Rails Interview Questions

[Qus-1]: What is Ruby?

[Ans]: Ruby is a dynamic, reflective, general purpose,interpreted,pure object-oriented programming language with a super clean syntax that makes programming elegant and fun. Ruby successfully combines Smalltalk's conceptual elegance, Python's ease of use and learning, and Perl's pragmatism. Ruby originated in Japan in the early 1990s, and has started to become popular worldwide in the past few years as more English language books and documentation have become available.

[Qus-2]: Why ruby?

[Ans]:

1. It's pure object oriented language.
2. It's a dynamic language, you can do a lot at run time.
3. Ruby code is more human readable which makes it easy to understand.
4. In ruby in less line of code you can do a lot. No curly braces to look out for .
5. Next is the DRY concept (Don't Repeat Yourself). Ruby has lot of gems available to do lot of major tasks. It's In built methods also make life easy while coding in ruby.
6. Method chaining is one of the fascinating part of ruby. We can chain lot of methods to get the result reducing number of lines of code.
7. Monkey Patching: With ruby it is very easy to monkey patch ruby's built in methods which gives developers a great power while coding.
8. Great community, lot of developers to help.

[Qus-3]: Why Rails?

[Ans]:

Ruby on rails is one of the popular development frameworks for regular websites as well as kind of tools, web and mobile apps or combinations of all.

1. **DRY** - Everything is defined in single place which makes it easy to maintain code.
2. **Convention over configuration** -
    1. **Tables and Foreign Keys**: Tables are named in plurals(users, orders etc). Foreign Key has singular model name with underscore id etc.
    2. Now features like convention over configuration provides advantages like it becomes very easy for someone new to the project grasp the coding. That is it makes it more easy to understand. This saves a lot of time when comes to development where you have specified standard to follow.
3. **Default Locations:** With architecture of MVC we know where to find a specific code. Also rails makes it easy by providing separate place for storing code related to testing, libraries, configuration, languages etc.
4. **ORM:** Object Relational Mapping Class map to the table, object map to single row of the table, and object attributes map to the columns of the row. This makes interaction with database easy.
5. **Reuse of code:** Gems and plugins for many common functionalities like authentication, pagination, testing, etc.
6. **Agile practices:** Rails encourages agile practice with ever changing real world requirements.
7. **Security:** Rails prevents SQL injection attacks, XSS(Cross site Scripting) by default.

Some of the following reasons which makes it popular among others:

Higher Flexibility - In Ruby on Rails, it is easy to mold the platform to suit different business needs more efficiently than other frameworks. We can quickly mock up a working prototype, and then see how things work and then build on the idea.

Faster Development Speed - In Rails, there are lots of in-built components(called gems) available which we can re-use to build the application rather than building everything from scratch.

Agile Development at its best - Ruby on Rails is developed using the MVC design pattern along with the agile methodology which is well suited for web application development with fast changing requirements.

Integrated testing tools - Rails features a convenient testing tool as it automatically starts producing the skeleton test code in background at the time of development. Rails tests can also simulate browser requests and thus you can test your application's response without having to test it over browser.

Compatibility - Rails development is compatible with a wide range of web servers and databases; which ultimately makes the web deployment easier.

Cost-effective Web solutions - As it is an open source framework this helps in developing cost-effective web applications without sacrificing performance, speed and quality.

[Qus-4]: Disadvantages of ruby?

[Ans]: It uses lot of memory.

Open classes - enables you to change methods of classes . This can lead to problem in debugging if not coded properly.

[Qus-5]: Disadvantages of rails?

[Ans]: It uses lot of memory.

[Qus-6]: What is Memorization?

[Ans]: Memorization is the process of storing a computed value to avoid duplicated work by future calls.

[Qus-7]: What are the string methods in ruby that you have used?

[Ans]: 1. include?

Returns true if str contains the given string or character.

ex: "hello".include? "lo" #=> true

2. index

Returns the index of the first occurrence of the given substring or pattern (regexp) in str

ex: "hello".index('e') #=> 1

"hello".index(/[aeiou]/, -3) #=> 4

3. downcase

"hEllO".downcase #=> "hello"

4. empty?

Returns true if str has a length of zero.

ex: "hello".empty? #=> false

5. gsub
Returns a copy of str with the all occurrences of pattern substituted for the second argument.
ex: "hello".gsub(/[aeiou]/, '*') #=> "h*ll*"

6. length
Returns the character length of str.
7. reverse
8. size
9. strip
Returns a copy of str with leading and trailing whitespace removed.
ex: "\tgoodbye\r\n".strip #=> "goodbye"
10. to_s
11. upcase

[Qus-8]: What are the array methods in ruby that you have used?

[Ans]: 1. empty?
2. reverse
3. size
4. sort
5. uniq
6. compact
Removes nil elements from the array.
"a", nil, "b", nil, "c"].compact! #=> ["a", "b", "c"]
7. delete
8. flatten
a = [1, 2, [3, [4, 5]]]
a.flatten(1) #=> [1, 2, 3, [4, 5]]
9. insert
a.insert(2, 99) #=> ["a", "b", 99, "c", "d"]
10. join
Returns a string created by converting each element of the array to a string, separated by the given separator.
["a", "b", "c"].join("-") #=> "a-b-c"
11. pop
12. push
13. sample
14. shift

[Qus-9]: In which Programming language was Ruby written?

[Ans]: Ruby was written in C language and Ruby on Rails written in Ruby.

[Qus-10]: What is meant by dynamic programming?

[Ans]: Program that, at runtime, execute many common behaviors that other languages might perform during compilation. These behaviors could include extension of the program, by adding

new code, by extending objects and definitions, or by modifying the type system, all during program execution.

[Qus-11]: What is meant by reflective programming?

[Ans]: reflection is the ability of a computer program to examine and modify the structure and behavior (specifically the values, meta-data, properties and functions) of an object at runtime.

[Qus-12]: What is meant by Object-oriented programming?

[Ans]: Object-oriented programming (OOP) is a programming paradigm that represents concepts as "objects" that have data fields (attributes that describe the object) and associated procedures known as methods. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

[Qus-13]: What is Object?

[Ans]: Object = state + behaviour
An object is defined by its state, the data that is holds and things that it can do with data known as the behaviour.

[Qus-14]:Define a Class.

[Ans]: classes hold data, have methods that interact with that data, and are used to instantiate objects.

[Qus-15]:What is self?

[Ans]: Self in ruby represents the current object

[Qus-16]:What are the object-oriented programming features supported by Ruby?

[Ans]: Classes,Objects,Inheritance,Singleton methods, polymorphism(accomplished by overriding and overloading) are some oo concepts supported by ruby.

[Qus-17]:What are the operating systems supported by Ruby?

[Ans]: Windows and linux operating systems are supported by the Ruby.

[Qus-18]:What is the difference between Symbol and String?

[Ans]: -The symbol in Ruby on rails act the same way as the string but the difference is in their behaviors that are opposite to each other.
-The difference remains in the object_id, memory and process time for both of them when used together at one time.
-Strings are considered as mutable objects. Whereas, symbols, belongs to the category of immutable.
-Strings objects are mutable so that it takes only the assignments to change the object information. Whereas, information of, immutable objects gets overwritten.
-Symbols are used to show the values for the actions like equality or non-equality to test the symbols faster than the string values.

[Qus-19]: What is Rails? And what are the components of Rails?

