Solutions aux exercices du chapitre 4 : Ruby

Automne 2017

Exercice 4.1 Définition et utilisation de méthodes diverses avec plusieurs sortes d'arguments.

Soit la méthode suivante :

```ruby
def foo( x, y, z = nil )
  return x + y * z if z

  x * y
end
```

Indiquez ce qui sera affiché par chacun des appels suivants :

```ruby
# a.
puts foo( 2, 3 )

# b.
puts foo 2, 3, 5

# c.
puts foo( "ab", "cd", 3 )

# d.
puts foo( "ab", "cd" )
```

Solution :

```
# a.
puts foo( 2, 3 )
6

# b.
puts foo 2, 3, 5
17

# c.
puts foo( "ab", "cd", 3 )
abcdcdcd

# d.
puts foo( "ab", "cd" )
methodes-ex.rb:4:in '*': no implicit conversion of String
                          into Integer (TypeError)
    from methodes-ex.rb:4:in 'foo'
    from methodes-ex.rb:15:in '<main>'
```

Exercice 4.2 Définition et utilisation de méthodes diverses avec plusieurs sortes d'arguments.

Soit la méthode suivante :

```
def bar( v = 0, *xs )
  m = v
  xs.each do |x|
    m = [m, x].max
  end

  m
end
```

Indiquez ce qui sera affiché par chacun des appels suivants :

```
# a.
puts bar

# b.
puts bar( 123 )

# c.
puts bar( 0, 10, 20, 99, 12 )
```

Solution :

```
# a.
puts bar
0

# b.
puts bar( 123 )
123

# c.
puts bar( 0, 10, 20, 99, 12 )
99
```

Exercice 4.3 Définition d'une méthode avec plusieurs sortes d'arguments.

Soit le segment de code suivant :

```
def foo( x, *y, z = 10 )
  x + y.size + z
end

puts foo( 10, 20, 30 )
```

Qu'est-ce qui sera affiché?

Solution :

```
methodes-ex.rb:34:
   syntax error, unexpected '=', expecting ')'
def foo( x, *y, z = 10 )
                    ^
```

Exercice 4.4 Méthodes pour lire et modifier le titre d'un cours.

Pour la classe `Cours`, définissez une méthode qui permet d'obtenir le titre d'un cours et une autre méthode qui permet de **modifier** le titre d'un cours.

Utilisez ensuite cette dernière méthode pour changer le titre du cours `inf1120` en `"Programmation Java I"`.

Solution :
Différentes solutions dans différents styles :

```
# A la Java.
  def get_titre
    @titre
  end

  def set_titre( nouveau_titre )
    @titre = nouveau_titre
  end

  inf1120.set_titre( "Programmation Java I" )

# A la Ruby, version longue et explicite.
  def titre
    @titre
  end

  def titre=( nouveau_titre )
    @titre = nouveau_titre
  end

  inf1120.titre = "Programmation Java I"

# A la Ruby, version courte.
  attr_reader :titre
  attr_writer :titre

  inf1120.titre = "Programmation Java I"

# A la Ruby, version la plus courte.
  attr_accessor :titre

  inf1120.titre = "Programmation Java I"
```

Exercice 4.5 Une méthode pour identifier un sous-ensemble de préalables d'un cours.

Pour la classe `Cours` :

a. Définissez une méthode `prealables` qui reçoit en argument un `predicat` — une lambda-expression — et qui retourne la liste des préalables du cours qui satisfont ce `predicat`.

b. Utilisez la méthode `prealables` pour obtenir les préalables du cours `inf3105` dont le sigle contient la chaine `"INF"`.

Remarque : Pour ce dernier point, vous devez utiliser une expression de *pattern-matching*. En Ruby, l'expression suivante retourne un résultat *non nil* si `x`, une chaine, matche le motif `INF` :

```
/INF/ =~ x
```

Plus précisément, l'expression retourne `nil` si le motif n'apparait pas dans la chaine, sinon elle retourne la position du premier *match*.

Solution :

```
# Solution 1: avec des appels explicites a each

# a.
def prealables( predicat )
  resultat = []
  @prealables.each do |cours|
    resultat << cours if predicat.call(cours)
  end

  resultat
end


# b. Exemples d'appels de la methode.
inf3105.
  prealables( lambda { |c| /INF/ =~ c.sigle.to_s } )

inf3105.
   prealables( lambda { |c| /INF/ =~ c.sigle } )

inf3105.
  prealables( ->(c) { /INF/ =~ c.sigle } )
#-------------------------------------------------------


# Solution 2: avec select
# a.
def prealables( predicat )
  @prealables.select { |c| predicat.call(c) }
end

# b. Appel de la methode
# Inchange
```

Exercice 4.6 Mise en oeuvre fonctionnelle de `Cours#to_s`.

