John Larkin
12/28/17
Coursera: Ruby on Rails: An Introduction
Class Notes

Ruby on Rails: An Introduction

Course Highlights:

-

WEEK 1 – Welcome and Setting up the Development Environment

1.1 Course Introduction

- Great for rapid development
- Able to rapidly prototype
- Twitter uses RoR
- Course 1
  - Basic flow of how information comes in
  - Functional application
- Course 2:
  - How you interact with DB
- Course 3:
  - Really dives in on this and how you interact well with MongoDB and RoR
  - Implementation of NoSQL DB
- Course 4:
  - RESTApi is used commonly (FB, Twitter)
  - Full stack web developer – service side could be great, but we still need to care about the client facing side, i.e. frontend
  - User interaction needs to be perfect
  - *HTML, CSS, JavaScript* – design and turn it into real website
- Course 5:
  - AngularJS addresses a ton of the issues with front end web dev
  - Helps to make things faster
  - You'll be able to build a full website where majority of lift is on client side
- Capstone: which I'm not going to do really

Module 1: Setting up the Dev Env

- 3 topics
  - Software installation
    - Installation of Ruby and RoR
  - Coding editors
    - Probably sublime
  - Git
    - Duh VCS
    - Going to deploy to the cloud
    - Going to be helpful when deploying an application

Remote Repos and Github

- Linke remote repo with your local repo
    - **Git remote add alias remote_url**
- **origin – default alias for a cloned repo**
- Most of this is default git

<u>WEEK 2 – Introduction to Ruby</u>

2.0 Getting to Know Ruby

2.0.a Ruby Basics

- History
    - Invented by Yukihiro Matsumoto
    - Popularized by Ruby on Rails framework
- Dynamic, OO, Elegant, expressive, and declarative
- Designed to make programmers happy
- So like this is going to be iterating over something three times:
    - **3.times**
- Ruby basics
    - # to comment
    - 2 space indentation for each nested level
    - Everything is evaluated
- Printing to console
    - **puts**
        - print string to console; as in put string
    - **p**
        - Prints out internal representation of object – great for debugging.
- Variables
    - snake_case
- Constants
    - ALL_CAPS or FirstCap
- Classes (and Modules)
    - CamelCase
- Semicolons
    - Don't need them. Don't include them.
- Extremely expressive

2.0.b. Flow of Control

- if / elsif/ else
    - No parentheses or curly races
    - Use end to close flow control block
    -
- case
- until / unless?

- **unless** – basically, if something is not equal to something else
  - essentially like a not equal to
- **until –** opposite of while; executes until a condition is met

- while / for
- Triple equals
- Flow of control: modifier form (meaning a ton of stuff on one line)
- Only things that are false are:
  - **nil** object
  - **false** object
- Triple equals
  - Use double equal most of the time
  - Equal in its own way
  - Special kind of equals
  - You can use this with regex
- Case expressions
  - Similar to a series of if statements
    - **age = 21**
  - **case**
  - **when age >= 21**
  - **puts …**
  - **when 1 == 0**
  - **puts …**
  - Specify a target next to case and each when clause is compared to target
    - So like
    - **Name = 'Fisher'**
    - **case name**
    - **when # comparison to name**
    - **when # comparison to name**
  - No fall through logic
  - The only case that actually matches gets executed
- For loop
  - **for i in 0..2**
  - **puts i**
  - **end**
  - but most commonly, each / times preferred

2.0.c. Functions

- Functions/methods
  - Function is defined *outside* of a class
  - Method is defined *inside* a class
  - However, in Ruby they are all methods
- Methods

-
    - Parentheses are option when defining and calling a method
    - Used for clarity
- No need to declare type of parameters
- Return keyword is optional – last executed line is returned
- Expressive method names
    - Method names can end with:
        - ? – predicate methods (normally return boolean values)
        - ! – dangerous side effects
    - Also note: number.zero is a method
- Default arguments
    - Pretty simple
    - **def factorial_with_default(n=5)**
        - **n == 0 ? 1 : n * factorial_with_default(n-1)**
    - **end**
- Splat
    - * prefixes parameter inside
    - Can apply to middle parameter or any one

2.0.d Blocks

- Basically, chunks of code that get executed
- Enclosed between either curly braces {} or the **do** and **end** blocks
- Passed in as the last argument
- Convention
    - Use {} when block content is single line
    - Do and end when block content spans multiple lines
    - Often used as iterators
- Examples
    - 1.times { puts “Hello World!”}
    - 2.times do |index|
    -   if index > 0
    -     puts index
    -   end
    - end
- Coding with blocks
    - Implicit:
        - Use **block_given?** to see if block was passed in
        - Use **yield** to “call” the block
        - Ex:
            - **def two_times_implicit**
            -   **return “No block” unless block_given?**
            -   **yield**
            -   **yield**

        - **end**
  - Explicit:
    - Use **&** in front of the last parameter
    - Use **call** method to call block
    - Ex:
      - **def two_times_explicit (&i_am_a_block)**
      - **return "No block" if i_am_a_block.nil?**
      - **i_am_a_block.call**
      - **i_am_a_block.call**
      - **end**
    - *Explicit is a little more direct*
- Summary
  - Just code that you can pass into methods
  - Can either use blocks *implicitly* or *explicitly*

