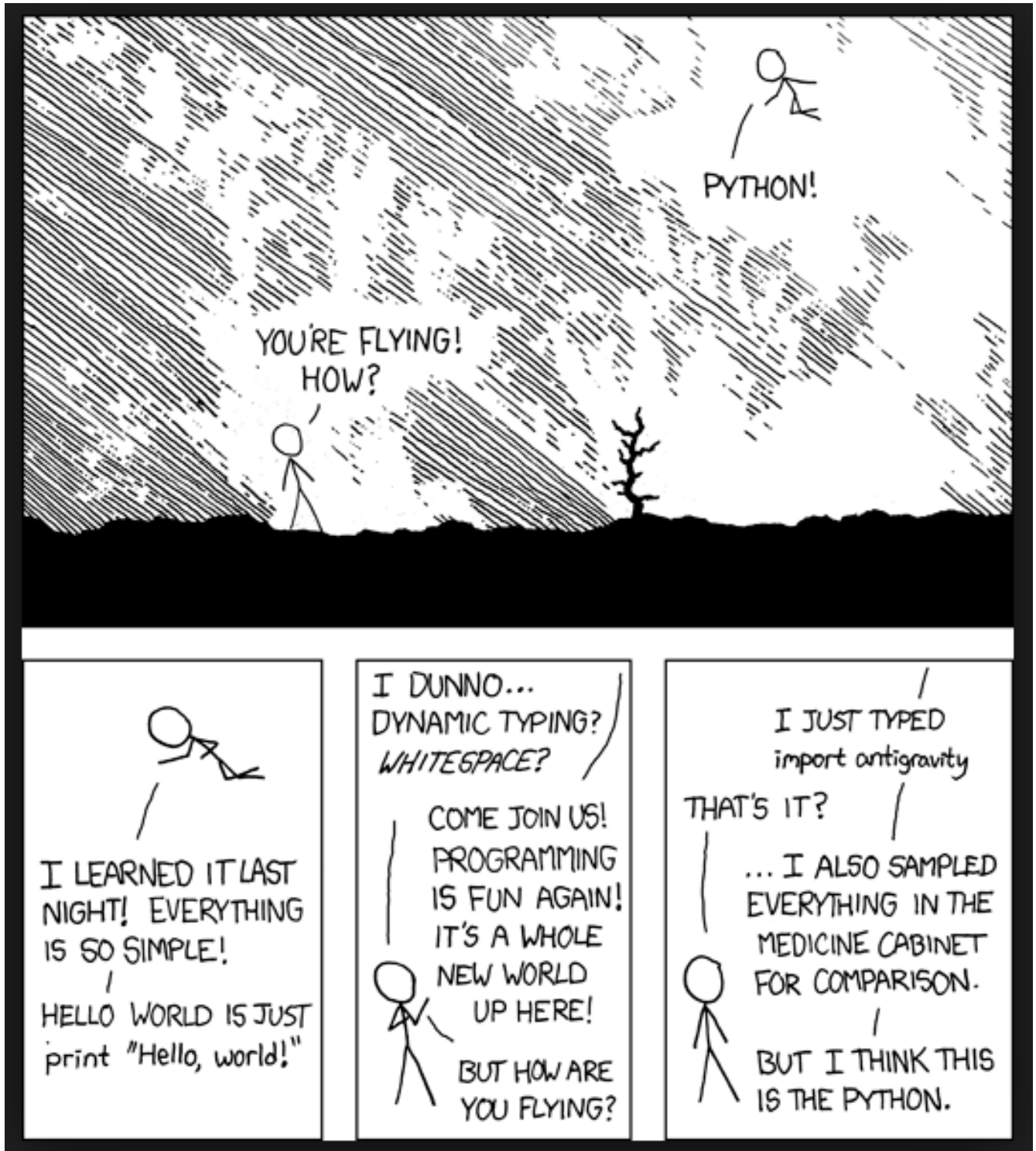
Compléments au vidéo «*Python for Ruby Programmers*» de M. Leone

Guy Tremblay
Professeur

Département d'informatique
UQAM
http://www.labunix.uqam.ca/~tremblay_gu

INF600A
4 décembre 2018


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Contenu

1. Introduction

Introduction

- Nous allons regarder la vidéo ($\approx$ 35 minutes) de Mike Leone intitulée «*Python for Ruby Programmers*»

  `www.youtube.com/watch?v=maSlTKMzR3Q`

  Cette présentation a été faite à la «*Los Angeles Ruby Conference 2013*».

