Introduction au ruby

11 avril 2017

Ruby

Un langage de script dynamique, interprété, orienté objet :

- Classes
- Methodes
- Exceptions

Plan

Main

- Pas de fonction main()
- Execution du code au fur et a mesure
- Références à des fonctions non définies autorisées
- Appel à une fonction non définie -> `NoMethodError`

Syntaxe

```
# commentaire
str = "toto\ntutu"
str2 = 'tata'
ary = [1, 2, 3]
hash = { 1 => 2, 3 => 4 }
float = 1.2
range = 1..2
range_exclu = 1...3
regex = /(a|b|c)/
symbol = :bla
heredoc = <<EOS
  bla
EOS
lol = { 1 => [2, 'haha'], /w/ => { 't' => :f } }
```

Structures de contrôle

```
i = 0
while i >= 0
    if i % 15 == 0
        puts i
    else
        # moo
    end
    i += 1
    break if (i > 50)
end
```

Switch case

```
case i
when Integer
      # si i est un entier
when "hoho"
      # si i est la chaine "hoho"
when /fu/
      # si i matche la regexp /fu/
when 1..12
      # si i est >= 1 et <= 12
when "a", "bla", 128
      # si a est parmi ces valeurs
else
      # what else
end
```

- Prend la première branche qui matche
- Pas de “fall through” comme en C (break implicite)
- Type de test différent suivant le cas
  - Comparaison via l'operateur `test === var`

Nommage des variables

- variables ou methodes
- Constantes
- :symbole
- $var_globale
- @var_instance
- @@var_classe (rare)

Méthodes

Déclaration

```
def blublu(arg0, arg1=nil)
    if arg1
        puts "blabla #{arg0} et #{arg1}"
        return arg1.length
    else
        puts "blublu #{arg0}"
    end
    28
end
```

- 2ème argument optionnel
  - Vaut *nil* si non précisé
- *return* implicite : valeur de retour = 28

Méthodes

Invocation

```
blublu(52)
blublu "haha", "hoho"
blublu({1 => 2, 3 => 4})
blublu 1 => 2, 3 => 4
```

- Parenthèses autour des arguments optionnelles
  - Si parenthèses, pas d'espace avant

Classes

Classe

```
class Foo < Object
    def bla
        puts "foobla"
    end

    # methode de classe
    def self.hoho
        puts "hoho"
    end
end

f = Foo.new
f.bla

Foo.hoho
```

- Nom de classe = constante
- Héritage simple (défaut = Object)

Blocs (lambda)

lambda

```
bloc = lambda { |i, j| puts i+j }
bloc.call(0, 0)
bloc.call(12, 28)
bloc2 = lambda do |i| puts i end
```

- Similaire à une fonction
- Déclare ses arguments entre |

Blocs (fonction)

fonction

```
[1, 2, 3, 4].each { |i| puts i }
```

- Raccourci syntaxique
- Appels déterminés par la fonction

Blocs (chaînage)

enchaînement

```
[1, 2, 3, 4].map { |i|
    i*4
}.find_all { |i|
    i >= 10
}.min
```

résolution

```
[4, 8, 12, 16].find_all { |i| i >= 10 }.min
[12, 16].min
12
```

- Point placé après l'accolade fermante
- Récepteur = valeur de retour de la fonction

Blocs (yield)

yield

```
def iter(&b)
    b.call(4)
    b.call(6)
end
iter { |i| i/2 }

def iter2
    yield(4)
    yield(6)
end
iter2 { |i| puts i }
```

- Encore du sucre syntaxique
- Valeur de yield = dernière valeur évaluée dans le bloc

Blocs (closure)

closure

```
i = 0

bloc = lambda { i = 1 ; j = 2 }
bloc.call

puts i # 1
puts j # NameError
```

- Réutilise les variables définies en-dehors si possible

Exceptions

rescue

```
begin

File.open('nonexistant')

raise "Bouh"

rescue
puts "Erreur : #{$!}", $!.backtrace
end
```

Plan

Types de base

- Integer
- String
- Array
- Hash

- true
- false
- nil

Regexp, Float, Range, Time, Symbol, File...

Integer

Gestion des entiers de taille arbitraire :

- Fixnum et Bignum sous-classes de Integer
- Transitions automatiques
  - 1 « 128
  - i = 1024 ; i *= 1024 ; i *= 1024 ; i *= 1024 ; i *= 1024

Integer

Integer

```
4.times { |i|
    # 0, 1, 2, 3
}
4.to_s    # "4"
100.to_s(16)    # "64"
```

Array

Array methods

```
['a', 'b'].length     # 2
[1, 2].each { |i| ... }
[1, 2].map { |i| f(i) }   # [f(1), f(2)]
[1, 2].find { |i| i % 2 == 0 }   # 2
[1, 2].find_all { |i| i % 2 == 0 }   # [2]
[2, 1].sort      # [1, 2]
[2, 1].sort_by { |i| 4*i+2 }    # [1, 2]
[1, 2] << 3      # [1, 2, 3]  (modifie l'objet)
[1, 2, 3].shift     # 1  (receveur = [2, 3])
[1, 2, 3].pop      # 3  (receveur = [1, 2])
```

String

String interpolation

```
"foo #{42} bar"          # "foo 42 bar"

"moo#{if i > 0 ; 'aa' ; else ; 'bb' ; end}"
   # "mooaa" ou "moobb"

"moo#@somevar"
```

String

String methods

```
"abcd".length        # 4
"abcde"[2, 2]        # "cd"
"ab|cd|ef".split("|")      # ["ab", "cd", "ef"]
str.split(/regexp/)
"42".to_i  =>  42
"20".to_i(16) => 32 # 0x20
"%s %i %04x" % ['aa', 2, 10]      # "aa 2 000a"
"AAAA".unpack('C*')     # [0x41,0x41,0x41,0x41]
"abcd".index("c")       # 2
```

Experimentations Ruby

- `ruby test.rb`
- `ruby -e 'puts :lol'`
- `irb`

Documentation :

- `ri String#unpack`