

# PennApps Ruby Workshop

Justin Kim



# What does this do?

```
3.times do  
  print 'Hello, world!'  
end
```

# Why Ruby?

Optimized for programmer happiness

Used for [Ruby on Rails](#), an extremely popular web framework



# Outline

1. Intro to Ruby
2. Intro to the Web and MVC
3. Ruby on Rails

# Installing Ruby

- Use [Ruby Version Manager \(RVM\)](#) to manage and install Ruby versions
  - Use this even if you plan on just using one version
- We will be using version [2.3.1](#)
- When you have trouble remembering what methods do, use [Ruby Docs](#)

# Printing in Ruby

- You can print a value with three different commands `print`, `puts`, and `p`
  - `print` outputs the value and returns `nil`
  - `puts` outputs the value with a newline and returns `nil`
  - `p` both outputs and returns the value
    - I'll be using this in my examples since it also formats output better
- I will denote output with `#=>`

```
p 'hello world' #=> "hello world"
```

# Running Ruby

- Use a REPL (Read-Execute-Print-Loop) with the `irb` command in terminal
- Execute `.rb` files with the `ruby` command:  
`ruby file.rb`

# Methods

- Parentheses around arguments can be omitted if unambiguous
- Methods have implicit returns

```
def hi
  'hello, there'
end

def hello(name)
  puts "hello, #{name}"
end

puts hi #=> "hello, there"
hello('Matz') #=> "hello, Matz"
hello 'DHH' #=> "hello, DHH"
```



# Creating a Class

- Use the `class` and `end` keywords
- A class can be instantiated with the `new` method
- It is convention to write class names in PascalCase

```
class Student
end
student = Student.new
p student #=> #<Student:0x007ff7989d44b8>
```

# Instance Methods

Methods defined in a class are instance methods by default

```
class Student
  def greet
    puts 'hi'
  end
end
student = Student.new
student.greet #=> "hi"
```

# Constructors

If a method is named `initialize`, then it will be executed when the class is instantiated

```
class Student
  def initialize
    puts 'hi'
  end
end
Student.new #=> "hi"
```

# Inheritance

- Classes can inherit from another class with the `<` operator
  - Simply place it after the class declaration and name the class
- Thus, a class can gain all of its parent class's methods, both public and private

```
class Bird
end
class Penguin < Bird
end
p Penguin.superclass #=> "Bird"
```

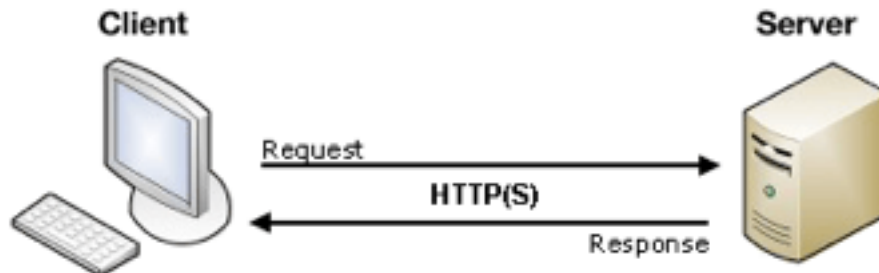
# Installing Gems

- Ruby libraries are called gems
- The command to install them is `gem install gem_name`
  - When installed, the gem is installed in the current Ruby version's gem directory
- To use a gem, pass the name of the gem as a string to the `require` method at the top of the file (e.g. `require 'pry'`)

# The Web

# HTTP

- Stands for Hypertext Transfer Protocol
- A **client** (e.g. web browser, phone, computer, etc.) sends a **request** to a **server**
- The **server** receives this **request** and sends back a **response**
- This **response** is usually a web page (i.e. HTML with accompanying files) or data, usually in XML or JSON



# HTTP Verbs

- The five most common types of HTTP requests are:
  - GET
  - POST
  - PUT/PATCH
  - DELETE



# GET Request

- This is usually the default type of request sent
  - When you enter a URL or click a link, a GET request is sent for the web page
  - When a web page updates, it probably sent a GET request behind the scenes to get the new data
- It should only be used to *get* something