2.0.e Files

- Reading from File
  - File.foreach('test.txt') do |line|
  - puts line
  - p line
  - p line.chomp # chomps off newline character at the end of the line
  - p line.split # array of words in line
- Reading from Non existing file
  - You would get an error pretty much immediately
  - Stops execution
- Handling Exceptions
  - Begin
  - File.foreach('do not exist.txt') do |line|
  - puts line.chomp
  - end
  - rescue Exception => e
  - puts e.message
  - puts "Let's pretend this didn't happen
  - end
- Alternative to Exceptions
  - **if File.exist? 'test.txt'**
  - **File.foreach('test.txt') do |line|**
  - **puts line.chomp**
  - **end**
  - **end**
  - This is good for very simple cases
- Writing to a File

- ○ **File.open("test1.txt", "w") do |file|**
- ○ **file.puts "One line"**
- ○ **file.puts "Another"**
- ○ **end**

- Environment variables
  - puts ENV["EDITOR"]
- Summary
  - Files automatically closed at the end of the block
  - Either use exception handling or check for existence of the file before accessing

2.1 Collections and String APIs
2.1.a Strings

- Strings
  - Single-quote literal strings are ***very*** literal
  - Allow escaping of ' with \
  - Show almost everything as is
  - Double quoted strings interpret special characters like \n and \t
  - Allow string interpolation
- Strings / Interpolation
  - single_quoted = 'ice_cream \n followed by it\'s a party!'
  - double_quoted = "ice_cream \n followed by it\'s a party!"
    - this will show the newline
- Interpolation is only avaialable for double-quoted strings
  - **def multiply (one, two)**
  - **"#{one} multiplied by #{two} equals #{one * two}"**
  - **end**
  - This converts the entire thing to a string and also does the computation
- More strings
  - String methods ending with ! modify the existing string
    - Most others just return a new string
  - Can also use %Q{long multiline string}
    - Same behavior as a double-quoted string
- Example
  - **My_name = " tim"**
  - **Puts my_name.lstrip.capitalize # => Tim**
  - **P my_name # => " tim"**
  - **My_name.lstrip! # destructive! Removed the leading space**
  - **My_name[0] = 'K'**
  - **Puts my_name # => Kim**
  - **Cur_weather = %Q{it's a hot day outside
    grab your umbrellas…}**
  - **Cur_weather.lines do |line|**

- **line.sub! 'hot', 'rainy' # substitute 'hot' with 'rainy**

- Strings API
  - include? other string
- Symbols
  - **:foo** – highly optimized string
  - Constant names that you don't have to pre-declare
  - "Stands for something" string type
- Symbols (cont)
  - Guaranteed to be *unique* and *immutable*
  - Can be converted to a String with **to_s**
  - Can convert from String to Symboll with **to_sym**
- Symbol can be representation of a method name
- Symbols and Strings are similar… you must determine which makes more sense to use
- Summary
  - Interpolation lets you finish your thougth
  - Strings have a lot of really useful API

2.1.b Arrays

- Arrays
  - Collection of object references (auto-expandable – no fixed size)
  - Indexed using **[]**
  - Can be indexed with neg numbers or ranges
  - Heterogeneous types allowed
  - Can use %w{str1 str2} for string array creation
- Examples
  - arr_words = %w{ what a great day today! }
  - puts arr_words[-2] # day
  - puts "#{arr_words.first} - #{arr_words.last}" # what – today!
  - p arr_words[-3, 2] # ["great", "day"] (go back 3 and take 2)
  - p arr_words[2..4]
  - can also do join
- Modifying Arrays
  - Append: **push** or **<<**
  - Remove **pop** or **shift**
  - Randomly pull elements out with **sample**
  - Sort of reverse with **sort!** and **reverse!**
    - **sort** without exclamation returns a new copy of the array
- Examples
  - # You want a stack?
  - stack = []; stack << "one"; stash.push ("two")
  - puts stack.pop # two
  - # you want a queue?

- Queue = []; queue.push “one”; queue.push “two”
- Puts queue.shift # one
- If you specify and insert into an index that is beyond the range, it’s going to create **nils** for everything else

- Other Array Methods
    - **each –** loop through array
        - takes a block
    - **select –** filter array by selecting
        - takes a block
    - **reject –** filter array by rejecting
        - pretty much the opposite of the one above
    - **map –** modify each element in the array
        - maps every element to a new element based on the block passed in
- Important api: **http://ruby-doc.org/core-2.2.0/Array.html**
- Example
    - a = [1, 3, 4, 7, 8, 10]
    - new_arr = a.select { |num| num < 10}
      .reject { |num| num.even?}
    - p new_arr # [1,3,7]
- Summary
    - Arrays API is very flexible and powerful
    - Lots of ways to process elements

