# 01 Introducing and installing Ruby gems and Ruby on Rails

Objectives

* Understand Ruby on Rails
* Verify Rails installation
* Understand and installing Ruby gems
* Use Ruby ERb for templating
* Create a new Rails application: 5 minute app
* Create a new controller and view: in a Hello World
* Enable default routing, launch Rails server, and test application
* Survey Rails application file structure
* Create a basic controller and view
* Understand HTTP URI structure
* Understand Rails request handling cycle

## Understanding Ruby on Rails

### What is Ruby on Rails?

* Ruby on Rails (“RoR”) is a full-stack web application framework built in Ruby
* *Full stack*: web server interaction, database interaction, template rendering, URI routing
* Makes wide use of standard software engineering patterns
  + Active record: database tables represented as classes whose objects represent individual records
  + Convention over configuration: standard naming and location conventions varied only if necessary
  + Don’t repeat yourself (DRY): every piece of knowledge should have a single source in the system
  + Model-View-Controller (MVC): data is kept distinct from both its appearance and how it is manipulated
* Developed by David Heinemeier Hansson while with 37Signals.com, released as open source in February 2005

### Who is using Ruby on Rails?

* Basecamp, Twitter, Groupon, Github, Shopify, Livingsocial, and many more

<http://rubyonrails.org/applications>

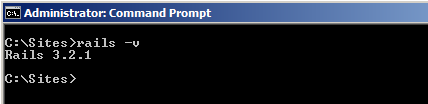
## Verifying Rails installation

### How do I install ROR again?

* The harder way  
  <http://rubyonrails.org/download>
* The very easy way  
  <http://railsinstaller.org/>

### How do I verify whether I have rails installed?

* Open a terminal or command window
* Use the rails-v command to check your installed version



## Understanding and installing Ruby gems

### What is RubyGems?

* RubyGems is the standard package manager for RoR
* *Package manager*: a standard format, installer, and distribution server for programs and libraries
  + A *program* is independently runnable code executed by a user or system to achieve a purpose
  + A *library* is reusable code used by a program to help it achieve its purpose
* *Gem*: Ruby program or library organized in a defined way, supported by description(s) of its contents
* gem: command line tool invoked to install a specified gem
* Gems are commonly (but not exclusively) distributed through <http://rubygems.org/>

### Where all can I find gems?

* Frequently used sources include

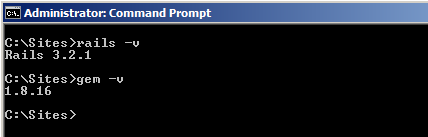
<http://rubygems.org/>

<https://www.ruby-toolbox.com/>

<http://rubyforge.org/>

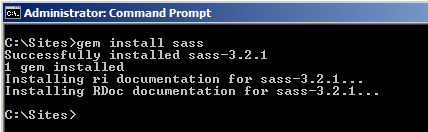
### How do I know if RubyGems in installed?

* Use the gem –v command to check for your installed version



### How do install a Gem?

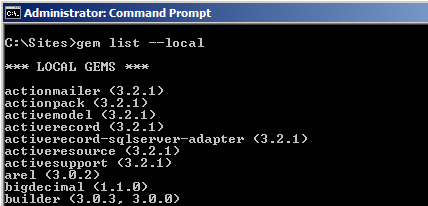
* Use the gem install <gem name> command
* Sass is a widely used Gem for simplified and extended CSS handling in RoR



* Once installed, Gems are generally made available as
  + Libraries you may require in your code
  + Command line tools you may use (rails itself is installable with gem)

### How do I know what Gems I have installed?

* Use gem --help to reach the starting information for built in Gem help files
* Use gem list --local to see what gems you have installed



## Using Ruby ERb for templating

### What is templating?

* *Templating* is when a document (HTML, XML, etc.) has variables, conditions, and loops embedded in it
* This enables, for example, a web page to be customized based on Form or URL data sent in its request
* A *template* is a text file designed to be processed by ERB
* Ruby supports templating through its built-in ERB library
* Ruby on Rails makes extensive use of templating

### How does ERB work?

* ERB loads and looks through a text file for (primarily) expressions and scriptlets
* The expressions and scriptlets are run, and the result is output wherever ERB is told
* By convention, ERB templates end with a .erb file extension  
  my\_template.erb
* By Rails convention, templates also include their format as a file extension. For example:  
  my\_template.html.erb  
  my\_template.json.erb  
  my\_template.rss.erb

### How does an ERB expression work?

* *Expressions* in the document are evaluated and replaced with their result

<p>Hi <%= @name %>! The date and time are now <%= Time.now %></p>

*Hi Fred! The date and time are now 2012-09-21 10:39:31 -0700*

### How does an ERB scriptlet work?

* *Scriptlets* in the document are run as code in relation to any text they surround
  + Expressions and scriptlets are very often used in combination

<ul><% for @worker in @employees %>  
 <li><%= @worker %></li>  
<% end %></ul>

* *Betty*
* *Barney*
* *Fred*
* *Wilma*

### Exercise BOOK - Using ERB for templating

* Complete this exercise from the exercise book

## Creating a new Rails application in five minutes

### How can you create a Rails app in five minutes?

* The instructor will provide a quick demo creating and configuring a (very simple) new RoR application
* Alternately, the instructor may narrate this video:  
  <http://www.youtube.com/watch?v=Wb7OrbbEXZQ>

### What are some important command line tools?

* RoR uses a wide variety of command line tools for code generation and application maintenance
* Each are discussed more later, but here is a quick introduction:
  + rails new <app\_name>: creates a new application
  + rails server: launches local WEBrick development web server to enable local test requests
  + rails generate <generator> <args> <options>: generate boilerplate code for an asset type
  + rake: used to execute a wide variety of administrative tasks
  + rails console: create an IRB sandbox to safely test code for your application
  + rails dbconsole: create a sandbox to safety test SQL queries for you application's database

### Exercise BOOK - creating a Hello World application

* Complete this exercise from the exercise book

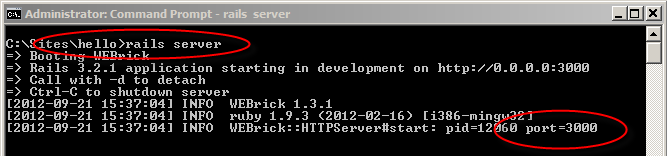
## Enabling default routing, launch Rails server, and test application

### Introducing Rails routing

* URLs (URIs) used with Rails apps do not point to a specific directory
* Instead, Rails examines the URI of incoming HTTP requests and *routes* them to specified code
* Which URIs are routed to which code is configured in this file  
  <app\_name>/config/routes.rb
* A default routing rule at the bottom of routes.rb can be enabled (uncommented) for simpler applications  
  match ':controller(/:action(/:id))(.:format)'

### Launching the development web server

* Ruby on Rails includes the WEBrick HTTP server to enable URI testing during development
* To launch WEBrick
  1. In a terminal or command window, navigate to your application root folder, such as:  
     /Sites/<app\_name>
  2. Use the rails server command



### Exercise BOOK - RUNNING and requesting a Hello World application

* Complete the exercise from your exercise book

### What is in the /app folder of a Rails application?

* app/assets: images, javascripts, and stylesheets to be requested by RoR templates
* app/controllers: switchboard classes which pass incoming data to models and providing outgoing data to views
* app/helpers: utility classes which provide methods to be used across many views
* app/mailers: support classes for sending email from the application
* app/models: classes representing and providing access to database tables
* app/views: ERB templates called to render data into pages
* app/views/layouts: template "master pages" to consolidate common view code (e.g., site menus, etc.)

### What else is in the default folder structure of a Rails Application?

* config: routing, database, and runtime environmental configuration files
* db: scripts managing database tables
* doc: location for auto-generated documentation created with RubyDoc
* lib: developer-built code to be shared across the application, along with rake tasks
* log: system and developer generated log files
* public: static HTML, standard HTTP error pages, and favicon.ico files for the application
* script: code used by command line tools to generate other code
* test: code for mocks, unit tests, fixtures, and functional tests
* tmp: location for session variables, temporary files, cached data, etc.
* vendor: location for external third-party code added to the application as a plug-in

## Creating a Basic controller and view

### How do you create a controller and view?

* A *controller* and *view* together create a requestable "page" for your application
* To create a controller use the controller generator with the rails command  
  rails generate controller <controller\_name> <view\_name>
* If a view\_name is provided then two things are created
  + A template at app/views/[controller\_name]/[view\_name].html.erb
  + An action (method) named view\_name in app/controllers/[controller\_name]
* As you may expect, an action and view named “index” will provide the default view for a controller

### Exercise - creating an index page for the hello application

* Complete the exercise from the exercise book

## Understanding HTTP URI structure

### What is a URI?

* Uniform Resource Locators (URLs) are a conceptual subset of Uniform Resource Identifiers (URIs)
* Wikipedia can provide detail if needed, but Rails documentation and files refer to incoming addresses as URIs  
  <http://en.wikipedia.org/wiki/Uniform_resource_identifier>
* In terms of what you need to know right now, a URL and URI are identical

### How does Rails understand a URI?

* By default, Rails recognizes four main parts of a URI for routing purposes  
  http://server/controller/action/id.format
* Compare this to the default route you saw in routes.rb (parentheses mean that portion is optional)  
  match ':controller(/:action(/:id))(.:format)'
* Parameter ("ID") and format values are discussed in a later lesson

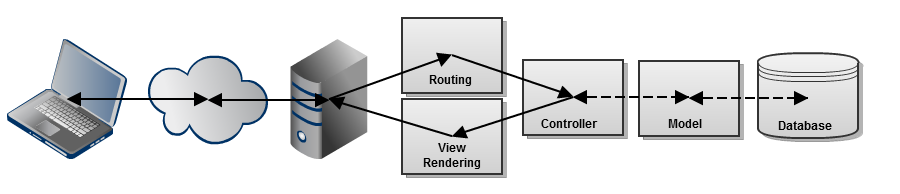
### How does this relate to an actual URL … er, URI?