# POST Request

- This should be used to *send* data from the client to the server
- While you can technically use GET requests to send data as well, you should absolutely use POST requests if you're sending data
  - It's much more robust and secure
- This is the default type of request sent when submitting a form (e.g. log in)

# PUT/PATCH Request

- This should be used to *update* something on the server
- Technically, you can use a POST request to update as well, but it is convention to use a PUT or PATCH request
- The main difference between a PUT request and a PATCH request:
  - A PUT request is used to update an entire record
  - A PATCH request is only used to update part of it

# DELETE Request

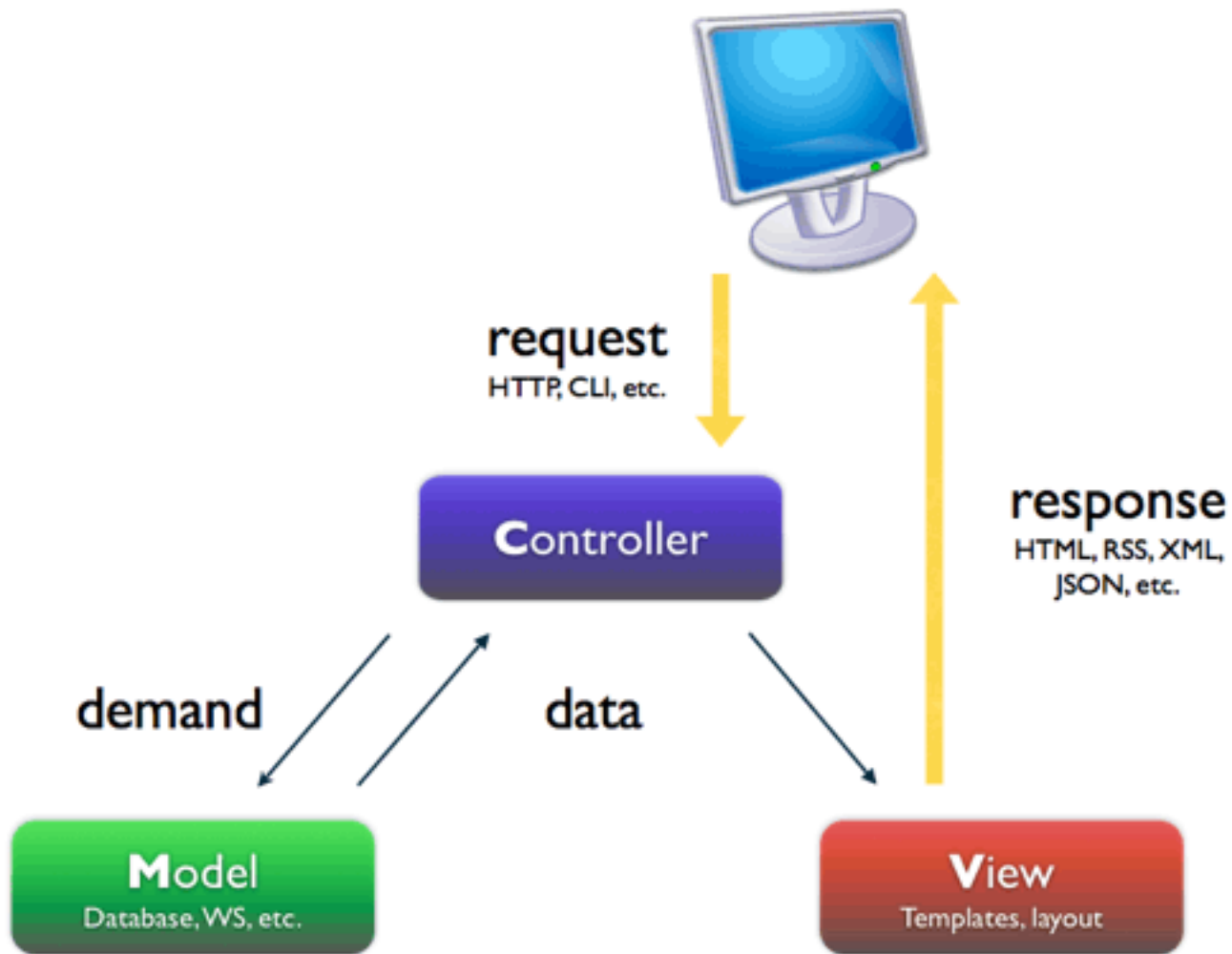
- This should be used to *delete* something on the server
- Technically, you can use a POST request to delete as well, but it is convention to use a DELETE request

