The following list of Technical questions and answers with examples
and references

General

1. What is ruby?

   A scripting language more powerful than Perl and more object-oriented than Python.

   Invented in 1993 with inspiration from Perl, Phyton, Smalltalk, Eiffel, Ada, and Lisp.
   Designed for programmer productivity and joy instead of machine efficiency.
   Follows the principle of least surprise, the language should minimize confusion.
   It's easy to work with and to love Ruby.

2. Advantages and Disadvantages of ruby

   Advantages of ruby:-

   http://www.unlimitednovelty.com/2012/06/ruby-is-faster-than-python-php-and-perl.html

   Elegant

   It is considered as elegant because it's *orthogonal*. That's a fancy way of saying that similar operations apply to similar operands.
   Simple example: + on integers adds them; on floating point numbers, ditto. On big integers, too. On strings, it concatenates them (which you'd kind of expect too). Now this is not a big deal for +, you kind of expect it from any decent programming language. But there are operations like map or filter, they work on lists (they should!) but they also work on arrays and in fact on anything that can be enumerated or iterated through.
   I like how array (or list) indexing works, you can use positive integer indexes to index from the start, or negative indexes to specify a position back from the end of the structure, you can specify a range to pull out a subset... this works for lists, arrays and (sub)strings too. It works on the right side of an assignment (=), it works on the left side too (you can assign to a substring, thus replacing part of a string). So you don't need a substring_replace function, you just leverage an existing, general concept.
   It's also nice that Ruby does closures (= code blocks) so you can specify a function simply by wrapping it in a pair of braces. There are places where it's appropriate to specify a function inline, and Ruby lets you do it conveniently.

   Powerful

   1>Object-oriented (everything is an object)
   Ruby is pure object-oriented language and everything appears to Ruby, as an object. Every value in Ruby is an object, even the most primitive things:

strings, numbers and even true and false. Even a class itself is an *object* that is an instance of the *Class* class.

2>It supports an extensive set of libraries providing built-in support for so many features that require additional programming in other languages

Ruby's built-in library provides you with a rich set of classes that form the foundation for your Ruby programs.There are classes for manipulating text(String),operating system services and abstractions (IO,File,Process,etc.),numbers(Integer,Fixnum, etc.), and so on.**Using these basic building blocks,you can build powerful Ruby programs.**

for example ,

```
object.clone
object.dup
object.private_methods

string.capitalize
string.chomp
string.reverse
```

3>Dynamic (duck typing)

Almost everything in Ruby is done at runtime.Types of variables and expressions are determined at runtime as are class and method definitions.You can even generate programs within programs and execute them.

http://www.tekniqal.com/2009/3/Duck-Typing-in-Ruby (check this also)

calling any method on any object, regardless of its class.

```
class Duck
  def quack
    puts "Quaaaaaack!"
  end

  def feathers
    puts "The duck has white and gray feathers."
  end
end

class Person
  def quack
    puts "The person imitates a duck."
  end

  def feathers
    puts "The person takes a feather from the ground and shows it."
  end
```

```
end

def in_the_forest d
  d.quack
  d.feathers
end

def game
  donald = Duck.new
  john = Person.new
 p john.type
  in_the_forest donald
  in_the_forest john
end
game
```

Out Put :

```
Person
Quaaaaaack!
The duck has white and gray feathers
the person imitates duck
feather from ground
```

Readable

1>The code comments itself
2>Leads to better naming practices
3>It is written in plain english so that everyone can understand it

Concise

1>Very expressive language
2>Less lines of code, higher productivity
e.g.

```
-199.abs # 199
"ruby is cool".length # 12
"Your Mom".index("u") # 2
"Nice Day!".downcase.split(//).sort.uniq.join # " !acdeiny"
say = "I love Ruby"
say['love'] = "*love*"
5.times { puts say }
```

=> "I *love* Ruby" "I *love* Ruby" "I *love* Ruby" "I *love* Ruby" "I *love* Ruby"

Disadvantages of ruby:-

1>Ruby is Slow
About 10x slower than Java

2>Ruby is New

- •Not many developers or customers
- •No common IDE
  - –Most coders use the Mac text-editor Textmate
  - –IDEs: RadRails, RubyMine, 3rd Rail, Netbeans
- •No common application server
  - –Passenger's Apache module has momentum

3. difference between ruby 1.8, 1.9

for this differentiation i used 1.8.7 and 1.9.3 ruby
link used http://slideshow.rubyforge.org/ruby19.html#1

Sr. No	Topic	example	ruby 1.8 output	ruby 1.9 output
1	Single characters strings	?c	99 (ASCII value of c)	“c”
2	String Index	"cat"[1]	97 (ASCII value of a)	“a”
3	{"a","b"} conversion to hash{a => b} (No longer support)	{1, 2}	{1 => 2}	SyntaxError: (irb):2: syntax error, unexpected ',', expecting tASSOC {1,2}
4	Array.to_s Now Contains Punctuation (use join)	[1,2,3].to_s	“123”	“[1,2,3]”
5	Colon No Longer Valid In When Statements (use semicolon, then or newline)	case 'a'; when /\w/: puts 'word'; end	word	SyntaxError: (irb):9: syntax error, unexpected ':', expecting keyword_then or ', ' or ';' or '\n'
6	Block Variables	i=0; [1,2,3].each	3	0