2.1.c Ranges

- Used to express natural consecutive sequences
- 1..20, ‘a’..’z’
- Two main rules
    - Two dots → all-inclusive
        - 1..10 (**1 is included, 10 is included)**
    - Three dots → **end-exclusive**
        - 1…10 (1 is included, 10 IS EXCLUDED)
    - The more dots you have, the less you have at the end
- Ranges
    - Very efficient
    - Only start and end stored
    - Can be converted to an array with **to_a**
    - Used for **conditions** and **intervals**
- Examples
    - puts (1…10) === 5.3 # true
    - puts (‘a’…’r’) === “r” # false, end –exclusive
- Summary
    - Useful for consec sequences

- Convert a range to an array for more functionality

2.1.d Hashes

- Hashes
  - *Indexed collection* of object references
  - Created with either **{}** or **Hash.new**
  - Also known as **associative arrays**
  - Index(key) can be anything
    - Not just an int as is the case with arrays
  - Accessed using **[]**
  - Values set using
    - **=>** (creation)
    - **[]** (post creation)
- Example
  - editor_props = { “font” => “Arial”, “size” => 12, “color” => “red}
  - editor_props.length
- Hashes
  - Accessing a value in the Hash for which an entry does not exist
  - **nil** is returned
  - **BUT if you create a Hash with Hash.new(0), then 0 is going to be returned instead.**
- Example
  - **word_freq = Hash.new(0)**
  - **sentence = “Chicka chicka boom boom”**
  - **sentence.split.each do |word|**
  - **word_frequency[word.downcase] += 1**
  - **end**
- More hahse
  - The order of putting things into **Hash** maintained
  - If using symbols as keys, can use symbol: syntax
  - If a Hash is the last argument to a method, you can drop the curlies
- Block and Hash Confusion
  - a_hash = {:one => “one”}
  - puts a_hash
  - # can’t do puts { :one => “one”}
  - # ruby gets confused and think it’s a block
  - To get around this you can use parenthesis
  - Or you can just drop the blocks all together

2.2 Object Orientated Programming in Ruby

2.2.a Classes

- OO Review

- Identify things your program is edaling with
- Classes are **things** (blueprints)
    - Containers of methods
- Objects are instances of those things
- Objects contain instance variables (state)

- Instance variables
    - **Begin with @**
    - Not declared
        - Spring into existence when first used
    - Available to all instance methods of the class
- Object creation
    - Classes are factories
        - Calling **new** method creates an instance of class
        - **new causes initialize**
- Example
    - class Person
    -   def initialize (name, age) # constructor
    -     @name=name
    -     @age = age
    -   end
    -   def get_info
    -     @additional_info = "Interesting"
    -     "Name: #{@name}, age: #{@age}"
    -   end
    - end
- Accessing Data
    - **Instance variables are private**
        - **Cannot be accessed from outside th4e class**
    - **Methods have public access by default**
    - **To access instance variables, need to define getters/setter**
- Example
    - def name
    -   @name
    - end
    - def name= (new_name)
    -   @name = new_name
    - end
- Easier syntax for accessing data
    - **attr_accessor – getter and setter**
    - **attr_reader – getter only**
    - **attr_writer – setter only**
- Example

- class Person
-   attr_accessor :name, :age
- end

- Sometimes we want to use a more intelligent constructor
- Self
    - Inside instance method, **self** refers to the object itself
    - Usually using **self** for calling other methods of the same instance is extraneous
    - Sometimes using **self** is required
    - Outside instance method def, **self** refers to the class itself
- Summary
    - Objects are created with new
    - Use the short form for setting/getting data
    - Don't forget self when required

2.2.b Class Inheritance

- || operator evaluates the left side; if true, returns it, else it returns the right side
- @x = @x || 5 will retrun 5 the first time and @x the next time
- short form
    - @x ||=5
- This is really helpful for setting an instance variable to something the first time
- Class Methods
    - Invoked **ON** the class (as opposed to an instance of the class)
    - Self OUTSIDE of the method definition refers to the **Class** object
    - Three ways to define class methods
        - Class variables begin with @@
- Example
    - class MathFunctions
    -   def self.double(var)
    -     times_called; var * 2
    -   end
    -   class << self
    -     def times_called
    -       @@times_called ||=0; @@times_called += 1
    -     end
    -   end
    - end
    - def MathFunctions.triple(var)
    -   times_called; var * 3
    - end
- Class Inheritance
    - Every class implicitly inherits from Object
    - Object inherits from BasicObject

- No multiple inheritance
  - Mixins are used instead
- Class SmallDog < Dog
-   Def bark
-     “barks quietly”
-   end
- end

2.2.c Modules

- Module
  - Container for classes, methods, and constants (or other modules)
  - Like a Class but cannot be instantiated
- Module as Namespace
  - module Sports
    - class Match
      - attr_accessor :score
    - end
  - end
  - module Patterns
    - class Match
      - attr_accessor :complete
    - end
  - end
  - match1 = Sports::Match.new
  - match2 = Patterns::Match.new
- Module as Mixin
  - Interfaces in OO
  - Contract defines what a class could do
  - Mixins provide a way to share ready code among multiple classes
- Example
  - module SayMyName
    - attr_accessor :name
    - def print_name
      - puts “Name: #{@name}”
    - end
  - end
  - class Person
    - include SayMyName
  - end
  - person = Person.new
  - person.name = “Joe”
  - person.print_name = # Name:joe

