

Ruby

votre prochain langage de programmation préféré

Guillaume Cottenceau

LinuxDays Genève 22 Mai 2007



À propos de l'intervenant

- Architecte logiciel spécialiste Linux
- Utilise des langages de programmation depuis une vingtaine d'années
- Utilise surtout Java professionnellement
- Amateur de Ruby depuis 6 ans
- Fondateur de Romandie.rb, groupe des utilisateurs de Ruby dans la région



Ruby est...

- Un langage interprété
- Rapide et facile à utiliser (productivité)
 - il suit le *principe de la moindre surprise*
- Complètement orienté-objet
- Un logiciel libre (GPL) qui a 12 ans, créé par Yukihiro Matsumoto au Japon
- Le langage de *Ruby on Rails*, ce qui a contribué à le faire connaître plus largement



Portée de cette présentation

- Illustrer l'utilisation de base de Ruby
- Montrer un éventail des principaux attraits de Ruby par rapport aux autres langages courants
- Susciter votre envie de découvrir plus avant le langage



Hello, world!

- La tradition (qui remonte apparemment à 1974) est de commencer par *Hello world...*

```
puts "Hello, world!"
```

```
=> Hello, world!
```

- Syntaxe simple et élégante, pas besoin de terminateur d'instruction ";" s'il n'y a qu'une seule instruction sur une ligne



Une API efficace

- Utilisation simple des regexp (comme Perl) :

```
if nom =~ /calmy.*rey/i
  puts "Bonjour présidente !"
end
```

- Beaucoup de méthodes utiles, par exemple :

```
chaine = "J'adore Ruby"
chaine["adore"] = "ADORE"
=> "J'ADORE Ruby"
```



Une API efficace

- Forme simplifiée des boucles, des intervalles et du formatage des chaînes :

```
for i in 1 .. 12
    puts "M. #{nom}, voyons le mois n°#{i}"
end
```

- Assignment en parallèle, par exemple pour tourner un groupe de variables :

```
a, b, c = b, c, a
```



Définir une classe

- On définit une classe avec le mot-clé **class**, et des méthodes avec le mot-clé **def** :

```
class Voiture
  def klaxon
    puts "pouet !"
  end
end
```



Appel de méthode

- On crée un objet avec la méthode de classe **new** et on appelle les méthodes avec l'opérateur “.” :

```
ma_voiture = Voiture.new
```

```
ma_voiture.klaxon
```

```
=> pouet !
```



Portée des variables

- Un caractère spécial précédant le nom d'une variable sert à traduire sa portée :
 - rien : variable locale
 - @ : variable d'instance (attribut)
 - @@ : variable de classe (variable statique)
 - \$: variable globale



Réouverture d'une classe

- On peut réouvrir une classe pour ajouter la création d'une variable de classe et d'une variable d'instance :

```
class Voiture  
  
    @@vitesse_min = 15  # variable de classe  
  
    def avance  
  
        @vitesse = 30  # variable d'instance  
  
    end  
  
end
```



Réouverture des classes de base

- On peut même réouvrir les classes de base :

```
class String
  def pair?
    return length % 2 == 0
  end
end

puts "Ruby".pair?
=> true
```



Tout est objet

- (Presque) tout est objet, y compris les nombres :

```
3.next
```

```
=> 4
```

```
1.2.floor
```

```
=> 1
```



Méthodes singleton

- Il est possible d'ajouter ou de spécialiser une méthode dans une instance d'objet uniquement :

```
def ma_voiture.klaxon
  puts "pweeeet !"
end
```

- Cette méthode n'existera pas pour les autres instances de la classe *Voiture*



Utilisation des opérateurs

- Les opérateurs mathématiques ont une signification utile dans plusieurs classes de base, par exemple “-” implémente la différence ensembliste pour les tableaux :

$[1, 2, 4, 8, 16] - [1, 2, 3, 5]$

$\Rightarrow [4, 8, 16]$



Utilisation des opérateurs

- Utilisation des opérateurs avec nos classes :

```
class Voiture
  def +(autre_voiture)
    return [ self, autre_voiture ]
  end
end

( ma_voiture + Voiture.new ).size
=> 2
```



Les blocs pour raccourcir le code

- Les blocs sont utilisés pour passer aisément du code plutôt que des données en paramètre d'une méthode :

```
nombre = [ 4, 8, 15, 16, 23, 42 ]  
nombre.each { |i| puts i }
```

```
File.readlines("/tmp/data")  
  .each { |ligne| puts ligne.length }
```



Les blocs comme fermetures

- Les blocs sont des *fermetures*, ils permettent une interaction avec l'environnement :

```
def add_progressbar(progress_monitor)

  pbar = Gtk::ProgressBar.new

  fenetre_principale.add(pbar)

  Gtk.timeout_add(100) {

    pbar.fraction = progress_monitor.get

  }

end
```



Les modules pour organiser

- Les modules servent à organiser le code :

```
module Format

  def Format.total_caracteres(chaines)

    nb = 0

    chaines.each { |ch| nb += ch.length }

    return nb

  end

end
```



Mixer les modules

- Les modules peuvent contenir des *méthodes d'instances*
- Elles sont ensuite disponibles pour les classes qui incluent ces modules (c'est un *mixin*)
- C'est presque comme de l'héritage multiple, mais ce n'est pas de la spécialisation - on ne peut pas hériter les modules



Exemple de mixin

- Le module standard *Comparable* a des méthodes d'instance qui définissent plusieurs opérateurs de comparaison (“<”...)
- Pour réaliser la comparaison demandée, les méthodes appellent l'opérateur “<=>”, en partant du principe qu'il sera disponible
- Dans une classe, il suffit donc de mixer *Comparable* et de définir “<=>” pour obtenir tous les opérateurs de comparaison à moindre coût



Exemple de mixin

```
class Voiture
  include Comparable

  def <=>(autre_voiture)
    return long <=> autre_voiture.long
  end
end

puts ma_voiture > autre_voiture
=> true
```



Threads dans l'interpréteur

- Ruby supporte le multi-threading
- L'implémentation est dans l'interpréteur :
 - pro : portable sur tous les OS, sans aucune API nécessaire ni effort de développement
 - con : tous les threads sont bloqués lors des appels système
- C'est très pratique dans la majorité des cas



Exemple de multi-threading

```
chemin_iso = demande_chemin(...)  
md5_ok = false  
thr = Thread.new {  
    md5_ok = calcule_md5_iso(chemin_iso)  
}  
demande_options_gravure(...)  
thr.join  
  
if md5_ok  
    ...
```



Autres fonctionnalités notables

- Exceptions
- Gestion automatique de la mémoire (garbage collector “mark & sweep”)
- Réflexivité
- Extensibilité



Ruby

votre prochain langage de programmation préféré

Guillaume Cottenceau

LinuxDays Genève 22 Mai 2007