http://server/controller/action/id.format

* http://127.0.0.1/hello/index.html
  + invokes the index action (method) of the hello controller
  + which calls index.html.erb to render the page
* http://127.0.0.1/hello/index/2.json
  + Invokes the index action (method) of the hello controller
  + passing a parameter ("ID") value of 2
  + and calls index.json.erb to render the result
* http://127.0.0.1/hello/
  + invoke the index action of the hello\_controller by default

## Understanding Rails request handling cycle

### How does Rails process a web request?



## Summary

### What have you learned in this Lesson?

* Ruby on Rails is a full stack web application framework which uses standard software engineering patterns
* Full stack means it handles web server and database interaction, URI routing, and template rendering
* Patterns used in RoR include Active Record, Convention over Configuration, DRY, and MVC
* RubyGems is the RoR package manager providing a standard format, installer, and distribution for programs and libraries
* Gems are installed using the gem install <gem\_name> command
* ERB is the templating library for RoR
* A template is a text file with embedded expressions and scriptlets
* Rendering a template means to execute its code and return the resulting text
* New RoR applications are created with the rails new <app\_name> command
* Because RoR manages URI routing, a routes.rb file must be configured
* You launch the development web server with the rails server command
* A controller is a class exposing methods ("actions") a URI may invoke
* A view is an ERB template called by a controller to render a response for its requesting URI

# 02 Introducing controllers, layouts, views, and styling

Objectives

* Understand Rails CSS handling
* Determine the layout for a view
* Use view content within a layout
* Introduce routing (check with Kyle)
* Set a default page

## Review of prior lesson

### What are Some answers to these questions

1. What is a Ruby gem?
2. What are some examples of software engineering patterns?
3. What does templating mean?
4. How do you create the foundation of a Ruby on Rails application?
5. What is WEBrick?
6. What do the terms model, view, and controller mean as they are used in RoR

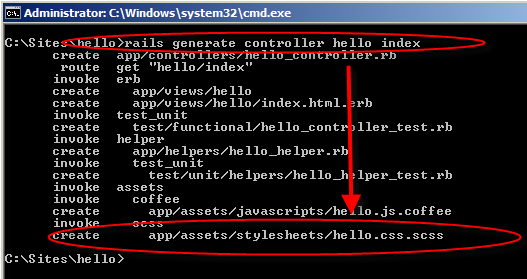
### Some answers to these questions

1. What is a Ruby gem?
   1. A Ruby program or library in a standard distribution format
2. What are some examples of software engineering patterns?
   1. Active record, Convention over configuration, Don’t repeat yourself (DRY), Model-View-Controller (MVC)
3. What does templating mean?
   1. To embed expressions and scriplets within a text document for runtime processing
4. How do you create the foundation of a Ruby on Rails application?
   1. Using the rails new <app\_name> command
5. What is WEBrick?
   1. The development and testing web server distributed with RoR
6. What do the terms model, view, and controller mean as they are used in RoR
   1. *Model* refers to the current state of application date; *View* refers to a display presented to a user; and *Controller* refers to how the association between Model and View are handled

## Understanding Rails CSS handling

### how does generating a controller generate a css file?

* Rails 3.1 introduced *asset management* which provides a centralized place to manage site-wide CSS
* The rails generate controller command creates a default CSS file for the controller’s default view



### what is a layout?

* A *layout* is an ERB template which serves as a “master” template into which a requested template is wrapped
* Layouts are defined in app/views/layouts
* Rails provides a default application.html.erb layout
  + runs for all requested templates unless another is specified
  + includes a stylesheet\_link\_tag expression which causes links to two CSS files to be inserted
  + includes a yield expression specifying where the requested template content is inserted

<!DOCTYPE html>

<html>

<head>

<title>Hello</title>

<%= stylesheet\_link\_tag "application", :media => "all" %>

<%= javascript\_include\_tag "application" %>

<%= csrf\_meta\_tags %>

</head>

<body>

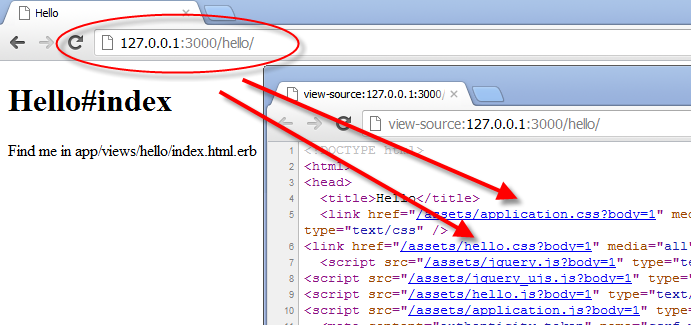
<%= yield %>

</body>

</html>

### What css is included by the default layout?

* Two CSS files are included in a requested template by the default layout from app/assets/stylesheets
  + application.css: potentially shared by all site pages
  + hello.css.scss: created by rails generate controller for generated view templates
* Both are included in templates affected by the default layout
* Sass is an extended version of CSS supported by Rails, which is implemented in .scss files
  + <http://sass-lang.com/>



### Exercise – configuring site wide and template specific css

* Complete the exercise from the exercise book

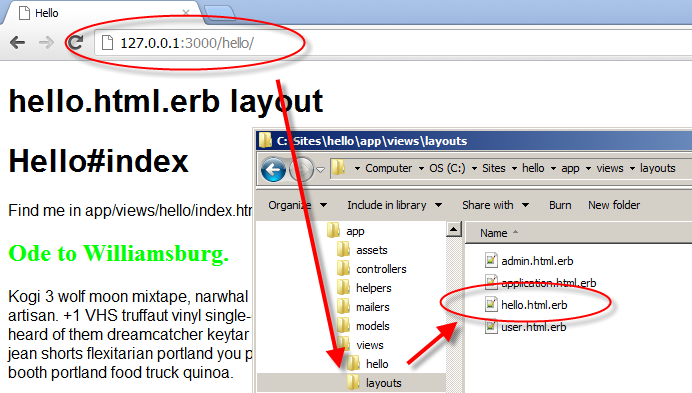
## determining the layout for a view

### What are the options for determining the layout for a view?

* Generally, a layout runs for every requested view template
* Which layout runs is determined in one of these ways
  + A layout with a name matching the requested view runs, if available
  + The controller for a view specifies a particular layout using its layout variable
  + The application.html.erb layout runs by default if no other layout is specified

### How do you determine the layout based on the controller name?

* Rails looks for a layout matching the requested controller name



### Exercise – creating a view specific layout

* Complete the exercise from the exercise book

### How do you determine the layout from within the controller?

* A controller can specify the layout for its view by assigning its layout variable
* The value could be set statically, as shown, or at runtime by a method

class HelloController < ApplicationController

layout "admin"

def index

end

end

### Exercise – Determining the layout from the controller

* Complete the exercise from the exercise book

## using view content within the layout

### How do you assign layout content from a view?

* Remember, HTML page structure (e.g., <head>) is normally set in the layout, not the view
* A view templates can implement a content\_for method to assign content for use by its layout
* To assign content within a view  
   <% content\_for :my\_content\_name do %>  
  My Page Title  
  <% end %>
* To use assigned content within a layout  
  <title><%= yield :my\_content\_name %></title>

### Exercise – assigning layout content from its view

* Complete the exercise from the exercise book

## Setting a default page

### How do you change the Rails default page?

* Rails displays the content at public/index.html as its site root page
* By default, this is the Rails “Welcome aboard” page
* This page is not a template, and so cannot include dynamic content
* To configure Rails to use a specified controller and its template as the site root page
  + Uncomment the root :to => assignment in config/routes.rb
  + Modify this to assign the desired controller and action, for example  
    root :to => 'hello#index'
  + Delete the default public/index.html site root page
  + Restart the Rails server

### Exercise – changing the site root page to a controller and its view

* Complete the exercise from the exercise book

## Summary

### What have you learned in this Lesson?

* Rails 3.1 provides centralized asset management for images, JavaScript, and CSS at app/assets
* A layout is a template which wraps a view template to provide common content for a set of views
* The default application layout includes centrally managed CSS into each template-generated page
* Sass is a tool built in to Rails 3.1 to extend the capabilities of CSS files
* In a template, the <%= yield %> expression is replaced by other content
* The default application layout may be overridden by naming a new layout the same as a controller
* The layout to be used for a controller may be assigned using its layout variable
* In a view template, content can assigned for display in its layout
  + assign in a <% content\_for :name do %> block
  + display in a <%= yield :name %> expression in the template
* The application root can be changed by
  + assigning root in config/routes.rb to a controller action
  + deleting public/index.html
  + restarting the Rails server

# 03 Introducing models and ActiveRecord

Objectives

* Introduce models by manually creating a form and related model
* Introduce rake:db to implement a model defined in a migration
* Introduce ActiveRecord to save and access model data

## review of previous lesson

### What are some answers to these questions?

1. What is the purpose of the app/assets directory?
2. What is Sass?
3. How does a layout work in relation to a view?
4. What are three ways Rails may choose a layout for a requested view?
5. What is the difference between <% … %> and <%= … %> when used in a template?
6. What does using a content\_for block in a template enable?