Donnez une mise en oeuvre, dans un style fonctionnel, de la méthode `to_s` de la classe `Cours` vue précédemment.

Solution :

```
def to_s
  sigles_prealables = @prealables
    .map { |c| c.sigle.to_s }
    .join( " " )

  "< #{@sigle} '#{@titre}' ( #{sigles_prealables} ) >"
end
```

Exercice 4.7 Méthode `mystere` sur un `Array`.

Que fait la méthode suivante? Quel nom plus significatif pourrait-on lui donner?

```
class Array
  def mystere( p )
    reduce( [[], [], []] ) do |res, x|
      res[1 + (x <=> p)] << x

      res
    end
  end
end
```

Solution :
Il s'agit d'une méthode `partitionner`, pouvant être utilisée pour faire un tri rapide (*quicksort*) :

```
class Array
  def partitionner( p )
    reduce( [[], [], []] ) do |res, x|
      res[1 + (x <=> p)] << x
      res
    end
  end

  def trier
    return [] if empty?

    petits, egaux, grands = partitionner( self[0] )
    petits.trier + egaux + grands.trier
  end
end
```

Une version plus simple à comprendre de la méthode `partitionner` est la suivante :

```
  def partitionner( pivot )
    [ select { |x| x < pivot },
      select { |x| x == pivot },
      select { |x| x > pivot }
    ]
  end
```

L'avantage de la première méthode ci-haut, plus complexe, est qu'elle ne nécessite **qu'une seule passe** sur le tableau, alors que la deuxième solution requiert trois (3) «passes».

Exercice 4.8 Pourquoi la méthode `<<` retourne-t-elle `self`?

Pourquoi la méthode `<<` retourne-t-elle `self`?
Que se passe-t-il si on omet `self`?

Solution :

Retourner `self` permet *le chainage des appels de méthodes* :

```
>> require_relative 'ensemble'
=> true
>> puts Ensemble.new << 10 << 20 << 10
{ 10, 20 }
```

Si on omet `self`, on ne peut plus chainer les appels, sinon le résultat ne sera pas correct :

```
>> require_relative 'ensemble-sans-self'
=> true

>> puts Ensemble.new << 10 << 20 << 10
10
20
10

>> e = Ensemble.new
=> #<Ensemble:0x00000002853c28 @elements=[]>

>> e << 10
=> [10]

>> e << 10
nil

>> e << 10 << 10
NoMethodError: undefined method '<<' for nil:NilClass
               from (irb):4
               from /home/tremblay_g/.rvm/rubies/ruby-2.2.5/bin/irb:11:in '<m
```

Exercice 4.9 Mises en oeuvre de `map` et `select`.

Supposons que dans la classe `Array`, on veuille définir les méthodes `map` et `select`, et ce utilisant `each` ou `each_index`. Quel code faudrait-il écrire?

```
class Array
  def map
    ...
  end

  def select
    ...
  end
end
```

Remarque : *Conceptuellement*, dans la vraie classe `Array`, ces méthodes sont disponibles simplement parce que la classe `Array` **inclut** le module `Enumerable`. **En pratique**, la mise en oeuvre de ces méthodes pour la classe `Array` est faite de façon spécifique à cette classe, pour des raisons d'efficacité — notamment, méthodes écrites en C dans Ruby/MRI.

Solution :

```ruby
class Array
  def map
    res = Array.new( size )
    each_index do |i|
      res[i] = yield( self[i] )
    end

    res
  end

  def select
    res = []
    each do |x|
      res << x if yield(x)
    end

    res
  end
end
```

Par contre, voici la «vraie» mise en oeuvre en Ruby/MRI de la méthode `map` :

```c
static VALUE
rb_ary_collect(VALUE ary)
{
    long i;
    VALUE collect;

    RETURN_SIZED_ENUMERATOR(ary, 0, 0, ary_enum_length);
    collect = rb_ary_new2(RARRAY_LEN(ary));
    for (i = 0; i < RARRAY_LEN(ary); i++) {
        rb_ary_push(collect, rb_yield(RARRAY_AREF(ary, i)));
    }
    return collect;
}
```

Exercice 4.10 Objet `MatchData`.

