Ruby on Rails

Jackie Askins &
Sanjana Sarkar

Install Ruby & Rails

- **Windows**
  - Install the CIS 196 Virtual Machine (http://bit.ly/cis196-vm)
  - The VM will have Ruby, Rails, and a text editor pre-installed
  - Install node on the VM with `sudo apt-get install nodejs`
- **Mac/Linux**
  - You can use the VM or install everything locally
  - Install the Ruby Version Manager (RVM) (https://rvm.io/)
  - Install Ruby with `rvm install 2.4.1`
  - Install Rails with `gem install rails -v 5.1.4`
  - Install a text editor, like Sublime Text (https://www.sublimetext.com/3)

Ruby

About Ruby

- Dynamically Typed
- Strongly Typed
- Interpreted
- Basically everything in Ruby is an object

Writing & Running Ruby Code

- **Use a REPL (Read-Execute-Print-Loop):**
  - In the Command Line:
    - Run `irb` & execute lines of Ruby code
    - Exit with `quit`
  - `Great for testing!`
- **Running Ruby programs:**
  - Write Ruby code in files ending in .rb
  - In the Command Line:
    - Change into the directory containing the file
    - Run: `ruby file_name.rb`

Printing

- You can output data in 3 different ways:
  - `print` outputs the value and returns `nil`
  - `puts` outputs the value with a new line and returns `nil`
  - `p` outputs and returns the value
- We will use `p` in most of our examples with `#=>` to denote the output

```
p 'Hello world!' #=> "Hello world!"
```

Numerics

- The Numeric class represents numbers of different types in Ruby
- Numbers are also objects (so we can call methods on them)

```ruby
1 # This is an Integer
15.7 # This is a Float

# We can call methods on Numerics
p 4.even? #=> true
```

Strings

- Strings can be written in single quotes or double quotes
- For most purposes, you'll want to use single quotes

```
p 'This is a string!' #=> "This is a string!"
p "This is also a string!" #=> "This is also a string!"

p "You have to use double quotes to include escape characters: \n"
p "You have to use double quotes to interpolate code: #{5 + 5}"
#=> "You have to use double quotes to interpolate code: 10"

'You should use single quotes otherwise'
```

Booleans & Nil

- Booleans are represented with `true` and `false`
- `nil` is like Java's null
  - It represents nothingness
  - It's also an object!

Variables

- Ruby is dynamically typed
- We don’t need to specify the type of the variable
- We can assign & re-assign variables to objects of different types
- It is convention to make variable names snake_case

```
foo = 'hello' # No need to specify the type
p foo #=> "hello"
foo = 15 # No error!
p foo #=> 15
```

Arrays

- Arrays can have mixed types

```
my_arr = [1, 'two', 3.3, true, nil]
p my_arr.first #=> 1
p my_arr[3] #=> true
```

Hashes

- Hashes map keys to values (like maps in Java)

```
my_hash = { 1 => 'one', 'two' => false }
p my_hash[1] #=> "one"

new_hash = { one: '1', two: 2 }
p new_hash[:two] #=> 2
```

Iterators

- For loops exist in Ruby, but iterators are preferable

```ruby
[0, 1, 2].each do |num|
  print num
end #=> 012

[3, 4, 5].each { |num| print num } #=> 345
```

Flow Control

```ruby
if num < 0
  p "#{num} is negative"
elsif num == 0
  p "#{num} is equal to 0"
else
  p "#{num} is positive"
end

p 'Three is odd' if 3.odd? #=> "Three is odd"
p 'Three is not even' unless 3.even? #=> "Three is not even"
```

Methods

```ruby
def hello_world
  'Hello World'
end
p hello_world #=> "Hello World"

def hello(name)
  "Hello #{name}"
end
p hello('Sanjana') #=> "Hello Sanjana"

def goodnight(name = 'Moon')
  "Goodnight #{name}"
end
p goodnight('Jackie') #=> "Goodnight Jackie"
p goodnight #=> "Goodnight Moon"
```

Classes

- Class names should be in PascalCase
- Instantiate a new instance with the `new` method

```
class MyClass
end
my_instance = MyClass.new
p my_instance.class #=> MyClass
```

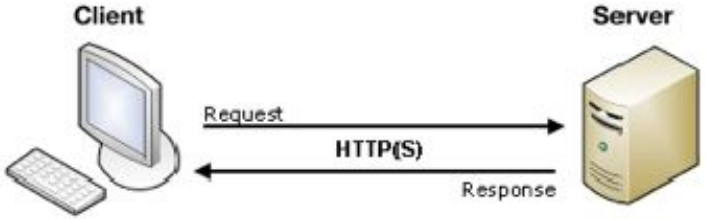
Managing Dependencies

- Ruby libraries are called gems
- The command to install them is `gem install gem_name`
- Ruby programs use a Gemfile to manage its list of gems
- You can install all gems in a Gemfile by running:
  - `gem install bundler` (Only the first time)
  - `bundle install`

HTTP and MVC

HTTP

- Stands for **H**yper**T**ext **T**ransfer **P**rotocol
- A **client** sends a **request** to a **server**
- The **server** receives the **request** and sends back a **response**
- The **response** is generally in the form of a webpage (i.e. HTML) or data (i.e. XML or JSON)



HTTP Verbs

- **GET**
  - Default type of request
  - Should only be used to GET data
- **POST**
  - Used to send data from client to server
  - More secure that a GET request
- **PUT/PATCH**
  - Used to update something on the server
- **DELETE**
  - Used to delete something on the server

