This tutorial is designed to help people get oriented with the basics of Ruby on Rails, and to acquaint them with the SCI&U Open Source Health Coaching Project and Program. It assumes you are using a Mac computer. Note also that the steps in this tutorial are drawn in large part from Rim Lomas' 2014 Tutorial on Medium.com titled "How to make a simple job board in Ruby on Rails". Rik Lomas' posts can be found at https://medium.com/@riklomas

1. Installing Ruby on Rails

There are tools built into your Mac or Unix system that let you play around with the command line. On a Mac, you want to use Terminal which can be found in your Applications folder in the subfolder Utilities. Open Terminal.

Terminal "commands" can change folders, make a new file, list the files in a folder, and more. Open Terminal, type this into your command line the following and press enter:

```
say "I am an open source coder"
```

We'll be using the command line from here so have it ready to go.

You may or may not already have Rails installed. To see if you do, you can type in and press enter:

```
rails --version
```

If you have it installed, it will say something like "Rails 6.0.3.3". If not, you will get an error.

If you don't see an error, to update to the latest version of Rails, type in and press enter.

```
gem update rails
```

If you do see an error, you will need to install Rails. Type the following and press enter.

```
gem install rails
```

When you type this, things should happen. If you get a permissions error, you may need to add the word "sudo" to the start of either command and it will ask for your computer's password, like this:

```
sudo gem install rails
```

You should have the latest version of Rails set up on your computer now.

2. Create a Folder for your Project

The next step is we need to decide where to put our project. By default, the command line will drop us into what's called the "home directory". This is where your user account on the computer is.

Make a folder in which we will story all of your Ruby on Rails code. To do this, type:

```
mkdir Sites
```

The command mkdir is made of "mk" and "dir". First part is "make" and the second is "directory" (another way to say folder). So we're telling the computer to make a directory named "Sites".

Let's go inside the Sites folder by "changing directory" (or cd for short). Type:

```
cd Sites
```

3. Create a Project

You can call your Rails project whatever you like, just as long as it doesn't have any spaces in the name and it's all lower case, e.g. no to "Health Browser" but yes to "healthbrowser". To create your project, type in the following and press enter.

```
rails new healthbrowser
```

Rails will create all the files and folders you need for your project, and it will download some gems, which are plugins that contain helpful code. This process may take a minute or two depending on your internet connection.

Then, go into the folder of the project:

```
cd healthbrowser
```

Rails has generated a lot of files and folders in this folder. These are the skeleton for our health resource browser.

4. Install a Code Editor

There are lots of good code editors like Atom and Sublime Text 3. Download one of them and install it to your computer. Once it is installed, open your project files in the health browser folder with the editor of your choice.

5. Website Plan

Typically you will want to start your project by designing and planning it. There are tools to do this but a pen and paper can suffice.

On our demonstration site we want to create a home page which will be a list of health resources, and an area to add health resources.

The best way to plan anything out is to draw and get feedback from potential users. With a good set of drawings that have been reviewed by people who understand what users want, we can start coding.

6. Models, views and controllers

Rails separates sites into three different parts; models, views and controllers.

Models structure the data that your site will hold. You can think of models as if they were sheets in an Excel (or Numbers) spreadsheet. There might be a model for a health resource, for example; this might contain a column for the resource title, its description and its URL. Each row in the spreadsheet corresponds to a different record for a specific health resource.

Views contain the HTML for the site. The views are used to present data in the website to users. If you don't know much about HTML, look at Jon Duckett's HTML and CSS Book.

Controllers allow the parts of the Rails site to talk with one another. For example, when we're on the homepage, we want all the health resources from our spreadsheet to go into our views in a particular way. When we try to add a health resource, we will need to make a new row in our spreadsheet and we show the user a form to complete.

7. Making Features

In general when you add new features to your website you will:

- Generate any models needed for the feature
- Update the database
- Generate any controllers needed
- Fill in your models, controllers and views
- Finish feature, think of new feature and repeat

8. Generating Code

We're going to call our model "HealthResource". The names of the models will be capitalized and singular. Other models we could have include "User" or "Comment" but not "users".

Our HealthResource model will have three columns (ignoring the number column) which are title, description and URL. We're going to store them all as phrases or strings.

We can store data in other formats such as numbers, true or false (called “booleans”), and dates. For what we need, we’re keeping with strings.

In our command line, lets type in:

```
rails generate model HealthResource title:string description:string url:string
```

9. Generating a Database

When you make a model, it doesn’t automatically go into your database. The database is the place you store data. This doesn’t live within Ruby or Rails but in SQLite3, which is a database. SQLite3 comes as part of Rails by default, but we could use other ones such as MySQL or PostgreSQL.

