Ruby

What is it

- "More powerful than perl, more object-oriented than python..." -Matz
- Object oriented programming language
- Strongly and dynamicly duck typed
- Interpreted - no VM, no compiler
- Descended from lisp, smalltalk, and perl
- The best programming language ever

Why Ruby?

"principle of least surprise"

DRY

I tried to make people enjoy programming and concentrate on the fun and creative part of programming when they use Ruby

-Yukihiro Matsumoto (Matz)

I believe people want to express themselves when they program

First things first

- Installation http://www.ruby-lang.org/en/
- Documentation
  - http://www.ruby-doc.org/
  - ri - command line doc tool
- Running
  - Stand alone
  - Interactive

```
$ ruby myprog.rb
```

```
$ irb
irb(main):001:0>
```

Using IRB

- Use load and require to load files into the interpreter
- obj.inspect - print out a nice string representation
- GNU readline support. Tab completion.

```
$ irb -r irb/completion
```

- Ask an object for its methods `[].methods`
- Save configuration using .irbrc

Basics

- Basic Types
  - strings
  - arrays
  - hash tables
  - regular expressions
- Control - {false,nil} are logically false. everything else is true
- Exceptions - begin rescue ensure
- Lambda expressions and closures in the form of blocks

Blocks

- Used to pass code to objects (think closure)
- Used extensively in the ruby libraries
- Must be comfortable with them to be a good ruby programmer

```
# code between {}
{3 + 2 }

#or code between do...end
do
  3+2
end
```

Using blocks

```ruby
#blocks are used by passing them to methods
def use_block
  puts "start"
  yield
  yield
  puts "end"
end
use_block {puts "-> inside"}

#blocks can accept parameters
def msg
  yield "dave"
end
msg {|who| "a personal message for #{who}"}
#=> "a personal message for dave"
```

Blocks for iteration

- Used widely in ruby libraries for iteration

```
#used for iteration
names = %w{fred ginger mary anne}
names.each {|n| puts "hi, #{n}"}
5.times {puts "numbers are object too!"}
```

- Many possibilities. The File class uses them to ensure a file gets closed

```
#and for file IO
File.open("ex.rb") {|f| puts f.read}
```

What's in a name

- @var - instance variable
- @@var - class variable
- $var - global variable
- Var - class name, module name or constant
- Some conventions for special characters
  - empty? - boolean test
  - sort! - in place modification

Object-Oriented

- Everything is an object! Everything! Everything!
- Use mixins for multiple inheritance
- Every class is open - redefine a method whenever you want

Classes

```
class Beer
  def initialize
    @brand = "generic"
    @amt   = 12 #oz
  end
  def sip(s)
    @amt -= s
  end
  def amt #getter
    @amt
  end
end
```

```
class BlackButtePorter < Beer
  def initialize
    super
    @brand = "Deschutes Brewery"
  end
  def sip(s)
    super(s)
    @amt -= s
    puts "ahhhh..."
  end
end
```

```
b = BlackButtePorter.new
1.upto(3) {|i| b.sip(i)}
b.amt
```

Classes

- Class definitions are executable
  - attr_reader, attr_writer, attr_accessor - generate getters and setters
  - include - add methods on the fly
- Many hooks for meta-programming
- Class definitions are always open unless explicitly frozen by the programmer
- Other boring stuff too like access control

```ruby
class A
  attr_accessor :a
  attr_reader   :r
  attr_writer   :w

  #I can write my own version
  def my_attr(sym)
    A.class_eval <<HERE
      def #{sym}
        @#{sym}
      end
      def #{sym}=(v)
        @#{sym}=v
      end
HERE
  end
```

```ruby
#or even using blocks
def my_cattr(sym)
  A.class_eval do
    v = "@"+sym.to_s
    #getter
    define_method(sym.to_s) do
      instance_variable_get(v)
    end
    #setter
    define_method(sym.to_s+"=") do |val|
      instance_variable_set(v,val)
    end
  end
  nil
 end
end #class end
```

Modules

- Are not classes - can not create instances of modules
- Provide namespace
- Implement mixin feature
- Ruby uses mixins to handle multiple inheritence in a clean way
- A module included in a class adds methods to the class

```ruby
module Observable
  def observers
    @observer_list ||= []
  end
  def add_observer(o)
    observers << o
  end
  def notify_observers(*args)
    observers.each { |o|o.update(args)}
  end
end

#update the beer class, all existing
#objects get these methods too!
class Beer
  include Observable
end
```

```ruby
class BarBot
  def update(o)
    puts "next order: #{msg}"
  end
end

b = Beer.new
bb = BarBot.new
b.add_observer bb
b.notify_observers("another please")
```

Modules put to good use

- Enumerable module
  - User implements each and gets map, inject, member, ...
- Comparable module
  - Implement <=> and get <,<=, ==, >=, >

What else?

- Unit testing - RUnit
- Embedded documentation - RDoc
- Threads
- C interface
- Network programming
- System calls
- ...

Ruby on Rails

- Web application framework
- Uses the dynamic power of ruby
- “Convention over configuration”
- Integrated unit testing
- http://media.rubyonrails.org/video/rails_take2_with_sound.mov

DRb

- Distributed Ruby
- Similar to Java's RMI (Remote Method Invocation)
- Allows local process to transparently call methods on remote objects
- We are going to use it to create a chat server

Fun Time

- Create a ruby script that can connect to my server and display a message
- Script should take the message to display as a command line argument

```
$ ruby announce.rb “this is the best day ever”
```

Remote Methods

announce : string => nil

motd => string

Resources

- Pickaxe book (Programming Ruby by Dave Thomas)
- http://www.ruby-doc.org/core/
- http://www.ruby-doc.org/stdlib/
- http://www.zenspider.com/Languages/Ruby/QuickRef.html
- http://www.rubygarden.org/
- http://rubyforge.org/

Chat Server

- Implement a chat server client using DRb
- Client is a command line interface (think irc)
- Need to be able to send a message to the server and process local commands, such as #quit
- Useful modules OptionParser, Abbrev

Remote Methods

register : client => nil
say : string, string => nil
signoff : client => nil

Client Methods

update : string,string => nil
name => string