More on Ruby

Good Style

- variables_use_snake_case
- methods_do_too
- ClassNamesAreCamelCased
- CONSTANTS_ARE_IN_CAPITALS

Members

```
>> class Person
>>    attr_reader :birthday
>>    attr_accessor :hair_colour
>>
?>    def initialize(birthday)
>>       @birthday = birthday
>>    end
>> end

>> p = Person.new("Monday") => #<Person: ...>

>> p.hair_colour = "Brown" => "Brown"
>> p. hair_colour => "Brown"

>> p.birthday = "Thursday"
NoMethodError: undefined method `birthday=' for
#<Person:0x1011a65b0 @hair_colour="Brown",
@birthday="Monday">
```



```
class Module
  def attr_reader (*syms)
    syms.each do |sym|
      class_eval %q{def #{sym}
                      @#{sym}
                    end}
    end
  end
  def attr_writer (*syms)
    syms.each do |sym|
      class_eval %q{def #{sym}= (val)
                      @#{sym} = val
                    end}
    end
  end
end
```

:symbol

- Identifier as a string of characters
  - `:name`
  - `:"a name"`
  - `:"Name of #{name}"`
- Symbol literals always yield the same object

Delimited input

- There are many ways to create strings, arrays, etc. in Ruby
  - %q %w %r %x
  - Delimiters can be any nonalphabetic, non-multibyte character


```
%q{def method(a)
     a.each{ |e| puts e }
   end}
=> "def method(a) \n     a.each{ |e| puts
e } \n   end"

%q/Can "use 'any' style of quotes"/
=> "Can \"use 'any' style of quotes\""

%W-a, b, c, d-
=> ["a", "b", "c", "d"]

%r(^a)
 => /^a/

%x(ls)
=> "Code\nL6\nL6.tar\nL6_DYPL_VT12.key
\nL6_DYPL_VT12.pdf\n"
```


Beyond the Interpreter

Modules?

Importing files

- Loading a file evaluates it
- Two ways:
  - *load*—loads file every time it is called
  - *require*—only loads the file once

Programs In Files?

```
class WordList
  def initialize(file_name, filter_str)
    @file_name, @filter_str = file_name, filter_str
    @matches = Array.new
  end

  ...

end

wl = WordList.new("words.txt", "aeoiuy")
puts wl.matches
```

- Write the code using your preferred tool
- Save it as "filename.rb"
- Run in terminal with "ruby filename.rb"

Importing files, cont'd

```
# full path required (with or without .rb)
require './path/to/file'
=> true

require './path/to/file.rb'
=> false

# path required (with .rb) if not full used
# as relative to "running directory"
load 'path/to/file.rb'
=> true

load 'path/to/file.rb'
=> true
```


Example - modules

```
# part_program.rb
x = "inwards!"
puts("I can sing #{x}")

def i_am_a_one_man_band
    puts("Non-stop rock'n roll, baby!")
end
```

```
# other_part_program.rb
require './part_program'

puts("Inwards singing!")

i_am_a_one_man_band()
```

Any code living at the topmost level of a module will be executed when the module is imported

in terminal:

```
dartanjang$ ruby other_part_program.rb
I can sing inwards!
Inwards singing!
Non-stop rock'n roll, baby!
```

Example - modules

```
# part_program.rb
x = "inwards!"
puts("I can sing #{x}")

def i_am_a_one_man_band
    puts("Non-stop rock'n roll, baby!")
end
```

```
# other_part_program.rb
require './part_program'

puts("Inwards singing!")

i_am_a_one_man_band()
```

Any code living at the topmost level of a module will be executed when the module is imported

in terminal:

```
dartanjang$ ruby other_part_program.rb
I can sing inwards!
Inwards singing!
Non-stop rock'n roll, baby!
```


Order may matter

```
# order_matters.rb
puts successor(3)

def successor(x)
  x + 1
end
```

in terminal:

```
dartanjang$ ruby order_matters.rb
order_matters.rb:1: undefined method
`successor' for main:Object (NoMethodError)
```

Note: The above is also true for Python

Methods

Method or Variable?

