Introduction

We're going to turn the Mad Libs program into a web app. This time, we're going to use an HTML form to enter the data, though.

To get started, create a file called *web_madlibs.rb*.

ERB

```
require 'sinatra'

get '/madlibs' do
  erb :questions
end

post '/madlibs' do
  animal = params[:animal]
  color = params[:color]
  person = params[:person]
  object = params[:object]
  adjective = params[:adjective]
  verb = params[:verb]
  "The #{adjective} #{animal} started to #{verb} because the #{person} ran away with the #{object
end

__END__

@@questions
<!doctype html>
<html>
  <header>
    <title>Madlibs</title>
  </header>
  <body>
    <form method="POST" action="/madlibs">
      <p>Animal:</p>
```

```
      <input name="animal">
      <p>Color:</p>
      <input name="color">
      <p>Person:</p>
      <input name="person">
      <p>Object:</p>
      <input name="object">
      <p>Adjective:</p>
      <input name="adjective">
      <p>Verb:</p>
      <input name="verb">
      <input type="submit" value="Create Madlib">
    </form>
  </body>
</html>
```

This is the most complicated program we've created so far, but don't worry, it's not as bad as it looks if we break it down into chunks.

First of all, it starts with a familiar `require sinatra` that we use on all our web apps. Then goes the route handler that is quite straightforward. It states that if the user visits the route `/madlibs`, then we will use ERB to display a view called `questions`.

A **view** is a piece of HTML code that gets displayed by the browser, and **ERB** stands for **Embedded Ruby** that is a templating engine is used to generate HTML. In this example we're going to be using **inline views**, which means that you write them at the bottom of the file after the `end` declaration. The name of the view comes in the following format: `@@questions`. This is the same name given in the route handler that called the view earlier. We called the view called `questions` using the `erb` method.

The view itself is actually just a bit of HTML that displays a form.

Starting the Server

Start your webserver by running

```
ruby web_madlibs.rb
```

Testing the Application

Navigate to `http://localhost:4567/madlibs`.

Method Post

Go back to the code and take a closer look at the form. There are two very important attributes:

- `method="POST"` - it tells Sinatra to use the HTTP POST verb to send the form to the route defined by the second attribute.
- `action="/madlibs"`.

We can then create a route handler to deal with what happens when this form is posted:

```
post '/madlibs' do
  animal = params[:animal]
  color = params[:color]
  person = params[:person]
  object = params[:object]
  adjective = params[:adjective]
  verb = params[:verb]
  "The #{adjective} #{animal} started to #{verb} because the #{person} ran away with the #{object}
end
```

We use `post` at the beginning instead of the verb `get`, because in all of our previous route handlers, we've only been concerned with the getting of pages. This time we want to deal with a route that has been posted like the one from our form.

Inside the block we can see that the code deals with the information submitted by the form. Each of the input fields in the form has a `name` attribute that tells Sinatra where to save the value entered in that field in the `params` hash.

Reviewing the Program

For example in the form we have:

```
<input name="animal">
```

and in the route handler we then get this value:

```
animal = params[:animal]
```

Now go back to the browser, fill in the form and submit it to test how this all is working!