INF600A Langages de script et langages dynamiques

Guy Tremblay
Professeur

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

11 janvier 2017

Aperçu

1. INF600X Sujets spéciaux en informatique et génie logiciel

INF600X Sujets spéciaux en informat. et génie logiciel

Objectifs

Ce cours à contenu variable vise à permettre d'aborder de nouvelles approches prometteuses en informatique et génie logiciel non couvertes par les autres activités de la banque de cours.

Sommaire du contenu

Le contenu du cours variera selon l'évolution du domaine.

INF600A Langages de script et langages dynamique

- Instance du cours INF600X
- Nouveau cours que j'ai proposé à l'automne 2015
- Cours offert pour la première fois à l'hiver 2016

2. INF600A : Description du cours

Description «officielle» du cours INF600A

Objectifs

L'objectif du cours est l'introduction à la programmation à l'aide de langages de script et de langages dynamiques.

Description «officielle» du cours INF600A

Contenu

Principales caractéristiques des langages de script.

Introduction à un langage de script de bas niveau : variables, structures de données et de contrôle, définitions et appels de fonction, manipulation de chaînes et *pattern-matching*.

Introduction à un langage de script de haut niveau : Interprétation, compilation et exécution ; évaluation interactive ; variables, méthodes et typage dynamique ; structures de données (tableaux, chaînes, symboles, hashes) ; structures de contrôle ; définitions et appels de méthodes ; classes et modules ; manipulation de chaînes, expressions régulières et pattern-matching ; lambda-expressions et blocs/fermetures.

Description «officielle» du cours INF600A

Contenu (suite)

Aperçu de langages de script avancés : langage objet avec prototypes, langage fonctionnel parallèle, langage pour GUI.

Utilisation d'un langage de script pour la coordination de programmes et de tâches (glue langage) ; traitement de données textuelles (*pattern matching*) ; mise en œuvre de DSL (langages spécifiques au domaine) (fluent interface, métaprogrammation). Exemples avancés d'outils et de DSL : cadre de tests unitaires, cadre de tests d'acception, assemblage de logiciels, développement d'applications en lignes de commandes, développement d'applications Web.

Approfondissement d'un langage de script de haut niveau : Métaprogrammation, extension de classes, métaclasses, méthodes appelées ou créées dynamiquement, appels indirects via des «*hooks*».

3. INF600A : De quoi traite ce cours ?

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*
- sed, awk
- Perl
- PHP
- Tcl/Tk
- Python
- JavaScript
- Ruby
- Groovy
- R
- Io
- Lua
- Elixir
- …

3.1 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 claire ou utile ☹

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*[*]

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

∗. Accès facile

Quelques caractéristiques des langages de script

Selon David Barron, auteur d'un livre sur les langages de script

- *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 : 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

```
$ 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

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

Compilation et exécution

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

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

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 ?

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

3.2 Qu’est-ce qu’un langage dynamique ?

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

Typage dynamique en Ruby : Exemple simple

```
$ 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"
```

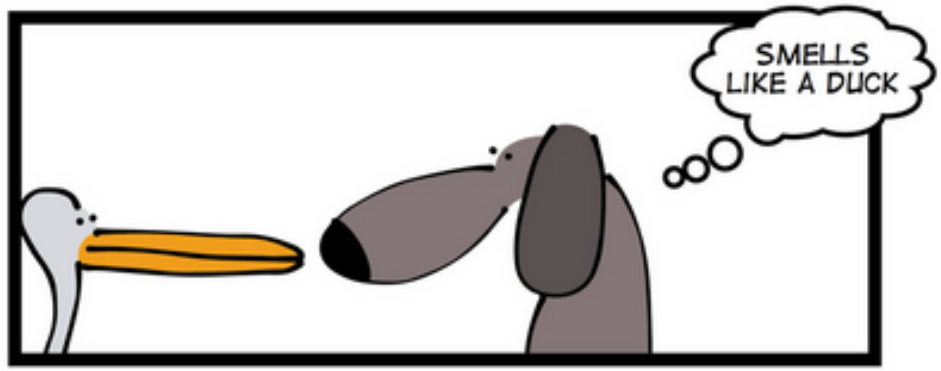
Note : `irb` = *interactive Ruby* = *read-eval-print loop*

Typage dynamique en Ruby : «*Duck typing*»

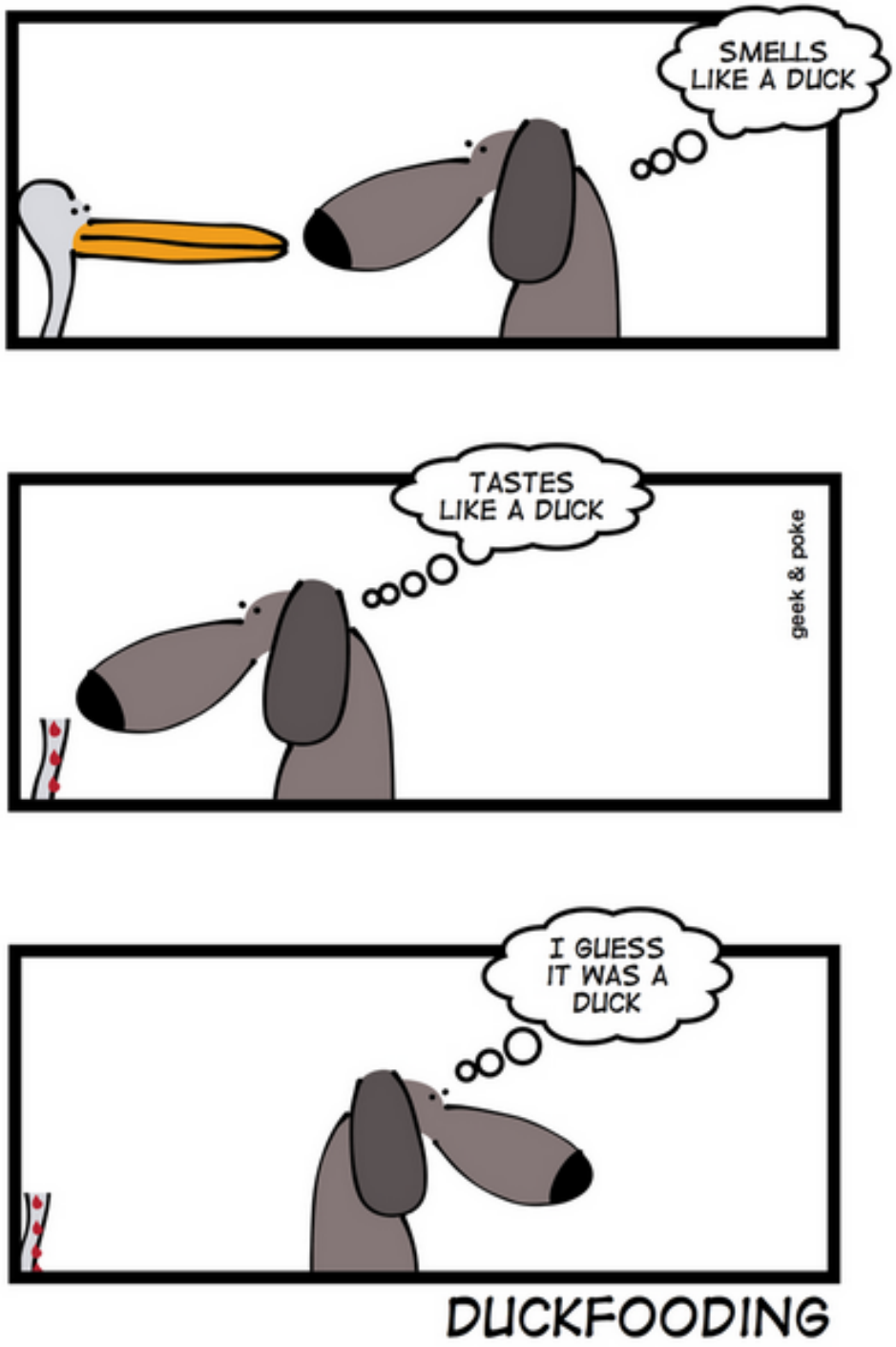
Qu'est-ce que le «*duck typing*» ?

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"
```

