GENERAL ASSEMBLY

INTRO TO RUBY ON RAILS

Jonathan Young

HELLO!

JONATHAN YOUNG

- Software Engineer @ Springleaf Financial
- Part-time instructor for General Assembly
- Teach for America alumnus
- Northwestern University grad
- Coding bootcamp grad

AGENDA - DAY 1

- How the Internet works
- Command Line Basics
- Git / Github
- Ruby basics

AGENDA - DAY 2

- MVC for web apps
- HTML — Building blocks
- Ruby on Rails
- Creating our first web app
- Deploying our first app to Heroku

INSTALLATION AND SETUP

DO YOU HAVE THE FOLLOWING INSTALLED?

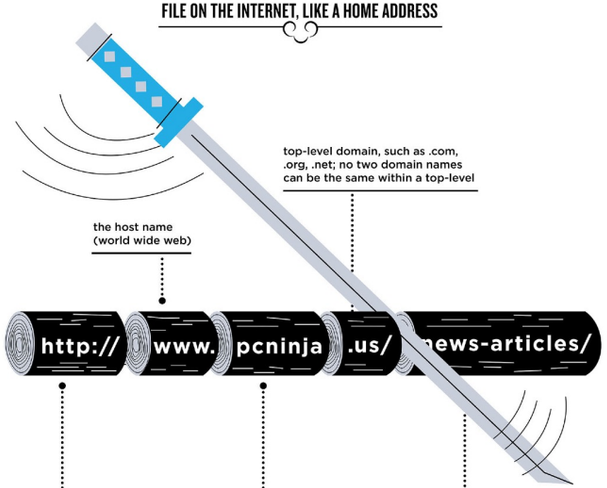
- Sublime Text 2
- Ruby (either rvm or rbenv) through the links provided by Christa
- Rails 4 or higher
- iTerm (optional - can use Terminal)
- Git
- A Github account

HOW DOES THE INTERNET WORK?

A BROAD OVERVIEW

DISSECTING A URL

URL = UNIFORM RESOURCE LOCATOR, THE ADDRESS OF A SPECIFIC WEBSITE OR FILE ON THE INTERNET, LIKE A HOME ADDRESS



The URL really says, "Use the Web (http://) to find a host server named 'www' in the 'pcninja.us' network, and look in the 'news-articles' folder so I can pull out a particular file.

HOW THE INTERNET WORKS

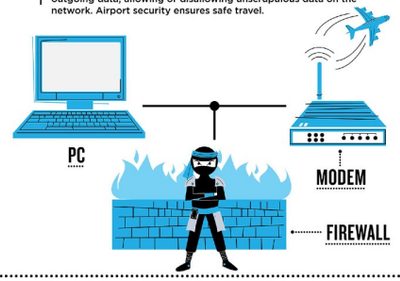
SO HOW EXACTLY DOES THE INTERNET WORK? What's going on in that cloudy, tangled web? A lot of little ninja are working at super speeds to bring you the data you seek!

STEP 1 DEPARTURE

You type 'pcninja.us' into the **web browser** of your computer. Hop in the ninja mobile, and prepare for an adventure!

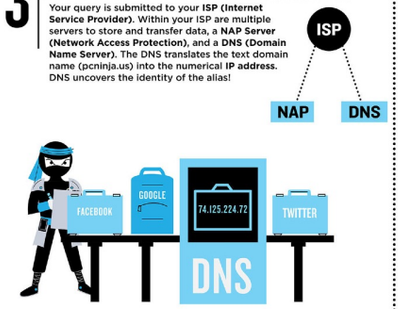
STEP 2 THE AIRPORT

Your computer is connected to the Internet through a **modem** and/or **router**, a jumping-off point to other networks in the world. **Firewalls**, in your browser and/or modem, monitor incoming and outgoing data, allowing or disallowing unscrupulous data on the network. Airport security ensures safe travel.



STEP 3 CUSTOMS

Your query is submitted to your **ISP (Internet Service Provider)**. Within your ISP are multiple servers to store and transfer data, a **NAP Server (Network Access Protection)**, and a **DNS (Domain Name Server)**. The DNS translates the text domain name (pcninja.us) into the numerical **IP address.** DNS uncovers the identity of the alias!

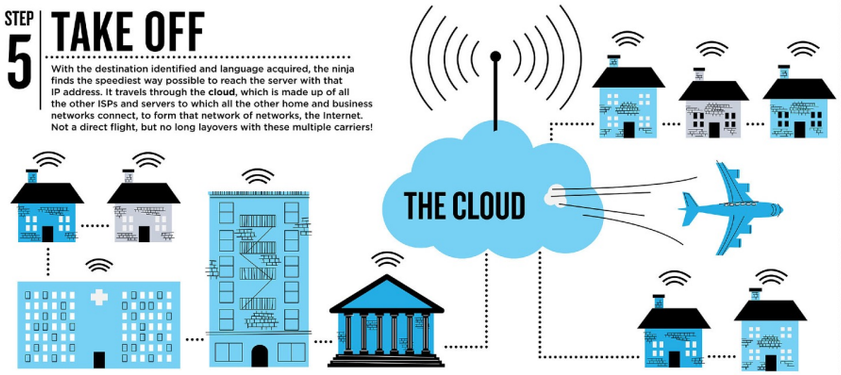


STEP 4 BUCKLE UP

Your browser maps itself to the desired IP address and establishes the hypertext transfer protocol (http), or language used to communicate on the **World Wide Web**. Good idea to take a pocket translator!

STEP 5 TAKE OFF

With the destination identified and language acquired, the ninja finds the speediest way possible to reach the server with that IP address. It travels through the **cloud**, which is made up of all the other ISPs and servers to which all the other home and business networks connect, to form that network of networks, the Internet. Not a direct flight, but no long layovers with these multiple carriers!

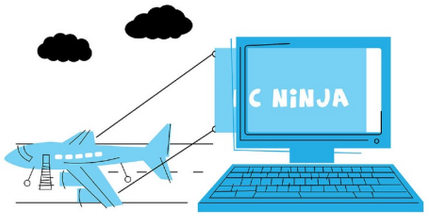


STEP 6 LANDING

Jumping from server to server on the Web, the ninja finally locates the target server hosting the target IP address for pcninja.us. A connection is established with that website and your computer. Please make sure your seats are in the upright position, we're ready to land!

STEP 7 INCOMING!

The ninja makes an even quicker return journey, bringing to your computer screen the graphical **website** of pcninja.us, which is full of data, pictures, and contact information. Thank you for flying with PC Ninja, and enjoy your browse!