- Ruby syntax can make it hard to tell if a symbol denotes a variable or method

Method or Variable?

```
def a
  print "Function 'a' called\n"
  99
end

for i in 1..2
  if i == 2
    print "a=", a, "\n"
  else
   a = 1
   print "a=", a, "\n"
  end
end
```

```
# Output
a=1
Function 'a' called
a=99
```


Method or Function?

```
>> def m
>>   puts "#{self}, #{self.class}"
>> end
=> nil

>> m
main, Object
=> nil
```


Method or Function?

```
>> def m
>>   puts "#{self}, #{self.class}"
>> end
=> nil
>> m
main, Object
=> nil
```

```
#Python:

>>> def func():
...   print(self)
...
>>> func()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in func
NameError: global name 'self' is not
defined
```


Method Calling

- ? and !
- Leaving out parentheses
- Mutators with =
- Operators
- Last expression's value is the return value of the method

Method Calling

```
2 + 4
2.+(4)
2.+ 4

class Example
  def foo=(arg); @foo = arg; end
end

ex = Example.new
ex.foo = 23
```



```
class Private
  private
  def secret
    puts "This is my secret"
  end

  public
  def not_secret
    puts "This is not secret"
    secret
    self.secret
  end
end

> p = Private.new => #<Private:0x007f81c10627b0>
> p.not_secret
This is not secret
This is my secret
NoMethodError: private method `secret' called for #<Private:
0x007f81c10627b0>
   from (irb):38:in `not_secret'
   from (irb):42
   from /Users/beatrice/.rvm/rubies/ruby-1.9.2-p290/bin/irb:
16:in `<main>'
```


Variable Arguments

```
def example(*list)
  p list
end

example 1, 2, 3, 4, 5
=> [1, 2, 3, 4, 5]

def example(list); p list; end

example 1, 2, 3, 4, 5
ArgumentError: wrong number of
arguments (5 for 1)
```


Default Values

```
> def add_numbers(first, second=2)
>   puts first + second
> end
=> nil

> add_numbers(3)
5
=> nil

> add_numbers(3,5)
8
=> nil
```

Keyword? ... No ...

```
> def add_numbers(params)
>   puts params[:first] + params[:second]
> end
=> nil

> add_numbers(:second => 3, :first => 4)
7
=> nil
```


Exceptions

Exception Handling

- Ruby exception handling works like in Python
- Global variable $! stores current exception

Exception Handling

```
begin
  # ...
rescue ExceptionType => var_name
  # ...
rescue ExceptionType1, ExceptionType2 => var_name
  # ...
rescue => var_name # Matches any error
  # ...
ensure # Always run, cf. Java's finally
  # ...
end
```

Exception Handling

```
file = File::open("temp", "w")
begin
  line = file.readlines
  # ...
rescue Exception => error
  puts "Error occured: #{error}"
ensure
  file.close() unless file.nil?
end
```


Exception Handling

```
raise # re-raises current exception
raise ClassName
raise ClassName, args
raise "Error" # = RuntimeError, "Error"

# Examples:

raise "Some error occurred"

raise "Missing name" if name.nil?

raise MyError, "my message"

if i >= names.size raise IndexError,
                          "#{i} >= size (#{names.size})"
end
```


Classes

```
class Person
  attr_accessor :name, :age
end

# Classes are global constants
Person
=> Person

# Classes are regular values
my_class = Person
p = my_class.new
p.class
=> Person
```

Inheritance

```
class SubPerson < Person
  def method( arg )
    # implicit argument passing
    super
  end
end

ssp = SubPerson.new
ssp.method( "foo" )
# Prints "In SubPerson with foo"
```


Inheritance, cont'd

```
class SubSubPerson < SubPerson
  def method( arg )
    # explicit argument passing
    super(arg.to_s + "bar")
  end
end

ssp = SubSubPerson.new
ssp.method( "foo" )
# Prints "In SubSubPerson with foobar"
```


Open Classes

- Classes and modules in Ruby can be modified at any time
- If a class is defined again, it is extended (attributes are possibly overwritten)


```
class Ex; def foo=(a); @foo=a; end; end

ex = Ex.new
ex.foo = "Deeo"
ex.foo
NoMethodError: undefined method `foo'

class Ex; def foo; @foo; end; end

ex.foo
=> "Deeo"
```

Method aliasing

```
class Person
  def method; puts "In #{self.class}"; end
  alias_method :method_old, :method
  def method( arg )
     puts "In #{self.class} with #{arg}"
  end
end

Person.new.method_old
# Prints "In Person"

Person.new.method( “arg” )
# Prints "In Person with arg"
```



```
class Array
  def has_duplicates?
    elems = []
    dups = []
    self.each do |e|
      dups << e if elems.include? e
      elems << e
    end
    return dups == [] ? false : dups
  end
end

[1,2,2,3,4,5,5,6,1].has_duplicates?
=> [2, 5, 1]
```


