**--Core Ruby--**

**1) What is Ruby?**

**A)** Ruby is an object oriented, open source, scripting language.

**2)Why Ruby?**

**A)**Ruby has a wealth of other features, among which are the following.

* Ruby has exception handling features, like Java or Python, to make it easy to handle errors.
* Ruby features a true mark-and-sweep garbage collector for all Ruby objects. No need to maintain reference counts in extension libraries. As Matz says, “This is better for your health.”
* Writing C extensions in Ruby is easier than in Perl or Python, with a very elegant API for calling Ruby from C. This includes calls for embedding Ruby in software, for use as a scripting language. A SWIG interface is also available.
* Ruby can load extension libraries dynamically if an OS allows.
* Ruby features OS independent threading. Thus, for all platforms on which Ruby runs, you also have multithreading, regardless of if the OS supports it or not, even on MS-DOS!
* Ruby is highly portable: it is developed mostly on GNU/Linux, but works on many types of UNIX, Mac OS X, Windows 95/98/Me/NT/2000/XP, DOS, BeOS, OS/2, etc.

**3)What is Agile Development programming and what are its strength's?**

**A)**Agile software development refers to a group of [software development methodologies](http://en.wikipedia.org/wiki/Software_development_methodologies) based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams

Advantages:

* Customer satisfaction by rapid, continuous delivery of useful software
* Working software is delivered frequently (weeks rather than months)
* Projects are built around motivated individuals, who should be trusted
* Continuous attention to technical excellence and good design
* Simplicity
* Self-organizing teams

Regular adaptation to changing circumstances

**4)What is difference between Symbol and String?**

**A)**Symbol are same like string but both behaviors is different based on object\_id, memory and process time (cpu time) Strings are mutable , Symbols are immutable.

**5)Is ruby support multiple inheritance?**

**A)**Ruby Supports only Single Inheritance.

You can achieve Multiple Inheritance through MIXIN concept means you achieve using module by including it with classes.

**6) What is module and mixin?**

**A)**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.

**Mixin:**

Ruby offers a very neat alternative concept called mixin. Modules can be imported inside other class using mixin. They are then mixed-in with the class in which they are imported**.**

Here’s an example:

module Debug

  def whoAmI?

    "I am #{self.to\_s}"

  end

end

class Photo

 include Debug

end

ph = Photo.new

"I am : #<Photo:0x007f8ea218b270>"

As you can see above the class Debug and it’s method “whoamI?” were mixed-in (added) with the class Photo.

That’s why you can now create an instance of the Photo class and call the whoAmI? method.

ph.whoAmI?

 => "I am : #<Phonograph:0x007f8ea218b270>"

**7)How to define a method body dynamically?**

**A)** An instance method can be defined dynamically with

Module#define\_method(name, body),

where name is the method’s name given as a Symbol, and body is its body given as a Proc, Method, UnboundMethod, or block literal. This allows methods to be defined at runtime, in contrast to def which requires the method name and body to appear literally in the source code.

class Conjure

def self` a.conjure(name, lamb)

define\_method(name, lamb)

end

end

# Define a new instance method with a lambda as its body

Conjure.conjure(:glark, ->{ (3..5).to\_a \* 2 })

Conjure.new.glark #=> [3, 4, 5, 3, 4, 5]

Module#define\_method is a private method so must be called from within the class the method is being defined on. Alternatively, it can be invoked inside class\_eval like so:

Array.class\_eval do

define\_method(:second, ->{ self.[](1) })

end

[3, 4, 5].second #=> 4

Kernel#define\_singleton\_method is called with the same arguments as Module#define\_method to define a singleton method on the receiver.

File.define\_singleton\_method(:match) do |file, pattern|

File.read(file).match(pattern)

end

File.match('/etc/passwd',/root/) #=> #<MatchData "root">

What is a Range?

Range is a great way to declare continuous variables. You should use it to declare arrays and other types of collections.

range1 = (1..4).to\_a

 => [1, 2, 3, 4]

puts range1

1

2

3

4

You can also create strings in this format and it fills in the interim values automatically.

range2 = ('bar'..'bat').to\_a

puts range2

bar

bas

bat

Since the end result of using range is an array you can also iterate over it just like any other array.

range2.each do |str|

   puts "In Loop #{str}"

end

This produces the result as shown below:

In Loop bar

In Loop bas

In Loop bat

**8)How to implement Singleton pattern?**

**A)Singleton means single instance.**

So, the goal of a singleton pattern is to write a class definition but only allow the creation of the single instance of that object.

