## Interview questions and answers

## 

## MetaProgramming?

## class CarModel   FEATURES = ["engine", "wheel", "airbag", "alarm", "stereo"]   FEATURES.each do |feature|     define\_method("#{feature}\_info=") do |info|       instance\_variable\_set("@#{feature}\_info", info)     end     define\_method("#{feature}\_info") do       instance\_variable\_get("@#{feature}\_info")     end     define\_method "feature\_price=" do |price|       instance\_variable\_set("@#{feature}\_price", price)     end     define\_method("#{feature}\_price") do       instance\_variable\_get("@#{feature}\_price")     end   end end

## 

## 

## How do they relate?

### So, how does this all fit together?

### Out of these pieces, a web request will hit your web server first. If the request is something Rails can handle, the web server will do some processing on the request, and hand it off to the app server. The app server uses Rack to talk to your Rails app. When your app is done with the request, your Rails app sends the response back through the app server and the web server to the person using your app.

### More specifically, Nginx might pass a request to Unicorn. Unicorn gives the request to Rack, which gives it to the Rails router, which gives it to the right controller. Then, your response just goes back through the other way.

# difference between collection route and member route in ruby on rails?

# A member route will require an ID, because it acts on a *member*. A collection route doesn't because it acts on a collection of objects. Preview is an example of a member route, because it acts on (and displays) a single object. Search is an example of a collection route, because it acts on (and displays) a collection of objects.

### Resource routing allows you to quickly declare all of the common routes for a given resourceful controller. Instead of declaring separate routes for your index, show, new, edit, create, update and destroy actions, a resourceful route declares them in a single line of code:

### resources :photos

### Sometimes, you have a resource that clients always look up without referencing an ID. A common example, /profile always shows the profile of the currently logged in user. In this case, you can use a singular resource to map /profile (rather than /profile/:id) to the show action.

### resource :profile

### It's common to have resources that are logically children of other resources:

### resources :magazines do   resources :ads end

### You may wish to organize groups of controllers under a namespace. Most commonly, you might group a number of administrative controllers under an admin namespace. You would place these controllers under the app/controllers/admin directory, and you can group them together in your router:

### namespace "admin" do   resources :posts, :comments end

### By default the :id parameter doesn't accept dots. If you need to use dots as part of the :id parameter add a constraint which overrides this restriction, e.g:

### resources :articles, id: /[^\/]+/

### This allows any character other than a slash as part of your :id.

### What is the difference between save and save! in Rails?

Save! performs all validations and callbacks. If any validation returns false, save! throws an error and cancels the save.

Save does not throw any error in the case above, but cancels the save. Also, the methods for validations can be bypassed.

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries.

Rack

Rack provides a modular and adaptable interface for developing web applications in [Ruby](https://en.wikipedia.org/wiki/Ruby_programming_language). By wrapping [HTTP requests](https://en.wikipedia.org/wiki/Http_request#Request_message) and responses it unifies the API for web servers, web frameworks, and software in between (called [middleware](https://en.wikipedia.org/wiki/Middleware)) into a single method call.

Rack

provides a minimal interface between web servers that support Ruby and Ruby frameworks.

Rake

Rake is a build language, similar in purpose to make and ant. Like make and ant it's a Domain Specific Language, unlike those two it's an internal DSL programmed in the Ruby language. In this article I introduce rake and describe some interesting things that came out of my use of rake to build this web site: dependency models, synthesized tasks, custom build routines and debugging the build script.

RESTful API

A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data.

It means when a RESTful API is called, the server will transfer to the client a representation of the state of the requested resource.

1) Explain what is Ruby on Rails?

* Ruby: It is an object oriented programming language inspired by PERL and PYTHON.
* Ruby is a pure object-oriented language and everything appears to Ruby as an object. Every value in Ruby is an object, even the most primitive things: strings, numbers and even true and false. Even a class itself is an *object* that is an instance of the *Class* class. This chapter will take you through all the major functionalities related to Object Oriented Ruby.
* A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package. The data and methods within a class are called members of the class.
* Rails: It is a framework used for building web application

2) **Explain what is class libraries in Ruby?**

Class libraries in Ruby consist of a variety of domains, such as data types, thread programming, various domains, etc.

3) **Mention what is the naming convention in Rails?**

* Variables: For declaring Variables, all letters are lowercase, and words are separated by underscores
* Class and Module: Modules and Classes uses MixedCase and have no underscore; each word starts with a uppercase letter
* Database Table: The database table name should have lowercase letters and underscore between words, and all table names should be in the plural form for example invoice\_items
* Model: It is represented by unbroken Mixed Case and always have singular with the table name
* Controller: Controller class names are represented in plural form, such that OrdersController would be the controller for the order table.

4) **Explain what is “Yield” in Ruby on Rails?**

Yield is A Ruby method that receives a code block invokes it by calling it with the “Yield”.

