B'more On Rails

Wednesday October 25, 2006

Ruby: A Programmer's Best Friend

Anocha Yimsiriwattana
Medical Decision Logic, Inc.

Hello World

```
#!/usr/bin/ruby

puts 'Hello World'
```

When was the last time you had fun programming?

Purpose of Programming Language

- Teach the computer what to do.
- Describe the problem to solve.
- Express programmers' thought in a form of program.

- **Even better if**:
  - Helps our thought
  - Guides us to be better programmers
  - Allows us to enjoy programming
  - **But** does not brainwash us, just "influenced"

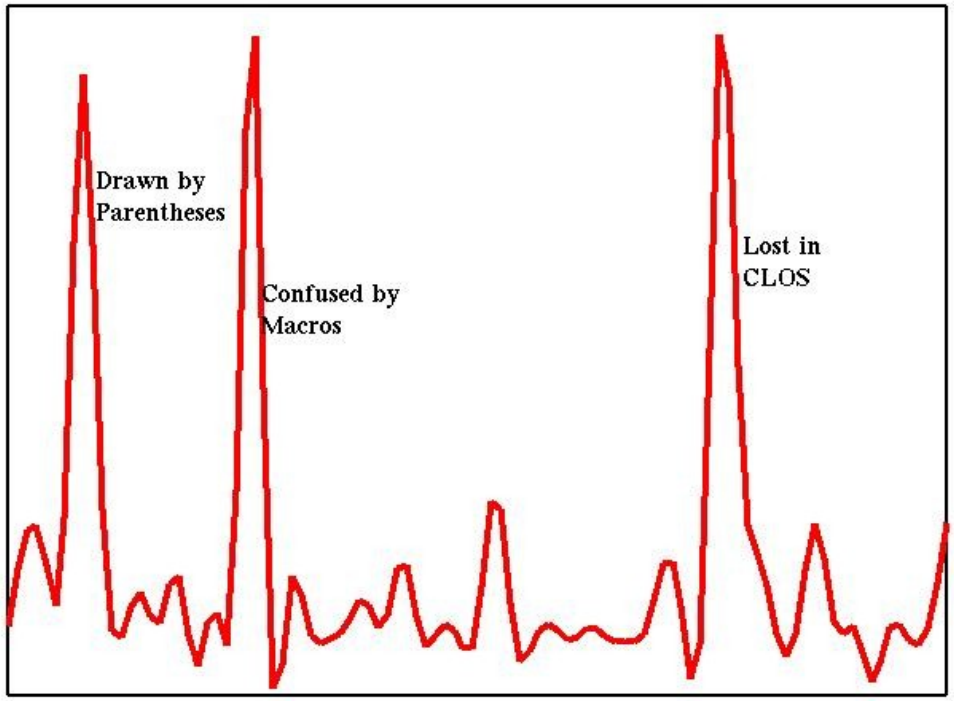
Source: The Ruby Programming Language

Well ...

- Ruby: puts “Hello world!”
- Java: threeVeryLongLines.weHopeWork...
- Perl: #$<!&;
- Lisp: ((a(((b)))(c)))

Source: The Top 10 Reasons, The Ruby Programming Language Suck!

Spending Brain Power



Lisp

Source: The Ruby Programming Language, http://www.ruby-doc.org/whyruby

Forgive me, Ruby

- Omitted system class
- No ; need
- Optional ()

- do ... end or { ... }

- Simple declarations:
  - local_var = ...
  - @instance_var = ...
  - $class_var = ...
  - $$global_var = ...
- Ruby Idiom!

```
# same as
# $stdout.puts('Hello world!')
puts 'Hello world!'


# actually is, three = 1+(2)
three = 1 + 2

# Both work
3.times do      3.times {
   ...             ...
end             }

# simply assign a value
index = 0
@table_name = 'protocols'

# myvar = value if myvar.nil?
myvar ||= value
```

Ruby is Object oriented

- Everything is an object
  - number, code blocks, etc

```
full = "John Frank"
list = full.split
#list => ['John', 'Frank']

2.times do
    list.each do | name |
        puts 'Hello ' + name
    end
end

2.class   # => Fixnum
{}.class  # => Hash

# Twenty minutes ago!
# Well ... work only in Rails.
20.minuts.ago
5.day + 4.hours + 10.minutes
```