### some answers to these questions

1. What is the purpose of the app/assets directory?  
   *To store image files, JavaScript files, and CSS files to be used in common throughout a Rails application*
2. What is Sass?  
   *A Rails feature (plug in) enabling the use of variables and logic within CSS files*
3. How does a layout work in relation to a view?  
   *A layout wraps a view as a “master page” to enable easy reuse of common content*
4. What are three ways Rails may choose a layout for a requested view?  
   *Default layout (application.html.erb)  
   Named layout for controller (hello.html.erb for /hello)  
   Controller chooses layout by assigning a layout name to its layout variable*
5. What is the difference between <% … %> and <%= … %> when used in a template?  
   *<% … %> surrounds code to execute in relation to the text in which it’s embedded  
   <%= %> surrounds an expression to be evaluated and have its result display as text content*
6. What does using a content\_for block in a view template enable?  
   *Defining content in a view template which can be programmatically accessed in its layout template*

## introducing models by Manually creating a form and model

### How do you accept a browser parameter?

* Rails stores browser-submitted values in an automatically-generated hash named params
* The value from an HTML form field named review\_name would be accessed as params[:review\_name]

class ReviewsController < ApplicationController

def add

@review\_name = params[:review\_name]

…

* Remember that values in a controller’s instance variables are accessible by its related layout and view templates

### How do you create a simple data input form for your model?

* Rails supports helper methods which enable views to quickly generate and manage HTML forms
* A very simple form in a view intended to gather and display movie names could look like this

<h2>Recent book: <%= @movie\_name %></h2>

<%= form\_tag :action => 'add' do %>

<p>Movie name: <%= text\_field\_tag 'movie\_name', @movie\_name %></p>

<%= submit\_tag 'Add Movie' %>

<% end %>

### Exercise – creating a simple input and display form

* Complete the exercise from the exercise book

### What is a model?

* In the MVC software design pattern, the model represents the current state of all application data
* In Rails, the application model is implemented using a software design pattern called Active Record
* The Active Record pattern represents rows of data as objects, for example
  + Review: a class whose objects representing one review in the Reviews table
* All database interaction is managed through ActiveRecord objects, no SQL is required
* ActiveRecord is built into Rails  
  <http://ar.rubyonrails.org/>  
  <http://api.rubyonrails.org/classes/ActiveRecord/Base.html>
* Rails can generate some or all of the model code

### How do you manually implement a model?

* It’s helpful to look under the hood of things, even if Rails can generate them
* To create a basic model use the model generator with the rails command  
  rails generate model <model\_name>
* Rails expects a model to be named for a singular data type (e.g., book, movie, worker, etc.)
* Rails generates code to
  + create a database table named plurally for this type (e.g., books, movies, workers, etc.)
  + create a child of ActiveRecord::Base named for the model (e.g., book, movie, worker, etc.)

### how do you manually edit a migration to define table columns?

* A *migration* is a class derived from ActiveRecord::Migration which defines how a database table should be built
* It is generated – except for column definitions – in the db/migrate directory when a basic model is generated
* The change method defines the columns to be created and their data types

class CreateReviews < ActiveRecord::Migration

def change

create\_table :reviews do |t|

**t.string :movie\_name**

**t.string :reviewer\_name**

**t.string :synopsis**

t.timestamps

end

end

end

* ActiveRecord and migrations are discussed in more detail later in this course

### What types of data can ActiveRecord hold?

* ActiveRecord maps these data types to the corresponding type for the underlying database
* :binary – binary large objects (“BLOBS”) like pictures
* :boolean – true|false values
* :date – db specific representation for a date
* :datetime – db specific representation for a date and time
* :decimal – fractional numbers
* :float – fractional numbers
* :integer – non-fractional numbers
* :primary\_key – unique ID for each record in a table
* :string – list of up to 255 characters
* :text – list of characters up to db capacity for this type
* :time – db specific representation for a time
* :timestamp – dp specific representation for a time

### How do you manually edit a model to expose accessors for model data?

* A *model* is a class derived from ActiveRecord::Base which represents a single record of data for a table
* It is generated – except for accessors related to columns – in the app/models directory when a basic model is generated
* Accessors are defined for the data each record of the corresponding table will expose

class Review < ActiveRecord::Base

attr\_accessible :movie\_name

attr\_accessible :reviewer\_name

attr\_accessible :synopsis

end

## introducing rake Using rake:db to implement a model

### What is the rake tool?

* Rake is Rails software task management tool, often used to automate moving, compiling, and deleting Ruby files
* Rake executes tasks defined in *rakefiles* which describe tasks to be completed using Ruby anonymous function blocks
* Introduction to using rake  
  <http://guides.rubyonrails.org/command_line.html#rake>
* User guide to rake  
  <http://docs.rubyrake.org/user_guide/index.html>
* Rake is discussed in more detail throughout the course

### What does the rake:db migrate task do?

* Rake tasks may be grouped into namespaces for organization, like rake db:[task\_name] for database related tasks
* rake db:migrate : executes migrations defined in a Rails application, to modify its database tables
* Rake tasks for databases are discussed in more detail in the later lesson specifically on Migrations

## Introducing ActiveRecord to save and access model data

### How do you interact with a database using a model?

* Models support methods to create, read, update, and delete records
* A model can be used to create a new record with specified values, and return a true|false for success or failure

@review = Review.create({

:movie\_name => ‘Battleship Potemkin’,

:reviewer\_name => ‘Lavrentiev’,

:synopsis => ‘History!’

})

* A model can retrieve all records, all records where certain values match, or find one record by an ID value

@reviews = Review.all  
@reviews = Review.all.where(“reviewer\_name = ?”, params[:reviewer\_name])

@review = Review.find(params[:id])

* Models and their methods are discussed in more detail later in this course

### How do you create a simple data display form for your model?

* Rails supports helper methods which enable views to quickly generate and manage HTML display
* A very simple form in a view intended to display a list movie names could look like this

<p>Movie list:</p>

<ul>

<% @movies.each do |movie| %>

<li><%= movie.movie\_name %></li>

<% end %>

</ul>

* Forms, input, validation, and display are discussed in more detail later in the course

### How can you see information being posted from the browser?

* Rails supports a debug function which can be displayed as a view template expression to display browser input

<%= debug(params) %>

* Other arguments to the debug function, each providing different information sets, include assigns, controller, base\_path, flash, request, response, session

### Exercise – manually creating and using a model

* Complete the exercise from the exercise book

## Summary

### What have you learned in this Lesson?

* form parameters are retrieved in a controller through the params[:value\_name] hash
* view templates support helper methods for form generation such as form\_tag, text\_field\_tag, submit\_tag
* The Model in Rails MVC is implemented through the Active Record pattern
* An ActiveRecord model representing a records in a table of Books would be a class named Book
* Rails can generate a model and related code using rails generate model <model\_name>
* A *migration* is derived from ActiveRecord::Migration and defines how to create a database table
* A *model* is derived from ActiveRecord::Base and exposes accessors for data from the columns of its table
* Rake is Rails task execution tool
* rake db:migrate causes a Rails application to run its migration classes to (re)create it tables
* A model can be used to create, find, retrieve all or all.where records
* The all method of a model returns a collection you can loop to display in a form

# 04 Creating forms for data models

Objectives

* Generate scaffolding for a model
* Understand REST and Rails scaffolding
* Understand and create form field elements with helpers
* Create custom helper methods

## review of previous lesson

### What are some answers to these questions?

1. How do you access form field parameters within a controller?
2. What are some form helper methods?
3. What is a Rails model?
4. What is a Rails migration?
5. What is the purpose of Rake?
6. How might you retrieve all reviews for a movie named ‘Battleship Potemkin’?

### some answers to these questions

1. How do you access form field parameters within a controller?

*Using the params hash*

1. What are some form helper methods?  
   *form\_tag, text\_field\_tag, submit\_tag*
2. What is a Rails model?

*A class derived from ActiveRecord::Base representing one record from a table and enabling access to that table*

1. What is a Rails migration?

*A class derived from ActiveRecord::Migration defining how a table should be created*

1. What is the purpose of Rake?

*To execute a set of tasks defined in a rakefile*

1. How might you retrieve all reviews for a movie named ‘Battleship Potemkin’?  
   *Movie.all.where(“movie\_name = ?”, “Battleship Potemkin”)*

## Generating scaffolding for a model

### What is Rails scaffolding?

* Rails supports generating a scaffold for a particular model

**rails generate scaffold <model\_name> <column\_name:data\_type> …**

* A movie review application could be quickly generated using

rails generate scaffold Review movie\_name:string synopsis:text

* Generating a scaffold for a model creates
  + A model class exposing the relevant accessors
  + A migration to create the relevant table(s)
  + A route configuration to map requests to the controller
  + A controller to move data between the components
  + Four views to show a record, edit it, list all records, or add a new record
  + Additional supporting files for scripting, CSS, testing, and helper methods

### How is scaffolding useful?

* Scaffolding is useful for
  + simple applications and client demonstrations
  + demonstrating best practices regarding REST based design (more on this ahead)
* Scaffolding may be too simplistic for some applications

### Exercise – creating a Rails application using scaffolding

* Complete the exercise from the exercise book

## Understanding REST and Rails scaffolding

### What is REST?

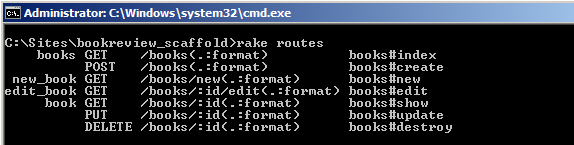
* REST stands for REpresentational State Transfer
* Web applications traditionally use only the HTTP GET and POST methods from the browser
  + GET: data displayed in the URL query string
  + POST: data hidden in the request packet
* REST is an approach to web application development making full use of five standard HTTP protocol methods  
   GET POST *PUT DELETE HEAD*
* Using only GET and POST can lead to ugly URLs, unbookmarkable applications, caching issues, and related problems
* “REST”-ful URIs should always cause the same result and provide the same output
* Rails supports older browsers which may not support PUT and DELETE using hidden form field based workaround

### How does Rails handle “REST”-ful requests?

* Scaffolding adds a *resources* entry to config/routes.rb

resources :books

* A resources entry equates to a set of routes defined for the Books resource
* All routes defined in an application can be viewed using the rake routes command
* Review which route patterns are mapped to which action (method) in the books controller



### What actions (methods) does a restful controller provide?

* Rails provides 7 actions when generating a REST-ful application programming interface (API)
  + *index*: return all model data, list it in the requested format
  + *create*: create new model object using values passed as parameters, display success or failure
  + *new*: create new empty model object, display an input form in the requested format
  + *edit*: return record specified by ID, display it editably in the requested format
  + *show*: return record specified by ID, display it non-editably in the requested format
  + *update*: retrieve record specified by ID, update it using values passed as parameters, display success or failure
  + *destroy*: delete record specified by ID, redirect to index

### What is the :format value in a Rails URI?

* What appear as file extensions (e.g., .html, .json) in a URI are assigned to a format variable within the controller
* The respond\_to method of a controller can examine the format variable and determine how to format the response

# GET /books/1

# GET /books/1.json

def show

@book = Book.find(params[:id])

respond\_to do |format|

format.html # show.html.erb

format.json { render json: @book }

end

end

* A format of .html means the relevant HTML view template should be run, but other formats can also be defined