Qu'est-ce qui sera imprimé par les instructions `p` suivantes :

```
code_permanent = /(\w{4})     # NOMP
                  (\d{2})     # Annee
                  (\d{2})     # Mois
                  (\d{2})     # Jour
                  ([^\D]{2})
                 /x

m = code_permanent
      .match "CP: DEFG11229988."

p m[1]
p m[5]
p m.pre_match
p m.post_match
```

Solution :

```
# m[1]
"DEFG"

# m[5]
"88"

# m.pre_match
"CP: "

# m.post_match
"."
```

Exercice 4.11 Utilisation de `ARGV` et `ENV`.

Soit le script suivant :

```
$ cat argv2.rb
#!/usr/bin/env ruby

ENV['NB'].to_i.times do
  puts ARGV[0] + ARGV[1]
end
```

Qu'est-ce qui sera imprimé par les appels suivants :

```
# a.
NB=3 ./argv2.rb 3 8

# b.
NB=2 ./argv2.rb [1, 2] [3]

# c.
unset NB; ./argv2.rb [1009, 229342] [334]
```

Solution :

```
# a.
NB=3 ./argv2.rb 3 8
38
38
38

# b.
NB=2 ./argv2.rb [1, 2] [3]
[1,2]
[1,2]

# c.
unset NB; ./argv2.rb [1009, 229342] [334]
```

Note : ''.to_i == 0

Exercice 4.12 Ruby, un langage interprété?

Soit l'affirmation suivante : «Ruby est un langage interprété».

Cette affirmation est-elle vraie ou fausse?

Solution :

Vraie, mais en partie seulement, pas complètement... puisqu'il y a quand même un processus de **compilation** lorsqu'on utilise Ruby — comme en Java!

On trouve ci-bas un script Ruby (C-Ruby, MRI) et son code octet généré par le processus de compilation, lequel code octet est ensuite exécuté par la machine virtuelle YARV.

```
$ rvm use ruby-2.1.4
```

```
$ cat hello0.rb
puts "Bonjour le monde!"

$ irb
>> puts RubyVM::InstructionSequence.
        compile( IO.readlines("hello0.rb").join ).
        disasm
== disasm:
<RubyVM::InstructionSequence:<compiled>@<compiled>>==========
000 trace            1 (   1)
002 putself
003 putstring        "Bonjour le monde!"
005 opt_send_simple <callinfo!mid:puts, argc:1, FCALL|ARGS_SKIP>
007 leave
=> nil
```

> **YARV** *(Yet another Ruby VM) is* **a bytecode interpreter** *that was developed for the Ruby programming language by Koichi Sasada. The goal of the project was to greatly reduce the execution time of Ruby programs.*
>
> *Source : https://en.wikipedia.org/wiki/YARV*

Et ci-bas le même programme, mais cette fois traité par `jrubyc`, qui compile un programme Ruby et génère du cote octet pour la JVM (*Java Virtual Machine*).

```
$ rvm use jruby

$ cat hello0.rb
puts "Bonjour le monde!"

$ jrubyc hello0.rb

$ ls -l hello0.class
-rw-r-r-.  1 tremblay tremblay 3414 25 nov 09:42 hello0.class

$ java\
  -cp .:/home/tremblay/.rvm/rubies/jruby-1.7.16.1/lib/jruby.jar\
  hello0
Bonjour le monde!
```

Exercice 4.13 Performances de Ruby.

Pourquoi les performances d'un programme Ruby sont-elles généralement moins bonnes (programme plus lent ☹) que celles d'un programme Java?

Solution :

Ruby est un langage **dynamique** — typage dynamique, *dispatch* dynamique des méthodes, etc. Donc, plusieurs aspects d'un programme Ruby sont traités à l'exécution plutôt qu'à la compilation.

> *Classes in Java are closed, which is one of the reasons Java can run pretty fast. In contrast, Ruby's classes are open, which means you can add new things to them at any time. Keeping that option open is perhaps one of the reasons Ruby runs so slow. But that flexibility is also why Ruby has Rails.*
>
> *Source :* «Programming is Hard, Let's Go Scripting», *L. Wall, 2007,* `http://www.perl.com/pub/2007/12/06/soto-11.html`

De plus, les compilateurs Java utilisent aussi la technique de compilation JIT = *Just In Time Compilation* ⇒ le code octet est compilé en *code machine* puis exécuté directement par la machine plutôt que par la machine virtuelle.

...