[Ans]: Rails is a extremely productive web-application framework written in Ruby language by **David Hansson in 2004.**

- 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 **M**odel-**V**iew-**C**ontroller (MVC) pattern.
- that you could develop a web application at least ten times faster with Rails than you could with a typical Java framework or any other framework.

Components of Rails:

[1]: Action Pack: Action Pack is a single 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.

[Qus-20]: Explain about RESTful Architecture.

[Ans]: 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)

[Qus-21]: Why Ruby on Rails?

[Ans]: There are lot of advantages of using ruby on rails.

1. **DRY Principal(Don't Repeat Your Self):** It is a principle of software development aimed at reducing repetition of code. "*Every piece of code must have a single, unambiguous representation within a system*"

2. **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.

3. **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.

4. **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.

5. **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.

6. **Metaprogramming:** Metaprogramming techniques use programs to write programs.

7. **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.

8. **Rest Support:** As explained above.

9. **Action Mailer:** As explained above.

[Qus-22]: What do you mean by render and redirect_to?
[Ans]: 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.

[Qus-23]: What is ORM in Rails?
[Ans]: ORM stands for Object Relationship Model that models the classes and helps in setting a relationship between the existing models.
It allows the classes to be mapped to the table that is present in the database and objects get mapped to the rows in database table.
It shows the relationship that exists between the object and frame it using the model to display the output to the users.
It keeps the data in the database according to its relationships and performs the functions accordingly.
It maps the database in case of any updates and update for the same if the fields are changed or new values are entered.

[Qus-24]: How many Types of Associations Relationships does a Model have?
[Ans]: When you have more than one model in your rails application, you would need to create connection between those models. You can do this via associations. Active Record supports three types of associations:

- **one-to-one:** A one-to-one relationship exists when one item has exactly one of another item. For example, a person has exactly one birthday or a dog has exactly one owner.
- **one-to-many :** A one-to-many relationship exists when a single object can be a member of many other objects. For instance, one subject can have many books.
- **many-to-many :** A many-to-many relationship exists when the first object is related to one or more of a second object, and the second object is related to one or many of the first object.

You indicate these associations by adding declarations to your models: has_one, has_many, belongs_to, and has_and_belongs_to_many.

[Qus-25]: What are helpers and how to use helpers in ROR?
[Ans]: 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.

[Qus-26]: What is Filters?
[Ans]: 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.

[Qus-27]: What is MVC? and how it Works?
[Ans]: 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.

[Qus-28]: What is Session and Cookies?
[Ans]: 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

[Qus-29]: What is request.xhr?
[Ans]: A request.xhr tells the controller that the new Ajax request has come, It always return Boolean values (TRUE or FALSE)

[Qus-30]: What things we can define in the model?
[Ans]: There are lot of things you can define in models few are:
1. Validations (like validates_presence_of, numeracility_of, format_of etc.)
2. Relationships(like has_one, has_many, HABTM etc.)
3. Callbacks(like before_save, after_save, before_create etc.)
4. Suppose you installed a plugin say validation_group, So you can also define validation_group settings in your model
5. ROR Queries in Sql
6. Active record Associations Relationship

[Qus-31]: How many types of callbacks available in ROR?
[Ans:]

(1) before_validation
(2) before_validation_on_create
(3) validate_on_create
(4) after_validation
(5) after_validation_on_create
(6) before_save
(7) before_create
(8) after_create
(9) after_save

[Qus-32]: How to serialize data with YAML?

[Ans]: 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.

[Qus-33]: How to use two databases into a single application?

[Ans]: 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"

[Qus-34]: Changes between the Rails Version 2 and 3?

[Ans]:

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

[Qus-35]: What is TDD and BDD?

[Ans]: TDD stands for Test-Driven-Development and BDD stands for Behavior-Driven-Development.

[Qus-36]: What are the servers supported by ruby on rails?

[Ans]: 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**

[Qus-37]: What do you mean by Naming Convention in Rails.

[Ans]:

- **Variables:** Variables are named where all letters are lowercase and words are separated by underscores. E.g: total, order_amount.
- **Class and Module:** Classes and modules uses MixedCase and have no underscores, each word starts with a uppercase letter. Eg: InvoiceItem
- **Database Table:** Table name have all lowercase letters and underscores between words, also all table names to be plural. Eg: invoice_items, orders etc
- **Model:** The model is named using the class naming convention of unbroken MixedCase and always the singular of the table name. For eg: table name is might be orders, the model name would be Order. Rails will then look for the class definition in a file called order.rb in /app/model directory. If the model class name has multiple capitalized words, the table name is assumed to have underscores between these words.
- **Controller:** controller class names are pluralized, such that OrdersController would be the controller class for the orders table. Rails will then look for the class definition in a file called orders_controlles.rb in the /app/controller directory.

Summary

1. **Model Naming Convention**

Table: orders
Class: Order
File: /app/models/order.rb
Foreign key: customer_id
Link Table: items_orders

2. **Controller Naming Convention**

Class: OrdersController
File: /app/controllers/orders_controller.rb
Layout: /app/Layouts/orders.html.erb

3. **View Naming Convention**

Helper: /app/helpers/orders.helper.rb
Helper Module: OrdersHelper
Views: /app/view/orders/…(list.html.erb for example)

4. **Test Naming Convention**

Unit: /test/unit/order_test.rb
Functional: /test/functional/orders_controller_test.rb
Fixtures: /test/fixtures/orders.yml

[Qus-38]: What is the log that has to seen to check for an error in ruby rails?
[Ans]: 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.

[Qus-39]: How you run your Rails application without creating databases?
[Ans]: 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]

[Qus-40]: How to use sql db or mysql db without defining it in the database.yml?
[Ans]: You can use ActiveRecord anywhere

```
require “rubygems”
require “active_record”
ActiveRecord::Base.establish_connection({
    :adapter=> ‘postgresql’, :user=>’foo’, :password=> ‘abc’, :database=>’whatever’})
```

[Que-41]: GET and POST Method?
[Ans]: 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.

[Que-42]: Difference between PUT and POST?
[Ans]: PUT is used for update or creates,Whereas POST is used for create

[Que-43]: What is the difference between rails patch and put http methods ?
[Ans]: PATCH is the correct HTTP verb to map to the #update action. The semantics for PATCH allows for partial updates, whereas PUT requires a complete replacement.

[Que-44]: Write a program to show the functionality of request.xhr in Ruby on Rails?

[Ans]: Request from the request.xhr displays the control er that manages and creates the new AJAX that is being handled by the new control er.

The Boolean values that are returned should generate only TRUE or FALSE. These values are used to be inserted for this purpose. The request.xhr is being shown in a program shown below as:

```
def create
 return unless request.xhr?
 @imageable = find_imageable
 @image = @imageable.images.build(params[:imageable])
 @image.save
 render :layout => false
 end
```

[Que-45]: What is Fingerprinting?

[Ans]:Fingerprinting is a technique that makes the name of a file dependent on the contents of the file. When the file contents change, the filename is also changed. For content that is static or infrequently changed, this provides an easy way to tell whether two versions of a file are identical, even across different servers or deployment dates.

For example a CSS file global.css could be renamed with an MD5 digest of its contents: Global-908e25f4bf641868d8683022a5b62f54.css

[Que-46]: What is lambda and proc? What is the difference? State it with an example.