So let’s update our database with our new model. In our command line, type in:

```
rake db:migrate
```

You should see something like “CreateHealthResources: migrated” when you press enter.

You might have noticed that this command starts with “rake” not “rails”. Rake is a way to run particular ruby commands on your command line. The migrate command is updating (or migrating) our database to the very latest version that it needs to be.

10. Adding Views and Controllers

Even though we have just one health resource model, we have two pages on the site: a home page (or index page) and an add page (or new page).

Rails wants to control which health resources you’re getting on which page. Our home page will behave differently than the new health resource page. This is where our controller comes in.

We only need one controller at this point that deals with health resources. Just like with models, Rails makes controllers conform to conventions. Controllers are lower case and plural — so our controller would be for “healthresources”, “users” etc.

We also want to make two pages to go along with this controller, the home page (in Rails, we call this the index page) and the add page (in Rails, we call this the new page).

Go back to our command line and type in:

```
rails generate controller healthresources index new
```

When you do this Rails will generate a controller called healthresources and an index and new page with that controller.

11. Adding URLs

To add the URLs for the index and new pages we need to go back to our code editor. We need to find the file inside the config folder called routes.rb as this is where we can add our URLs. Keep the first lines intact and at the end add these lines:

```
resources :health-resources
root 'healthresources#index'
```

To run the code go to the command line and type in:

```
rails server
```

Now, if we go to http://0.0.0.0:3000 or http://localhost:3000 in our browser, our code will run. The home page is still blank but we will fix that.

12. Populating the Health Resource Model with Valid Entries

We want to make sure that every health resource definitely has a title, a description and a URL, otherwise the site will be full of blanks.

In our code editor, look in the model folder within the app folder.

Notice in here we have a file called healthresource.rb. This is the model that we generated earlier. Open this file and change:

```
class HealthResource < ActiveRecord::Base
end
```

to:

```
class HealthResource < ActiveRecord::Base
  validates :title, presence: true
  validates :description, presence: true
  validates :url, presence: true, uniqueness: true
end
```

This makes sure that our health resource titles exist. In other words, we won't add resources with titles that are blank. Same thing with each resource's description. For our resource's URL, we're making sure that it isn't blank and it is unique as we don't want any duplicate resource posts.

These are called Rails validations and there's a list of them here.

13. Adding some Health Resources

At the moment, there aren't any health resources on our site. Our spreadsheet is empty. So let's go to our new health resource page first. Go to http://0.0.0.0:3000/healthresources/new or http://localhost:3000/healthresources/new and you should see a blank page.

We want this page to allow us to add health resources to our database To do this we will go to our health resource controller. Using our code editor, go to the app folder, then the controllers folder, and open healthresources_controller.rb.

We'll add code between lines 5 and 6, the "def new" and the "end".

"def" is just short for define. The code in between this def and the end will create a new row in our job model.

Change:

```
def new
end
```

to:

```
def new
  @healthresource = HealthResource.new
end
```

HealthResource.new means "Health Resource model, can we make a new one of you?". But what about the first bit "@healthresource"? This is a variable that we will use to send the new health resource we've created to our HTML in our view so we can use it for display.

Next thing to do is edit the HTML on our health resource page. Let's go to the app folder, then views folder, then healthresources folder and find new.html.erb.

We want to put a form in place of the dummy text. Replace all the text in this file with:

```
<%= form_for @healthresource, url: healthresources_path do |f| %>
        <div class = "section">
        <label>Title</label><br>
        <input name="healthresource[title]" type="text" />
        <p></p>
        <label>Description</label><br>
        <textarea name="healthresource[description]"></textarea>
        <p></p>
```

```
        <label>URL</label><br>
        <input name="healthresource[url]" type="text" />
        <p></p>
        <input name="commit" type="submit" value="Save Health Resource" />
        </div>
<% end %>
```

This makes a form for the new health resource with the columns in the order that we want. Data from this form will be sent to our controller, and to the “create” function in our controller.

Try the form out. It won’t work … because we don’t have a “create” function yet!

Let’s go back to our health resource controller and add this create function to it. Add this:

```
def create
  @healthresource = HealthResource.new
  if @healthresource.save
    redirect_to root_path
  else
    render "new"
  end
end
```

Finally, we have to make sure that the data we use to create a new health resource conforms to our model and is send securely. To do this, add these lines just after your create method in the same healthresources_controller.rb file:

```
private
def  healthresource_params
   params.require(:healthresource).permit(:title, :description, :url)
end
```

Then change line 10 in your create function from:

```
@healthresource = HealthResource.new
```

to

```
@healthresource = HealthResource.new(healthresource_params)
```

This makes a new health resource from the form data but only if we have a title, description and url data.