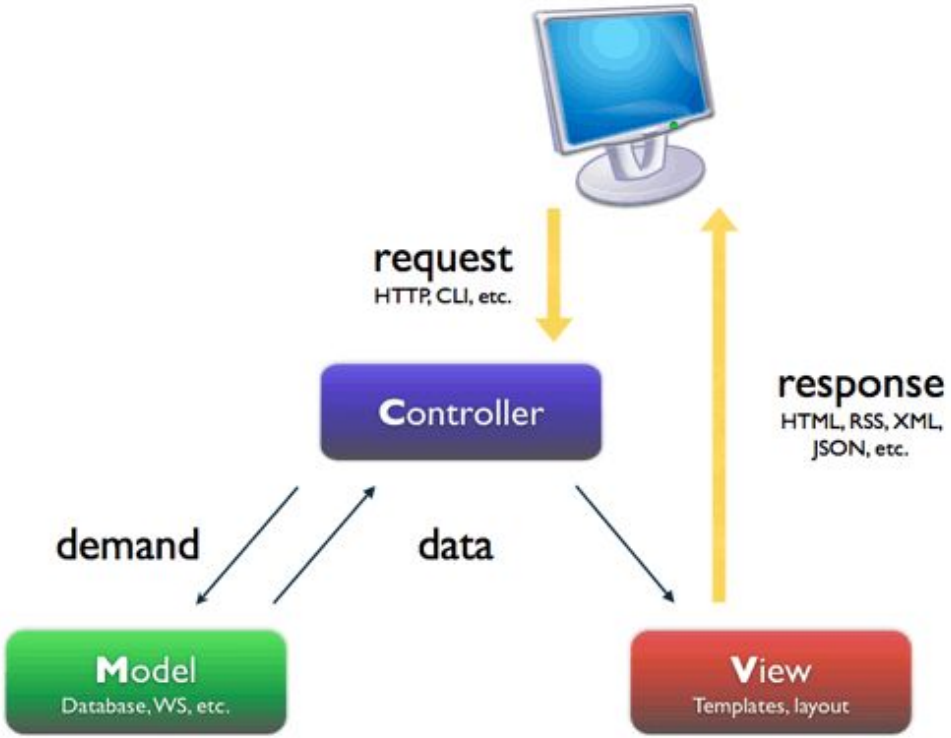
MVC (Model, View, Controller)

- It’s an architectural pattern (a way for code to be organized)
- Rails code is organized with MVC

MVC Layers

- Model
  - Main place where the database is accessed
  - Most of the application's logic goes here
- View
  - This is what the user sees & interacts with
- Controller
  - Depending on what route (think URL) you're on, the controller may:
    - Gets info from the model - why?
    - Renders an HTML view with the info to be shown to the user

MVC Layers


request
HTTP, CLI, etc.
Controller
response
HTML, RSS, XML,
JSON, etc.
demand
data
Model
Database, WS, etc.
View
Templates, layout

Ruby on Rails

Ruby on Rails

- Also referred to as Rails
- It is a very opinionated web framework
  - Allows you to get a website up and running very quickly
- Revolutionized web development from 2005 to 2015

Creating a Rails app

- Run `rails new app_name`
- This creates a directory with the name of your app with the directories & files common to a Rails app
- This also installs all of the gems in the Gemfile

Important Components of Rails Apps

- Built-in SQL database to contain data (SQLite by default)
- `app/` directory contains:
  - `models/`, `views/` & `controllers/` subdirectories
- `config/routes.rb` is used to manage our app's routes (think URL)

Rails Commands

- `rails server` (or `rails s` for short)
  - This starts the server on port 3000 by default
  - You can visit the app by going to http://localhost:3000 in a web browser
- `rails console` (or `rails c` for short)
  - This starts an interactive console
  - Useful for working with models

Controllers in Rails

- Controllers get placed inside of `app/controllers/`
- The naming convention is `name_controller.rb`
  - Class name: `NameController`
- Controllers should be modular
  - Should have a `UsersController`, `PostsController`, `PagesController`, etc.
- Each request corresponds to a controller method (known as an action)

Views in Rails

- If an action makes a GET request, we need to show the user a view
- Our views will use HTML with Ruby code embedded in it
- They are associated with a controller & an action so they will be placed inside of `app/views/controller_name/action_name.html.erb`

Rails Generators

- Generators are used to quickly create code & files by running a simple command
- Here are some of the most useful generators:
  - `scaffold` (arguably made Rails famous)
  - `controller`
  - `model`

Generating Controllers & Views

- We can create a controller and a corresponding view with the `controller` generator
- If we want to create our app's homepage:
  - `rails g controller pages home`

Routes

- We need a way to specify what action should be used when a user visits a specific URL
- We do this inside of config/routes.rb
- For each route, we need to specify the path, request type, controller, and action
  - i.e. `get '/', to: 'pages#home'`
  - Setting up the root page: `root 'welcome#home'`

Models in Rails

- Inside of Rails, our model classes correspond to database tables
- Models get placed inside of `app/models/`

Scaffold Generator

- rails g scaffold helped make Rails famous
- Generates controllers, views, routes, models
  - `rails g scaffold model_name col1:type col2:type`

Associations

- Associations allow us to create relationships between models
- Common ones are `has_many` and `belongs_to`

```
class Teacher < ApplicationRecord
  has_many :students
end

class Student < ApplicationRecord
  belong_to :Teacher
end
```

Interacting with the Database

- To create the database:
  - `rails db:create`
- To apply pending changes (migrations):
  - `rails db:migrate`
- If you end up needing to drop your database (NEVER in production):
  - `rails db:drop`

REST

- Stands for **RE**presentational **S**tate **T**ransfer
- A set of conventions to expose certain HTTP endpoints
- Convenient for CRUD (Create-Read-Update-Delete) apps

Reference

The demo code and slides will be at this link:

tiny.cc/sp18railscode