- Pour les diapositives de cette présentation :

  `https://speakerdeck.com/mleone/python-for-ruby-programmers`

- Les diapositives qui suivent donnent quelques explications additionnelles, pour compléter la vidéo.

- Une autre ressource pour un survol rapide de Python :

  `www.slideshare.net/MattHarrison4/learn-90`

2. Zen of Python

The Zen of Python (PEP 20)

- Vingt (20) aphorismes qui décrivent les principes sous-jacents à Python et à son style de programmation
- Source :
  `https://www.python.org/dev/peps/pep-0020/`
- PEP = *Python Enhancement Proposals*

Aphorisme

Phrase, sentence qui résume en quelques mots une vérité fondamentale. (Exemple : Rien n'est beau que le vrai.)

Source: `http://www.larousse.fr/dictionnaires/francais/aphorisme/4459`

The Zen of Python (PEP 20)

- Vingt (20) aphorismes qui décrivent les principes sous-jacents à Python et à son style de programmation

Mais la liste de vingt aphorismes n'en contient. . . que 19

The Zen of Python (PEP 20)

- Vingt (20) aphorismes qui décrivent les principes sous-jacents à Python et à son style de programmation

Mais la liste de vingt aphorismes n'en contient. . . que 19

Numéro 20 =

- *20. You must discover this for yourself, grasshopper*
- *20. There are only 19*
- *20. There is no rule 20*

Une ou plusieurs façons de faire la même chose ?

Python — Règle #13

There should be one—and preferably only one—obvious way to do it.

Une ou plusieurs façons de faire la même chose ?

Python — Règle #13

There should be one—and preferably only one—obvious way to do it.

Ruby

Ruby inherited the Perl philosophy of having more than one way to do the same thing. [. . .] I want to give [Ruby users] the freedom to choose. People are different. People choose different criteria. But if there is a better way among many alternatives, I want to encourage that way by making it comfortable. So that's what I've tried to do.

Y. Matsumoto (concepteur de Ruby)

Source: `http://www.artima.com/intv/rubyP.html`

Une ou plusieurs façons de faire la même chose ?

Python — Règle #13

There should be one—and preferably only one—obvious way to do it.

8. Python Do you have permission from Your Leader to read this blog? Python is like the Scientology of Programming Languages. Everything has to be done the way the Prophet said. Or we are going to frown at you.

Look at us! We run a cult, and pretend it's a programming language.

Explicite vs. Implicite

Python — Règle #3

Explicit is better than implicit.

Explicite vs. Implicite

Python — Règle #3

Explicit is better than implicit.

Ruby

I love implicit code, or, as we call it in Ruby on Rails, Convention over Configuration.
[. . .]
I embrace implicit code. I embrace the context. I swoon over magic. If you too fancy such sparkle, Ruby on Rails will probably resonate. If not, no worries, most other programming environments pledge allegiance to the defeat of magic and oppose the implicit. Pick your poison.

D.H. Hansson (concepteur de Ruby on Rails)

Source:

`https://m.signalvnoise.com/programming-with-a-love-of-the-implicit-66629bb81ee7`

3. 13 similarités entre Python et Ruby

3. *Arrays*

Tableaux : `list` en Python, `Array` en Ruby

Les `list` Python ne sont pas aussi «dynamiques» que les `Array` de Ruby

```
>>> a = [10, 20]

>>> a[-1]
20
>>> a[-2]
10

>>> a[4] = 30
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
```

Ruby

```
>> a = [10, 20]; a[4] = 30; a
=> [10, 20, nil, nil, 30]
```

Tableaux : `list` en Python, `Array` en Ruby

Plusieurs méthodes de `list` ne peuvent pas être chainées

```
>>> a.append(30)
>>> a.append(40)
>>> a
[10, 20, 30, 40]

>>> a.append(50).append(60)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'NoneType' object
                has no attribute 'append'
```

Ruby

```
a << 30 << 40

a.push(50).push(60)
```

6. *Strong functional programming paradigms*

Style fonctionnel avec `map`, `filter` et al.