This can be achieved nicely with the singleton gem as shown below:

require 'singleton'

 class Logger

  include Singleton

  def initialize

    @log = File.open("logfile.txt", "a")

  end

  def log(msg)

    @log.puts(msg)

  end

end

Adding the singleton as a mixin to the

Logger.instance.log('This is just a test message')

The code above will create a single instance of Logger and simply put the message in the logger file.

Singleton patterns are mostly used for DB instance, Logger instance, etc. —- cases where there should be ONE and only ONE instance of the object that is used.

Sometimes you might like to actually hold on to the logger object and use it everywhere you can do so by the following command:

logObj = Logger.instance

Notice you cannot use the Logger.new to create an object instance because this is a singleton object and therefore calling ‘new’ would fail.

**---Rails---**

**1)What is rails?**

**A)**Rails is a extremely productive web-application framework written in Ruby language by David Hansson.

* Rails are an open source Ruby framework for developing database-backend web applications.

Rails include everything needed to create a database-driven web application using the Model-View-Controller (MVC) pattern.

**2) Why rails or Rails components?**

**A)**There are lot of advantages of using ruby on rails.

1) DRY(DONT REPEAT YOURSELF)

2) Convention over Configuration

3) Gems And Plug-ins

4) Rack Support

5) Meta Programming

6) Scaffolding

7) Rest

8) Bundler

9) Action Mailer

**Components of Rails:**

**[1]: Action Pack:** Action Pack is a sin--gle gem that contains Action Controller, Action View and Action Dispatch. The “VC” part of “MVC”.

* **Action Controller:** Action Controller is the component that manages the controllers in a Rails application. The Action Controller framework processes incoming requests to a Rails application, extracts parameters, and dispatches them to the intended action. Services provided by Action Controller include session management, template rendering, and redirect management.
* **Action View:** Action View manages the views of your Rails application. It can create both HTML and XML output by default. Action View manages rendering templates, including nested and partial templates, and includes built-in AJAX support.
* **Action Dispatch:** Action Dispatch handles routing of web requests and dispatches them as you want, either to your application or any other Rack application. Rack applications are a more advanced topic and are covered in a separate guide called Rails on Rack.

**[2]: Action Mailer:** Action Mailer is a framework for building e-mail services. You can use Action Mailer to receive and process incoming email and send simple plain text or complex multipart emails based on flexible templates.

**[3]: Active Model:** Active Model provides a defined interface between the Action Pack gem services and Object Relationship Mapping gems such as Active Record. Active Model allows Rails to utilize other ORM frameworks in place of Active Record if your application needs this.

**[4]: Active Record:** Active Record are like Object Relational Mapping (ORM), where classes are mapped to table, objects are mapped to columns and object attributes are mapped to data in the table

**[5]: Active Resource:** Active Resource provides a framework for managing the connection between business objects and RESTful web services. It implements a way to map web-based resources to local objects with CRUD semantics.

**[6]: Active Support:** Active Support is an extensive collection of utility classes and standard Ruby library extensions that are used in Rails, both by the core code and by your applications.

**3) What is MVC architecture and how it works?**

**A)**MVC tends for Model-View-Controller, used by many languages like PHP, Perl, Python etc. The flow goes like this:

Request first comes to the controller, controller finds and appropriate view and interacts with model, model interacts with your database and send the response to controller then controller based on the response give the output parameter to view, for Example your url is something like this:

**http://localhost:3000/users/new**

here users is your controller and new is your method, there must be a file in your views/users folder named new.html.erb, so once the submit button is pressed, User model will be called and values will be stored into the database.

**4)What are the changes between rails2 and rails3?**

**A)**

1. Introduction of bundler (new way to manage your gem dependencies)
2. Gemfile and Gemfile.lock (where all your gem dependencies lies, instead of environment.rb)
3. HTML5 support

**5)What servers supported by rails?**

**A)**RoR was generally preferred over WEBrick server at the time of

writing, but it can also be run by:

* **Lighttpd** (pronounced **‘lighty’**) is an open-source web server more optimized for speed-critical environments.
* **Abyss Web Server-** is a compact web server available for windows, Mac osX and Linux operating system.

