

INF600A Langages de script et langages dynamiques

Guy Tremblay
Professeur

Département d'informatique
UQAM

http://www.labunix.uqam.ca/~tremblay_gu

INF600A
4 septembre 2018

Pourquoi ce sigle de cours ?

INF600A

1 INF600X Sujets spéciaux en informatique et génie logiciel

2 Description «officielle» du cours

3 De quoi traite ce cours ?

- Qu'est-ce qu'un langage de script ?
- Qu'est-ce qu'un langage dynamique ?
- Pourquoi apprendre de nouveaux langages ?
- Pourquoi bash et Ruby ?

4 Plan de cours automne 2018

- Objectifs
- Préalables
- Contenu du cours
- Évaluation
- Matériel pédagogique
- Laboratoires

1. INF600X Sujets spéciaux en informatique 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.

- Nouveau cours que j'ai proposé et créé (automne 2015)
- 1^{er} cours créé dans la «série» INF600X
- Cours offert pour la 3^e fois :
 - 1 Hiver 2016
 - 2 Hiver 2017
 - 3 Automne 2018

- Cours créé et enseigné par Jean Privat

- Cours offert une fois :
 - 1 Hiver 2018

2. Description «officielle» du cours

Le cours vise à introduire les étudiant-e-s à la programmation à l'aide de **langages de script** et de **langages dynamiques**.

<http://info.uqam.ca/INF600X/>

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.

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'acceptation, 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, métaclasse, méthodes appelées ou créées dynamiquement, appels indirects via des «*hooks*».

3. 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 ?

- sed, awk
- *scripts shell*
- Perl
- Tcl/Tk
- PHP
- Ruby
- Python
- JavaScript
- CoffeeScript
- Groovy
- Julia
- PowerShell
- R
- Io
- Lua
- Elixir
- ...

- sed, awk
- *scripts shell*
- Perl
- Tcl/Tk
- PHP
- Ruby
- Python
- JavaScript
- CoffeeScript
- Groovy
- Julia
- PowerShell
- R
- Io
- Lua
- Elixir
- ...

Le site https://en.wikipedia.org/wiki/Category:Scripting_languages donne une liste de **194** langages !

3.1 Qu'est-ce qu'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>

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

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

Mais...

Ces définitions... ne semblent pas très claires ou utiles



Quelles sont les principales 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

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

*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

- *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*

- *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*

- *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*

Qu'est-ce que ça signifie ?

- *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*

Qu'est-ce que ça signifie ?

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

28

Code source

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

Compilation et exécution

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

Note : Sous réserve que le fichier `hello1.rb` soit **exécutable** !

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!
```

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

*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

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

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

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

```
$ cat fichiers-contenant.rb
#!/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
```

Note : $& \approx \cap$

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

```
$ cat fichiers-contenant.rb
#!/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
```

Note : & \approx \cap

Quelles sont des utilisations typiques des langages de script ?

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

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

Pourquoi utilise-t-on des
langages de script 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

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 ?

*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 Programming Languages*», L.D. Paulson, IEEE Computer, Feb. 2007

Typage dynamique en Ruby : Exemple simple

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

>> x = 2
=> 2

>> x.class
=> Integer

>> x
=> 2
```

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

>> x.class
=> String

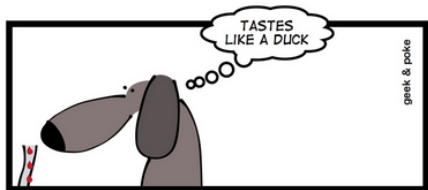
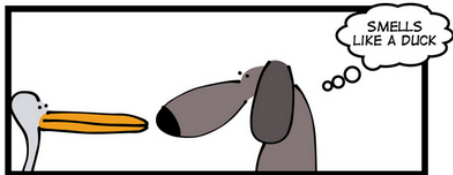
>> x
=> "abc"
```

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

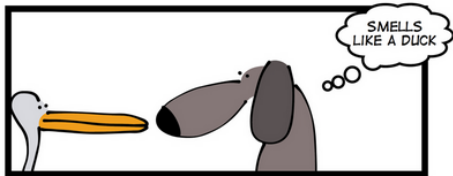
Qu'est-ce que le «*duck typing*» ?

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

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



DUCKFOODING



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
```

Et les tableaux sont hétérogènes !

```
>> a = [10, 20, 30]
=> [10, 20, 30]
>> a.size
=> 3
>> a[5]
=> nil

>> a[5] = "def"
=> ??
>> a.size
=> ??
>> a
=> ??
```

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

50

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

50

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)

51

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 !!

