__/__/____

UNIT-1. (SL).

Package Management with Ruby gems.

* Ruby gems is a package manager for the ruby programming lang.
* It provides a standard format for distributing ruby programs and libraries
* It acts as a tool design to easily manage the installation of gems & a server for distributing them.
* The interface of Rubygems is a command line tool called gem
* It can install & manage the libraries (the gem).
* Ruby gems integrate with ruby runtime loader.
* The ruby runtime loader helps to find and load installed gems from standarized library folders.
* The public repo is most commonly used for gem management (Github)

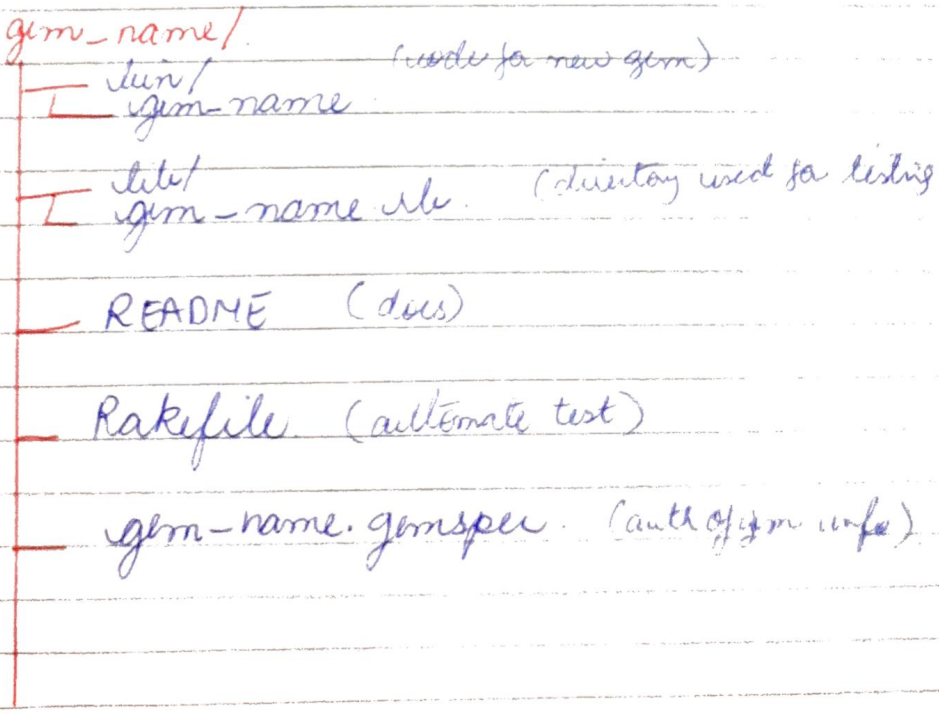
Gem file → bundle install → Gem file.lock
bundle update.

Gem file: min dependencies for app to work correctly.

Gem file.lock: Actual libraries used to satisfy dependencies

Structure

* Every gem contains a name, version and platforms
* Gems work only on ruby designed for a particular platform
* It is based on CPU architecture OS type and version
* Each gem consists of
  1) code
  2) documentation
  3) gem Specification
* The code organisation follows the following structure for a gem which is called gem-name



__/__/__

Working :-

gem command [edit]

The gem command is used to build, upload, download & install gem packages.

gem usage [edit]

Ruby gem is very similar to apt-get, portage yum & npm in functionality

Installation:
gem install mygem

Uninstallation
gem uninstall mygem

Listing installed gems
gem list

Listing available gems, eg:
gem list -r

Create RDOC documentation for all gem
gem rdoc -all.

Adding a trusted certificate
gem cert -a

__/__/____

Download but do not install a gem.

gem fetch my-gem.

Search available gems. eg:-
gem search STRING_remote
gem package building [edit]

RUBY AND WEB.