[Ans]: Lambda takes block and converts it into an object.

```
l = lambda { |a| a+1 }
puts l.call(99)
```

Proc is also another way of converting a block into an object.

```
l = Proc.new { |a| a+1 }
puts l.call(99)
```

Difference between Proc and lambda:

i. Parameters:

Proc is not strict in terms of number of arguments passed to it.

```
l = Proc.new {|a,b,c| p a,b,c}
l.call(1,2,3,4) => It ignores 4th parameter
```

Lambda is strict

```
l = lambda {|a,b,c| p a,b,c}
l.call(1,2,3,4) => ArgumentError
```

ii. Return:

```
def method
  puts “at top”
  pr = Proc.new {return }
  pr.call
```

```
  puts "at end"
end

puts "before call"
method
puts "after call"
```

```
output:
before call
at top
after call

def method
  puts "at top"
  pr = lambda {return }
  pr.call
  puts "at end"
end

puts "before call"
method
puts "after call"
```

```
output:
before call
at top
at end
after call
```

So the difference is the return in lambda exists from proc itself (exist from the block of code) whereas Proc.new exists from the surrounding context

[Que-47]: What is the purpose of load, auto_load, and require_relative in Ruby?

[Ans]:*Load allows the process or a method to be loaded in the memory and it actually processes the execution of the program used in a separate file.
It includes the classes, modules, methods and other files that executes in the current scope that is being defined.
It performs the inclusion operation and reprocesses the whole code every time the load is being called.
*Auto_load: this initiates the method that is in hat file and al ows the interpreter to cal the method.
*require_relative: al ows the loading to take place of the local folders and files.

[Que-48]: What is the difference between Class and Module?

[Ans]:

	class	module
instantiation	can be instantiated	can not be instantiated
usage	object creation	mixin facility provide a namespace
superclass	module	object
consists of	methods, constants and variables	methods, constants and classes
methods	class methods, instance methods	module methods, instance methods
inheritance	inherits behavior and can be base for inheritance	No inheritance
inclusion	can not be included	can be included in classes/modules by using *include* command (includes all instance methods as instance methods in class/module)
extension	can not extend with extend command (only with inheritance)	module can extend instance by using *extend* command (extends given instance with singleton methods from module)

[Que-49]: What is the Difference between Application Server and Web Server ?

[Ans]: 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.
About Mongrel server:
Mongrel work as web as wel as app server if you are talking about dev environment
But in production, mongrel alone cannot work it will be too slow
so we need a web server in front of mongrel

[Que-50]: What is the difference between XSS and CSRF?

[Ans]: XSS trys to steal your credentials while CSRF uses them.

[Que-51]: Ruby Support Single Inheritance/Multiple Inheritance ?

[Ans]: Ruby Supports only Single Inheritance. You can achieve Multiple Inheritance through MIXIN concept means you achieve using module by including it with classes.

[Que-52]: What is the Difference between Gem and Plug-in?

[Ans]: GEM

1. Gem is a packaged ruby application using the packaging system defined by RubyGems.

2. Rails itself is a Gem

3. We can install, upgrade and query the gem version.

4. Gem instal ed for Ruby interpreter can be used system wide by that interpreter.

Plug-in

1. Plugin is an extension of Rails Framework.

2. Cannot be upgraded by using a command. To upgrade one have to uninstall and then install Upgraded version.

3. Has to be hooked into rails application. (Has to have init.rb)

4. Have an install.rb file.

5. Can only be used application wide.

15. Mixins in Ruby

[Que-53]: Explain Caching in rails.

There are 4 types of caching available in rails:

i. page caching:

In this type of caching the requested page is served by the webserver(i.e, apache, nginx) without having to communication with rails application.

ii. Action caching:

Page Caching cannot be used for actions that have before filters - for example, pages that require authentication. This is where Action Caching comes in. Action Caching works like Page Caching except the incoming web request hits the Rails stack so that before filters can be run on it before the cache is served. This allows authentication and other restrictions to be run while still serving the result of the output from a cached copy.

(Action caching is deprecated in rails 4)

iii. Fragment caching:

Fragment Caching allows a fragment of view logic to be wrapped in a cache block and served out of the cache store when the next request comes in.

As an example, if 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 %>
```

iv. SQL Caching:

Query caching is a Rails feature that caches the result set returned by each query so that if Rails encounters the same query again for that request, it will use the cached result set as opposed to running the query against the database again.

```
class ProductsController < ApplicationController
        def index
```

```
                # Run a find query
                @products = Product.all
...
                # Run the same query again
                @products = Product.all
              end
        end
```

[Que-54]: What is the difference between eager loading and lazy loading?

[Ans]: One way to improve performance is to cut down on the number of SQL queries. You can do this through eager loading.

User.find(:all, :include => :friends)
Here you are firing only two queries :

1) One for all users.
2) One for all friends of users .
Lazy Loading :

cars = Car.where(:colour => 'black')
The query is not executed, until you do something like:
cars.each {|c| puts c.name }

[Que-55]: What is the difference between create and new?

[Ans]: Say you are creating a blog and you have a model object called post. In the controller the new method will send a new post object to the view (a form). When the user fills out the form in the view and clicks submit, the form will POST the data to the controller's create method where it will be saved.

[Que-56]: What is Ruby Gems?

[Ans]: A. Ruby Gem is a software package, commonly called a "gem". Gem contains a packaged Rubyapplication or library. The Ruby Gems software itself allows you to easily download, install andmanipulate gems on your system.

[Que-23]: What is the difference between Observers & Callbacks ?

[Ans]: Observers allow you to factor out code that doesn't really belong in models. For example, a User model might have a callback that sends a registration confirmation email after the user record is saved, but you don't really want this code in the model because it's not directly related to the model's purpose.
Observers allow you to have that clean separation because you don't have all that callback code in your models. Observers depend a model (or models), not the other way around.

[Que-57]: What is purpose of RJs in Rails ?

[Ans]: RJS is a template (similar to an html.erb file) that generates JavaScript which is executed in an eval block by the browser in response to an AJAX request. It is sometimes used

(incorrectly?) to describe the JavaScript, Prototype, and Scriptaculous Helpers provided by Rails.

[Que-58]:What is a symbol and how it differs from variable?

[Ans]: A Symbol in Ruby is basically the same thing as symbol in the real world. It is used to representor name something. Symbols are very commonly used to represent some kind of state, for example :

order.status = :canceled

order.status = :confirmed

Variables and Symbols are different things. A variable points to different kind of data. In Ruby, a symbol is more like a string than a variable.

In Ruby, a string is mutable, where as a symbol is immutable. That means that only one copy of a symbol needs to be created. symbols are often used as the equivalent to enums in Ruby, as well as keys to a dictionary (hash).

[Que-59]: What is Gem file and Gemfile.lock?

[Ans]: The Gem file is where you specify which gems you want to use, and lets you specify which versions. The Gemfile.lock file is where Bundler records the exact versions that were installed. This way, when the same library/project is loaded on another machine, running bundle install will look at the Gemfile.lock and install the exact same versions, rather than just using the Gem file and installing the most recent versions.

[Que-60]:What is the purpose of layouts?

[Ans]: Layouts are partial ruby/html files that are used to render the content pages.

There are placed in the folder: app/views/layouts

Items that you would typically put in this folder are things like headers/footers, navigation elements, etc.

[Que-61]:How can you implement method overloading?

[Ans]: In the case of Ruby method overloading is not supported.

However, it does support the overall goal of passing variable number of parameters to the same method. You would implement it like this:

```
class MyClass
  def initialize(*args)
    if args.size < 2  || args.size > 3
      puts 'This method takes either 2 or 3 arguments'
    else
      if args.size == 2
        puts 'Found two arguments'
      else
        puts 'Found three arguments'
      end
    end
```

```
  end
end
```

The output can be seen here:

```
MyClass.new([10, 23], 4, 10)
Found three arguments
MyClass.new([10, 23], [14, 13])
Found two arguments
```

[Que-62]: What is a Range?

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

[Que-63]: Does Ruby support constructors? How are they declared?

[Ans]: 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
```

[Que-64]: What is the difference between symbol and string in ruby?