- Enumerable Module
  - map, select, reject, detect, etc
  - Used by Array class and many others
  - Provide an implementation for **each** method
  - And then you can include it in your own class
- Example
  - class Player
    - attr_reader :name, :age, :skill_level
    - def initialize (name, age, skill_level)
      - @name = name
      - @age = age
      - @skill_level
    - end
    - def to_s
      - "<#{name}: #{skill_level}(SL), #{age}(AGE)>"
    - end
  - end
- Enumerable in Action
  - require_relative 'player'
  - require_relative 'team'
- Modules allow you to mixin useful code into other classes
- Require relative is useful for including other ruby files relative to the current ruby code

2.2.d Scope

- Methods and classes begin new scope for variables
- Example
  - v1 = "outside"
  - class MyClass
  -   def my_method
  -     p v1 # exception thrown
  -     p local_variables # prints out a list of all the local_variables
  - end
- Scope constants
  - Pretty intuitive
- Scope block
  - Blocks inherit outer scope
  - Block is a closure
    - Remembers the context in which it was defined and then uses that context whenever
- Block – local scope
  - A variable created inside the block is only available to the block
  - Params to the block are always local to the block

John Larkin
12/28/17
Coursera: Ruby on Rails: An Introduction
Class Notes

2.2.e Access Control

- Three levels of access control
- Controlling access
- How private is private access?
- Access control
    - When designing, how much do you want to expose?
    - Encapsulation: try to hide the internal representation of the object so you can change it later
    - Three levels
        - Public
        - Protected
        - Private
- Specifying access control
    - Two ways
        - Specify public projected or private
            - Everything until the next access control keyword will be of that level
        - Define the methods regularly and then specify public, private, protected access level and **list** the comma separated methods under those levels using method symbols
    - Example
        - class MyAlgorithm
        -   private
        -     def test1
        -       “Private”
        -     end
        -   protected
        -     def test2
        -       “Protected
        -     end
        - end
    - Example 2
        - class Another
        -   def test1
        -     “Private, as declared later”
        -   end
        -   private :test1
        - end
    - Access control meaning
        - Public methods – no access control is enforced
        - Protected methods – **can be invoked by the objects of defining class or**

        subclasses
    - Private methods – cannot be invoked with an explicit receiver
        - Setting an attribute can be invoked with explicit receiver
- Summary
    - Public and private access controls are used the most

2.3 *Unit Testing with RSpec*
2.3.a Introduction to Unit Testing

- Ensure your code works
- Serves as documentation for devs
- Refactor to make sure you didn't break anything
- Enter Test::Unit
    - Ruby takes testing very seriously
    - Has Test::Unit shipped with it
    - Ruby 1.9 stripped Test::Unit to a minimum
    - Member of the XUnit family (Junit, CppUnit)
    - Basic idea: extend **Test::Unit::TestCase**
    - Prefix method names with **test_**
    - If one of the methods fails, others keep going (good thing)
    - Can use **setup()** and **teardown()** methods for setting up behavior that will execute before **every** test method
- Example
    - class Calculator
        - attr_reader :name
        - def initialize(name)
            - @name=name
        - end
        - def add(one,two)
            - one – two
        - end
    - Then your testing would look like:
        - require 'test/unit'
        - require_relative 'calculator'
        - class CalculatorTest < Test::Unit::TestCase
            - def setup
                - @calc = Calculator.new('test')
            - end
            - def test_addition
                - asset_equal 4, @calc.add(2,2)
            - end
    - then run **ruby calculator_test.rb**
    - Also good mneumoic to remember is **EACH**

        - Expected first, then actual

2.3.b Introduction to RSpec

- Testing with RSpec
    - Test::Unit "does the job" but it would be nice if tests would be more descriptive, more English-like
    - The writing of the tests is more intuitive as well as the output from running the tests
- Installing
    - Easy… **gem install rspec**
- **describe()**
    - Set of related tests (a.k.a. example group)
    - Takes either a **String** or **Class** as an argument
    - All specs must be inside a describe block
    - No class to subclass
- **before()** and **after()** methods
    - before and after methods are similar to setup and teardown
    - Can pass in either **:each** or **:all** (infreq used) to specifyc whether the block will run before/after each test or once before/after all tests
    - **before :all** could be useful if you only want to connect to DB once
- **it()**
    - Main logic happens inside the it() method
- Example
    - **require 'rspec'**
    - **require_relative '../calculator'**
    - **describe Calculator do**
        - **before { @calculator = Calculator.new('RSpec calculator')}**
        - **it "should add 2 numbers correctly" do**
            - **expect(@calculator.add(2,2)).to eq 4**
        - **end**
        - **it "should subtract 2 numbers correctly" do**
            - **expect(@calculator.subtract(4,2)).to eq 2**
        - **end**
    - **end**
- Summary
    - RSpec makes testing more intuitive

