Ruby : l’essentiel à savoir

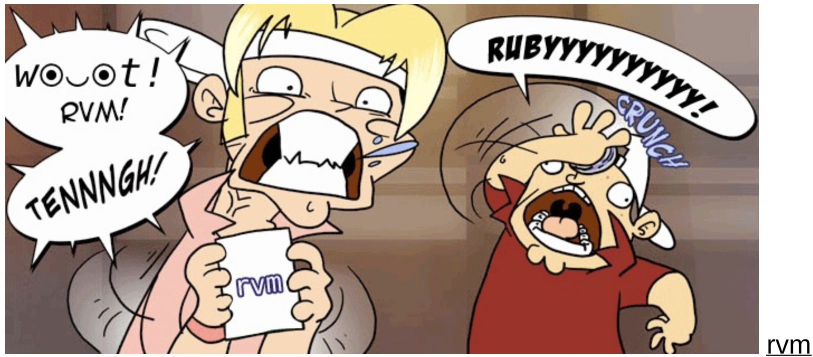
En substance :

- appréhender le langage
- voir ce qui est original

Plan

- Ruby : origine et parenté
- la syntaxe
- focus sur le côté dynamique

Installation : utiliser rvm



rvm

RVM permet de gérer plusieurs version de ruby simultanément, ainsi que plusieurs ensembles de bibliothèques (gemset), typiquement 1 par projet.

```
# install rvm dans ~/.rvm
# instructions à exécuter sous le shell bash
# N7 : proxy
# export http_proxy="http://proxy.enseeiht.fr:3128"
# git config --global http.proxy $http_proxy
# echo "proxy = proxy.enseeiht.fr:3128" > ~/.curlrc
$ bash < <(curl -s https://rvm.beginrescueend.com/install/rvm)
# configuration des chemins d'accès aux binaires
$ echo '[[ -s "$HOME/.rvm/scripts/rvm" ]] && . "$HOME/.rvm/scripts/rvm" # Load RVM function' >> ~/.bashrc
# source ~/.bashrc ou ouvrir un nouveau terminal
```

Installation de ruby

Les versions majeures de ruby :

- MRI/YARV : 1.8.7, 1.9.2
- Jruby : tourne sur une JVM, permet d'accéder à du code Java
- Rubinius : écrit en Ruby, version 2 bon support des threads.

```
#debian/ubuntu
# apt-get install build-essential zlib1g-dev libssl-dev libreadline-dev
$ rvm install 1.9.2 # compter 7 min environ
$ rvm use 1.9.2 --default
$ ruby -v
ruby 1.9.2p180 (2011-02-18 revision 30909) [x86-64-linux]
```

Ruby : le début

- Ruby est un langage interprété
- shebang : `#!/usr/bin/env ruby` en première ligne

```
#!/usr/bin/env ruby
# les expressions sont séparées par des retours à la ligne
# plusieurs expressions par lignes si on sépare par un `;`
puts "hello world"
```

```
# exécution
$ ruby hello.rb
hello world
$ chmod +x hello.rb
$ ./hello.rb
hello world
```

- interpréteur de commande `irb`, permet d'essayer

```
# install rvm dans ~/.rvm
$ irb
>> 1 + 1
=> 2
>> require './hello'
hello world
=> true
```

Ruby en résumé

Langage interprété, orienté objet, typage dynamique, généraliste, libre, avec :

- focus sur la simplicité et la productivité
- syntaxe élégante et concise, ET lisible
- open source
- racines : Lisp, Perl, Smalltalk
- Principle Of Least Surprise

Créé par Yukihiro Matsumoto, a.k.a. Matz, première version publique en 1995.

Dernière version stable : 1.9.2, aout 2010.

Pour comparer : Perl 1987, Python 1990

Ruby, les origines : plombier Perl

- traitement de chaînes de caractères : `split, join, upcase ...`
- regexp
- scriptologie classique : variable, tableau, hash, typage implicite
- la plomberie : one liner `ruby -pe` …
- $_ !

```
>> "a:b:c".split(':')
=> ["a", "b", "c"]
>> ["Bob", "est", "une", "eponge"].join(" ")
=> "Bob est une eponge"
```

- exécution de commandes externes avec `` `command` `` et `system(command)`

```
>> puts `hostname`
karma.enseeiht.fr
=> nil
>> system("touch /tmp/toto")
=> true
```

- nommage des identifiants

```
variable_locale = 42
nom_methode
CONSTANTE = 42 ARGV ENV STDIN STDOUT
NomDeClasse
NomDeModule
Classe::Imbriquée
$variable_globale
$$ # id processus
$? #exit d'une commande externe
$: # LOAD_PATH (chemin vers les librairies)
```