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

Code source

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

def abs( x )
  x >= 0 ? x : y
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 )
  x >= 0 ? x : y
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
```

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`

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

  def val
    @x
  end
  private :val

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

Quelques appels

```
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`

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

  def val
    @x
  end
  private :val

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

Quelques appels

```
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
```

3.3 Pourquoi apprendre de nouveaux langages ?

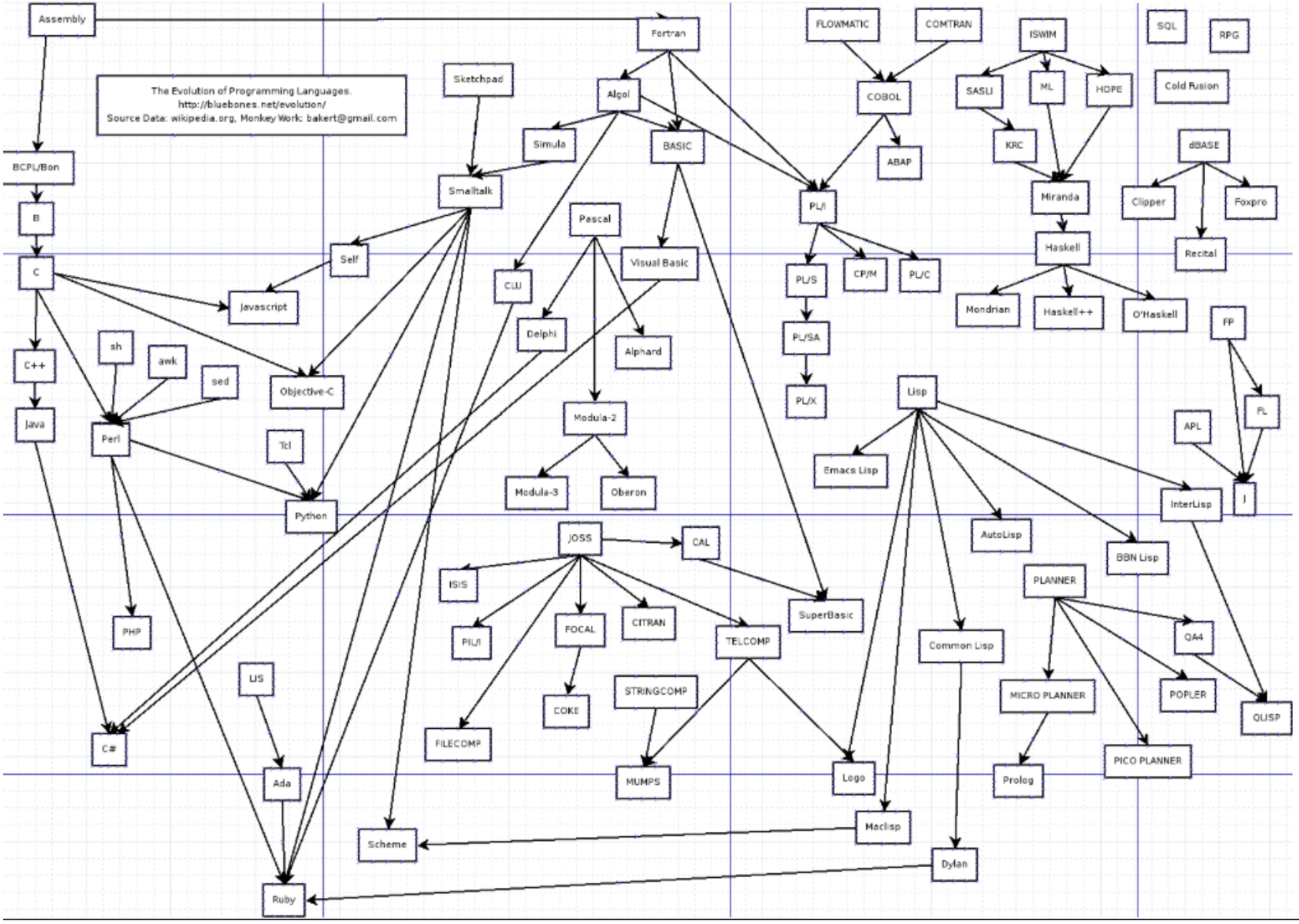
Il existe un (très !) grand nombre de langage de programmation

A catalog maintained by Bill Kiinersley 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



Quand le marteau est le seul outil qu'on connait, on voit des clous partout !

Quand le marteau est le seul outil qu'on connait, on voit des clous partout !

Autres variantes :

- *«I call it the law of the instrument, and it may be formulated as follows : Give a small boy a hammer, and he will find that everything he encounters needs pounding.»* (Kaplan, 1964)
- *«I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail.»* (Maslow, 1966)
- *«If all you have is a hammer. . . eventually you'll hit a nail !»*

Des nouveaux langages nous fournissent des nouveaux concepts, des nouveaux outils

*

A programmer can think quite well in just about any language. Many of us cut our teeth in BASIC, and simply learning how to think computationally allowed us to think differently than we did before. But then we learn a radically different or more powerful language, and suddenly we are able to think new thoughts, thoughts we didn't even conceive of in quite the same way before.

It's not that we need the new language in order to think, but when it comes along, it allows us to operate in different ways. New concepts become new tools.