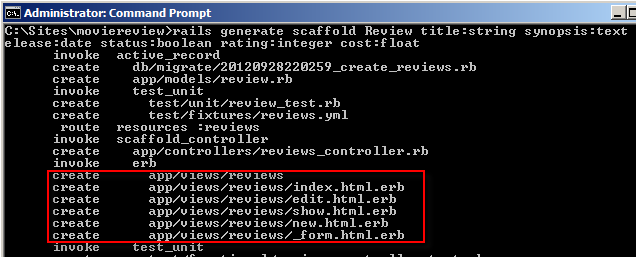
### Exercise – understanding rest-ful URIs

* Complete the exercise from the exercise book

## understanding and Creating form field elements

### What views does Rails scaffolding create by default?

* Rails scaffolding generates four views for a model
* Rails scaffolding also generates one partial view (“partial”)



### What is a partial view?

* A *partial* enables the same code to be reused in several places (DRY: Don’t Repeat Yourself)
* Partials are
  + saved with a preceding underscore in their file name  
    \_form.html.erb
  + displayed using the render command in an action or view without the preceding underscore  
    <%= render form %>

### Exercise – create and review multi-field forms

* Complete the exercise from the exercise book

### What is the form\_for command?

* A FormHelper or “helper method” is designed to help build and render HTML content
* form\_for is a helper which understands models and know how to generate HTML forms for them
  + if a model is empty, a blank input form is generated
  + if a model has value, the values are displayed in the form fields for editing
* For most controls, Rails helper methods generate the code needed to set the field value when editing the form
  + Rails does not generate radio buttons, though, so this behavior must be built manually

### What is different between model and non-model based form helpers?

* Non-model based form helpers

<%= form\_tag(“/price\_lookup”, :method => “get”) do %>

<%= label\_tag(:product\_id, “Product ID:”) %>  
 <%= text\_field\_tag(:product\_id) %>

<%= submit\_tag(“Price Lookup”) %>

<% end %>

* Model based form helpers

<%= form\_for(@product) do |f| %>

<%= f.label(:product\_id) %>

<%= f.text\_field(:product\_id) %>  
 <%= f.submit %>

<% end %>

### What helper methods are built in?

* Both model (“\_tag” suffix) and non-model (“form\_for” and “f.” prefix) helpers are available for each of these

|  |  |  |  |
| --- | --- | --- | --- |
| label | text\_field | text\_area | check\_box |
| radio\_button | select | date\_select | time\_select |
| datetime\_select | password\_field | submit | hidden |

### How does a label helper method work?

* Helper method for a model named Review (f is a reference to the form\_for object itself)

<%= f.label :title %>

<%= f.text\_field :title %>

* Generated HTML, label identified as for the corresponding review\_title input tags

<label for="review\_title">Title</label>

<input id="review\_title" name="review[title]" size="30" type="text" />

### How do text\_field and text\_area helpers work?

* Helper method for a model named Review

<%= f.text\_field :title %>

* Generated HTML (field named review[title] to identify the model variable assigned to it)

<input id="review\_title" name="review[title]" size="30" type="text" />

* Helper method for a model named Review, could optionally specify :cols => value and :rows => value

<%= f.text\_area :synopsis %>

* Generated HTML

<textarea cols="40" id="review\_synopsis" name="review[synopsis]"  
rows="20"></textarea>

### How does a checkbox helper work?

* Helper method for a model named Review
  + As with all HTML helpers, could optionally assign CSS class as :css => ‘class\_name’
  + If needed, can override checked status as :checked => ‘checked’

<%= f.check\_box :status %>

* Generated HTML, corresponding hidden field ensures value sent even if unchecked

<input name="review[status]" type="hidden" value="0" />

<input id="review\_status" name="review[status]" type="checkbox" value="1" />

### How does a date\_select helper work?

* date\_select, time\_select, and datetime\_select each work similarly to this example
* Helper method for a model named Review

<%= f.date\_select :release %>

* Generated HTML

<select id="review\_release\_1i" name="review[release(1i)]">

<option value="2007">2007</option>

... (current marked selected)

<option value="2017">2017</option>

</select>

<select id="review\_release\_2i" name="review[release(2i)]">

<option value="1">January</option>

... (current marked selected)

<option value="12">December</option>

</select>

<select id="review\_release\_3i" name="review[release(3i)]">

<option value="1">1</option>

... (current marked selected)

<option value="31">31</option>

</select>

### how does a number helper work?

* Helper method for a model named Review

<%= f.number\_field :rating %>

* Generated HTML, server side validation errors distinguishing float, decimal, and integer display at top of form

<input id="review\_rating" name="review[rating]" type="number" />

### How could you create a radio button set?

* There is a radio\_button helper method, but Rails does not generate radio button sets
* Set the radio button labels and values as a hash in the controller

def initialize

@ratings = {"G" => "General Admission", "PG" => "Parental Guidance",

"R" => "Restricted", "X" => "Adults Only"}

end

* Loop over the hash to generate the radio buttons and use a condition to set which is checked in the form

<fieldset>

<legend>Rating</legend>

<% @ratings.each do |label, code| %>

<% if :rating == code %>

<%= f.radio\_button :rating, label, :checked %>

<% else %>

<%= f.radio\_button :rating, label %>

<% end %>

<label for="<%= 'movie\_rating\_' + code %>">

<%= label %>

</label>

<br/>

<% end %>

</fieldset>

### How could you manually populate a select control?

* Like radio buttons, there is a select helper method, but Rails will automatically not generate it
* Set the select labels and values as a hash (or two-dimensional array) in the controller

def initialize

@ratings = {"General Admission" => "G", "Parental Guidance" => "PG",

"Restricted" => "R", "Adults Only" => "X"}

end

* Assign the has to the select helper

<%= f.select :rating, @ratings %>

### How do you assign CSS or other attributes to a helper?

* CSS or other inline attributes can be added to a Rails helper method
* Attributes added to the helper will be added to the generated tag, or override attributes already to be set

<%= form\_for (@person) do |f| %>  
<%= f.text\_field :name, :class => “MyClass”, :size=”45” %>

<input id=”person\_name” name=”person[name]” type=”text”

class=”MyClass” size=”45” />

### How are input validation errors displayed?

* *Validation* means to verify user input matches relevant data type and format rules
* Validation is discussed in more detail later in the course
* Rails scaffolding validates input on the server – not browser – and displays errors in \_form.html.erb

<% if @review.errors.any? %>

<div id="error\_explanation">

<h2><%= pluralize(@review.errors.count, "error") %>

prohibited this review from being saved:</h2>

<ul>

<% @review.errors.full\_messages.each do |msg| %>

<li><%= msg %></li>

<% end %>

</ul>

</div>

<% end %>

### exercise – customizing a rails scaffolding form

* Complete the exercise from the exercise book

## Creating helper methods

### what is a customer form helper method?

* A rails form helper method generates HTML content and returns it to the view where it is called
* Rails generates helper method modules for the application and models in app/helpers
  + application\_helper.rb: methods available in all views
  + [model\_name]\_helper.rb: methods available in views presented for this model and its controller
* Helper methods are defined in the ActionView::Helpers::FormBuilder class
  + creating and using a customized FormBuilder is covered in a later lesson

### How does the radio\_button helper method work, called outside of a form?

* So far, each helper method discussed has been called within a form\_for block

<%= form\_for(@review) do |**f**| %>

...  
 <%= **f**.radio\_button :rating, label, :checked %>

* Each of the helper methods can be also called independently of the form\_for helper (e.g., in a custom helper)

radio\_button(model\_name, target\_property, value)

* The radio\_button helper marks a radio button as checked if target\_property matches the value

### how could you write a custom helper?

* A custom helper takes form values, generates an appropriate string of HTML, and returns the HTML
* Built in form helpers can be called by a custom form helper
* Custom helper method defined in app/helpers/reviews\_helper.rb

module ReviewsHelper

def radio\_buttons(model\_name, source\_hash, target\_property, legend)

html = ''

html << '<fieldset><legend>' + legend + '</legend>'

source\_hash.each do |key, value|

html << radio\_button(model\_name, target\_property, value)

html << label(target\_property, value)

html << '<br/>'

end

html << '</fieldset>'

return html.html\_safe

end

end

* Custom helper method called in \_form.html.erb

<%= radio\_buttons("review", @ratings, :rating, "Rating") %>

### What is the html\_safe method?

* Rendering string values into a browser can be dangerous
* Rails strings support a true|false .html\_safe assertion, enabling strings to be flagged as being safe for display
* If a string is not flagged as safe, Rails views may escape angle brackets to prevent malicious script from running

### exercise – writing a customer helper method

* Complete the exercise in the exercise book

## Summary

### What have you learned in this Lesson?

* Rails scaffolding generates a migration, route configuration, controllers, views, and support files for a model
* A *partial* is a file named with a preceding underscore (\_form.html.erb) included in another file using the render method (<%= render ‘form’ %>)
* Partials support implementation of the Don’t Repeat Yourself (DRY) design pattern in applications
* Representational State Transfer (REST) is an approach to remote method invocation taking full advantage of the HTTP syntax: GET POST PUT DELETE HEAD
* REST-ful URIs (URLs) should always create the same result
* A resources entry in config/routes.rb creates route patterns for index, create, new, edit, show, update, and destroy methods of a corresponding controller
* The rake routes command displays all routes configured for an application
* A URI "file extension" is mapped to the :format value of the request which may be used to determine output
* The form\_for method, used as a tag, may wrap a form and support use of form helper methods for generating HTML
* There are built in form helpers methods for all standard HTML form elements
* Rails scaffolding does not generate radio button or select sets, they can be hand-coded
* Methods defined in app/helpers may be called in views to support HTML or other output generation
* Strings support the .html\_safe method to flag strings guaranteed safe for browsers

# 05 Validating form input

Objectives

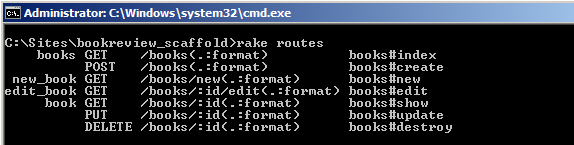
* Implement declarative validation
* Customizing data validation
* Introduce regular expressions
* Implement custom validation method

## review of previous lesson

### What are some answers to these questions?

1. What routes are configured by using a resources declaration in config/routes.rb?
2. How would you name and then use a partial named *fred*?
3. What are some characteristics of REST-ful URIs?
4. How can you easily examine all routes configured for an application?
5. What is the purpose of the form\_for method within a view?
6. What is the purpose of a form helper method?
7. Where would you define application-wide form helper methods?

