Lesson **Weekend**

Workshops (/workshops) / Intro to Programming Workshop (/workshops/intro-to-programming-workshop) / Styling with CSS

Text

Now that we've added content to our webpage using HTML, we can style and format this content using something called **CSS**.

Using CSS

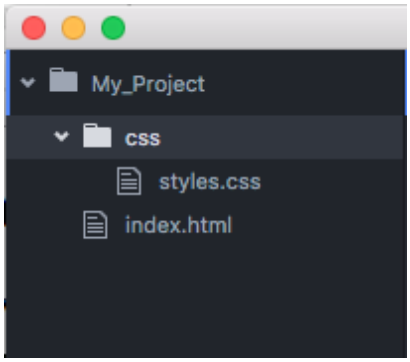
CSS stands for **Cascading Style Sheet**. Just as HTML defines *what* appears on our page, CSS defines how it *looks*.

Let's begin by creating a new style sheet from our Terminal.

Use `pwd` to verify that your Terminal is within the project folder, and enter the following commands:

- `mkdir css` This will create a new folder called *css* where we can store our CSS.
- `touch css/styles.css` This will create a new CSS file within the *css* directory.

Back in Atom, we see that our *css* folder exists and contains our new file.



Open the *styles.css* file, and let's write a styling rule.

css/styles.css

```
h1 {
  color: blue;
}
```

This style can be broken down into a few sections.

- `h1` tells our CSS that this style should be applied to any element of this type. It will be universal for all `<h1>` elements on our page.
- `{...}` These are called **curly-braces** by most programmers. Some call them *handlebars* or *wave-braces* or other things, but they define the beginning and end of the style we're trying to define. Every rule within these symbols will be applied to the `<h1>` elements.
- `color:` This tells the rule what **property** of the element we're trying to change. In this case, we're trying to change the color.
- `blue;` The tells our page what **value** we want that property to have.

You can combine multiple rules in each style. Let's add another rule to this style.

css/styles.css

```
h1 {
  color: blue;
  background-color: gray;
}
```

This rule will change the background color of the `<h1>` elements.

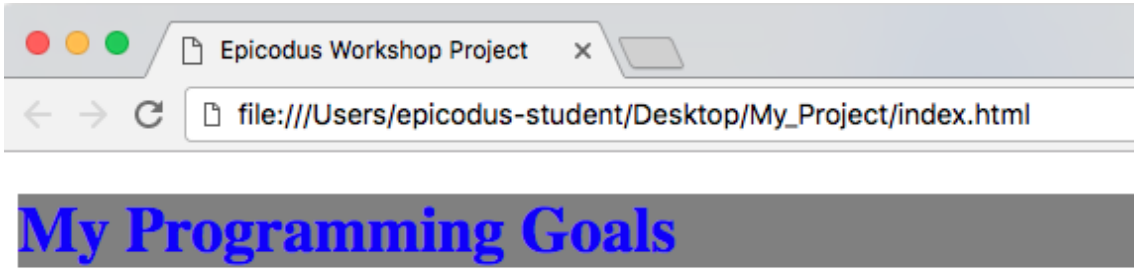
By the way, you can spell gray either way, gray/grey CSS knows there are two spellings.

We need to do one more thing before we'll see any changes on our page. We need to tell the HTML to use these styling rules. Let's reopen *index.html* in Atom, and add this line to the `<head>` of our HTML.

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

This `<link` element in the `<head>` tells the HTML to find the *styles.css* file and apply it to the code it's going to render from this HTML file. Pretty neat, right?

Let's take a look at our page. We can see that the title has been styled!