2.3.c RSpec Matchers

- Hands to and not_to methods on all outcome of expectations
- **to()/not_to()** methods take one parameter – a matcher
    - **be_true / be_false**
    - **eq 3**

    - **raise_error(SomeError)**
- **be_predicate – boolean**
    - If the object on which the test is operating has a predicate method, you auto get the **be_predicate** matcher
    - **Be_nil** is a valid matcher because every predicate method has a :nil? Method

WEEK 3 – Introduction to Ruby on Rails

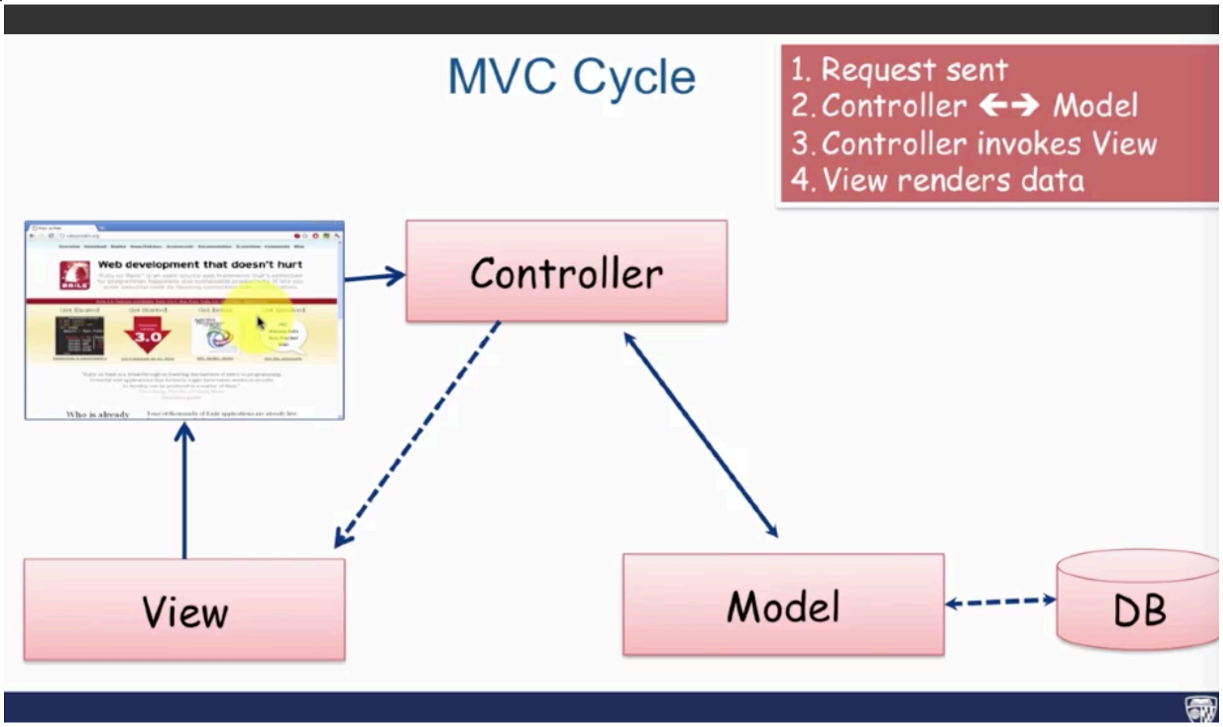
3.0 Core Concepts

3.0.a Welcome to Module 3: Introduction to Ruby on Rails

- Core principles
- Model View Controller
    - Principle that applies to a lot of web frameworks as well
- Convention Over Configuration
    - Following conventions helps applications be built very quickly

3.0.b Introduction to Rails

- **Framework for making dynamic web applications**
- Dynamic
    - Content that is gotten from a database or something like that
    - Html is just going to be static (i.e. not dynamic)
    - Created by David Heinemeier Hansson
        - Also a racecar driver
- Who is Using Rails?
    - Hulu
    - Twitter
    - Github
    - White pages
- Why use Rails?
    - **Convention Over Configuration (COC)**
    - Less code to write
    - Learn it once and then know what to expect the next time
- Why Use Rails?
    - **Database Abstraction Layer**
    - No need to deal with low-level DB details
    - No more SQL (Almost)
    - ORM
        - Object Relational Mapping
        - Abstracting the code to interact with DB using Ruby
        - Mapping your database to your Ruby Classes
- Why else?
    - Agile-friendly
    - **DRY** principle

- Cross-platform
- SQLite
    - Rails uses SQLite for database by default
    - **<u>Self-contained, serverless, zero-configuration, transactional, relationsal SQL database engine</u>**
    - Claim: Most widely deployed SQL database engine in the world
- MVC: Model View Controller
    - Well-established software pattern used by many web and desktop frameworks
    - **Separation of concerns**
    - **Model – represents the data the application is working with (and poss business logic)**
    - **View – representation of that data (visually)**
    - **Controller – interaction between model and view**
- MVC Cycle
    - 

MVC Cycle

1. Request sent
2. Controller ←→ Model
3. Controller invokes View
4. View renders data

- Summary
    - Rails is good with **RAPID PROTOTYPING**
    - MVC and COC enable you to **think less and do more**