Syntaxe: les types

Typage implicite, typage fort

```
i = 1 # Integer
r = 1..10 # Range
t = "titi" # String
sym = :symbole # chaîne de caractères non modifiable
$glop = 4 # variable globale
TOTO = "toto" # constante, non modifiable
```

Tableaux

```
a = [1, "toto"]
a[0] # => 1
a << t # => [1, "toto", "titi"]
a.shift # => 1
a # => ["toto", "titi"]
```

RegExp : idem Perl

```
s = "Perl est excellent pour le
traitement des chaînes de caractères"
correction = s.sub(/Perl/, "Ruby")
s =~ /excel/ # => 9, position du matching
s =~ /perl/ # => nil, no match
s =~ /perl/i # => 0

# Extract the parts of a phone number

phone = "123-456-7890"

if phone =~ /(\d{3})-(\d{3})-(\d{4})/
  ext  = $1
  city = $2
  num  = $3
end
```

Syntaxe: les types

Chaînes de caractères et symboles

```
s = '123\ntoto' # => "123\\ntoto"
i = 4
mess = "2+2=#{i}" # => interpolation
sym = :symbole
sym.to_s => "symbole"
"titi".to_sym => :titi
```

Le vrai, le faux

- le faux: `nil`, `false`
- `nil` : valeur d'une variable non assignée, i.e. objet innaccessible
- le vrai : tout le reste, y compris `0`, `""`

Hash

```
h = {"key" => "val", :a => t} # on utilise souvent des symboles pour les clefs
h["key"] => val
```

Syntaxe : le reste

Structures de contrôle

```
if s =~ /Perl/
  puts "on parle de perl ici ?"
end
puts "on parle de Ruby !" if correction =~ /Ruby/
```

while until unless case/when …

Chaînage d'opérateurs

pour les amoureux des pipes en shell

```
"PHRASE EN    MAJUSCULE".split(/\s/).map{|w| w.downcase}.join(':')
#=> "phrase:en:majuscule"
```

Itérateurs sur les types enumérés

```
(1..10).each do |i|
 puts i
end # bloc paramétré

a.each{|i| puts i.inspect} # syntaxe alternative de bloc
```

En conséquence, on écrit peu de while/for/until

```
5.times do
  puts "for est mort, vive Ruby"
end
```

Entrées/Sorties

```
>> File.open("/tmp/test","w") do |f|
?> f.puts "écriture"
>> f.puts "dans un fichier"
>> end
=> nil
>> puts File.open("/tmp/test"){|f| f.read}
écriture
dans un fichier
=> nil
```

Syntaxe : les blocs

bloc =~ méthode anonyme

- 2 syntaxes, par convention :
- une seule ligne : `{ |i,j| do_stuff }`
- sur plusieurs lignes : `do |i,j| do_stuff end`

```ruby
# Print out a list of of people from
# each person in the Array
people.each do |person|
  puts "- #{person.name}"
end

# A block using the bracket syntax
5.times { puts "Ruby rocks!" }

# Custom sorting
[2,1,3].sort! { |a, b| b <=> a }
=> [3, 2, 1]
```

On peut passer un bloc en paramètre à une méthode.
`yield` est utilisé pour donner le contrôle au bloc.

```ruby
# define the thrice method
def thrice
  yield
  yield
  yield
end

# Output "Blocks are cool!" three times
thrice { puts "Blocks are cool!" }
```

yield avec paramètre :

```ruby
# define the thrice method
def thrice
  yield(1)
  yield(2)
  yield(3)
end

thrice { |i| puts "#{i} Blocks are cool!" }
```

Syntaxe : les méthodes

- conventions de nommage `methode_dangereuse!` , `methode_true_false?`, `methode_affectation=`

```
def plop(arg1, arg2="default value", *array_argument)
# arguments : arg1, arg2 avec une valeur par défaut
# array_argument est un tableau qui capturera toutes les valeurs restantes
# argument optionnel : un block
puts arg1.inspect
puts arg2.inspect
puts array_argument.inspect
puts "a block was given" if block_given?
valeur_de_retour = 2
end

# appel
# les arguments avec une valeur par défaut sont optionnels
# de même que l'argument *array_argument
# de même que le bloc
# une méthode renvoie la valeur de la dernière instruction
>> plop("toto")
"toto"
"default value"
[]
=> 2
>> plop("toto","titi")
"toto"
"titi"
[]
=> 2
>> plop("toto","titi",1,"2y",45){ puts "et un block"}
"toto"
"titi"
[1, "2y", 45]
a block was given
=> 2
```