### some answers to these questions

1. What routes are configured by using a resources declaration in config/routes.rb?  
   
2. How would you name and then use a partial named *fred*?

\_fred.html.erb <%= render ‘fred’ %>

1. What are some characteristics of REST-ful URIs?  
   Always return same results, make full use of HTTP protocol
2. How can you easily examine all routes configured for an application?  
   rake routes
3. What is the purpose of the form\_for method within a view?  
   Provide context for helper methods called within the view
4. What is the purpose of a form helper method?  
   To generate correctly formatted output
5. Where would you define application-wide form helper methods?  
   app/helpers/application\_helpers.rb

## Implementing declarative validation

### What is input validation?

* Applications are as good as their data
* *Validation* means to verify input is of the correct data type before storing it, notifying the user of any errors

### How are validation rules defined in Rails?

* Rails defines validation rules in the model(s) for an application
* By enforcing validation in the model, the rules apply regardless how the model is invoked
* Validation helpers enable easy implementation of validation rules using the general syntax  
  validates [field\_name], [validation\_type], [ … additional options ]
* Validation may also be implemented using more readable helper names, such as

validates\_presence\_of [field name], [options]

validates\_length\_of [field name], [options]

validates\_format\_of [field name], [options]

validates\_inclusion\_of [field name], [options]

validates\_uniqueness\_of [field name], [options]

validates\_numericality\_of [field name], [options]

* Rails also supports custom validation using either *regular expressions* or custom methods

### How do you implement a declarative validation rule?

* Assign and configure validation rules in the application model. For example:

# ensure value is provided in the password field named password  
validates\_presence\_of :password  
  
# ensure password is 8 to 14 characters long, and provide a custom error message  
validates\_length\_of :password, :in => 8..14, :message => “8 to 14 characters!”

# ensure password contains at least one number  
validates\_format\_of :password, :with => /[0-9]/

# ensure release\_year is numeric, not too early, integer, and optional (nil ok)  
validates\_numericality\_of :release\_year, :greater\_than => 1910, :less\_than\_or\_equal\_to => Time.now.year, :only\_integer => true, :allow\_nil => true

### How do you specify when validation occurs?

* By default, validation occurs on both creation and update of a model
* The :on parameter enables specifying validation which occurs on one but not the other

:on => ‘create’

:on => ‘update’

* Example: enforcing a rule for new passwords that
  + was not originally required
  + and, will not be enforced during profile updates

validates\_format\_of :password, :with => /[0-9]/, :on => ‘create’

### exercise – implementing validation helpers

* Complete the exercise in the exercise book.

## Customizing data validation

### what is a regular expression?

* Regular expressions (“regex”) are a syntax for testing whether a string matches a pattern
* Regex syntax is involved enough to be beyond the scope of this course, but useful resources include  
  <http://www.oreillynet.com/pub/a/ruby/excerpts/ruby-learning-rails/ruby-guide-regular-expressions.html>

<http://rubylearning.com/satishtalim/ruby_regular_expressions.html>

* Regex may be used in form validation, using the :with parameter
* A previous example of a regex demonstrates requiring a character in the range from 0 through 9

validates\_format\_of :password, :with => /[0-9]/

* A more involved example demonstrates attempting to require a valid email address

validates\_format\_of :email,  
with => /\A([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})\Z/i,  
:message => “may not contain a valid email address”

## Implementing custom validation methods

### How do you conditionally validate depending on field values?

* Validation helpers support an :if parameter
* A custom method, which returns true|false, can be implemented to determine *if* a validation rule should apply

validates\_presence\_of :synopsis, :if => :movie\_is\_g\_or\_pg?,

:message => "required for G and PG"

def movie\_is\_g\_or\_pg?

if self.rating == "General Admission" || self.rating == "Parental Guidance"

return true

else

return false

end

end

### exercise – conditionally validating fields

* Complete the exercise in the exercise book

## Summary

### What have you learned in this Lesson?

* Validation means to verify input before storing it
* Rails implements validation in the model rather than the view
* Validation helpers can test presence, length, format, numericality, and more
* Helpers support parameters to set which field should be validated and how
* Helpers can be set to test only on create or update of a model
* Regular expressions are a syntax to test if a string matches a pattern
* Fields can be conditionally validated by assigning a custom method to the :if parameter

# 06 Handling file uploads and automating form building

Objectives

* Understand Ruby file i/o (Tempfile)
* Implement file upload
* Introduce FormBuilder
* Implement custom form builder
* Modify an existing form helper

## review of previous lesson

### What are some answers to these questions?

1. Why does Rails enforce validation in the model and not the controller or view?
2. How could you validate whether a field’s value is between a range of two numbers?
3. Can you enforce validation only when a model is being edited? How?
4. What is a regular expression?
5. How could require a value for a field only if another field has a specified value?

### some answers to these questions

1. Why does Rails enforce validation in the model and not the controller or view?  
   *So that it is applied regardless whether the model is invoked by form or direct REST invocation*
2. How could you validate whether a field’s value is between a range of two numbers?  
   *validates\_numericality\_of :name, :greater\_than => 0, :less\_than\_or\_equal\_to => 10*
3. Can you enforce validation only when a model is being edited? How?  
   *:on => ‘update’*
4. What is a regular expression?  
   *A syntax defining a pattern to be matched against a string*
5. How could require a value for a field only if another field has a specified value?  
   *Writing a boolean method and calling it with a validation helper’s* :if *parameter*

## Understanding Ruby file i/o (Tempfile)

### How can you temporarily store data to the file system?

* Ruby Tempfile class creates a file in user’s /temp directory and opens it for *write* access

require ‘Tempfile’  
myfile = Tempfile.new(‘Demo.txt’)  
myfile << ‘Some sample text’  
myfile.flush  
myfile.close

### How do you read and write from a file?

* Ruby File class lets you create, write, save, and read files
* Open a file to append new files written to it, then close

file = File.new(“myfile.txt", "a")

time = Time.new

file.write(time.localtime.to\_s + "\n")

file.write(sentence + "\n")

file.close

* Open a file to read and print lines from it, then close

file = File.new(“myfile.txt", "r")

while(line = file.gets)

puts line

end

file.close

## Implementing file upload

### How do you enable a rails form to upload a file?

* Configure the form to encode multipart/form-data

<%= form\_for(@review, :html => { :multipart => true }) do |f| %>

* Implement a file\_field element on the form

<div class="field">

<%= f.label :image %><br/>

<%= f.file\_field :image %>

</div>

### How do you store uploaded files?

* Code to handle upload commonly placed in the model (or a module to be included in the model)
* Why not the controller?
* Model code must
  + Determine where to store the data, and by what name
  + Create the file, write data to it, and save it
  + Save related data about the file, like its file extension
  + Expose helper methods to provide the file path, URI, and whether the file exists
* View code asks the model if it has an image, and if so, where it’s located

### Where do you store uploaded files?

* Commonly files are stored on the file system (could also store as BLOB: binary large object in database)
* If storing images for web page access, store in the /public directory in a relevant sub-directory

IMAGE\_STORE = "#{Rails.root}/public/image\_store"

* May modify database to store additional file data, such as its type (file extension, .jpg, .png, etc.)

rails generate migration add\_image\_type\_to\_review

...

class AddImageTypeToReview < ActiveRecord::Migration

def change

add\_column :reviews, :image\_type, :string

end

end

...

rake db:migrate

### How do you store the uploaded file data?

* Uploaded file is available in an automatic variable called file\_data
* If uploading through a file\_field named :image, would assign file\_data to that name in the model

def image=(file\_data)

unless file\_data.blank?

# assign file\_data to instance variable for use in store\_image

@file\_data = file\_data

# assign image type (extension) to self.image\_type

self.image\_type = file\_data.original\_filename.split('.').last.downcase

end

end

### How do you store the uploaded file data?

* Use a callback, like after\_save, to call a method to store the uploaded data after other data has been saved

after\_save :store\_image

* Create a directory and file as needed, write the file data to it, and save it

def store\_image

if @file\_data

# create directory at IMAGE\_STORE if it does not exist

FileUtils.mkdir\_p IMAGE\_STORE

# save image data to this location

File.open(image\_filename, 'wb') do |f|

f.write(@file\_data.read)

end

# nil file\_data in memory so it won't be resaved

@file\_data = nil

end

end

### How do you store the uploaded file data?

* Writer helper methods to provide the filename, URI, and whether a model has a related file

def image\_filename

return "#{IMAGE\_STORE}/#{id}.#{image\_type}"

end

def image\_uri

return "/image\_store/#{id}.#{image\_type}"

end

def has\_image?

return File.exists? image\_filename

end

### How do you store the uploaded file data?

* If a model has an image, display it in the show.html.erb view

<p>

<b>Image:</b>

<% if @review.has\_image? %>

<%= image\_tag @review.image\_uri %>

<% else %>

No image available

<% end %>

</p>

### Exercise – implementing file upload and display

* Complete the exercise in the exercise book

## Implementing a custom form builder

### what is a formbuilder?

* ActionView::Helpers::FormBuilder is the class which defines the various form helper methods
* A new class can be derived from it, to add new helpers or override built in helpers with new behavior

class CustomFormBuilder < ActionView::Helpers::FormBuilder

end

* The customized FormBuilder can be assigned to a form\_for element to modify its helpers

<%= form\_for(@review, :html => { :multipart => true },

:builder => CustomFormBuilder) do |f| %>

### How could you create a new form helper?

* In a class derived from FormBuilder declare a method which returns HTML, possibly using an existing helper
* For example, create a select control pre-populated with a list of countries

def country\_select(method, options={}, html\_options={})

create\_countries\_array

select(method, @countries, options, html\_options)

end

def create\_countries\_array

@countries = Array.new

@countries.push(["Australia","Australia"])