Source: `http://www.cs.uni.edu/~wallingf/blog/archives/monthly/2016-12.html#e2016-12-16T14_14_26.htm`

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»

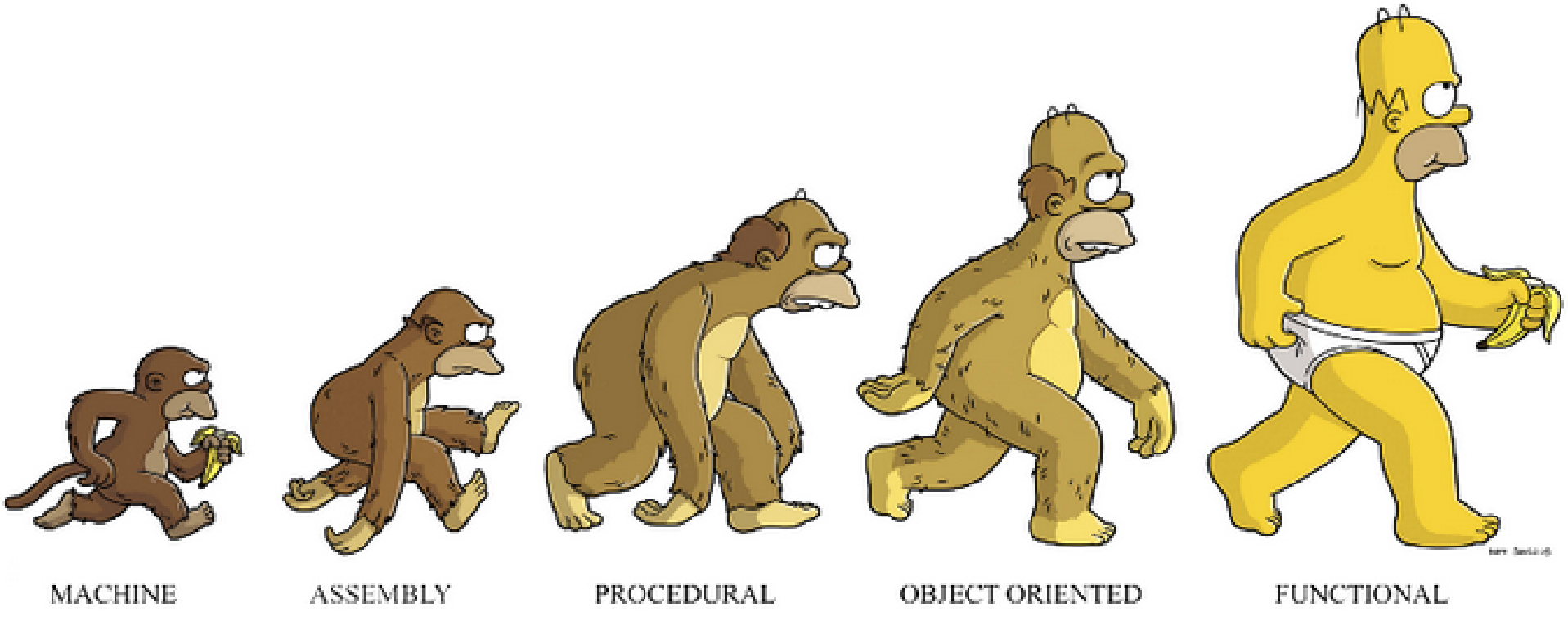
Paradigme de programmation $\approx$ Façon d'aborder un problème de programmation, à l'aide de langages qui supportent bien certains mécanismes d'abstraction et de modularité.

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

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

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

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



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

Impératif		**Déclaratif**	
Procédural	**Objet**	**Fonctionnel**	**Autre**
FORTRAN	Simula	Lisp	**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 mentionnés plus bas 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

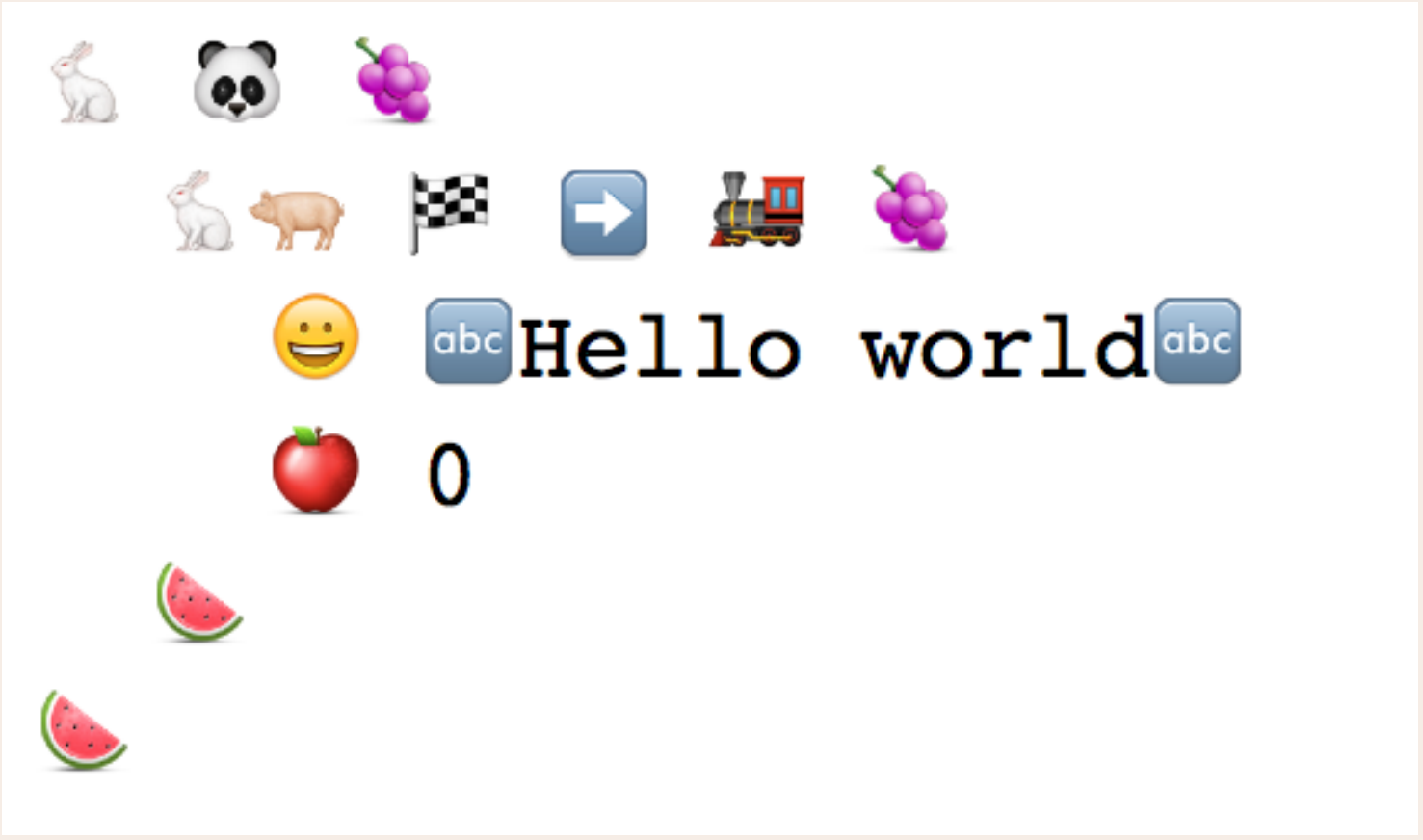
```
PUT THE NUMBER LXXII ONTO THE TOP OF THE PROGRAM STACK
GET THE TOP ELEMENT OF THE STACK AND CONVERT IT TO AN ASCII CHARACTER
AND OUTPUT IT FOR THE CURRENT PERSON USING THIS PROGRAM TO SEE
PUT THE NUMBER CI ONTO THE TOP OF THE PROGRAM STACK
GET THE TOP ELEMENT OF THE STACK AND CONVERT IT TO AN ASCII 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!"
```