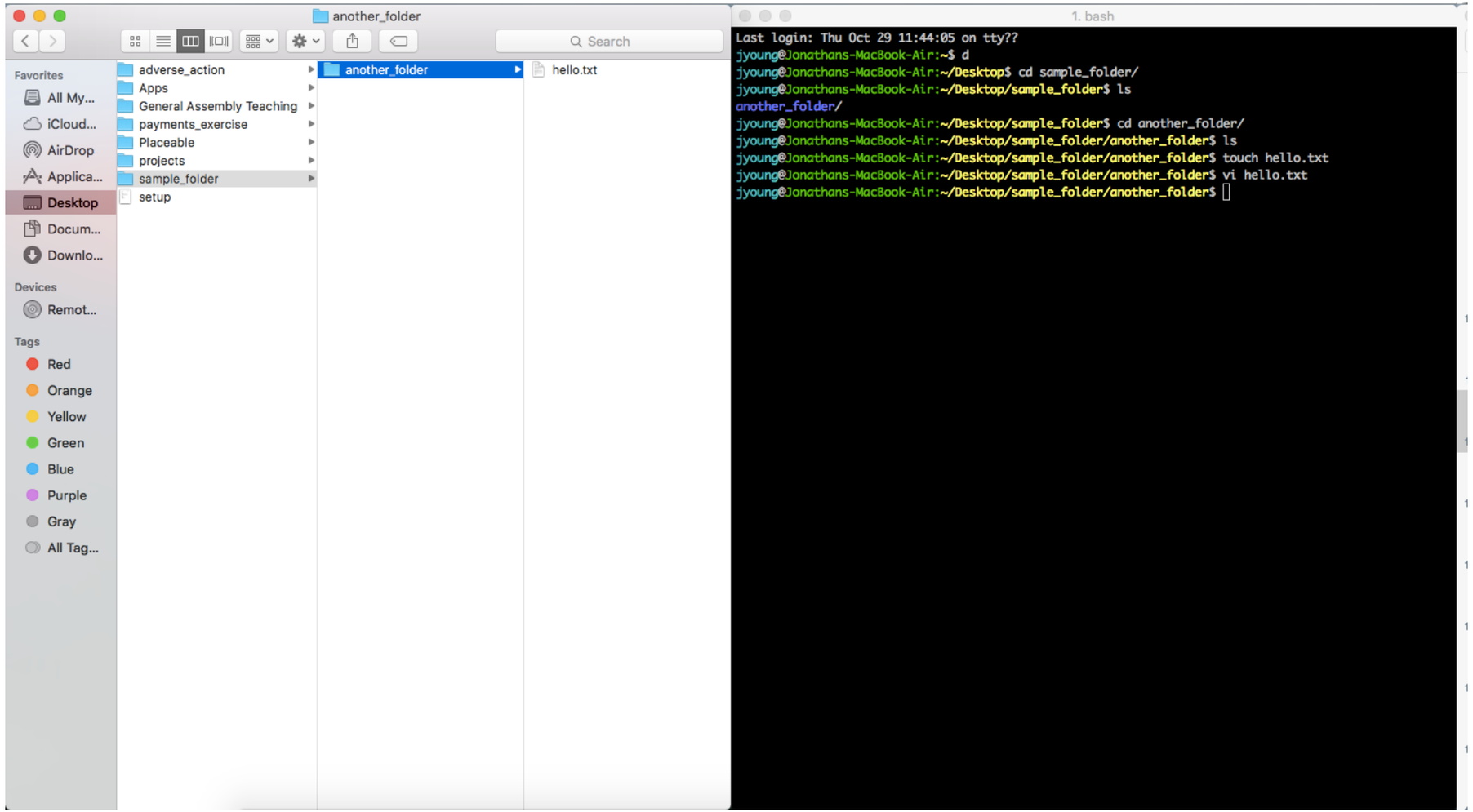
A BROAD OVERVIEW - WHAT YOU NEED TO KNOW

- You make a request by typing in a URL
- URL **!=** letters but URL **==** numbers (IP address)
- www.google.com is actually 64.233.160.0 (one of their many IP addresses)
- ISP + DNS finds what server (computer) that IP address lives on
- Important to note: the “cloud” is actually someone else’s computer. Nothing else.
- The server will accept your request and if everything looks good, it will return you a response that your ISP will relay back to your screen

THE COMMAND LINE

THE COMMAND LINE

Non-Developer Workflow Developer Workflow

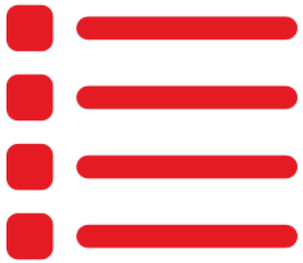


WHAT CAN I DO WITH THE COMMAND LINE?

ADD FOLDER

REMOVE FOLDER

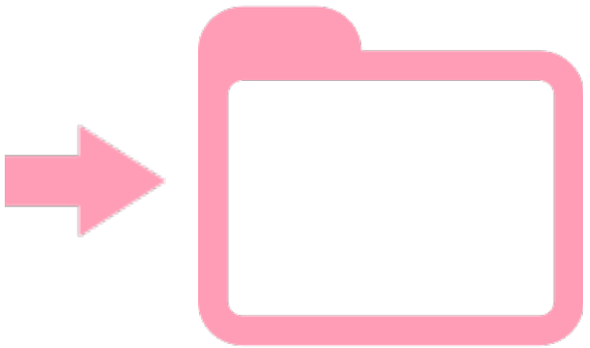
LIST CONTENTS



ADD FILE

REMOVE FILE

NAVIGATE TO ANOTHER FOLDER



THE COMMAND LINE

Commands		
Action	Type	What does it do?
mkdir	`Folder`	`Creates new folder`
touch	`File`	`Creates new file`
rmdir or rm -rf	`Folder`	`Removes a folder`
rm	`File`	`Removes a file`
ls	`Folder`	`Lists everything in a folder`
cd	`Folder`	`Changes what directory you are in`

COMMAND LINE — CODE ALONG!

COMMAND LINE — YOUR TURN

- Make a folder on the Desktop called ‘command_line_exercise’
- Inside that folder, make a folder called ‘exercise_1’
- Create an empty file called ‘awesome.txt’
- Go back out one directory and create another folder called ‘exercise_2’
- Change into that directory
- Change back out
- Delete the ‘exercise_2’ folder
- List everything in the current directory you are in
- Change into the ‘exercise_1’ folder and list everything in it
- Call me over!

GITHUB: SOURCE CONTROL

GIT? GITHUB?

- Git != Github
- **Git** is a type of version control for developers so that multiple people can work on multiple/the same project(s) at the same time on disposable branches
- Master branch
- Git records every change that a developer makes with a timestamp
- **Github** is where Git records are stored

INITIALIZE A REPOSITORY

- **Repository -** Place where the history of your work is stored
- First command you'll run in a new project
- Creates .git subdirectory in the project root
- Only needs to be executed **once** per project

```
$ git init
```

CLONE A REPOSITORY

- if you are contributing to an already existing project, you will need to clone it
- Let’s clone a repo together!
- Only needs to be executed **once** per project

```
$ git clone
```

GIT ADD

- Adds a change in the working directory to the staging area.
- Tells git that you want to include those changes in the next commit

GIT COMMIT

- Saves a copy or ‘snapshot’ of the current state of the project.
- These are ‘safe’ versions of the project - you can go back to them and git won’t do anything to them unless you tell it to.

```
$ git add
```

```
$ git commit
```