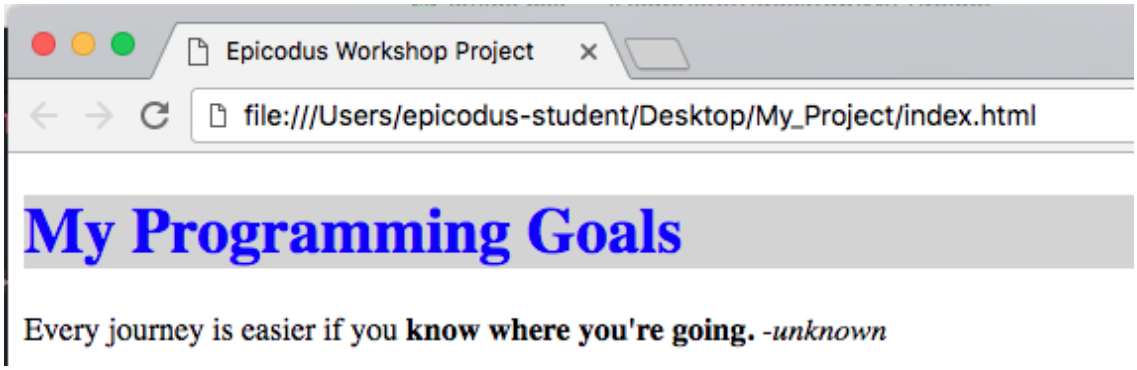


It may not be the prettiest style, but we certainly have changed it. In fact, that gray looks a little too dark. Let's change the style.

css/styles.css

```
h1 {
  color: blue;
  background-color: lightgray;
}
```

That's better, and easier to read.

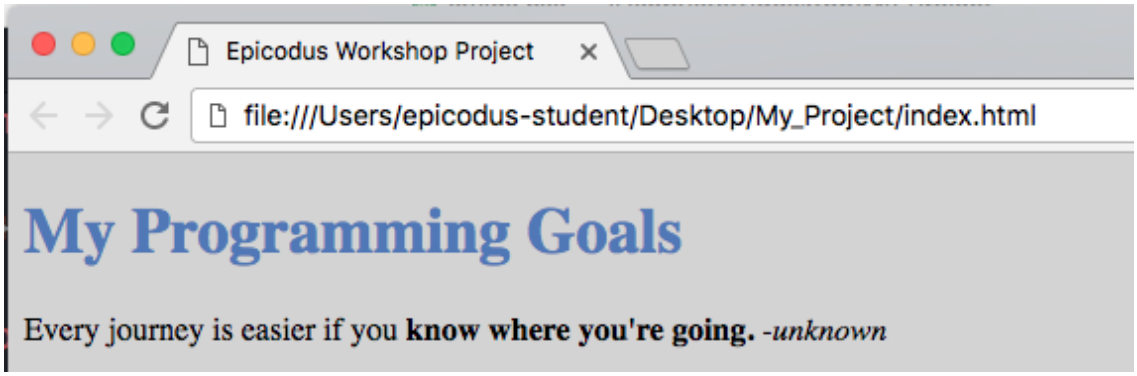


You can look up a list of all the colors CSS has defined, or use a **hex-color**. Let's use a hex-color to change the text to a lighter blue. And while we're at it, let's put that background color behind the entire body of the page.

css/styles.css

```
h1 {
  color: #5179ba;
}

body {
  background-color: lightgray;
}
```



There, it's starting to look better. Spend a little time using this system to style some of the other elements on your page.

Using a class or id

So far we've been styling everything based on it's element type. An `<h1>`, or a `<p>` element, for example. But what if we want to target only a single element, or group of elements ignoring others of the same type?

We can use two powerful tools in HTML to do this. The **class** and the **id**.

- An **id** is a single identifying attribute, given to an HTML element.
- A **class** is an attribute given to multiple HTML elements.

Let's modify our list of goals in the HTML to see how this works. We'll add a few more items, and give classes and ids.

index.html

```
...
<ul id="goal-list">
  <li class="list-item">Learn HTML</li>
  <li class="list-item" id="css-item">Learn CSS</li>
  <li class="list-item">Learn JavaScript</li>
  <li class="list-item">Learn Git</li>
</ul>
...
```

Let's also add an `id` to our image.

```
...
<img id="picture" src="https://tinyurl.com/lfkubwe" />
...
```

Now that we've assigned some ids and classes, let's write some styles for them and see the effects. Add these styles to your CSS file.

Note: * We use a `#` to tell the CSS that we're referring to an *id* * We use a `.` to indicate that we're targeting a *class*.

