*This course is based on RubyLearning's "**Introduction to Sinatra**" book. Do consider buying this eBook to support the time and energy put into the various courses by RubyLearning's mentors and to help RubyLearning bring to you relevant eBooks and courses related to the Ruby programming language.*

set method

set is a method in Sinatra that is used to set some application variables (that you wish to access throughout your app) or some configuration option. For example:

```
set :dbname, 'productiondb'


get '/whatdb' do
  'We are using the database named ' + settings.dbname
end
```

Sinatra includes a number of built-in options that control whether certain features are enabled. Settings are application-level variables that are modified using one of the **set**, **enable**, or **disable**methods and are available within the request context via the **settings** object.

A detailed note on **Sinatra Settings and Configuration** is available here – http://www.sinatrarb.com/configuration.html andhttp://ididitmyway.heroku.com/past/2010/11/9/sinatra_settings_and_configuration/

before block (filter)

```
before do
end
```

The **before** block is designed to be used for code that is to be executed before every request.*Instance variables set in filters are accessible by routes and templates*. We shall see later on how to use the **before** block. It's a good place to set some defaults.

pass filter

Using **pass** your route can shift the processing to the next matching route. The route block is immediately exited and control continues with the next matching route. If no matching route is found, a 404 is returned.

```
get '/guess/:who' do
  pass unless params[:who] == 'Frank'
  "You got me!"
end


get '/guess/*' do
  "You missed!"
end
```

status

If you want to set your own status response instead of the normal 200 (Success), you can use the status-helper to set the code, and then still render normally:

```
get '/' do
  status 404
```

```
  "Not found"
end
```

Now let us analyze our trivial Sinatra app `day2_1_e1.rb`

Sinatra applications can be run directly:

```
ruby day2_1_e1.rb [-h] [-x] [-e ENVIRONMENT] [-p PORT] [-s HANDLER]
```

Options are:

```
-h # help
-p # set the port (default is 4567)
-e # set the environment (default is development)
-s # specify rack server/handler (default is thin)
-x # turn on the mutex lock (default off) – currently not used
```

Note that Sinatra uses port 4567 by default.

*Sinatra does not support application reloading and you need to stop the server and re-start it every time you make a change to the **day2_1_e1.rb** application.*

*Rack aims to provide a minimal API for connecting web application servers (WEBrick in my case) and web frameworks (Sinatra). Our Ruby web application **day2_1_e1.rb** is built using the web framework Sinatra. Sinatra is built on top of Rack.*
*Thus **day2_1_e1.rb** is a Rack app. Here, the web application server WEBrick runs our Rack app **day2_1_e1.rb**. Rack supports "middleware" - components that sit between the server and your application, monitoring and/or manipulating the HTTP request/response.*

When we type (`http://localhost:4567`) in a browser window, a HTTP GET request is sent to the application **day2_1_e1.rb**

You should see in your browser window:
Hello Rubyists

This happens because when you use the above URL, Sinatra runs the **get** block and will return whatever you put inside it, to the web browser.

day2_1_e2.rb

Let us modify our program `day2_1_e1.rb` as follows, to see how the **before** block and the **set**method works:

```
require 'sinatra'

before do
  set :display_string, 'Welcome from RubyLearning again!'
end

get '/' do
  settings.display_string
end
```

Save the program in the file `day2_1_e2.rb` to the folder `sinatrafb`. Run this program and check out the results.

Links to the courseware

- Back to the **Beginning of the course**
- Back to **Day 2.1**
- Forward to **Day 3**

Please click on the **Like** button of this page. Comments appreciated.