	Now Shadow Local Variables	{|i|}; i		
7	Hash.index Deprecated (use Hash.key)	{1=>2}.index(2)	1	1 warning: Hash#index is deprecated; use Hash#key
8	Fixnum.to_sym Now Gone	5.to_sym	no error	NoMethodError: undefined method `to_sym' for 5:Fixnum
9	Hash Keys Now Unordered (Order is insertation order)	{:a=>"a", :c=>"c", :b=>"b"}	{:a=>"a", :b=>"b", :c=>"c"}	{:a=>"a", :c=>"c", :b=>"b"}
10	BasicObject More Brutal Than BlankSlate (use ::Math::PI)	**for 1.8** require 'blankslate' class C < BlankSlate; def f; Math::PI; end; end; C.new.f **for 1.9** class C < BasicObject; def f; Math::PI; end; end; C.new.f	true 3.14159265358979	NameError: uninitialized constant C::Math
11	instance_methods Now an Array of Symbols (Replace instance_methods.include? with method_defined?)	{}.methods.sort	[“!”, “!=”, “!~”, “<=>”, “==”,....]	[:!, :!=, :!~, :<=>, :==, ……..]
12	Alternate		example	example

	Syntax for Symbol as Hash Keys		{:a => b} redirect_to :action => show	{a: b} redirect_to action: show
13	Block Local Variables	[4,2].each {|value; t| t=value*value; puts t}	syntax error	16 4 => [4, 2]
14	Lambda Shorthand		p = lambda {|a,b,c| a+b+c} puts p.call(1,2,3) o/p - 6	p = -> a,b,c {a+b+c} puts p.(1,2,3) puts p[1,2,3] o/p - 6

4. What is Rails?
   •Web application framework
   •Open Source (MIT license)
   •Based on a existing application (Basecamp)
   •Provides common needs:
      –Routing, sessions
      –Database storage
      –Business logic
      –Generate HTML/XML/CSS/Ajax
      –Testing

5. who uses rails?
   •Internally: hp, intel, NASA
   •Externally:
      •style.mtv.com
      •www.getsatisfaction.com
      •www.basecamp.com
      •www.yellowpages.com
      •www.twitter.com
      •www.yelloyello.nl

6. Advantages and Disadvantages of rails
   Advantages of Rails:-
   1>Convention over configuration
   •Table and foreign key naming
      –Tables are named in plurals (users, orders, etc.)
      –Foreign key naming: user_id

•Default locations
–MVC, Tests, Languages, Plugins
•Naming
–Class names: CamelCase
–Files: lowercase_underscored.rb

2>Don't Repeat Yourself (DRY)

•Everything is defined in a single, unambiguous place
•Easier to find code
–Only need to look once
–Can stop looking when found
–Well defined places for most items
•Much easier to maintain code
–Faster to change
–Less inconsistency bugs

3>Object Relational Mapping(ORM)

Class map to the table, object map to single row of the table, and object attributes map to the columns of the row.

4>Model View Controller

•Model
–Object relationships (users, orders)
•Controller
–Business logic (perform a payment)
•View
–Visual representation (generate HTML/XML)

5>Reuse of code

•Gems and plugins, more then 1300
–For authentication, pagination, testing, etc.
•Git allows easy forking and merging
•Github website allows you to give back
•Acts as a portfolio

6>Agile practices

•Iterations (branch often, merge often)
•Test all the time (TDD, BDD)
Stories > Tests > Code > Continuous IntegrationStory

7>Security

•Rails prevents SQL injection attacks
•Rails prevents Cross Site Scripting (XSS)
–With html_escape in view: <%=h comment.body %>

Disadvantages of Rails:-

1>Rails is inefficient

•Rails uses a lot of memory, up to 150MB per instance
•Requires more application servers

2>

- •Lots of moving parts
  - –Rails, apps, gems, plugins
  - –Application server, webserver
- •Deployment via scripts
  - –Via the Capistrano tool
  - –Easy to break something

7. Difference between rails 2 and rails 3?
8. REST

(referred peepcode video on REST)
Rest stands for Representational State Transfer . It's the way of constructing URL's.
- Why use REST in RAILS?

Pre-Rest URL

```
:url => {:controller => "pages",
         :action => "home",
         :id => 42 }
```

Pre-Rest URL's contained the action ,

```
/assets/show/12
/assets/edit/12
/assets/update/12
/assets/destroy/12
```

In REST , a single URL identifies a resource,

```
/assets/12 => is used for show,update,delete.
```

Here the thing that is going to change is the http method.
i.e,
POST
GET
PUT
DELETE.

- To create a resource on the server, use POST.
- To retrieve a resource, use GET.
- To change the state of a resource or to update it, use PUT.
- To remove or delete a resource, use DELETE.

REST URL's are simpler.

```
map.resources :pages
```

Here , methods are dynamically generate
pages_url # => http://localhost:300/pages

pages_path # => /pages
pages_path(1) # => /pages/1

so,

GET	"/pages"	# => index
POST	"/pages"	#=> create
GET	"/pages/1"	#=> show
PUT	"pages/1"	#=> update
DELETE	"pages/1"	#=> destroy
GET	"/pages/new"	#=> new
GET	"/pages/1/edit	#=> edit

Forms:
POST => Create
<form action = "/people"
class = "new_person"
id = "new_person"
method = "post" >

PUT => Update
<form action = "/people/1"
class = "edit_person"
id = "edit_person"
method = "post" >
In case of update rails automatically gives a hidden field with a method of put.
<input name = "_method"
type = "hidden"
value = "put" >

- But how is this create or update method handled in rails?
form_for(@person)
POST to create if it is a new_record?
PUT to update otherwise.

http://bitworking.org/news/373/An-Introduction-to-REST
http://www.infoq.com/articles/rest-introduction

http://www.ibm.com/developerworks/java/library/j-cb08016/
http://tomayko.com/writings/rest-to-my-wife
http://edgeguides.rubyonrails.org/getting_started.html

9. why it is different from other languages ?
10. What is compiler ?