3.0.c Creating your First Application

- How to create and run your app
- Directory structure (CoC)
- Adding static pages to your application
- Creating First App

- ○ **rails new appname**
- ○ **rails new –h** for more operations
- ○ run

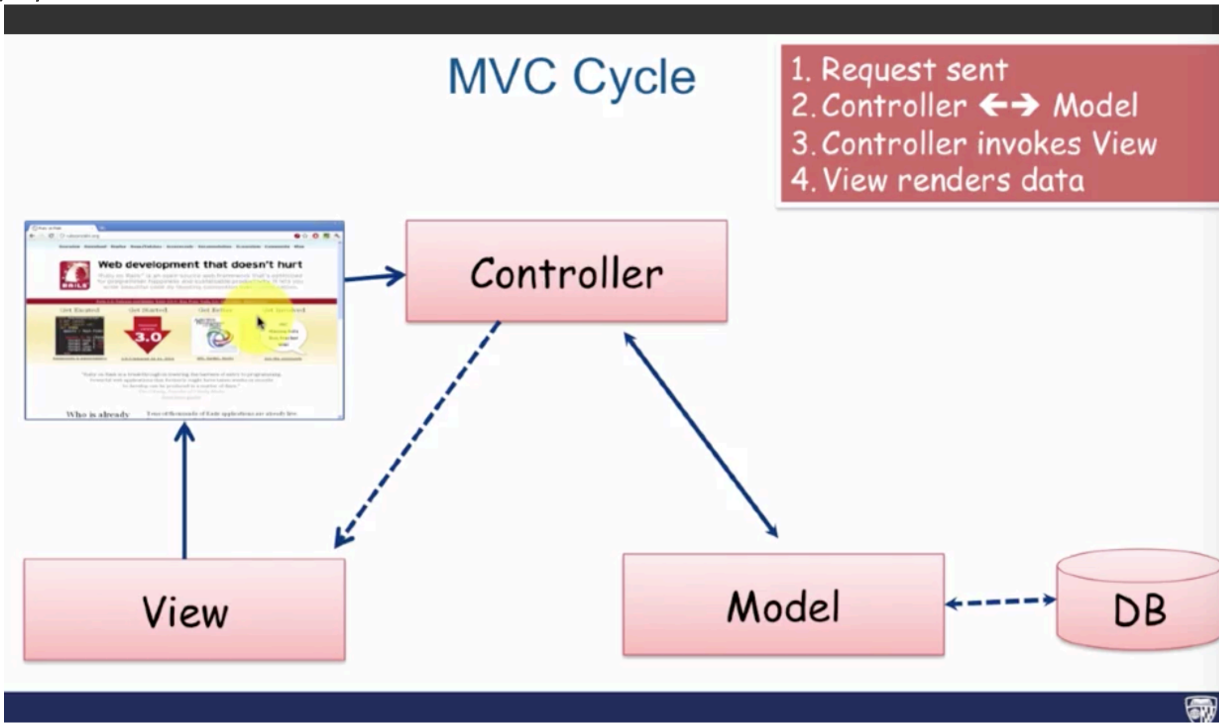
- **Bundler (gems manager)**
  - Cleans up the house and resolves dependency issues
- Version Control Your Rails App
  - Rails automatically generate .gitignore inside repo
  - **cd my_first_app**
  - **git init**
  - **git add .**
  - **git commit –m "Initial commit"**
- Running the App
  - **Rails alos provides a built-in web server**
  - **rails server**
- Running the App (cont)
  - Good at holding your hand
  - 1 use **bin/rails generate** to create your models and controllers
  - 2 set up a root route to replace the default place
  - 3 Configure your database
- Directory Structure Convention
  - app/ directory – controllers, views, models, helpers (most of the time)
  - config/ - which database are you going to be using (and username and password)
  - db/ - files related to your db and migration scripts (how to change from one database to another)
  - public/ - static files. Html files. All that boring shit.
  - Gemfile
  - Gemfile.lock – dependencies managed by Bundler
- public/hello_static.html
  - Server looks into **public** directory before looking anywhere else
  - So… if we want to add a completely static web page to our application – we can add it under **public** directory

3.0.d Controller and View

- How to generate controller
- Actions
- Embedded Ruby (ERB)
- Generating a Controller
  - Controllers contain **actions** (Ruby methods) and orchestrate web requests
  - Rails can quick generate a controller and 0 or more actions with associated views
  - **rails generate controller controller_name [action1 action2]**
- Generating a Controller Example
  - **rails g controller greeter hello**

- ERB (Embedded Ruby)
  - Looks like html but has an .erb extension
  - ERB is a templating library (similar to jSP) that lets you embed Ruby into your HTML
  - Two tag patterns to learn:
    - **<% …ruby code… %>** - evaluate Ruby code
    - **<%= …ruby code… %> -** output evaluated Ruby code
  - Whole point is to mix html static and Ruby code
- New hello.html.erb
  - <% random_names = ["Alex", "Joe"] %>
  - <h1> Greetins, <%= random_names.sample %></h1>
  - <p>The time now is <%= Time.now %></p>

