MGL7460 — Automne 2016

Langages de script et langages dynamiques

Guy Tremblay
Professeur

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

8 septembre 2016

Aperçu

1. Qu’est-ce qu’un langage de script ?

Quels sont les langages de script que vous avez déjà utilisés ?

Quels sont d'autres langages de script que vous connaissez ?

Quelques langages de script

- *scripts shell* (`bash`, `csh`, etc.)
- sed
- awk
- Perl
- Tcl/Tk
- PHP
- Python
- Ruby
- JavaScript
- Groovy
- R
- Io
- Lua
- Elixir
- …

Qu’est-ce qu’un langage de script ?

Définition d'un langage de script

A scripting language is a programming language designed for integrating and communicating with other programming languages.

Source: `https://www.techopedia.com/definition/3873/scripting-language`

Définition d'un langage de script

Scripting language : A high-level programming language that is interpreted by another program at runtime rather than compiled by the computer's processor as other programming languages (such as C and C++) are.

Source: `http://www.webopedia.com/TERM/S/scripting_language.html`

Définition d'un langage de script

A scripting or script language is a programming language that supports scripts, programs written for a special run-time environment that automate the execution of tasks that could alternatively be executed one-by-one by a human operator.

Scripting languages are often interpreted (rather than compiled). Primitives are usually the elementary tasks or API calls, and the language allows them to be combined into more complex programs.

Source:

`https://en.wikipedia.org/wiki/Scripting_language`

Définition d'un langage de script

Mais...

Aucune de ces définitions... ne semble très utile pour comprendre ce qui caractérise ces langages ☹

Un langage de script est interprété : vrai ou faux ?

Exécution de programmes Ruby avec Ruby/MRI

```
$ 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|AR
007 leave
=> nil
```

Exécution de programmes Ruby avec JRuby

```
$ rvm use jruby

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

$ ruby hello0.rb
Bonjour le monde!

$ jrubyc hello0.rb

$ ls -l hello0.class
-rw-r-r-. 1 tremblay [...] hello0.class

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

Quelles sont les principales caractéristiques des langages de script ?

Quelques caractéristiques des langages de script

Selon Larry Wall, le concepteur du langage Perl

Here are some common memes floating around :

- *Simple language*
- *"Everything is a string"*
- *Rapid prototyping*
- *Glue language*
- *Process control*
- *Compact/concise*
- *Worse-is-better*
- *Domain specific*
- *"Batteries included"*
- *Easy onramps*[1]

Source: «*Programming is Hard, Let's Go Scripting*», L. Wall, 2007

1. Accès facile

Quelques caractéristiques des langages de script
Selon David Barron

- *Integrated compile and run*
- *Low overheads and ease of use*
- *Enhanced functionality*
- *Efficiency is not an issue*

Source: «*The world of scripting languages*», D. Barron, 2000

Quelques caractéristiques des langages de script

Selon David Barron (bis)

Perhaps the most important difference [between conventional programming languages and scripting languages] is that scripting languages incorporate features that enhance productivity of the user in one way or another [. . .]

Source: «*The world of scripting languages*», D. Barron, 2000

Quelques caractéristiques des langages de script

- *Simple language*
- *"Everything is a string"*
- *Rapid prototyping*
- *Glue language*
- *Process control*
- *Compact/concise*
- *Worse-is-better*
- *Domain specific*
- *"Batteries included"*
- *Easy onramps*

- *Integrated compile and run*
- *Low overheads and ease of use*
- *Enhanced functionality*
- *Efficiency is not an issue*

Quelques caractéristiques des langages de script

- *Simple language*
- *"Everything is a string"*
- *Rapid prototyping*
- *Glue language*
- *Process control*
- *Compact/concise*
- *Worse-is-better*
- *Domain specific*
- *"Batteries included"*
- *Easy onramps*

- *Integrated compile and run*
- *Low overheads and ease of use*
- *Enhanced functionality*
- *Efficiency is not an issue*

Quelques caractéristiques des langages de script

- *Simple language*
- *"Everything is a string"*
- *Rapid prototyping*
- *Glue language*
- *Process control*
- *Compact/concise*
- *Worse-is-better*
- *Domain specific*
- *"Batteries included"*
- *Easy onramps*

Qu'est-ce que ça signifie ?

- *Integrated compile and run*
- *Low overheads and ease of use*
- *Enhanced functionality*
- *Efficiency is not an issue*

Quelques caractéristiques des langages de script

- *Simple language*
- *"Everything is a string"*
- *Rapid prototyping*
- *Glue language*
- *Process control*
- *Compact/concise*
- *Worse-is-better*
- *Domain specific*
- *"Batteries included"*
- *Easy onramps*

Qu'est-ce que ça signifie ?

- *Integrated compile and run*
- *Low overheads and ease of use*
- *Enhanced functionality*
- *Efficiency is not an issue*

Compilation et exécution intégrées

Compilation et exécution intégrées : Contre-exemple Java

Code source

```
$ cat Bonjour.java
class Bonjour {
  public static void main( String[] args ) {
      System.out.println( "Bonjour le monde!" );
  }
}
```

Compilation et génération du fichier .class

```
$ javac Bonjour.java
```

Exécution du fichier .class

```
$ java Bonjour
Bonjour le monde!
```

Compilation et exécution intégrées : Exemple Ruby

Code source

```
$ cat hello0.rb

puts "Bonjour le monde!"
```

Compilation et exécution

```
$ ruby hello0.rb
Bonjour le monde!
```

Compilation et exécution intégrées : Exemple Ruby

Code source

```
$ ls -l hello1.rb
-rwxr-xr-x. 1 tremblay tremblay 46 26 jun
2015 hello1.rb*

$ cat hello1.rb
#!/usr/bin/env ruby
puts "Bonjour le monde!"
```

Compilation et exécution

```
$ ??
Bonjour le monde!
```

Compilation et exécution intégrées : Exemple Ruby

Code source

```
$ ls -l hello1.rb
-rwxr-xr-x. 1 tremblay tremblay 46 26 jun
2015 hello1.rb*

$ cat hello1.rb
#!/usr/bin/env ruby
puts "Bonjour le monde!"
```

Compilation et exécution

```
$ ./hello1.rb
Bonjour le monde!
```

Simplicité et concision

Simplicité et concision : Contre-exemple Java

Code source

```
$ cat BonjourBis.java
class BonjourBis {
  public static void main( String[] args ) {
    int n = Integer.parseInt( args[0] );

    for( int i = 0; i < n; i++ ) {
      System.out.println( "Bonjour le monde!" );
    }
  }
}
```

Compilation et génération du fichier .class puis exécution

```
$ javac BonjourBis.java
$ java BonjourBis 3
Bonjour le monde!
Bonjour le monde!
Bonjour le monde!
```

Simplicité et concision : Exemple Ruby

Code source

```
$ cat hello-bis.rb
#!/usr/bin/env ruby

?? puts "Bonjour le monde!" ??
```

Compilation et exécution

```
$ ./hello-bis.rb 3
Bonjour le monde!
Bonjour le monde!
Bonjour le monde!
```

Simplicité et concision : Exemple Ruby

Code source

```
$ cat hello-bis.rb
#!/usr/bin/env ruby

ARGV[0].to_i.times { puts "Bonjour le monde!" }
```

Compilation et exécution

```
$ ./hello-bis.rb 3
Bonjour le monde!
Bonjour le monde!
Bonjour le monde!
```

Glue language

Glue language

*A glue language is a programming language (usually an interpreted scripting language) that is designed or suited for writing **glue code**—code to connect software components.*

Source:

`https://en.wikipedia.org/wiki/Scripting_language`

Glue language

In the UNIX world, a glue language is one that is able to start up another program, collect its output, process it and perhaps pass it as input to a third program, and so on.

Source: «*The world of scripting languages*», D. Barron, 2000

Un script bash illustrant la notion de *glue code*

Effet : Compte le nombre de «mots» dans des documents LaTeX.