[Ans]: A symbol is something you use to represent names or labels. Strings are mutable where as symbol are immutable objects.

Multiple declaration of symbols representing a single value are unique (i.e, no matter how many times you call same symbol it is stored in single location of memory)whereas every string declared with same value occupies different locations in memory and hence it is more memory efficient to use symbol as they refer to same address in the memory.

When to use symbols? Symbols are excellent choice for hash keys.

```
options = {}
options[:auto_save]     = true
options[:show_comments] = false
```

[Que-65]: What is the difference between Nginx and Apache?

[Ans]: - Nginx is based on event-driven architecture. Apache is based on process-driven architecture.

- Nginx doesn't create a new process for a new request. Apache creates a new process for each request.
- In Nginx, memory consumption is very low for serving static pages. But, Apache's nature of creating new process for each request increases the memory consumption.
- Apache has tons of features and provides lot more functionality than Nginx.
- Apache is well documented compared to nginx.

[Que-66]: Why should we use Nginx?

[Ans]: i. Nginx Architecture:
Nginx uses event driven architecture instead of threaded model. There is one master process that spawns a configured number of workers all waiting for an event on the main event loop. When a user sends a request an event is triggered and one of the workers picks up the request, it then servers it back to the user in response and waits for another request.
ii. Performance:
This type of architecture has the benefit of keep a low, consistent memory footprint which is great for handling small and large application.
iii. Documentation:
Nginx has a well documented api. This makes customizing nginx easy.

[Que-67]: What is Nginx?

[Ans]: High performance HTTP and proxy server(a proxy server is a server (a computer system or an application) that acts as an intermediary for requests from clients seeking resources from other servers.). When used with rails application Nginx will route request appropriately. Some requests may go to the rails app where as some request may be for static content. It supports hosting multiple sites.

[Que-68]: Explain Apache server.

[Ans]:Apache is thread based.In the traditional thread-based models, for each client there is one thread which is completely separate and is dedicated to serve that thread. This might cause I/O blocking problems when process is waiting to get completed to release the resources (memory, CPU) in hold. Also, creating separate processes consumes more resources.
The need for serving large number of concurrent requests is raising every day. The prediction of C10K problem (i.e 10,000 concurrent clients) started the research on web server architecture which could solve this problem. As a result Nginx architecture was developed.

[Que-69]: Difference between Join and include?

[Ans]:Joins will just joins the tables and brings selected fields in return. if you call associations on joins query result, it will fire database queries again
Include will eager load the included associations and add them in memory. Include loads all the included tables attributes. If you call associations on include query result, it won't fire any queries

[Que-70]:What is monkey patching?

[Ans]:Monkey patching is replacing methods of a class at runtime.

[Que-71]: What's new in Rails 4.1?

[Ans]:

1. Action Mailer Previews

Testing email templates in Rails has always been pretty painful. With the MailView gem into Rails 4.1 you can now easily create previews for your mailers and view them in the browser from http://localhost:3000/rails/mailers

```
# In /test/mailers/previews/notifier_preview.rb
class NotifierPreview < ActionMailer::Preview
```

```
  def welcome
    # Mock up some data for the preview
    user = FactoryGirl.build(:user)

    # Return a Mail::Message here (but don't deliver it!)
    Notifier.welcome(user)
  end
end
```

2. Action Pack Variants

As web developers, we are well aware that we have fully transitioned into the post-PC era. As much as I love responsive design, it is not a silver bullet for the multi-device web. In many cases, it is more appropriate to tailor your views to serve the most relevant content and workflow for specific device categories.

With Action Pack Variants, this will become much easier in Rails 4.1:

```
class ApplicationController < ActionController::Base
  before_action :detect_device_variant
  private
    def detect_device_variant
      case request.user_agent
      when /iPad/i
        request.variant = :tablet
      when /iPhone/i
        request.variant = :phone
      end
    end
end

class PostController < ApplicationController
  def show
    @post = Post.find(params[:id])
    respond_to do |format|
      format.json
      format.html             # /app/views/posts/show.html.erb
      format.html.phone       # /app/views/posts/show.html+phone.erb
      format.html.tablet do
        @show_edit_link = false
      end
    end
  end
end
```

iii. secrets.yml:

Rails 4.1 generate a new secrets.yml file in the config folder. By default, this file contains the application's secret_key_base, but it could also be used to store other secrets such as access keys for external APIs.

[Que-72]: What is sprockets?

[Ans]: Sprockets is a Ruby library for compiling and serving web assets. It features declarative dependency management for JavaScript and CSS assets, as well as a powerful preprocessor pipeline that allows you to write assets in languages like CoffeeScript, Sass, SCSS and LESS.

[Que-73]: What is Polymorphic Association?

[Ans]: In Rails, Polymorphic Associations allow an ActiveRecord object to be associated with multiple ActiveRecord objects. A perfect example for that would be comments in a social network like Facebook. You can comment on anything on Facebook like photos, videos, links, and status updates. It would be impractical if you were to create a comment model (photos_comments, videos_comments, links_comments) for every other model in the application. Rails eliminates that problem and makes things easier for us by allowing polymorphic associations.

[Que-74]: What is rake?

[Ans]: rake is command line utility of rails. "Rake is Ruby Make, a standalone Ruby utility that replaces the Unix utility 'make', and uses a 'Rakefile' and .rake files to build up a list of tasks. In Rails, Rake is used for common administration tasks, especially sophisticated ones that build off of each other."

[Que-75]: Is it possible to embed partial views inside layouts? How?

[Ans]: Yes it is possible. You Embed partial views inside the file /app/views/layout/application.html.erb and then whenever you render any page this layout is merged with it.

[Que-76]: How can you list all routes for an application?

[Ans]: By writing rake routes in the terminal we can list out all routes in an application.

[Que-77]: What is a sweeper in rails?

[Ans]: Sweepers are the terminators of the caching world and responsible for expiring caches when model objects change. They do this by being half observers, half filters and implementing callbacks for both roles.

[Que-78]: What is eager loading, lazy loading and overeager loading?

[Ans]: Generally three levels are there:

1. Eager loading: you do everything when asked. Classic example is when you multiply two matrices. You do all the calculations. That's eager loading.
2. Lazy loading: you only do a calculation when required. In the previous example, you don't do any calculations until you access an element of the result matrix.
3. Overeager loading: this is where you try and anticipate what the user will ask for and preload it.

For Example:

Imagine a page with rollover images like for menu items or navigation. There are three ways the image loading could work on this page:

1.Load every single image required before you render the page (eager).
2.Load only the displayed images on page load and load the others if/when they are required (lazy).
3.Load only the displayed images on page load. After the page has loaded preload the other images in the background in case you need them (overeager).

[Que-79]: How can you implement method overloading?

[Ans]: You want to create two different versions of a method with the same name: two methods that differ in the arguments they take. However, a Ruby class can have only one method with a given name (if you define a method with the same name twice, the latter method definition prevails as seen in example p038or.rb in topic Ruby Overriding Methods). Within that single method, though, you can put logic that branches depending on how many and what kinds of objects were passed in as arguments.