3.0.e Routes

- Routing
- Rake
- How to analyze current routes
- Routes
  - Before the controller can orchestrate where the web request goes, the request needs to get routed to the controller
  - The route for hello action was auto generated with the rails g controller
- MVC(R) Cycle
  - 

- routes.rb

- All the routes need to be specified in the **config/routes.rb** file
- Let's add the route for the goodbye action
- It'l look like
  - **Rails.application.routes.draw do**
    - **get 'greeter/hello' => "greeter#hello"**
      - This syntax is saying go to controller / action
      - So you can map different things to the name if you do this
      - **'greeter/hello' => 'greeter#whatsgood'**
    - **get 'greeter/goodbye'**

- Rake
  - **Ruby's make**
  - No XML – written entirely in Ruby
  - Rails uses rake to automate app-related tasks
    - Database, running tests, etc
  - **rake –tasks**
- Individual Rake Task
  - Can zero-in on an individual rake task and what it does with **–describe** flag
  - **rake –describe task_name**
  - **rake –describe routes**
    - Print out all defined routes in match order, with names. Target specific controller with CONTROLLER=x
- **Rake Routes**
  - **rake routes**
- Summary
  - Router directs the request to the right controller
  - **rake routes** lets you see which routes are currently defined

3.1 Diving Deeper into Rails
3.1..a Moving Business Logic Out of View

- Moving business logic out of View and into Controller in order to comply with MVC
- Action Methods Inside Controller
  - If the action (method) is not really doinganything (i.e. empty), we can remove it
  - As long as there is a proper route defined and there is a properly named view file/template, the action method does not have to be there… Rails will find the correct template by convention
- Controller: New Look
  - **class GreeterController < ApplicationController**
    - **# def hello**
    - **# end**
    - **# def goodbye**
    - **# end**
  - **end**

  - This will still work totally find
  - So what's the point of having them there?
  - Business logic does not belong in the **View**
- Moving Business Logic Out
  - Instance variables from the controller are available **inside the view**
  - **class GreeterController < ApplicationController**
    - **def hello**
      - **random_names = ["Alex", "Joe", "Michael"]**
      - **@name = random_names.sample**
      - **@time = Time.now**
      - **@times_displayed ||=0**
      - **@times_displayed += 1**
    - **end**
  - **end**
- Instance Variables in Rails
  - Unlike some frameworks, **<u>you cannot "store" values in the controller's instance variables in between requests</u>**
  - Alternatives?
    - Session (store in the http session)
    - Database (store in the database)
- Summary
  - Keep business logic **OUT of the view**
  - Instance variabels in the controller are available to view
  - Instance variables do not stick around between requests

3.1.b Helpers

- Helpers and using **link_to**
- Helpers
  - We've made the current time available through @time instance variable
  - What if we wanted to format that time?
    - Should it go into view? (then non-reusable)
    - Controller? Should be "view" agnostic
- Helpers
  - greeter_helper.rb module generated
  - Let's add a helper method
  - Example
    - **module GreeterHelper**
      - **def formatted_time(time)**
        - **time.strftime("%I:%M%p")**
      - **end**
    - **end**
    - Available to ALL views

  - Then you can put it in the hello.html.erb file
- Rail's Built-In Helpers: **link_to**
  - **link_to name, path**
    - Hyperlink generator that displayed the **name** and linked to the **path**
    - Path could either be a regular string or a route defined in the routes.rb ending with **_url** or **_path**
  - Instead of specifying a path, you specify a variable, automatically changes your page if the variable changes
  - **_url** and **_path** used interchangeable, but according to the spec full path is required in cases of redirection
- **link_to** in action
  - **#in hello.html.erb**
  - **<p><%= link_to "Google", "https://www.google.com" %></p>**
  - **<p><%= link_t "Goodbye", greeter_goodbye_path %></p>**
  - greeter_goodbye derived from routes.rb (see Prefix column in rake routes)
- Summary
  - Helpers are "macros" / "formatters" for your view
  - When using **link_to** there is no need to change things if a path changes

3.2 Building a Ruby on Rails Application
3.2.a Introduction to HTTParty

- Going to look at Ruby gems
- How to use HTTParty Ruby gem
- RubyGems
  - Just a package manager
- What are Restful Web Services?
  - Simple web services implemented using HTTP (and principles of REST) that:
    - Have a base URI
    - Support a data exchange format like XML or JSON
    - Support a set of HTTP operations (GET, POST, etc)
  - Flipping web on it's head
  - Thinkg about web as more of an MVC pattern
    - Really just stores those resources and you can get it in multiple different types of formats
    - Html isn't great to parse but xml and json are
- HTTParty Gem
  - Restful web services client (think your browser)
  - Browser is just your client from a web server
  - **Automatic parsing of JSON and XML into Ruby hashes**
  - Provides support for
    - Basic http authentication
    - And default request query params

