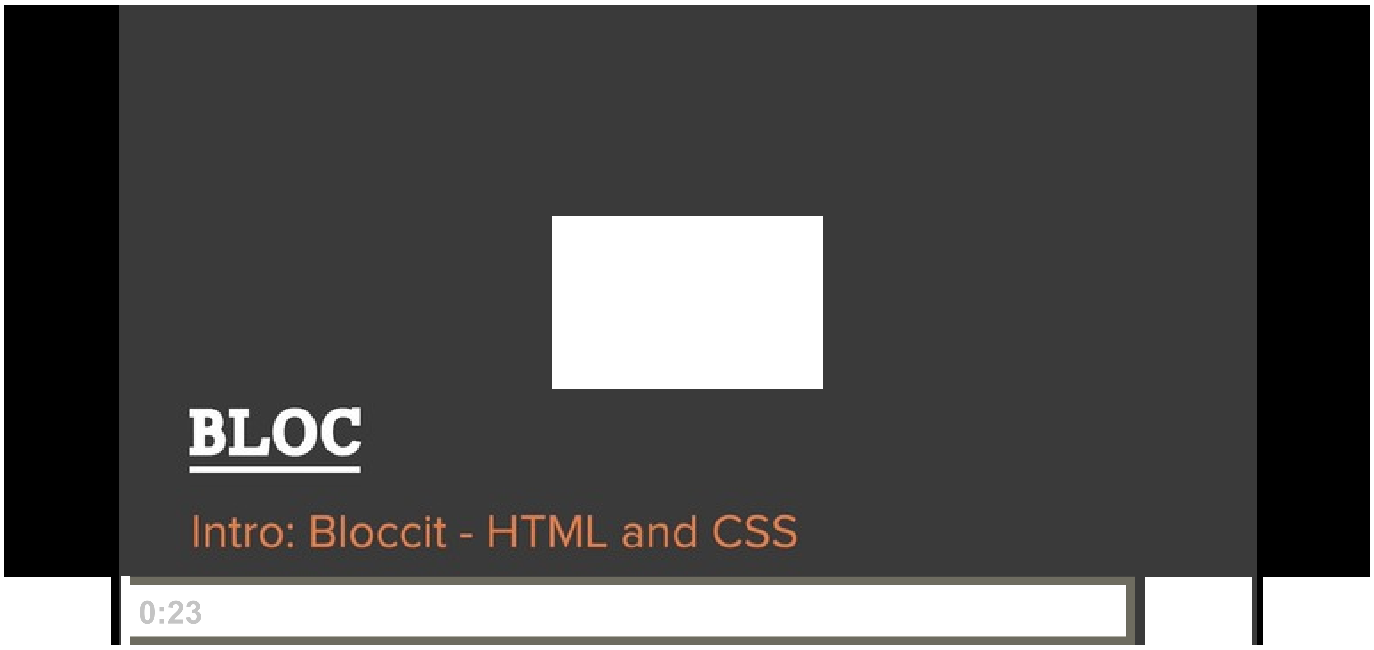
# 27 Rails: HTML and CSS

> “We never go out of style.”
>
> — Taylor Swift

## Structure and Style

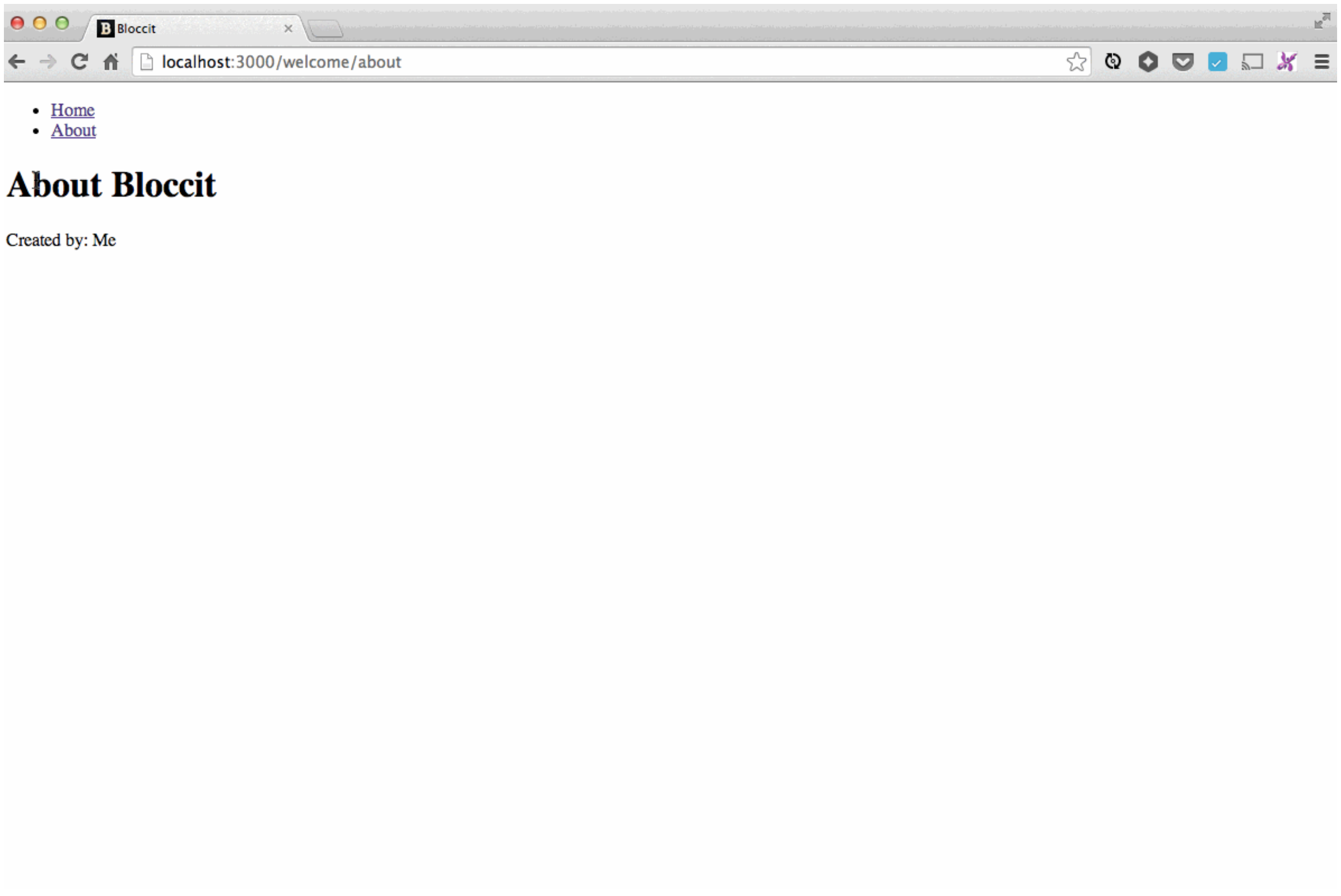


The two basic building blocks of web development are HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets). They provide the structure (HTML) and the style (CSS) for all web pages.

## Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

## HTML



- Home
- About

About Bloccit

Created by: Me

HTML is code that a browser can interpret and display as a web page. HTML by itself is not dynamic, but when used in conjunction with a language like Ruby and a framework like Rails, it can be *rendered* dynamically. Based on the type of request made by a user, an app will respond with different HTML.

## Modifying HTML

You've already seen the basic HTML generated by the `rails generate controller` command. For example, your `index` view looks like this:

app/views/welcome/index.html.erb

```
<h1>Welcome#index</h1>
<p>Find me in app/views/welcome/index.html.erb</p>
```

Modify the **index** view with custom HTML:

app/views/welcome/index.html.erb

```
- <h1>Welcome#index</h1>
- <p>Find me in app/views/welcome/index.html.erb</p>
+ <h1>Welcome to Bloccit</h1>
+ <p>This is the home page for Bloccit.</p>
```

> Try not to copy and paste. Writing code will enhance your cognitive and muscle memory.

Modify the **about** view in the same fashion:

app/views/welcome/about.html.erb

```
- <h1>Welcome#about</h1>
- <p>Find me in app/views/welcome/about.html.erb</p>
+ <h1>About Bloccit</h1>
+ <p>Created by: Me</p>
```

An HTML tag is composed of a term, such as `h1` or `p`, written between brackets `< >`. Text between HTML tags will be formatted according to their tag specification. Most HTML tags must be closed with `</ >`.

## HTML, Ruby, and `application.html.erb`

The **index** and **about** views have an `.html.erb` file extension. This file extension allows us to use HTML and Ruby (ERB stands for "embedded Ruby") in the same file. By integrating Ruby code with HTML, we can dynamically change the behavior of static HTML code, based on user input.