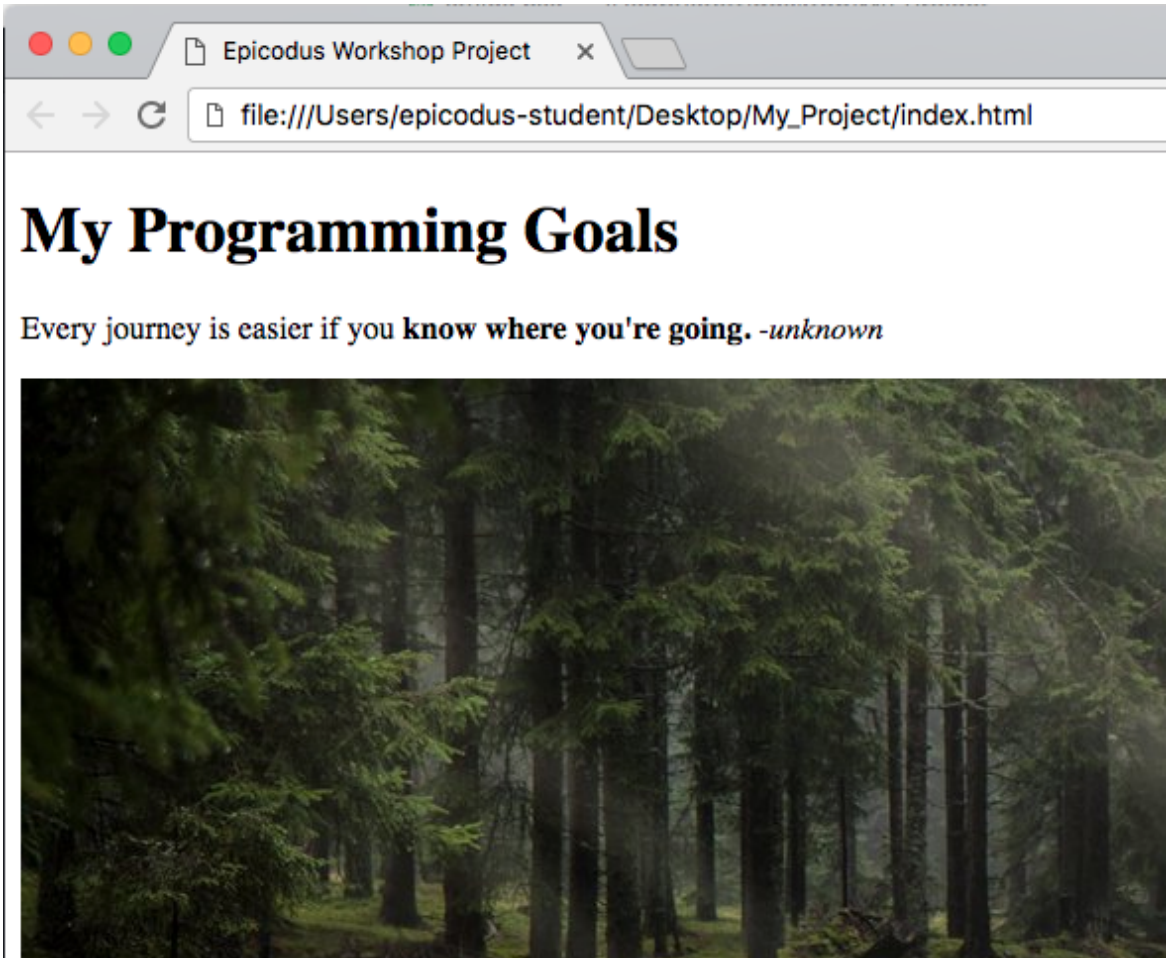
css/styles.css

```
.list-item {
  font-family: monospace;
}

#css-item {
  color: #5179ba;
}

#goal-list {
  border-style: solid;
  border-color: #5179ba;
  width: 30%;
}
```

Reload your page, and see how the styles have been changed.



My Goals:

I want to become a professional web developer. Here are the things I'll need to learn first.