5) **Explain what is ORM (Object-Relationship-Model) in Rails?**

ORM or Object Relationship Model in Rails indicate that your classes are mapped to the table in the database, and objects are directly mapped to the rows in the table.

6) **Mention what the difference is between false and nil in Ruby?**

In Ruby False indicates a Boolean datatype, while Nil is not a data type, it have an object\_id 4.

7) **Mention what are the positive aspects of Rails?**

Rails provides many features like

* Meta-programming: Rails uses code generation but for heavy lifting it relies on meta-programming. Ruby is considered as one of the best language for Meta-programming.
* Active Record: It saves object to the database through Activerecord Framework. The Rails version of Active Record identifies the column in a schema and automatically binds them to your domain objects using metaprogramming
* Scaffolding: Rails have an ability to create scaffolding or temporary code automatically
* Convention over configuration: Unlike other development framework, Rails does not require much configuration, if you follow the naming convention carefully
* Three environments: Rails comes with three default environment testing, development, and production.
* Built-in-testing: It supports code called harness and fixtures that make test cases to write and execute.

**8) Explain what is the role of sub-directory app/controllers and app/helpers?**

* App/controllers: A web request from the user is handled by the Controller. The controller sub-directory is where Rails looks to find controller classes.
* App/helpers: The helper’s sub-directory holds any helper classes used to assist the view, model and controller classes.

**9) Mention what is the difference between String and Symbol?**

They both act in the same way only they differ in their behaviors which are opposite to each other. The difference lies in the object\_id, memory and process tune when they are used together. Symbol belongs to the category of immutable objects whereas Strings are considered as mutable objects.

**10) Explain how Symbol is different from variables?**

Symbol is different from variables in following aspects

* It is more like a string than variable
* In Ruby string is mutable but a Symbol is immutable
* Only one copy of the symbol requires to be created
* Symbols are often used as the corresponding to enums in Ruby

**When to Use Symbols ??**

One of the most common uses for symbols is to represent method & instance variable names.

Array of Symbols

symbols = %i(a b c)

Strings = %w(a b c)

symbols = %i(asymbols = %i(a b c)b c)

**11) Explain what is Rails Active Record in Ruby on Rails?**

Rails activerecord is the Object/Relational Mapping (ORM) layer supplied with Rails. It follows the standard ORM model as

* Table map to classes
* Rows map to object
* Columns map to object attributes

**12) Explain how Rails implements Ajax?**

Ajax powered web page retrieves the web page from the server which is new or changed unlike other web-page where you have to refresh the page to get the latest information.

Rails triggers an Ajax Operation in following ways

* Some trigger fires: The trigger could be a user clicking on a link or button, the users inducing changes to the data in the field or on a form
* Web client calls the server: A Java-script method, XMLHttpRequest, sends data linked with the trigger to an action handler on the server. The data might be the ID of a checkbox, the whole form or the text in the entry field
* Server does process: The server side action handler does something with the data and retrieves an HTML fragment to the web client
* Client receives the response: The client side JavaScript, which Rails generates automatically, receives the HTML fragment and uses it to update a particular part of the current

**13) Mention how you can create a controller for subject?**

To create a controller for subject you can use the following command

C:\ruby\library> ruby script/generate controller subject

**14) Mention what is Rails Migration?**

Rails Migration enables Ruby to make changes to the database schema, making it possible to use a version control system to leave things synchronized with the actual code.

**15) List out what can Rails Migration do?**

Rails Migration can do following things

* Create table
* Drop table
* Rename table
* Add column
* Rename column
* Change column
* Remove column and so on

**16) Mention what is the command to create a migration?**

To create migration command includes

C:\ruby\application>ruby script/generate migration table\_name

**17) Explain when self.up and self.down method is used?**

When migrating to a new version, self.up method is used while self.down method is used to roll back my changes if needed.

**18) Mention what is the role of Rails Controller?**

The Rails controller is the logical center of the application. It facilitates the interaction between the users, views, and the model. It also performs other activities like

* It is capable of routing external requests to internal actions. It handles URL extremely well
* It regulates helper modules, which extend the capabilities of the view templates without bulking of their code
* It regulates sessions; that gives users the impression of an ongoing interaction with our applications

**19) Mention what is the difference between Active support’s “HashWithIndifferent” and Ruby’s “Hash” ?**

The Hash class in Ruby’s core library returns value by using a standard “= =” comparison on the keys. It means that the value stored for a symbol key cannot be retrieved using the equivalent string. While the HashWithIndifferentAccess treats Symbol keys and String keys as equivalent.

**20) Explain what is Cross-Site Request Forgery (CSRF) and how Rails is protected against it?**

CSRF is a form of attack where hacker submits a page request on your behalf to a different website, causing damage or revealing your sensitive data. To protect from CSRF attacks, you have to add “protect\_from\_forgery” to your ApplicationController. This will cause Rails to require a CSRF token to process the request. CSRF token is given as a hidden field in every form created using Rails form builders.

