CFG Advanced Ruby: Session 3
**Sinatra & GET/POST requests**

**Recap**

- What is RubyGems?
- How do you write comments in Ruby?
- What is the difference between
- 10 / 3 and 10 / 3.0 in Ruby? What is a function?
- Can functions take variables? Can
- functions have return values?
- What does a function without a return value return?

Sinatra: An Introduction

Sinatra is a framework for building lightweight web applications. We will install Sinatra using RubyGems - type the following into your terminal:

```
gem install sinatra
```

(if this doesn't work, you may have to add sudo to the front of the command and try again - the terminal will ask for your password).

Then we will create a baseline Sinatra application - call it **hello.rb** (.rb is the extension given to all Ruby files) - which will look like this:

```
require 'sinatra'

get('/') do
  "Hello
  World"
```

**end**

Imports

You might have noticed that at the very beginning of the file we are doing:

```
require 'sinatra'
```

this is Ruby's way of letting you import the code from different modules and libraries to use.

In this case, we already have Sinatra installed, so we can safely require it. If you tried to require a module or library that was not installed using RubyGems, then you would get an error.

### Sinatra routes

The rest of the code is using Sinatra's get function to define a *route* (a URL that we can recognise and respond to). The argument to the function is the URL we want to respond to - in this case we're specified the root URL ( / ). When Sinatra receives a request with this URL, it will run the code between the do .. end (this is called a *block*, and send back to the browser whatever we return - remember that Ruby will return whatever the last expression is.

- 
- 
- 
- 

Task:

Run the simple hello.rb application by doing ruby hello.rb and hitting enter Navigate to localhost:4567 in your browser and confirm that you see the message expected

Make a change to hello.rb - return something other than Hello World! - refresh the page in your browser and see what happens (you will need to reboot your server in the meantime by pressing Ctrl+C )

Try visiting a URL like localhost:4567/i/dont/exist - what happens?

Taking parameters from the URL

Take a look at the following code:

```
require 'sinatra'

get('/') do
  "hello"

end

get('/:name') do

  name =
  params[:name] name

end
```

The :name is a URL matcher - It will match / followed by any word, for example /rob , /rachel etc. Sinatra will make this value available as a parameter from the params hash. We then use the name variable to say hello to a particular user.

- 
- 

Task

Add the new route to your hello.rb file and check that it works by visiting localhost:4567/rob or your own name if you prefer

Try and make it so that visiting localhost:4567/bye/rob returns ‘Goodbye Rob’ or ‘Goodbye’ for any other name given

Serving your HTML using Sinatra

Sinatra lets you serve your HTML - this is possible using the erb function. All you have to do is provide the name of the template and the variables you want to pass to the template engine as keyword arguments. Here’s a simple example of how to render a template:

```
require 'sinatra'

get('/') do
  erb :hello

end

get('/:name') do

  @name = params[:name].capitalize
```

```
    erb :hello
end
```

Note: Sinatra will look for templates in the views folder.

Sinatra uses another library called **ERB** , which is a templating language for Ruby. It's extremely powerful, but it's beyond the scope of this course. If you are interested however, go ahead and read up on it and make use of it.

Here is what an example html file called hello.erb would look like, inside a views folder, that uses ERB:

```
<!doctype html>
<title>Hello from Sinatra</title>
<% if @name %>
  <h1>Hello <%= @name %>!</h1>
<% else %>
  <h1>Hello World!</h1>
<% end %>
```

Task

Extend the above code so that you can say goodbye to someone if you visit localhost:4567/goodbye/rob (or feel free to use any other name)

Try to add some more parameters to the URL and pass those onto your hello.html template - for example localhost:4567/goodbye/rob/day would say something like “Hello Rob! Have a good day”, while localhost:4567/goodbye/rob/night would say “Hello Rob! Have a good night”.

**GET & POST requests**

In the world of HTTP, there are a number of types of requests - we will be concentrating on two of them:

- GET when you want to retrieve data
- POST when you want to send data

Everything we have been doing so far has been using a GET request - let’s see how we can use a simple POST request. Let’s add a form to hello.erb :

```
<div id="contact-
  form"> <h1>Get In
  Touch!</h1>

  <form method="post"
    action="/signup"> <label
    for="name">Name: </label>

    <input type="text" id="name" name="name" value=""
placeholder="Rob P askin" required="required"
autofocus="autofocus"/>

    <label for="email">Email address: </label>

    <input type="email" id="email" name="email" value=""
placeholder="ro b@example.com" required="required"/>

    <input type="submit" value="submit" id="submit-
  button"> </form>

</div>
```

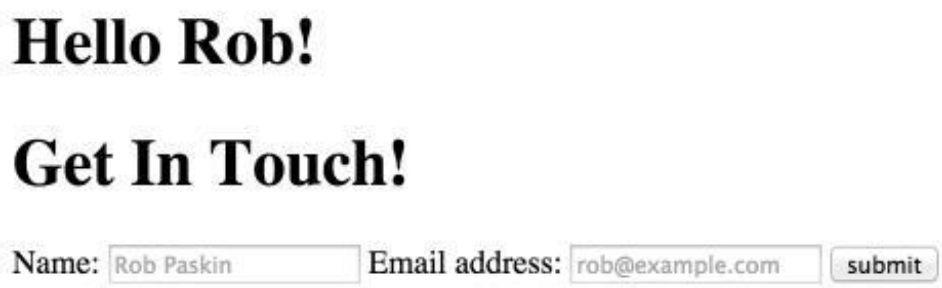
Pay attention to the form element which has an action equal to /signup - that's the URL which the form will be submitted to, relevant to the absolute URL you are at (in this case localhost:4567 )- so we need to create an entry in our hello.rb file:

```
post('/signup') do
  puts
  params[:name]
  puts
  params[:email]

  "All OK"
end
```

Notice that Sinatra makes our form values available in the same params hash as in our previous examples that used GET requests.

Now if you navigate to localhost:4567/rob then you should see something like this:



Task

Try submitting the form without entering any information and see what happens! Now fill in the form and submit it

Go to your terminal or command prompt and have a look to see if you can spot the name and e-mail address you entered and submitted

Extend the form in hello.erb to have another one or two input fields to capture some more information and ensure that this is printed out by hello.rb

Sinatra project structure

This is the recommended structure that your Sinatra projects/applications should follow:

```
/your_app
    hello.rb /
    public

        /js

            hello.js

        /css
```

hello.css

/views

hello.erb

Anything in your public folder is available under the root URL ( / or localhost:4567 ). In order to reference your CSS, JavaScript and images from within your HTML then you need to do something like this:

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

<script src="/js/hello.js"></script>
```

Homework

- Complete up to and including **Exercise 40** from *Learn Ruby The Hard Way*
- Head to Mailgun's website (www.mailgun.com) and create an account - we will be using Mailgun to programmatically send emails.

  Head to twitter.com and create yourself a Twitter account if you don't already have one - we will be using Twitter's API to fetch data - you will need to have an account in order to do that