@countries.push(["Canada","Canada"])

end

### How do you use a new form helper?

* Assign the custom FormBuilder to the form\_for element, to make its method available

<%= form\_for(@review, :html => { :multipart => true },

:builder => CustomFormBuilder) do |f| %>

* Use the custom methods within that form\_for element, along with the built in methods it inherits

<div class="field">

<%= f.label :country %><br/>

<%= f.country\_select :country %>

</div>

### exercise – customizing formbuilder to create a new form element

* Complete the exercise in the exercise book

## modifying existing form helpers

### how can you change the behavior of existing form helpers?

* In a custom FormBuilder, override an existing helper method by creating a new method with the same name
* Call the existing helper using the super keyword, but modify its results as appropriate
* For example, modify the label helper to display a red asterisk if its field will be validated as required by the model

def label(method, options={}, html\_options={})

html\_to\_add = ""

if options[:required]

html\_to\_add = "<span style='color:red;font-size:18'>\*</span>"

end

super(method, options) + html\_to\_add.html\_safe

end

### exercise – extending an existing form helper method using CSS

* Complete the exercise in the exercise book

## Summary

### What have you learned in this Lesson?

* Ruby reads and writes from the file system using the Tempfile and File classes
* Files can be open for read, write, and/or append operations
* Browser forms support file upload if marked for multipart/form-data
* Uploaded files are available in the model in the automatic file\_data variable
* Files uploaded for later display should be saved in the /public directory
* The model must know where and by what name its related files can be found
* A FormBuilder provides the helper methods for automating form building
* FormBuilder can be derived to add new helpers and customize existing helpers

# 07 Creating multi-model associations

Objectives

* Understand data model associations
* Implement one-to-many model association
* Understand Rails migrations
* Implement many-to-many model association

## review of previous lesson

### What are some answers to these questions?

1. When a file has been uploaded by a form, where and how do you access the uploaded file?
2. How do you modify a form to support file uploading?
3. Where are form helper methods defined?
4. How do you change which form builder will be used by a form?
5. How do you change the behavior of an existing form helper?
6. Why might you customize a form helper?

### some answers to these questions

1. When a file has been uploaded by a form, where and how do you access the uploaded file?  
   *Generally using the file\_data variable in the application model*
2. How do you modify a form to support file uploading?  
   *Set its encoding type to multipart/form-data*
3. Where are form helper methods defined?

*In ActionView::Helpers::FormBuilder, or whatever FormBuilder is assigned to a form as its :builder*

1. How do you change which form builder will be used by a form?  
   *Derive the FormBuilder class, and assigned the derived class to the form\_for :builder attribute*
2. How do you change the behavior of an existing form helper?  
   *Override its method in a custom FormBuilder class*
3. Why might you customize a form helper?  
   *Create conditional visual behaviors, pre-populate common data, design a consistent custom appearance*

## Understanding data model associations

### how does rails work with related and unrelated data?

* A database *table* is like a spreadsheet
* Rails interacts with each table through a *model*
* If the data in two tables is unrelated, write the code you need for each model
* But, what if the data in two separate tables is related somehow? For example:
  + one movie > many reviews
  + one director > many movies
  + many movies > many film festivals
  + one movie > one production studio
* When should data be separated? Should movies and reviews be separate models?

### What is relational database design?

* All the types of data needed for an application or system is known as its *domain*
* A database *table* defines a set of columns of specified data types for one part of a domain
  + tables may have *relationships* with other tables
* What tables and relationships should be defined for a domain is guided by *data normalization* standards
  + one to many relationships can be defined using primary key to foreign key relationships
  + many to many relationships can be defined using a join table
* A *primary key* is a unique identifier for each record in a table, often simply called “ID”
* A *foreign key* is a column defined for one table to contain primary key values from another related table
* A Rails *model* represents a record, its table, and its relationships
  + models may have *associations* with other models
* Rails provides assistance in defining associations between models
  + database design knowledge is helpful with Rails and, for some things, required
  + <http://en.wikipedia.org/wiki/Relational_database>

## Implementing one-to-many model association

### How do I create a one to many relationship in the underlying tables?

* The “many” table needs a foreign key column to hold a reference to the related “one” table
  + one movie has many reviews
  + each review belongs to one movie
* The Review table (and model) needs a foreign key column for a Movie

rails generate scaffold Review author:string review:text score:integer **movie\_id:integer**

* The Movie table (and model) does not need a foreign key column for a Movie

rails generate scaffold Movie title:string synopsis:text release:date

### How do I define a one to many relationship in the models?

* The many reviews each belong to one movie

class Review < ActiveRecord::Base

belongs\_to :movie

end

* Each movie has many reviews

class Movie < ActiveRecord::Base

has\_many :reviews

end

### How do I display available “ones” when creating a “many” record?

* Models support a find method for accessing its records
* One way to display movies from a Review form would be to find all records using the Movie model and display them within a Select control

<%= f.select :movie\_id, Movie.find(:all, :order => "title").collect { |movie| [movie.title, movie.id]} %>

### exercise – creating and using two models in one form

* Complete the exercise in the exercise book

### How do I determine what records have been created?

* Structured Query Language (SQL) is the primary way to interact with a relational database

<http://en.wikipedia.org/wiki/Sql>

* SELECT queries retrieve database records use this general format

SELECT [column names or \*] FROM [table name(s)] WHERE [conditions are true];

* All records in a Movies table could be retrieved using

SELECT \* FROM Movies;

* Rails supports a database console to run SQL queries against the current application

rails dbconsole

### How do I validate the “one” record exists when creating a related “many”?

* If a model belongs\_to another model, you should only create a record if its related record exists
* The existence of a related record can be validated using the validates\_existence command in a Model

class Review < ActiveRecord::Base

belongs\_to :movie

validates\_existence\_of :movie, :both => false

end

* The validates\_existence command is enabled by installing a Gem

gem install validates\_existence

* After installation, add the gem to [application\_name]/Gemfile

gem ‘validates\_existence’, ‘>= 0.4’

### exercise – validating the existence of related records

* Complete the exercise in the exercise book

### How do I destroy related “many” records when I destroy their “one”?

* If a movie record is destroyed, related reviews should be destroyed, else they are orphaned in the database
* The has\_many command enable destruction of dependently related records

class Movie < ActiveRecord::Base

has\_many :reviews, :dependent => :destroy

end

### exercise – listing and destroying related records

* Complete the exercise in the exercise book

### How do I create a route to display a list of related records?

* An additional member route can be added to the :movies resource in config/routes.rb

# comment out existing resources routing   
# resources :movies

# add member to extend :movies resources routing

resources :movies do

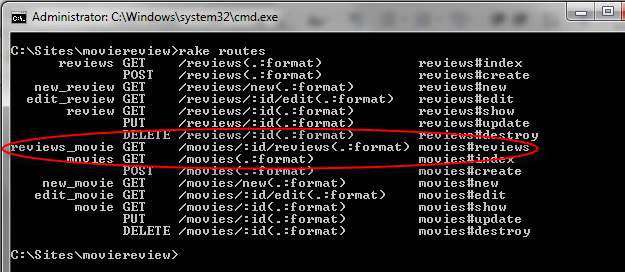
member do

get 'reviews'

end

end

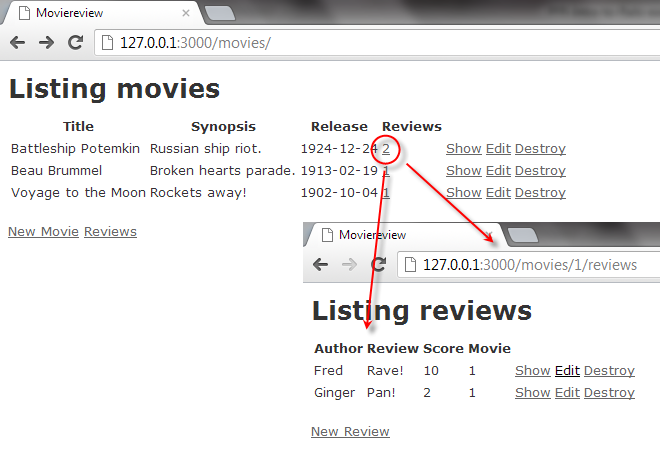
* The new route will use a reviews action in movies\_controller.rb



### How do I create a link to this new route?

* The new reviews\_movie route for each movie id can be displayed and linked using the link\_to command

<%= link\_to movie.reviews.count, reviews\_movie\_path(movie.id) %>



### How do I implement an action to gather and display a filtered review list?

* A new reviews action in movies\_controller.rb can
  + find the movie specified by the id passed in the URI
  + retrieve this movie’s reviews collection for use in the view
  + render these results using the reviews/index view, instead of the default movies index

def reviews

@movie = Movie.find(params[:id])

@reviews = @movie.reviews

respond\_to do |format|

format.html { render 'reviews/index' } # index.html.erb

format.json { render json: @movies }

end

end

### exercise – drilling down from a summary to a detail view

* Complete the exercise in the exercise book

### How can I centrally manage site wide navigation?

* Add a link\_to for each model in a new app/views/\_navigation.html.erb partial

<p>

<%= link\_to 'Movies', movies\_path %>

<%= link\_to 'Reviews', reviews\_path %>

<%= link\_to 'Festivals', festivals\_path %>

</p>

* Render this partial in the overall application layout in app/views/layouts/application.html.erb

...  
<body>

<%= render 'navigation' %>

<%= yield %>

...