**21) Explain what is Mixin in Rails?**

Mixin in Ruby offers an alternative to multiple inheritances, using mixin modules can be imported inside other class.

22) Explain how you define Instance Variable, Global Variable and Class Variable in Ruby?

* Ruby Instance variable begins with — @
* Ruby Class variables begin with — @@
* Ruby Global variables begin with — $

**23) Explain how you can run Rails application without creating databases?**

You can execute your application by uncommenting the line in environment.rb

path=> rootpath conf/environment.rb

config.frameworks = [ action\_web\_service, :action\_mailer, :active\_record]

**24) Mention what is the difference between the Observers and Callbacks in Ruby on Rails?**

* Rails Observers: Observers is as as Callback, but it is used when method is not directly associated to object lifecycle. Also, the observer lives longer, and it can be detached or attached at any time. For example, displaying values from a model in the UI and updating model from user input.
* Rails Callback: Callbacks are methods, which can be called at certain moments of an object’s life cycle for example it can be called when an object is validated, created, updated, deleted, A call back is short lived. For example, running a thread and giving a call-back that is called when thread terminates.

**25) Explain what is rake in Rails?**

Rake is a Ruby Make; it is a Ruby utility that substitutes the Unix utility ‘make’, and uses a ‘Rakefile’ and ‘.rake files’ to build up a list of tasks. In Rails, Rake is used for normal administration tasks like migrating the database through scripts, loading a schema into the database, etc.

**26) Explain how you can list all routes for an application?**

To list out all routes for an application you can write rake routes in the terminal.

**27) Explain what is sweeper in Rails?**

Sweepers are responsible for expiring or terminating caches when model object changes.

**28) Mention the log that has to be seen to report errors in Ruby Rails?**

Rails will report errors from Apache in the log/Apache.log and errors from the Ruby code in log/development.log.

**30) Mention what is the function of garbage collection in Ruby on Rails?**

The functions of garbage collection in Ruby on Rails includes

* It enables the removal of the pointer values which is left behind when the execution of the program ends
* It frees the programmer from tracking the object that is being created dynamically on runtime
* It gives the advantage of removing the inaccessible objects from the memory, and allows other processes to use the memory

**31) Mention what is the difference between redirect and render in Ruby on Rails?**

* Redirect is a method that is used to issue the error message in case the page is not found to the browser. It tells the browser to process and issue a new request.
* Render is a method used to make the content. Render only works when the controller is being set up properly with the variables that require to be rendered.

**32) Mention what is the purpose of RJs in Rails?**

RJs is a template that produces JavaScript which is run in an eval block by the browser in response to an AJAX request. It is sometimes used to define the JavaScript, Prototype and helpers provided by Rails.

**33) Explain what is Polymorphic Association in Ruby on Rails?**

Polymorphic Association allows an ActiveRecord object to be connected with Multiple ActiveRecord objects. A perfect example of Polymorphic Association is a social site where users can comment on anywhere whether it is a videos, photos, link, status updates etc. It would be not feasible if you have to create an individual comment like photos\_comments, videos\_comment and so on.

**34) Mention what are the limits of Ruby on Rails?**

Ruby on Rails has been designed for creating a CRUD web application using MVC. This might make Rails not useful for other programmers. Some of the features that Rails does not support include

* Foreign key in databases
* Linking to multiple data-base at once
* Soap web services
* Connection to multiple data-base servers at once

**35) Mention what is the difference between calling super() and super call?**

* super(): A call to super() invokes the parent method without any arguments, as presumably expected. As always, being explicit in your code is a good thing.
* super call: A call to super invokes the parent method with the same arguments that were passed to the child method. An error will therefore occur if the arguments passed to the child method don’t match what the parent is expecting.

**36) Explain about Dig, Float and Max?**

* Float class is used whenever the function changes constantly.
* Dig is used whenever you want to represent a float in decimal digits.
* Max is used whenever there is a huge need of Float.

**37) Explain how can we define Ruby regular expressions?**

Ruby regular expression is a special sequence of characters that helps you match or find other strings. A regular expression literal is a pattern between arbitrary delimiters or slashes followed by %r.

**38) Explain what is the defined operator?**

Define operator states whether a passed expression is defined or not. If the expression is defined, it returns the description string and if it is not defined it returns a null value.

39) List out the few features of Ruby?

* Free format – You can start writing from program from any line and column
* Case sensitive – The uppercase and lowercase letters are distinct
* Comments – Anything followed by an unquoted #, to the end of the line on which it appears, is ignored by the interpreter
* Statement delimiters- Multiple statements on one line must be separated by semicolons, but they are not required at the end of a line.

**40) Mention the types of variables available in Ruby Class?**

Types of variables available in Ruby Class are,

* Local Variables
* Global Variables
* Class Variables
* Instance Variables

**41) Explain how can you declare a block in Ruby?**

In Ruby, the code in the block is always enclosed within braces ({}).  You can invoke a block by using “yield statement”.