- Learn HTML
- Learn CSS
- Learn JavaScript
- Learn Git

My favorite website

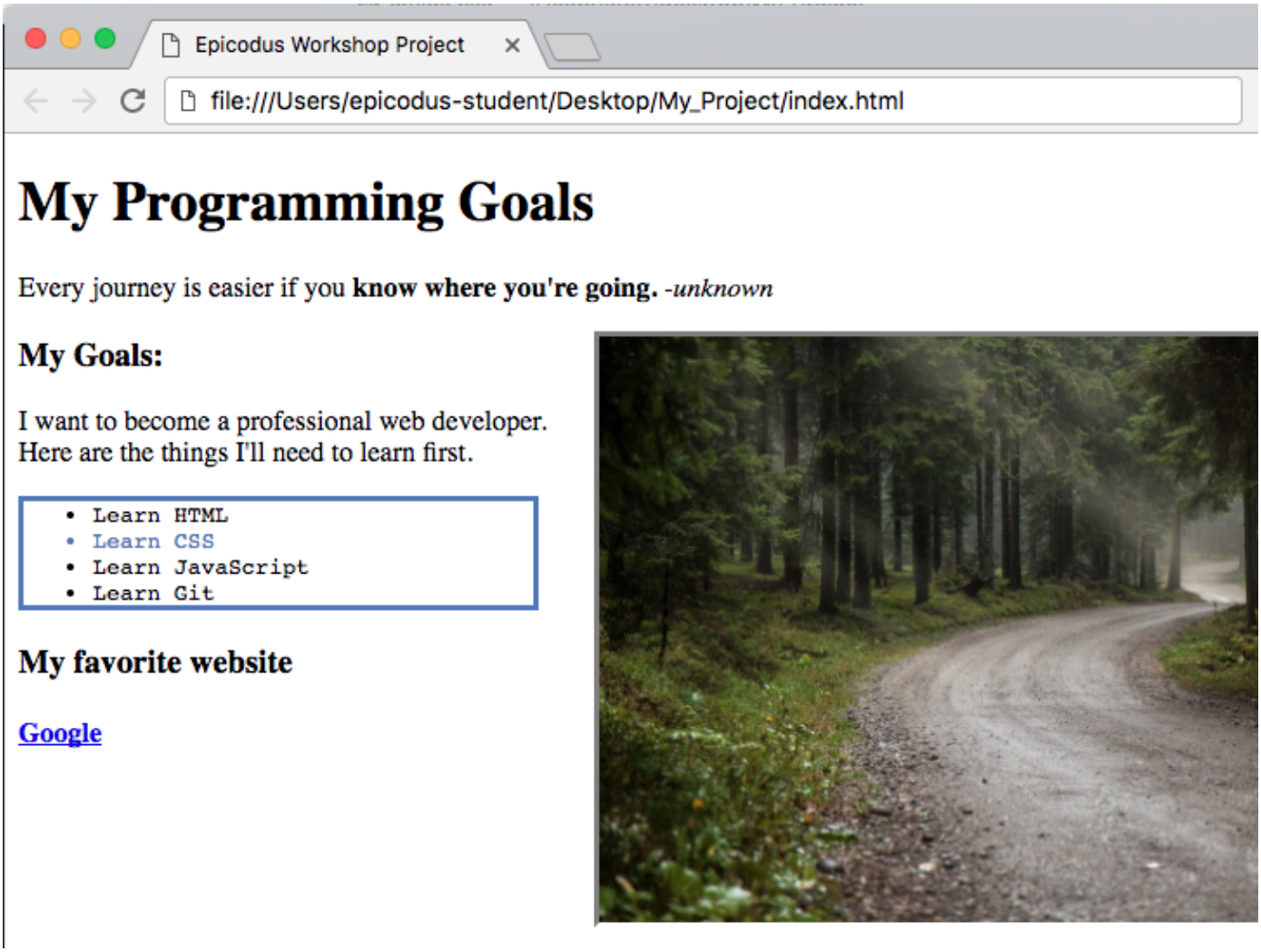
Google

Finally let's move and style that image a little bit. Give it the **id** `picture`. Now add a new style.

css/styles.css

```
...
#picture {
  width: 60%;
  float: right;
  border: solid;
  border-color: grey beige white grey;
}
...
```

Now that we've finished that, the page should look a little more designed.



There is a lot to learn when it comes to CSS and design, and we've barely touched the surface here. If you'd like to learn more you can access many great resources online, view our in-depth CSS curriculum (https://www.learnhowtoprogram.com/css) or even enroll in the CSS/Design course at Epicodus (https://www.epicodus.com/).

Using Bootstrap

CSS is a lot of work, takes attention to detail, and a keen eye. Sometimes we'd like to mock up a site, or simply get something looking better with a little less work. There are many existing CSS libraries out there to help us style our sites more efficiently.

One of the most common in called **Bootstrap**, built by twitter. Let's add it to our project and make a few minor changes to make our site look even better.