3.4 Pourquoi bash et Ruby ?

Pourquoi bash ?

Pourquoi bash ?

Parce que…

- c'est un «*must*» quand on utilise Unix/Linux
- c'est disponible par défaut sur (presque) toutes les machines Unix/Linux
- c'est parfait pour les tâches répétitives qu'on veut traiter de façon «*quick & dirty*»
- c'est, à la base, un *glue language* — l'ancêtre de tous les langages de script

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 : Hunt et Thomas sont les auteurs du célèbre «The Pragmatic Programmer—From Journeyman to Master» (2000).

Pourquoi Ruby ?

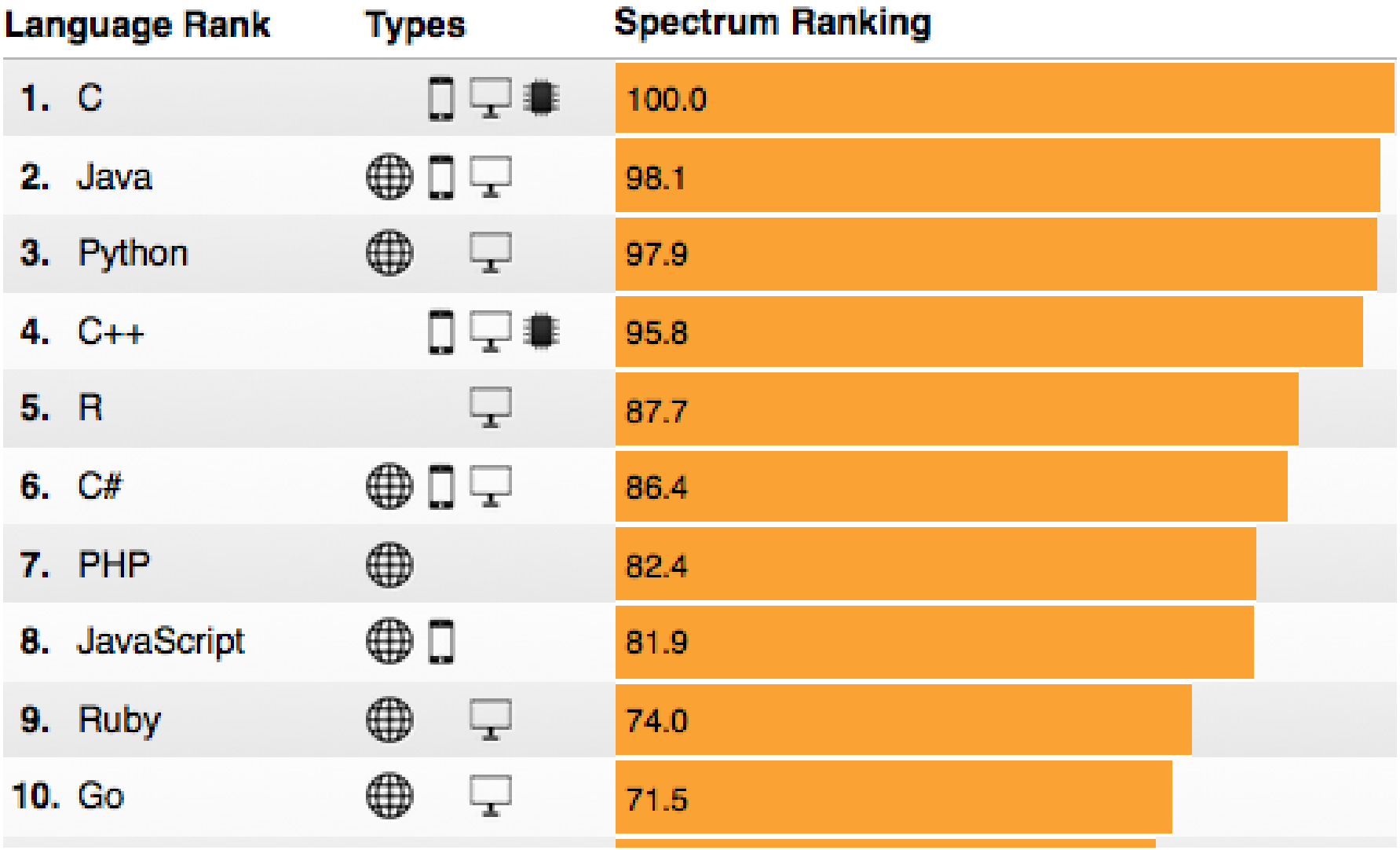
Parce que Ruby est un langage puissant et élégant !

Ruby *is "made for developer happiness" !*

Y. Matsumoto (concepteur du langage Ruby)

Pourquoi Ruby ?