* Ruby cant be called a web lang at all.
* Even so, web apps & web tools in general are among the most common uses of Ruby

Writing CGI Script

```
#! /usr/bin/ruby.
puts "HTTP/1.0 200 OK".
puts "content-type:text/html\n\n".
puts "<html><body>
        This is a test.
      </body></html>"
```

* The server test.cgi is requested from a web browser,
* Then the web server looks for test.cgi on the website

* Then it executes using the ruby interpreter
* The ruby Script returns a basic HTTP header & then returns a basic HTML document using cgi.rb.
* Ruby comes with a special library called cgi that enables more sophisticated interactions than those with the preceding CGI Scripts

Creating CGI Script that uses CGI.

```
#!/usr/bin/ruby

require 'cgi'
cgi = CGI.new
puts cgi.header
puts "<html><body> This is a test
      </body></html>"
```

Form processing.

```
#!/usr/bin/ruby

require 'cgi'
cgi = CGI.new
cgi['first name'] # => ["Shreya"]
cgi['Last name'] # => ["Malagi"]
```

CREATING FORMS IN HTML.

* CGI contains a huge no of methods used to create HTML.
* You will find one method per tag.
* In order to enable these methods you must create a CGI object by calling CGI.new

```
#!/usr/bin/ruby

require "cgi"
cgi = CGI.new("html5").
cgi.out {
  cgi.html {
    cgi.head {"\n" + cgi.title("This is a test")}} +
    cgi.body {"\n" +
      cgi.form {"\n" +
        cgi.hr +
        cgi.h1 {"A form:"} + "\n" +
        cgi.textarea("get_text") + "\n" +
        cgi.br +
        cgi.submit
      }
    }
  }
}
```

__/__/____

Output

content-type : text/html
content-length : 302

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
<HTML><HEAD>
<TITLE>This is a test</TITLE></HEAD>
<BODY>
<FORM METHOD = "post" ENCTYE =
                "application/x-www-form-urlencoded"
<HR>
<H1> A Form: </H1>
<TEXTAREA COLS = "70" NAME =
"get-text" ROWS = "10"></TETAREA>
<BR>
<INPUT TYPE = "submit">
</FORM>
</BOBY>
</HTML
```

COOKIES

* Cookies are client side files that contain user info
* cookie ends depending on the lifetime you set for it
* You dont need to start cookies as it is stored in your Local Machine

__/__/____

* The official maximum cookie size is 4KB.
* A cookie is not depended on Session.
* There is no function named unset cookie()

Sessions.

* Sessions are server side files which contains user info
* A Session ends when the user closes his browser.
* In PHP, before using $_SESSION, you have to write Session_start(); likewise for other lang's.
* Within Session you can store as much as data you like. The only limits you can reach is the maximum memory a script can consume at one time which is 128MB by default
* Session is dependent on cookie.
* Session_destroy() is used to destroy all registered data or to unset.

WEBSERVERS.

* Every website sits on a PC known as Web server.

* Server is always connected to the Internet.
* It is given a unique address made up of series of 4 no's b/w 0 & 256
* Eg 68.178.157.132 (or) 68.122.35.127

Guidelines for choosing webservers.
* Should be able to handle Server side Scripting
* Security features
* OS compatibility.
* Website building software.
* Web application you are going to run
* your budget

Choices of Webservers.
* Apache:-
  It is an HTTP server whose main purpose is to maintain an open source HTTP Server for all latest OS.
* Microsoft IIS:- IIS - Internet Information server is a web appln. developed by Microsoft compatible with Windows. It is 2nd most widely used server after the Apache HTTP Server.

Both of the apps are offered as a part of web hosting packages most of the. They differ in Server Side Scripting features like auth, access control mech, CGI support & ASP

SOAP AND WEBSERVICES

* SOAP - Simple Object Access Protocol.
* It is an XML based industry standard protocol for designing & developing web services.
* As it is XML based, it is platform & lang independent
* Scripted SOAP web services allows a service Now admin to create custom SOAP web services.
* Use the SOAP web services whenever possible since they are simple to implement & maintain.

LR works for SOAP services

* Using LR, we can directly send the SOAP requests to the webservice
* We do it by bypassing the UI & JSP pages.

Scripting

* Web service dont have UI
* we dep on dev team to get SOAP data.
* 3 ways LR can be used for scripting
  1) Using soap req
  2) " WSDL wizard.
  3) " web_custom req

1). Using SOAP request

i) Open VuGen and create a Web Services script

ii) Create a soap request call in the action.