Compiler is for compiling the program/s, compiler converts the code in the language which is understood by the machine. Compiler will parse all the code at one go and display all the errors in compilation cycle. Compiler will always dependent on compiled code (byte code / object code). Chaining a single line of code will require recompilation.
In compilation development time is more but run time is less

11. what is interpreter ?

Interpreter will interpret the code line by line, as soon as interpreter found error it will display the error and halt the interpretation cycle. Interpreter will always dependent on present code.
In interpretation development time is less but run time could be more.

12. and difference between them
    please refer above two points
13. What are object oriented concepts?
14. what are Http verbs?
    http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
15. what is MVC ?
16. What is cloud ?name the service provider of cloud
17. Difference between gems and plugins

Topic	Gem	Plugin
Definition	A Gem is a packaged Ruby application using the packaging system defined by RubyGems.	A Rails plugin is a packaged Ruby application that extends the core Rails framework.
Example	Has a name (e.g. rake) and a version (e.g. 0.4.16).Rails is also installed as a Gem.	Has a name (e.g. Goldspike) and a version (e.g. 1.1.1).
Scope	Installed in the Ruby or JRuby installation and is available to all applications run using that interpreter.	Installed in a specific Rails app only.
Packaging System	Rails gem is installed in jruby-1.0\\lib\\ruby\\gems\\1.8\\gems\\rails-1.2.3 as: <DIR> bin <DIR> builtin 68,465 CHANGELOG <DIR> configs <DIR> dispatches	Goldspike plugin is installed invendor\\plugins\\rails-integration directory of the application as: 7,089 build.xml 1,141 LICENSE.txt <DIR> plugins 6,675 pom.xml

	<DIR> doc <DIR> environments 307 fresh_rakefile <DIR> helpers <DIR> html <DIR> lib 1,072 MIT-LICENSE 11,969 Rakefile 8,001 README The lib directory contains all the gem source code.	1,447 README <DIR> samples plugins/goldspike directory consists of 24 init.rb 25 install.rb <DIR> lib 549 Rakefile 536 README <DIR> tasks <DIR> test The lib directory contains all the plugin source code.
Management	Gems are managed on your computer using the gem command. You can install, remove and query gem packages using the gem command.	Plugins can be installed using after creating a Rails application and then invoking the command ruby script\\plugin install [url or name of plugin]
Load Path	gem command adds the lib directory of your gem to the load path of Ruby.	Rails adds the lib directory of plugin in your application's load path.
Specific Features		• Bundled Rake tasks that get automatically loaded into your Rakefile • An installation hook (install.rb) • An application initialization hook for the plugin to inject itself (init.rb) • ability to bundle additional generators (for the script/generate script)

Publish Format	A Gem may be published as Plugin, for example ActiveRecord-JDBC.	A Plugin cannot be published as Gem.

18. master n slave architecture? How rails handles this?

12.agile development?

13. HTTP status?

http://www.sitepoint.com/restful-rails-part-i/

HTTP is a request response protocol.

200 OK

You will get from a GET, PUT or a DELETE request. It means that the request checked out and the appropriate action has been taken

201 Created

Notifies you that the POST command successfully created the posted object.

404 Not found

Means the request resource wasn't found. You can get this from a GET, PUT or DELETE.

406 Not Acceptable

The verb isn't allowed at the resources you requested

500 Internal Server Error

Something went horribly wrong

14. Difference between GET and POST.

GET sends data to server in a query string . Post sends data via HTTP headers.

- **Fundamental Difference is probably the Visibility** - GET request is sent via the URL string (appended to the URI with a question-mark as separator), which is visible whereas POST request is encapsulated in the body of the HTTP request and can't be seen.
- **Length** - Since, GET request goes via URL, so it has a limitation for its length. It can't be more than 255 characters long (though this is browser dependent, but usually the max is 255 characters only). Whereas no such maximum length limitation holds for the POST request for the obvious reason that it becomes a part of the body of the HTTP request and there is no size limitation for the body of an HTTP request/response.
- **Performance** - GET request is comparatively faster as it's relatively simpler to create a GET request and the time spent in the encapsulation of the POST request in the HTTP body is saved in this case. In addition, the maximum length restriction

facilitates better optimization of GET implementation.

- **Type of Data** - GET request is sent via URL string and as we all know that URL can be text-only, so GET can carry only text data whereas POST has no such restriction and it can carry both text as well as binary data.
- **Caching/Bookmarking** - again for the obvious reason that a GET request is nothing but an URL hence it can be cached as well as Bookmarked. No such luxuries with a POST request.
- **FORM Default** - GET is the default method of the HTML FORM element. To submit a FORM using POST method, we need to specify the method attribute and give it the value "POST".
- **Data Set** - GET requests are restricted to use ASCII characters only whereas POST requests can use the 'enctype' attribute with a value "multipart/form-data" to use the Universal Multiple-Octet Coded Character Set (UCS).

When to use what?

The "get" method should be used when the form is idempotent (i.e., causes no side-effects). Many database searches have no visible side-effects and make ideal applications for the "get" method.

If the service associated with the processing of a form causes side effects (for example, if the form modifies a database or subscription to a service), the "post" method should be used.

