make Elements

You've probably run into tags before if you've ever used the Internet. You've seen paragraph tags - `<p>`, or you've run into `<span>` or `<div>`. You may have used `<a href="http://example.com">Click here</a>` before to link to something.

Tags are the basic unit of HTML. Think of tags (anything that starts with < and ends with >) as *boxes*. The words in between the angle braces (> & <) are like labels to tell you what the box contains.

HTML *tags* are used to wrap *content*, by which we usually mean text or other tags.

```
<p>Some text.</p>
```

The tags above are `<p>` & `</p>`, and the content is "Some text." The *opening tag*is the `<p>`, and the *closing tag* is the `</p>`. These are the bounds of the box, they define it's beginning and end.

When tags are read by the browser, they form an HTML `element`. The use of `p` tags surrounding text above creates an HTML element.

Self-closing tags

Not all tags need a beginning and an end, some tags are self-closing and do no need to wrap content. This is because they aren't thought of as enclosing anything. Think of them as boxes that are already closed and taped up, so there's no need to close them.

```
<img src="foo.com/thing.png">
```

```
<input value="Input Here!">
```

```
<hr>
```

Attributes

The above examples contain *attributes*, which are more information attached to an *element*. These attributes are usually to generate content or to act as a reference for other technologies like CSS & JS.

This image (`<img>`) has a *source* and *alternate* attribute. The source (`src`) attribute tells the element what image file to display, and the alternate (`alt`) attribute tells the element what text to display if the image can't be displayed for some reason.

```
<img src="foo.com/thing.png" alt="A cool image of a foo thing.">
```

This *anchor* (`<a>`) link has a hypertext reference (`href`) attribute. The reference tells the link tag where to take you when clicked.

```
<a href="/contact">Contact Page</a>
```

This title has a `class` attribute. It tells the browser what styles to apply to the element.

```
<h1 class="content-title">You Will Never Guess What This Puppy Does With Her Brunch.</
```

Structure

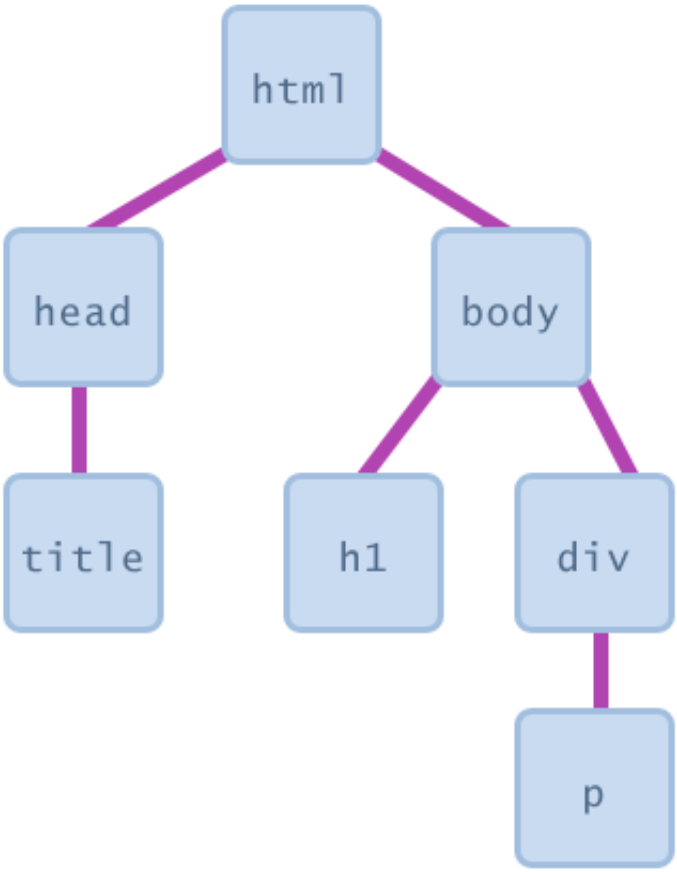
XML can be thought of as a tree structure, which is similar to a family tree. Each element has a *parent*, it sometimes has *siblings* and it also may have *children*. Imagine a large box that you put smaller boxes in- the smaller boxes are contained within the larger box. They can't be in multiple boxes at once, so the larger box can be thought of as their *parent* or *container*. The smaller boxes can contain still smaller boxes, aka *children*. Another way to think of this is to imagine making a family tree for a group of jellyfish. They reproduce asexually, so they each have only one parent.

Here's an example:

```
<html>
    <head>
        <title>The title is nested in the head</title>
    </head>
    <body>
        <h1>This header is nested in the body>
        <div>
            <p>This paragraph is nested in the div tag above it, which is itself neste
        </div>
    </body>

</html>
```

You'll note that it's easier to visualize what is contained within what because we're using tabs to indent every time we open up a new tag. This isn't important for the computer to read and render the code you write, it's to make the code easier for a programmer to read and use.
This image is how the above XML can be visualized.



Exercise 1 - Our First Page

HTML pages always have the following structure:

- HTML `<html>`
  - head `<head>`
    - links to CSS stylesheets, some javascript links, meta-data, the `<title>` tag
  - body `<body>`
    - page content `<h1>`, `<p>`, `<div>`, `<ul>`, `<li>` etc.

Let's create a simple HTML page called `index.html` in a directory called `html-exercise`, then open it. **Do the following in your terminal, rather than using Finder.app.**

```
$ mkdir html-exercise
$ cd html-exercise
$ git init
$ touch index.html
$ open index.html
```

Chrome opens the `index.html` file, and because of the `.html` extension, it knows to present it to us as a normal webpage. Looking in chrome though, we have a blank screen! Pretty useless. Begin by creating the root level element, `<html>`. Make sure to close it with the corresponding `</html>` tag. Next, in the same fashion, create a `head` and a `body`. Use the example above if you're not sure what it should look like.