```
Soap_request("Step name =",
"URL =",
"SOAP Envelop =",
"Snapshot =",
"ResponseParam =",
LAST);
```

Step-Name: Text describing the web services step.
URL - URL of the web service.
SOAP Envelop - The field has XML string that is accepted by the web service.
Snapshot - Name of the .inf file that stores the snapshot of the step.
Response & Response Param - The field stores the response that comes back from a server.

__/__/____

RUBY TK

* Std GUI for ruby is TK.
* Started out for scripting GUI for TCL.
* Cross-platform GUI.
* TK runs on Windows, Mac, Linux etc.
* Basic component of TK-based app is called **widget**.
* component - also called **window**.
* TK apps follow **widget hierarchy**.
* Main widget — **root widget**
  * can be created for making a new instance of the TK root class.
* Most TK follow same cycle
  a) create widget
  b) place them in the Interface
  c) bind events associated with each widget to a method.
* 3 geometry managers
  a) place
  b) grid
  c) pack.

They are responsible for controlling the size & location of each of the widgets in the Interface.

Installation of Ruby TK.

* The ruby TK bindings are distributed with Ruby
* TK is a seperate Installation
* Windows users can download a single click TK installation Active State's Active Tcl.
* Mac and Linux users may not need to install it because there is a great chance that is already along with O.S.
* But if not, you can download prebuilt packages or get the source from Tcl developer exchange.

Simple TK Applications.

* A typical structure for Ruby/TK is to create a root window
* Add widgets to build UI.
* Then start the main event loop by calling TK.mainloop.

```
require 'tk'
root = TkRoot.new { title "Hi" }.
TkLabel.new(root) do
  text 'hi'
  pack { padx 15; pady 15; side "left" }
end.
Tk.mainloop
```

__/__/____

Ruby/TK Widget Classes

* There is a list of various Ruby/TK classes which can be used to create a desired GUI using Ruby/TK.
* _____ creates & manipulates _____ widgets

TK frame	TK messageBox
TK Button	TK Scrollbar
TK Label	TK Canvas
TK Entery	TK Scale
TK Check Button	TK text
TK Radio Button	TK toplevel
TK list Box	TK Spinbox
TK ComboBox	TK ProgressBar
TK Menu	TK DialogBox
TK MenuButton	

TK Binding Events / Event Handling

* Ruby/TK supports event loop which receives event from O.S.
* There are 2 things like button press, keystrokes, mouse movement, etc
* When new frame is created it has no diff event handling
* frames are not intended to be interactive

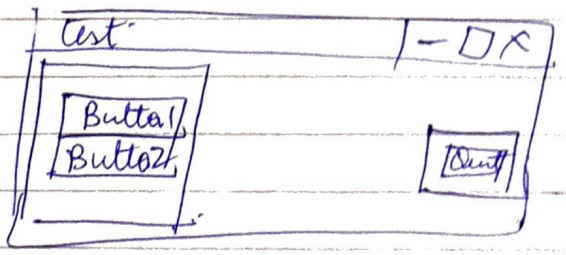
* Eg: to catch the button release event for the first mouse button on the same widget, you'd write.

```
some_widget.bind('<Button-Release-1>') {
    .... code block to handle the event...
}
```

Ex:

```
require 'tk'

f1 = TkFrame.new {
    relief 'sunken'
    borderwidth 3
    background "red"
    padx 15
    pady 20
    pack ('side' => 'left')
}


f2 = TkFrame.new {
    relief 'groove'
    borderwidth 1
    background "yellow"
    padx 10
    pady 10
    pack ('side' => 'right')
```

```
TK.Button.new(f1){
  text 'Button1'
  command {puts "push button1!!\n"}
  pack ('fill' => 'x')
}

TK.Button.new(f1){
  text 'Button2'
  command {puts "push button2!!\n"}
  pack ('fill' => 'x')
}

TK.Button.new(f2){
  text 'Quit'
  command 'exit'
  pack ('fill' => 'x')
}

TK.mainloop
```



CANVAS

* The canvas element is a part of HTML 5.
* It allows for dynamic, scriptable rendering of 2d shape and bitmap images.
* It is a low level procedural model that updates a bitmap and does not have a built in scene graph.
* But through WebGL it allows 3D shapes & images to be displayed.
* HTML 5 canvas also helps in making 2D games.
* A canvas consists of drawable design region defined in HTML code with height & width attributes.
* Javascript code may access the canvas through a full set of drawing functions
* Thus allowing for dynamically generated graphics.

__/__/____

Program.