Parce que Ruby est parmi les 10 langages les plus utilisés (IEEE Spectrum, 2016)



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

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

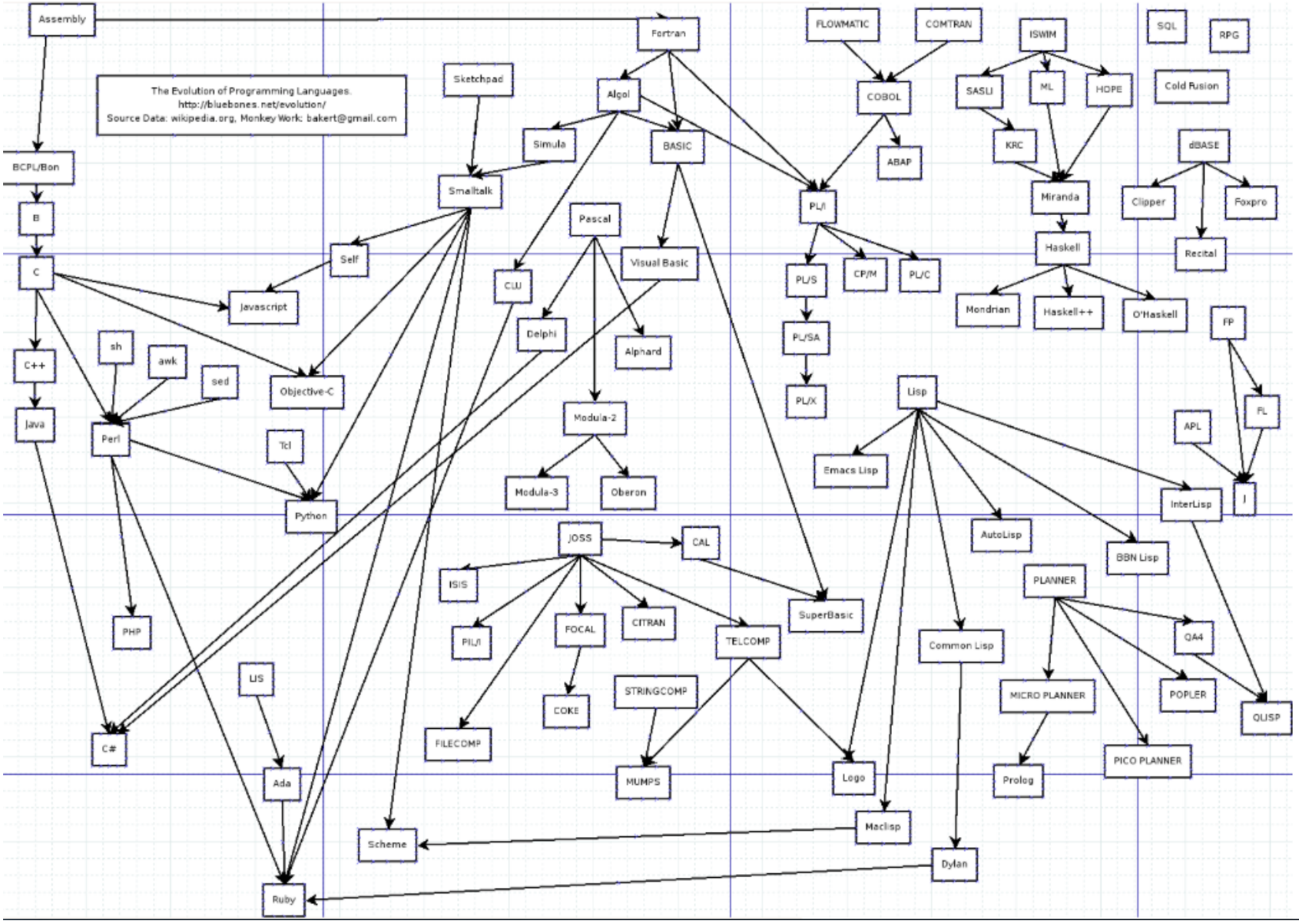
Pourquoi Ruby ?
Parce que... Ruby on Rails

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	

Les principaux paradigmes et langages que vous aurez utilisés. . .

. . . à la fin de votre bac, si vous faites INF600A et INF2005/3005

Impératif		Déclaratif	
Procédural	**Objet**	**Fonctionnel**	**Autre**
Assembleur			**SQL**
			Prolog
	C++		
C	**Java**	**Haskell**	
bash	JavaScript		
	Ruby		

Les principaux paradigmes et langages que vous pourrez comprendre. . .

. . . à la fin de votre bac, si vous faites INF600A et INF2005/3005

Impératif		Déclaratif	
Procédural	**Objet**	**Fonctionnel**	**Autre**
Assembleur			**SQL**
			Prolog
	C++		
C	**Java**	**Haskell**	
bash	JavaScript		
Perl	**Ruby**		
	Python		

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'ai des goûts et des intérêts personnels... qui m'ont conduit à créer ce cours :

- Ruby ☺
- Python ☹

L'outil de correction Oto (noyau seulement, sans 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

No perfect language

Extrait d'une entrevue avec Y. Matsumoto, le créateur de Ruby

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† is probably Python.

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

†. Le créateur de Python

No perfect language

Extrait d'une entrevue avec Y. Matsumoto, le créateur de Ruby

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† is probably Python.

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

†. Le créateur de Python

No perfect language

Extrait d'une entrevue avec Y. Matsumoto, le créateur de Ruby

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[†] *is probably Python.*

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

†. Le créateur de Python

No perfect language

Extrait d'une entrevue avec Y. Matsumoto, le créateur de Ruby

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† is probably Python.

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

†. Le créateur de Python

No perfect language

Extrait d'une entrevue avec Y. Matsumoto, le créateur de Ruby

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.

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

No perfect language

Extrait d'une entrevue avec Y. Matsumoto, le créateur de Ruby

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.

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

4. INF600A : Plan de cours hiver 2017

4.1 Objectifs

Objectifs du cours

Général :
Initier les étudiant-e-s à la **programmation** à l'aide de langages de script et de langages dynamiques.

Objectifs du cours

Spécifiques :
À la fin du cours, l'étudiant-e devrait être capable . . .

- d'expliquer ce qui différencie les langages de script des langages «ordinaires» et d'en comprendre les avantages et limites ;

Objectifs du cours

Spécifiques :
À la fin du cours, l'étudiant-e devrait être capable . . .

- d'expliquer ce qui différencie les langages de script des langages «ordinaires» et d'en comprendre les avantages et limites ;
- **de lire** des scripts et programmes écrits dans divers langages (`bash`, Ruby, Python, Perl)) ;

Objectifs du cours

Spécifiques :
À la fin du cours, l'étudiant-e devrait être capable . . .

- d'expliquer ce qui différencie les langages de script des langages «ordinaires» et d'en comprendre les avantages et limites ;
- **de lire** des scripts et programmes écrits dans divers langages (`bash`, Ruby, Python, Perl)) ;
- **d'écrire** des scripts `bash`, notamment pour automatiser des tâches ;

Objectifs du cours

Spécifiques :
À la fin du cours, l'étudiant-e devrait être capable . . .