52

Réouverture d'une classe «primitive»

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

```
4.foo
```

```
=> ??
```


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

52

Réouverture d'une classe «primitive»

```
class Integer
  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"
=> ??
```

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 langages 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.

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

«*Ada* replaced hundreds of other programming languages in use by the DoD [Department of Defense] and its counterparts in other nations.

By 1996, the DoD was using 37 languages, compared to 450 when the DoD common programming language effort began.»

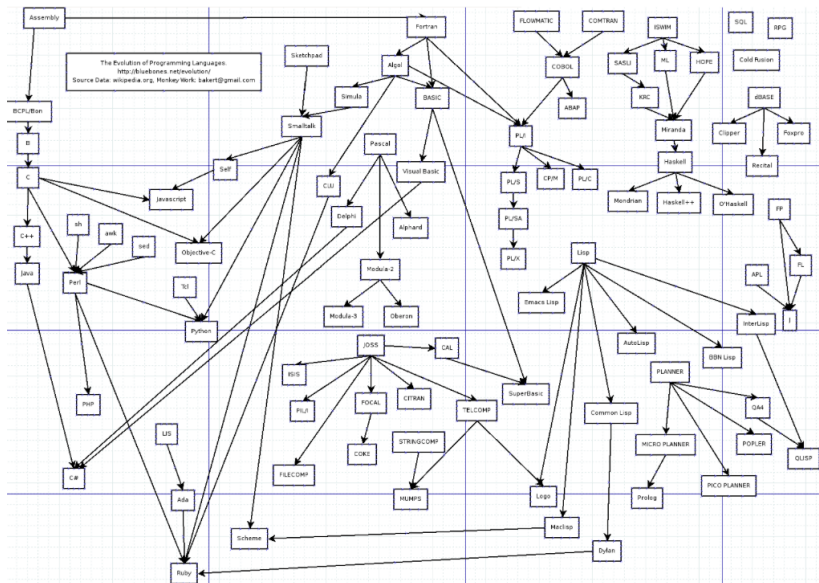
Source: *IEEE Computer*, Septembre 2017

Note : *Ada* est un langage fortement typé et **basé**[†] objets conçu au début des années 80 pour le compte du DoD (*Department of Defense*) des États-Unis.

†. Et non pas «orienté» objets — pas d'héritage.

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 connaît, on voit des clous partout !

58



Quand le marteau est le seul outil qu'on connaît, 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!»*

*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.*

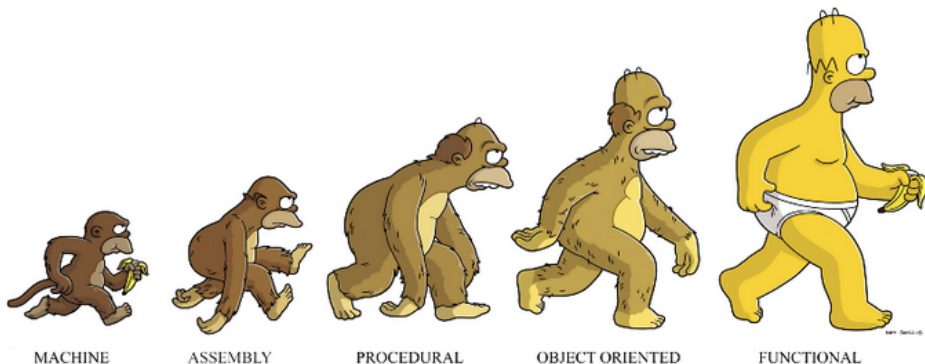
Source: [http://www.cs.uni.edu/~wallingf/blog/archives/monthly/2016-12.html#](http://www.cs.uni.edu/~wallingf/blog/archives/monthly/2016-12.html#e2016-12-16T14_14_26.htm)

[e2016-12-16T14_14_26.htm](http://www.cs.uni.edu/~wallingf/blog/archives/monthly/2016-12.html#e2016-12-16T14_14_26.htm)

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

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	⇒	procédures sous-routines
Paradigme objets	⇒	objets classes
Paradigme fonctionnel	⇒	valeurs fonctions

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!»