- Lots of Restful APIs Out There
    - Every self respecting web service normally has some restful api that it provides
    - In addition to the html
    - Most popular APIs?
        - Facebook
        - Google Maps
        - Fitbit
        - LinkedIn
        - Bloomberg
        - Twitter
        - Instagram
    - The html is just one of the formats of information that's stored on websites
- HTTParty Usage
    - **include HTTParty** module
    - can specify
        - **base_uri** for your requests
        - **default_params** (API developer key for example)
        - **format** to tell it which format things are coming in
    - Coursera itself has a Restful API
- Specify a **q** request parameter
- First param is specified by ? and then others specified by &
- HTTParty Example
    - **require 'httparty'**
    - **require 'pp' # pretty print**
    - **class Coursera**
        - **include HTTParty**
        - **base_uri 'https://api.coursera.org/api/catalog.v1/courses'**
        - **default_params fields: 'smallIcon,shortDescription' q: 'search'**
        - **format :json**
        - **def self.for term**
            - **get("", query: {query: term})["elements"]**
        - **end**
    - **end**
    - **pp Coursera.for "python"**
    - Get back a giant hash which has elements as it's key

3.2.b Bundler

- Provides a consistent environment for Ruby projects by tracking and installing the exact gems and versions that are needed
- Bundler
    - Lets you specify gems for the Rails app inside Gemfile
    - Preffered way to manage gem dependencies

    - **bundle install** or **bundle** after specifying a new gem in the Gemfile
    - You can instruct rails through Gemfile to only load certain gems in specific Rails environment
- Which version of Gem?
    - **gem “thin”, “~>1.1”**
    - called the perssimistic version constraint
        - drops the final digit, then increments to get the upper limit version number
    - so that top statement would be equiv to
    - **gem “thin”, “>=1.1”, “< 2.0”**
- Bundler require
    - Occasionally, the name of the gem to be used inside **require** statement is different than the name of the gem
    - **gem ‘sqlite3-ruby’, require: ‘sqlite3’**
- **Gemfile – Example**
    - **source ‘http://rubygems.org’**
    - **gem ‘rails’, ‘4.2.3’**
    - **gem ‘sqlite3’**
    - Can change the version of rains just through bundle update
    - **Gemfile.lock**
        - This file contains the actual gem versions
- Summary
    - Bundler manages gem dependencies
    - Loads gems on application startup

3.2.c Rails and HTTParty Integration

- HTTParty Integration – Gemfile
    - Specify version of httparty
    - **gem ‘httparty’, ‘0.13.5’**
    - Then shutdown server
    - Run **bundle**
    - Then you need to restart the server
- Coursera Model
    - Based on convention, controllers are named plural and model is singular
- Courses Controller
    - Fill in **index** action
- courses/index.html.erb
    - image_tag creates a link to an image

3.2.d CSS, Parameters & Root Path

- Adding basic styling to our view
- Making the app **more dynamic** with a request parameter

John Larkin
12/28/17
Coursera: Ruby on Rails: An Introduction
Class Notes

- Routing the root path
- Layout
    - **views/layout/application.html.erb** serves as view's container (unless overridden)
    - Each individual page gets displayed inside the body of this page
    - You do need to specify which css files you want to include
- Terms
    - Zebrafiy – when you switch between backgrounds
- Adding Some CSS
    - When you generate a controller, you get the controller name + .scss
    - SCSS – it's all sass
        - Sass super-set of normal CSS
        - You could use regular css inside sass files
    - **courses.scss**
        - table {
            - border-collapse: collapse;
        - }
        - td {
            - padding: 12px;
        - }
        - .even {
            - background-color: #D6E55
        - }
    - Then you need to modify view to include CSS classes
    - **index.html.erb**
        - **<h1> Searching for - <%= @search_term %></h1>**
        - **<table border="1">**
            - **<tr>**
                - **<th>Image</th>**
                - **<th>Name</th>**
            - **</tr>**
            - **<% @courses.each do |course| %>**
                - **<tr class=<%= cycle('even', 'odd') %>>**
                    - **<td><%= image_tag(course["smallIcon"])%></td>**
                    - **<td><%= course["name"] %></td>**
                    - **<td><%= course["shortDescription"] %></td>**
                - **</tr>**
            - **<% end>**
        - The cycling bit literally comes through even and odd
- **params** helper
    - it would be nice to specify the search term
    - Use **params** Hash to retrieve the value (name of param becomes a symbol/key in

    Hash
  - Returns nil if request param is not passed in
  - No changes to the model or the view, **only** to the Controller
- Example
  - **class CoursesController < ApplicationController**
    - **def index**
      - **@search_term = params[:looking_for] || 'jhu'**
      - **@courses = Coursera.for(@search_term)**
    - **end**
  - **end**
- This will default to 'jhu' if nothing is passed in
- One Final Twist: RootPath
  - What if we want to specify the root path?
  - We can specify it to go to the index action
  - **Just modify routes.rb**
    - **Root 'courses#index'**
    - This means courses controller, action index
- Summary
  - Minor CSS changes can dramatically enhance the app
  - **params** helper parses request parameters
  - Easy to change the root path by tweaking **routes.rb**

3.3 Deploying to Heroku and Verification
3.3.a Deploying to Heroku

-