- d'expliquer ce qui différencie les langages de script des langages «ordinaires» et d'en comprendre les avantages et limites ;
- **de lire** des scripts et programmes écrits dans divers langages (`bash`, Ruby, Python, Perl)) ;
- **d'écrire** des scripts `bash`, notamment pour automatiser des tâches ;
- **d'écrire** des scripts et programmes objets en Ruby, tout en sachant utiliser le style fonctionnel lorsqu'approprié ;

4.2 Préalables

Préalables officiels vs. effectifs

Préalables «officiels»

- INF3105 Structures de données et algorithmes
- INF3180 Fichiers et bases de données

Préalables officiels vs. effectifs

Préalables «officiels»

- INF3105 Structures de données et algorithmes
- INF3180 Fichiers et bases de données

Préalables effectifs

- Connnaître Java (prog. orientée objets)
- \+ **un autre langage objet** (C++ ?)
- \+ (idéalement) un langage d'un style non impératif (SQL ?) ;

Préalables officiels vs. effectifs

Préalables «officiels»

- INF3105 Structures de données et algorithmes
- INF3180 Fichiers et bases de données

Préalables effectifs

- Connnaître Java (prog. orientée objets)
- \+ **un autre langage objet** (C++ ?)
- \+ (idéalement) un langage d'un style non impératif (SQL ?) ;
- Connaître les bases de Linux en ligne de commandes ;

Préalables officiels vs. effectifs

Préalables «officiels»

- INF3105 Structures de données et algorithmes
- INF3180 Fichiers et bases de données

Préalables effectifs

- Connnaître Java (prog. orientée objets)
- \+ **un autre langage objet** (C++ ?)
- \+ (idéalement) un langage d'un style non impératif (SQL ?) ;
- Connaître les bases de Linux en ligne de commandes ;

- Avoir le goût de découvrir de nouveaux langages ;

Préalables officiels vs. effectifs

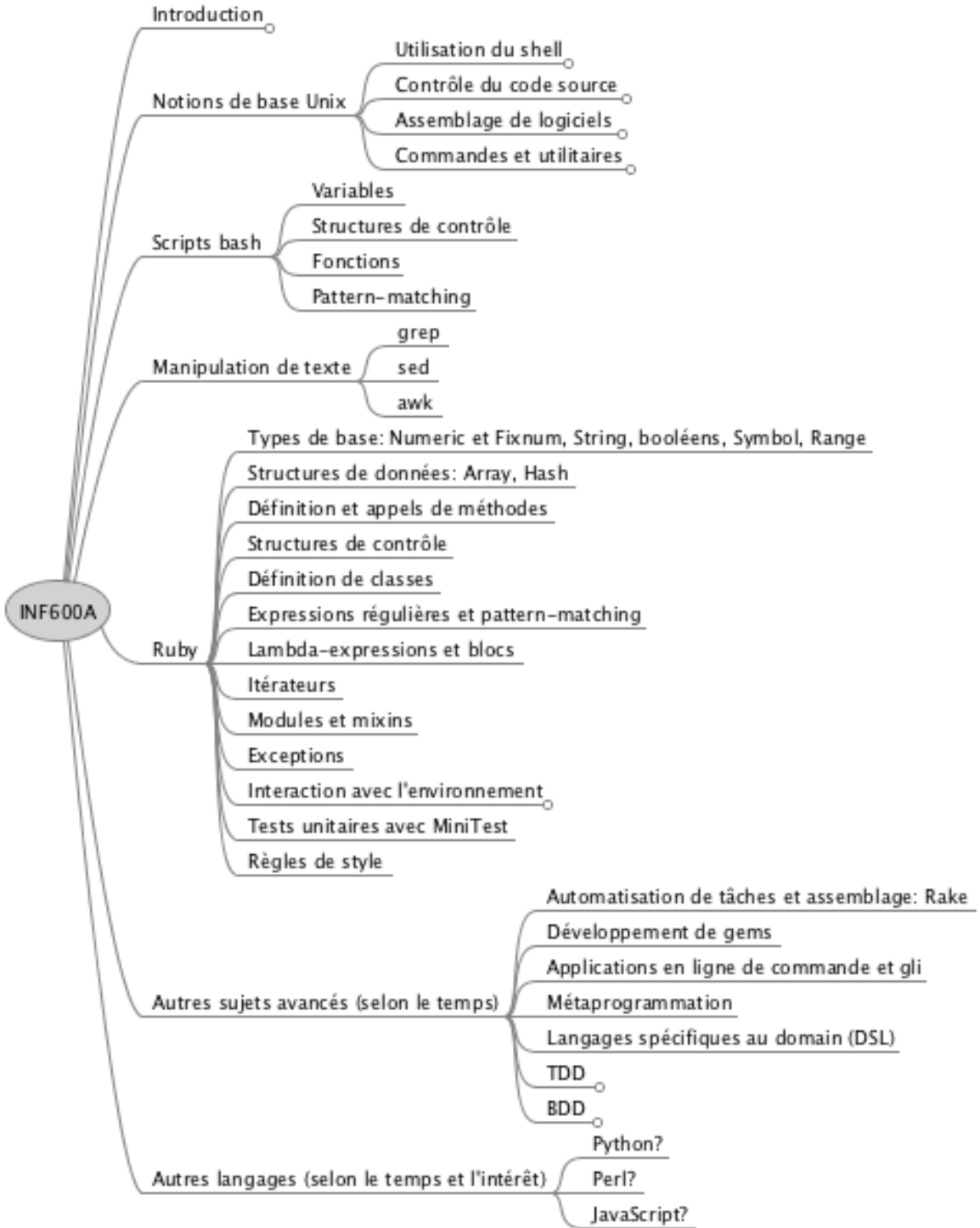
Préalables «officiels»