```
#!
cat *.tex \
 | sed '/\\begin{figure}/,/\\end{figure}/d' \
 | sed '/\\begin{table}/,/\\end{table}/d' \
   ...
 | sed '/\\documentclass/d' \
 | sed '/\\usepackage/d' \
 | sed '/\\input/d' \
 | sed 's/\\item//' \
 | sed 's/%.*$//'\
 | grep -v "^%" \
 | tr "~" " " \
 | tr "\t" "\n"\
 | tr " " "\n"\
 | grep -v '\\'  \
 | wc -w
```

Un script Ruby illustrant la notion de *glue code*

Effet = Imprime les noms des fichiers du répertoire courant qui contiennent tous les mots spécifiés en argument.

```
#!/usr/bin/env ruby

def fichiers_avec( mot )
  %x{ grep -l #{mot} * }.split("\n")
end

fichiers_avec_tous_les_mots = fichiers_avec( ARGV.shift )

while mot = ARGV.shift
  fichiers_avec_tous_les_mots &= fichiers_avec( mot )
end

p fichiers_avec_tous_les_mots
```

Un script Ruby illustrant la notion de *glue code*

Effet = Imprime les noms des fichiers du répertoire courant qui contiennent tous les mots spécifiés en argument.

```
#!/usr/bin/env ruby

def fichiers_avec( mot )
  %x{ grep -l #{mot} * }.split("\n")
end

fichiers_avec_tous_les_mots = fichiers_avec( ARGV.shift )

while mot = ARGV.shift
  fichiers_avec_tous_les_mots &= fichiers_avec( mot )
end

p fichiers_avec_tous_les_mots
```

Quelles sont des utilisations typiques des langages de script ?

Utilisations «traditionnelles»

- Administration de systèmes
- Contrôle d'applications et automatisation de tâches
- Traitement de données textuelles — `sed`, `awk`, `perl`
- Prototypage rapide

Utilisations plus «récentes» (Web !)

- Traitement de formulaires HTML — scripts CGI (`Perl`)
- HTML dynamique — *client-side scripting* (JavaScript)
- Sites Web générés dynamiquemqent — *server-side scripting* (Ruby on Rails, Node.js)

Pourquoi utilise-t-on des langages de script de haut niveau — plutôt que, par exemple, des scripts `bash` ?

Langages de script de bas niveau vs. langages de haut niveau

[C]onventional scripting languages have syntactic and other limitations, however, which tend to limit their efficiency and expressive power. For instance, even "advanced" Unix shells (e.g., bash and ksh) have very limited support for concurrency, data structuring, information hiding, object-oriented programming, regular expressions, etc.

Source: «*Scripting Languages*», R. Morin and V. Brown, MacTech, Vol. 15, no. 9, 1999

Langages de script de bas niveau vs. langages de haut niveau

*

Responding to these deficiencies, language developers have created extended scripting languages such as Perl, Python, and Tcl/Tk, among others. These languages are still able to invoke and glue together other programs, manipulate files, and set values on-the-fly, but they are also appropriate for writing much larger applications.

Source: «*Scripting Languages*», R. Morin and V. Brown, MacTech, Vol. 15, no. 9, 1999

2. Qu’est-ce qu’un langage dynamique ?

Qu’est-ce qu’un langage à typage dynamique ?

Typage statique vs. typage dynamique

Typage statique

Les vérifications de type sont faites à la compilation

$\Rightarrow$

Le programme ne pourra pas être exécuté s'il contient une erreur de type

$\Rightarrow$

Aucune erreur «de type» ne peut survenir à l'exécution

Typage dynamique

Le type d'un objet n'est pas connu à la compilation ou peut varier en cours d'exécution

$\Rightarrow$

Certaines vérifications peuvent/doivent être faites à l'exécution

Il existe plusieurs formes de typage statique

Typage monomorphique et explicite — C, Java, Go, etc.

```
$ more foo.c
int foo( int x ) {
  return 2 * x + 1;
}

int main() {
  foo( "abc" );
}
```

```
$ gcc foo.c
foo.c:6:8: warning: incompatible pointer to
      integer conversion passing 'char [4]' to
      parameter of type 'int' [-Wint-conversion]
  foo( "abc" );
       ^~~~~
```

Il existe plusieurs formes de typage statique

Typage polymorphique et implicite — Haskell

```
$ ghci
GHCi, version 7.10.3: http://www.haskell.org/ghc/[...]

Prelude> let foo x = 2 * x + 1

Prelude> foo "abc"
<interactive>:4:1:
    No instance for (Num [Char]) arising from a use of foo
    In the expression: foo "abc"

Prelude> :show bindings
foo :: Num a => a -> a = _
```

Donc : On peut avoir du typage statique même sans indiquer explicitement les types = ?

Il existe plusieurs formes de typage statique

L'inférence de types a été introduite par ML, un langage fonctionnel

Typage polymorphique et implicite — Haskell

```
$ ghci
GHCi, version 7.10.3: http://www.haskell.org/ghc/[...]

Prelude> let foo x = 2 * x + 1

Prelude> foo "abc"
<interactive>:4:1:
    No instance for (Num [Char]) arising from a use of foo
    In the expression: foo "abc"

Prelude> :show bindings
foo :: Num a => a -> a = _
```

Donc : On peut avoir du typage statique même sans indiquer explicitement les types = Inférence de types !

Il existe plusieurs formes de typage statique

L'inférence de types se généralise maintenant aux langages impératifs et objets

Typage implicite — C++11

```
int add( int x, int y ) {
    return x + y;
}

int main() {
  auto sum = add(5, 6); // [...] sum will be of type int
  return 0;
}
```

Typage implicite — C++14

```
auto add( int x, int y ) {
  return x + y;
}
```

Il existe plusieurs formes de typage statique

Certaines formes d'analyses de types peuvent être très sophistiquées

Typage avec *dependent types* — Idris

```
add: (x: Nat) -> (y: Nat)
              -> {auto smaller: LT x y}
              -> Nat
add x y = x + y

add 2 1       # => Erreur de compilation!!!!
```

Et il existe plusieurs formes de typage dynamique

Ruby = Duck typing — voir plus loin

```
$ irb
>> "3" + 3
??


>> "a" * 3
??