Les fonctions Python ne peuvent pas être chainées, donc les appels doivent être imbriqués

```
>>> a
[10, 20, 30, 40]

>>> def double(x):
...     return 2 * x
...

>>> filter( lambda x: x >= 50, map(double, a) )
[60, 80]
```

Ruby

```
>> a
=> [10, 20, 30, 40]

>> double = ->(x) { 2 * x }
> #<Proc:0x000000016042e8@(irb):16 (lambda)>

>> a.map(&double).select { |x| x >= 50 }
=> [60, 80]
```

Note : C'est bien «`&double`» et non «`&:double`» !

12. *Strong object reflection features*

Accès aux méthodes d'un objet

Pas de notion de `Symbol` en Python


```
>>> len(dir(10))
64

>>> dir(10)
['__abs__', '__add__', '__and__', '__class__', '__cmp__',
...
'__subclasshook__', '__truediv__', '__trunc__', '__xor__',
'bit_length', 'conjugate', 'denominator', 'imag', 'numerator',
'real']
```

Ruby

```
>> 10.methods.size
=> 130

>> 10.methods
=> [:to_s, :inspect, :-@, :+, :-, :*, :/, :div, :%,
    ...
    :enum_for, :equal?, :!, :!=,
    :instance_eval, :instance_exec, :__send__, :__id__]
```

4. 13 différences entre Python et Ruby

1. *No blocks*

Killer feature

In marketing terminology, a killer feature is any attribute of a product that, for a particular type of use, becomes essential to those users due to its usefulness. The killer feature effectively kills off competitors from the market that needs it.

Source: `https://en.wikipedia.org/wiki/Killer_feature`

2. *First-class Functions*

Les méthodes Ruby ne sont pas des objets de première classe

Ruby

```
>> def process_numbers( a, b, f ); f.call(a, b) end
=> :process_numbers

>> def plus(x, y); x + y end
=> :plus

>> process_numbers 10, 20, ->(x, y) { x + y }
=> 30

>> process_numbers 10, 20, plus
??

>> process_numbers 10, 20, :plus
??

>> process_numbers 10, 20, method(:plus)
??
```

Les méthodes Ruby ne sont pas des objets de première classe

Ruby

```
>> def process_numbers( a, b, f ); f.call(a, b) end
=> :process_numbers

>> def plus(x, y); x + y end
=> :plus

>> process_numbers 10, 20, ->(x, y) { x + y }
=> 30

>> process_numbers 10, 20, plus
...ArgumentError: wrong number of arguments (0 for 2)

>> process_numbers 10, 20, :plus
...NoMethodError (undefined method 'call' for :plus:Symbol)

>> process_numbers 10, 20, method(:plus)
=> 30
```

3. *One-line lambdas, by design*

On peut définir une méthode dans une autre méthode... mais elle est visible partout !

Ruby

```
>> def foo( x )
     def bar(x); 2*x end

     bar( x+1 )
   end
=> :foo

>> foo 10
=> 22

>> bar 20
=> 40
```

6. *Python forces indentation*

Python «*forces indentation*»


```
>>> def double(x):
... return 2*x
  File "<stdin>", line 2
    return 2*x
           ^
IndentationError: expected an indented block

>>> def double(x):
...     return 2*x
...
>>>
```

D'autres langages utilisent l'indentation, mais ce sont généralement des langages purement fonctionnels

Exemple : Miranda utilise le *offside rule* pour introduire des définitions auxiliaires avec `where`

Miranda

```
quadsolve a b c
  = error "Racines complexes", if delta < 0
  = [-b/(2*a)],                if delta = 0
  = [racine1, racine2],        if delta > 0
    where
      racine1 = -b/(2*a) + radix/(2*a)
      racine2 = -b/(2*a) - radix/(2*a)
      delta = b*b - 4*a*c
      radix = sqrt delta
```

8. *Instead of* `Enumerable`,
Python has built-in function

Seulement trois fonctions de base ☹

There are three built-in functions that are very useful when used with lists : `filter()`, `map()`, *and* `reduce()`

Source: `https://docs.python.org/2/tutorial/datastructures.html`

Mais. . .

Seulement trois fonctions de base ☹

There are three built-in functions that are very useful when used with lists : `filter()`, `map()`, *and* `reduce()`

Source: `https://docs.python.org/2/tutorial/datastructures.html`

Mais. . .
Python a aussi des constructions de *list comprehension*, comme dans les langages fonctionnels (Miranda, Haskell)

Quelques exemples simples de *list comprehension*


```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]


>>> [x**2 for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]


>>> [(x, y) for x in range(3) for y in ['a', 'b']]
[(0, 'a'), (0, 'b'), (1, 'a'), (1, 'b'), (2, 'a'), (2, 'b')]


>>> [(x, y) for x in range(3)
...          for y in ['ab', 'cd']
...          if len(y) != x]
[(0, 'ab'), (0, 'cd'), (1, 'ab'), (1, 'cd')]
```

C'est Miranda qui a introduit les *list comprehensions*... en 1985 !

