Ruby and Rails Cheat Sheet

Variables

Variables allow us to store values as a name understandable by humans. Unlike constants, the value of a variable can be set and reset/overwritten as many times as you want or need to.

Syntax

```ruby
my_variable     = 10
a_list          = ['Hello', 'I', 'am', 'an', 'array']
my_calculation  = my_variable * 2  #=> Returns 20
complex_var     = a_list.join(' ') #=> "I am an array"
```

Working with Strings

Strings are just a set of characters enclosed in double or single quotes. Double quotes allow you to do things differently than single quotes. Strings, like all other built-in Ruby classes have methods on them to allow you to manipulate their value.

Escape characters and the difference between single and double quotes

```ruby
puts "Hello, I'm a string" #=> "Hello, I'm a string"
puts 'Hello, I\'m a string' #=> "Hello, I'm a string"
puts "1 + 1 is #{1 + 1}" #=> "1 + 1 is 2"
puts '1 + 1 is #{1 + 1}' #=> '1 + 1 is #{1 + 1}'
```

Get input from user in the terminal and concatenate strings

```ruby
gets #=> Gets input from terminal
name = gets.strip #=> Gets input from user and removes newlines and extra whitespace
puts "Hello, #{name}"
### OR ###
puts "Hello " + name #=> Less efficient, verbose, not commonly used syntax
```

Built in string methods

```ruby
name = "tony"
name.upcase #=> TONY
name.capitalize #=> Tony
```

Update a variable *in-place*

```
name = 'sam'
name.capitalize #=> 'Sam'
name #=> 'sam'
name.capitalize! #=> 'Sam'
name #=> 'Sam'
```

More built in methods

```
grocery_list = 'Milk, Eggs, 40oz malt liquor' #=> This is a string
grocery_list.split(', ') #=> ['Milk', 'Eggs', '40oz malt liquor']
```

Arrays

Arrays are basically lists of values. Arrays can hold an infinite number of values all of which can be a different type.They're useful for storing related information. Anything you'd use a list to keep track of in real life would be useful in an array.

Syntax for defining and looping over arrays

```
todo = ['Walk the dog', 'Feed the cat', 'Sacrifice chicken to God of fertility']

todo.each { |item| puts "You need to #{item}" }
## Equivalent to: ##
todo.each do |item|
    puts "You need to #{item}"
end
```

Blocks are unnamed functions. In this example we show how we would loop over an array manually and then show the same function being run by passing a block to the ".each" method of the array class.

```
## Manually looping over an array ##
todo = ['Walk the dog', 'Feed the cat', 'Sacrifice chicken to God of fertility']
def show_todo(task)
    puts "You need to #{task}"
end

for item in todo.length
    show_todo item
end

## .each is the equivalent of the above code ##
todo.each do |item|
    puts "You need to #{item}"
end
```

Conditionals

Conditionals allow us to compare values. Conditionals can have an infinite number of conditions to check but its smart to just keep them to a handful of checks.

```
age = gets.strip
if age >= 18 && age < 21
    "You can buy tobacco but not alcohol"
elsif age >= 18 && age >= 21
    "You can buy alcohol, tobacco, and firearms"
else
    "I got lazy and stopped writing conditions but there are many elsifs we can insert here"
end
```

Sublime Text Shortcuts

See http://billpatrianakos.me/blog/2014/09/11/sublime-text-keyboard-shortcuts/ for a table of useful shortcuts for Sublime Text. Remember, on Windows and Linux, just replace the Command key with the Control key.

Rails Commands

Create a new project

`rails new app_name` - The rails new command takes one argument, the application name. This creates a folder of the same name and sets up all the initial files and folder structure you need to get started.

Start a local server

`rails s` or `rails server` - This will start your rails app and it will be available at http://localhost:3000

Rails Generators

Controllers - `rails g controller ControllerName [method names here]` - Controller names are always plural and camel-case. You may optionally specify a number of actions that the controller should have by default. Doing so will automatically set up default routes for each and will create the corresponding view files as well. Do not include the square brackets in your command. That's just to show that action names are optional.

The controller action names that have special meaning to Rails are: **new, create, edit, update, index, show, destroy**

Models - `rails g model MyModel [attribute:type another_attribute:type]` - Model names are always singular and camel-case. You do not have to specify the attributes and types for your model but you should never leave them off. Model attributes will correspond to a column in your database. These are the valid attribute types: **integer, primary_key, decimal, float, boolean, binary, string, text, date, time, datetime, timestamp**

ERB Syntax

ERB (embedded Ruby) is Ruby code embedded in an HTML file. When you have Ruby code that evaluates to some value that should be displayed on a page, you wrap that code in ERB tags like this: `<%= my_variable_or_code %>`

When you have Ruby code that does some work but does not need to output a value to the screen you leave off the equals sign like this: `<% if true %>…<% end %>`

Example
The following is plain ruby code and its ERB counterpart. Assume that `@users` is an array of user names:

```ruby
@users.each do |user|
    puts "User name is: #{user}"
end
```

```erb
<% @users.each do |user| %>
    User name is: <%= user %>
<% end %>
```

View Helpers

`form_for`
This helper is a method that takes a model object and automatically creates a form that will create, edit, or update an instance of that model. It is good practice to create a _form.html.erb partial and use it on your new and edit pages. This helper will automatically send the form data to the correct URL as long as you are using the Rails action name conventions. For example, if you have a User model with create, edit, update, new actions in a Users controller, then the

form_for helper will automatically fill in the correct form fields on the edit page and point the form's action attribute to the URL responsible for updating the User resource. On the new page, the form_for helper will point the form to the controller's create action. Remember, this helper always needs a model instance passed to it even if it is a blank model instance (like when you call ModelName.new). This helper opens a block that takes one argument. You can use this block to generate the form fields.

link_to

This helper creates a link. The first parameter is the text that will be displayed to the user and the second parameter is the URL that the link should point to.

```
<%= link_to “Signup”, signup_path %>
```

image_tag

Similar to the link helper, this helper takes an image name or path as the first parameter and alt text as a second parameter. If not given a full path, this helper will look for images in your app's assets/images folder.

```
<%= image_tag “avatar.jpg” %>
```