>> "3" == 3
??
```

Note : `irb` = *interactive Ruby* = *read-eval-print loop*

Et il existe plusieurs formes de typage dynamique

Ruby = Duck typing — voir plus loin

```
$ irb
>> "3" + 3
TypeError:
   no implicit conversion of Fixnum into String
   from (irb):5:in '+'
   from (irb):5
   from /Users/tremblay/.rvm/rubies/ruby-2.1.4/bin/irb:1
   '<main>'


>> "a" * 3
=> "aaa"

>> "3" == 3
=> false
```

Note : `irb` = *interactive Ruby* = *read-eval-print loop*

Et il existe plusieurs formes de typage dynamique

JavaScript = Effectue de très (trop ! ?) nombreuses conversions implicites

```
$ jsc-1
>>> "3" + 3
??

>>> "3" * 3
??

>>> "3" == 3
??
```

Et il existe plusieurs formes de typage dynamique

JavaScript = Effectue de très (trop ! ?) nombreuses conversions implicites

```
$ jsc-1
>>> "3" + 3
33

>>> "3" * 3
9

>>> "3" == 3
true
```

Le typage dynamique en Ruby

Typage dynamique en Ruby : Le type d'une variable peut changer en cours d'exécution

```
$ irb
>> p x
NameError: undefined local
   variable or method 'x'
   for main:Object
   ...

>> x = 2
=> 2

>> x.class
=> Fixnum

>> p x
2
=> 2
```

```
>> x = "abc"
=> "abc"

>> x.class
=> String

>> p x
"abc"
=> "abc"
```

Typage dynamique en Ruby : Certaines vérifications sont faites uniquement à l'exécution

Code source

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

def abs( x )
  if x >= 0 then x else -y end
end

p abs( ARGV[0].to_i )
```

Exemples d'appels du script

```
$ ./abs.rb 123
123

$ ./abs.rb -285
??
```

Typage dynamique en Ruby : Certaines vérifications sont faites uniquement à l'exécution

Code source

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

def abs( x )
  if x >= 0 then x else -y end
end

p abs( ARGV[0].to_i )
```

Exemples d'appels du script

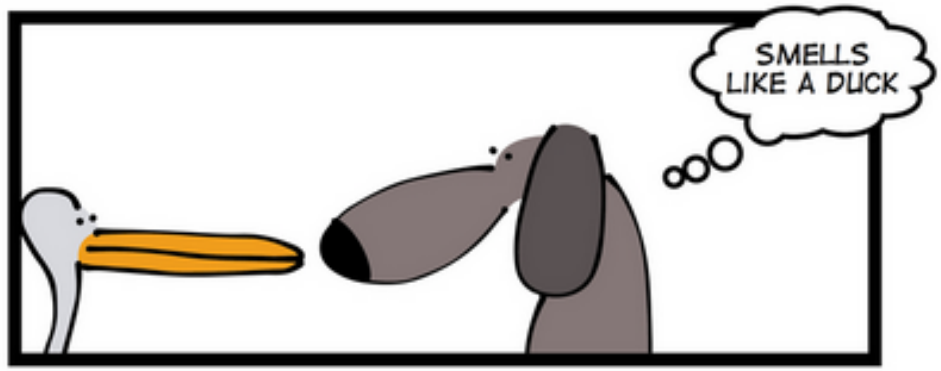
```
$ ./abs.rb 123
123

$ ./abs.rb -285
NameError: undefined local variable or method `y' for main:Object
   abs at ./abs.rb:4
   (root) at ./abs.rb:7
```

Typage dynamique en Ruby : «*Duck typing*»

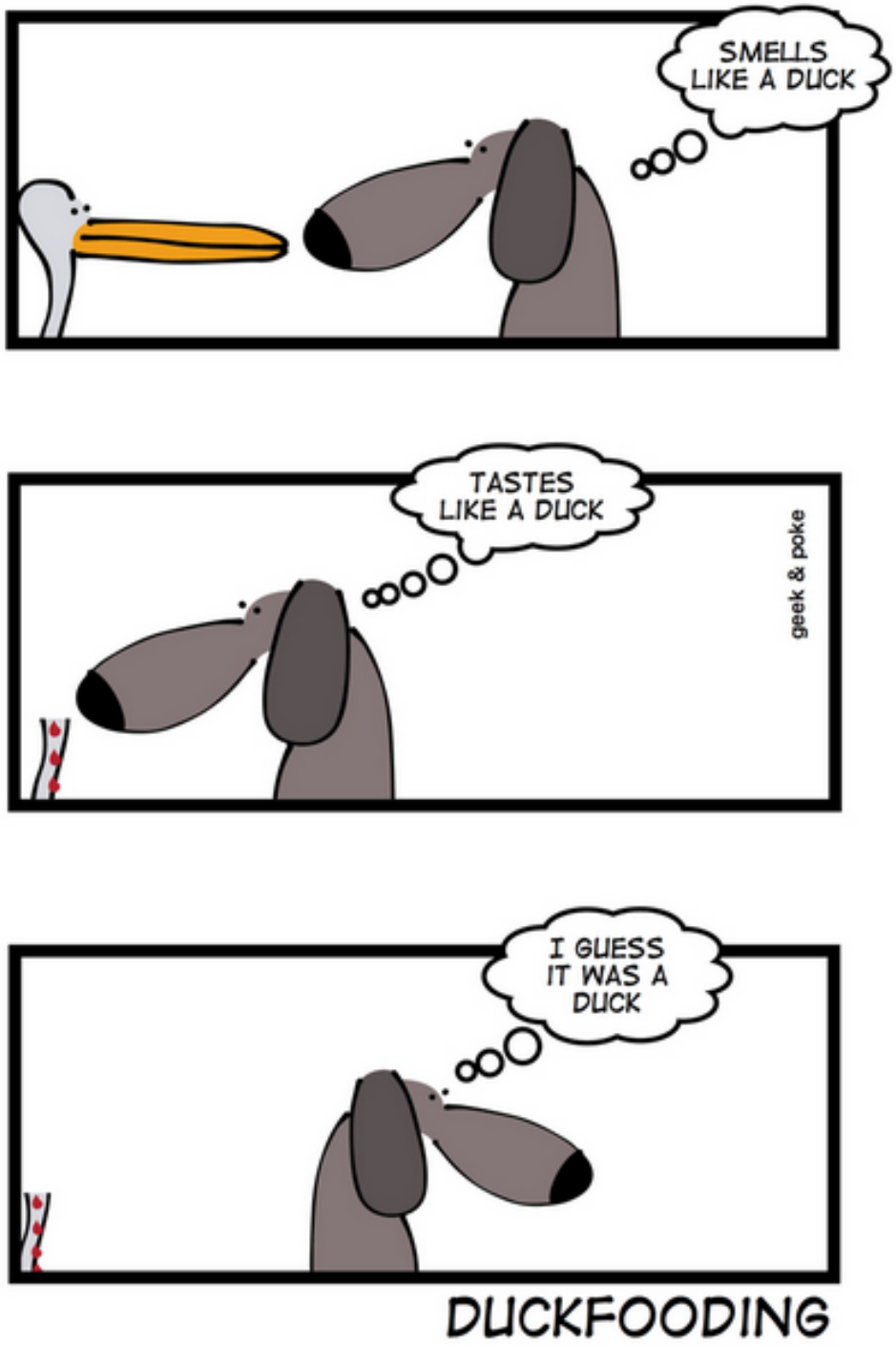
Qu'est-ce que le «*duck typing*» ?

Le type (ou la classe) d'un objet n'a pas d'importance.

Ce qui compte = les messages auxquels répond l'objet.



DUCKFOODING



If it walks like a duck,
and swims like a duck
and quacks like a duck,
then it must be a duck !

Typage dynamique en Ruby : «*Duck typing*»

Le *duck typing* permet une forme de typage polymorphique

Définition de méthode

