More Sinatra

We've learned the Sinatra basics. Now let's round out our knowledge.

Auto-reloader

It's been very annoying to have to restart the server every time you make a change.

Let's fix that.

Auto-reloader

In your `Gemfile` add:

```
gem "sinatra-contrib"
```

In your `server.rb` add:

```
require "sinatra/reloader" if development?
```

Auto-reloader

If you are using `require_relative` and you change an external file you will still have to restart manually.

Pry in Sinatra

Pry is awesome and even works in your routes and views.

IRON HACK

Pry in Sinatra

To use Pry in a route:

```ruby
get "/users/:username" do
  @username = params[:username]

  binding.pry

  erb(:user_profile)
end
```

Pry in Sinatra

In a view just add:

```
<% binding.pry %>
```

Pry in Sinatra

Your browser will stay on a loading screen until you exit Pry in the terminal.

IRON HACK

Status code

For most cases, the default status code of 200 works just fine.

Occasionally you will want to change it.

Status code

In Sinatra, you change the status code with the `status` method inside a route.

```ruby
get "/foo" do
  status(418)
  "I'm a tea pot!"
end
```

If you recall, HTTP is a stateless protocol. It doesn't remember any previous user data in a request.

But for many things in the Web, keeping information about users is essential.

T-shirt store

Let's say we want to buy some ironic t-shirts online.

You find three different t-shirts you liked. Time to make the purchase!

You go to checkout... but your cart is empty.

Sessions

Using sessions is very simple.
First we have to enable them.

```
enable(:sessions)
```

T-shirt store

Since HTTP is stateless, the server can't remember that you added those shirts to your cart.

How can we solve this problem?

Sessions are a mechanism to solve this problem. They save the state of your current... session.

Sessions

Once enabled, we can access another special hash called `session`.

```
enable(:sessions)

session
```

Sessions

You can only access `session` inside a route.

```ruby
enable(:sessions)

get "/session_test" do
  session
end
```

Sessions

Think of it as your storage space for anything you want to be saved between requests.

```
enable(:sessions)

get "/session_test" do
  session
end
```

Sessions

It’s a hash so you can save things to it under a key.

```
enable(:sessions)

get "/session_test" do
  session[:saved_value]
end
```

Sessions

Let’s save a value we get from a URL parameter.

```ruby
enable(:sessions)

get "/session_test/:text" do
  text = params[:text]
  session[:saved_value] = text
end
```

Sessions

Now visit that URL to save
a value in the session.

localhost:4567/session_test/pizza

Sessions

We need to add another route to show the current value.

```ruby
get "/session_show" do
  "Now in the session: " + session[:saved_value]
end
```

Sessions

Any route has access to the session.
Add this to a couple of existing views:

```
<small>
  Currently in the session:
  <b><%= session[:saved_value] %></b>
</small>
```

Exercise

Create a hash of valid usernames and passwords.
Use sessions to implement a profile page that can only be accessed by a logged in user.

You should have a log in page with a form.

Finally, don't forget to add a link to log out.