Apache and nginx

**6)What is the difference between webserver and application server?**

**A)**apache, nginx, IIS are web servers  
mongrel, webrick, phusion passenger are app servers

App server is something which works with particular programming language and parses and executes the code  
since mongrel and webrick can only work with rails, so they are app servers

Web servers are servers which can take the request from the browser.  
Web servers normally works on port 80 though we can change the port in configuration  
since mongrel and webrick can take that request directly, so they can be thought of as web servers but web servers do have a lot of other functionality like request pipeline, load balancing etc.  
App servers lack these functionalities.

**7)What is the difference between render and redirect\_to?**

**A)**render causes rails to generate a response whose content is provided by rendering one of your templates. Means, it will direct goes to view page.

redirect\_to generates a response that, instead of delivering content to the browser, just tells it to request another url. Means it first checks actions in controller and then goes to view page**.**

**8)What is the difference between session and cookies?**

**A)**Session is used to store user information on the server side. Maximum size is 4 kb. Cookies are used to store information on the browser side or we can say client side.

**9)What is the difference between GET and POST?**

**A)** GET is basically for just getting (retrieving) data, whereas POST may

involve anything, like storing or updating data, or ordering a product,

or sending E-mail.

**10)What is scaffolding difference between static an Dynamic?**

**A)Scaffolding:** Scaffolding is a meta-programming method of building

database-backend software application. It is a technique supported

by MVC frameworks, in which programmer may write a specification,

that describes how the application database may be used. There are

two type of scaffolding:

**-static:** Static scaffolding takes 2 parameter i.e your controller name and model name.

**-dynamic**: In dynamic scaffolding you have to define controller and model one by one.

**11)What are filters?When it is called?**

**A)**Filters are methods that run “before”, “after” or “around” a controller action. Filters are inherited, so if you set a filter on ApplicationController, it will be run on every controller in your application.

**12)Difference between plug-in and gem?**

**A) Gems and Plugins:** RubyGems is a package manager for the Ruby

programming language that provides a standard format for

distributing ruby programs and library.

**Plugins:** A Rails plugin is either an extension or a modification of the core

framework. It provides a way for developers to share bleeding-edge

ideas without hurting the stable code base. We need to decide if our

plugin will be potentially shared across different Rails applications.

* If your plugin is specific to your application, your new plugin will be **vendored** plugin.
* If you think, your plugin may be used across applications build it as a **gemified** plugin.

$rails generate plugin –help //vendored plugin

$rails plugin –help //gemified plugin

Most common plugin is **AutoStripAttributes** which helps to remove un-necessary whitespaces from Active Record or Active Model attributes. Its good for removing accidental spaces from user inputs.

**13)What are helpers? How it is use?**

**A)**Helpers (“view helpers”) are modules that provide methods which are automatically usable in your view. They provide shortcuts to commonly used display code and a way for you to keep the programming out of your views. The purpose of a helper is to simplify the view.

**14)Difference between observers and callbacks?**

**A)**Validations allow you to ensure that only valid data is stored in your database.

Example: validates\_presence\_of :user\_name, :password  
validates\_numericality\_of :valueWe can write custom validation also as

def validate  
errors.add(:price, “should be a positive value”) if price.nil?|| price < 0.01  
end

Callbacks and observers allow you to trigger logic before or after an alteration of an object’s state.

Callbacks are methods that get called at certain moments of an object’s life cycle. With callbacks it’s possible to write code that will run whenever an Active Record object is created, saved, updated, deleted, validated, or loaded from the database.

Callbacks are hooks into the life cycle of an Active Record object that allow you to trigger logic before or after an alteration of the object state. This can be used to make sure that associated and dependent objects are deleted when destroy is called (by overwriting before\_destroy) or to massage attributes before they’re validated (by overwriting before\_validation)

Observersare similar to callbacks, but with important differences. Whereas callbacks can pollute a model with code that isn’t directly related to its purpose, observers allow you to add the same functionality outside of a model. For example, it could be argued that a User model should not include code to send registration confirmation emails. Whenever you use callbacks with code that isn’t directly related to your model, you may want to consider creating an observer instead.

**15)How can implement internationalization in rails?**

**A)**Ruby ships with i18n which is an internationalization gem.

You need to create locale files and save them under the config/locales directory as:

en.yml

es.yml

fr.yml

The keys should match for each of these files.

en:   main\_page:     hello: â€œHelloâ€     welcome: â€œWelcome to   My Companyâ€ es:   main\_page:     hello: â€œHolaâ€     welcome: â€œBienvenido a Mi Empresaâ€ fr:   main\_page:     hello: â€œSalutâ€     welcome: â€œBienvenue Ã  Mon Entrepriseâ€

 In your code you would need to specify that the text would be locale specific. So change it to something like this:

.content   %h1     = t("main\_page.hello")

%p     = t("main\_page.welcome")

Then you have to select the actual locale.

