IWDD100

Ruby Libraries, Sinatra, ERB

RubyGems - the Ruby library system

- A library is a collection of functions, usually for a specific purpose, that have a well-defined interface
- The system in Ruby for the creation and use of libraries is called **RubyGems**
- A Ruby library itself is known as a **Gem**

Using RubyGems

- Installation: `$ gem install mygem`
- Uninstallation: `$ gem uninstall mygem`
- List all installed gems `$ gem list --local`
- When using RVM - the `sudo` command is typically not necessary to install gems. Only use it if you have problems with permissions

Bundler

- Bundler is a gem that makes installing a **gem set** for each of your specific projects much easier

- A **gem set** is a set of gems locked at specific versions to be used in your project, a set of **dependencies**

- To install bundler:

```
$ gem install bundler
```

Gemfile

- A Gemfile is simply a list of all of the RubyGems your project needs to function
- Bundler uses the Gemfile to know which gems to install
- The file itself should **always** have the name **Gemfile**, with no extension after it

Gemfile

- At the top of your Gemfile, you need to specify a source for your gems and the version of Ruby you're using on your machine

- Almost every open-source gem is hosted at http://rubygems.org

```
source 'http://rubygems.org'
ruby '2.0.0'
```

Gemfile

- After these initial two lines, you can start listening gems using the following syntax:

```
gem 'sinatra'
gem 'haml'
gem 'activerecord'
gem 'sinatra-activerecord'
gem 'rake'
gem "bcrypt-ruby"
```

Gemfile

- If you need to fetch a gem from a git repository somewhere, you can use this syntax:

```
gem 'rack-flash3', :git => "git://github.
com/treeder/rack-flash.git"
```

Installing the gems in your Gemfile

- Once you've added necessary gems to your Gemfile, `cd` into the directory it resides in and run the `bundle install` command:

```
$ cd ~/dev/my_project_directory
$ bundle install
```

Give it a go

- Create a new directory for your first Ruby project
- In Sublime Text, create a Gemfile with the gem named “geolocater”
- `cd` to your project directory in the Terminal
- Run `bundle install`

Give it a go

- Create your main Ruby file inside the folder, with a name like `app.rb`

- Require the Geolocater gem and some test code:

```
require ‘geolocater’
record = Geolocater.geolocate_ip
("64.119.207.255")
puts record
```

10 minute break

Sinatra

- A free and open source web application framework similar to Rails
- Allows you to build a web application with only one file
- Easily extensible to add database and other common web app functionality

Your first Sinatra web application

- Create a new project folder with a Gemfile containing `sinatra`
- Run bundle install
- Create a main Ruby file ending in .rb
- In that file, put this code:

```
require 'sinatra'

get '/' do
  "Hello World"
end
```

Your first Sinatra web application

- Open your web browser and navigate to `http://localhost:4567`
- Boom! You should see Hello World.

What's going on here?

- Sinatra is a **DSL** or a **D**omain **S**pecific **L**anguage

- It's a library that adds functionality to the core Ruby library so that you can easily declare web page routes

HTTP Methods

- `Resource -` a file, most of the time full of HTML content

- `GET -` Requests a representation of the specified resource, “load this page”

- `POST -` Requests the server to accept data input and process it as a new entity of a specified resource, “process this form data”

Sinatra Breakdown

Whoops.

Real Sinatra Breakdown

```
1 require 'sinatra'
2
3 get '/' do
4  "Hello World"
5 end
```

- On line 1, we require the Sinatra library to gain access to its functionality
- On line 3, we declare a “route” to be accessed using the `HTTP GET` method whenever the user hits “/”, the top level of the site
- On line 3, the `do` keyword indicates that this is a Ruby **block**
- On line 4, we specific what this block should return to the user
- On line 5, we end the Ruby block

Exercise

- Create a Sinatra “app” that serves up 3 different pieces of text depending upon which URL the user hits on your site

- Keep in mind that you need to restart your app every time you change the main Ruby file

- To restart the app, kill it with CTRL + C then type `ruby yourapp.rb`

Templating

- Wouldn’t it be nice to use HTML in your new website instead of just plain text?
- Enter ERB - “**E**mbedded **R**u**B**y”
- ERB is just like HTML except you can put Ruby in it!

Using ERB

- Create a folder inside of your project folder called `views`
- Inside of this folder, create a file called `home.erb` and put some HTML in it
- To render this view inside of a Sinatra route, use the following code:

```
erb :home
```

Using ERB

```
get ‘/home’ do
  erb :home
end
```

Exercise

- Make one of your routes for the three-route app you created before into a route that displays an ERB view instead of returning plain text

Running Ruby code within ERB

- Try adding the following code to your ERB file:

```
<% 10.times do %>
  <b>
    THE PLANET IS 'SPLODIN'!
  </b>
<% end %>
```

Running Ruby code within ERB

- To clarify, the syntax for running ruby code is:

```
<% #put your Ruby code here %>
```

- The syntax for running Ruby code **and** displaying its output is:

```
<%=“hi” + “there” %>
```

- Notice the = sign, this is what indicates you’d like the result of your Ruby code to be displayed

Exercises

- Add Ruby code to your ERB view that:
  - Assigns an array of names to a variable
  - Loops over that array variable and displays each name inside of a <b> tag

- **Bonus exercise:** Try creating a navigation menu inside of a <ul> that takes navigation items from an array and uses a loop to avoid repeating <li> and <a> tags

Final Exercise

- Take your personal website project and convert it into a Ruby/Sinatra-based website.
  - Start with the basics: declare the correct routes in your main app file.
  - Then move over the views into the /views folder.
  - Don’t worry about images or CSS for now, just get the HTML to show up.