Modules

- An abstract class without a super class | A name space | A fat interface
  - Can be mixed with classes
  - Inclusion order has semantics
  - Effectively multiple inheritance

Modules are abstract

```
module Greet
  def greet( arg )
    puts "Hello #{arg}, I'm #{name}"
  end
end


m = Greet.new
NoMethodError: undefined method `new'
for Greet:Module
```


Module inclusion

```
class Person
  # Include Greet in class definition
  include Greet

  attr_accessor :name
  def initialize( n ); @name = n; end
end

p = Person.new( "Ruby" )
p.greet( "Python" )
# Prints "Hello Python, I'm Ruby"
```


Module inclusion, cont'd

```
class Person
  include Comparable
  attr_reader :rank

  @@Rank = {:mgr=>3, :prg=>2, :tst=>1}

  def initialize( name, rank = :tst )
    @name, @rank = name, rank
  end

  def <=>( other )
    @@Rank[@rank] - @@Rank[other.rank]
  end
end
```

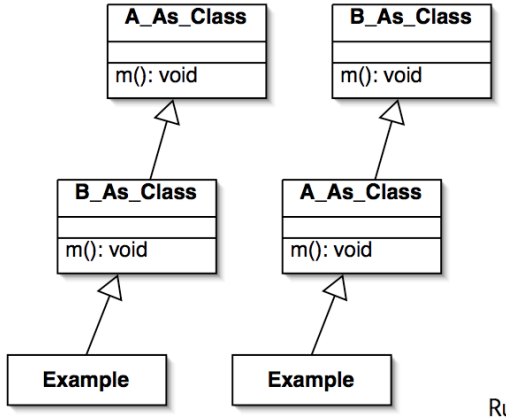

Inclusion order

```
module A; def m; puts "A"; end; end
module B; def m; puts "B"; end; end

class Example1; include A, B; end

class Example2; include A; include B; end

Example1.new.m
# Prints "A"

Example2.new.m
# Prints "B"

class Example1; include B, A; end

class Example2; include B; include A; end

Example1.new.m
# Prints "B"

Example2.new.m
# Prints "A"
```

Inclusion order

```
module A; def m; puts "A"; end; end
module B; def m; puts "B"; end; end

class Example1; include A, B; end

class Example2; include A; include B; end

Example1.new.m
# Prints "A"

Example2.new.m
# Prints "B"

class Example1; include B, A; end

class Example2; include B; include A; end

Example1.new.m
# Prints "A"

Example2.new.m
# Prints "B"
```


~ what goes on



Ruby tries to hide this fact

Procs, blocks, lambdas

Closures

- Blocks of code as objects
  - Defined in a specific context
  - Access to local variables, etc. where it was created
  - Can be passed around, stored in a variable etc.

Blocks

```
# using blocks
def map( list )
  for elem in list
    yield elem
  end
end

sum = 0
map( [1, 2, 3] ) { |elem| sum += elem }
puts sum

map( [1, 2, 3] ) do |elem|
  sum += elem
end
```


Blocks, cont'd

```
# Uses of blocks
File::open( file_name ) do |file|
  ...
end # closes the file automagically

(1..256).to_a.collect do |element|
  element if element % 2 == 0
end.compact
```


"Capturing" Blocks

```
>> def what_am_i(&block)
>>   block.class
>> end
=> nil

>>
?> puts what_am_i {}
Proc
=> nil
```


"Releasing" Blocks

```
 > def plus_or_times
 >   print "(t)imes or (p)lus: "
 >   times = gets
 >   print "number: "
 >   number = Integer(gets)
 >   if times =~ /^t/
 >     calc = lambda {|n| n*number }
 >   else
 >     calc = lambda {|n| n+number }
 >   end
 >   puts((1..10).collect(&calc).join(", "))
 > end
 => nil

 > plus_or_times
(t)imes or (p)lus: t
number: 2
2, 4, 6, 8, 10, 12, 14, 16, 18, 20
 => nil
```