**42) Explain what is the difference between put and putc statement?**

Unlike the puts statement, which outputs the entire string onto the screen. The Putc statement can be used to output one character at a time.

**43) Explain what is a class library in Ruby?**

Ruby class libraries consist of a variety of domains, such as thread programming, data types, various domains, etc. These classes give flexible capabilities at a high level of abstraction, giving you the ability to create powerful Ruby scripts useful in a variety of problem domains. The following domains which have relevant class libraries are,

* GUI programming
* Network programming
* CGI Programming
* Text processing

**44) In Ruby, it explains about the defined operator?**

The defined operator tells whether a passed expression is defined or not.  If the expression is not defined, it gives null, and if the expression is defined it returns the description string.

**45) Mention what is the difference in scope for these two variables: @@name and @name?**

The difference in scope for these two variables is that:

* @@name is a class variable
* @name is an instance variable

**46) Mention what is the syntax for Ruby collect Iterator?**

The syntax for Ruby collect Iterator collection = collection.collect.

**47) In Ruby code, often it is observed that coder uses a shorthand form of using an expression like array.map(&:method\_name) instead of array.map { |element| element.method\_name }. How this trick actually works?**

When a parameter is passed with “&” in front of it. Ruby will call to\_proc on it in an attempt to make it usable as a block.  So, symbol to\_Proc will invoke the method of the corresponding name on whatever is passed to it. Thus helping our shorthand trick to work.

**48) Explain what is Interpolation in Ruby?**

Ruby Interpolation is the process of inserting a string into a literal.  By placing a Hash (#) within {} open and close brackets, one can interpolate a string into the literal.

**49) Mention what is the Notation used for denoting class variables in Ruby?**

In Ruby,

* A constant should begin with an uppercase letter, and it should not be defined inside a method
* A local must begin with the \_ underscore sign or a lowercase letter
* A global variable should begin with the $ sign. An uninitialized global has the value of “nil” and it should raise a warning. It can be referred anywhere in the program.
* A class variable should begin with double @@ and have to be first initialized before being used in a method definition

**50) Mention what is the difference between Procs and Blocks?**

The difference between Procs and Blocks,

* Block is just the part of the syntax of a method while proc has the characteristics of a block
* Procs are objects, blocks are not
* At most one block can appear in an argument list
* Only block is not able to be stored into a variable while Proc can

**51) Mention what is the difference between a single quote and double quote?**

A single-quoted strings don’t process ASCII escape codes, and they don’t do string interpolation.

**51) Mention what is the difference between a gem and a plugin in Ruby?**

* Gem: A gem is a just ruby code. It is installed on a machine, and it’s available for all ruby applications running on that machine.
* Plugin: Plugin is also ruby code, but it is installed in the application folder and only available for that specific application.

**53) Mention what is the difference between extend and include?**

* include : mixes in specified module methods as instance methods in the target class
* extend : mixes in specified module methods as class methods in the target class

**54) What is gemset?**

A gemset is just a container you can use to keep gems separate from each other.

The Big Idea: creating a gemset per project allows you to change gems (and gem versions) for one project without breaking all your other projects. Each project need only worry about its own gems. This is a Good Idea, and the wait time for installing large gems like Rails is usually worth it.

*That said*, if you're going to use the same version of Rails across all your projects and want to save time, you can install rails (and maybe rake as well) in the ['global' gemset](http://beginrescueend.com/gemsets/global/) - these gems are available in all gemsets *for that version of ruby*.

**55) How and when would you declare a Global Variable?**

* Global variables are declared with the ‘$’ symbol and can be declared and used anywhere within your program. You should use them sparingly to never.

**56) How would you create getter and setter methods in Ruby?**

* Setter and getter methods in Ruby are generated with the attr\_accessor method. attr\_accessor is used to generate instance variables for data that’s not stored in your database column.
* You can also take the long route and create them manually.

**57) Ruby - Describe the environment variables present in Ruby**

RUBYOPT, RUBYLIB, RUBYPATH, RUBYSHELL, RUBYLIB\_PREFIX......

58)Ruby - Interpolation is a very important process in Ruby

Interpolation is a very important process in Ruby, comment.

Inserting a string into a literal is called as interpolation. Interpolation is a very important process in Ruby. Interpolation can be done by using only one way by embedding # within {}. A new name is referred to the copy of the original name.

**59)Ruby Overloading Methods**

1. # The Rectangle constructor accepts arguments in either
2. # of the following forms:
3. #   Rectangle.new([x\_top, y\_left], length, width)
4. #   Rectangle.new([x\_top, y\_left], [x\_bottom, y\_right])
5. class Rectangle
6. def initialize(\*args)
7. if args.size < 2  || args.size > 3
8. # modify this to raise exception, later
9. puts 'This method takes either 2 or 3 arguments'
10. else
11. if args.size == 2
12. puts 'Two arguments'
13. else
14. puts 'Three arguments'
15. end
16. end
17. end
18. end
19. Rectangle.new([10, 23], 4, 10)
20. Rectangle.new([10, 23], [14, 13])

**What is a module?**

A module is like a class. Except that it can’t be instantiated or subclassed.