Here's a Rectangle class that represents a rectangular shape on a grid. You can instantiate a Rectangle by one of two ways: by passing in the coordinates of its topleft and bottomright corners, or by passing in its topleft corner along with its length and width. There's only oneinitialize method, but you can act as though there were two.

```
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])
```

The above program p037rectangle.rb is incomplete from the Rectangle class viewpoint, but is enough to demonstrate how method overloading can be achieved. Also remember that the initialize method takes in a variable number of arguments.

[Que-80]: Can you give an example of a class that should be inside the lib folder?

[Ans]: Modules are often placed in the lib folder.

[Que-81]:. Where should you put code that is supposed to run when your application launches?

[Ans]: In the rare event that your application needs to run some code before Rails itself is loaded, put it above the call to require 'rails/all' in config/application.rb.

[Que-82]:. What is the purpose of Yield in Ruby?

[Ans]: Ruby provides a yield statement that eases the creation of iterators.
The first thing I noticed was that its behavior is different from the C# yield statement (which I knew from before).
Ruby's yield statement gives control to a user specified block from the method's body. A classic example is the Fibonacci Sequence:

```
class NumericSequences
  def fibo(limit)
    i = 1
  yield 1
 yield 1
  a = 1
  b = 1
 while (i < limit)
 t = a
                a = a + b
                b = t
                yield a
                i = i+1
              end
            end
            ...
          end
```

The fibo method can be used by specifying a block that will be executed each time the control

Reaches a yield statement. For example:

```
irb(main):001:0> g = Numeric Sequences::new
=> #<NumericSequences:0xb7cd703c>
irb(main):002:0> g.fibo(10) {|x| print "Fibonacci number: #{x}\n"}
Fibonacci number: 1
Fibonacci number: 1
Fibonacci number: 2
Fibonacci number: 3
Fibonacci number: 5
Fibonacci number: 8
Fibonacci number: 13
Fibonacci number: 21
```

Fibonacci number: 34
Fibonacci number: 55
Fibonacci number: 89

[Que-83]:. What is Bundler?
[Ans]: Bundler is a new concept introduced in Rails3, which helps to you manage your gems for the application. After specifying gems in your Gemfile, you need to do a bundle install. If the gem is available in the system, bundle will use that else it will pick up from the rubygems.org.
Here is a best link i got http://bundler.io/ to know more bundler more.

[Que-84]:. What id the difference between Static and Dynamic Scaffolding?
[Ans]: The Syntax of Static Scaffold is like this: ruby script/generate scaffold Home New
Where New is the model and Home is your controller, In this way static scaffold takes 2 parameter i.e your controller name and model name, whereas in dynamic scaffolding you have to define controller and model one by one

.

[Que-85]:. What is the difference between nil and false in ruby?
[Ans]: False is a boolean datatype Nil is not a data type it have object_id 4.

[Que-86]:. What deployment tool do you use?
[Ans]: Capistrano is a popular deployment tool, it allows developers to push code from their desktop to the servers. you can also use chef as a deployment tool for your project.

[Que-87]:. Why do some people say "Rails can't scale"?
[Ans]: Twitter was one of the first extremely high profile sites to use Rails. In roughly the 2006-2008 timeframe, the growth rate of Twitter made server errors ("fail whale") appearances a very common occurrence for users, prompting users and tech pundits to lay blame at Rails' feet. As is true with any software, the causes of scalability issues can be complex and multi-faceted. Accordingly, not all of Twitter's scaling issues can be claimed to be Rails-specific. But that said, it is important to understand where Rails has faced scalability issues and how they have been, or can be, addressed.
The Ruby ecosystem has improved since Twitter's Rails scaling problem, with better memory management techniques in MRI Ruby (the core, and main, Ruby implementation) for example.
Modern Rails applications typically mitigate scaling problems in one or more of the following ways:
- Implementing caching solutions (Rails 4 introduces good advances here)
- Leveraging (or implementing) server or platform solutions with automatic scaling built in
- Profiling costly operations and moving them out of Ruby or out of one monolithic Rails app
- Placing some operations in a background / worker queue to be completed at a later time (e.g., perform an export operation asynchronously, notifying the user by email with a download link when the export is completed)

While there has traditionally been a one-to-one mapping between websites and Rails app (i.e., one website = one Rails app), there's been an increasing

movement towards more of a Service Oriented Architecture (SOA)approach whereby performance critical parts of the app are split off into new/separate apps which the main app usually talks to via web service calls. There are numerous advantages to this approach. Perhaps most noteworthy is the fact that these independent services can employ alternate technologies as appropriate; this might be a lightweight / more responsive solution in Ruby, or services written in Scala (as in Twitter's case), orNode.js, Clojure, or Go.
But writing separate services isn't the only way to speed up a Rails app. For example, Github has an interestingarticle on how it profiled Rails and ended up implementing a set of C apis for performing text escaping on the web

[Que-88]: Thirty-Six Reasons I Love Ruby

[Ans]:

1.It's a *pure* OOP language. Am I being redundant? I don't think so. By this wemean that everything, including primitive data types such as strings and integers, is represented as an object. There is no need for wrapper classes such as Java has. And in addition, even constants are treated as objects, so that a method may be invoked with, for example, a numeric constant as a receiver.

2.It's a dynamic language. For people only familiar with more static languages such as C++ and Java, this is a significant conceptual leap. It means that methods and variables may be added and redefined at runtime. It obviates the need for such features as C's conditional compilation (#ifdef), and makes possible a sophisticated reflection API. This in turn allows programs to become more "self-aware" -- enabling runtime type information, detection of missing methods, hooks for detecting added methods, and so on. Ruby is related to Lisp and Smalltalk in this respect.

3.It's an interpreted language. This is a complex issue, and deserves several comments. It can be argued that performance issues make this a negative rather than a positive. To this concern, I reply with these observations: 1. First and foremost: A rapid development cycle is a great benefit, and it is encouraged by the interpreted nature of Ruby. 2. How slow is too slow, anyway? Do some benchmarks before you call it slow. 3. Though some will criticize me, I will say this anyway: Processors are getting faster every year. 4. If you absolutely need the speed, you can write part of your code in C. 5. Finally, in a sense, it is all a moot point, since no language is inherently interpreted. There is no law of the universe that says a Ruby compiler cannot be written.

4.It understands regular expressions. For years, this was considered the domain
of UNIX weenies wielding clumsy tools such as grep and sed, or doing fancy search-and-replace operations in vi. Perl helped change that, and now Ruby is helping, too. More people than ever recognize the incredible power in the super-advanced string and text manipulation techniques. Doubters should go and read Jeffrey Friedl's book Mastering Regular Expressions. So should non-doubters.

5.It's multi-platform. It runs on Linux and other UNIX variants, the various Windows platforms, BeOS, and even MS-DOS. If my memory serves me, there's an Amiga version.

6.It's derivative. This is a good thing? Outside of the literary world, yes, it is.

Isaac Newton said, "If I have seen farther than others, it is because I stood on the shoulders of giants." Ruby certainly has stood on the shoulders of giants. It borrows features from Smalltalk, CLU, Lisp, C, C++, Perl, Kornshell, and others. The principles I see at work are: 1. Don't reinvent the wheel. 2. Don't fix what isn't broken. 3. Finally, and especially: Leverage people's existing knowledge. You understand files and pipes in UNIX? Fine, you can use that knowledge. You spent two years learning all the printf specifiers? Don't worry, you can still use printf. You know Perl's regex handling? Good, then you've almost learned Ruby's.