Ruby
http://mtnwestrubyconf2008.confreaks.com/11farley.html
(http://ruby.runpaint.org/toc)

1. What is meta programing ?
2. Memory leak in ruby
3. Garbage collection in ruby
4. static method in ruby?why it is used ? explain with examples
5. thread
6. struct

Struct is a convenient way to bundle a number of attributes together,using accessor methods,without having to write an explicit class.(http://www.slideshare.net/metaskills/ruby-struct)

```
>>Customer = Struct.new(:name,:age)
```

```
>>manager = Customer.new("John",45)
=> #<struct Customer name="John", age=45>

Sub class :
        class Customer < Struct.new(:name,:age)
         def age
          "Your age is #{age}"
         end
        end
>> manager = Customer.new("Kushal",34)
>>puts manager.age
```

EQUIVALENT:

Customer = Struct.new(:name,:address) ⇒

```
class Customer
 attr_accessor :name
 attr_accessor :address

 def initialize(name, address)
  @name = name
  @address = address
 end
end
```

http://www.paulbutcher.com/2007/09/hidden-feature-in-rubys-struct/

7. moudle mix (include extends)
8. Singleton method

A **singleton method** is a method that is defined only for a single object . (**per object not per class**).

For example,

```
class Animal
 def heed using after llo
  "Hello!"
 end
end
>>john = Animal.new
```

```
>>john.hello   # Not a singleton method
=> Hello!
>>cat = Animal.new
def cat.hello
  "meow"
 end
>>cat.hello     #  Singleton method
=> Meow
>>dog = Animal.new
 def dog.hello
  "Bow Bow"
 end
>>dog.hello    # Singleton method
=>Bow Bow
```

9. Different types of ruby frameworks

http://www.thepixelart.com/ruby-frameworks-what-are-they-and-why-should-you-choose-one-which-one/

10. Difference between include and extend

http://ficate.com/explaining-include-and-extend?utm_source=rubyweekly&utm_medium=email

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

extend - adds the specified module's methods and constants to the target's metaclass (e.g. the singleton class) e.g.

- if you call Klazz.extend(Mod), now Klazz has Mod's methods (as class methods)
- if you call obj.extend(Mod), now obj has Mod's methods (as instance methods), but no other instance of of obj.class has those methods.
- extend is a public method

include - By default, it mixes in the specified module's methods as instance methods in the target module/class. e.g.

- if you call class Klazz; include Mod; end;, now all instances of Klazz have access to Mod's methods (as instance methods)
- include is a private method, because it's intended to be called from within the container class/module.

However, modules very often *override* include's behavior by monkey-patching the included method.

```
module Animal
  def speech
   puts " hello"
 end
end
class Cat
   include Animal
end
class Dog
  extend Animal
end
> c = Cat.new
> c.speech
=> hello
>Cat.speech  # NoMethodError: undefined method `speech'

>Dog.speech
>hello
>Dog.new.speech # NoMethodError: undefined method `speech' for #
```

11. What are array ,string, constants methods
12. What is duck typing?
        calling any method on any object, regardless of its class.

```
class Duck
  def quack
    puts "Quaaaaaack!"
  end

  def feathers
    puts "The duck has white and gray feathers."
  end
end

class Person
  def quack
    puts "The person imitates a duck."
  end

  def feathers
    puts "The person takes a feather from the ground and shows it."
```

```
  end
end

def in_the_forest d
  d.quack
  d.feathers
end

def game
  donald = Duck.new
  john = Person.new
 p john.type
  in_the_forest donald
  in_the_forest john
end

game
```

13. What is singleton class?
14. Define Class, Object.
15. what is instance ,static and class methods?

A class method is a method that resides at the class level. On the opposite, an instance method is a method that resides at the object level.

Examples,
Class Method - find (ActiveRecord)
Instance Method - save(ActiveRecord)

Static method == Class method (in ruby)

16. what is the difference between instance method and class method?
17. What is module?

(http://ruby.about.com/od/rubysbasicfeatures/a/module_3.htm)

In any programming language, you need some way to prevent naming clashes. This is especially true when you're using at least one third party library, where you have no control over the naming scheme. In the past, naming conventions would be used. It's not uncommon to see C programs with names like *mylibrary_some_function*. However, Ruby has a more powerful solution: modules.
A module is just a way to organize all of your methods, classes, constants, etc into a little section where they can't interfere with anything in another part of the program. So, library A might have a class called *Car*, but library B can also have a class called *Car* and they won't interfere as long as they're in different modules.

First think about what happens with name clashes in Ruby. If you define a method, then define another method with the same name, what happens? Ruby happily forgets about the first method and uses the second. Name clashes can be very confusing in Ruby, since there are no real alarm bells to go off.
The following code illustrates this plainly. There are two modules, unimaginatively called *ModuleA* and *ModuleB*, each containing a method of the same name.

```
#!/usr/bin/env ruby

module ModuleA
  def ModuleA.message
    puts "Hello from ModuleA"
  end
end

module ModuleB
  def ModuleB.message
    puts "Hello from ModuleB"
  end
end

ModuleA::message
ModuleB::message
```

The first thing that you'll notice is *ModuleA* or *ModuleB* has to be prepended to the method names in the definitions. The second thing you'll notice is that when you call these methods, you need to use a fully qualified name using the *::* operator.
A fully qualified name is a module name and either method or class name separated by the *::* operator. It unambiguously identifies a single method or class in a module. There's a shortcut around this though, but it may be counterproductive.
To prevent yourself from having to type module names all over the places, you can *include* a module into the current namespace. When you include a module, it will take all the symbols (methods, classes and constants) from the module and place them into the current namespace. It's as if they were defined in this namespace, and you no longer have to use the module name when referring to them.
You may be tempted to start doing this more often. After all, including modules into the global namespace will save on a lot of typing. This puts you right back to where you started though, if you start doing this to multiple modules, there can be potential name clashes. I usually only include modules into the current namespace if I'm using that module's vocabulary frequently, and avoid including other modules.
To include a module, use the *include* keyword.

```
#!/usr/bin/env ruby

# If your program uses methods from Math often,
```

```
# you may want to include Math

radius = 23

# Without include
a = Math.PI * radius ** 2

# With include
include Math
a = PI * radius ** 2
```

18. what is require and include ?difference ?