Nearly every view in a Rails application will have some unique HTML and Ruby code, but there is also common code that needs to be included in *all views*. Rather than repeating the same code in every view, we use `application.html.erb`. Each view (like **index** and **about**) is called from, and rendered inside, `application.html.erb`. In this way, you can think of `application.html.erb` as a container file that has HTML and Ruby code needed to run every view in a Rails app.

To help you understand the rendering process in `application.html.erb`, consider the following actions:

1. A user requests a view
2. The controller corresponding to the requested view invokes `application.html.erb`
3. `application.html.erb` inserts the appropriate view using `yield`

4. The complete web page is rendered and returned to the user

`yield` is used to invoke a block, which renders a given view inside the `application.html.erb` container. Code between `<%` and `%>` is interpreted as Ruby. If the `<% %>` contains an `=`, such as `<%= %>`, the result of the Ruby code is printed to the screen (i.e. rendered as HTML). If there is no `=` (only `<% %>`) then the Ruby code will be *executed* but not printed.

Modify the `application.html.erb` file to include additional HTML and Ruby that we need in every view in Bloccit:

app/views/layouts/application.html.erb

```
  <!DOCTYPE html>
  <html>
  <head>
    <title>Bloccit</title>
    <%= stylesheet_link_tag    "application", media: "all" %>
    <%= javascript_include_tag "application" %>
    <%= csrf_meta_tags %>
  </head>
  <body>
+   <ul>
+      <li><%= link_to "Home", welcome_index_path %></li>
+      <li><%= link_to "About", welcome_about_path %></li>
+   </ul>

    <%= yield %>

  </body>
  </html>
```

The `<ul>` and `<li>` tags render content as bulleted lists; (`ul`) stands for "unordered list"). Inside of the `<li>` (list item) tags, you called Rails' `link_to` method. `link_to` is a helper method available in views, and returns a valid HTML hyperlink (called an **anchor tag**). For example:

```
<%= link_to "Home", welcome_index_path %>
```

Will render the following HTML:

```
<a href="/welcome/index">Home</a>
```

## Rails Helper Methods

`link_to` is a Rails helper method that returns a string of HTML code. `link_to` takes two arguments, a string (`"Home"`) which will be the display name of the hyperlink, and a path (`welcome_index_path`). `welcome_index_path` is a Rails method, generated by the `routes` file. Type `rake routes` on your command line again, and you'll see that the route name in the first column is `welcome_index`. When you add `_path` to the route name, it's recognized as a helper method that returns `"/welcome/index"`.

> We could have typed `link_to "Home", "/welcome/index"` instead. The resulting link would have been identical, but Rails helpers are generally easier to use, and idiomatically correct.