In OOP paradigm you would store methods & variables that represent variables in a single class. Say you want to create an Employee representation then the employee’s name, age, salary, etc. would all go inside a Employee class, in a file called Employee.rb

Any methods that act on those variables would also go inside that class.

You can achieve the same effect by putting all the variables and methods inside a Employee module:

module Employee  
  ..variables.  
  ...methods  
end

The main difference between the class & module is that a module cannot be instantiated or subclassed.

Module are better suited for library type classes such as Math library, etc.

**61)How can you call the base class method from inside of its overridden method?**

If you are inside the overridden method in the derived class then a simple call to super will call the right method in the base class

class Parent  
  def try\_this()  
      puts "parent"  
  end  
end  
  
class Child < Parent  
  def try\_this()  
      super()  
      puts "child"  
  end  
end  
  
ch = Child.new  
ch.try\_this()

This generates the output

parent  
child

Now if you just want to call the base class without calling the derived class then the best way to do that is to simply assign an alias to the parent method like this:

class Parent  
  def knox  
    puts 'parent'  
  end  
end  
  
class Child < Parent  
  alias\_method :parent\_knox, :knox  
  def knox  
    puts 'child'  
  end  
end  
  
ch = Child.new  
ch.parent\_knox  
ch.knox

This allows you to call the base class method with the alias parent\_knox and the derived class method knox can be called directly.

parent  
child

**62)Does Ruby support constructors? How are they declared?**

Constructors are supported in Ruby. They are declared as the method *initialize,* shown below. The initialize method gets called automatically when Album.new is called.

class Album  
  def initialize(name, artist, duration)  
    @name     = name  
    @artist   = artist  
    @duration = duration  
  end  
End

**63)What is the purpose of environment.rb and application.rb file?**

There are two files where variables and configuration settings are stored.

- config/environment.rb : Environment settings go here

- config/application.rb : Application level global settings go here

config.time\_zone = 'Central Time (US & Canada)'

config.i18n.default\_locale = :de

config.filter\_parameters += [:password] # ensures that passwords are not logged

The same file is also used for configuring various environment settings such as:

config.action\_mailer.smtp\_settings # various email settings go here

**What is the purpose of config/environments/development.rb file?**

You would specify various config settings the development environment in this file.

 config.action\_controller.perform\_caching = false # to enable caching

This is because you typically do not want to enable caching in the development environment.

The same config setting in the production environment would be equal to true.

**64) What’s Rack?**

Rack is a Ruby package that provides an easy-to-use interface to the Ruby [Net::HTTP](http://ruby-doc.org/stdlib-2.2.0/libdoc/net/http/rdoc/Net/HTTP.html) library.

**65) What is Base class for all ruby classes ?**

The BasicObject class is the parent class of all classes in Ruby. Its methods are therefore available to all objects unless explicitly overridden. Prior to Ruby 1.9, Object class was the root of the class hierarchy. The new class BasicObject serves that purpose, and Object is a subclass of BasicObject. BasicObject is a very simple class, with almost no methods of its own. When you create a class in Ruby, you extend Object unless you explicitly specify the super-class, and most programmers will never need to use or extend BasicObject.

**66) What is the purpose of  “ ! and ? ” ?**

It's "just sugarcoating" for readability, but they do have common meanings:

* Methods ending in ! perform some permanent or potentially dangerous change; for example:
  + Enumerable#sort returns a sorted version of the object while Enumerable#sort! sorts it in place.
  + In Rails, ActiveRecord::Base#save returns false if saving failed, while ActiveRecord::Base#save! raises an exception.
  + Kernel::exit causes a script to exit, while Kernel::exit! does so immediately, bypassing any exit handlers.
* Methods ending in ? return a boolean, which makes the code flow even more intuitively like a sentence — if number.zero? reads like "if the number is zero", but if number.zero just looks weird.

In your example, name.reverse evaluates to a reversed string, but only after the name.reverse! line does the name variable actually *contain* the reversed name. name.is\_binary\_data? looks like "is name binary data?".

### 67)What is the Asset Pipeline?

The asset pipeline provides a framework to concatenate and minify or compress JavaScript and CSS assets. It also adds the ability to write these assets in other languages and pre-processors such as CoffeeScript, Sass and ERB. It allows assets in your application to be automatically combined with assets from other gems. For example, jquery-rails includes a copy of jquery.js and enables AJAX features in Rails.

The first feature of the pipeline is to concatenate assets, which can reduce the number of requests that a browser makes to render a web page. Web browsers are limited in the number of requests that they can make in parallel, so fewer requests can mean faster loading for your application.

Sprockets concatenates all JavaScript files into one master .js file and all CSS files into one master .css file. As you'll learn later in this guide, you can customize this strategy to group files any way you like. In production, Rails inserts an MD5 fingerprint into each filename so that the file is cached by the web browser. You can invalidate the cache by altering this fingerprint, which happens automatically whenever you change the file contents.