```
<canvas id="example" width="200"
        height="200">.
```

This text is displayed in your browser does not support HTML5 canvas.

using Javascript, you can draw on the canvas

```
var example : document.getElementById.
              ("Example");.
Var context : example.getContext("2d");.

context.fillStyle="red";.
context.fillRect(30,30,50,50).
```

→ This code draws a red rectangle on screen
→ It also proved save() & store()

* SCROLL MY.

⇒ In computer displays, filmaking, television production, & other kinetic display scrolling is sliding text, images or video across

__/__/____

a monitor or display, vertically
or horizontally.

* Swolling does not change the
layout of the text or pictures
but moulds the user view across
what is apparently a large image
that is not wholly seen

S.L. - UNIT-2

* Everything in Ruby - Object.
* Type of all Ruby Variables.

VALUE

pointer to ruby variables — Intermediate value eg Fixnum

Extending Ruby with C

Why Make a C extention?

- To access C libraries from Ruby
- To run CPU-intensive algorithms

your first C extintion

myextension.c:

```
#include <ruby.h>
void Init_myextension()
{
  printf("hello world!\n");
```

extconf.rb:

```
require 'mkmf'
$CFLAGS += '-std=gnu99'
create_makefile 'myextension'
```

Run these commands

```
ruby extconf.rb
make
```

__/__/____

test.rb

```
require_relative 'myextension'
```

* Defining a Ruby class.

What we want to do:

```
class MyClass
  def foo(arg1)
  end
end
```

Equivalent code in C:

```
#include <ruby.h>
VALUE foo (VALUE self, VALUE arg1)
{
  return Qnil;
}

void Init_myextension()
{
  VALUE cMyClass = rb_define_class(
  "MyClass", rb_cObject);
  rb_define_method (cMyClass,
   "foo", foo, 1);
}
```

__/__/____

* Defining class in Ruby.

Syntax :-

```
class Class_name
end.
```

Eg :-

```
class Animal.
@@ type_of_animal = 4
@@ no_of_animal = 3.
end.
```

* Creating Objects in Ruby.

Syntax :-

```
object_name = Class_name.new
```

Eg :-

```
class Box.          # name of class.
@@ No_of_colors = 3.   # variable of class.
end.

# Two objects of Box class.
sbox = Box.new
nbox = Box.new
```

2 obj of Box class

class name — dot operator — new method

* Defining Method in Ruby

Syntax:

```
def method_name.
# statements to be executed.
end.
```

Eg:

```
class GFG.          - class
def geeks           - method
puts "Hello Geeks"  - res print
end.                - end of method
end                 - end of class
```

```
obj = GFG.new.      - creating obj
obj.geeks           - calling method using obj.
```

Output:

Hello geeks.

* Passing Parameters to New Method.

```
class Vehicle.
def initialize (id, color, name).
@veh_id = id.
@veh_color = color.
@veh_name = name
```

```
#displaying values
puts "ID is : #@veh_id"
puts "Color is : #@veh_color"
puts "Name is : #@veh_name"
puts "\n"
end
end
# creating objs & passing parameters
xveh = Vehicle.new("1", "Red", "ABC")
yveh = Vehicle.new("2", "Black", "XYZ")
```

Output :-

ID is : 1	ID is : 2
Color is : Red	Color is : Black
Name is : ABC	Name is : XYZ

Here,

Vehicle = classname
def = keyword used to define initialize method in Ruby.
Id, color, name = parameters in initialize method
@veh_id, @veh@color = local variables in " ".

→ The parameters in new method is always enclosed in double quotes.

→ Initialize method is like a constructor whenever new objects are created, initialize method is called.

__/__/____

Steps for building your own extention in Ruby.

Step 1 :- Create a file named extconf.rb with these contents

1. require 'mkmf'
2. create_header
3. create_makefile 'foobar'

Step 2 :-
Run the ruby extconf.rb which will create two files for you Makefile and extconf.h. Now dont change these files

Step 3
Now create foobar.c, this name comes from create_makefile line in extconf.rb. You can change it if you want but it has to match your .c filename to work.

Step 4.
Inside foobar.c add this.