Procs

```
>>class Person
>>   attr_accessor :behaviour
>>   def do_your_thing()
>>      @behaviour.call if @behaviour
>>   end
>> end
=> nil

>> p.do_your_thing()
=> nil

>> p1 = Proc.new{ puts "Hello" }
>> p.behaviour = p1
=> #<Proc:0x000000010117c260@(irb):68>
>> p.do_your_thing()
Hello
=> nil
```


Procs, cont'd

```
def callbacks(procs)
  procs[:starting].call

  puts "Still going"

  procs[:finishing].call
end

callbacks(:starting => Proc.new { puts "Starting" },
          :finishing => Proc.new { puts "Finishing" })
```


Procs vs. lambdas

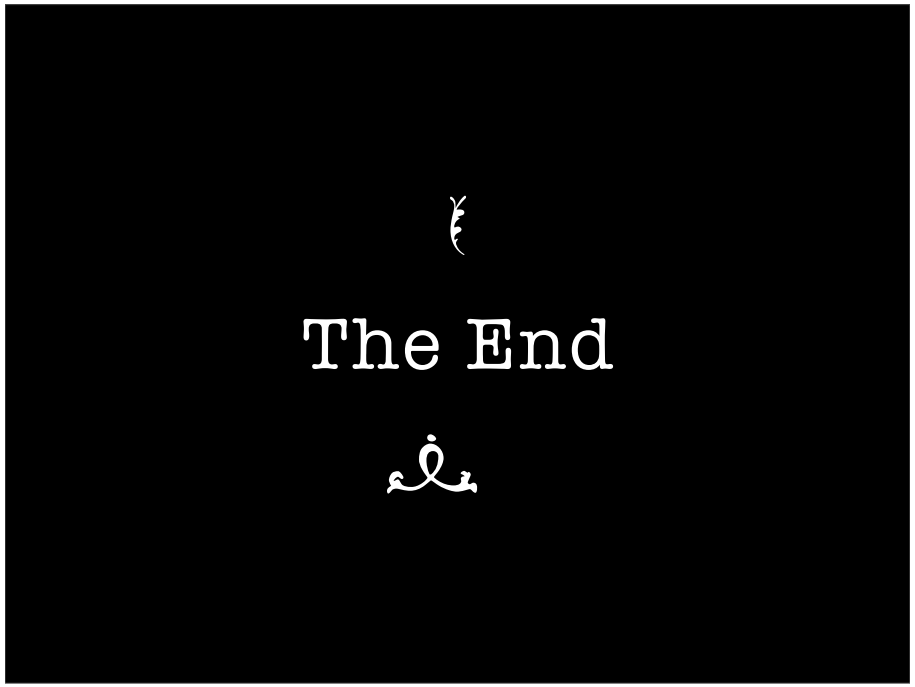
```
>> def args(code)
>>   one, two = 1, 2
>>   code.call(one, two)
>> end
=> nil

?> args(Proc.new{|a, b, c| puts "Give me a #{a}
                  and a #{b} and a #{c.class}"})
Give me a 1 and a 2 and a NilClass
=> nil

?> args(lambda{|a, b, c| puts "Give me a #{a}
               and a #{b} and a #{c.class}"})
ArgumentError: wrong number of arguments (2 for 3)
   from (irb):111
   from (irb):106:in `call'
   from (irb):106:in `args'
   from (irb):111
```


Procs vs. lambdas

```
>> def proc_return
>>   Proc.new { return "Proc.new"}.call
>>   return "proc_return method finished"
>> end
=> nil

?> def lambda_return
>>   lambda { return "lambda" }.call
>>   return "lambda_return method finished"
>> end
=> nil

?> puts proc_return
Proc.new
=> nil

>> puts lambda_return
lambda_return method finished
=> nil
```

“Capturing” Methods

```
>> def square(n)
>>   n ** 2
>> end
=> nil

>> s = method(:square)
=> #<Method: Object#square>
>> s.call(4)
=> 16
```

Further Reading

- Dave Thomas (with Chad Fowler and Andy Hunt) - “Programming Ruby”, a.k.a. The Pick-axe book
- http://www.rubyist.net/~slagell/ruby/
- http://www.robertsosinski.com/2008/12/21/understanding-ruby-blocks-procs-and-lambdas/
- http://slideshow.rubyforge.org/ruby19.html#1

Further Reading

- http://www.dcmanges.com/blog/ruby-dsls-instance-eval-with-delegation
- http://stackoverflow.com/questions/159797/is-ruby-a-functional-language
- http://carboni.ca/blog/p/Modules-How-Do-They-Work