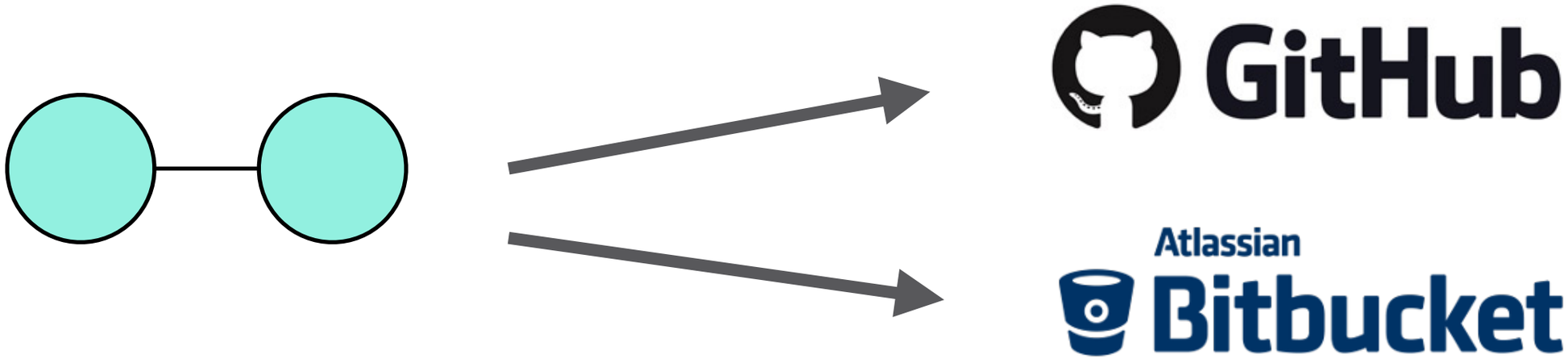
GIT STATUS

- Lists which files are staged, unstaged, and untracked.

```
$ git status
```

GIT PUSH

- Transfer local commits to remote repository




```
$ git push
```

THE COMMAND LINE

Commands	
Action	What does it do?
git init	`Makes the current repo a git repo`
git add [files_here]	`Groups changed files so that they can be committed`
git commit -m ‘Some commit message’	`Saves a snapshot of the project at a particular timestamp`
git push [remote] [branch]	`Sends/pushes your commits to Github`
git pull [remote] [branch]	`Pulls changes from Github to your local copy`

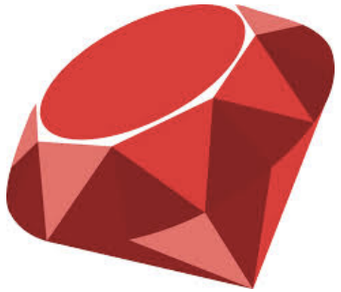
GITHUB — CODE ALONG!

GITHUB - YOUR TURN!

- Make a folder on the Desktop called ‘github_exercise’ and initialize it with Git
- Create a file called hello.txt
- Write your name on line 1 and your occupation on line 2
- Add/commit the file with a commit message of “added name and occupation”
- Push those changes to Github
- Inside of hello.txt, add a third line where you write what your favorite food is
- Add/commit that change with a commit message of “added favorite food”
- Push those changes to Github
- Call me over!

RUBY

RUBY != RUBY ON RAILS



- Ruby is a programming language
- Focus on simplicity and productivity.
- It has an elegant syntax that is natural to read and easy to write.

- Rails is a framework written in the Ruby language
- Open source web application framework that runs on Ruby
- Allows you to create web applications that query a database.
- Makes assumptions about what every developer needs when starting a project
- Write less, accomplish more

AN ELEGANT AND ARTFUL LANGUAGE

- Created by Yukihiro “Matz” Matsumoto - who often said that he was “trying to make Ruby natural, not simple,” in a way that mirrors life.
- It looks and reads a lot like regular English
- Ruby is an object-oriented language — everything in Ruby is an object

“Ruby is designed to make programmers happy”

ARITHMETIC OPERATORS

Arithmetic Operators		
Description	Operator	Example (a = 4 and b = 2)
Addition	+	`1 + 1`
Subtraction	-	`3 - 2`
Multiplication	*	`5 * 3`
Exponent	**	`3 ** 2 ("3 to the power of 2")`
Division	/	`10 / 2`
Modulus (returns remainder)	%	`10 % 3 (would equal 1)`

VARIABLES

- Variables store values.
- Values are assigned to variables using the = operator.
- Typically are “snake-cased” - start with a lowercase letter and use _ to separate words
- Ruby is case-sensitive, meaning capitalization counts!
- Variables starting with $ and @ have mean different things in Ruby, so best to just start with a lowercase letter

```
variable_name = value
```

STRINGS

- Created using the single/double quote character ‘ ‘ OR “ ”
- Strings can be added to each other “hello” + “ world” = “hello world”

```
‘ ’                    #  An empty string
‘Hello world’          # A non-empty string
```

STRINGS — INTERPOLATION

- Another way to build up strings is via **interpolation** using the syntax #{}
- Unless you are interpolating, all strings should be in single quotes

```
first_name = ‘Jonathan’

“#{first_name} Young”
```

=> ‘Jonathan Young’

NUMBERS - INTEGERS AND FLOATS (DECIMALS)

1 2 3 2.34 3.14159265359

BOOLEANS

true false

COMMENTS

- Start with a pound sign (#) and extend to the end of the line
- Ruby will ignore all this code
- Useful to future readers (including the author and other developers)

```
# This is a comment
```

```
> 1 + 2   # Here is some basic addition
=> 3
```

PRINTING AND GETTING USER VALUES

PRINT

Takes whatever you give it and prints it to the screen

```
print “Hello”
```

PUTS

- Similar to print, but adds a blank line after it prints your value
- Stands for “put ess” - or “put string”

```
puts “Hello”
```

GETS

Gets a value from the user (which can be saved to a variable)

```
response = gets
```

IRB + STRINGS

LOGICAL OPERATORS

Description	Operator	Example (a = 4 and b = 2)
Equal	`==`	`a == b` *false*
Not equal	`!=`	`a != b` *true*
Greater than	`>`	`a > b` *true*
Less than	`<`	`a < b` *false*
Greater than or equal to	`>=`	`a >= b` *true*
Less than or equal to	`<=`	`a <= b` *false*
Same value and data type?	`.eql?`	`1.eql?(1.0)` *false*

BOOLEANS

DECISION TIME!

- When you want to control how a computer goes through your program, use control flow!
- Control flow is either TRUE or FALSE (like booleans)
- Allows us to select different outcomes depending on user input or the result of a computation

```
if [condition]
  do something
end
```

```
if 1 > 2
  puts “greater than”
end
```

CONDITIONAL LOGIC - ELSE, ELSIF

```
if [condition]
  do something
elsif [condition]
  do something
else
  do something
end
```

```
if 1 > 2
  puts “greater than”
elsif 1 == 2
  puts “equal”
else
  puts “less than”
end
```

THE WHILE LOOP

- As long as the condition is true, the block of code will continue to execute
- When the condition becomes false, the code after the end of the loop will be executed
- Beware of infinite loops

```
while condition_is_true
  # do something
end
```

```
while number < 5
  print ‘hello’
end

number = 2
```

THE UNTIL LOOP

- Will execute a block of code until a condition is true
- Similar to while, except backwards
- When the condition becomes true, the code after the end of the loop will be executed

```
until condition_is_true
  # do something
end
```