```
def foo( a, b, c )
  a + b * c
end
```

Exemple d'utilisation

```
-10.respond_to? :+
=> true
+20.respond_to? :*
=> true


foo( -10, +20, 5 )
=> 90
```

Typage dynamique en Ruby : «*Duck typing*»

Le *duck typing* permet une forme de typage polymorphique

Définition de méthode

```
def foo( a, b, c )
  a + b * c
end
```

Exemple d'utilisation

```
"-10".respond_to? :+
=> true
"+20".respond_to? :*
=> true


foo( "-10", "+20", 5 )
=> ??
```

Typage dynamique en Ruby : «*Duck typing*»

Le *duck typing* permet une forme de typage polymorphique

Définition de méthode

```
def foo( a, b, c )
  a + b * c
end
```

Exemple d'utilisation

```
"-10".respond_to? :+
=> true
"+20".respond_to? :*
=> true


foo( "-10", "+20", 5 )
=> "-10+20+20+20+20+20"
```

Est-ce la seule caractéristique des langages dynamiques ?

Caractéristiques des langages dynamiques

Although software expert disagree on the exact definition, a dynamic language basically enables programs that can change their code and logical structures at runtime, adding variable types, module names, classes, and functions as they are running. These languages frequently are interpreted and generally check typing at runtime.

Source: «*Developers Shift to Dynamic Programmging Languages*», L.D. Paulson, IEEE Computer, Feb. 2007

Comportement dynamique des tableaux en Ruby : Allocation et modification dynamique

Et les tableaux sont hétérogènes !

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

>> a.size
=> 3

>> a[5]
=> nil

>> a[5] = "def"
=> ??

>> a.size
=> ??

>> a
=> ??
```

Comportement dynamique des tableaux en Ruby : Allocation et modification dynamique

Et les tableaux sont hétérogènes !

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

>> a.size
=> 3

>> a[5]
=> nil

>> a[5] = "def"
=> "def"

>> a.size
=> 6

>> a
=> [10, 20, 30, nil, nil, "def"]
```

Comportement dynamique des classes en Ruby : Une classe peut être modifiée dynamiquement

Réouverture d'une classe pour lui ajouter une méthode

```
class Foo
  def foo; "foo" end
end

f = Foo.new
=> #<Foo:0x13969fbe>

f.bar # => undefined method 'bar' for #<Foo:0x13969fbe>

class Foo
  def bar; "bar" end
end

f.bar
=> ??
```

Comportement dynamique des classes en Ruby : Une classe peut être modifiée dynamiquement

Réouverture d'une classe pour lui ajouter une méthode

```
class Foo
  def foo; "foo" end
end

f = Foo.new
=> #<Foo:0x13969fbe>

f.bar # => undefined method 'bar' for #<Foo:0x13969fbe>

class Foo
  def bar; "bar" end
end

f.bar
=> bar
```

Comportement dynamique des classes en Ruby : Une classe peut être modifiée dynamiquement (bis) *

Ajout dynamique d'une méthode à une classe

```
class Foo
  def foo; "foo" end
end

f = Foo.new
=> #<Foo:0x13969fbe>

f.baz # => undefined method 'baz' for #<Foo:0x13969fbe>

Foo.class_eval 'def baz; "baz" end'

f.baz
=> baz
```

Comportement dynamique des classes en Ruby : On peut réouvrir n'importe quelle classe ! !

Réouverture d'une classe «primitive»

```
class Fixnum
  def foo
    "foo" * self
  end
end

4.foo
=> ??
```

Comportement dynamique des classes en Ruby : On peut réouvrir n'importe quelle classe ! !

Réouverture d'une classe «primitive»

```
class Fixnum
  def foo
    "foo" * self
  end
end

4.foo
=> "foofoofoofoo"
```

Comportement dynamique en Ruby : Évaluation dynamique de code

Définition d'une classe `A`

```ruby
class A
  def initialize( x )
    @x = x
  end

  def val
    @x
  end
  private :val

  def plus( y )
    @x + y
  end
end
```

Quelques appels

```ruby
a = A.new( 10 )

a.instance_eval "@x"
=> 10

a.instance_eval "plus #{1+1}"
=> 12

a.val
=> NoMethodError: private
    method 'val' called [...]

a.instance_eval "val"
=> ??
```

Comportement dynamique en Ruby : Évaluation dynamique de code

Définition d'une classe `A`

```ruby
class A
  def initialize( x )
    @x = x
  end

  def val
    @x
  end
  private :val

  def plus( y )
    @x + y
  end
end
```

Quelques appels

```ruby
a = A.new( 10 )

a.instance_eval "@x"
=> 10

a.instance_eval "plus #{1+1}"
=> 12

a.val
=> NoMethodError: private
    method 'val' called [...]