The second feature of the asset pipeline is asset minification or compression. For CSS files, this is done by removing whitespace and comments. For JavaScript, more complex processes can be applied. You can choose from a set of built in options or specify your own.

The third feature of the asset pipeline is it allows coding assets via a higher-level language, with precompilation down to the actual assets. Supported languages include Sass for CSS, CoffeeScript for JavaScript, and ERB for both by default.

**68)diff b/w ruby 1.9.3 and 2.2.0?**

As of today, all support for Ruby 1.9.3 has ended. Bug and security fixes from more recent Ruby versions will no longer be backported to 1.9.3.

We highly recommend that you upgrade to Ruby 2.0.0 or above as soon as possible. Please contact us if you’d like to continue maintaining the 1.9.3 branch for some reason you can’t upgrade.

There are several benefits when using more recent versions of Ruby:

* Security (e.g. old versions might have unfixed issues) and maintenance
* Compatibility with gems you want to use (e.g. current versions might use kwargs that were introduced in Ruby 2.0)
* Performance (newer versions are usually faster, see [source](http://www.isrubyfastyet.com/))
* Documentation and learning (it might be harder to find solutions to problems or good blog articles for old versions)

That said: Try to keep up-to-date and use the most recent version that is supported by your environment and dependencies.

**69 )Diff in rails 4?**

##Notable Changes in Rails 4

###Ruby 1.9.3 Minimum

###No more vendor/plugins

## 70 )What is Bundler?

Bundler is a program for managing [gem](https://www.thoughtco.com/ruby-programming-4133469) dependencies in your Ruby projects. With Bundler you can specify which gems your program needs, what versions they should be at, it can help you install them, load them at runtime and distribute them with your software. Basically, it takes all of the guesswork out of installing the gems needed to run your Ruby projects.

**71 )RubyGems?**

The RubyGems software allows you to easily download, install, and use ruby software packages on your system. The software package is called a “gem” and contains a package Ruby application or library. ... Some gems provide command line utilities to help automate tasks and speed up your work.

#### 72 )What is a Proc?

Everyone usually confuses procs with blocks, but the strongest rubyist can grok the true meaning of the question.

Essentially, Procs are anonymous methods (or nameless functions) containing code. They can be placed inside a variable and passed around like any other object or scalar value. They are created by Proc.new, lambda, and blocks (invoked by the yield keyword).

#### 73 )What does self mean?

self *always* refers to the current object. But this question is more difficult than it seems because Classes are also objects in ruby. (Kudos to Stephen)

class WhatIsSelf  
  def test  
    puts "At the instance level, self is #{self}"  
  end  
  
  def self.test  
    puts "At the class level, self is #{self}"  
  end  
end  
  
WhatIsSelf.test   
#=> At the class level, self is WhatIsSelf  
  
WhatIsSelf.new.test   
#=> At the instance level, self is #<WhatIsSelf:0x28190>

**74) What is a module? What is its purpose? How do we use them with our classes? Create a module for the class you created in exercise 1 and include it properly.**

A module allows us to group reusable code into one place. We use modules in our classes by using the include reserved word, followed by the module name. Modules are also used as a namespace.

module Study  
end  
  
class MyClass  
  include Study  
end  
  
my\_obj = MyClass.new

### Lambda

Despite the fancy name, a lambda is just a function... peculiarly... without a name. They're anonymous, little functional spies sneaking into the rest of your code.

Lambdas in Ruby are also objects, just like everything else! The last expression of a lambda is its return value, just like regular functions. As boring and familiar as that all sounds, it gives us a lot of power.

As objects, lambdas have methods and can be assigned to variables. Let's try it!

l = lambda { "Do or do not" }  
puts l.call

l = lambda do |string|  
  if string == "try"  
    return "There's no such thing"   
  else  
    return "Do or do not."  
  end  
end  
puts l.call("try") # Feel free to experiment with this

Scope(name, scope\_options = {}) *public*

Adds a class method for retrieving and querying objects. A [scope](https://apidock.com/rails/ActiveRecord/NamedScope/ClassMethods/) represents a narrowing of a database query, such as where(:color => :red).select('shirts.\*').includes(:washing\_instructions).

**ActiveRecord Rails 3 scope vs class method**There was more of a difference in Rails 2.x, since named\_scopes did not execute your queries (so you could chain them), whereas class methods generally did execute the queries (so you could not chain them), unless you manually wrapped your query in a scoped(...) call.

In Rails 3, everything returns an ActiveRecord::Relation until you need the actual results, so scopes can be chained against class methods and vice versa (as long as the class methods return ActiveRecord::Relation objects, not some other object type (like a count)).

Generally, I use scope entries for simple one-liners to filter down my result set. However, if I'm doing anything complicated in a "scope" which may require detailed logic, lambdas, multiple lines, etc., I prefer to use a class method. And as you caught, if I need to return counts or anything like that, I use a class method.