Ruby is Object oriented

- Automatic constructor
- Easy accessors
  - all instance variables are private
  - all methods are public
- No need to use “self” everywhere.
- Doesn't require explicit “return” statement.

```
class Person
  attr_accessor :first_name
  attr_accessor :last_name

  def name
    first_name + ' ' + last_name
  end
  alias :to_s :name
end

# Simply initial and assign
person = Person.new
person.first_name = 'Thomas'
person.last_name = 'Eagle'
person.name # => 'Thomas Eagle'
person.to_s # => 'Thomas Eagle'
```

Ruby is Object oriented

- Ruby classes are open ->

  add methods to a class at any time

```
# somewhere later.
class Person

  alise :full_name :name

  def name
    last_name + ',' + first_name
  end

end

person.full_name # => 'Thomas Eagle'
person.name # => 'Eagle, Thomas'
person.to_s # => 'Thomas Eagle'
```

Organize your code with Module

- Modules provide
  - namespace
  - mixins
- Forgive me, Ruby
  - self::sin
  - self.sin
  - Trig::sin #true form

```
module Trig
  PI = 3.141592654
  def self::sin( x )
    ...
  end
end

module Moral
   BAD = 1
   def self::sin( badness )
      ...
   end
end

# same as Trig.sin( Tring::PI )
y = Trig::sin(Trig::PI)
wrong = Moral::sin( Moral::BAD )
```

Mixins, make sense!

- Rudy uses **single inheritance**, NOT multiple inheritance.
- Ruby uses "**Mixins**" to get the same benefit of multiple inheritance without drawbacks.
- Ruby classes can include the functionality of **any number** of mixins.

Mixins, make sense!

```
# file: my_debug.rb
module MyDebug
  def who_am_i?
    “#{self.class.name}“+
    “ : #{self.to_s}”
  end
end
```

```
require 'my_debug'

class Phonograph
  include MyDebug
  # ...
end

class EightTrack
  include MyDebug
  # ...
end

ph=Phonograph.new(“West End Bules”)
et=EightTrack.new(“Surrealistic P”)

ph.who_am_i?=>“Phonograph : Wes...”
et.who_am_i?=>“EightTrack : Sur...”
```

Mixins, make sense!

- Some standard mixins
  - **Comparable**
    <, <=, ==, >=, > by define method **<=>**
  - **Enumerable**
    map, include?, find_all by define method **each**

    max, min, sort ***if*** **<=>** is defined.

```
class Song
  include Comparable
  def initialize(name, duration)
    ...
  end
  def <=>( other )
    self.duration <=> other.duration
  end
end

sg1=Song.new('My way'   , 225)
sg2=Song.new('Bicylops', 260)

sg1 <=> sg2  # => -1
sg1 < sg2    # => true
sg1 == sg1   # => true
sg1 == sg2   # => false
sg1 > sg2    # => false
```

Containers, Blocks and Iterators

The more you code, the more you write classes that support iteration over their contents.

- Rich and powerful containers
  - **Array**, **Hash**, etc.
- Code blocks are everywhere
  - Any method can accept a block
  - Blocks can be called immediately or stored for later use.

Containers, Blocks and Iterators

```
# Array examples.
a = [ 3.141, "pie", 3, 5, 7, 9]

a.class   -> Array
a.length  -> 6
a[0]      -> 3.141
a[1]      -> "pie"
a[-1]     -> 9
a[-2]     -> 7
a[2,3]    -> [3,5,7]
a[-3,2]   -> [5,7]
a[1..3]   -> ["pie", 3, 5]
a[1...3]  -> ["pie", 3]
a[-3..-2] -> [5, 7]

a[0] = [1,2]   -> [[1,2],"pie", ...]
a[0,1] = [1,2] -> [1, 2, 3, 5, 7, 9]
a[0,3] = [1,3] -> [1, 3, 5, 7, 9]
```

```
# Hash examples.
h = {'dog'=>'canine',
     'cat'=>'feline',
     'donkey'=>'asinine' }

h.class       -> Hash
h.length      -> 3

h['dog']      -> 'canine'
h['cow']      -> nil
h['cow']='bowine'

h[:dog]       -> nil
```