> Now is a great time to `git add` your files, and `git commit` them. Remember to add the `-m` command and add a commit message.

Next we're going to dive into some documentation. Take a look at thedocumentation for the paragraph tag- there is a description as to what all the parts do, and several examples. If you copy-paste these examples, you'll notice they show up on the screen.

Using Mozilla Developer Network Element Reference we're going to create your first page. *Keep this page open as you code*- real developers don't try to memorize things, that's why we have google and documentation!

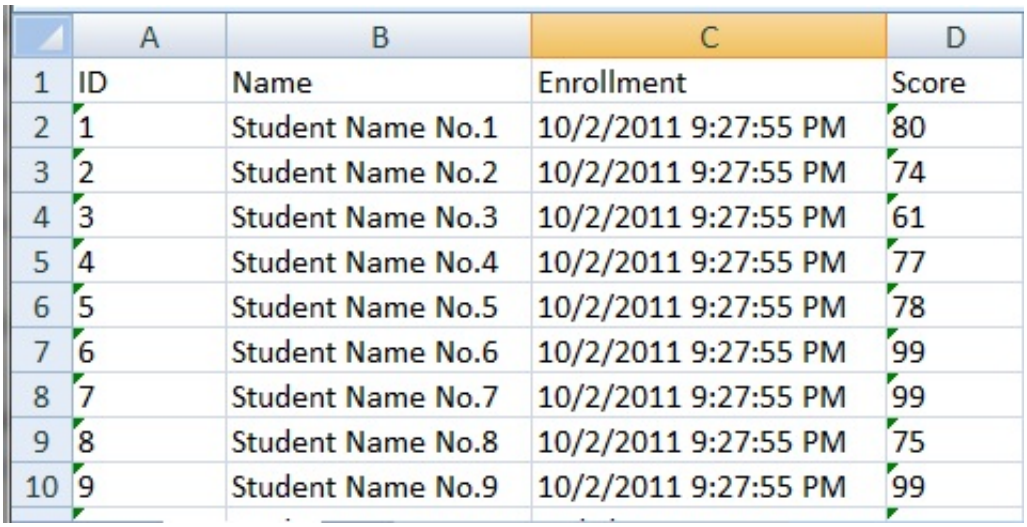
- Make an unordered list all of the cities you've lived in (then, commit your changes!)
- Make an ordered list of the top three cities you would like to visit (then, commit your changes!)
- Add a heading before each list that says what the list is (then, commit your changes!)
- Add a description under the headings of each list explaining how you came to live in each of those cities, and why you would like to visit those cities respectively. (then, commit your changes!)
- In the list of cities you've visited, add a span at the end of each city with any HTML character entity. (then, commit your changes!)
- In the second list, make every odd number strong, and every even numberemphasized. (then, commit your changes!)
- Finally add the following image somewhere in the page, and give it a height and width attribute with number values:https://students.galvanize.com/assets/galvanize-logo-ac9865cc4217b77aebbce9e63670dd96.svg (then, commit your changes!)
- **BONUS** add audio and video to the page, find them on MDN. Give it controls and make the video autoplay. (then, commit your changes!)

audio:https://upload.wikimedia.org/wikipedia/en/0/04/Rayman_2_music_sample.ogg

video: https://upload.wikimedia.org/wikipedia/en/2/28/Illusion_movie.ogg

Tables

Tables are how we display "tabular data" in HTML. What this really means is something like this:



	A	B	C	D
1	ID	Name	Enrollment	Score
2	1	Student Name No.1	10/2/2011 9:27:55 PM	80
3	2	Student Name No.2	10/2/2011 9:27:55 PM	74
4	3	Student Name No.3	10/2/2011 9:27:55 PM	61
5	4	Student Name No.4	10/2/2011 9:27:55 PM	77
6	5	Student Name No.5	10/2/2011 9:27:55 PM	78
7	6	Student Name No.6	10/2/2011 9:27:55 PM	99
8	7	Student Name No.7	10/2/2011 9:27:55 PM	99
9	8	Student Name No.8	10/2/2011 9:27:55 PM	75
10	9	Student Name No.9	10/2/2011 9:27:55 PM	99

Any time you have something that would be good in a spreadsheet, Tables are the way to go.

> Early in the history of the web, people tried to adapt tables for layout purposes. Today, we have CSS Grid Systems and tables are back to being used for their actual purpose. This is one of the first examples of how code can be "abused"- AKA used for a purpose other than what it was designed for. Tables worked well when you could be sure of what size everyone's screens were (remember when monitors were all the same size?) but nowadays they just break when viewed on a small screen (like a phone) or a big one (like a giant iMac monitor).

Table tags: `<table>,<thead>,<th>,<tbody>,<td>,<tr>,<tfoot>`

Take a look at the documentation for Tables. Keep this documentation up, and use it to find out how to use the elements you need in order to complete the exercise below. Table Documentation

Exercise

Create a new file, call it `favorites.html`. `git add` the file to your repository, and commit it.

- Set up the page with the proper structure, like the last exercise
- Create a table and add a table head element (then, commit your changes!)
- Create a row of table headers for `First Name`, `Last Name`, `Favorite Animal`
- Create a table body with one row and three columns
- Enter your first name, last name, and favorite animal in the corresponding columns (then, commit your changes!)

Assignment: File at least one Pull Request to a neighbor's page containing your information.