7.It's innovative. Is this in contradiction to #7 above? Well, partly; every coin has two sides. Some of Ruby's features are truly innovative, like the very useful concept of the mix-in. Maybe some of these features will be borrowed in turn by future languages. (Note: A reader has pointed out to me that LISP had mix-ins at least as far back as 1979. That's purely ignorance on my part; I shall have to find a better example, and make sure of it.)

8.It's a Very High-Level Language (VHLL). This is subject to debate, because this term is not in widespread use, and its meaning is even more disputable than that of OOP. When I say this, I mean that Ruby can handle complex data structures and complex operations on them with relatively few instructions, in accordance with what some call the Principle of Least Effort.

9.It has a smart garbage collector. Routines like malloc and free are only last night's bad dream. You don't even have to call destructors. Enough said.

10.It's a scripting language. Don't make the mistake of thinking it isn't powerful Because of this. It's not a toy. It's a full-fledged language that happens to make it easy to do traditional scripting operations like running external programs, Examining system resources, using pipes, capturing output, and so on.

11. it's versatile. It can do the things that Kornshell does well and the things that C does well. You want to write a quick ten-line hack to do a one-time task, or a Wrapper for some legacy programs? Fine. You want to write a web server, a CGI, or a chess program? Again, fine.

12. It's thread-capable. You can write multi-threaded applications with a simple API. Yes, even on MS-DOS.

13.It's open-source. You want to look at the source code? Go ahead. Want to Suggest a patch? Go ahead. You want to connect with a knowledgeable and helpful User community, including the language creator himself? You can.

14.It's intuitive. The learning curve is low, and once you get over the first hump, you start to "guess" how things work and your guesses are often correct. Ruby Endeavors to follow the Principle of Least Astonishment (or Surprise).

15. It has an exception mechanism. Like Java and C++, Ruby understands Exceptions. This means less messing with return codes, fewer nested if statements, Less spaghetti logic and better error handling.

16. It has an advanced Array class. Arrays are dynamic; you don't have to declare their size at compile-time as in, say, Pascal. You don't have to allocate memory for them as in C, C++, or Java. They're objects, so you don't have to keep up with their length; it's virtually impossible to "walk off the end" of an array as you might in C. Want to process them by index? By element? Process them backwards? Print them? There are methods for all these. Want to use an array as a set, a stack, or a queue? There are methods for these operations, too. Want to use an array as a lookup table? That's a trick question; you don't have to, since we have hashes for that.

17 Its extensible. You can write external libraries in Ruby or in C. In addition, you can modify the existing classes and objects at will, on the fly.
18 It encourages literate programming. You can embed comments in your code which the Ruby documentation tool can extract and manipulate. (Real fans of literate programming may think this is pretty rudimentary.)
19. It uses punctuation and capitalization creatively. A method returning a Boolean result (though Ruby doesn't call it that) is typically ended with a question mark, and the more destructive, data-modifying methods are named with an exclamation point. Simple, informative, and intuitive. All constants, including class names, start with capital letters. All object attributes start with an @ sign. This has the pragmatism of the old "Hungarian notation" without the eye-jarring ugliness.
20. Reserved words aren't. It's perfectly allowable to use an identifier that is a So-called "reserved word" as long as the parser doesn't perceive an amibiguity. This is a breath of fresh air.
21. It allows iterators. Among other things, this makes it possible to pass blocks of code to your objects in such a way that the block is called for each item in the array, list, tree, or whatever. This is a powerful technique that is worth exploring at great length.
22. It has safety and security features. Ruby borrows Perl's concept of tainting and allows different levels of control (levels of paranoia?) by means of the $SAFE variable. This is especially good for CGI programs that people will try to subvert in order to crack the web server.
23. It has no pointers. Like Java, and with a grudging nod to C++, Ruby does not have the concept of a pointer; there is no indirection, no pointer arithmetic, and none of the headaches that go with the syntax and the debugging of pointers. Of course, this means that real nuts-and-bolts system programming is more difficult, such as accessing a control-status register for a device; but that can always be done in a C library. (Just as C programmers drop into assembly when necessary, Ruby programmers drop into C when they have to!)
24. It pays attention to detail. Synonyms and aliases abound. You can't remember whether to say size or length for a string or an array? Either one works. For ranges, is it begin and end, or firstand last? Take your pick. You spell it indices, and your evil twin spells it indexes? They both work.
25. It has a flexible syntax. Parentheses in method calls can usually be omitted, as can commas between parameters. Perl-style quotes allow arrays of strings without all the quotation marks and commas. The return keyword can be omitted.
26. It has a rich set of libraries. There is support for threads, sockets, limited object persistence, CGI programs, server-side executables, DB files, and more. There is some support for Tk, with more on the way.
27. It has a debugger. In a perfect world, we wouldn't need debuggers. This is not a perfect world.
28. It can be used interactively. Conceivably it could be used as a sort of "Kornshell squared." (This is the most contested item on this page, and I am forced to concede that Ruby is not really good as a shell. I still maintain, though, that a Ruby-based shell would be a good thing.)
29. It is concise. There are no superfluous keywords such as Pascal's begin, then after if, do after while. Variables need not be declared, as they do not have types. Return types need not be specified for methods. The return keyword is not needed; a method will return the last evaluated expression. On the other hand... it is not so cryptic as C or Perl.
30. It is expression-oriented. You can easily say things like x = if a<0 then b else

c end.

31. It is laced with syntax sugar. (To paraphrase Mary Poppins: A spoonful of syntax sugar helps the semantic medicine go down.) If you want to iterate over an array x by saying for a in x, you can. If you want to say a += b instead of a = a + b, you can. Most operators are really just methods with short, intuitive names and a more convenient syntax.

***32. It has* operator overloading.** If I am not mistaken, this originated long ago in SNOBOL, but was popularized more recently by C++. It can be overdone or misused, but it can be nice to have. Additionally, Ruby defines the assignment version of an operator automagically; if you define +, you get += as a bonus.

***33. It* has infinite-precision integer arithmetic.** Who cares about short, int, long? Just use a Bignum. Admit it, you always wanted to find the factorial of 365. Now you can.

34. It has an exponentiation operator. In the old days, we used this in BASIC and FORTRAN. But then we learned Pascal and C, and learned how evil this operator was. (We were told we didn't even know how the evaluation was done -- did it use logarithms? Iteration? How efficient was it?) But then, do we really care? If so, we can rewrite it ourselves. If not, Ruby has the good old **operator you loved as a child. Enjoy it.

35. It has powerful string handling. If you want to search, substitute, justify, format, trim, delimit, interpose, or tokenize, you can probably use one of the built-in methods. If not, you can build on them to produce what you need.

36. It has few exceptions to its rules. The syntax and semantics of Ruby are more self-consistent than most languages. Every language has oddities, and every rule has exceptions; but Ruby has fewer than you might expect.

Reference: http://rubyhacker.com/

[Que-89]: What is Polymorphism?how it is achieved in ruby?

[Ans]: Polymorphism - the provision of a single interface to entities of different types Polymorphism is one of the fundamental features of object oriented programming,

but what exactly does it mean? At its core, in Ruby, it means being able to send the same message to different objects and get different results. Let's look at a few different ways to achieve this.

Inheritance

One way we can achieve polymorphism is through inheritance. Let's use the template method to create a simple file parser.