The require method does what include does in most other programming languages: run another file. It also tracks what you've required in the past and won't require the same file twice. To run another file without this added functionality, you can use the load method.
The include method takes all the methods from another module and includes them into the current module. This is a language-level thing as opposed to a file-level thing as with require. The include method is the primary way to "extend" classes with other modules (usually referred to as mix-ins). For example, if your class defines the method "each", you can include the mixin module Enumerable and it can act as a collection. This can be confusing as the include verb is used very differently in other languages.

19. Three main feature of ruby?

http://ruby-challenge.rubylearning.org/

1. Metaprogramming – this allows to build great custom DSLs (Shoulda, ActiveRecord, RSpec etc etc).
2. Prototype-based programming – this just switches the way you think when comparing to more mainstream languages (C#, Java). It allows to do things in so much easier and better way.
3. Lambdas/Blocks – needles to say that this feature is probably most commonly used in Ruby community which also simplifies the code very heavily. Even though Python has similar idiom, it is not the same at all and lambdas/blocks are considered to be one of unique Ruby features.

20. what is Block, proc, and lamda?

http://www.skorks.com/2010/05/ruby-procs-and-lambdas-and-the-difference-between-them/

There are three ways to create a Proc object in Ruby. Well, actually there are four ways, but the last one is implicit. Here they are:

1. **Using Proc.new**. This is the standard way to create any object, you simply need to pass in a block and you will get back a Proc object which will run the code in the

block when you invoke its call method.

```
proc_object = Proc.new { puts "I am a proc object" }
proc_object.call
alan@alan-ubuntu-vm:~/tmp$ ruby a.rb
I am a proc object
```

2. **Using the proc method in the Kernel module**. Being in the Kernel module, this method is globally available. It is basically equivalent to *Proc.new* in Ruby 1.9, but not in Ruby 1.8. In Ruby 1.8 the *proc* method is equivalent to *lambda*. As you may have guessed there is a difference between procs and lambdas (*see below*), so you need to be aware of whether you're using Ruby 1.8 or 1.9. If you're using 1.9 there is a way to find out if you're dealing with a proc or a lambda. The Proc object *lambda?* method:

```
proc_object = proc { puts "Hello from inside the proc" }
proc_object.call
puts "Is proc_object a lambda - #{proc_object.lambda?}"
alan@alan-ubuntu-vm:~/tmp$ rvm use 1.9.1
Using ruby 1.9.1 p378
alan@alan-ubuntu-vm:~/tmp$ ruby a.rb
Hello from inside the proc
Is proc_object a lambda - false
```

If you're in 1.8 this method will not exist at all. By the way, did you notice how I had to switch Rubies and was able to do so painlessly through the awesome power of RVM, hope you're already using it. As far as I am concerned, if you need to switch between different versions of Ruby, don't bother with the *proc* method, just use *Proc.new* and you will never have an issue.

3. **Using the Kernel lambda method**. Once again, this is globally available being in the Kernel module, using this method will create a Proc object, but it will be a lambda as opposed to a proc (*if you really need to know what the differences are at this point you can skip to the explanation :)*).

```
proc_object = lambda {puts &quot;Hello from inside the proc&quot;}
proc_object.call
puts &quot;Is proc_object a lambda - #{proc_object.lambda?}&amp;quot;
alan@alan-ubuntu-vm:~/tmp$ ruby a.rb
Hello from inside the proc
Is proc_object a lambda - true
```

4. **The implicit way**. When you write a method which will accept a block, there are two ways you can go about it. The first is to simply use the *yield* keyword inside your method. If there is a *yield* in a method, but there is no block you will get an error.

```
def my_method
 puts &quot;hello method&quot;
 yield
end
```

my_method **{puts &**quot;hello block**&**quot;**}**
my_method
alan@alan-ubuntu-vm:~/tmp$ ruby a.rb
hello method
hello block
hello method
a.rb:7:in `my_method': no block given (yield) (LocalJumpError)
from a.rb:11:in `<main>'

The other way is to use a special parameter in your method parameter list. This special parameter can have the same name but you will need to precede it with an ampersand. The block you pass into this method will then be associated with this parameter, you will infact no longer have a block – it will be a Proc object. You will simply need to call the Proc *call* method to execute the code in the block you passed in. The difference is, you can now return your block (*or rather the Proc object*) as the return value of the method, thereby getting your hands on a Proc object without having to use any of the above three methods explicitly.

def my_method**(&**amp;my_block**)**
puts &quot;hello method**&**quot;
my_block.call
my_block
end

block_var = my_method **{puts &**quot;hello block**&**quot;**}**
block_var.call
alan@alan-ubuntu-vm:~/tmp$ ruby a.rb
hello method
hello block
hello block

Those are all the different ways to get a Proc object in Ruby (*either a proc or a lambda*).

21. what is Mixin?

Ruby does not suppoprt mutiple inheritance directly but Ruby Modules have another, wonderful use. At a stroke, they pretty much eliminate the need for multiple inheritance, providing a facility called a *mixin*.