```
1. #include "ruby.h"
2. #include "extconf.h"
3.
4. void Init_foobar().
5. {
6.     // Your C code goes here.
7. }
```

∴ This is the basic skeleton of your C extension.
→ Notice that you have to use the name you declared on ext/conf.rb thus the mood Init_. In this case Init_foobar

* // Wrapping C Structures in Ruby.

* We got the vendors library that controls the audio CD Jukebox units and we already ready to write it into Ruby.
* The vendors header file looks like this

```
typedef struct _cdjb {
  int status;
  int request;
  void *data;
  char pending;
  int Unit_id;
  void *stats;
} CDJukebox;

// Allocate a new CDPlayer structure & bring it Online
CDJukebox *CDPlayerNew(int unit_id);

// Deallocate when done
void CDPlayerDispose(CDJukebox **ree);
```

// Seek a disk, track & notify progress

void CDPlayerSeek(CDJukebox *ptr,
                  int disc,
                  int track,
                  void (*done)(CDJukebox
                  *ptr, int percent));

// Others
// Report a statistic
double CDPlayerAvgSeekTime(CDJukebox *ptr);

* C Datatype Wrapping

1) VALUE Data_Wrap_Struct(VALUE
klass, void (*mark)(), void (*free)(),
void *ptr);

* It wraps the given C datatype ptr
* registers the given garbage
  collection routines
* Returns a VALUE pointer to a
  genuine Ruby object
* The C type of resulting object
  is T_DATA & its Ruby class is klass

2) VALUE Data_Make_Struct(VALUE klass,
C type, void (*mark)(), void (*free)(),
~~void *ptr~~, C type *);

* Allocates a structure of the indicated type first.
* Then proceeds as Data_Wrap_Struct
* c-type is the name of C datatype that you are wrapping

3) Data_Get_Struct(VALUE obj, c-type, c-type*)
* Returns to the original pointer
* This macro is type-safe wrapper around the macro DATA_PTR(obj) which evaluates the pointer.

→ The object created via by Data_Wrap_Struct is a normal ruby obj
→ Except that, it has additional C data type that can't be accessed from Ruby.
→ But since, its a seperate thing, how do you get rid of it when the garbage collector claims the object?
→ What if you have to release some more resources?
→ So Ruby uses mark and sweep garbage collection scheme.
→ During this mark phase, Ruby looks for pointers to areas of memory
→ It marks these areas as "in use".
→ At the end of mark phase, all memory that is referenced will be marked, & orphaned areas

__/__/____

will not be marked.
→ To participate in Ruby's mark & sweep, you must define a routine to free your structure
→ A routine to mark any references from your structure to other structures
→ Both routines take a void pointer as a reference to your structure.
→ The mark routine will be called by the garbage collector during its "mark" phase.
→ If your structure references other ruby objects, then your mark func needs to identify this obj using rb_gc_mark(value)
→ If the structure doesn't reference other ruby objects, you can simply pass 0 as a function pointer

* MEMORY ALLOCATION:

* You might sometimes need to allocate memory in an extension that won't be used for obj storage
* Perhaps you have got a giant bitmap for a bloom filter or an image / whole chunk of little structures that ruby does not use directly

* In order to work correctly with the garbage collector, you should use the following memory allocation routines.
* These routines do a little bit more work than standard malloc.
* For instance, if ALLOC_N determines that it can't allocate the desired amount of memory, it will invoke the garbage collector to try to reclaim some space.
* It will raise a NoMemError if it can't handle the requested amount of memory is invalid.

API.

1) type ALLOC_N (c_type, n)
It allocates n c_type objects, where, c_type is the literal name of the C type and not a variable of that type.

2) type ALLOC (c_type)
Allocates a c_type and casts the result to a pointer of that type.

3) REALLOC (var, c_type, n).
Reallocates n c_type & assigns the result to var, a pointer to a c_type.

4) type ALLOCA_N (c-type n")
Allocates memory for n c-type on the stack --- this memory will be automatically freed when the function invokes ALLOCA_N returns

How is memory Allocated?
* There are two basic types of memory allocation.