First let's create a GenericParser class that has a parse method. Since this is a template the only thing this method will do is raise an exception:

```
class GenericParser
def parse
raise NotImplementedError, 'You must implement the parse method'
end
end
```

Now we need to make a JsonParser class that inherits from GenericParser:

```
class JsonParser < GenericParser
def parse
puts 'An instance of the JsonParser class received the parse message'
```

```
end
end
```
Let's create an XmlParser to inherit from the GenericParser as well:
```
class XmlParser < GenericParser
def parse
puts 'An instance of the XmlParser class received the parse message'
end
end
```
Now let's run a script and take a look at how it behaves:
```
puts 'Using the XmlParser'
parser = XmlParser.new
parser.parse
puts 'Using the JsonParser'
parser = JsonParser.new
parser.parse
```
The resulting output looks like this:
```
Using the XmlParser
An instance of the XmlParser class received the parse message

Using the JsonParser
An instance of the JsonParser class received the parse message
```
Notice how the code behaves differently depending on which child class receives the parsemethod. Both the XML and JSON parsers modify GenericParser's behavior, which raises an exception.

[Que-90]: What is Duck Typing?

[Ans]: In statically typed languages, runtime polymorphism is more difficult to achieve. Fortunately, with Ruby we can use duck typing. We'll use our XML and JSON parsers again for this example, minus the inheritance:
```
class XmlParser def parse puts 'An instance of the XmlParser class received the parse message' end end class JsonParser def parse puts 'An instance of the JsonParser class received the parse message'
end end
```
Now we'll build a generic parser that sends the parse message to a parser that it receives as an argument:
```
class GenericParser
def parse(parser)
parser.parse
end
end
```
Now we have a nice example of duck typing at work. Notice how
the parse method accepts a variable called parser. The only thing required for this
to work is the parser object has to respond to the parse message and luckily both of our parsers do that!
Let's put together a script to see it in action:
ethod overriding, in object oriented programming, is a language feature that allows a subclass

to provide a specific implementation of a method that is already provided by one of its
superclasses. The implementation in the subclassoverrides (replaces) the implementation in
the superclass.
Here's an example p037xmtdovride.rb:

```
1. class A
2. def a
3. puts 'In class A'
4. end
5. end
7. class B < A
8. def a
9. puts 'In class B'
10. end
11.end
12.
13.b = B.new
14.b.a
```

The method a in class B overrides the method a in class A.

Usage of super

The way super handles arguments is as follows:

•When you invoke super with no arguments Ruby sends a message to the parent of
the current object, asking it to invoke a method of the same name as the method invoking super. It automatically forwards the arguments that were passed to the method from which it's called.

•Called with an empty argument list - super()-it sends no arguments to the higher-up
method, even if arguments were passed to the current method.

•Called with specific arguments - super(a, b, c) - it sends exactly those arguments.

[Que-91]:What is Active record?

]Ans]:-There are many reasons why Active Record is the smart choice
Simplified configuration and default assumptions (Convention over Configuration).
Associations among objects.
Automated mapping b/w tables and classes and b/w columns and attributes.
Data Validations.
Callbacks
Inheritance hierarchies.
Direct manipulation of data as well as schema objects.
Database abstraction through adapters.
Logging support.
Migration support.
Active Record integrated in other emerging frameworks like Merb.

[Que-92]: What is passanger?
[Ans]:-Easy and robust deployment of ruby on rails app on apache and ngix web servers' passenger is an intermediate to run the ruby language in Linux server.

[Que-93]: What is the difference between form_for and form_tag?
[Ans]:-form_tag and form for both are used to submit the form and its elements. The main difference between these two is the way of managing objects related to that particular model is different.
form_for
——-
We should use "form_for" tag for a specific model
It performs the "standard http post" which is having fields related to active record (model)
objects
form_tag:
——
It creates a form as a normal form. form_tag also performs the "standard http post" without
any model backed and has normal fields. This is mainly used when specific data need to be
submitted via form.
It just creates a form tag and it is best used for non-model forms.
Example:
<% form_tag '/articles' do -%>
<%= text_field_tag "article", "firstname" %>
<% end -%>

[Que-94]: What is the difference between include and extend?
[Ans]:- include makes the module's methods available to the instance of a class, while
extend makes these methods available to the class itself.
When you use include, the module's methods are added to the instances of the class. The log
method is:
Not available at the class level
Available at the instance level
Not available at the class level again
–
When you use extend, the module's methods are added to the class itself. The log method is:
Available at the class level.
Not available at the instance level.
Available at the class level.

[Que-95]: What is asset pipeline?
[Ans]:-.asset pipeline which enables proper organization of CSS and JavaScript.

[Que-96]: What is rails sweeper?

[Ans]:-. One sweeper can observe many Models, and any controller can have multiple sweepers

[Que-97]: What is the difference between the Rails version 2 and 3?
[Ans]:-. (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) A new .rb file in config/ folder, named as application.rb (Which has everything that
previously environment.rbhad)
* (4) Change in SQL Structure: Model.where(:activated => true)
* (5) All the mailer script will now be in app/mailers folder, earlier we kept inside app/models.
* (6) Rails3-UJS support. for links and forms to work as AJAX, instead of writing complex lines
of code, we write :remote => true
* (7) HTML 5 support.
* (8) Changes in the model based validation syntax: validates :name, :presence => true
* (9) Ability to install windows/ruby/jruby/development/production specific gems to Gemfile.
group :production do
gem 'will_paginate'
end.

[Que-98]: What is the Newest approach for find(:all) in Rails 3?
[Ans]:- Model. where(:activated => true)

[Que-99]: How does nil and false differ?
[Ans]:- nil cannot be a value, where as a false can be a value
A method returns true or false in case of a predicate, otherwise nil is returned.
false is a boolean data type, where as nil is not.
nil is an object for NilClass, where as false is an object of for FalseClass
[Que-100]: What are the new features of Rails4?
[Ans]:- 1. Ruby Versions
2. 'Gemfile'
3. 'Threadsafe' by Default
4. No More vendor/plugins
5. New Testing Directories
6. Strong Parameters
7. Renamed Callback
10. Queuing system
13. Cache Digests (Russian Doll Caching)
14. Turbolinks

[Que-101]: Difference between Validations, Callbacks and Observers?
[Ans]:- **Validations** allow you to ensure that only valid data is stored in your database.
Example: validates_presence_of :user_name, :password

validates_numericality_of :value
We 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)
Observers are 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.

[Que-102]: Different types of joins in SQL?
[Ans]:-Inner Join - Inner join creates a new result table by combining column values of two
tables (A and B) based upon the join-predicate. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row. The result of the join can be defined as the outcome of first taking the Cartesian product (or Cross join) of all records in the tables (combining every record in table A with every record in table B)—then return all records which satisfy the join predicate
Inner joins is further classified as equi-joins, as natural joins, or as cross-joins.
Equi-join
An equi-join, also known as an equijoin, is a specific type of comparator-based join, or theta join, that uses only equality comparisons in the join-predicate. Using other comparison operators (such as <) disqualifies a join as an equi-join
Natural join
A natural join offers a further specialization of equi-joins. The join predicate arises implicitly by comparing all columns in both tables that have the same column-names in the joined tables. The resulting joined table contains only one column for each pair of equally-named columns.
Cross join
CROSS JOIN returns the Cartesian product of rows from tables in the join. In other words, it will produce rows which combine each row from the first table with each row from the second table
Outer Join
An outer join does not require each record in the two joined tables to have a matching record. The joined table retains each record—even if no other matching record exists. Outer joins subdivide further into left outer joins, right outer joins, and full outer joins, depending on which table(s) one retains the rows from (left, right, or both).