# MVC

- Stands for Model-View-Controller
- Every community has different definitions and conventions for MVC
  - Ignore the conventions of other communities when writing Ruby
- Convention over configuration

# View

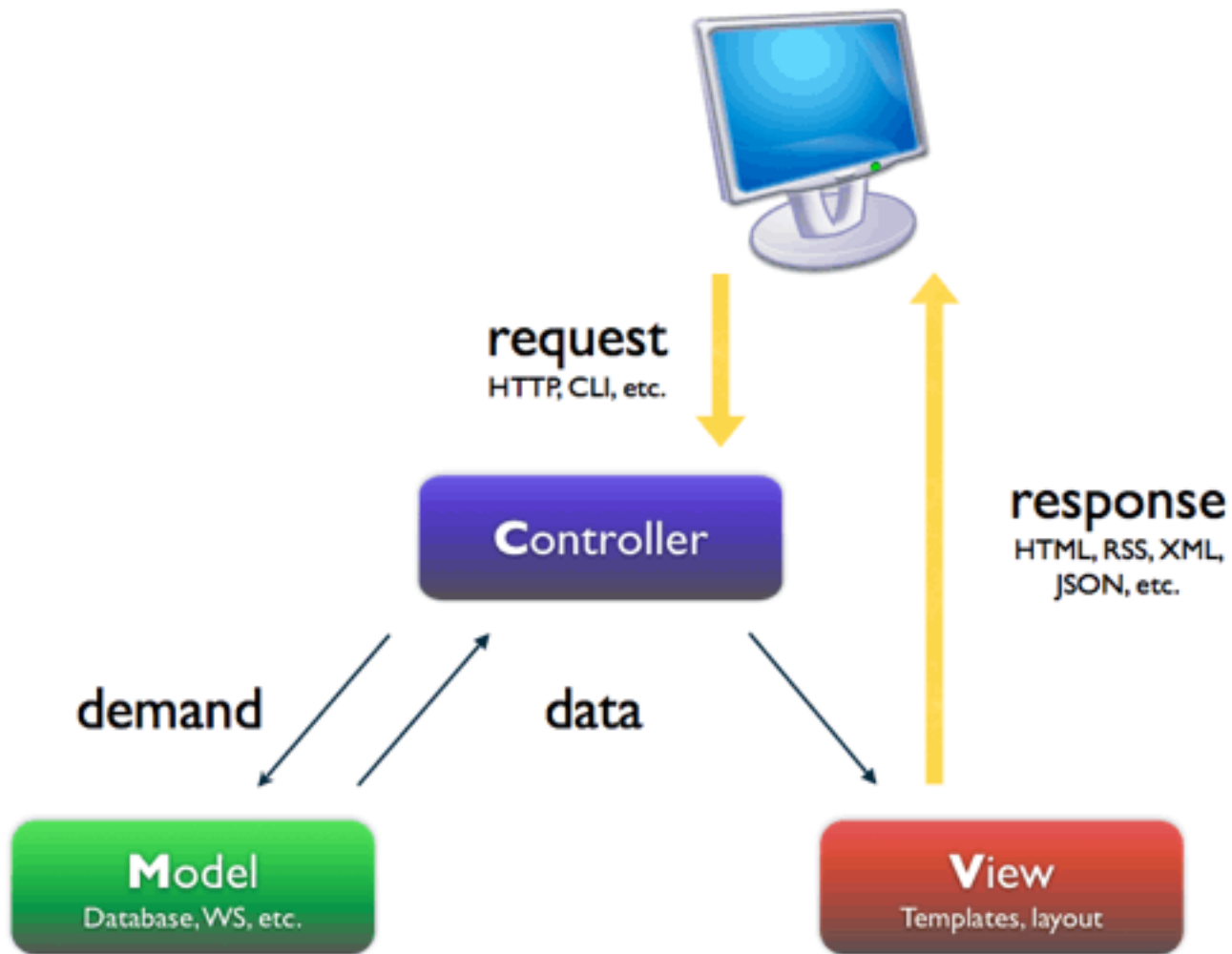
- The view is the layer of the application that the user will see
  - It is typically comprised of .html.erb files
  - It should have minimal logic in it
  - It can access instance variables defined in the controller
- The corresponding view files should be in a subdirectory of `views` named for the plural form of the corresponding model