Ruby les origines : Smalltalk pour l'objet

Classe

```
class Hello
end
```

variable d'instance (unique pour chaque objet): `@name`
variable de classe (partagée par tous les objets d'une même classe) : `@@name`

`initialize` : méthode appelé pour initialiser un objet

```
def initialize(stuff)
end
```

instancer un objet : `new()` : allocation mémoire et appel à `initialize`

```
h = Hello.new
```

- ramasse-miettes
- gestion des exceptions : idem Java `begin rescue end`, soulevées avec `raise`
- héritage simple : idem Java

```
class A < B
end
```

Objet ET dynamique

(presque) tout est objet

```
1.class          # => Fixnum
1.class.class    # => Class
(10**100).class  # => Bignum
nil.class        # => NilClass
true.class       # => TrueClass
Proc.new{ puts "un bloc n'est pas un objet mais …"}.class # => Proc
def method(&block)
  block.class # Proc
  block.call(params) # alternative à yield
end
```

Toutes les variables sont des références à des objets.

typage fort et dynamique

```
a = 1 ; a.class   # => Fixnum
a = "t" ; a.class # => String
```

Syntaxe Objet : la base, exemple

```ruby
class AnsweringThingee
  @@total_messages = 0

  def initialize
    @messages = []
  end

  def store(s)
    @messages << s
    @@total_messages+=1
  end

  def dump
    @messages.each_with_index do |m,i|
      puts "#{i.to_s}: #{m}"
    end
  end

  def self.total_messages
    @@total_messages
  end
end
```

```ruby
@un_truc   # variable d'instance, toujours privée !
@@un_autre # variable de classe, toujours privée !
```

```ruby
puts AnsweringThingee.total_messages => 0
r = AnsweringThingee.new
r2 = AnsweringThingee.new
r.store("premier message")
r.store("2e message")
puts AnsweringThingee.total_messages => 2
r2.store("autre répondeur")
puts AnsweringThingee.total_messages => 3
r.dump
 => 0: premier message
 => 1: 2e message
```

- objet courant : `self`
- référence à la superclasse : `super`

Passage de message

Subtilement différent d'appel de méthode

`"abcdef".length`
: envoie le message `length` à l'objet `"abcdef"`

`1 + 2`
: envoie le message + avec l'argument 2 à l'objet 1

`s[i]`
: envoie le message [] avec l'argument i à l'objet s

`methode`
: pas d'objet dans l'appel : message envoyé à `self`

`objet.methode_inconnu`
: l'objet ne connaît pas le message (i.e. n'a pas la méthode), alors `method_missing` reçoit le message

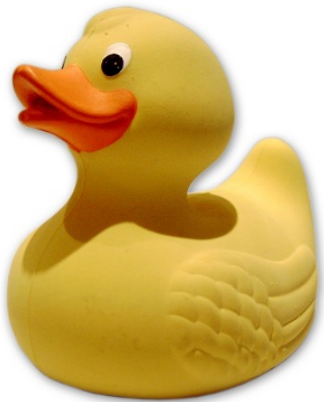
```ruby
class CsvItem
  def self.attributes=(attributes)
    @@attributes = attributes
  end

  def initialize(s=nil)
    @vals = Hash.new
    s.split(':').each_with_index do |val,i|
      @vals[@@attributes[i]] = val
    end unless s.nil?
    @vals
  end

  def method_missing(method, *args, &block)
    if @@attributes.include?(method.to_s)
      @vals[method.to_s]
    else
      super
    end
  end
end

CsvItem.attributes = "prenom:nom:age".split(':')
 => ["prenom", "nom", "age"]
moi = CsvItem.new("pierre:gambarotto:36")
moi.prenom
 => "Pierre"
moi.autre
NoMethodError: undefined method `autre' for #<CsvItem:0x000000011eb008>
```

csv.rb

Duck Typing

- If it walks like a duck,
- And talks like a duck,
  - Then we can treat it like a duck.
  - (who cares what it *really* is)

```
class Duck
  def talk() puts "Quack" end
end
class DuckMouse
  def talk() puts "Kwak" end
end
flock = [ Duck.new, DuckMouse.new ]
flock.each do |d| d.talk end
```



- `Enumerable(each)`
- `Comparable(<=>)`
- `<<(append)`

Le typage se définit donc comme un ensemble de signatures de méthodes compris par un objet