STORING LISTS OF VALUES

- An array is a data type that holds an ordered collection of values
- Can hold any be any type of object, numbers, strings, even other arrays!
- An array can be used to store a list of values in a single variable

There are 2 different ways to create an array:

```
variable = Array.new
```

```
variable = []
```

```
shapes = [“circle”, “triangle”, “square”]
```

QUICK DETOUR

https://prezi.com/khg0wb0dqio-/ga/

KEY-VALUE PAIRS

- Collections of key-value pairs
- Have indexes, similar to arrays, but their indexes are called keys
- Indices don’t have to be numbers

```
hash = {}    #defining an empty hash
```

```
hash = {     #defining a hash with key/value pairs
  key1: value1,
  key2: value2
}
```

Example:

```
user = {
  name: “Jonathan”,
  age: 30
}
```

EACH

ITERATE OVER AN ARRAY

- Does something for each item in the array
- Takes a variable between | |, which is a placeholder for the item of the array you are currently on.

```
array = [1, 2, 3, 4, 5]
```

```
array.each do |item|
  do something
end
```

ITERATE OVER A HASH

- Similar to a iterating over arrays
- Needs two placeholder variables to represent each key/value pair

```
hash = [key1: val1, key2: val2]
```

```
hash.each do |key, value|
  do something
end
```

KEEP YOUR CODE “DRY”

- Groups program logic together so you don’t have to repeat yourself
- Can pass variables to methods

Define:

```
def method_name(arguments)
  # Code to be executed
end
```

Call:

```
method_name(arguments)
```

METHODS — SMASHABLE

MVC FOR WEB APPS

WHAT IS MVC?

An Architectural pattern that describes a way to structure our application and the responsibilities and interactions for each part in that structure.

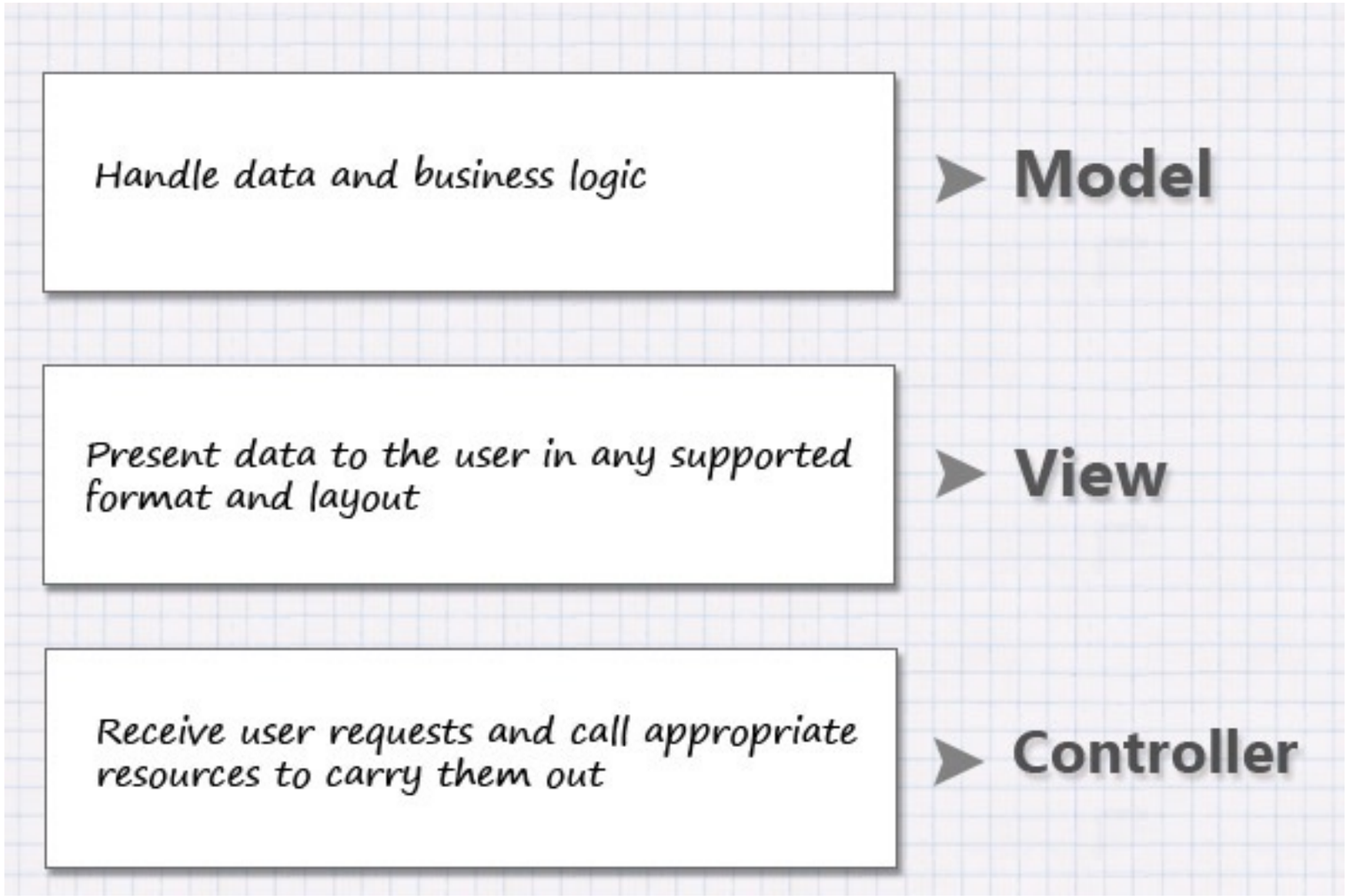
MVC – LET'S BREAK IT DOWN

M **MODEL**

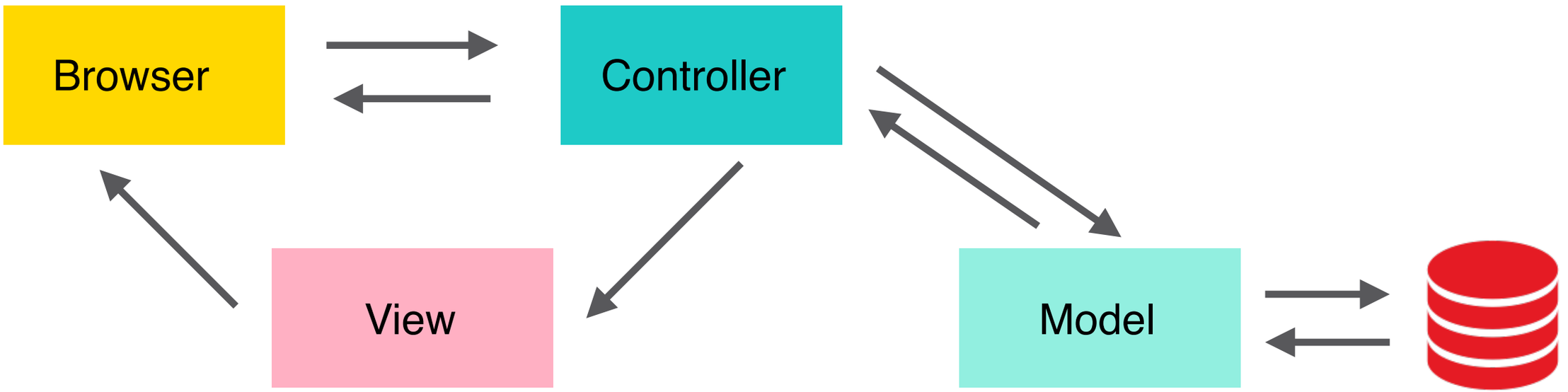
V VIEW

C CONTROLLER

MVC


Handle data and business logic
Model
Present data to the user in any supported format and layout
View
Receive user requests and call appropriate resources to carry them out
Controller