### Exercise – adding site wide navigation

* Complete the exercise in the exercise book

## using migrations with databases

### how do i create a migration and what does this mean?

* Migrations are classes enabling you to modify your database in staged, controllable, and reversible ways
* When Rails generates a migration, a class is created and registered in an internal database to track its use

rails generate migration CreateFestivalsMovies

* This command creates db/migrate/[timestamp]\_create\_festivals\_movies.rb

class CreateFestivalsMovies < ActiveRecord::Migration

def up

end

def down

end

end

### What is the basic anatomy of a migration?

* Migrations may expose three basic methods (Rails uses change when generating a model)
  + up – executes when the migration is run
  + down – executes when the migration is rolled back
  + change – executes both when the migration is run and (in reverse) when rolled back
* Databases can be modified in numerous ways within a migration

add\_column add\_index change\_column

change\_table create\_table drop\_table

remove\_column remove\_index rename\_column

* Migrations support many data types, and translate them for the underlying database

:binary :boolean :date

:datetime :decimal :float

:integer :primary\_key :string

:text :time :timestamp

### How might an up/down migration work?

* A migration could create and index a table when put up, then remove and drop them when taken down

class CreateFestivalsMovies < ActiveRecord::Migration

def up

create\_table :festivals\_movies, :id => false do |t|

t.integer :festival\_id, :null => false

t.integer :movie\_id, :null => false

end

# add index to optimize speed

add\_index :festivals\_movies, [:festival\_id, :movie\_id], :unique => true

end

def down

remove\_index :festivals\_movies, :column => [:festival\_id, :movie\_id]

drop\_table :festivals\_movies

end

end

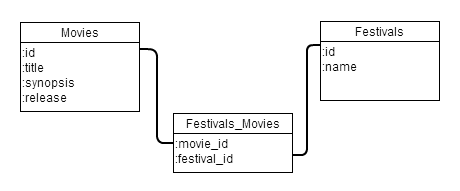
### What are some options for running migrations?

* The Rails rake tool enables several ways for working with migrations
* rake db:migrate – call the up method of all migrations not yet run, in date order
* rake db:migrate VERSION=[timestamp] – run all migrations forward or back to this one
* rake db:migrate:up VERSION=[timestamp] – run the up method of this particular migration
* rake db:migrate:down VERSION=[timestamp] – run the down method of this particular migration
* rake db:rollback – run the down method or reverse the change of the most recent migration
* rake db:rollback STEP=[number] – rolls back this number of migrations from the current one
* rake db:migrate:redo STEP=[number] – roll back then re-run this number of migrations
* rake db:reset – drop the current database and recreate its schema

## Implementing many-to-many model association

### How do I create many to many relationships between tables?

* Each record in a table is uniquely identified by its *primary key* (usually called id)
* This relationship requires creating a *junction* (aka *cross-reference*) table



### How do I create a junction table?

* Supporting a many to many relationship in Rails requires creating a junction table

class CreateFestivalsMovies < ActiveRecord::Migration

def up

create\_table :festivals\_movies, :id => false do |t|

t.integer :festival\_id, :null => false

t.integer :movie\_id, :null => false

end

# add index to optimize speed

add\_index :festivals\_movies, [:festival\_id, :movie\_id], :unique => true

end

def down

remove\_index :festivals\_movies, :column => [:festival\_id, :movie\_id]

drop\_table :festivals\_movies

end

end

### How do I implement this association in my models?

* Rails supports *has and belongs to many* (HABTM) through two similar commands and approaches

has\_and\_belongs\_to\_many :festivals

has\_many :festivals, :through => :festivals\_movies

* The has\_many :through approach supports a junction table with added custom data about the relationship
* The has\_and\_belongs\_to\_many approach expects a plain junction table with no added custom data

### Exercise – enabling many to many associations

* Complete the exercise in the exercise book

## Summary

### What have you learned in this Lesson?

* A database *table* is like a spreadsheet
* A *model* represents a table and its records
* Data in different models may be related one to many or many to many
* Tables have *relations* and models have *associations*
* A *primary key* is a unique identifier for each record in a table
* A *foreign key* is a reference to another record's primary key
* One to many associations in a model are described using has\_many and belongs\_to
* Structured Query Language (SQL) interacts with relational databases
* SELECT queries retrieve data from specifies tables and columns
* The rails dbconsole tool enables local querying of the development database
* The validates\_existence gem may be installed and used to validate one to many associations
* The has\_many command supports a :dependent => :destroy parameter
* Additional member routes can be added to REST-ful resources routing
* Migrations enable the staged creation and rollback of database assets
* Migrations can create many assets supporting many data types
* The up, down, and change methods of a migration are executed by rake
* The rake db command supports many ways to run and roll back migrations
* Many to many relations require a *junction table* with primary key references to each related table
* Many to many associations are supported using has\_and\_belongs\_to\_many or has\_many :through

# 08 Debugging and testing in Rails

Objectives

* Logging
* Using the Ruby debugger
* Understand unit, functional, and integration testing
* Write and use a unit test

## review of previous lesson

### What are some answers to these questions?

1. What is the difference between a relation and an association?
2. What are some examples of one to many and many to many relationships?
3. What is a foreign key in a database?
4. What commands are used to enable one to many associations?
5. How can you ensure a record exists before associating another record with it?
6. What is the purpose of a Rails migration?

### some answers to these questions

1. What is the difference between a relation and an association?  
   *Tables have relations and models have associations*
2. What are some examples of one to many and many to many relationships?  
   *One to many: days of a month, chocolates in a box, reviews for a movie  
   Many to many: movies and theaters, projects and contractors*
3. What is a foreign key in a database?  
   *A column containing the value of the primary key of a related record in another table*
4. What commands are used to enable one to many associations?  
   *has\_many, belongs\_to*
5. How can you ensure a record exists before associating another record with it?  
   *Implement the validates\_existence gem*
6. What is the purpose of a Rails migration?  
   *To make a controllable change to a database*

## DEBUGGING AND Logging

### What is debugging?

* Debugging is the process of using various tools to find and remove code errors
* Various tools may be used during debugging
  + Logging
  + Exceptions
  + Variable output commands and expressions
  + Parameter filters
  + Interactive debugger
* Detailed discussion of Rails debugging can be found here

<http://guides.rubyonrails.org/debugging_rails_applications.html>

### What is a log file and where do I find them?

* Log files provide insight into runtime application performance
* Rails normally writes to up to three log files to [application\_name]/log
  + development.log
  + test.log
  + production.log

### How do I write specific values to a log file?

* You can write custom messages to the log in both code and template files
* Write to log a model, controller, or other code file

logger.info ‘This will appear in the log’

* Write to log in a view or other ERB template

<% logger.info ‘This will also appear in the log’ %>

* Both display as output and write to log in a view or other ERB template

<%= logger.info ‘This will appear in the log and on the page’ %>

### How do I prevent passwords from being written to log files?

* Logging of sensitive information being sent from the browser can be suppressed
* Add filter parameter method to app/controllers/application.rb to block logging specified parameter(s)

filter\_parameter\_logging :password

### Exercise – Writing values to the log file

* Complete the exercise in the exercise book

## Using the rails console and ruby debugger

### What is the Rails console and how do I use it?

* The Rails console is an IRB instance which runs in context of an application and its objects
* Launch Rails console with persistent changes to the application

rails console

* Launch Rails console with changes rolled back when the console session ends

rails console --sandbox

### How do you display variable values from a controller or view?

* Specific variable values can be displayed as output
* Display variable from a controller by raising an exception and converting output to YAML
  + This approach raises an error message which displays the specified variable valuess

raise @variable\_name.to\_yaml

* Display variable from a view template

<%= debug(@variable\_name) %>

### Exercise – displaying variables as debug output

* Complete the exercise in the exercise book

### What is the Ruby debugger and how do I install it?

* A debugger enables line by line execution of code beginning at a pre-defined *break point*
* The Ruby debugger is installed as a Gem, and should be used only on development systems

gem install ruby-debug19

* Once installed, uncomment or add this to the application Gemfile

gem ‘ruby-debug19’, :require => ‘ruby-debug’

### How do I configure code for the debugger?

* Add the debugger command in source code where line by line execution should begin

def reviews

@movie = Movie.find(params[:id])

@reviews = @movie.reviews

**debugger**

respond\_to do |format|

format.html { render 'reviews/index' } # index.html.erb

format.json { render json: @movies }

end

end

### How do I launch and use the debugger?

* Launch the Rails server in debugging mode

rails server --debugger

* Interact with the application, and use these commands from the (rdb:<line\_number>) command line
  + list: display where code execution has halted
  + next (or step): execute the next line of code
  + cont: leave the debugger and continue normal code execution
  + quit: leave the debugger and shut down Rails server

### Exercise – Install, configure, and use the Ruby debugger

* Complete the exercise in the exercise book

## introducing unit, functional, and integration testing

### What is a test database fixture?

* A test database *fixture* is a pre-configured data set for the database to ensure a consistent test environment
* Fixtures
  + are automatically created at test/fixtures when models are generated
  + are written in YAML and may be customized to provide more realistic data
  + need to be customized to support associations
  + need to be manually verified against validation rules before using
* Test databases are rebuilt to a default state using fixtures between tests to provide consistency
* Once manually customized, test fixtures by running rake test

### What three runtime environments are available in Rails?

* Rails supports three runtime modes
  + Development: default mode, no caching so code changes are run immediately
  + Test: caching plus fixtures to provide consistent test environment
  + Production: caching and abbreviated logging to maximize speed
* Mode is selected when Rails server is launched

rails server -e test

# or –e production or –e development

## introducing software testing

### What are some approaches to software testing?

* Unit testing: automated approach to checking fine-grained behavior (e.g., validation works correctly with sample data?)
  + In Rails, unit testing focuses on model validations and associations
  + Generating a model generates stub test code at test/unit
* Functional testing: automated approach to checking action (method) behavior
  + In Rails, functional testing focus on controller actions
  + Generating a controller generates stub test code at test/functional
* Integration testing: automated approach to checking all interactions in a full request cycle
  + In rails integration testing tests full stack: models, controllers, routes, etc.
  + Rails generates stub test code at test/integration

### What tools and approaches are commonly used in Rails software testing?

* Software testing is a complex topic largely beyond the scope of an introductory course
  + <http://guides.rubyonrails.org/testing.html>
* Test Driven Development (TDD): tests designed and built before code is written
* Commonly used Rails testing tools include
  + RSpec: <http://rspec.info>
  + Cucumber: <http://cukes.info>
  + Capybara: <https://github.com/jnicklas/capybara>
  + Factory Girl (test data management): <https://github.com/thoughtbot/factory_girl>