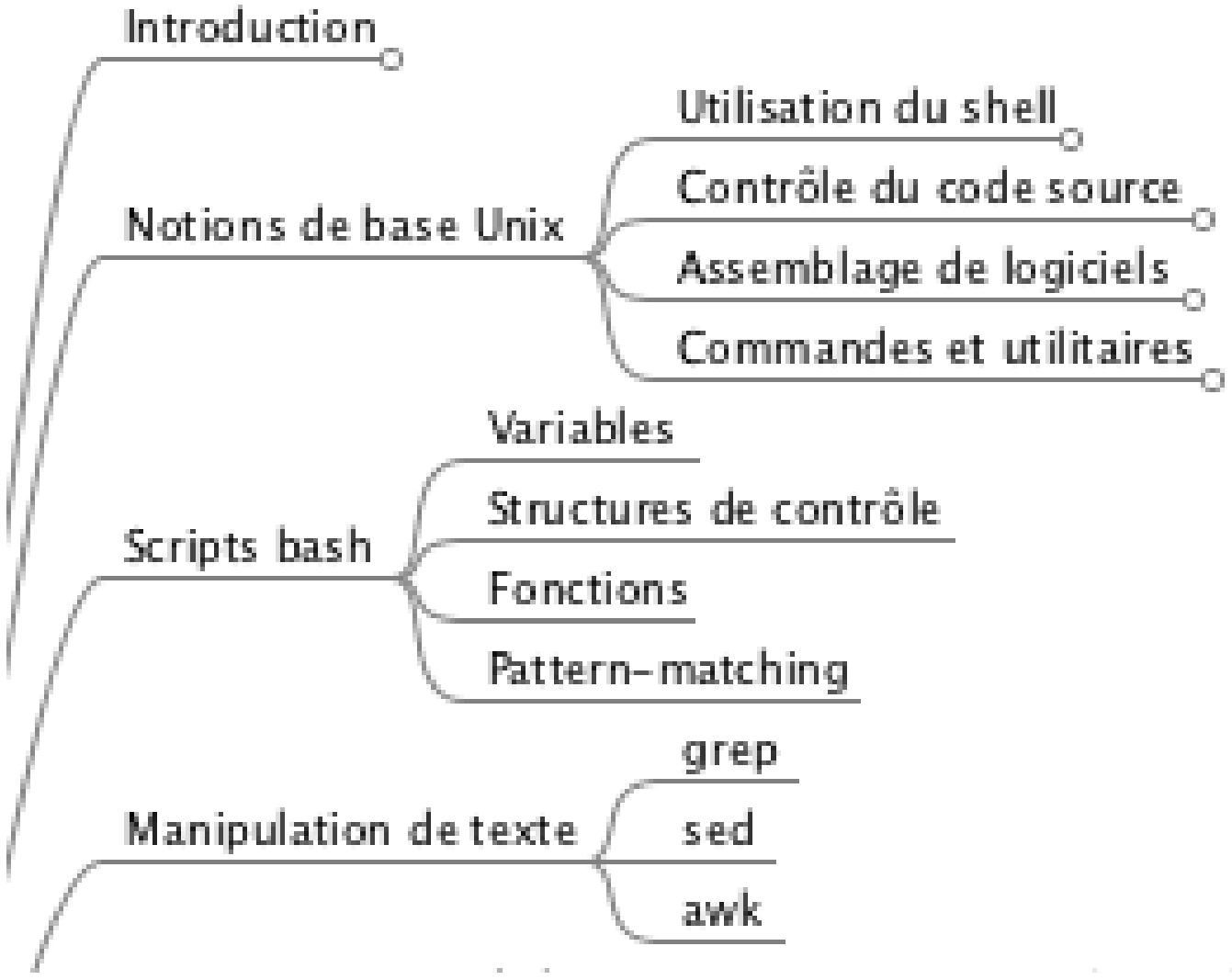
- INF3105 Structures de données et algorithmes
- INF3180 Fichiers et bases de données