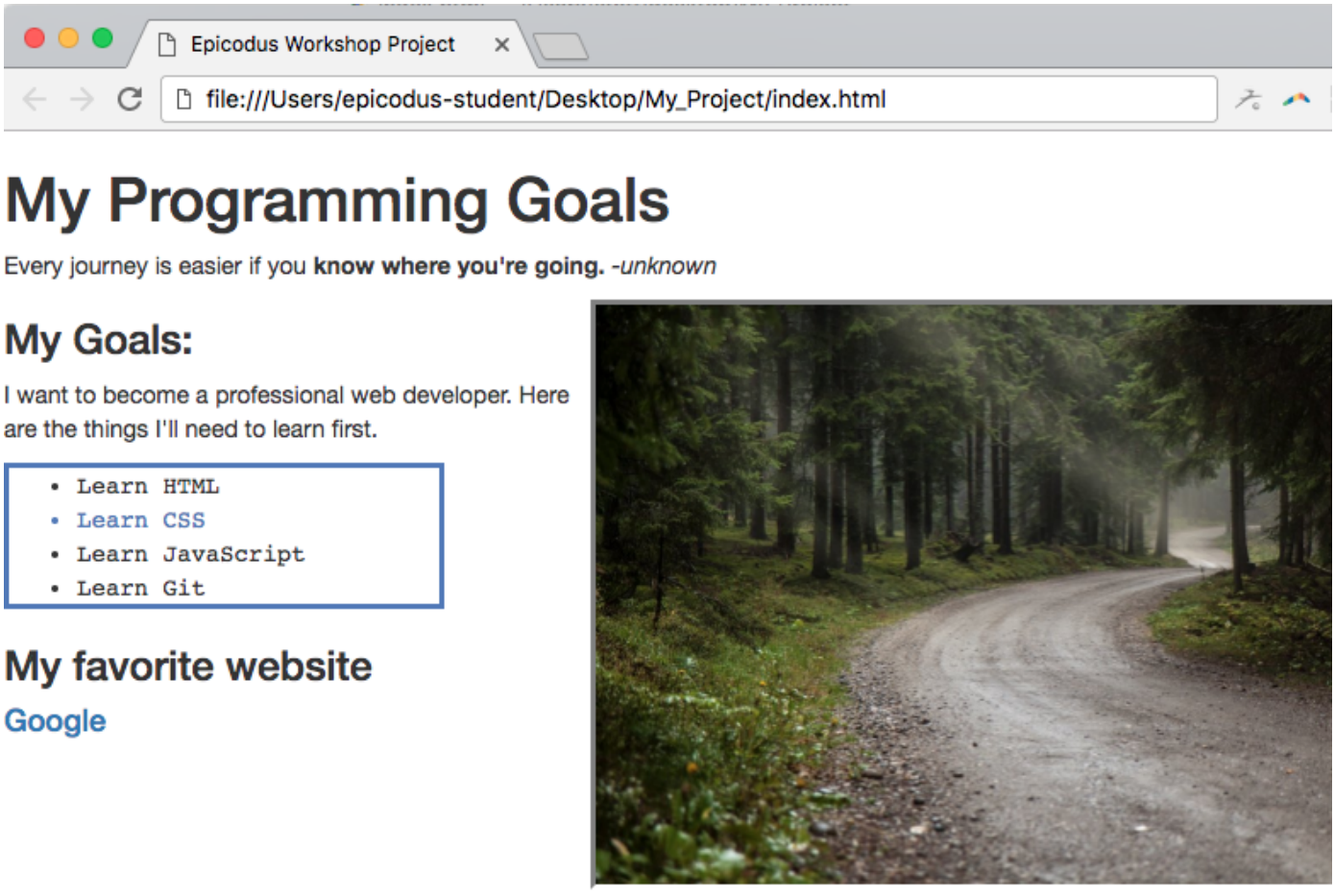
In our HTML, just above where we linked our *styles.css* file, let's add a new line of code. It should look like this.

index.html

```
...
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstra
p/3.3.7/css/bootstrap.min.css">
<link rel="stylesheet" href="css/styles.css">
<title>Epicodus Workshop Project</title>
...
```

This is a css file, just like the one we wrote, but contains thousands of styles and rules, and is located online instead of inside our project folder. It also defines special classes we can use on our html. Let's do that now.

When we refresh our page, things look a little different.