# Controller

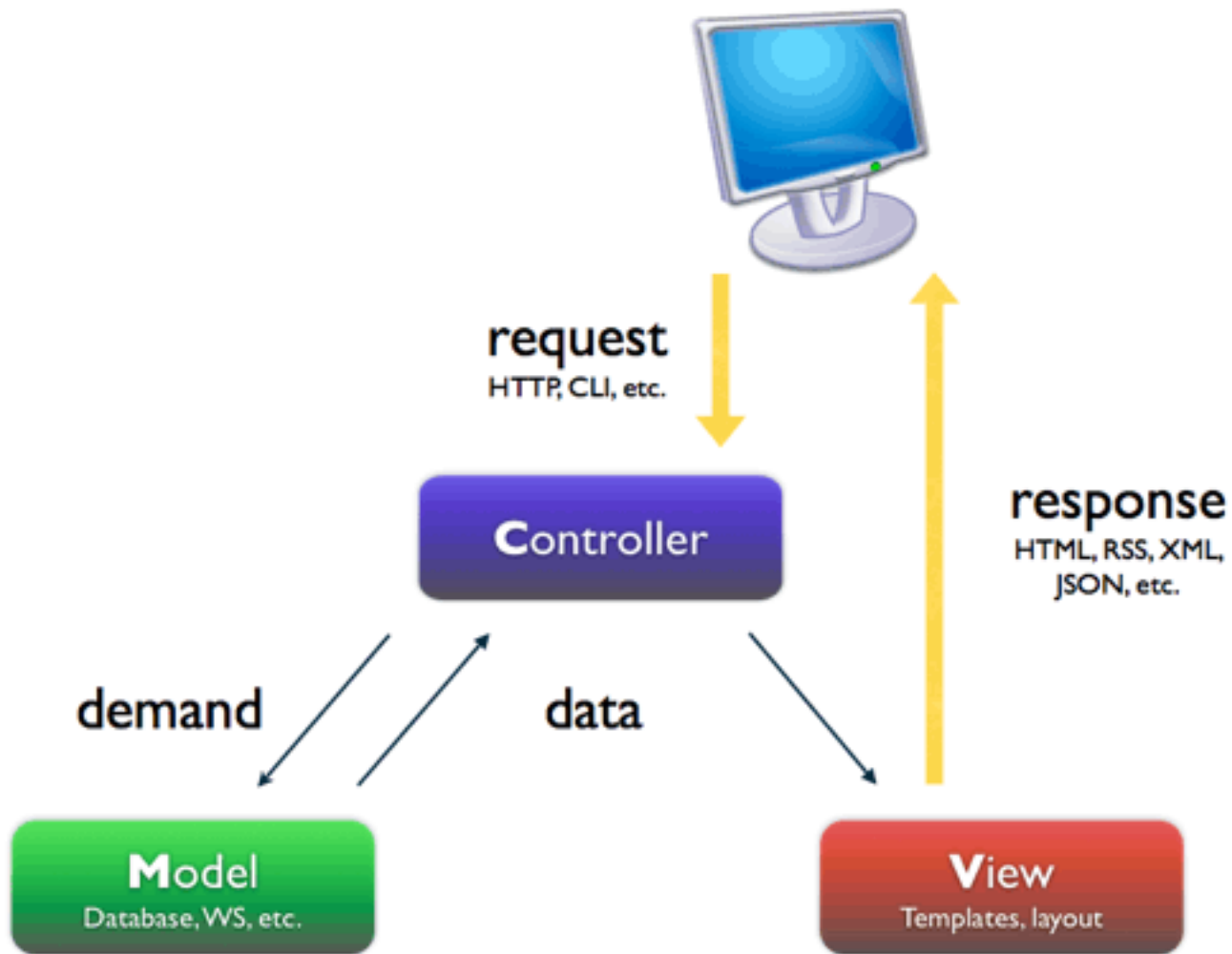
- The controller is the layer of the application that handles HTTP requests
  - It should pass off as much logic to the model as possible
  - It can define instance variables for the view to use
- This layer of the application should be the most static out of the three
- The naming convention is the plural form of the corresponding model with `Controller` (e.g. `UserController`)