Mixins give you a wonderfully controlled way of adding functionality to classes. However, their true power comes out when the code in the mixin starts to interact with code in the class that uses it.

Let us examine the following sample code to gain an understand of mixin:

```
module A
  def a1
  end
```

```
  def a2
  end
end
module B
  def b1
  end
  def b2
  end
end

class Sample
include A
include B
  def s1
  end
end

samp=Sample.new
samp.a1
samp.a2
samp.b1
samp.b2
samp.s1
```

Module A consists of the methods a1 and a2. Module B consists of the methods b1 and b2. The class Sample includes both modules A and B. The class Sample can access all four methods, namely, a1, a2, b1, and b2. Therefore, you can see that the class Sample inherits from both the modules. Thus you can say the class Sample shows multiple inheritance or a *mixin*.

22. What are instance variables?
23. Difference between class and modules ?
    -> Modules are collections of methods and constants. They cannot generate instances. -> Classes may generate instances (objects), and have per-instance state (instance variables).
    -> Modules may be mixed in to classes and other modules. The mixed-in module's constants and methods blend into that class's own, augmenting the class's functionality. Classes, however, cannot be mixed in to anything.
    -> A class may inherit from another class, but not from a module.
    -> A module may not inherit from anything.

27. What is self?
    Self returns current object it also called as default reciver

it exists in the instance variable
ex:
class Myclass
self
end
=>Myclass

28. What is the difference between load and require and include?
    Load :loads a source file every time it is called
    require : loads the source file one time
29. What is the difference between clone and dup?
    Clone copies the entire object as is, including the any state information the object may contain. While the duplicate call (.dup) does not copy all of the state while copying the basic data.
    `clone' copies everything; internal state, singleton methods, etc.
    `dup' copies object contents only (plus taintness status).

    obj1 = []
    obj1.freeze
    obj2 = obj1.clone
    obj3 = obj1.dup
    p obj1.frozen?, obj2.frozen?, obj3.frozen?
    # => true, true, false

    So that makes me feel that clone has to do more work than dup does. Hence dup might be more beneficial for performance compared to clone (unless you really need to use clone).

30. Questions on Object's methods:
    http://ruby-doc.org/docs/ProgrammingRuby/html/ref_c_object.html
    ["==", "===", "=~", "__id__", "__send__", "class", "clone", "display", "dup", "enum_for", "eql?", "equal?", "extend", "freeze", "frozen?", "hash", "id", "inspect", "instance_eval", "instance_exec", "instance_of?", "instance_variable_defined?", "instance_variable_get", "instance_variable_set", "instance_variables", "is_a?", "kind_of?", "method", "methods", "nil?", "object_id", "private_methods", "protected_methods", "public_methods", "respond_to?", "send", "singleton_methods", "taint", "tainted?", "tap", "to_a", "to_enum", "to_s", "type", "untaint"]

31. Difference between “include” and “require”.

load will load and execute a Ruby program (*.rb).

require loads Ruby programs as well, but will also load binary Ruby extension modules (shared libraries or DLLs). In addition, require ensures that a feature is never loaded more than once.

32. What are tainted objects ?

Any Ruby object derived from some external source (for example, a string read from a file, or an environment variable) is automatically marked as being tainted.

33. What is an OpenStruct?
(http://ruby.about.com/od/rubysbasicfeatures/f/structs.htm)
An OpenStruct is like a Struct, except you don't have to define a list of attributes on construction. Instead, the first time you try to assign to an attribute that doesn't yet exist, the OpenStruct class will create it for you. This gives you all the benefits of hashes without the limitations of a Struct.

```
#!/usr/bin/env ruby
require 'ostruct'

i = OpenStruct.new
i.name = "Watermelons"
i.id = 1337
i.price = "$5.40"

puts i.name
puts i.id
puts i.price
```

There is, however, one huge caveat when using these classes. The attributes in these examples were chosen very carefully. If you run the second example, you'll see a warning about using the *object_id* method instead of id, and the id returned is wrong. What is this all about?
Well, these are normal objects, after all. You still have to worry about the methods it already inherited from the Object class. In the first example, everything appears to have worked. However, since Struct creates new methods for each attribute, it's overwritten *Object#id* on the object.
The second example is a little more tricky. When you try to use any assignment to an attribute, it first checks to see if that method exists. Since *Object#id* exists, it assumes is can assign to it and tries to. This doesn't work at all, and the value you get when you read the id attribute is incorrect.

34. What is Enumeration?

35.write a program to find largest number in array

and. Smallest too

36.why ruby is dynamic?
37. What is reflection in rails ?
38. private , public and protected methods?
http://www.skorks.com/2010/04/ruby-access-control-are-private-and-protected-methods-only-a-guideline/

39. Difference between chomp and chop method ?

Chomp returns a new string with seperators like \n \r removed from end of the line .

Chop removes last character from the end of the line .

40. What is the difference between gsub and sub methods for Ruby Strings?

The difference is that sub only replaces the first occurrence of the pattern specified, whereas gsub does it for all occurrences (that is, it replaces globally).

```
>> "hello".sub('l', '*')

=> "he*lo"

>> "hello".gsub('l', '*')

=> "he**o"
```

Rails

why rails? How is it different from other frame works?