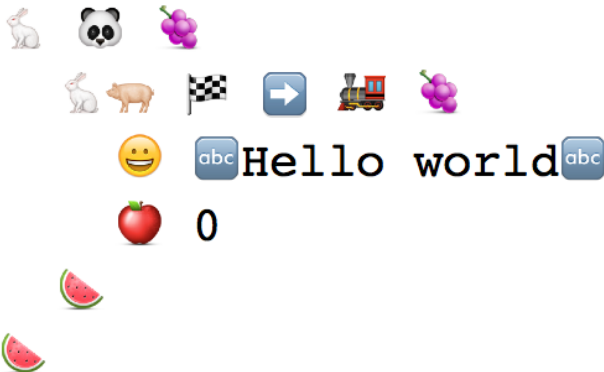
Verbose

```
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
```

Fait : Les langages suivants sont Turing-complets

Diverses versions d'un programme «Hello world!»

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



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](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

72

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 ?

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 livre
«*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 ?

Était parmi les 10 premiers jusqu'en 2016 !

<https://fossbytes.com/most-popular-programming-languages/>

June 2018	Jun 2017	Change	Programming Language	Ratings	Change
1	1		Java	15.37%	0.0088
2	2		C	14.94%	0.0809
3	3		C++	8.34%	0.0261
4	4		Python	5.76%	0.0143
5	5		C#	4.31%	0.0078
6	6		Visual Basic .NET	3.76%	0.0065
7	8	↑	PHP	2.88%	0.0011
8	7	↓	JavaScript	2.50%	-0.53%
9	-	↑	SQL	2.34%	0.0234
10	14	↑	R	1.45%	-0.70%
11	11		Ruby	1.25%	-0.97%
12	18	↑	Objective-C	1.18%	-0.78%
13	16	↑	Visual Basic	1.15%	-0.86%
14	9	↓	Perl	1.15%	-1.16%
15	12	↓	Swift	1.15%	-1.06%

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	Déc. 2015
2.4	Déc. 2016
2.5	Déc. 2017
2.6	Déc. 2018
3.0	2020

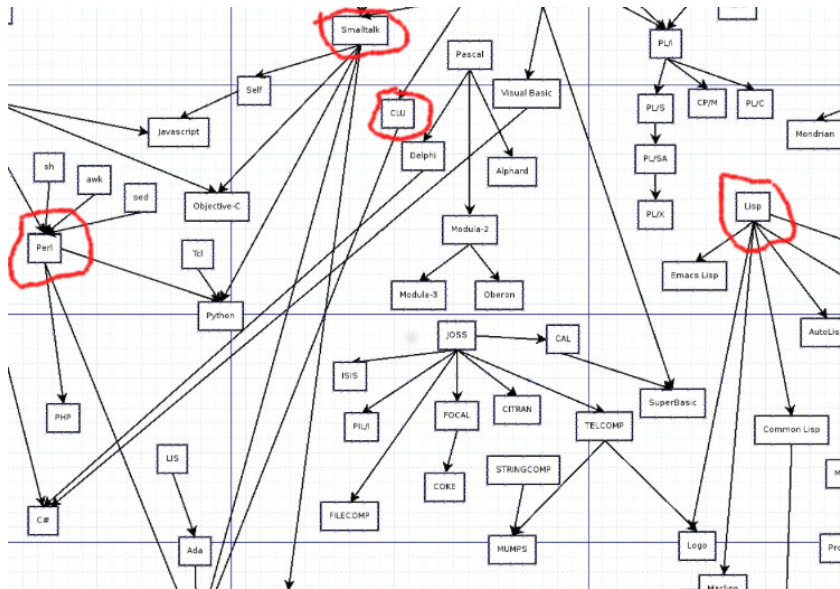
Rails	
Version	Année
1.0	2005
...	...
2.0	2007
...	...
3.0	2010
...	...
4.0	2013
...	...
4.2	Déc. 2014
5.0	Juin 2016
5.1	Mai 2017
5.2	Avril 2018

«*Top 20 Ruby on Rails web applications*»

- Github
- Airbnb
- Basecamp
- Kickstarter
- Slideshare
- Couchsurfing
- ...
- DigitalOcean
- ...

<https://dev.to/iriskatastic/top-20-ruby-on-rails-web-applications>

Généalogie de divers langages de programmation : Les principaux ancêtres de Ruby



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	<i>Uniform Access Principle</i>
Perl	1987	expressions régulières et <i>pattern matching</i>
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			
		Ruby	

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

87

... 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			
Perl		Ruby	
		Python	

Et JavaScript ?

Et JavaScript ?