Miranda

```
uns = 1 : uns

repeter a = r
            where r = a : r
uns' = repeter 1

nats = [0..]
pairs = [0, 2..]

premiers = crible [2..]
           where
           crible (p : xs) =
             p : crible [n | n <- xs; n mod p ~= 0]
```

C'est Miranda qui a introduit les *list comprehensions*... en 1985 !

Miranda

```
subsets []
  = [[]]
subsets (x : xs)
  = [[x] ++ y | y <- ys] ++ ys
    where
      ys = subsets xs
```

```
Miranda subsets [1, 2, 3]
[[1,2,3],[1,2],[1,3],[1],[2,3],[2],[3],[]]
```

Miranda (comme Haskell) est un langage fonctionnel pur... et paresseux

Miranda

```
Miranda take 10 uns
[1,1,1,1,1,1,1,1,1,1]

Miranda take 20 uns
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]

Miranda take 20 pairs
[0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38]

Miranda take 10 premiers
[2,3,5,7,11,13,17,19,23,29]
```

Langage fonctionnel paresseux
$\Rightarrow$ Évalue uniquement les expressions requises
$\Rightarrow$ Permet la manipulation de liste potentiellement infinies

11. `«self»` *as an argument to every class instance method*

Python oblige les méthodes d'instance à avoir un argument explicite `self`


```
class Document:
    def __init__( self, titre, annee, auteurs ):
        self.titre = titre; self.annee = annee
        self.auteurs = auteurs

    def titre( self ):
        return self.titre

    def annee( self ):
        return self.annee

    def auteurs( self ):
        return self.auteurs

    def __str__( self ):
        return "#<'%s', %s, %s>" % \
            ( self.titre, self.auteurs, self.annee )
```

Note : `__init__` ≡ `initialize` et `__str__` ≡ `to_s`.

Dans certains contextes, l'attribut `self` doit aussi être explicitement spécifié en Ruby ☹

Aucun `self` dans la méthode `incrementer`

Ruby

```
class Compteur
  attr_accessor :val

  def incrementer( x )
    val = val + x
  end
end
-----------------------------------
>> f = Compteur.new
=> #<Compteur:0x00000001e77588>
>> f.val = 23
=> 23
>> f.val
=> 23
>> f.incrementer( 88 )
NoMethodError: undefined method '+' for nil:NilClass
               from /home/tremblay_gu/INF5171/Programmes/Ruby/
...
```

Dans certains contextes, l'attribut `self` doit aussi être explicitement spécifié en Ruby ☹

Un `self` pour lire l'attribut dans la méthode `incrementer`

Ruby

```
class Compteur
  attr_accessor :val

  def incrementer( x )
    val = self.val + x
  end
end
-------------------------------------
>> f = Compteur.new
=> #<Compteur:0x00000001f8b4b0>
>> f.val = 23
=> 23
>> f.incrementer( 88 )
=> 111
>> f.val
=> 23
```

Dans certains contextes, l'attribut `self` doit aussi être explicitement spécifié en Ruby ☹

Un `self` pour écrire l'attribut dans la méthode `incrementer`

Ruby

```
class Compteur
  attr_accessor :val

  def incrementer( x )
    self.val = val + x
  end
end
-----------------------------------
>> f = Compteur.new
=> #<Compteur:0x00000001f8b4b0>
>> f.val = 23
=> 23
>> f.incrementer( 88 )
=> 111
>> f.val
=> 111
```

Note : Par défaut, lorsque Ruby rencontre un indentificateur à gauche d'une affectation, sans appel de méthode, il suppose que c'est une variable locale.