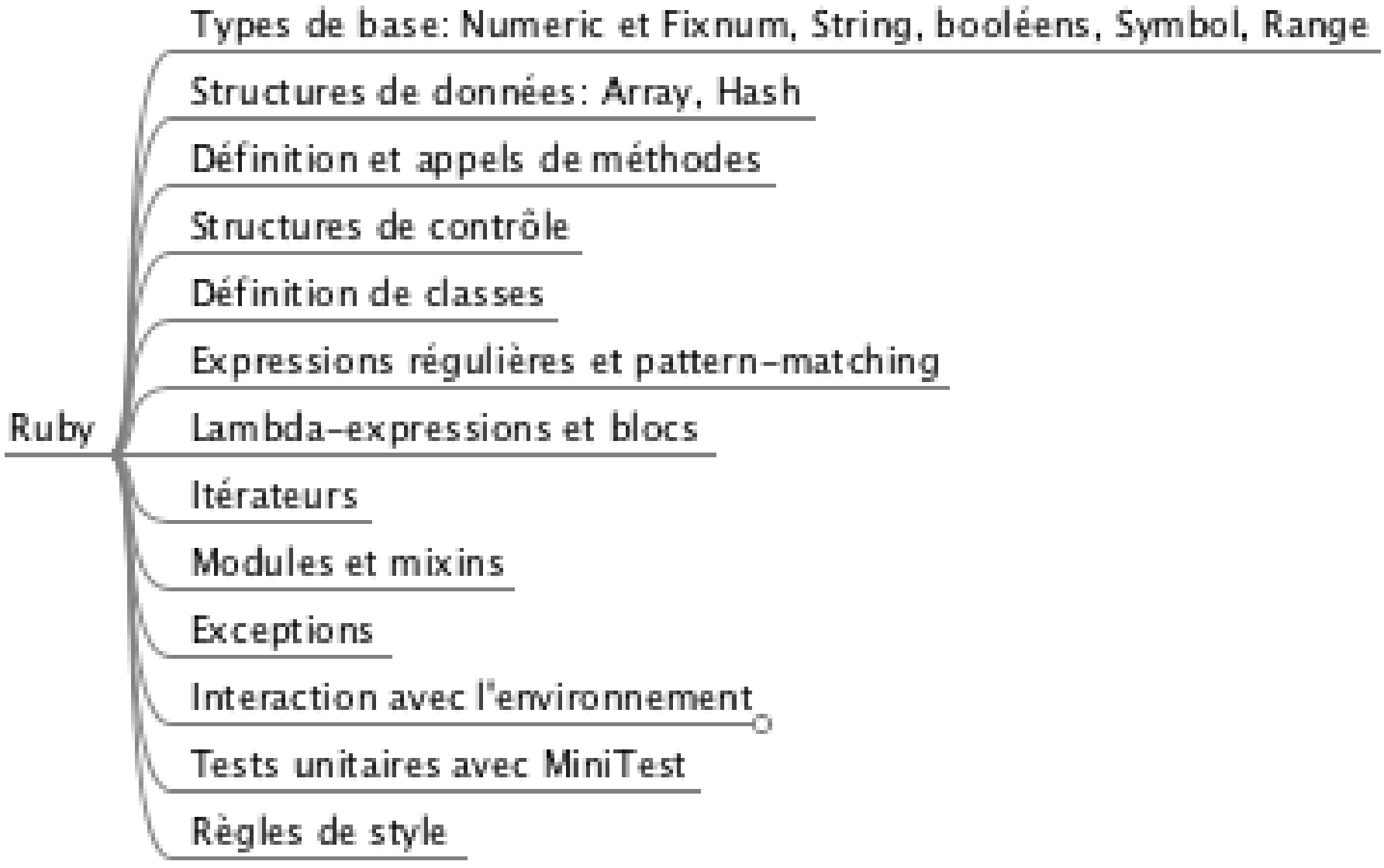
Préalables effectifs

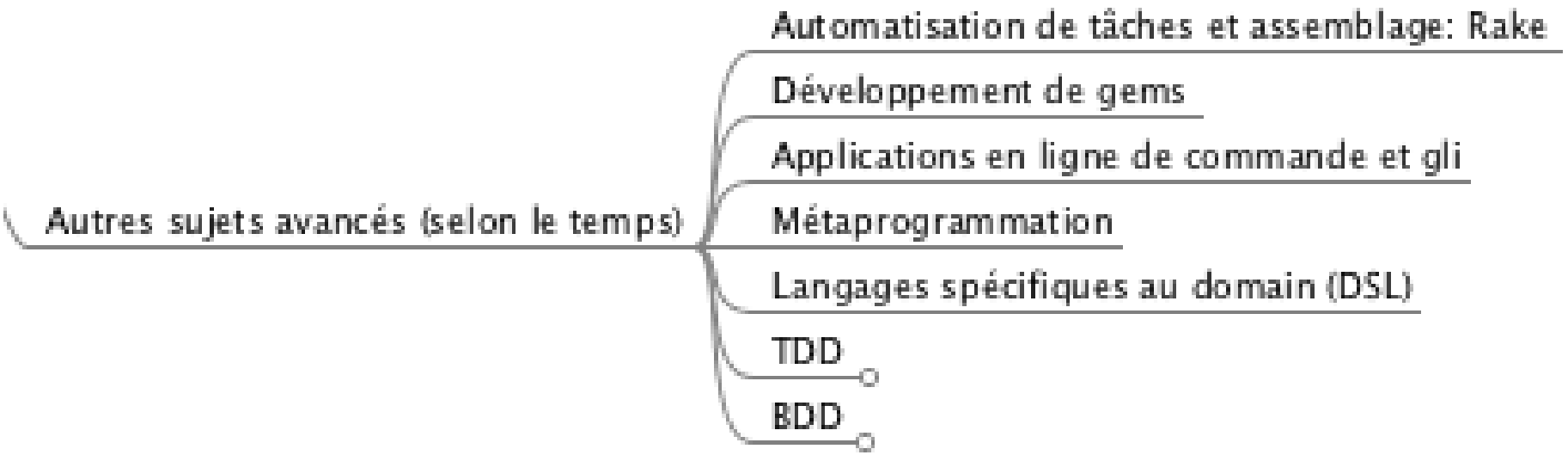
- Connnaître Java (prog. orientée objets)
- \+ **un autre langage objet** (C++ ?)
- \+ (idéalement) un langage d'un style non impératif (SQL ?) ;
- Connaître les bases de Linux en ligne de commandes ;
- Avoir le goût de découvrir de nouveaux langages ;
- Aimer programmer !

4.3 Contenu du cours


INF600A
Introduction
Notions de base Unix
Utilisation du shell
Contrôle du code source
Assemblage de logiciels
Commandes et utilitaires
Scripts bash
Variables
Structures de contrôle
Fonctions
Pattern-matching
Manipulation de texte
grep
sed
awk
Ruby
Types de base: Numeric et Fixnum, String, booléens, Symbol, Range
Structures de données: Array, Hash
Définition et appels de méthodes
Structures de contrôle
Définition de classes
Expressions régulières et pattern-matching
Lambda-expressions et blocs
Itérateurs
Modules et mixins
Exceptions
Interaction avec l'environnement
Tests unitaires avec MiniTest
Règles de style
Autres sujets avancés (selon le temps)
Automatisation de tâches et assemblage: Rake
Développement de gems
Applications en ligne de commande et gli
Métaprogrammation
Langages spécifiques au domain (DSL)
TDD
BDD
Autres langages (selon le temps et l'intérêt)
Python?
Perl?
JavaScript?


Introduction
Notions de base Unix
Utilisation du shell
Contrôle du code source
Assemblage de logiciels
Commandes et utilitaires
Scripts bash
Variables
Structures de contrôle
Fonctions
Pattern-matching
Manipulation de texte
grep
sed
awk


Ruby
Types de base: Numeric et Fixnum, String, booléens, Symbol, Range
Structures de données: Array, Hash
Définition et appels de méthodes
Structures de contrôle
Définition de classes
Expressions régulières et pattern-matching
Lambda-expressions et blocs
Itérateurs
Modules et mixins
Exceptions
Interaction avec l'environnement
Tests unitaires avec MiniTest
Règles de style


Autres sujets avancés (selon le temps)
Automatisation de tâches et assemblage: Rake
Développement de gems
Applications en ligne de commande et gli
Métaprogrammation
Langages spécifiques au domain (DSL)
TDD
BDD


Autres langages (selon le temps et l'intérêt)
Python?
Perl?
JavaScript?

4.4 Évaluation

Évaluation proposée

Examens

- Un examen intra (**22 février**) $\Rightarrow$ 25 %
- Un examen final (**26 avril**) $\Rightarrow$ 30 %

Note : L'intra est à la 7^e semaine, donc avant la semaine de relâche (27/02–3/03) et avant la date limite d'abandon (13/03)

Travaux pratiques

- Trois (3) travaux pratiques $\Rightarrow$ 45 %
- Faits seul ou en équipe de deux (2)

- 3^e travail = Sujet «au choix» (en Ruby ou . . .)

4.5 Matériel pédagogique

Notes de cours et diapositives

- Du matériel sera disponible sur mon site Web :
  `http://www.labunix.uqam.ca/~tremblay/INF600A/Materiel/`

- Toutefois…

Notes de cours et diapositives

- Du matériel sera disponible sur mon site Web :
  `http://www.labunix.uqam.ca/~tremblay/INF600A/Materiel/`

- Toutefois. . . ces notes de cours et diapositives seront disponibles

Notes de cours et diapositives

- Du matériel sera disponible sur mon site Web :
  `http://www.labunix.uqam.ca/~tremblay/INF600A/Materiel/`

- Toutefois. . . ces notes de cours et diapositives seront disponibles de façon incrémentale au fur et à mesure. . . que je les mettrai à jour !

Manuel de référence (suggéré mais non obligatoire)

D. Thomas, A. Hunt, and C. Fowler.
Programming Ruby 1.9 & 2.0 : The Pragmatic Programmer's Guide.
Addison-Wesley, 2013.
(En vente à la COOP : prix membre $\approx$ 62.55 \$ + taxes)

Autres références

Voir plan de cours détaillé sur le web :
`http://syllabus.uqam.ca/files/1482168749_2017_Hiver_INF600A.pdf`

4.6 Laboratoires

Laboratoires

Selon la description officielle du cours

«Ce cours comporte un atelier de deux heures par semaine en laboratoire informatique.»

Mais. . .

Laboratoires

Selon la description officielle du cours

«Ce cours comporte un atelier de deux heures par semaine en laboratoire informatique.»

Mais. . .

- Il n'y aura pas nécessairement un atelier à chaque semaine

Laboratoires

Selon la description officielle du cours

«Ce cours comporte un atelier de deux heures par semaine en laboratoire informatique.»

Mais. . .

- Il n'y aura pas nécessairement un atelier à chaque semaine
- Ce sera la première fois où des laboratoires seront offerts (grève du SÉTUE à l'hiver 2016 ☹)

Questions ?