a.instance_eval "val"
=> 10
```

3. Pourquoi apprendre de nouveaux langages ?

Pourquoi est-ce utile de connaitre plusieurs langages de programmation ?

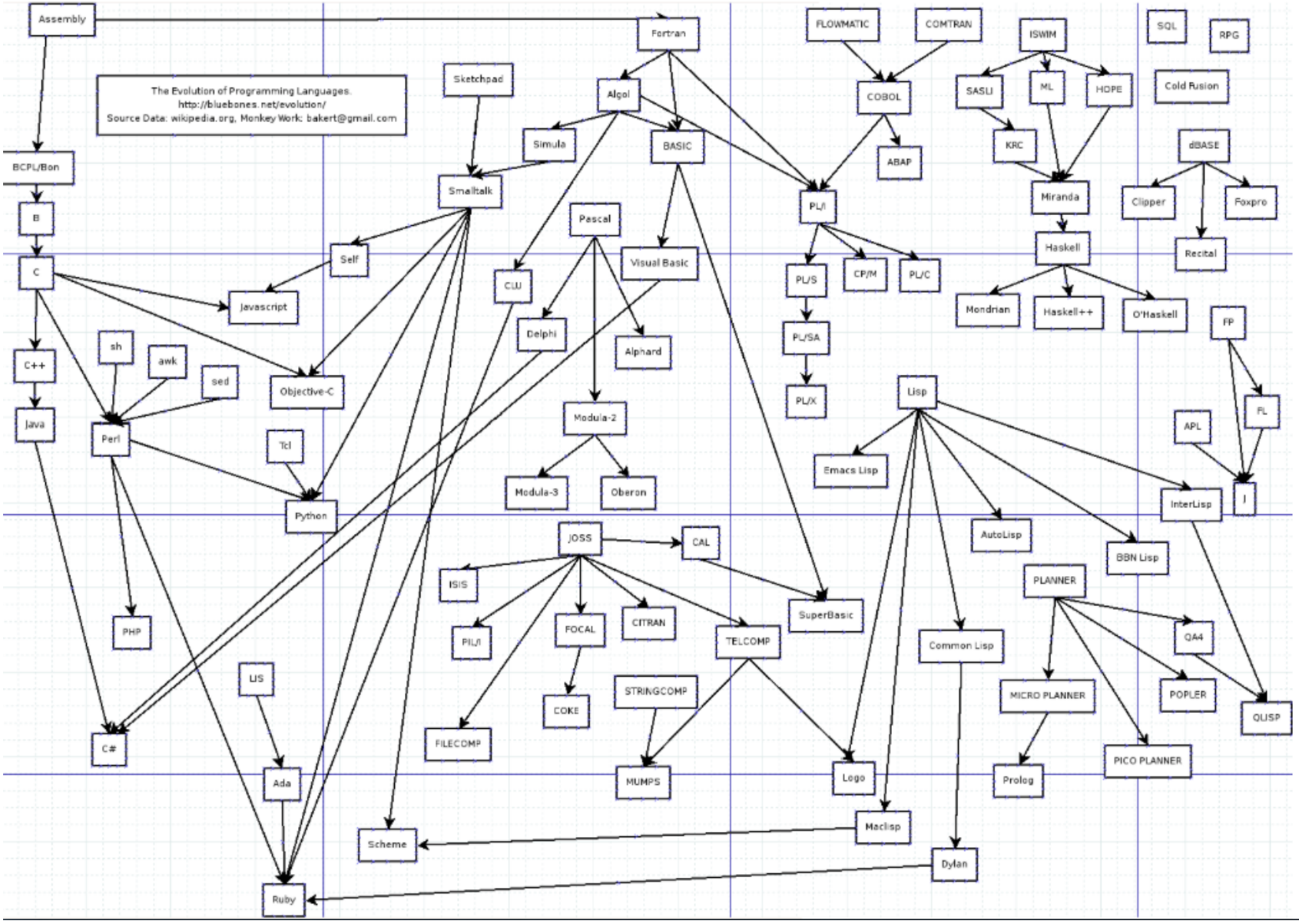
Il existe un (très !) grand nombre de langage de programmation

A catalog maintained by Bill Kinnersley of the University of Kansas lists about 2,500 programming languages. Another survey, compiled by Diamuid Piggot, puts the total even higher, at more than 8,500.

Source*: B. Hayes, «Computing Science : The Semicolon War», American Scientist, vol. 94, no. 4, p. 299-303, 2006.*

Généalogie de divers langages de programmation

https://www.madetech.com/blog/pros-and-cons-of-ruby-on-rails



«Si le marteau est le seul outil que vous avez, vous voyez des clous partout !»

La notion de «paradigme de programmation»

Programming paradigms are heuristics used for algorithmic problem solving. A programming paradigm formulates a solution for a given problem by breaking the solution down to specific building blocks and defining relationship among them.

Stolin & Hazzan, 2007

La notion de «paradigme de programmation»

Paradigme de programmation $\approx$ Façon d'aborder un problème de programmation, à l'aide de langages qui supportent bien certains mécanismes d'abstraction et de modularité.

Les principaux paradigmes de programmation et langages étudiés au bac. à l'UQAM

Impératif		**Déclaratif**	
Procédural	**Objet**	**Fonctionnel**	**Autre**
FORTRAN	Simula	Lisp/Scheme	**SQL**
Algol	Smalltalk	ML	**Prolog**
Pascal	**C++**	Miranda	
C	**Java**	**Haskell**	
bash	Ruby	Elixir	

Paradigme procédural	$\Rightarrow$	procédures sous-routines
Paradigme fonctionnel	$\Rightarrow$	valeurs fonctions
Paradigme objets	$\Rightarrow$	objets classes

Les principaux paradigmes de programmation et langages étudiés au bac. à l'UQAM

Impératif		**Déclaratif**	
Procédural	**Objet**	**Fonctionnel**	**Autre**
FORTRAN	Simula	Lisp/Scheme	**SQL**
Algol	Smalltalk	ML	**Prolog**
Pascal	**C++**	Miranda	
C	**Java**	**Haskell**	
bash	Ruby	Elixir	

- Langage avec typage statique et types explicites
- Langage avec typage statique et types implicites
- Langage avec typage dynamique
- Langage sans typage

En théorie, tous les langages de programmation sont aussi puissants les uns que les autres

Langage Turing-complet

En informatique ou en logique, un système Turing-complet est un système formel ayant une puissance de calcul au moins équivalente à celle des machines de Turing.

Source: `https://fr.wikipedia.org/wiki/Turing-complet`

Fait

Tous les langages mentionnés plus haut sont Turing-complets !

Fait

Tous les langages qui suivent sont Turing-complets !

Fait : Les langages suivants sont Turing-complets

Diverses versions d'un programme «Hello world !»

Befunge