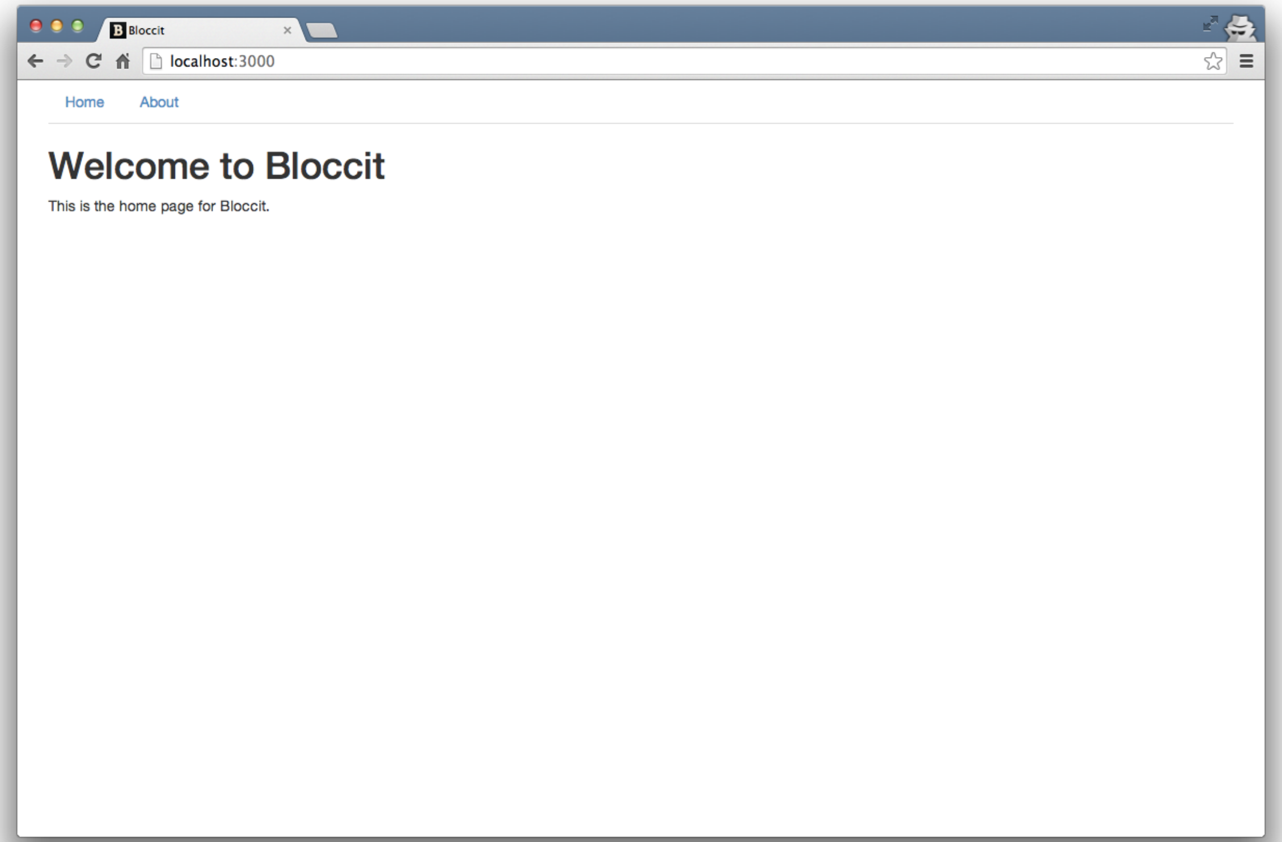
Start the Rails server and make sure that you are directed to the correct pages when you click the "Home" and "About" links.

Terminal

```
$ rails s
```

> Rather than starting and stopping your server frequently, you can open multiple tabs in your terminal. You can leave one open for Git and Rails commands and one for the Rails server. Remember, if you change a route or a config file, or add a new Ruby gem, you'll need to restart your server. For all other changes - like changes to controller, view, or model files, a restart is *not* necessary.

# CSS



Welcome to Bloccit

This is the home page for Bloccit.

CSS (Cascading Style Sheets) is the language used to style and position HTML elements on a web page. CSS has a simple syntax with three different types of elements: selectors, properties, and values.

## CSS Selectors

A CSS selector designates the HTML tag to be modified. Let's use CSS to modify the font color of `<h1>` tags :

app/assets/stylesheets/welcome.scss

```
  ...
+ h1 {
+   color: red;
+ }
```

View the **index** and **about** pages. The content between the `<h1>` tags should be red now. There are a few important things to note about the code above:

1. The CSS we added was in the `welcome.scss` file. When we ran `rails generate controller`, `welcome.scss` was created. By Rails convention, each controller has a corresponding stylesheet and view.
2. Similar to the `.html.erb` extension, the `.scss` extension provides us with some additional syntax options (known as **Sass**) to enhance default CSS capabilities. Sass is not a core part of this program, but you'll have a chance to experiment with it in the projects phase with your mentor, if you choose to do so.
3. We wrote a CSS *selector* and *declaration*. The selector, `h1`, specifies which element to modify. The declaration, composed of a *property* (`color`) and *value* (`red`), specifies how to modify it. We turned all `<h1>` tags red by using the `h1 { }` selector.

There are two other ways to select HTML elements: `id` selectors and `class` selectors. HTML elements can have `classes` and `ids`. Classes are used to conceptually group elements for styling while IDs are used to uniquely identify elements. We can use either for CSS styling:

```
<h1 class='class-name' id='id-name'>Content</h1>
```

To select those elements for styling, we use `.` to refer to a class name and `#` to refer to an ID name. To see how this works, let's create an ID selector, named "index-title":

app/assets/stylesheets/welcome.scss

```
  h1 {
    color: red;
  }

+ #index-title {
+   text-align: center;
+   font-size: 15px;
+   color: blue;
+ }
```

We'll use the "index-title" ID selector to style a `<p>` element:

app/views/welcome/index.html.erb

```
  <h1>Welcome to Bloccit</h1>
- <p>This is the home page for Bloccit.</p>
+ <p id="index-title">This is the home page for Bloccit.</p>
```

We specified `id="index-title"` after the `p` tag. The `id` can be named anything, but must match the name used in the CSS. Refresh the **index** view to see the newly styled HTML.