Python vs. Ruby : Argument `self` (méthodes d'instance) vs. préfixe `self` (méthodes de classe)

Une version de `Document` qui prend note de tous les documents créés et qui peut retourner ces documents

```
class Document:
    documents_crees = []

    @staticmethod
    def les_documents():
        return Document.documents_crees

    def __init__( self, titre, annee, auteurs ):
        self.titre = titre
        self.annee = annee;
        self.auteurs = auteurs
        Document.documents_crees.append(self)
-----------------------------------------------
print Document.les_documents()
```

Python vs. Ruby : Argument `self` (méthodes d'instance) vs. préfixe `self` (méthodes de classe)

Une version de `Document` qui prend note de tous les documents créés et qui peut retourner ces documents

```
class Document
  ...
  def self.les_documents
    @les_documents ||= []
  end

  def initialize( titre, annee, auteurs )
    @titre = titre
    @annee = annee
    @auteurs = auteurs
    Document.les_documents << self
  end
end
-----------------------------------------
puts Document.les_documents
```

13. *Ruby has stronger metaprogramming features*

On peut utiliser les annotations `property` et `setter` pour définir des attributs de style `attr_accessor` ★44

```
class Document:
    def __init__( self, titre, annee, auteurs ):
        ...

    @property
    def titre( self ):
        return self.titre

    @titre.setter
    def titre( self, nouveau_titre ):
        self.titre = nouveau_titre
    ...

if __name__ == "__main__": # Si dans programme principal
    d1 = Document("DSLs in Action", 2011, ["D. Ghosh"])
    d1.titre = "DSLs in Action (1st ed.)"
    print d1.titre # => "DSLs in Action (1st ed.)"
```

Un décorateur est un patron de conception bien connu

Décorateur (patron de conception)

En génie logiciel, un décorateur est le nom d'une des structures de patron de conception.

Un décorateur permet d'attacher dynamiquement de nouvelles responsabilités à un objet. Les décorateurs offrent une alternative assez souple à l'héritage pour composer de nouvelles fonctionnalités.

Source: `https://fr.wikipedia.org/wiki/Décorateur_(patron_de_conception)`

Un décorateur permet de modifier le comportement d'une autre méthode, par exemple, trace d'exécution ★46

Un peu comme la programmation par aspects (AOP = *Aspect Oriented Programming*)

```
def tracer(func):
   def func_tracee( x ):
      print "+++ On entre dans", func.__name__
      res = func( x )
      print "--- On sort de", func.__name__, "=>", res
      return res
   return func_tracee

@tracer
def inc( x ):
   print "On est dans foo: x = %s" % x
   return x + 1
-----------------------------------------
>>> foo( 12 )
+++ On entre dans inc
On est dans foo: x = 12
--- On sort de inc => 13
```

5. Conclusion

Maintenant que vous connaissez Ruby...

- Vous devriez pouvoir facilement approfondir Python par vous-même... si vous ne le connaissez pas déjà !?

Maintenant que vous connaissez Ruby...

- Vous devriez pouvoir facilement approfondir Python par vous-même... si vous ne le connaissez pas déjà !?
- Si vous voulez faire du développement d'applications Web...

Maintenant que vous connaissez Ruby...

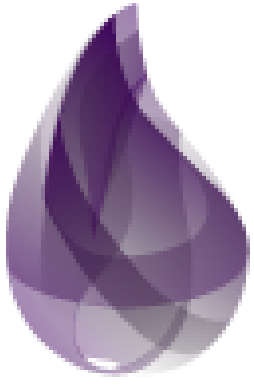
- Vous devriez pouvoir facilement approfondir Python par vous-même... si vous ne le connaissez pas déjà !?
- Si vous voulez faire du développement d'applications Web...

Maintenant que vous connaissez Ruby...

- Vous devriez pouvoir facilement approfondir Python par vous-même... si vous ne le connaissez pas déjà !?
- Si vous voulez faire du développement d'applications Web...Rails
- Si la programmation fonctionnelle concurrente vous intéresse...

Maintenant que vous connaissez Ruby...

- Vous devriez pouvoir facilement approfondir Python par vous-même... si vous ne le connaissez pas déjà !?
- Si vous voulez faire du développement d'applications Web...Rails
- Si la programmation fonctionnelle concurrente vous intéresse...



elixir

Pour terminer, une citation présentée... lors du premier cours !

No perfect language

That's Ruby's main difference from other language designs. I emphasize the feeling, in particular, how I feel using Ruby.

I didn't work hard to make Ruby perfect for everyone, because you feel differently from me. No language can be perfect for everyone. I tried to make Ruby perfect for me, but maybe it's not perfect for you.

The perfect language for Guido van Rossum[a] is probably Python.