MVC



GET / POST / PUT / DELETE

- Get = Just brings up a website
- Post = Writes data to the website
- Put = Update existing data
- Delete = self-explanatory

CREATING OUR FIRST WEB APP

RAILS PRINCIPLES

DRY

- Don’t repeat yourself!!!!
- Concise, consistent code that is easy to maintain

CONVENTION OVER CONFIGURATION

- Sensible defaults.
- Makes assumptions about what developers need to start a project
- Speeds up development, there’s not as much code to maintain

LET'S CREATE A BLOG! BUT FIRST, LET'S BRAINSTORM ABOUT OUR POSTS

GENERATE A RAILS APPLICATION WITH SCAFFOLDING

- First step with any new application/project
- Scaffolding is cheating but a very efficient way to get something off the ground

```
$ rails new blog
```

```
$ rails g scaffold Post title body:text
```

RAILS SERVER

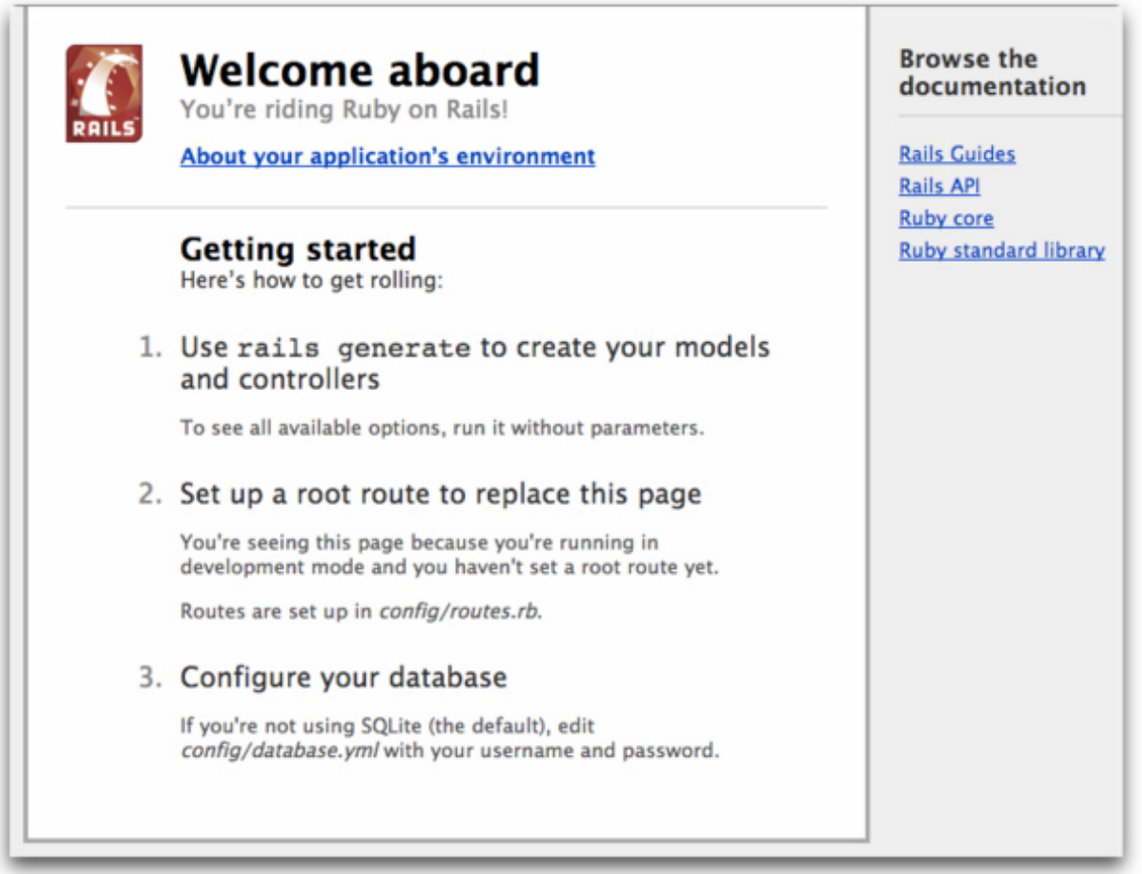
- Launches a small web server that comes with ruby.
- Use this command to access the webpage within your browser

```
$ rails server
```

Shorthand:

```
$ rails s
```

LET'S CREATE A BLOG!



MIGRATING A DATABASE

- Check your db/migrate folder and open that file in there

```ruby
class CreatePosts < ActiveRecord::Migration
  def change
    create_table :posts do |t|
      t.string :title
      t.text :body

      t.timestamps
    end
  end
end
```

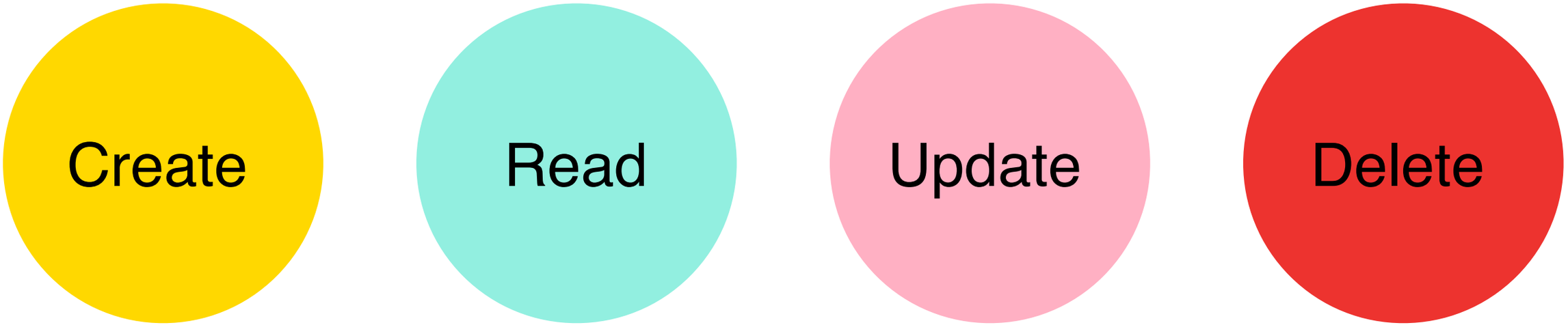
```
$ rake db:migrate
```

DATABASE MIGRATIONS

- Migrations are Ruby classes that make it simple to create and modify database tables

LET'S SEE WHAT WE'VE MADE!

- Visit localhost:3000/posts
- All of our functionality has been built! Before we dive into everything, let's talk about CRUD.



LET'S PLAY

- Let's create a new blog post together!