Now let's use a class selector to select a group of elements that should be styled the same way:

app/assets/stylesheets/welcome.scss

```
  h1 {
    color: red;
  }

  #index-title {
    text-align: center;
    font-size: 15px;
    color: blue;
  }

+ .posts {
+   border: solid;
+   margin: 5px;
+   padding: 5px;
+   height: 200px;
+ }
```

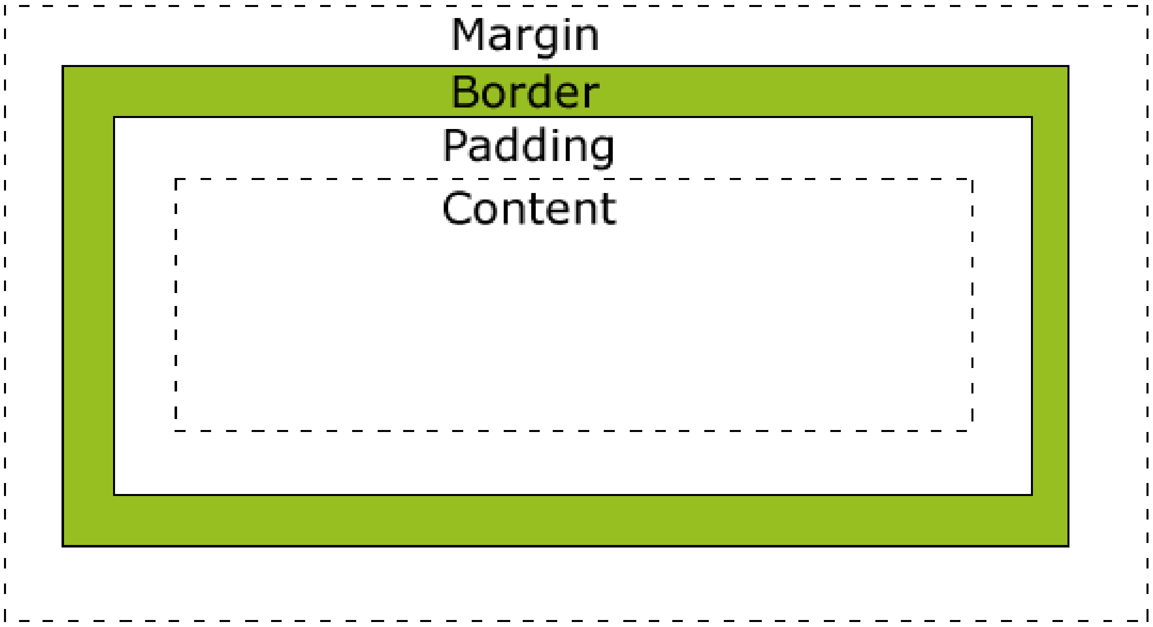
The `.posts` class will apply to any element with the corresponding `class` designation. Update the **index** view to use the `.posts` class:

app/views/welcome/index.html.erb

```
  <h1>Welcome to Bloccit</h1>
  <p id="index-title">This is the home page for Bloccit.</p>
+ <div class="posts">Post 1 goes here.</div>
+ <div class="posts">Post 2 goes here.</div>
```

## The Box Model

CSS can affect more than the color and size of elements, it can also position them. CSS positioning is based on the Box Model:



The Box Model applies to any HTML element, for which each component of the box can be set using CSS. Add the CSS code below, which uses a tag selector to define some box styles for `section`:

app/assets/stylesheets/welcome.scss

```
 ...
+ section {
+   background: #5998ff;
+   border-style: solid;
+   border-width: 5px;
+   margin: 5px 5px 5px 5px;
+   padding: 5px 5px 5px 5px;
+ }
```

We made a thick, solid border with the `border-style` and `border-width` declarations so we can easily see what our box stylings are accomplishing. Then we set the `margin` and `padding` for our section.

