# Overview

* Different from most programming languages because it takes advantage of many conventions to reduce development time
* Two of Rails' guiding principles
  + DRY - "Don't Repeat Yourself" - suggests that writing the same code repeatedly is a bad thing.
  + Convention Over Configuration - means that Rails makes assumptions about what you want to do and how you're going to do it, rather than requiring you to specify every little thing through endless configuration files.
* A Framework is a universal, reusable software platform to develop applications, products and solutions. Software frameworks include support programs, compilers, code libraries, tool sets, and application programming interfaces (APIs) that bring together all the different components to enable development of a project or solution.
* One characterizing part of Rails is its approach to connecting object-oriented software with a relational database
  + Maps tables to classes, rows to objects, and columns to fields of the objects
  + Object relational mapping – ORM
* Model view controller
  + Model code
    - Manages how data is written and read to your data base
    - Model code objects represent things that exist in the system's problem domain
  + View code
    - Part of the application that is presented to the user
    - It's also called the presentation layer. In a web application, the view mostly generates web pages
  + Controller code
    - Brain of the application
    - Determines how user interacts with the system
    - Controls which data is accessed from the model
    - Determines which parts of the view will present it

# Creating a Rails Application

* To see the available commands, type rails (with no arguments)
* To create a new app, type rails new app\_name
* This creates a new folder with the following directories
  + app/ Contains the controllers, models, views, helpers, mailers and assets for your application. You'll focus on this folder for the remainder of this guide.
  + bin/ Contains the rails script that starts your app and can contain other scripts you use to deploy or run your application.
  + config/ Configure your application's runtime rules, routes, database, and more. This is covered in more detail in Configuring Rails Applications
  + config.ru/ Rack configuration for Rack based servers used to start the application.
  + db/ Contains your current database schema, as well as the database migrations.
  + Gemfile and Gemfile.lock These files allow you to specify what gem dependencies are needed for your Rails application. These files are used by the Bundler gem. For more information about Bundler, see the Bundler website
  + lib/ Extended modules for your application.
  + log/ Application log files.
  + public/ The only folder seen to the world as-is. Contains the static files and compiled assets.
  + Rakefile This file locates and loads tasks that can be run from the command line. The task definitions are defined throughout the components of Rails. Rather than changing Rakefile, you should add your own tasks by adding files to the lib/tasks directory of your application.
  + README.rdoc This is a brief instruction manual for your application. You should edit this file to tell others what your application does, how to set it up, and so on.
  + test/ Unit tests, fixtures, and other test apparatus. These are covered in Testing Rails Applications
  + tmp/ Temporary files (like cache, pid and session files)
  + vendor/ A place for all third-party code. In a typical Rails application, this includes Ruby Gems and the Rails source code (if you optionally install it into your project).
* Create a new rails appRails (-T omits the test for Ruby's test framework)  
   newmyrottenpotatoes –T
  + Review the directory structure

# Rubygems

* System for managing Ruby libraries (gems)
* Edit the gem file to specify that we will use HAML templating rather than ERB
  + Add this on line 2 and 3 of Gemfile:   
    #use Haml for templates  
    gem 'haml'
* Run bundle install –without production to check if any gems are missing
* Start the app  
  rails server –p $PORT –b $IP
  + Should show Welcome to Rails
  + Adding movies to the URL will result in a routing error because we haven't set the routes
* Check log/development.log for errors

# Routes

* Run rake routes to see that we haven't specified any routes  
  bundle exec rake routes
* Can use a Rails shortcut to create RESTful routs for the four basic CRUD actions
* Replace the contents of config/routes with   
  Myrottenpotatoes:: Application.routes.draw do  
   resources :movies  
   root :to => redirect(' /movies')  
  end
* Save the routes file and run rake routes again
* Got to the moviews URL
  + New error – missing controller

# Databases and Migrations

* Persistence layer
* Rails uses SQLite by default
* Use migrations – a portable script for changing the database schema
* Steps
  + Create a migration describing changes to make
  + Apply the migration to the database using rake
* Command:  
  rails generate migration create\_movies
* Creates a new file under db/migrate
  + Edit this file

class CreateMovies < ActiveRecord::Migration

def change

create\_table :movies do |t|

t.string 'title'

t.string 'rating'

t.text 'description'

t.datetime 'release\_date'