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

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



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



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

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



- 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 (le «noyau», sans l'application Web)

Avec les tests unitaires mais sans les tests fonctionnels

Language	files	blank	comment	code
Ruby	242	4158	2458	<u>12430</u>
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

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

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. Plan de cours automne 2018

4.1 Objectifs

Général :

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

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 ;

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`) ;

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 ;

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 objet** en Ruby, **tout en sachant utiliser le style fonctionnel lorsqu'approprié** ;

4.2 Préalables

Préalables «officiels»

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

Préalables «officiels»

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

Préalables effectifs

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

Préalables «officiels»

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

Préalables effectifs

- Connaître Java (prog. orientée objet)
- + **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»

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

Préalables effectifs

- Connaître Java (prog. orientée objet)
- + **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»

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

Préalables effectifs

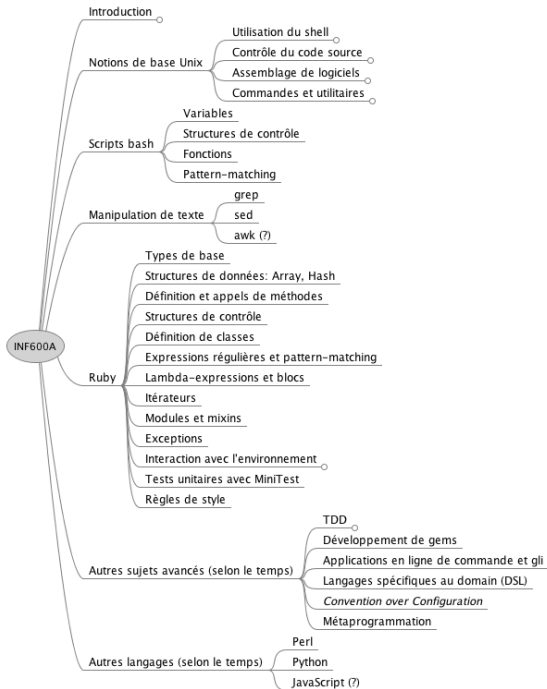
- Connaître Java (prog. orientée objet)
- + **un autre langage objet** (C++ ?)
- + (idéalement) un langage **d'un style non impératif (SQL ?)** ;

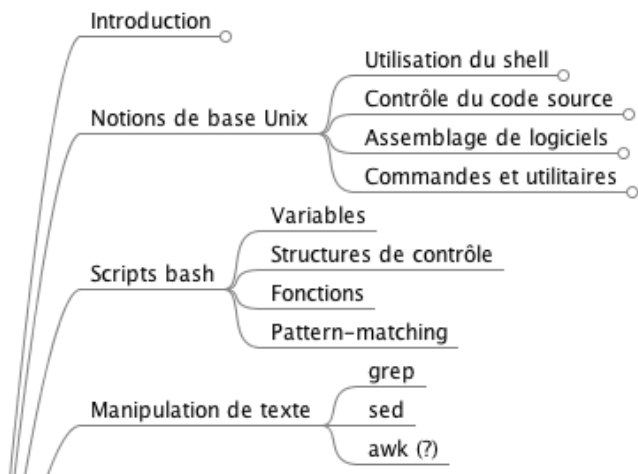
- Connaître les bases de Linux en ligne de commandes ;

- Avoir le goût de découvrir de nouveaux langages ;

- **Aimer programmer !**

4.3 Contenu du cours





INF600A

Ruby

Types de base

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)

TDD

Développement de gems

Applications en ligne de commande et gli

Langages spécifiques au domain (DSL)

Convention over Configuration

Métaprogrammation

Autres langages (selon le temps)

Perl

Python

JavaScript (?)

4.4 Évaluation

Examens

- Un examen intra (**16 octobre**) ⇒ 25 %
- Un examen final (**11 décembre**) ⇒ 30 %

Note : L'intra est à la 7^e semaine, donc **avant** la date limite d'abandon (07/11)

Travaux pratiques

- Trois (3) travaux pratiques ⇒ 45 %
- Faits seul ou en équipe de deux (2) personnes
- 3^e travail = Sujet «au choix»

4.5 Matériel pédagogique

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

- Toutefois...

- Du matériel **sera** disponible sur mon site Web :
`http://www.labunix.uqam.ca/~tremblay_gu/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 !



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)

Voir plan de cours détaillé sur le web :

http://syllabus.uqam.ca/files/1535053858_2018_Automne_INF600A.pdf

4.6 Laboratoires

Selon la description officielle du cours

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

Mais...

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
- Il n'y a pas de labo cette semaine (6 septembre)

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
- Il n'y a pas de labo cette semaine (6 septembre)
- Le-la démonstrateur-trice n'a pas encore été trouvé-e 😞

Questions ?