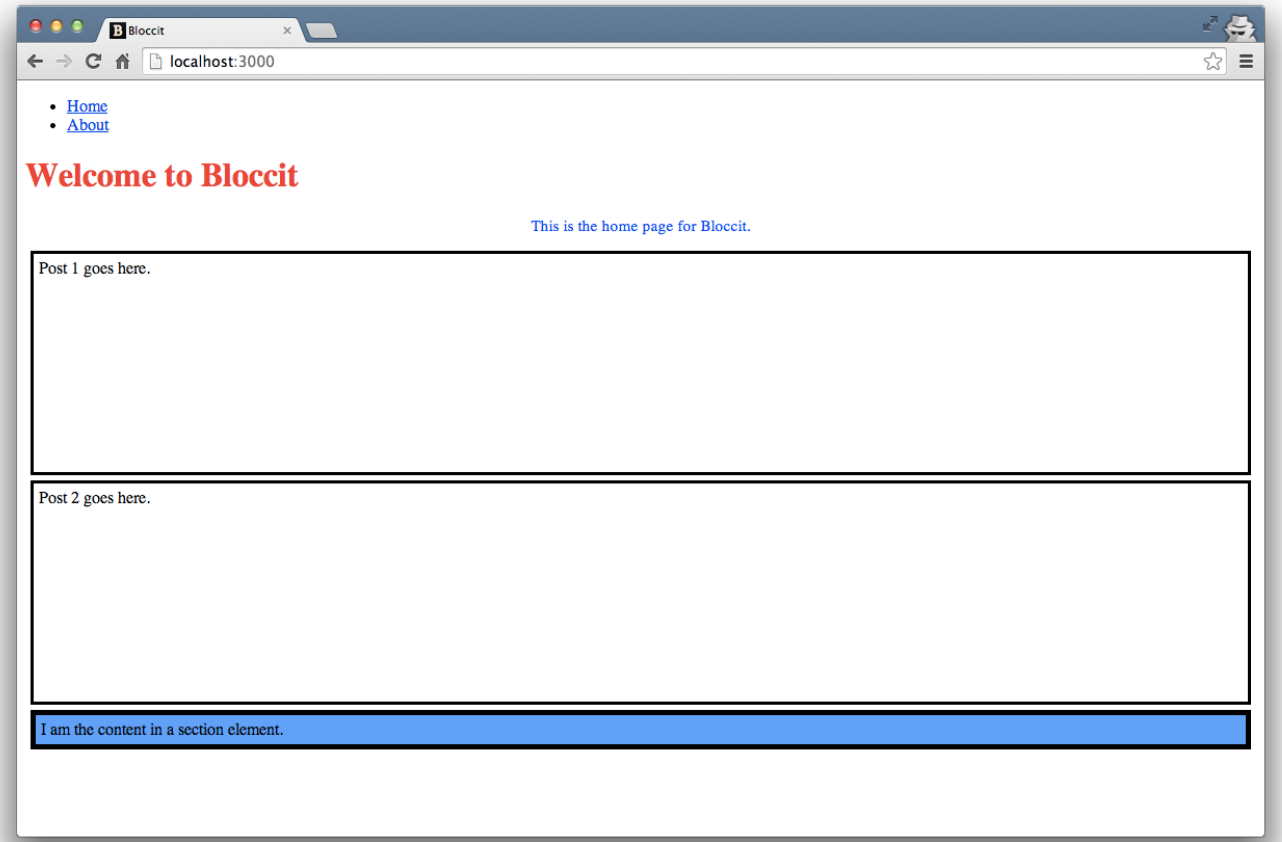
> The `margin` and `padding` values are set clockwise around the box. The first value is the `top`, the second is the `right`, third is `bottom`, and fourth is `left`.

Making the border visible with the `solid` declaration allows us to see the outline of the `section` element.

Add a `<section>` element to the **index** view:

app/views/welcome/index.html.erb

```
  <h1>Welcome to Bloccit</h1>
  <p id="index-title">This is the home page for Bloccit.</p>
  <div class="posts">Post 1 goes here.</div>
  <div class="posts">Post 2 goes here.</div>
+ <section>I am the content in a section element.</section>
```



Welcome to Bloccit

This is the home page for Bloccit.

Post 1 goes here.

Post 2 goes here.

I am the content in a section element.

Change the values of the `margin` and `padding` properties and observe how the box model adjusts. Every HTML element has a box model. When you are trying to make your app look pixel-perfect, you'll find yourself constantly adjusting properties and values of boxes throughout your CSS code.