* When you declare a variable or an instance of a structure, or class
* The memory for that object is allocated by O.S
* The name you declare for an obj can be used to access that block of memory

Steps to find Memory Leak?
* Check for any unused gems in the Gemfile and remove them
* Check the issue tracker of each gem still present in the Gemfile for reports of the memory leak
* Run Rubycop with the rubycop-performance extension
* Visually review the ruby code for possible memory leaks

__/__/____

Different Storage Allocation Strategies are:

1) Static allocation :- lays out storage for all data objects at compile time.

2) Stack allocation :- manages the run time storage as a stack.

3) Heap allocation :- allocates & deallocates storage as needed at run time from a data area known as heap.

* RUBY TYPE SYSTEMS :-

* Ruby is a dynamically typed lang,
* which means the Interpreter tries to infer the data types of variables & object properties at runtime.
* This generally leads to programs being more dynamic & fast to code & the Interpreter / compiler loading fast.
* Static Typing VS Dynamic Typing is an age old issues for programming languages.
* Statistically Typed ™ Lang are suitable for larger projects but are often less flexible.

* Dynamically Typed lang's allow for rapid development, but scaling teams & codebases with them can be difficult.
* OTL's implement Type checking options (PHP, Python)

RBS:-

* We defined a new language called RBS for type structures in Ruby 3.
* The signatures are written in .rbs file which is diff from ruby code.
* The benefit of having diff files is it does not require changing ruby code to start type checking.
* You can opt in type checking without changing any part of your work flow
* RBS is a language to describe the structure of Ruby program.
* It gives developers control of the code & what classes & methods are defined.
* The biggest benefit is that the type definition can be validated against both the implementation & execution

__/__/____

Key features in RBS.

* The development of a type system for a dynamically typed lang like Ruby differs from ordinal statistically typed lang.
* There a lot of ruby code in the world already & a type system for ruby should support as many of them as possible.
* We can show a snip char of ruby code & how we can write types for them.

Ducktyping

* It is a popular programming style among rubyists that assume an objet will respond to certain set of methods.
* The benfit of ducktyping is flexibility.
* It does not require inheritance, mixin or implement declarations.
* If an obj has speific method it work.
* The assumption is that this is hidden in the code, making code diff to read
* To accomodate duck typing we introduced interface types.
* An interface type represents a set of methods independent from concrete

__/__/____

classes & modules.

Non Uniformity

* Non uniformity is another code pattern of letting an expression have different types of values
* Its also popular in Ruby & introduced
  - When you define a local variable which stores instances of two different classes.
  - When you write an heterogeneous collection.
  - When you return two diff types of val from a method.

Ruby programming with Types.

1) Finding more bugs: We can detect an undefined method calls, an undefined constant reference, & more things a dynamic lang might have missed.

2) Nil Safety: Type checkers based on RBS have a concept of optional types, a type which allows the value to be nil.

3) Better IDE Integration :- Parsing RBS files gives IDE's better understanding of ruby code. Method name completions run faster. On-the-fly error reporting detects more problems. Refactoring can be more reliable.

4) Guided duck typing :- Interface types can be used for duck typing. It helps API users understand what they can do more precisely. This is safer version of duck typing.

* EMBEDDING A RUBY INTERPRETER.

~~Ex~~ addition

```
#include "ruby.h"

main() {
  ruby_init();
  ruby_script("embedded");
  rb_load_file("start.rb");
  while(1) {
    if (ruby_to_do_ruby) {
      ruby_run();
    }
  }
}
```

To initialize ruby interpreter you need to call ruby_init(). But some platforms you may need to take special steps

