Ruby for Java Programmers

CS 169 Spring 2012

Armando Fox

Outline

- Three pillars of Ruby (§3.1)
- Everything is an object, and every operation is a method call (§3.2–3.3)
- OOP in Ruby (§3.4)
- Reflection and metaprogramming (§3.5)
- Functional idioms and iterators (§3.6)
- Duck typing and mix-ins (§3.7)
- Blocks and Yield (§3.8)

Ruby 101 *(ELLS §3.1)*

Armando Fox

Ruby is...

- Interpreted
- Object-oriented
- Everything is an object
- Every operation is a method call on some object
- Dynamically typed: objects have types, but variables don't
- Dynamic
- add, modify code at runtime (metaprogramming)
- ask objects about themselves (reflection)
- in a sense *all* programming is metaprogramming

Naming conventions

- ClassNames use UpperCamelCase

```
class FriendFinder ... end
```

- methods & variables use snake_case

```
def learn_conventions ... end
def faculty_member? ... end
def charge_credit_card! ... end
```

- CONSTANTS (scoped) & $GLOBALS (not scoped)

```
TEST_MODE = true        $TEST_MODE = true
```

- *symbols:* immutable string whose value is itself

```
favorite_framework = :rails
:rails.to_s == "rails"
"rails".to_sym == :rails
:rails == "rails"  # => false
```

Variables, Arrays, Hashes

- There are no declarations!
- local variables must be assigned before use
- instance & class variables ==nil until assigned
- OK: x = 3; x = 'foo'
- Wrong: Integer x=3
- Array: x = [1,'two',:three]
  x[1] == 'two' ; x.length==3
- Hash: w = {'a'=>1, :b=>[2, 3]}
  w[:b][0] == 2
  w.keys == ['a', :b]

Methods

•Everything (except fixnums) is pass-by-reference

```
def foo(x,y)
  return [x,y+1]
end

def foo(x,y=0)  # y is optional, 0 if omitted
  [x,y+1]       # last exp returned as result
end

def foo(x,y=0) ; [x,y+1] ; end
```

•Call with: a,b = foo(x,y)or a,b = foo(x) when optional arg used

Basic Constructs

- Statements end with ';' or newline, but can span line if parsing is unambiguous

✔raise("Boom!") unless ✖ raise("Boom!") (ship_stable) unless (ship_stable)

- Basic Comparisons & Booleans: == != < > =~ !~ true false nil
- The usual control flow constructs
-

```
if cond (or unless cond)
  statements
[ elsif cond
  statements ]
[else
  statements]
end
```

```
while cond (or until cond)
  statements
end
1.upto(10) do |i| ...end
10.times do...end
collection.each do |elt|...end
```

Strings & Regular Expressions(try rubular.com for your regex needs!)

"string", %Q{string}, 'string', %q{string}

a=41 ; "The answer is #{a+1}"

•match a string against a regexp:

"fox@berkeley.EDU" =~ /(.*)@(.*)\.edu$/i

/(.*)@(.*)\.edu$/i =~ "fox@berkeley.EDU"

•If no match, value is false

•If match, value is non-false, and $1...$n *capture* parenthesized groups ($1 == 'fox', $2 == 'berkeley')

/(.*)$/i or %r{(.*)$}i
or Regexp.new('(.*)$', Regexp::IGNORECASE)

•A real example...

http://pastebin.com/hXk3JG8m

```
rx = {:fox=>/^arm/,
      'fox'=>[%r{AN(DO)$}, /an(do)/i]}
```

Which expression will evaluate to non-nil?

- ☐ "armando" =~ rx{:fox}
- ☐ rx[:fox][1] =~ "ARMANDO"
- ☐ rx['fox'][1] =~ "ARMANDO"
- ☐ "armando" =~ rx['fox', 1]