## CSS Frameworks and Bootstrap

We've added some simple styles above, but we have a long way to go in making Bloccit look presentable. While you could write comprehensive style sheets from scratch, it's much more efficient to use a CSS framework. A CSS framework comes with many different style and position classes that you can use directly or customize further.

One of the most popular CSS frameworks is **Bootstrap**, which we'll use for Bloccit. Bootstrap also provides layouts, forms, buttons, icons, Javascript functions, and more. Let's start by installing bootstrap and using a few basic style classes:

1. Stop the Rails server, open `Gemfile` and add the **bootstrap-sass** gem at the bottom of the `Gemfile` (make sure it is **outside** of a `group`):

   Gemfile

   ```
    ...
   + gem 'bootstrap-sass'
   ```

2. Run `bundle install` to install the new gem.
3. Rename `application.css` to `application.scss`:

   Terminal

   ```
    $ mv app/assets/stylesheets/application.css app/assets/stylesheets/application
   ```

4. Add the following lines to the bottom of `application.scss`, to integrate the newly installed Bootstrap with our application:

   app/assets/stylesheets/application.scss

   ```
   + @import "bootstrap-sprockets";
   + @import "bootstrap";
   ```

5. Include the Bootstrap file in `app/assets/javascripts/application.js`:

app/assets/javascripts/application.js

```
  //= require jquery
  //= require jquery_ujs
  //= require turbolinks
+ //= require bootstrap
  //= require_tree .
```

To use Bootstrap's styles, add some of its layout classes to our container HTML:

app/views/layouts/application.html.erb

```
  <!DOCTYPE html>
  <html>
  <head>
    <title>Bloccit</title>
+   <meta name="viewport" content="width=device-width, initial-scale=1">
    <%= stylesheet_link_tag    "application", media: "all" %>
    <%= javascript_include_tag "application" %>
    <%= csrf_meta_tags %>
  </head>
  <body>
-   <ul>
+   <div class="container">
+     <ul class="nav nav-tabs">
        <li><%= link_to "Bloccit", welcome_index_path %></li>
        <li><%= link_to "About", welcome_about_path %></li>
      </ul>

      <%= yield %>
+   </div>

  </body>
  </html>
```

The "viewport meta" tag added inside the `<head>` with a `content` attribute value of `width=device-width, initial-scale=1` instructs browsers on small, high-pixel density screens (such as retina iPhones) to display our pages at a regular, readable size. Without this tag, our pages won't scale properly.

`container`, `nav`, and `nav-tabs` are classes provided by Bootstrap. By assigning these classes to HTML elements like `<div>` and `<ul>`, you are styling them with default Bootstrap properties and values.

Remove the CSS rules you added to `welcome.scss` as well as the HTML changes you made in the `index` view. They were for demonstration only. Your **index** view should look like this again:

app/views/welcome/index.html.erb

```
<h1>Welcome to Bloccit</h1>
<p>This is the home page for Bloccit.</p>
```

Restart the Rails server and observe the changes.

서버 다시 종료하고 켜지 않으면 Bootstrap Syntax Error뜬다!!

## Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details. Deploy to Heroku with `git push heroku master`.

# 27. Rails: HTML and CSS

**Assignment** | **Discussion** | **Submission**

Create a new Git feature branch for this assignment. See **Git Checkpoint Workflow: Before Each Assignment** for details.

Use CSS and HTML to create a button in your `about.html.erb` page that links to one of your social media accounts:

1. Read about **CSS positioning** and the **Box Model** and use what you learn to center the text inside of the button.
2. Create a CSS class in `welcome.scss` called `.social-btn` and use the attributes you read about to shape and position it. Attributes like display, height, width, background, border, color, text-align and font will help you create a basic button.
3. Read our resource on Chrome's **Web Inspector**. Start Rails Server and experiment with paddings, margins, and other stylings in the inspector.
4. Use the `ActionView` helper method `link_to` to link your button to your favorite social media account. This link should be on the welcome **about** view.

> Hint: You can pass an external link as the URL parameter to the `link_to` method in a string:

```
<%= link_to "Follow me", "https://www.facebook.com/trybloc", class: "
```

Commit your assignment in Git. See **Git Checkpoint Workflow: After Each Assignment** for details. Submit your commit to your mentor.

---

## Solution

**Do not watch this video until after you've attempted to complete the assignment.** If you struggle to complete the assignment, submit your best effort to your mentor *before watching a solution video*.

**HTML & CSS Solution**

assignment completed

?