WHAT HAPPENS WHEN WE TRY TO CREATE A BLANK POST?

- You tell me!
- How do we validate that a user submits a title and a body?

```
class Post < ActiveRecord::Base
  attr_accessible :body, :title

  validates_presence_of :body, :title
end
```

CLASSES - MODELS

- By convention, class names start with a capital letter and use CamelCase instead of underscores
- A class has the ability to create other objects that are of its kind.

```
class MyClass
  def initialize(param1)
    @param1 = param1
  end
end
```

```
class Article
  def initialize(title, content)
    @title    = title
    @content  = content
  end
end
```

```
new_object = MyClass.new(param)
```

```
article = Article.new(“title”, “body”)
```

CLASSES - MODELS

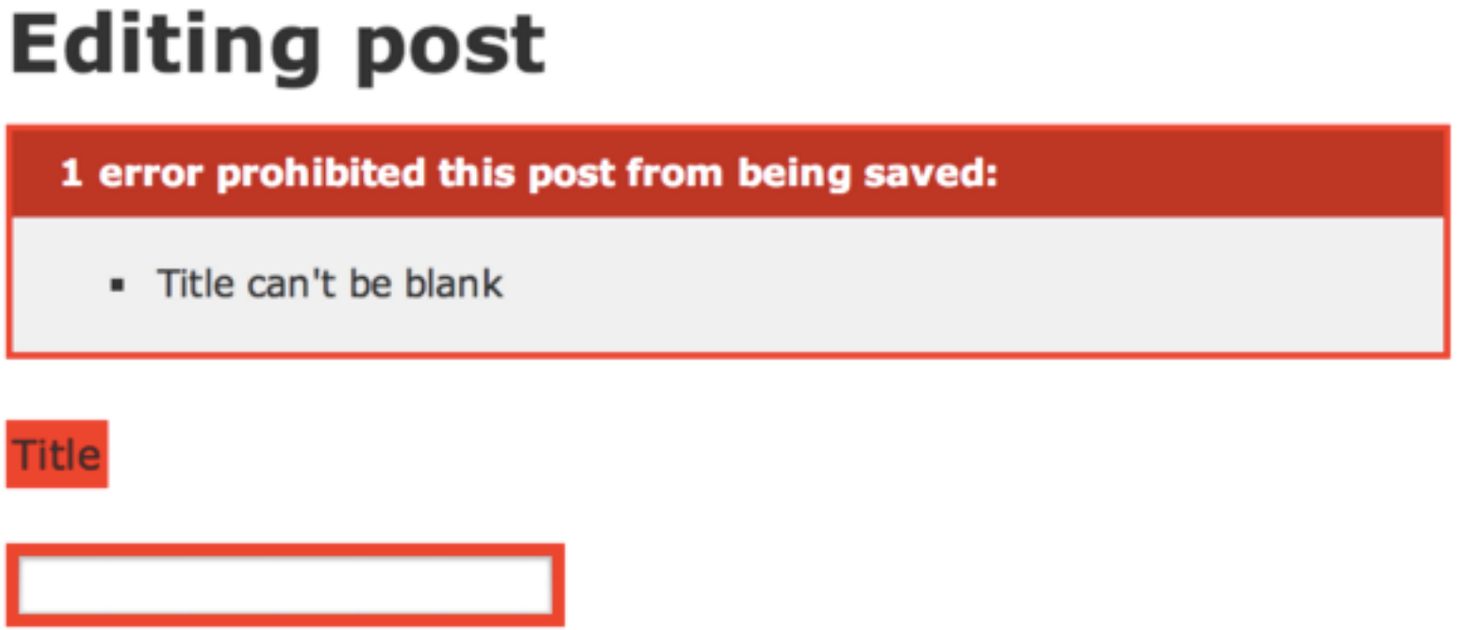
- Models contain all the data and all the methods that it knows about itself
- A class has the ability to create other objects that are of its kind.

```
class Person
  def initialize(name, age)
    @name = name
    @age = age
  end

  def breathe
    “inhale / exhale”
  end
end
```

VALIDATIONS CONTINUED

- We've created validations for Posts. Let's try to break them. Can you?



PROBLEM - OUR POST IS UGLY.

- Let's visit a post. It's pretty ugly.
- Inside of app/views/posts/show.html.erb, let's put this code instead. It should be prettier.

```
<p id="notice"><%= notice %></p>

<h2><%= link_to_unless_current @post.title, @post %></h
2>
<%= simple_format @post.body %>

<%= link_to 'Edit', edit_post_path(@post) %> |
<%= link_to 'Back', posts_path %>
```

ANOTHER PROBLEM – OUR BLOG IS UGLY.

- We need to make our blog pretty. Instead of rewriting code over and over, let's keep it DRY and use something called a *partial*
- A partial is reusable HTML code within an application.
- Using the command line, lets create a file app/views/posts/_post.html.erb
- Underscores tells Rails / you that it is a partial

```
 <h2><%= link_to_unless_current @post.title, @post %></h
2>
 <%= simple_format @post.body %>
```

OUR BLOG IS UGLY.

- Inside of show.html.erb, let's put that reusable code in place
- After you do so, make sure everything still works!

```
<p id="notice"><%= notice %></p>

<%= render :partial => @post %>

<%= link_to 'Edit', edit_post_path(@post) %> |
<%= link_to 'Back', posts_path %>
```

OUR BLOG IS UGLY.

- Finally, let's change our index page

```
<h1>Listing posts</h1>

<%= render :partial => @posts %>

<%= link_to 'New Post', new_post_path %>
```

PROBLEM – OUR BLOG IS TOO SIMPLE!

- What is a blog without comments for trolls to enter?
- Shut down your server and run this: rails generate resource Comment post:references body:text
- Migrate the database again and let’s go over the database schema

LINKING POSTS TO COMMENTS AND VICE VERSA

- A post has_many comments and a comment belongs_to a post
- In the Post class, make sure it looks like this:

```
class Post < ActiveRecord::Base
  attr_accessible :body, :title

  has_many :comments

  validates_presence_of :body, :title
end
```

PROBLEM – PEOPLE CAN CREATE COMMENTS OUTSIDE OF THE POST CONTEXT

- rake routes

```
jyoung@Jonathans-MacBook-Air:~/Desktop/blog$ rake routes
      Prefix Verb   URI Pattern                  Controller#Action
    comments GET    /comments(.:format)          comments#index
             POST   /comments(.:format)          comments#create
 new_comment GET    /comments/new(.:format)      comments#new
edit_comment GET    /comments/:id/edit(.:format) comments#edit
     comment GET    /comments/:id(.:format)      comments#show
             PATCH  /comments/:id(.:format)      comments#update
             PUT    /comments/:id(.:format)      comments#update
             DELETE /comments/:id(.:format)      comments#destroy
       posts GET    /posts(.:format)             posts#index
             POST   /posts(.:format)             posts#create
    new_post GET    /posts/new(.:format)         posts#new
   edit_post GET    /posts/:id/edit(.:format)    posts#edit
        post GET    /posts/:id(.:format)         posts#show
             PATCH  /posts/:id(.:format)         posts#update
             PUT    /posts/:id(.:format)         posts#update
             DELETE /posts/:id(.:format)         posts#destroy
```