**More details** [**here**](http://blog.crowdint.com/2011/02/15/internationalizing-your-rails-app.html)

**16) How to implement caching in rails?**

**A)**Rails offers multiple ways to cache content.

Fragment caching is my favorite because it gives you the choice to fragment to pull a portion from the cache and the remaining from a real-time DB call.

Say you wanted to show all the orders placed on your website in real time and didn’t want to cache that part of the page, but did want to cache the part of the page which lists all products available, you could use this piece of code:

<% Order.find\_recent .each do |o| %>

  <%= o.buyer.name %> bought <%= o.product.name %>

<% end %>

<% cache do %>  All available products:

  <% Product.all.each do |p| %>

    <%= link\_to p.name, product\_url(p) %>

  <% end %>

<% end %>

Another technique that works well for static pages is page caching. This technique is often used for home pages and is super fast.

class ProductsController < ActionController

   caches\_page:index

  def index

    @products = Products.all

  end

end

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

**A)**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.

**18)What is request.xhr?**

**A):** A request.xhr tells the controller that the new Ajax request has come, It always return Boolean values (TRUE or FALSE)

**--DB Migrations and Routing--**

**1)What is ORM?**

**A)**ORM tends for Object-Relationship-Model, where Classes are mapped

to table in the database, and Objects are directly mapped to the rows

in the table.

**2)What is Rails Migration and What can it do?**

**A)**Migrations are a convenient way to alter your database schema over time in a consistent and easy way. They use a Ruby DSL so that you don't have to write SQL by hand, allowing your schema and changes to be database independent.

* create\_table(name, options)
* drop\_table(name)
* rename\_table(old\_name, new\_name)
* add\_column(table\_name, column\_name, type, options)
* rename\_column(table\_name, column\_name, new\_column\_name)
* change\_column(table\_name, column\_name, type, options)
* remove\_column(table\_name, column\_name)
* add\_index(table\_name, column\_name, index\_type)
* remove\_index(table\_name, column\_name)

**3)What is Rake?**

**A)** Rake is a popular ruby gem that makes the job of running tasks simpler.

Rake is most often used for DB tasks

The common DB commands are:

rake db:migrate

rake db:reset

You can use cron to schedule rake tasks.

Sometimes you would create a dataloader.rake file and put it in the lib/tasks folder so that it can be used to populate the database on startup.

**4) What is the difference between has\_one and belongs\_to?**

**A)**A has\_one relationship is used to define a 1:1 relationship between two objects.

Examples are:

* A project has\_one projectManager
* A sandwich has\_one buyer
* A sandwich has\_one seller

A belongs\_to relationship on the other hand is used to define the reverse association for the same 1:1 relationship that is defined using the has\_one keyword.

class Employee < ActiveRecord::Base

has\_one :office

end

class Office < ActiveRecord::Base

belongs\_to :employee # foreign key - employee\_id

end

The important thing to keep in mind is that you need to declare both associations in order for the relationships to work correctly.

**5) What is polymorphic association?**

**A)** A polymorphic association is what one would call “open-ended” association of one class with multiple objects.

Let’s say you have a generic class called “Picture”.

Now this “Picture” class might be used to store Pictures for Employees, Products and Dogs. They all have pictures and they all are associated to the Picture class.

Hence, this type of open-ended association is called “Polymorphic” association.

Let’s see how we can declare them.

class Picture < ActiveRecord::Base

belongs\_to :imageable, :polymorphic => true

end

class Employee < ActiveRecord::Base

has\_many :pictures, :as => :imageable

end

class Product < ActiveRecord::Base

has\_many :pictures, :as => :imageable

end

class Dog < ActiveRecord::Base

has\_one :picture, :as => :imageable

end

Notice that the Dog object has a 1:1 relationship with the Picture class, whereas the Product class has a 1:many relationship with the Picture class.

This capability prevents the headache of having of declare new classes for each new type of Picture.

The polymorphic relationship would look somewhat like this.

**6) What is scope?**

**A)** Scopes are nothing more than SQL scope fragments. By using these fragments one can cut down on having to write long queries each time you access content.