What are associations?types and explain with examples
In Rails, an *association* is a connection between two Active Record models.
Rails supports six types of associations:

- belongs_to
- has_one

- has_many
- has_many :through
- has_one :through
- has_and_belongs_to_many

Explain has_many :through .
This sets up many-to-many connection with another model . This requires creation of third model .
For example ,
Orders and Albums:
One album can have many orders . Similarly one order can contain many albums.
This requires us to create a third model LineItem.

```
class Album < ActiveRecord::Base
  has_many :line_items
  has_many :orders, through => :line_items
end

class Order < ActiveRecord::Base
  has_many :line_items
  has_many :albums, through => :line_items
end
```

Explain has_and_belongs_to_many.
It creates a direct many-to-many connection with another model , with no intervening model (but still need to create a join table in database).

When should we use has_and_belongs_to_many and has_many :through?
i. Use has_many :through relationship if you need to work with relationship model (Third join model) as an independent entity . For example LineItem model can be used for billing .
ii. Use has_many :through if you need validations , callbacks , or extra attributes on the join model . Else use has_and_belongs_to_many for simple joins .

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

1 Creating an Object
- before_validation
- after_validation

- before_save
- around_save
- before_create
- around_create
- after_create
- after_save

2 Updating an Object

- before_validation
- after_validation
- before_save
- around_save
- before_update
- around_update
- after_update
- after_save

3 Destroying an Object

- before_destroy
- around_destroy
- after_destroy

what are filters?how many types of filters?
Filters are methods that are run before, after or “around” a controller action.
. A common before filter is one which requires that a user is logged in for an action to be run. You can define the filter method this way:

```
class ApplicationController < ActionController::Base
  before_filter :require_login

  private

  def require_login
    unless logged_in?
      flash[:error] = "You must be logged in to access this section"
      redirect_to new_login_url # halts request cycle
    end
  end

  # The logged_in? method simply returns true if the user is logged
  # in and false otherwise. It does this by "booleanizing" the
  # current_user method we created previously using a double ! operator.
  # Note that this is not common in Ruby and is discouraged unless you
  # really mean to convert something into true or false.
  def logged_in?
    !!current_user
```

```
  end
end
```

What are observers?
imagine a User model where we want to send an email every time a new user is created. Because sending emails is not directly related to our model's purpose, we should create an observer to contain the code implementing this functionality.

```
$ rails generate observer User
class UserObserver < ActiveRecord::Observer
end


class UserObserver < ActiveRecord::Observer

  def after_create(model)
    # code to send confirmation email...
  end
end
```

what is name scope?explain with examples
difference between new, create!, build in rails . And should we use them ?

What are active support?
How are nested forms handled?
what are helpers?
Helpers are modules that provide methods which are automatically usable in your view . The purpose of a helper is to simplify the view . Its best if the view file is to be kept short .
what are partials?
what are rails components ?

what is action dispatchers?
It is a new rails component which lives in "Action Pack ".
It is responsible for all processing involved in dispatch requests: request and response handling , HTTP status code , file uploads , URL & parameters passing , session storage and so on .
what is ActiveRecord?
what is ORM?
ActionController::Dispatcher?
Difference between redirect and render?

Render: It will display the template without calling action (controller method) .

Redirect: It calls the action in the conroller.

Difference between collection and members in routing ?

Difference between form_for and form_tag?

What are rails initializers?

What is respond_to?

What is bundler?

Bundler is used to install and include the gems needed by the rails application. It is a ruby library which makes dependency problem easier .

What is RACK?

http://rubylearning.com/blog/a-quick-introduction-to-rack/

http://tekpub.com/view/rack/1

http://jasonseifer.com/2009/04/08/32-rack-resources-to-get-you-started

Rack aims to provide a minimal API for connecting web servers supporting Ruby (like WEBrick, Mongrel etc.) and Ruby web frameworks (like Rails, Sinatra etc.)./13. Rs. 125.00
Free shipping

The premise of Rack is simple – it just allows you to easily deal with HTTP requests.

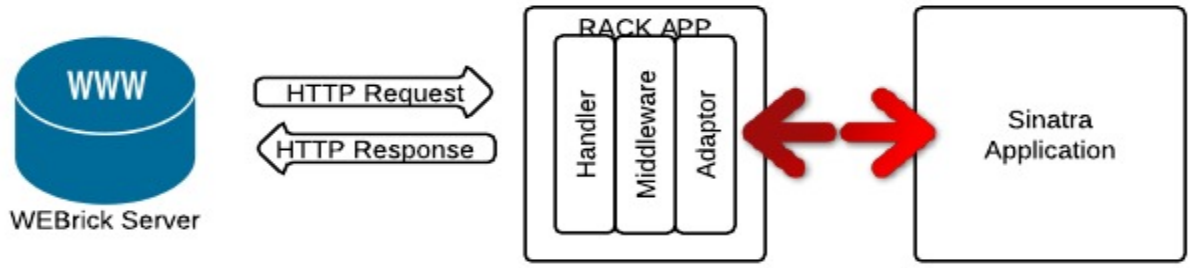
A Rack application is a Ruby object that has a call method, which has a single argument, the *environment*, (corresponding to a HTTP request) and returns an array of 3 elements, *status*,*headers* and *body* (corresponding to a HTTP response). Rack includes *handlers* that connect Rack to all these web application servers (WEBrick, Mongrel etc.).

Rack includes *adapters* that connect Rack to various web frameworks (Sinatra, Rails etc.).

Between the server and the framework, Rack can be customized to your applications needs using *middleware*. The fundamental idea behind Rack middleware is – come between the calling client and the server, process the HTTP request before sending it to the server, and processing the HTTP response before returning it to the client.

So basically rack middleware is used to process the HTTP request and response . Rack accepts a request from web server and provide a response.