```
0"!dlroW ,olleH">:#,_@
```

Brainfuck

```
++++++++++[>+++++++>++++++++++>+++>\
+<<<<-]>++.>+.+++++++..+++.>++.<<\
+++++++++++++++.>.+++.------.\
--------.>+.>.
```

Source: `http://esolangs.org/wiki/Hello_world_program_in_esoteric_languages`

Fait : Les langages suivants sont Turing-complets

Diverses versions d'un programme «Hello world !»

Verbose

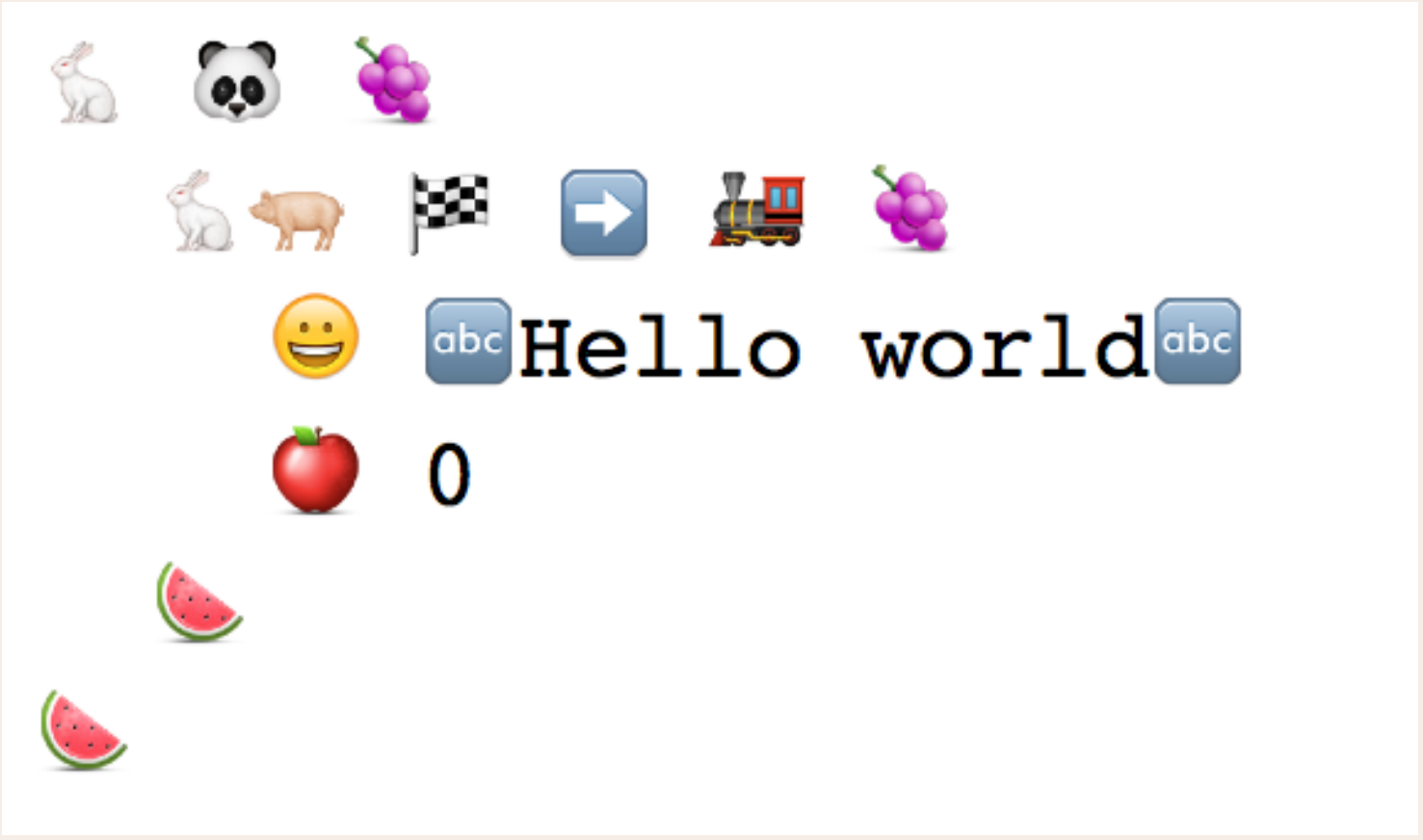
```
PUT THE NUMBER LXXII ONTO THE TOP OF THE PROGRAM STACK
GET THE TOP ELEMENT OF THE STACK AND CONVERT IT TO AN ASCII CHARACTER
AND OUTPUT IT FOR THE CURRENT PERSON USING THIS PROGRAM TO SEE
PUT THE NUMBER CI ONTO THE TOP OF THE PROGRAM STACK
GET THE TOP ELEMENT OF THE STACK AND CONVERT IT TO AN ASCII CHARACTER
AND OUTPUT IT FOR THE CURRENT PERSON USING THIS PROGRAM TO SEE
PUT THE NUMBER CVIII ONTO THE TOP OF THE PROGRAM STACK
GET THE TOP ELEMENT OF THE STACK AND CONVERT IT TO AN ASCII CHARACTER
AND OUTPUT IT FOR THE CURRENT PERSON USING THIS PROGRAM TO SEE
GET THE TOP ELEMENT OF THE STACK AND CONVERT IT TO AN ASCII CHARACTER
AND OUTPUT IT FOR THE CURRENT PERSON USING THIS PROGRAM TO SEE
PUT THE NUMBER CXI ONTO THE TOP OF THE PROGRAM STACK
...
AND OUTPUT IT FOR THE CURRENT PERSON USING THIS PROGRAM TO SEE
PUT THE NUMBER CVIII ONTO THE TOP OF THE PROGRAM STACK
GET THE TOP ELEMENT OF THE STACK AND CONVERT IT TO AN ASCII CHARACTER
AND OUTPUT IT FOR THE CURRENT PERSON USING THIS PROGRAM TO SEE
PUT THE NUMBER C ONTO THE TOP OF THE PROGRAM STACK
GET THE TOP ELEMENT OF THE STACK AND CONVERT IT TO AN ASCII CHARACTER
AND OUTPUT IT FOR THE CURRENT PERSON USING THIS PROGRAM TO SEE
PUT THE NUMBER XXXIII ONTO THE TOP OF THE PROGRAM STACK
GET THE TOP ELEMENT OF THE STACK AND CONVERT IT TO AN ASCII CHARACTER
AND OUTPUT IT FOR THE CURRENT PERSON USING THIS PROGRAM TO SEE
```

Source: `http://esolangs.org/wiki/Hello_world_program_in_esoteric_languages`

Fait : Les langages suivants sont Turing-complets

Diverses versions d'un programme «Hello world !»

Emoji code (http://www.emojicode.org/)



Source: https://gist.github.com/EranAvidor/0b9a491bbe365bbf2cbe

Fait : Les langages suivants sont Turing-complets

Diverses versions d'un programme «Hello world !»

Anguish

Here's an Anguish program that prints `Hello World`:

Fait : Les langages suivants sont Turing-complets

Diverses versions d'un programme «Hello world !»

Anguish

Here's an Anguish program that prints `Hello World`:

You may be familiar with funky esoteric languages like Ook or even Whitespace. Those are fun and neat, but I've decided to dial up the crazy a notch and make a completely invisible programming language!

Source: `http://blogs.perl.org/users/zoffix_znet/2016/05/anguish-invisible-programming-language-and-invisible-data-theft.html`

Différents langages sont généralement utilisés pour des tâches différentes

- Un langage de programmation est un outil
- Des tâches différentes requièrent des outils différents
- Les divers langages ont des forces/faiblesses variées :
  - Portabilité
  - Sécurité
  - Efficacité d'exécution
  - Simplicité et expressivité $\Rightarrow$ Efficacité de programmation

Différents langages sont généralement utilisés pour des tâches différentes

- Un langage de programmation est un outil
- Des tâches différentes requièrent des outils différents

Différents langages sont généralement utilisés pour des tâches différentes

- Un langage de programmation est un outil
- Des tâches différentes requièrent des outils différents
- Les divers langages ont des forces/faiblesses variées :
  - Portabilité
  - Sécurité
  - Efficacité d'exécution
  - Simplicité et expressivité $\Rightarrow$ Efficacité de programmation

Les programmes sont conçus pour être exécutés, mais aussi pour être lus et modifiés

Assembleur