Réutiliser du code

Héritage simple, idem Java

```
class List < ActiveRecord::Base
  def redefine_method
    super # reference à la méthode de la classe parente
  end
end
```

Modifier des classes existantes !

```
>> a = [1,2,3]
a.first
=> 1
a.second
NoMethodError: undefined method `second' for [1, 2, 3]:Array
```

```
class Array
  def second
    self[1]
  end
end
>> a.second
=> 2
```

Mixins : héritage multiple

Module

- est un espace de nommage propre (idem classe)
- peut contenir des définitions de méthodes (idem classe)
- ne peut pas être instancié
- peut être inclus dans une classe, les méthodes du module deviennent alors des méthodes d'instance de la classe (`include ModuleName`)
- peut étendre une classe, les méthodes du module deviennent alors des méthodes de classe (`extend ModuleName`)

```ruby
require "forwardable"

class Stack
  extend Forwardable

  def_delegators :@data, :push, :pop, :size, :first, :empty?

  def initialize
    @data = []
  end
end
```

```ruby
module Debug
  def debug
    "#{self.class.name} (\##{self.object_id}): #{self.to_s}"
  end
end

class One
  include Debug
  def to_s
    "I am the one"
  end
end

class Two
  include Debug
  def to_s
    "Two is better than one"
  end
end

o = One.new
t = Two.new

o.debug # => "One (#70277190313000): I am the one"
t.debug # => "Two (#70277190155100): Two is better than one"
```

Le côté dynamique

Introspection

- `methods`, `instance_methods`, `class_methods`
- `instance_variables`, `constants`, `class_variables`
- voir le contenu de la classe `Object` pour plus de méthode.

Manipulation dynamique de code

`eval`, `class_eval`, `instance_eval`, `define_method`

```
Toto # => NameError: uninitialized constant Object::Toto
eval("class Toto \n end") # => nil
Toto #=> Toto
Toto.instance_methods - Object.instance_methods # => []
Toto.class_eval("def yo\nputs 'yoooo'\nend") # => nil
Toto.new.yo # affiche : yoooo
```

```
module MyForwardable
  def def_delegators(ivar, *delegated_methods)
    delegated_methods.each do |m|
      define_method(m) do |*a, &b|
        obj = instance_variable_get(ivar)
        obj.send(m,*a, &b)
      end
    end
  end
end
```

Adaptation

- `method_missing`
- `included` : méthode appelée sur un module inclus dans une classe
- DSL : exemple des getters/setters `attr_reader, attr_writer, attr_accessor`

```
 def new_attr_reader(*args)
  args.each do |e|
    define_method(e) do
      instance_variable_get("@#{e}")
    end
  end
end

class A
  new_attr_reader :a, :b, :c

  def initialize
    @a = 1
    @b = 2
    @c = 3
  end
end

a = A.new
p [a.a, a.b, a.c] #=> [1, 2, 3]
```

Exemple complet

```ruby
module Record
  def attribute(name)
    @attributes ||= []
    @attributes << name.to_sym
    define_method(name.to_sym) do |val=nil|
      @values[name.to_sym] = val if val
      @values[name.to_sym]
    end
    define_method((name.to_s+"=").to_sym){ |val| @values[name.to_sym] = val }
  end

  def attributes
    @attributes
  end
end

class Personne
  extend Record

  attribute :nom
  attribute :prenom
  attribute :age

  def initialize(&block)
    @values = {}
    instance_eval(&block) if block_given?
  end
end
```

record.rb

```ruby
>> p = Personne.new
=> #<Personne:0x00000001e293b0 @values={}>
>> p.age = 42
=> 42
>> p.nom = "Eponge"
=> "Eponge"
>> p.prenom = "Bob"
=> "Bob"

# DSL
p = Personne.new do
  prenom "Marcel"
  nom "Marceau"
  age 24
end
=> #<Personne:0x00000002133f88 @values={:prenom=>"Marcel", :nom=>"Marceau", :age=>24}>
```

Écosystème

Documentation

```
$ rvm docs generate-ri
$ ri String#split
```

Beaucoup de documentations de très bonne qualité sur le web.

Gestion des bibliothèques

`gem` va chercher et installe les bibliothèques ruby :

```
$ gem install ruby-net-ldap
$ irb
>> require "net/ldap"
=> true
```

`bundler` permet de spécifier les dépendances d'une applications dans un fichier `Gemfile` et de les installer :

```
# Gemfile
source "http://rubygems.org"

gem "ruby-net-ldap", :lib => "net/ldap"
```

```
$ bundle install
# installe toutes les gems spécifiées dans Gemfile
```