### How do you builD a simple unit test?

* Rails generates a test class derived from ActiveSupport::TestCase which requires test\_helper
* Unit testing relies on the assert statement used in method with a test\_ prefix
* An assert statement
  + expects its argument to be true, and returns a test failure if it is false
  + may be followed by a string describing what has failed if the assertion is false
* Whether a test is checking for success or failure determine if the assertion is true or false

require 'test\_helper'

class ReviewTest < ActiveSupport::TestCase

def test\_score\_too\_high

# test if a too-high score is not valid

review = Review.new

review.score = 15

assert !review.valid?, "Failed due to too-high score"

end

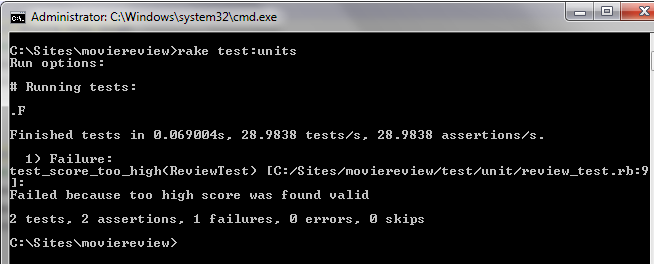
end

### What assertions are supported by Rails?

* Rails supports assertion types beyond assert(expression, message) including
  + assert\_equal(object1, object2, message)
  + assert\_not\_equal(object1, object2, message)
  + assert\_kin\_of(class, object, message)
  + assert\_respond\_to(object, :method, message)
  + many more
* <http://guides.rubyonrails.org/testing.html#assertions-available>

### How do you run the tests you’ve created?

* All defined tests are executed by the rake test:units command



### Exercise – building and running a simple unit test

* Complete the exercise in the exercise book

## Summary

### What have you learned in this Lesson?

* Debugging involves multiple tools: logging, exceptions, variable output, parameter filters, interactive debugger, etc.
* Log data is captured to monitor runtime behavior in development.log, test.log, production.log
* Data can be written to the log using logger.info
* In an ERB template, <%= logger.info 'message' %> will both display and log the message
* Browser parameters can be filtered to prevent logging using filter\_parameter\_logging :param\_name
* The Rails console is a Ruby IRB instance running in context of a Rails application, launched using

rails console

* A sandboxed Rails console rolls its change back when you quit the session, launched using

rails console --sandbox

* In a controller, debug information can be displayed using an uncaught exception, raised using

raise @variable\_name.to\_yaml

* In a view, debug information can be output to the page using

<%= debug(@variable) %>

* The Ruby debugger is installed as a gem

gem install ruby-debug19

* The Ruby debugger is configured in the Gemfile as

gem 'ruby-debug19', :require => 'ruby-debug'

* When rails server -- debugger is launched, code execution stops at debugger breakpoints
* During a debug session, examine code using the list, next, cont, and quit commands
* *Fixtures* are data sets created and re-created during testing to ensure consistency
* Rails supports three runtime modes: development, test, and production

rails server -e test

* Rails *unit testing* focuses on verifying the behavior of models
* Rails *functional testing* focuses on verifying the behaviors of controllers
* Rails *integration testing* focuses on verifying all behavior involved in a specific request
* Numerous test suites are available to support *Test Driven Development* (TDD)
* In a unit test, the assert statement is used to verify if specified behavior succeeds or fails
* Unit tests are executed using the rake test:units command

# 09 Authenticating and managing users

Objectives

* Understand sessions and cookies
* Understand authentication
* Store identities and user data
* Implement basic authentication
* Survey available authentication gems (Devise, Omniauth)

## review of previous lesson

### What are some answers to these questions?

1. How would you write a message to the log file?
2. If a controller has a hash named @employees, how could you display its values in the view?
3. How do you start a debugging session?
4. What is a key difference between the Rails server development, test, and production environments?
5. What command is commonly used within a specific unit test?
6. How do you cause a set of unit tests to be run?

### some answers to these questions

1. How would you write a message to the log file?

*Using logger.info*

1. If a controller has a hash named @employees, how could you display its values in the view?

*Using <%= debug(@employees) %>*

1. How do you start a debugging session?

*Using rails server --debugger*

1. What is a key difference between the Rails server development, test, and production environments?

*No caching is performed in the development environment, the most current code loads for each request*

1. What command is commonly used within a specific unit test?

*assert something\_true, “failure message”*

1. How do you cause a set of unit tests to be run?

*rake test:units*

## understanding sessions and cookies

### What is a session?

* By default, a web server stores no data between a browser’s HTTP requests
* A *session* is a set of browser HTTP requests for which some data is stored between requests
* Session data commonly includes the browser user’s identity (name, login, shopping cart, etc.)
* To work, the web server must uniquely identify each browser’s particular HTTP requests while the session lasts
* Commonly, web servers identify browsers by asking them to store a cookie

### What is a cookie?

* A *cookie* is
  + a small text file, limited by most browsers to 4kb maximum size per cookie
  + offered by a browser by a web server using the Set-Cookie HTTP response header
* Browsers send all cookies accepted from a domain back on each subsequent request to that domain, only
* Cookies are stored in the browser along with the setting domain, timestamp, and intended duration
  + Browser storage is not secure
  + Sensitive data (credit card numbers, etc.) should never be set in a cookie
* Cookies allow web servers to provide a temporary unique ID to a browser, to enable session management

### How does Rails set and access cookies?

* Rails creates a cookies hash for each page request, commonly accessed in the controller
* Values in the cookies hash are passed to and from the browser via HTTP headers
* In working with cookies, distinguish three places data may be stored:
  + Variables: in memory
  + Parameters: sent by browser
  + Cookies: in memory (cookies hash), sent by browser, and returned by web server

def login

@previous\_memory\_name = cookies[:stored\_name]

@memory\_name = params[:input\_name]

cookies[:stored\_name] = @memory\_name

end

### Exercise – setting and accessing cookies

* Complete the exercise in the exercise book

### How can cookies be configured

* Cookies may be directly set as a simple value

cookies[:name] = “Fred”

* Cookies may also be comprised of several values to configure their behavior

cookies[:name] = {:value => “Fred”, :secure => true, :expires => 5.minutes}

* Cookies can be configured with
  + :value – usually a short string, commonly a database key for a user record
  + :domain – domain serving the cookie, or a subdomain within it; only requests to these will receive it
  + :path – the path of the file serving the cookie, or part of the path; only requests to these will receive it
  + :expires – how long the cookie will remain in and be sent from the browser
  + :secure – if true, cookie will only be sent over HTTPS connections
  + :http\_only – if true, will be sent with HTTP/S requests, but not be available to Javascript on the page

## Using session management in Rails

### What is session management?

* While values can be persisted across requests using cookies, doing can be cumbersome and insecure
* Rails session management
  + tracks the user during the session with a single cookie
  + allows you to store any number of values in memory, keyed to that user’s session cookie
* The session object
  + is available to the controller and view
  + can store any number of values in memory for duration of the current user’s session

### Exercise – using session data

* Complete the exercise in the exercise book

## Understanding authentication

### what is authentication?

* *Authentication* – verifying that someone or something is who they claim to be
* *Authorization* – allowing an authenticated user the privilege(s) needed to use a system

### What is involved in a simple authentication process?

1. User requests a page secured by a filter testing for a session variable
2. If no session variable exists, user is directed to a login screen
3. User inputs credentials (e.g., name and password)
4. System checks whether the credentials are known and valid
5. System recalls user data associated with these credentials from the model (database)
6. System initiates a new user session variable with the recalled information
7. User is redirected to an initial application screen
8. User now passes the security filter for each page due to the session variable

## Implementing rails authentication

### how does rails 3 implement simple authentication?

* There are numerous ways to implement authentication
* Rails 3 provides a new, streamlined mechanism using ActiveModel has\_secure\_password
* The BCrypt gem is used to provide encryption support for passwords
* BCrypt is pre-installed with Rails 3, but must be configured in the Gemfile

# To use ActiveModel has\_secure\_password

gem 'bcrypt-ruby', '~> 3.0.0'

### Exercise – implementing rails 3 authentication

* Complete the exercise in the exercise book

## Surveying available authentication gems (Devise, Omniauth)

### What other authentication frameworks are available?

* Numerous Rails authentication frameworks are available which may extend simple authentication
  + Configurable user data management
  + Support for various back end authentication sources (OpenID, Facebook, etc.)
* OmniAuth: <http://www.omniauth.org/>
* Devise: <https://github.com/plataformatec/devise>

### What have you learned in this Lesson?

* By default, no data is stored between HTTP requests from browsers
* A session is a group of requests among which data is shared
* A set of HTTP requests must share a common ID for the server to share data among them
* Session IDs are commonly managed by setting and checking for cookies
* A cookie is a text string up to 4kb (on most browsers) usually used for unique IDs
* Cookies are not secure, so should be used only for temporarily unique IDs
* Rails sets and retrieves cookie values using the automatically created cookies[:name] hash
* Cookies are set and retrieved by a specific domain or sub-domain
* Cookies expire after a defined timeout period and may or may not be stored between sessions
* The session[:name] hash enables in-memory storage of variables between HTTP requests
* The cookies needed to maintain the session are managed by Rails
* Authentication means to verify a user's identity
* Authorization means to provide use privileges to a user
* Rails 3 provides a has\_secure\_password authentication mechanism
* Each page request can be filtered to require a user session

# 10 Collaborative Lab / Final Review

Objectives

* Collaboratively complete the final lab application
* Review the final lab code
* Review the course

## review of previous lesson

### What are some answers to these questions?

1. What is a user session?
2. How does Rails create and manage a session?
3. What is a cookie and how should they be used?
4. What is the difference between authentication and authorization?
5. How can a Rails view be made secure by requiring authentication?

### some answers to these questions

1. What is a user session?

*A group of HTTP requests with a common ID enabling data sharing among the requests*

1. How does Rails create and manage a session?

*By dropping a unique cookie in a browser, then checking for it on each subsequent request until the session expires*

1. What is a cookie and how should they be used?

*Text files up to 4kb stored on a browser by an HTTP server, used to uniquely identify user sessions*

1. What is the difference between authentication and authorization?

*Authentication is who you are, authorization is what you get to do*

1. How can a Rails view be made secure by requiring authentication?

*By adding a before\_filter to a controller which tests for a valid user session*

# collaborative lab completion

* Work collaboratively with other students to complete the final lab application
* Review the final lab application with the instructor