```
section .data
        helloMsg:     db 'Hello world!',10
        helloSize:    equ \$-helloMsg
section .text
        global _start
_start:
        mov eax,4
        mov ebx,1
        standard)
        mov ecx, helloMsg
        mov edx, helloSize
        int 80h

        mov eax,1
        mov ebx,0
        int 80h
```

Brainfuck

```
++++++++++[>+++++++>++++++++++>+++>\
+<<<<-]>++.>+.+++++++..+++.>++.<<\
+++++++++++++++.>.+++.------.\
--------.>+.>.
```

Ruby

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

4. Pourquoi Ruby ?

Pourquoi Ruby ?

Parce que Ruby est un langage puissant et élégant !

More than any language with which we have worked, Ruby stays out of your way. You can concentrate on solving the problem at hand, instead of struggling with compiler and language issues. That's how it can help you become a better programmer : by giving you the chance to spend your time creating solutions for your users, not for the compiler.

«Programming Ruby (First edition)», A. Hunt & D. Thomas

Note : A. Hunt et D. Thomas sont les auteurs de «The Pragmatic Programmer—From Journeyman to Master» (2000).

Pourquoi Ruby ?

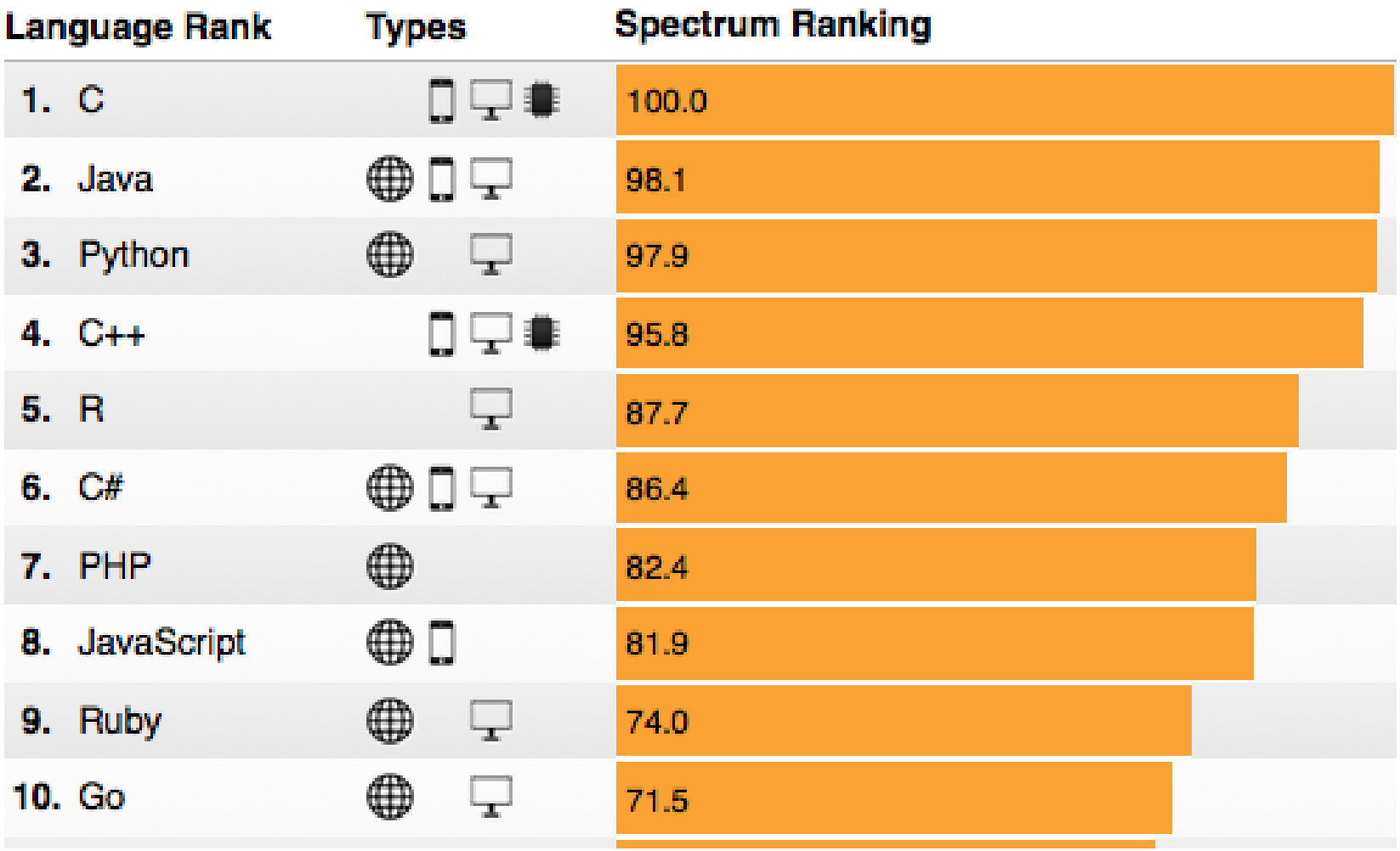
Parce que Ruby est un langage puissant et élégant !

Ruby *is "made for developer happiness" !*

Y. Matsumoto (concepteur du langage Ruby)

Pourquoi Ruby ?

Ruby fait partie des 10 premiers langages du palmarès «*The 2016 Top Programming Languages*» (IEEE Spectrum).



Source: `http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2016`

Note : La liste comporte 48 langages et «*Rankings are created by weighting and combining 12 metrics from 10 sources.*»

Pourquoi Ruby ?

Parce que. . . Ruby on Rails !

*Ruby on Rails, or simply **Rails**, is a web application framework written in Ruby [. . .].*

Rails is a model–view–controller (MVC) framework, providing default structures for a database, a web service, and web pages. [. . .]

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

Pourquoi Ruby ?

Historique de Ruby et Ruby on Rails

Ruby	
Version	**Année**
1.0	1996
...	...
Progr. Ruby	2000
...	...
1.8	2003
...	...
1.9	2007
...	...
2.0	2013
...	...
2.3.0	Déc. 2015

Rails	
Version	**Année**
1.0	2005
...	...
2.0	2007
...	...
3.0	2010
...	...
4.0	2013
...	...
4.2.5	Nov. 2015

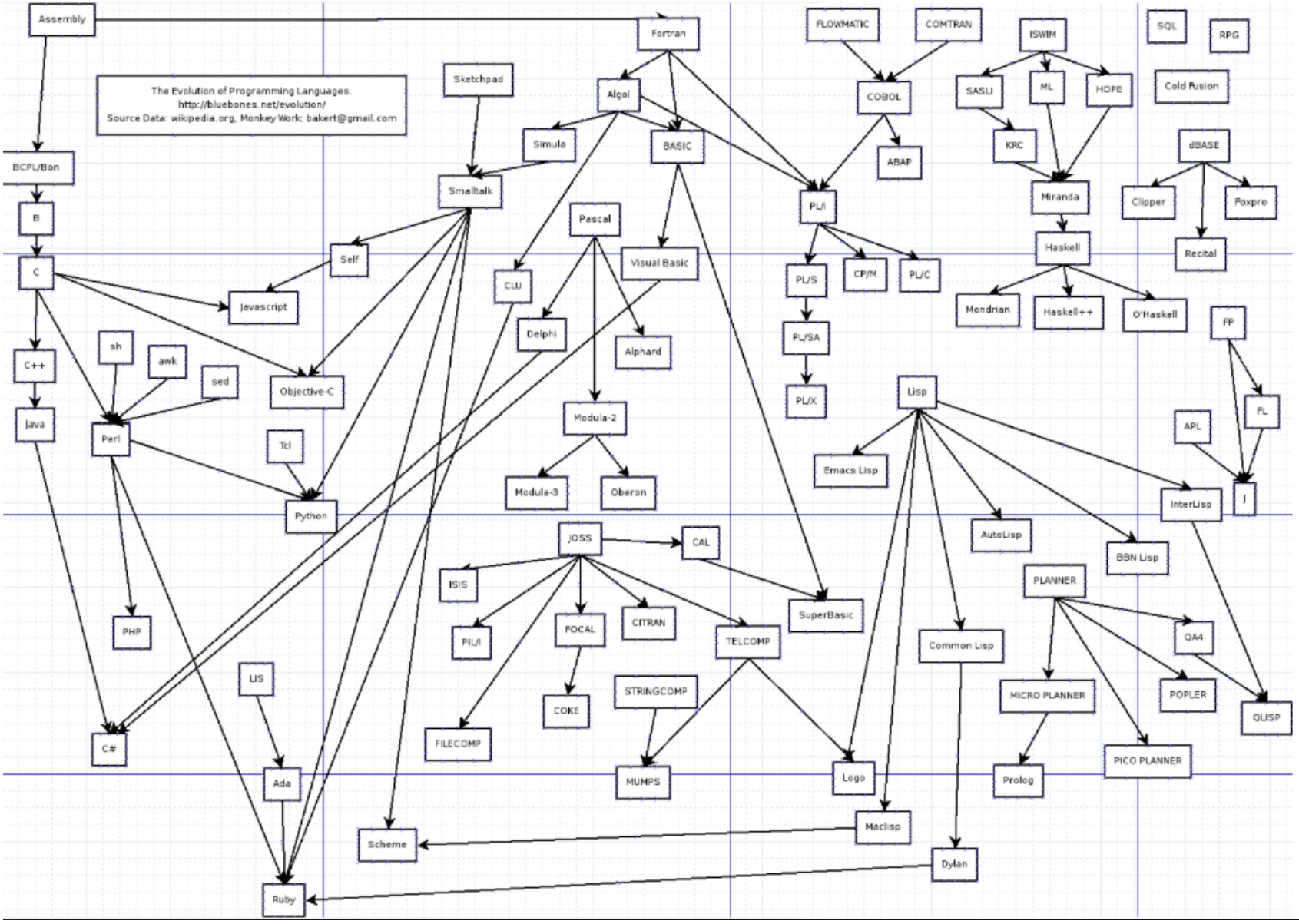
Pourquoi Ruby ?
Parce que... Ruby on Rails

Quelques compagnies qui utilisent Rails :

- Twitter
- Shopify
- Airbnb
- Heroku
- Github
- Couchsurfing
- Kickstarter
- Slideshare
- ...

Généalogie de divers langages de programmation

https://www.madetech.com/blog/pros-and-cons-of-ruby-on-rails



Pourquoi Ruby ?
Les ancêtres du langage Ruby

Langage	**Année**	**Caractéristiques**
Lisp	1958	Approche fonctionnelle Métaprogrammation
CLU	1974	Itérateurs
Smalltalk	1980	Langage objet pur, blocs de code GUI, sUnit
Eiffel	1986	*Uniform Access Principle*
Perl	1987	Expressions régulières et *pattern matching*
Ruby	1993	

Pourquoi Ruby ?

Parce que les programmeurs ont des «goûts» et des styles de programmation différents

- Ruby est un langage de script à typage dynamique
- Python est un langage de script à typage dynamique

Pourquoi Ruby ?

Parce que les programmeurs ont des «goûts» et des styles de programmation différents