```
# if defined (NT).
    NTInitialize (&argc, &argv).
# endif.
# if defined (__MACOS__) && defined
  (__MWERKS__).
  argc = ccommand (&argv);.
# endif.
```

EMBEDDING RUBY INTERPRETER.

* In addition to extending Ruby adding C code, you can also turn the problem around and embedd ruby itself within your application
* We can interact Ruby by CAPI or by evaluating String
* First we will interact by evaluating String of Ruby code.
* To initialize the ruby interpreter, we need to call ruby-sysinit() to pickup command line arguments used by Ruby, Ruby_init-Stack. to setup the ruby stack and ruby-init to initialize the interpreter itself.
* The call to ruby-init-loadpulls adds any directions to be searched for libraries.
* This kind of hands of manipulation of ruby programs within C code is easy.
* But it has 2 major drawbacks
  a) we have to keep storing things in globals & extracting the value from those globals to use them
  b) we are not doing any real error checking.
* API

__/__/____

* This gives us finer grained control and also let us handle errors.
* We can do this by initializing the interpreter as normal.
* Then rather than evaluating strings instead involve specific methods in Ruby code.
* When these methods return, your C code gets control back.

Embedded Ruby API.

1) Void ruby_init(")
Sets up and initializes the interpreter.

2) Void ruby_options (int argc, char **argv").
Gives ruby interpreter the command line options

3) Void ruby_Script (char* name")
Sets the name of the Ruby script (and $0) to name.

4) Void rb_load_file (char* file").
loads the diff file into the interpreter

5) Void ruby_run(")
Runs the interpreter

EMBEDDING RUBY TO OTHER LANG.

* The ruby code syntax is familiar to users of perl, python, java etc.

C and C++.

Ruby and C have healthy symbiotic relationship.

Similarity with C:

As with C in Ruby the strings go in double quotes and the strings are mutable.

Similarity with C++.

As with C++, private, public, protected do some similar jobs, exception handling is also in similar manner.

Java:-

Java is mature, tested and is fast.

Similarities:-

As with Java in ruby

__/__/____

Memory is managed in a garbage collector, objects are strongly typed, There are public, private and protected members.

Perl

Perl docs are awesome.

Similarities

As with perl in Ruby parenthesis are often optional. Strings work identically the same.

PHP

PHP has widespread use for web apps

Similarities

As with PHP in ruby, ruby is dynamically typed like in PHP, Some variable start with $ like PHP.

Python

Python is another very nice general purpose programming lang.

Similarities

As with python in ruby, objects are strongly & dynamically typed. String literals can span multiple lin

__/__/____

UNIT-3

Characteristics of SL

1) Integrated Compile & Run :
   - Interpreted lang
   - Operate - immediate execution
   - no need to compile
   - no need to link extensive libraries

2) Low Overheads & ease of use
   - Variables can be declared by use
   - The no of diff data types is usually limited
   - Everything is string by content it will be converted as no (vice versa)
   - No of data structures is limited

3) Enhanced functionality
   - EF in some area
   - most lang provide string manipulation based on use of RE while
   - other lang provide ease access to low level OS facility

4) Efficiency is not an Issue
   Ease of use is achieved at expense of efficiency

5) Interpreted from Source code

6) Designed for use by end user

7) Use abbreviation

8) Scripts are distributed.

9) Ease of use

10) Lack of compile link load sequence

* PACKAGES

Package - collection of code - own namespace.
Namespace - symbolic table
↳ prevents name collisions b/w packages.
Packages → scalars → const of modules.
↳ won't hit variables & fun
outside of module's own namespace
stay in effect - until other package stmt
is invoked
explicitly refer as :: package qualifier

```
#!/usr/bin/perl
# This is main package
$i=1;
print "Package name : ", __PACKAGE__, "$i\n";
```

```
package Foo;
# This is Foo package.
$i = 10;
print "Package name : ", --PACKAGE--,
      "$i\n";

package main;
# This is again main package.
$i=100;
print "Package name : ", --PACKAGE--, "$i\n";
print "Package name : ", --PACKAGE--, "
      "$Foo::i\n";
1;
```

Result.

```
Package name : main
"       "    " Foo 10.
"       "    " main 100.
"       "    : main 10.
```

* PERL MODULES.

- reusable package.
- defined in library file.
- module name same as package name.
- .pm as extention.
- fun. require & use will load a module.
- both use list of search paths in @INC to find module.

__/__/____

* both i call the eval fun to process the code.
* the i; at the bottom causes eval to evaluate to TRUE