It provides a rack middleware which is a filter used to intercept a request and handle the response differently.



what is scaffolding?
what all consisting in Controllers?

what are validations?name the validation helpers
Validations are used so that only valid data are stored in database .There are several ways to validate data before it is saved into your database, including native database constraints, client-side validations, controller-level validations, and model-level validations

1. acceptance

Validates that a checkbox on the user interface was checked when a form was submitted. This is typically used when the user needs to agree to your application's terms of service, confirm reading some text, or any similar concept.

```
class Person < ActiveRecord::Base
  validates :terms_of_service, :acceptance => true
end
```

2 validates_associated

You should use this helper when your model has associations with other models and they also need to be validated.

```
class Library < ActiveRecord::Base
  has_many :books
  validates_associated :books
end
```

3 confirmation

You should use this helper when you have two text fields that should receive exactly the same content. For example, you may want to confirm an email address or a password. This validation creates a virtual attribute whose name is the name of the field that has to be confirmed with "_confirmation" appended.

```
class Person < ActiveRecord::Base
  validates :email, :confirmation => true
```

```
end
<%= text_field :person, :email %>
<%= text_field :person, :email_confirmation %>

class Person < ActiveRecord::Base
  validates :email, :confirmation => true
  validates :email_confirmation, :presence => true
end
```

4 exclusion

```
class Account < ActiveRecord::Base
  validates :subdomain, :exclusion => { :in => %w(www us ca jp),
    :message => "Subdomain %{value} is reserved." }
end
```

5 format

This helper validates the attributes' values by testing whether they match a given regular expression, which is specified using the :with option.

```
class Product < ActiveRecord::Base
  validates :legacy_code, :format => { :with => /\A[a-zA-Z]+\z/,
    :message => "Only letters allowed" }
end
```

7 length

This helper validates the length of the attributes' values. It provides a variety of options, so you can specify length constraints in different ways:

```
class Person < ActiveRecord::Base
  validates :name, :length => { :minimum => 2 }
  validates :bio, :length => { :maximum => 500 }
  validates :password, :length => { :in => 6..20 }
  validates :registration_number, :length => { :is => 6 }
end
```

8 numericality

This helper validates that your attributes have only numeric values. By default, it will match an optional sign followed by an integral or floating point number. To specify that only integral numbers are allowed set :only_integer to true.

```
class Player < ActiveRecord::Base
  validates :points, :numericality => true
  validates :games_played, :numericality => { :only_integer => true }
end
```

9 presence

This helper validates that the specified attributes are not empty. It uses the blank? method to check if the value is either nil or a blank string, that is, a string that is either empty or consists of whitespace.

```
class Person < ActiveRecord::Base
  validates :name, :login, :email, :presence => true
end
```

10 uniqueness

This helper validates that the attribute's value is unique right before the object gets saved. It does not create a uniqueness constraint in the database, so it may happen that two different database connections create two records with the same value for a column that you intend to be unique. To avoid that, you must create a unique index in your database.

```
class Account < ActiveRecord::Base
  validates :email, :uniqueness => true
end
```

- Difference between between development mode and production mode ?

caching is disabled by default for development and test, and enabled for production.(config.action_controller.perform_caching) .
In development mode as we send a url model , controller gets loaded everytime . Views get loaded only if changes are made in the view file (view files create a template with timestamp everytime we make changes to it).

what are testing tools supported by rails?
Rspec , cucumber etc .

tell me different types of testing?

what is polymorphic associations?
http://railscasts.com/episodes/154-polymorphic-association

In polymorphic associations , a model can belong to more than one model.
For example , a comment model can belong to different models like Article , Events , Picture .
Suppose if we dont create a polymorphic association in this case then we need to create different comment models for Article as say , ArticleComment , for Event model EventComment and so on . So polymorphic is more efficient way of dealing with this kind of problems .

```
class Comment < ActiveRecord::Base
  belongs_to :commentable, :polymorphic => true
```

end

In the Article, Picture and Event classes we define the relationship like this.

has_many :comments, :as => :commentable

How can you change the default configuration for Model names ?
Explain migrations .
Rails generate migration name

```
create_table “table” do |t|
t.string :name
t.text :description
end
```

Migration methods:
Table migration methods:
- rename_table(table, new_name)
- drop_table(table)
- create_table(table.options) do |t|
….columns…
end

column migration methods:
- add_column(table,column,type,options)
ex. add_column(“admin_users”,”username”,:string,:limit => 25)
- remove_column(table,column)
- rename_column(table,column,new_name)
- change_column(table,column,type,options).

how do you read a file in ruby?
File.open(filename, r+)

NamedScope?
what is the difference between string and symbol?
http://www.rubytips.org/2008/01/26/what-is-a-ruby-symbol-symbols-explained/
What is single table inheritance?
http://code.alexreisner.com/articles/single-table-inheritance-in-rails.html
What is eager loading?
What is lazy loading ?
What is auto loading ?
What is XSS ?

Why changes made in lib file does not reflect until we restart server in development mode ?
What is protect_from forgery?
what is difference between form_for and form_tag?
What is convention over configuration? How can we configure any convention in rails ? examples?

Database

1)How do you store xml data?
2)Normalization
3)Master and slave
4)joins

5)difference between joins and include

i. http://stackoverflow.com/questions/38549/sql-difference-between-inner-and-outer-join
ii. http://www.codinghorror.com/blog/2007/10/a-visual-explanation-of-sql-joins.html

git

what is difference between git and svn
what is conflict?

Javascript

Different types of javascript frameworks?

HTML5

New

CSS

Jquery

Web-servers

Open Source Operating Systems
UBUNTU
UNIX commands

Programming questions and logical questions