# How does load differ from require in Ruby?

require searches for the library in all the defined search paths and also appends .rb or .so to the file name you enter. It also makes sure that a library is only included once. So if your application requires library A and B and library B requries library A too A would be loaded only once.

With load you need to add the full name of the library and it gets loaded every time you call load - even if it already is in memory.

Rails provides four different ways to load association data. In this blog we are going to look at each of those.

# Preload, Eagerload, Includes and Joins

## Preload

Preload loads the association data in a separate query.

User.preload(:posts).to\_a  
  
# =>  
SELECT "users".\* FROM "users"  
SELECT "posts".\* FROM "posts"  WHERE "posts"."user\_id" IN (1)

This is how includes loads data in the default case.

Since preload always generates two sql we can’t use posts table in where condition. Following query will result in an error.

User.preload(:posts).where("posts.desc='ruby is awesome'")  
  
# =>  
SQLite3::SQLException: no such column: posts.desc:  
SELECT "users".\* FROM "users"  WHERE (posts.desc='ruby is awesome')

With preload where clauses can be applied.

User.preload(:posts).where("users.name='Neeraj'")  
  
# =>  
SELECT "users".\* FROM "users"  WHERE (users.name='Neeraj')  
SELECT "posts".\* FROM "posts"  WHERE "posts"."user\_id" IN (3)

### Includes

Includes loads the association data in a separate query just like preload.

However it is smarter than preload. Above we saw that preload failed for query User.preload(:posts).where("posts.desc='ruby is awesome'"). Let’s try same with includes.

User.includes(:posts).where('posts.desc = "ruby is awesome"').to\_a  
  
# =>  
SELECT "users"."id" AS t0\_r0, "users"."name" AS t0\_r1, "posts"."id" AS t1\_r0,  
      "posts"."title" AS t1\_r1,  
      "posts"."user\_id" AS t1\_r2, "posts"."desc" AS t1\_r3  
FROM "users" LEFT OUTER JOIN "posts" ON "posts"."user\_id" = "users"."id"  
WHERE (posts.desc = "ruby is awesome")

As you can see includes switches from using two separate queries to creating a single LEFT OUTER JOIN to get the data. And it also applied the supplied condition.

So includes changes from two queries to a single query in some cases. By default for a simple case it will use two queries. Let’s say that for some reason you want to force a simple includes case to use a single query instead of two. Use references to achieve that.

User.includes(:posts).references(:posts).to\_a  
  
# =>  
SELECT "users"."id" AS t0\_r0, "users"."name" AS t0\_r1, "posts"."id" AS t1\_r0,  
      "posts"."title" AS t1\_r1,  
      "posts"."user\_id" AS t1\_r2, "posts"."desc" AS t1\_r3  
FROM "users" LEFT OUTER JOIN "posts" ON "posts"."user\_id" = "users"."id"

In the above case a single query was done.

## Eager load

eager loading loads all association in a single query using LEFT OUTER JOIN.

User.eager\_load(:posts).to\_a  
  
# =>  
SELECT "users"."id" AS t0\_r0, "users"."name" AS t0\_r1, "posts"."id" AS t1\_r0,  
      "posts"."title" AS t1\_r1, "posts"."user\_id" AS t1\_r2, "posts"."desc" AS t1\_r3  
FROM "users" LEFT OUTER JOIN "posts" ON "posts"."user\_id" = "users"."id"

This is exactly what includes does when it is forced to make a single query when where or order clause is using an attribute from posts table.

## Joins

Joins brings association data using inner join.

User.joins(:posts)  
  
# =>  
SELECT "users".\* FROM "users" INNER JOIN "posts" ON "posts"."user\_id" = "users"."id"

In the above case no posts data is selected. Above query can also produce duplicate result. To see it let’s create some sample data.

def self.setup  
  User.delete\_all  
  Post.delete\_all  
  
  u = User.create name: 'Neeraj'  
  u.posts.create! title: 'ruby', desc: 'ruby is awesome'  
  u.posts.create! title: 'rails', desc: 'rails is awesome'  
  u.posts.create! title: 'JavaScript', desc: 'JavaScript is awesome'  
  
  u = User.create name: 'Neil'  
  u.posts.create! title: 'JavaScript', desc: 'Javascript is awesome'  
  
  u = User.create name: 'Trisha'  
end

With the above sample data if we execute User.joins(:posts) then this is the result we get

#<User id: 9, name: "Neeraj">  
#<User id: 9, name: "Neeraj">  
#<User id: 9, name: "Neeraj">  
#<User id: 10, name: "Neil">

We can avoid the duplication by using distinct .