Extrait d'une entrevue avec Y. Matsumoto (créateur de Ruby)

Source: `http://www.artima.com/intv/rubyP.html`

a. Le créateur de Python

Pour terminer, une citation présentée... lors du premier cours !

No perfect language

That's Ruby's main difference from other language designs. I emphasize the feeling, in particular, how I feel using Ruby.

I didn't work hard to make Ruby perfect for everyone, because you feel differently from me. No language can be perfect for everyone. I tried to make Ruby perfect for me, but maybe it's not perfect for you.

The perfect language for Guido van Rossum[a] is probably Python.

Extrait d'une entrevue avec Y. Matsumoto (créateur de Ruby)

Source: `http://www.artima.com/intv/rubyP.html`

---

a. Le créateur de Python

Pour terminer, une citation présentée... lors du premier cours !

No perfect language

That's Ruby's main difference from other language designs. I emphasize the feeling, in particular, how I feel using Ruby.

I didn't work hard to make Ruby perfect for everyone, because you feel differently from me. No language can be perfect for everyone. I tried to make Ruby perfect for me, but maybe it's not perfect for you.

The perfect language for Guido van Rossum[a] *is probably Python.*

Extrait d'une entrevue avec Y. Matsumoto (créateur de Ruby)

Source: `http://www.artima.com/intv/rubyP.html`

a. Le créateur de Python

Pour terminer, une citation présentée… lors du premier cours !

No perfect language

That's Ruby's main difference from other language designs. I emphasize the feeling, in particular, how I feel using Ruby.

I didn't work hard to make Ruby perfect for everyone, because you feel differently from me. No language can be perfect for everyone. I tried to make Ruby perfect for me, but maybe it's not perfect for you.

The perfect language for Guido van Rossum[a] is probably Python.

Extrait d'une entrevue avec Y. Matsumoto (créateur de Ruby)

Source: `http://www.artima.com/intv/rubyP.html`

---

a. Le créateur de Python

Pour terminer, une citation présentée... lors du premier cours !

No perfect language

That's Ruby's main difference from other language designs. I emphasize the feeling, in particular, how I feel using Ruby.

I didn't work hard to make Ruby perfect for everyone, because you feel differently from me. No language can be perfect for everyone. I tried to make Ruby perfect for me, but maybe it's not perfect for you.

The perfect language [pour vous est peut-être] Python.

Extrait d'une entrevue avec Y. Matsumoto (créateur de Ruby)

Source: `http://www.artima.com/intv/rubyP.html`

Pour terminer, une citation présentée... lors du premier cours !

No perfect language

That's Ruby's main difference from other language designs. I emphasize the feeling, in particular, how I feel using Ruby.

I didn't work hard to make Ruby perfect for everyone, because you feel differently from me. No language can be perfect for everyone. I tried to make Ruby perfect for me, but maybe it's not perfect for you.

The perfect language [pour vous est peut-être] Ruby.

Extrait d'une entrevue avec Y. Matsumoto (créateur de Ruby)

Source: `http://www.artima.com/intv/rubyP.html`

Pour terminer, une citation présentée… lors du premier cours !

No perfect language

That's Ruby's main difference from other language designs. I emphasize the feeling, in particular, how I feel using Ruby.

I didn't work hard to make Ruby perfect for everyone, because you feel differently from me. No language can be perfect for everyone. I tried to make Ruby perfect for me, but maybe it's not perfect for you.

The perfect language [pour vous est peut-être] ? ? .

Extrait d'une entrevue avec Y. Matsumoto (créateur de Ruby)

Source: `http://www.artima.com/intv/rubyP.html`

Mais au moins vous savez qu'il existe d'autres styles de langage que Java/C/C++… et donc vous avez plus de choix !

Python vs. Ruby vs. Perl

8. Python Do you have permission from Your Leader to read this blog? Python is like the Scientology of Programming Languages. Everything has to be done the way the Prophet said. Or we are going to frown at you.

Look at us! We run a cult, and pretend it's a programming language.

9. Ruby Look at me! I'm the language of the cool people. The ones who buy one cup of coffee, and sit for 8 hours in Starbucks to get free wifi. And talk loudly about how cool they are.

And in spite of all the talk about being programmer friendly, Ruby is used mainly by the Rails crowd, many of whom know no programming. Hey Ruby! Shut up for a minute, so the rest of us can talk.

10. Perl

Dudeyouaresuchamessylanguage,Isometimeswonderhowanyonewritesanythingwithyou.Ireallycan'tunderstandanycode.