# Add fields that let Rails automatically keep track

# of when movies are added or modified:

t.timestamps

end

end

end

* Save the file and apply the migration to create the table  
  rake db:migrate
* Seed the database by copying this code into db/seeds and running rake db:seed to execute it  
  <https://gist.github.com/056aae02801cf42a0199>

# Model

* Rails uses the Active Record design pattern to "connect" models to the database
* Create the movie model by creating a file app/models/movie.rb  
  class Movie < ActiveRecord::Base  
  end
* Active Record uses convention over configuration to infer database table names from the names of the model classes and to infer the names and types of columns(attributes) associated with a given kind of model.
* Active record focuses on the CRUD actions
* The name of the model should have an uppercase letter and be singular
* The related database table will be lowercase and plural

# Controllers and Views

* A controller's purpose is to receive specific requests for the application.
  + Routing decides which controller receives which requests. Often, there is more than one route to each controller, and different routes can be served by different actions. Each action's purpose is to collect information to provide it to a view
  + The controller code is in MoviesController class which is defined in app/controllers/movies\_controller.rb
  + All methods are named in lower case
* A view's purpose is to display this information in a human readable format.
  + An important distinction to make is that it is the controller, not the view, where information is collected. The view should just display that information
  + The related views are found in app/views/movies
* To implement the index action we will define the index method in the controller and an an index.html.haml view template
  + App/controllers/movies\_controller.rb

class MoviesController < ApplicationController

def index

@movies = Movie.all

end

* + App/views/movies/index.html.haml

-# This file is app/views/movies/index.html.haml

%h1 All Movies

%table#movies

%thead

%tr

%th Movie Title

%th Rating

%th Release Date

%th More Info

%tbody

- @movies.each do |movie|

%tr

%td= movie.title

%td= movie.rating

%td= movie.release\_date

%td= link\_to "More about #{movie.title}", movie\_path(movie)

* Going to url/movies should show a list of movies
* To format views and add code to every page create the following file app/views/layouts/application.html.haml and add the following code

* + <https://gist.github.com/8478137df254b44cead4>
* Add CSS styling: <https://gist.github.com/3822f74516236abc0cfa>
* Add show action
  + Show action in controller: <https://gist.github.com/ac6dc92fdddc42d96689>
  + Show view: <https://gist.github.com/216ab745dc38f6a68b3d>

# Forms – New and Create

* Need a way to get to the add view
  + Add a link to the end of the index file

* + <https://gist.github.com/3c860b07425a97e5f089>
* Add a new method to the movies\_controller to set up a method to create a new blank Movie instance for the view to reference
  + <https://gist.github.com/746181f6684fe6ed0518151b530b55bd>
* Use fill in form tag helps to create the new form – details in video
  + New.html.haml
  + <https://gist.github.com/acb276aefb62630a0b28>

# Create, Redirection and Flash

* When creating a new item, we use the principle of least privileges. To do this we need to provide a list of which elements are permitted to be assigned to model attributes
* Define movie\_params at top of controller

def movie\_params

params.require(:movie).permit(:title, :rating, :description, :release\_date)

end

* After an item is created we want to go back to the index. To do this we use the rails redirect\_to action
* To display a confirmation message we used the special method flash[:notice]
  + flash[:warning] is used when things go wrong
  + We will modify application.html.haml so that the notice or warning is displayed on any view
  + Modify application.html.haml

* + - <https://gist.github.com/eb5c11f870f407697a12>
* Create action
  + <https://gist.github.com/832e80e008d827f12aba>

# Update and Destroy

* Update requires two actions (like create)
  + One to display the form with editable information (edit)
  + One to apply the updated information(update)
* First need a way to specify the action
  + Add code to the show view: <https://gist.github.com/eb5c11f870f407697a12>
* Create the edit view
  + edit.html.haml: <https://gist.github.com/df5ab3a1654e60f85c44>
* Add the edit and update methods to the controller
  + <https://gist.github.com/60a85a64799ea238c045>
* Delete is accomplished by calling destroy
  + Add the destroy method to the controller
    - <https://gist.github.com/88956c30e536f99a31cf>
* Provide access method to the delete action on the show page
  + <https://gist.github.com/eb9e552ed2366f308b38>