Introduction

To finish this lesson off, we're going to be converting our program into a web app using Sinatra.

Create a file called *web_guess_the_number.rb*:

```
require 'sinatra'

enable :sessions

get '/guess' do
  session[:number] = rand(1..100)
  session[:attempts] = 0
  @message = "I'm thinking of a number between 1 and 100, try and guess what it is"
  erb :guess
end

post '/guess' do
  number = session[:number]
  guess = params[:number].to_i
  session[:attempts] += 1
  redirect to('/success') if guess == number
  if guess < number then @message = "Too small, try again"
  elsif guess > number then @message = "Too big, try again"
  end
  erb :guess
end

get '/success' do
  attempts = session[:attempts]
  "Well done, you guessed my number in #{attempts} attempt#{'s' if attempts > 1}!"
end

__END__

@@guess
<!doctype html>
```

```
<html>
  <header>
    <title>Guess the Number</title>
  </header>
  <body>

    <p><%= @message %></p>
    <form method="POST" action="/guess">
      <input name="number">
      <input type="submit" value="Guess">
    </form>
  </body>
</html>
```

For this program, we'll need to keep note of the information after each request, such as the number that needs to be guessed by the player. The HTTP protocol the web uses is **stateless**, which means that information isn't stored between each request. One way to get around this is to use **sessions**. This is really easy to do in Sinatra. You just need to add

```
enable :sessions
```

This will use **cookies** to store information. You need to be aware that these are not secure and can be intercepted and decrypted by a hacker. It's not so much of a problem in this case, as it's only a game, but do keep this in mind if you ever plan to store sensitive information in the future.

In the next part of the program we create the route handler, which will be accessed when the player goes to the page with the URL `/guess`. This sets up the game by using the session to store a random number from `1` to `100` and the number of attempts.

As I mentioned earlier, we use sessions to store information throughout the life of the program. Otherwise, the information will be lost after each request. To do this, we use the `session` hash. It's used to store any information that goes in the session. Think of it as of variable that is stored throughout the life of the program.

Instance Variables

Next we define what's called an **instance variable** called `message`. Instance variables always start with the `@` symbol. They're available in the view that's associated with the route handler. At the end of the route handler we call the view called `guess` using the `erb` method.

In this view, the string that's assigned to the instance variable `message` will be available to be shown in the view. This view is a basic web page that contains a form for submitting the guess.

There is also a reference to the instance variable `message`. When we use `erb`, we can employ a special tag `<%= %>` that is used to evaluate Ruby:

```
<p><%= @message %></p>
```

Any Ruby code that goes inside these tags will be evaluated and then outputted in the HTML. In this case, the string that is stored in the `message` instance variable will be displayed.

Testing the Code

Go into the browser now and navigate to `http://localhost:4567/guess`. You will see the message was the string that was stored in the instance variable called `message`. It's been outputted in the HTML from that special Ruby tag.

Now let's have a look what happens when the form is submitted.

Postback

The `method` attribute of the form tells that we're going to use the HTTP POST method. We're also providing a route where it will be posted to, using the `action` attribute. This is called a **postback** and it's when you post a form to the same URL of where the form is located. In this case, this URL `/guess` that was the same URL where we got the form from and it's going to post the form to exactly the same URL, but it's going to use a different method.

Remember when we got the page we used the GET method and now we'll stick with POST. Notice in the input field we have a name attribute called `number`. This is the value that will be stored in the `params` hash when the form is posted.

Here is a route handler that deals with the form when it is posted:

```
post '/guess' do
  number = session[:number]
  guess = params[:number].to_i
  session[:attempts] += 1
  redirect to('/success') if guess == number
  if guess < number then @message = "Too small, try again"
  elsif guess > number then @message = "Too big, try again"
  end
  erb :guess
end
```

We create a variable called `number` and assign it to the `session` variable that we created before. That was the random number created by the game earlier. Then we grab the guess from the form that was submitted in the input box and as I mentioned this will be stored in the `params` hash using the same name as was in the `name` attribute in the form.

All values submitted by a form will be submitted as strings and we want it to be a number, so at the end we add the `to_i` method to convert it to an integer. Next we increase the number of guesses that have been made.

Redirect Method

After this, we use an `if` statement to redirect to a success page if the guess is equal to the number. We use Sinatra's built in `redirect` and `to` methods that work together. These methods make it a very easy statement to read: we want to redirect to `/success` URL if the guess is equal to the number. Remember, it's two equal signs if you want to check for equality instead of assigning a value. If the user isn't redirected, then their guess must have been wrong.

Else If Logic

So, we then use `if` and `elsif` logic to test if the guess was too big or too small and set the message accordingly.

We finish off in the last line using the `erb` method once again to display another view called `guess`. This is the same view that we had before, the form will be displayed again, and all that will change is the instance variable `message`.

Route Handler

All there's left to do is to create the route handler to deal with a successful guess. This will just be a GET request:

```
get '/success' do
  attempts = session[:attempts]
  "Well done, you guessed my number in #{attempts} attempt#{'s' if attempts > 1}!"
end
```

We simply display a congratulatory message that informs the user of how many attempts it took using interpolation.

Restart the Server

Save the file and restart your server by holding `Ctrl+C` to stop it and then type

```
ruby web_guess_the_number.rb
```

in the terminal.

Testing

So now reload the page and observe the result!