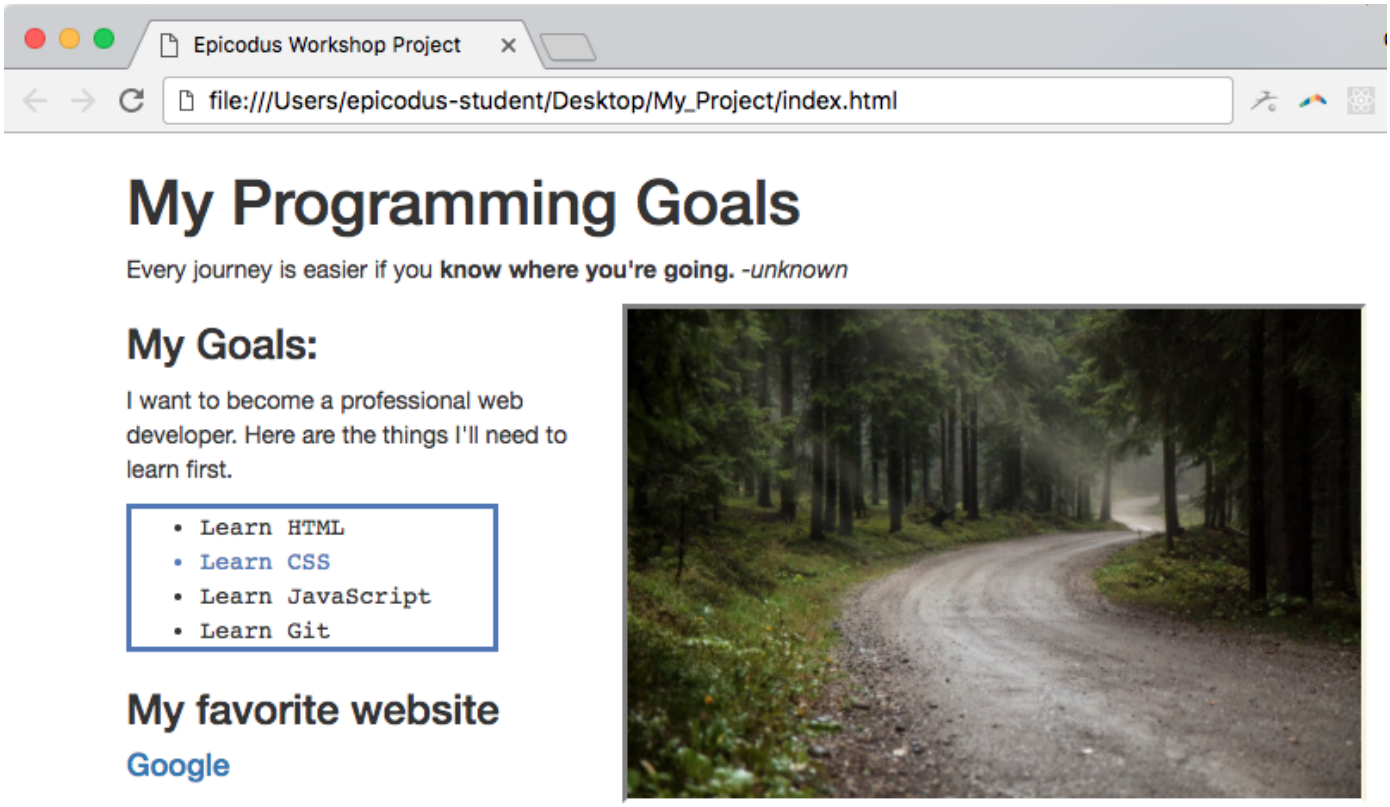
One problem that jumps out is that the text is very tight on the left of the screen. Let's use a bootstrap class to fix that.

We're going to use a new HTML element as well, called a `<div>`. A `<div>` makes divisions on the page and is used to group parts of our page together. Let's put all the code within the `<body>` within a `<div>`. We'll also give it a bootstrap class, `container`.

index.html

```
...
<body>
  <div class="container">
    ...
  </div>
</body>
...
```

When we go look at our refreshed page, we can see how bootstrap fixed a lot of our spacing, and gave the page a nice look.



If you want to see what the page looked like before and after we added this styling, you can comment out the styling links in the HTML head. Commenting code is a way to leave it on the page, but not have it rendered. If you highlight a line of code and press `command + /` on your keyboard, you can see the code will be grayed out within Atom. Try this, and reload your page. To un-comment the code, simply repeat the process.

Let's move on to make our webpage a little more interactive.