# Model

- The model is the layer of the application with the crux of the logic
- It should be the main place where the database is accessed
- The naming convention is the singular form of the name



# REST

- Stands for Representational State Transfer
  - A set of conventions to expose certain HTTP endpoints
- It is convenient for Create, Read, Update, Delete (CRUD) apps
- The below example is for a model representing movies

Path	Verb	Usage
/movies	GET	Show a list of movies
/movies/new	GET	Show form to create a new movie
/movies	POST	Create a new movie
/movies/:id	GET	Show a specific movie
/movies/:id/edit	GET	Show a form to update a movie
/movies/:id	PUT/PATCH	Update a movie
/movies/:id	DELETE	Delete a movie

# Ruby on Rails

# About Ruby on Rails

- Also called RoR or most commonly Rails
- It is a web framework, similar to Sinatra, but it has far more features and opinions
- We will use version 4.2.4
  - Note, Rails 5 recently came out

# Making a Rails App

- After you `gem install rails`, run `rails new app_name`
- Rails will then create a directory of your Rails app with all the basic directories and files
- It will also `bundle install` all the default gems
- Check out the [Rails documentation](#)

# Rails Commands

- `rails server` (or `rails s`) will start the Rails app
  - By default, the server uses WEBrick
  - To use other servers, put the server's gem in the Gemfile (Heroku recommends using puma)
- `rails console` (or `rails c`) will start the Rails console



# Rails Generate

- `rails generate` (or `rails g`) will generate various files for you
- There are several different kinds of generators
- The most useful ones (for me at least) are `migration`, `model`, and `controller`
- `rails g scaffold` helped make Rails famous
  - Generates tests, controllers, views, routes, models, and migrations
  - It's too vanilla for actual use though

# Rails Generate Syntax

- `rails g generator_name model_name`
- For `migration`, `model`, and `scaffold`, you can also specify attributes (the column titles)
  - `rails g generator_name model_name  
column1:type column2:type`
- For example, `rails g migration Item  
name:string price:float`
- For foreign keys, use the `references` type

# Controller

- Each route is defined as methods according to its corresponding RESTful route
- By default, it will render the corresponding view file
  - You can render whatever view with the `render` method
  - `render :show` will render the model's `show.html.erb` page

# Routes

- The routes are managed in the `config/routes.rb` file
- `resources :pluralized_model_name` generates all seven RESTful routes
- To define custom routes, use `http_verb`  
`'route' => 'model_name#method'`
  - `get 'users/hello' => 'user#hello'` would connect the `hello` method in `UserController` to a GET request to `users/hello`

# Rails Directory

- Notice that most of this directory structure should be familiar to you
- The main directories not introduced to you yet are: `lib`, `log`, `test`, `tmp`, and `vendor`
- `test` contains test files, `log` contains error logs, and `tmp` contains temporary files
- `app/helpers` contains modules

# Asset Pipeline

- Introduced in [Rails 2011](#)
- It is a way to load resources (i.e. images, javascripts, and stylesheets)
- It is comprised of the `app/assets`, `lib`, and `vendor` directories

# What Should Go Where

- The custom code you write specific to your application should go in `app/assets`
- The custom code you write not specific to your application should go in `lib`
  - It's pretty rare to use this
- 3rd party libraries should go in `vendor`

# Sprockets

- Sprockets is an asset packaging system
  - The stylesheet one can be found at `app/assets/stylesheets/application.css`
  - The javascript one can be found at `app/assets/javascripts/application.js`
- They are both loaded in the head tag
  - Notice how javascript files are loaded with `<%= javascript_include_tag %>`
  - stylesheet files are loaded with `<%= stylesheet_link_tag %>`



# Sprocket Application Files

- To require a file, append an = to the beginning of the commenting delimiter (e.g. `// =` for javascript)
- `require_tree` . requires all files in the directory
  - I try not to use this because it loads the files in alphabetical order
  - the order usually matters for me
- Never write any javascript/css code directly in the application files
  - `require_self` will let you do this