User.joins(:posts).select('distinct users.\*').to\_a

Also if we want to make use of attributes from posts table then we need to select them.

records = User.joins(:posts).select('distinct users.\*, posts.title as posts\_title').to\_a  
records.each do |user|  
  puts user.name  
  puts user.posts\_title  
End

## Cron

If you’re reading this article, it’s probably because you’ve heard of cron jobs, cron tasks, or crontab. Cron is a piece of software written for \*nix-type operating systems to help with the scheduling of recurring tasks.You may want to use cron to schedule certain recurring actions in your Rails application, such as checking each day for expired accounts or clearing out expired sessions from your database.

It’s pretty easy to start working with cron jobs. You can start editing your cron tasks using the crontab command:

**Turbo link**

Turbolinks makes following links in your web application faster. Instead of letting the browser reload the JavaScript and CSS between each page change, and spend extra HTTP requests checking if the assets are up-to-date, we keep the current instance alive and replace only the body and the title in the head.

### How would you create getter and setter methods in Ruby?

Setter and getter methods in Ruby are generated with the attr\_accessor method. attr\_accessor is used to generate instance variables for data that’s not stored in your database column.

You can also take the long route and create them manually.

### What is a Filter ? When it is called?

Filters are methods that are called either before/after a controller action is called.

Say a user requests a controller action such as user dashboard/index

In such a case a filter can be setup so that the UserDashboard/index page is only accessible to loggedin users by adding the following lines towards the beginning of the page:

class UserDashboardController &lt; ApplicationController

before\_filter :confirm\_logged\_in, :except => [:login, :attempt\_login, :logout] def index .... end

|  |  |
| --- | --- |
| 1  2  3  4  5 | class UserDashboardController &lt; ApplicationController     before\_filter :confirm\_logged\_in, :except =&gt; [:login, :attempt\_login, :logout] |

**Disadvantages of Ruby on Rails?**

Runtime Speed - The most cited argument against Ruby on Rails is that it's "slow". I would agree, certainly when compared to the runtime speed of NodeJS or GoLang. Though in reality, the performance of a Ruby application is incredibly unlikely to be a bottleneck for a business. In 99% of cases, the bottleneck is going to be elsewhere, such as within the engineering team, IO, database or server architecture etc. When you get to a significant enough scale to have to worry about Rails runtime speed, then you're likely to have a incredibly successful application (think Twitter volume) and will have many scaling issues to deal with.

* Boot Speed - The main frustration we hear from developers working in Rails is the boot speed of the Rails framework. Depending on the number of gem dependencies and files, it can take a significant amount of time to start, which can hinder developer performance. In recent versions of Rails this has been somewhat combatted by the introduction of [*Spring*](https://github.com/rails/spring), but we feel this could still be faster.
* Documentation - It can be hard to find good documentation. Particularly for the less popular gems and for libraries which make heavy use of mixins (which is most of Rails). You'll often end up finding the test suite acts as documentation and you'll rely on this to understand behaviour. This isn't itself a bad thing, as the test suite should be the most up-to-date representation of the system, however, it can still be frustrating having to dive into code, when sometimes written documentation would have been much quicker.
* Multithreading - Rails supports multithreading, though some of the IO libraries do not, as they keep hold of the GIL (Global Interpreter Lock). This means if you're not careful, requests will get queued up behind the active request and can introduce performance issues. In practice, this isn't too much of a problem as, if you use a library that relies on GLI, you can switch to multiprocess setup. The knock-on effect of this is your application ends up consuming more compute resources than necessary, which can increase your infrastructure costs.
* ActiveRecord - AR is used heavily within the Ruby on Rails world and is a hard dependency for many of the RubyGems. Although we think it's a great design pattern, the biggest drawback we see is that your domain becomes tightly coupled to your persistence mechanism. This is far from ideal and can lead to bad architecture decisions. There are many ways to work around this, some of which are included in this [*7 Patterns to Refactor Fat ActiveRecord Models*](http://blog.codeclimate.com/blog/2012/10/17/7-ways-to-decompose-fat-activerecord-models/)article. We would like to see Rails become less reliant on ActiveRecord.

**Lambda vs Proc:**

* Both lambda and Proc are Proc objects.

proc = Proc.new { puts "Hello world" }  
lam = lambda { puts "Hello World" }  
proc.class # returns 'Proc'  
lam.class  # returns 'Proc'

* Lambdas check the number of arguments, while procs do not

lam = lambda { |x| puts x } # creates a lambda that takes 1 argument  
lam.call(2)                 # prints out 2  
lam.call           # ArgumentError: wrong number of arguments (0 for 1)  
lam.call(1,2,3)    # ArgumentError: wrong number of arguments (3 for 1)

proc = Proc.new { |x| puts x } # creates a proc that takes 1 argument  
proc.call(2)                   # prints out 2  
proc.call                      # returns nil  
proc.call(1,2,3)    # prints out 1 and forgets about the extra arguments

* A lambda will return normally, like a regular method. But a proc will try to return from the current context.

‘return’ inside of a lambda triggers the code right outside of the lambda code

‘return’ inside of a proc triggers the code outside of the method where the proc is being executed