Now you should be able to add a new health resource. When you do you will go to the home page. That's great but we can't see any health resources on the home page!

14. The Home Page

Remember that Rails calls our home page the index page. In our controller we want to get all our health resources to show our users on this page.

Let's change our index page in our health resources controller from:

```
def index
end
```

to:

```
def index
  @healthresources = HealthResource.all
end
```

This will get all the health resources from the HealthResource model and send them to the HTML view as @healthresources (we could name @healthresources anything we want by the way, @all_healthresources for example).

Last thing we need to do is add the resources to our index page's HTML. This is in the app folder, in the views folder, in the healthresources folder, and in the file index.html.erb

We want to loop over each health resource and wrap it up in some HTML. Let's change all the text in index.html.erb to:

```
<% @healthresources.each do |healthresource| %>
  <div class="healthresource section">
    <h2><%= link_to healthresource.title, healthresource.url %></h2>
    <p><%= healthresource.description %></p>
  </div>
<% end %>
```

This is a loop. It takes each individual resource from the controller and adds some HTML to each one. If there are 5 resources, there will be 5 tags with the class "healthresource" and "section", each with a <h2> tag and a <p> tag inside.

You should now be able to add as many health resources as you like to your website; the more you add, the more there will be on the home page.

15. Styling

Let’s go to our assets — these would be familiar to anyone who has done any front-end web development. This is where your CSS, your Javascript and your images live.
For styling, we need to edit our stylesheets. For this, we’re going to go our app folder, then our assets folder, then stylesheets folder and open application.css.

Replace all text with:

```
input,textarea { width: 75%; padding: 10px; }
a, label { color: #fff; }
body { background-color: #fff; background-image: url("https://sci-and-u.ca/images/city-overlay2.png"); background-size: cover; background-position: top;  background-repeat: no-repeat; min-height: 100vh; }

.grey { background-color: #333; width: 100%; text-align: center; color: #fff; padding: 10px;}
.section { background-color: #6699FF; min-height: 10vh; margin-top: 10px; margin-bottom: 10px; text-align: center; padding: 10px; }
```

Next add this line to the top of your healthresources/index.html.erb file:

```
<div class="grey"><h1>Health Resources</h1></div>
```

Add this line to its bottom:

```
<p><div class="grey"><%= link_to "Add a health resource", new_healthresource_path %></div></p>
```

Finally, add this line to the top of your healthresources/new.html.erb file

```
<div class="grey"><h1>Add Health Resources</h1></div>
```

Add this to its base:

```
<p><div class="grey"><%= link_to "Back to index", "/" %></div></p>
```

Note that the parts at the bottoms of the page contain the phrase ‘link_to’. This is part of Rails and will make a link that goes to whateever URL is provided. The phrase “new_healthresource_path “ will create a URL to the new health resource page. This just saves you from having to write <a> tags yourself.

That’s it! You should now have a health resource browser that you can add to as you like.

16. Getting Help

Stuck with any part of Rails? The best places to look are Google and Stack Overflow. If you get an error, copy and paste it into Google and see if anyone else has had the same problem before. If not, search on Stack Overflow and if nothing good, sign up and post a new question.

Commons errors are:

- Missing or too many “end”’s in your model or controller
- Check the difference between <%= and <% .
- Spaces and non-spaces between things like colons
- Typos — they happen a lot, watch out for @healthresourcs instead of @healthresourcss for instance
- Not saving files — it’s a common mistake that professionals make

17. Next Steps

Other things you could build using exactly the same tutorial:

- A blog (replace health resource with post)
- A link board like Delicious (replace health resource with link)
- A real estate listing site (replace health resource with house)
- A review site (replace health resource with album/movie/gig)
- A news site like Hacker News or Designer New (replace health resource with story)

Further resources

- The official Rails guides are great: http://guides.rubyonrails.org/index.html
- RailsCasts are amazing, they’ve not been updated in a while as Ryan the owner is AWOL (come back soon Ryan) but they’re still very useful: http://railscasts.com/
- Recommendation by Gaz Aston – The Ruby on Rails Guide eBook: http://www.amazon.co.uk/Ruby-Rails-Guide-Stefan-Wintermeyer-ebook/dp/B00E25KVLW
- For beginner Ruby, check out Why’s (Poignant) Guide to Ruby. Again, the author “Why” is AWOL (come back soon Why) but the book is a hilarious and fascinating way to learn Ruby. http://mislav.uniqpath.com/poignant-guide/book/
- For the best gems and services worth checking out, try my previous article. https://medium.com/@riklomas/my-favourite-ruby-gems-services-89fb47341c05