PROBLEM - PEOPLE CAN CREATE COMMENTS OUTSIDE OF THE POST CONTEXT

- Solution: we nest comments within posts because a comment cannot exist without a post and only allow them to be created.

```
jyoung@Jonathans-MacBook-Air:~/Desktop/blog$ rake routes
       Prefix Verb   URI Pattern                              Controller#Action
post_comments POST   /posts/:post_id/comments(.:format) comments#create
        posts GET    /posts(.:format)                        posts#index
              POST   /posts(.:format)                        posts#create
     new_post GET    /posts/new(.:format)                    posts#new
    edit_post GET    /posts/:id/edit(.:format)               posts#edit
         post GET    /posts/:id(.:format)                    posts#show
              PATCH  /posts/:id(.:format)                    posts#update
              PUT    /posts/:id(.:format)                    posts#update
              DELETE /posts/:id(.:format)                    posts#destroy
```

LET'S CREATE COMMENTS

- app/controllers/comments_controller.rb

```
class CommentsController < ApplicationController
        def create
                @post = Post.find(params[:post_id])
                @comment = @post.comments.create!(param
s[:comment])
                redirect_to @post
        end
end
```

LET'S GET COMMENTS INTO THE VIEW!

- app/views/posts/show.html.erb

```
<p id="notice"><%= notice %></p>

<%= render :partial => @post %>

<%= link_to 'Edit', edit_post_path(@post) %> |
<%= link_to 'Back', posts_path %>

<h2>Comments</h2>
<div id="comments">
        <%= render :partial => @post.comments %>
</div>
```

LET'S GET COMMENTS INTO THE VIEW! (CTD)

- Create a file from the command line called app/views/comments/_comment.html.erb

```
<%= div_for comment do %>
        <p>
                <strong>
                        Posted <%= time_ago_in_words(comm
ent.created_at) %> ago
                </strong>
                <br/>
                <%= comment.body %>
        </p>
<% end %>
```

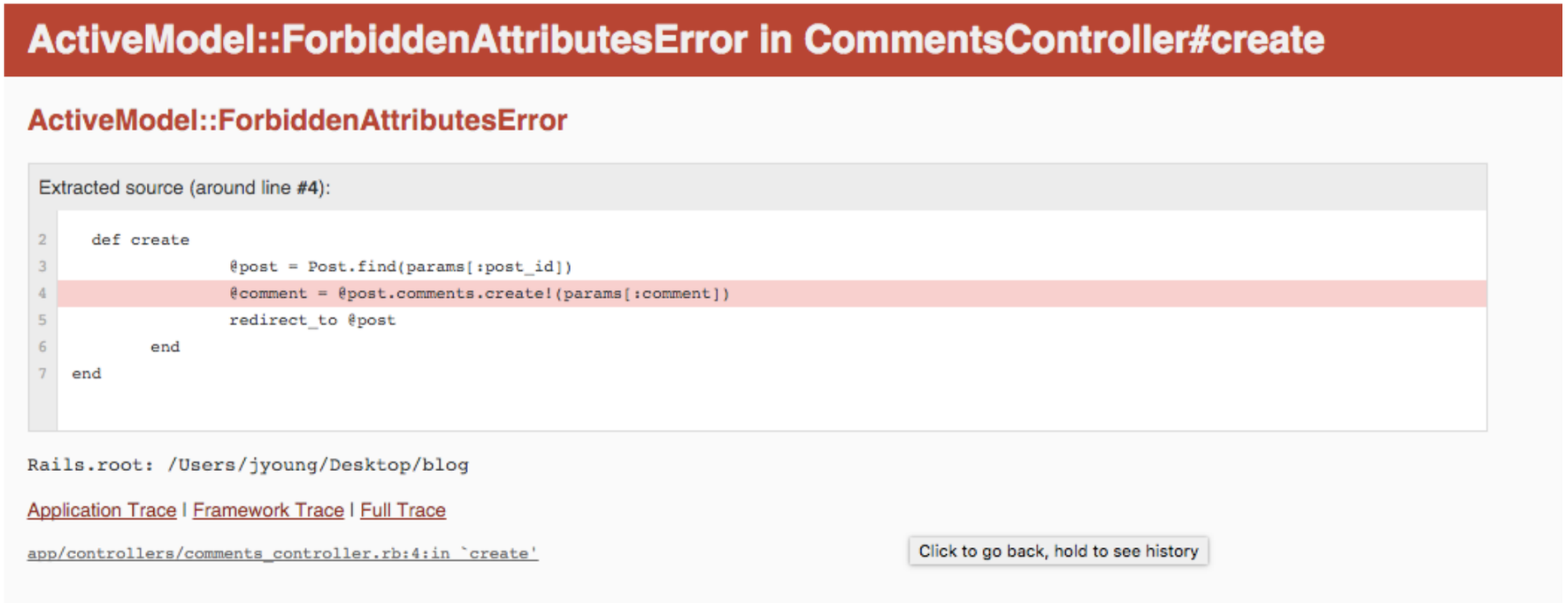
LET'S GET COMMENTS INTO THE VIEW! (CTD)

- Back in the app/views/posts/show.html.erb file, we'll need a form for creating comments

```
<%= form_for [@post, Comment.new] do |f| %>
        <p>
                <%= f.label :body, "New comment" %><br/>
                <%= f.text_area :body %>
        </p>
        <p><%= f.submit "Add comment" %></p>
<% end %>
```

GIVE IT A SHOT!

- Uh Oh!



RAILS VERSIONING

- The newest version of Rails ensures that only the attributes you say you want go into your database.
- In app/controllers/comments_controller.rb, replace the broken line with:

```
@comment = @post.comments.create!(params.require(:comment).permit!)
```

WE HAVE A WEB APP!

Congrats, we did it!

Pat yourself on the back!!!

DEPLOYING TO HEROKU

HEROKU

- You need an account: heroku.com
- The starter documentation is available through a quick Google search
- Once you get an account, you'll need the Heroku tool belt: toolbelt.heroku.com

DATABASES WITH HEROIC

- We've been using SQLite, but Heroku does not allow SQLite.
- We will instead use another open-source database called PostgreSQL

```
source 'https://rubygems.org'

gem 'rails', '3.2.13'

group :development, :test do
        gem 'sqlite3'
end

group :production do
        gem 'pg'
end

group :assets do
  gem 'sass-rails',   '~> 3.2.3'
  gem 'coffee-rails', '~> 3.2.1'

  gem 'uglifier', '>= 1.0.3'
end

gem 'jquery-rails'
```

bundle install --without=production

INITIALIZE YOUR GIT REPOSITORY, ADD AND COMMIT IT WITH A MESSAGE

- Now we are ready to deploy!

HEROKU

- heroku login
- heroku create
- git push heroku master
- heroku run rake db:setup

NEXT STEPS

GENERAL ASSEMBLY — FULL-TIME COURSES

Filter by topic | ALL FORMATS | **FULL-TIME COURSES** | PART-TIME COURSES | CLASSES & WORKSHOPS | EVENTS

Make a career change with our 8–12 week, all-day, full-time programs.

Mon, 9 March

Web Development Immersive

12-weeks. All day, every day. Learn the skills to become an entry-level web developer and the resources to get a job in this intensive program.