Blocks and Iterators

- Any method can accept a block
- Blocks can be called immediately ...

```
#Iteration examples.
a = [ 1, 2, 3, 4, 5, 6, 7, 8, 9]

b = a.map { |x| x if x % 2 == 0 }
b.compact          -> [2, 4, 6, 8]
a.find { |x| x > 6 }  -> 7

#Block example.
def three_times
  yield
  yield
  yield
end

three_times { puts 'Hello' } ->

    Hello
    Hello
    Hello
```

Blocks and Iterators

... stored for later use,
or ignore it.

```
def single_tag( name )
  “<#{name}/>\n”
end
def block_tag( name, blk)
  “<#{name}>\n”+ blk.call +
   ”</#{name}>\n”
end
def tag( name, &block )
  if [“hr”, “br”].include? name
    single_tag( name )
  else
    block_tag( name, block)
  end
end
```

```
tag('hr') # ->
  <hr/>

tag('div') {'Content'} # ->
  <div>
  Content
  </div>

tag('div') do
  tag('label') { 'Name : ' } +
  tag('hr')
end # ->

  <div>
  <label>
  Name : </label>
  <hr/>
  </div>
```

Dynamic Programming

Lawless!

- Duck typing
  - aka Dynamic typing. Based on signatures, not class inheritance.
- Dynamic Dispatch
- Dynamic Behavior
  - reflection, scope reopening, class_eval, instance_eval, ...

Dynamic Programming

- Class scope reopening:
  - Overloading method,
  - Mixins: Comparable, Enumberable
  - etc
- **Object singleton class**

```
ary = [1,2,3]

class << ary      # define a class
  def [](index)   # redefined
    puts “accessing #{index}”
    super
  end
end

ary[1]->
# print
  accessing 1
# return
  1
```

Dynamic Programming

- Method missing

```
class HtmlBuilder
   def single_tag(name) ...
   def block_tag(name, blk) ...
   def tag(name, &block) ...

   def method_missing( name, &blk)
     tag( name.id2name, blk)
   end
end
```

```
hb = HtmlBuilder.new

hb.hr -> <hr/>
hb.div { 'Content' } ->
  <div>
  Content
  </div>

hb.div do
  hb.label { 'Name : ' } +
  hb.hr
end # ->

  <div>
  <label>
  Name : </label>
  <hr/>
  </div>
```

Dynamic Programming

- Duck typing philosophy

“if it walks like a duck and quacks like a duck, it's a duck!”

- Respond to signatures!

```
def append_five( obj )
  obj << 5
end
File.open(“five”, “w”) do |f|
  f.puts append_five( [1,3] )
  append_five( f )
end

# file “five”content:
  1
  3
  5
  5

File.send “open”, “five”, “w”do
 ...
end
File.methods
```

Dynamic Programming

- Create template using class_eval
- instance_eval, module_eval !

```
class MyTree
  def initialize( name, opts={} )
    self.const_set( name +
      "_TREE_OPTIONS", opts)

    code = <<-CODE
      def #{name}_tree_options
        #{name}_TREE_OPTIONS
      end
      def #{name}_get_items ...
    CODE
    class_eval code
  end
end

mt = MyTree 'mine', {:size=>5}
mt.mine_TREE_OPTIONS -> {:size=>5}
mt.mine_get_items ...
```

... and much more!

Ruby -> High Productivity

- Ruby Core 100% documented
- Tons of standard libraries
  - Read/Write: CSV, XML, YAML
  - Talk to: Email, FTP, Web (CGI)
- Tons of support tools
  - Debugger, R-Docs, RI, Test Unit, gems
- Communities
  - Tutorials available for various skill levels
  - Mailing list, Usenet, Web forums, Books!
- **RubyOnRails**
  - High productivity Web Application Framework!
- Various plugins, libraries, gem packages, ... you name it!

When was the last time you had fun programming?