- Ruby est un langage de script à typage dynamique
- Python est un langage de script à typage dynamique

$\Rightarrow$

- Si on connaît Python, il est facile d'apprendre Ruby

Pourquoi Ruby ?

Parce que les programmeurs ont des «goûts» et des styles de programmation différents

- Ruby est un langage de script à typage dynamique
- Python est un langage de script à typage dynamique

$\Rightarrow$

- Si on connaît Ruby, il est facile d'apprendre Python

Pourquoi Ruby ?

Parce que les professeurs qui programment ont des «goûts» et des styles de programmation différents

- Ruby est un langage de script à typage dynamique
- Python est un langage de script à typage dynamique

$\Rightarrow$

- Si on connaît Ruby, il est facile d'apprendre Python

Parce que j'utilise Ruby dans plusieurs projets...

L'outil de correction Oto

(noyau, pas l'application Web)

Avec les tests unitaires mais sans les tests fonctionnels

Language	files	blank	comment	code
Ruby	**242**	4158	2458	**12430**
Java	78	1075	852	4116
Haskell	11	231	282	437
C Shell	5	27	26	160
Bourne Shell	9	54	28	149
C	10	17	1	148
make	2	43	6	91
Korn Shell	1	2	0	8
Bourne Again Shell	3	1	0	6
C/C++ Header	1	0	0	1
SUM:	362	5608	3653	17546

La bibliothèque de programmation parallèle PRuby

Avec vs. sans les tests unitaires et les tests fonctionnels

Avec les tests unitaires et les tests fonctionnels

Language	files	blank	comment	code
Ruby	**120**	2028	2141	**6929**
make	1	7	0	14
SUM:	121	2035	2141	6943

La bibliothèque de programmation parallèle PRuby

Avec vs. sans les tests unitaires et les tests fonctionnels

Sans les tests unitaires et les tests fonctionnels

Language	files	blank	comment	code
Ruby	**15**	294	941	**1003**
make	1	7	0	14
SUM:	121	2035	2141	6943

5. Utilisation de Ruby dans le cours MGL7460

Utilisation de Ruby dans le cours MGL7460

- Prochaine partie de cours : Présentation de quelques points clés de Ruby = «Introduction au langage Ruby par des exemples»

Utilisation de Ruby dans le cours MGL7460

- Prochaine partie de cours : Présentation de quelques points clés de Ruby = «Introduction au langage Ruby par des exemples»
- **À vous de lire plus en détail les notes de cours** ou tout autre matériel disponible sur le Web (beaucoup !)

Utilisation de Ruby dans le cours MGL7460

- Prochaine partie de cours : Présentation de quelques points clés de Ruby = «Introduction au langage Ruby par des exemples»
- **À vous de lire plus en détail les notes de cours** ou tout autre matériel disponible sur le Web (beaucoup !)

- Quelques travaux pratiques en laboratoire utiliseront Ruby — tests unitaires, DSL, métaprogrammation, intégration continue, ...

Utilisation de Ruby dans le cours MGL7460

- Prochaine partie de cours : Présentation de quelques points clés de Ruby = «Introduction au langage Ruby par des exemples»
- **À vous de lire plus en détail les notes de cours** ou tout autre matériel disponible sur le Web (beaucoup !)

- Quelques travaux pratiques en laboratoire utiliseront Ruby — tests unitaires, DSL, métaprogrammation, intégration continue, ...

- Le premier projet sera en Ruby [2] — tests unitaires avec *expectations*

2. À moins que vous ne connaissiez déjà très bien Ruby !