12 Week Course

9 Mon, 9 March
9:00 - 5:30pm

Info Session:
Thu, 11 December at 6:00pm

GENERAL ASSEMBLY — PART-TIME COURSES

Filter by topic | ALL FORMATS | FULL-TIME COURSES | **PART-TIME COURSES** | CLASSES & WORKSHOPS | EVENTS

Mon, 12 January

Front-End Web Development

In this 10-week course, students learn to code, speak the language and implement their own designs by learning HTML, CSS, and JavaScript.

10 Week Course

12 Mon, 12 January
6:00 - 9:00pm

Tue, 13 January

User Experience Design

In this 12-week course, students learn to build wireframes, implement best practices for common design patterns and analyze business goals from a user perspective.

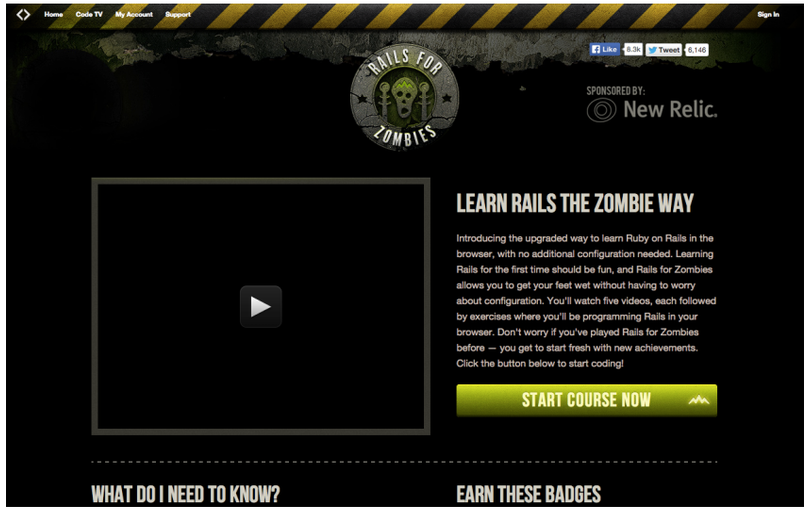
12 Week Course

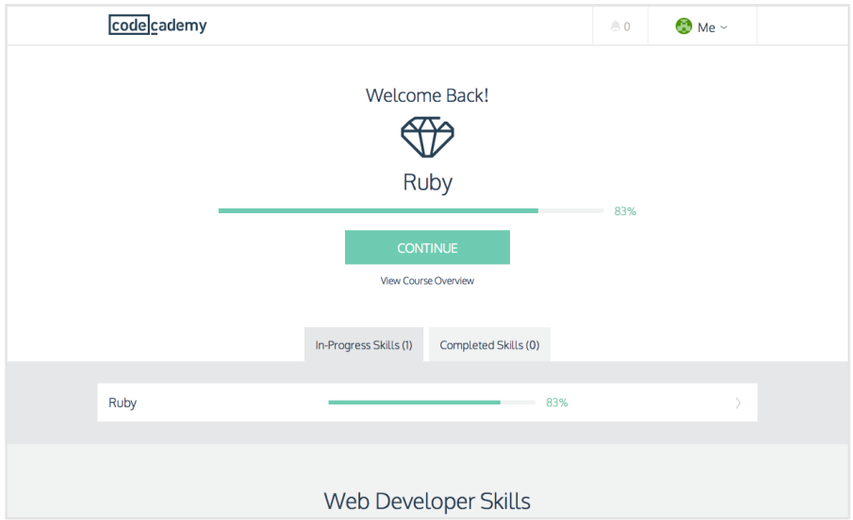
13 Tue, 13 January
6:30 - 8:30pm

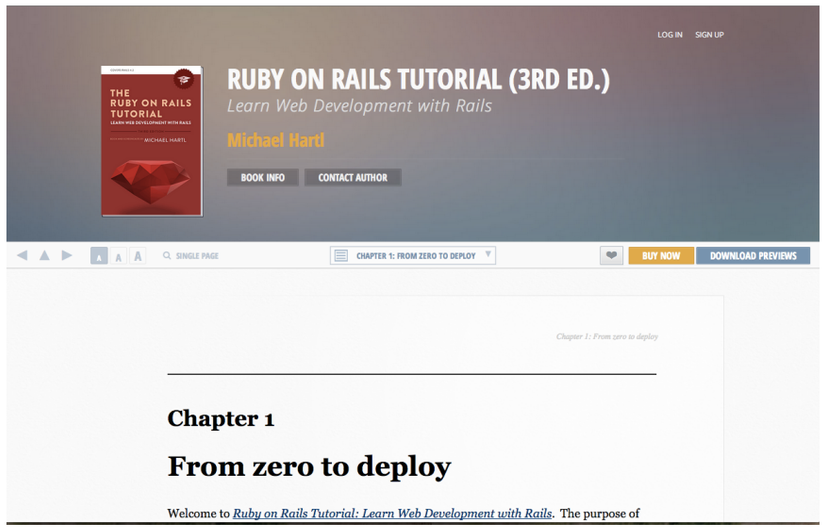
Info Session:
Mon, 24 November at 5:30pm

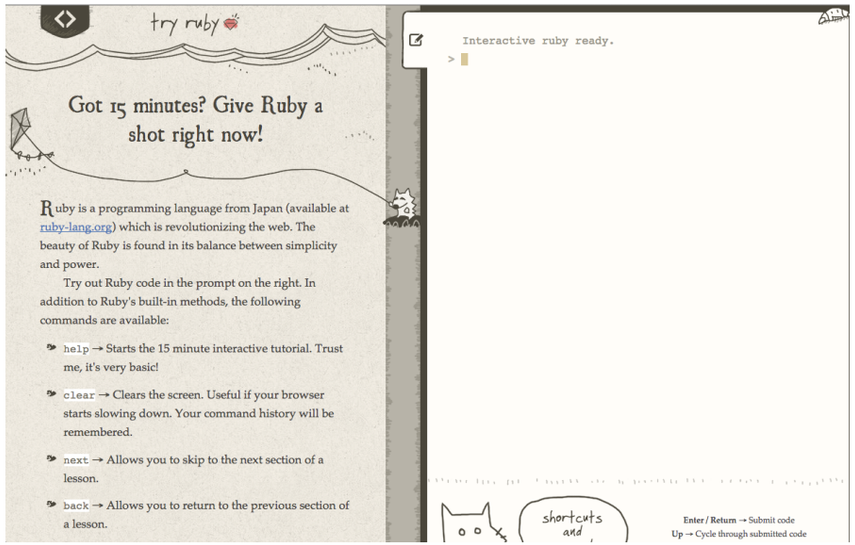
Mon, 9 March

ONLINE LEARNING









Q&A

ADDITIONAL RESOURCES

DNS / Internet —> https://howdns.works/ep1/

Rails —> https://www.railstutorials.org

DOCS —> https://ruby-doc.org

RubyMonk —> https://rubymonk.com

THANKS!

JONATHAN YOUNG

- jonyoungg@gmail.com
- @JYoung217 everywhere

A special thank-you to Sarah Holden for providing the outline of this presentation