Say you typically access content as shown below:

@posts = Post.where("published\_at IS NOT NULL AND posts.published\_at <= "+ Time.now)

Ruby offers you a nice way to put the where condition inside a scope statement as shown below.

class Post < ActiveRecord::Base

scope :published, lambda {

{ :conditions =>

["posts.published\_at IS NOT NULL AND posts.published\_at <= ?", Time.now]

}

}

scope :recent, :order => "posts.published\_at DESC"

end

Now you can simply access the published posts as: Post.published

@posts = Post.published

Also, you can access recent posts as

@recent\_posts = Post.recent

**7) What is eager loading?**

**A)** Eager loading is a great optimization strategy to reduce the number of queries that are made against the DB.

Say you are finding 10 employees and then you are looking for their post codes. Then your query would appear something like this:

clients = Client.limit(10)

clients.each do |client|

puts client.address.postcode

end

This may seem fine at first look but really this implementation leaves much to be desired. It makes 11 DB calls just to get the results.

Now you can optimize this query by making a slight change in the request like this:

clients = Client.includes(:address).limit(10)

clients.each do |client|

  puts client.address.postcode

end

This new request makes two SQL calls like this:

SELECT \* FROM clients LIMIT 10

SELECT addresses.\* FROM addresses

    WHERE (addresses.client\_id IN (1,2,3,4,5,6,7,8,9,10))

So, as you can see it really loads a lot more upfront and therefore it is called eager loading.

**8) How to serialize data with YAML?**

**A)**YAML is a straight forward machine parsable data serialization format,

designed for human readability and interaction with scripting language

such as Perl and Python.

YAML is optimized for data serialization, formatted dumping,

configuration files, log files, internet messaging and filtering.

**9) How to use two databases into a single application?**

**A)** magic multi-connections allows you to write your model once, and use them for the multiple rails databases at the same time.

* **sudo gem install magic\_multi\_connection**
* After installing this gem, just add this line at bottom of your environment.rb

**require “magic\_multi\_connection”**

**10)How you run your Rails application without creating databases?**

**A)**You can run your application by uncommenting the line in environment.rb

path=> rootpath conf/environment.rb

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

**11) How to use sql db or mysql db without defining it in the database.yml?**

**A)** You can use ActiveRecord anywhere

require “rubygems”

require “active\_record”

ActiveRecord::Base.establish\_connection({

:adapter=> ‘postgresql’, :user=>’foo’, :password=> ‘abc’, :database=>’whatever’})

**12)Explain RESTFUL architecture?**

**A)**

**RESTful:** REST stands for Representational State Transfer. REST is an architecture for designing both web applications and application programming interfaces (API’s), that’s uses HTTP

* RESTful interface means clean URLs, less code, CRUD interface.
* CRUD means Create-READ-UPDATE-DESTROY.

You might heard about HTTP verbs, GET, POST. In REST, they add 2 new verbs, i.e, PUT, DELETE.

There are 7 default actions, those are – index, show, new, create, edit, update, destroy.

**Action VERB**

index GET(used when you retrieve data from database)

show GET

new GET

create POST(used when you create new record in database)

edit GET

update PUT(used when you are updating any existing record in database)

destroy DELETE(used when you are destroying any record in database)

**---Deployment---**

**1)Which Deployment tool you have Used?**

**A)**Capistrano is a popular deployment tool — it allows developers to push code from their desktop to the servers.

**2) What are the key deployement challenges you have faced?**

**A)**heroku makes deployment easy.

Things that I sometimes run into are:

> Mismatched gem versions between local and production environment

> Some lessons learned:

»» Use image\_tag helper each time

»» Specify root path in ENV variable

»» Configure assets pipeline by setting: config.assets.enabled = true in the config/application.rb file

Configure Capistrano script to precompile assets

**3)Where can you find log file in your application?**

**A)**Rails will report errors from Apache in log/apache.log and errors from

the ruby code in log/development.log. If you having a problem, do have

a look at what these log are saying.

**-----------------END----------------------------------------------**

1. **Convention over Configuration:** Most web development framework for .NET or Java force you to write pages of configuration code. If you follow suggested naming conventions, Rails doesn’t need much configuration.
2. **Rack Support:** Rake is a software task management tool. It allows you to specify tasks and describe dependencies as well as to group tasks in a namespace.
3. **Metaprogramming:** Metaprogramming techniques use programs to write programs.
4. **Bundler:** Bundler is a new concept introduced in Rails 3, which helps you to manage your gems for application. After specifying gem file, you need to do a bundle install.