Left outer join
The result of a left outer join (or simply left join) for table A and B always contains all records of the "left" table (A), even if the join-condition does not find any matching record in the "right" table (B). This means that if the ON clause matches 0 (zero) records in B, the join will still return a row in the result—but with NULL in each column from B. This means that a left outer join returns all the values from the left table, plus matched values from the right table (or NULL in case of no matching join predicate). If the right table returns one row and the left table returns more than one matching row for it, the values in the right table will be repeated for each distinct row on the left table.
Right outer join
A right outer join (or right join) closely resembles a left outer join, except with the treatment of the tables reversed. Every row from the "right" table (B) will appear in the joined table at least once. If no matching row from the "left" table (A) exists, NULL will appear in columns from A for those records that have no match in B. A right outer join returns all the values from the right table and matched values from the left table (NULL in case of no matching join predicate). Result of right outer joins can also be produced with left outer joins by switching the table order
Full outer join
A full outer join combines the effect of applying both left and right outer joins. Where records in the FULL OUTER JOINed tables do not match, the result set will have NULL values for every column of the table that lacks a matching row. For those records that do match, a single row will be produced in the result set (containing fields populated from both tables).
Self-join
A self-join is joining a table to itself.

[Que-103]: Difference between InnoDB and MyISAM storage engine in MySQL?
[Ans]:- InnoDB: A transaction-safe (ACID compliant) storage engine for MySQL that hascommit, rollback, and crash-recovery capabilities to protect user data. InnoDB row-level locking (without escalation to coarser granularity locks) and Oracle-style consistent nonlocking reads increase multi-user concurrency and performance. InnoDB stores user data in clustered indexes to reduce I/O for common queries based on primary keys. To maintain data integrity, InnoDB also supports FOREIGN KEY referential-integrity constraints. InnoDB is the default storage engine as of MySQL 5.5.5.
MyISAM: The MySQL storage engine that is used the most in Web, data warehousing, and other application environments. MyISAM is supported in all MySQL configurations, and is the default storage engine prior to MySQL 5.5.5.
For example, customer bank records might be grouped by customer in InnoDB but by transaction date with MyISAM, so InnoDB would likely require fewer disk seeks and less RAM to retrieve and cache a customer account history
The major differences between these two storage engines are :
• InnoDB supports transactions which is not supported by tables which use MyISAM storage engine.
• InnoDB has row-level locking, relational integrity i.e. supports foreign keys, which is not possible in MyISAM.
• InnoDB ‘s performance for high volume data cannot be beaten by any other storage engines available.
Tables created in MyISAM are known to have higher speed compared to tables in InnoDB. But since InnoDB supports volume, transactions, integrity it’s always a better option which you are dealing with a larger database. A single database can have tables of different storage engines.

To check engines you can use command:
mysql> Show engines;
You can change the engine while creating the table by command:
CREATE TABLE test name varchar(30) ENGINE = InnoDB;
It is also possible to convert from one engine to the other by command:
ALTER TABLE my_table ENGINE=new_engine;
When you will execute the above command, complete process will be as such - the table will get locked, dumped to a tmp space, then rebuilt with the new engine. Also you will be losing innodb-only info (foreign keys, etc.) and features if you're going to MyISAM.

[Que-104]: Difference between Argument and Parameter in ruby on rails?
[Ans]:- A parameter represents a value that the method expects you to pass when you call it.An argument represents the value you pass to a method parameter when you call the method. The calling code supplies the arguments when it calls the method.
In simple words, parameters appear in method definitions; arguments appear in method calls.
For example, in below method, variables param1 and param2 are the parameters
def foo_method(param1, param2):
.....
end
while calling the method, arg1 and arg2 are the arguments
foo_method(arg1, arg2)

[Que-105]: Plugins?
[Ans]:- Plugins are the principal mechanism to reuse code. The open-source community is at full-bloom when it comes about plugins. The Rails Plugins site has a really good list of such plugins from the community

[Que-106]: What is a Proc?
[Ans]: Procs, short for procedures, act similar to blocks, but can be saved as variables and reused. Think of them as blocks you can call over and over again on multiple arrays.

[Que-107]: What is a module?

[Ans]: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.

[Que-107]: How will you implement a singleton pattern?
[Ans]: 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.

[Que-108]: How will you implement an observer pattern?
[Ans]: Let's review first what an observer pattern is all about.
The observer pattern (sometimes known as publish/subscribe) is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods. It is mainly used to implement distributed event handling systems.
You might have used them in other programming languages as listener objects. You use them whenever a button is clicked on the screen and a method gets called automatically.

As in the case of the singleton pattern, the observer pattern is also implemented by mixing in a module.
In the Ruby implementation, the notifying class mixes in the Observable module, which provides the methods for managing the associated observer objects.
And, the observers must implement the update method to receive notifications.
Here's an example. Say you want to send an SMS alert to users if a company stock drops then you can do something like this:

```
require "observer"
require "observer"
  class Ticker # Periodically fetch a stock price
    include Observable
          attr_accessor :price
    def initialize symbol, price
      @symbol = symbol
          @price = price
          end
```

```
        def run
      lastPrice = nil
      loop do
        @price = @price+Random.rand(11)
        print "Current price: #{price}\n"
        if @price != lastPrice
          changed               # notify observers
          lastPrice = @price
           notify_observers(Time.now, @price)
         end
       end
    end
  end

  class Warner
     def initialize ticker
     ticker.add_observer(self)   # all warners are observers
  end
end

class SMSAlert < Warner
   def update time, price       # callback for observer
      print "--- #{time.to_s}: SMS Alert for price: #{price}\n"
   end
end

class EmailAlert < Warner
   def update time, price       # callback for observer
      print "+++ #{time.to_s}: Email Alert Price changed to #{price}\n"
   end
 end
```

Now let's initialize the classes and run them:

```
ticker = Ticker.new("MSFT", 307)
SMSAlert.new(ticker)
EmailAlert.new(ticker)
 ticker.run
 Current price: 312
 --- 2012-02-22 01:26:04 -0800: SMS Alert for price: 312
 +++ 2012-02-22 01:26:04 -0800: Email Alert Price changed to 312
 Current price: 321
 --- 2012-02-22 01:26:04 -0800: SMS Alert for price: 321
 +++ 2012-02-22 01:26:04 -0800: Email Alert Price changed to 321
 Current price: 323
 --- 2012-02-22 01:26:04 -0800: SMS Alert for price: 323
 +++ 2012-02-22 01:26:04 -0800: Email Alert Price changed to 323
 Current price: 329
 --- 2012-02-22 01:26:04 -0800: SMS Alert for price: 329
 +++ 2012-02-22 01:26:04 -0800: Email Alert Price changed to 329
```

[Que-108]: What is the purpose of environment.rb and application.rb file?
[**Ans]:**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.
How can you define a constant?
Create a new file as shown below under: config/initializers/my_constants.rb
COLORS = ['white', 'red', 'green']

[Que-109]: How can you secure a rails application?
[**Ans]:**Rails has a lot of in-built capabilities to deal with common web-security issues.

> SQL Injection

> Cross-Site

> Session fixation and Session hijacking

> Captcha

[Que-110]: Can you tell me a few good community resources for Rails?
[Ans]: Stackoverflow, Github, various Meetup groups