Ruby : une petite présentation

En substance :

- Ruby : origine et parenté
- la syntaxe
- focus
- les stars du monde ruby

À propos de moi …

Hybride 60/40

- ingénieur système
  - maintenance de parc unix, desktop/serveurs
- développeur
  - applis web pour la gestions de personnes (BD, Ldap, web services)
- contextes multiples : département info « n7, labos « INPT « PRES
- parcours sur le scripting : C, bash, perl, ruby
- parcours sur le développement : C, php, Java, ruby

Papa plombier : Perl

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

Syntaxe: la base

Typage implicite

```
i = 1
r = 1..10
t = "titi"
$glop = 4 # variable globale
```

Tableaux

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

Hash

```
h = {"key" => "val", a => t}
h["key"] => val
```

RegExp : idem Perl

```
s = "Perl est excellent pour le
traitement des chaînes de caractères"
correction = s.sub(/Perl/, "Ruby")
```

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 …
```

Itérateurs sur les types enumérés

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

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

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

Chaînage d'opérateurs

pour les amoureux des pipes en shell

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

Maman superhéroïne : Smalltalk

- ramasse-miettes
- gestion des exceptions : idem Java
- héritage simple : idem Java

```
class A < B
end
```

(presque) tout est objet

```
1.class          # => Fixnum
1.class.class    # => Class
(10**100).class # => Bignum
nil.class        # => NilClass
true.class       # => TrueClass
```

tout est modifiable

```
class Integer
  def ultimate_answer?
    self == 42
  end
end

(14 * 3).ultimate_answer? # => true
```

typage fort et dynamique

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

Syntaxe Objet : la base

```ruby
class AnsweringThingee
  @@total_messages = 0

  def initialize
    @messages = []
  end

  def store(s)
    @messages << s
  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
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
```

```ruby
@un_truc   # variable d'instance
@@un_autre # variable de classe
```

Passage de message

Pas 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

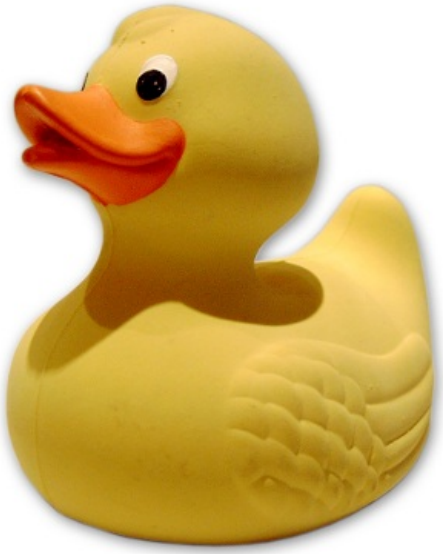
```ruby
class CsvItem
  def self.attributes=(attributes)
    @@attributes = attributes
  end

  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
```




COIN !

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 de la classe

```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"
```

Ecosystème

- interpréteur:

```
irb
> class A
>  end
=> nil
> a = A.new
=> #<A:0x00000002980bc8>
```

- rubygems : format de distribution de bibliothèque, et distribution par réseau

```
gem install ruby-net-ldap
```

- différentes VM, rvm

`rvm` permet de faire cohabiter plusieurs environnements d'exécution (VM) et librairies associées et de basculer facilement de l'un à l'autre.

```
gamba@karma:~$ rvm system
gamba@karma:~$ ruby -v
ruby 1.8.7 (2010-01-10 patchlevel 249) [x86_64-linux]
gamba@karma:~$ rvm 1.9.2
gamba@karma:~$ ruby -v
ruby 1.9.2p0 (2010-08-18) [x86_64-linux]
```

- la communauté

Les incontournables

Rails (2005)

Framework web

- MVC
- Convention over Configuration
- Don’t Repeat Yourself

À marqué le monde du développement web, il suffit de voir les copies

Exemples de réalisation:

- http://github.com
- Twitter
- Redmine

puppet, chef

- outils de spécification de configuration

Mon avis : Many Goods, Some Bads, No Ugly

Ruby c’est bien car …

- syntaxe
- possibilités
- écosystème: RAILS, réutilisation de bibilothèques C
- qualités des codes existants
- en lien avec son temps: XP, Agile, BDD …

Conséquences: rapide à mettre au point, ludique, instructif, addictif

Ruby, moins bien car …

- jeune (1995): moins de code existant, moins implanté dans les chaumières
- fragmentation des VMs
- efficacité toute relative (nette amélioration 1.9.2)
- programmation concurrente

Conclusion

Actualité

Le bon moment pour s'y mettre

- ruby 1.9.2
- passenger : module pour Apache/Nginx
- Rails 3

2 niveaux d'utilisation

- au niveau script de base : perl en plus (re)lisible
- à haut niveau : parfait *sauf* efficacité
- Ruby : langage proche de l'idéal POUR MOI (admin sys/développement web)

Questions

Gems utilisées pour cette présentation

- kramdown : convertisseur Markdomn => Html
- coderay : colorisation syntaxique
- slideshow : texte syntaxe wiki => présentation Html