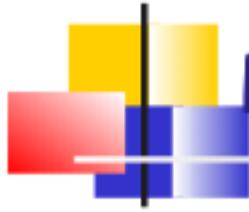


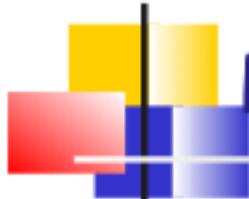
Ruby, Panorama

Pierre Jacoboni
LIUM - Université du Maine
2006



Plan

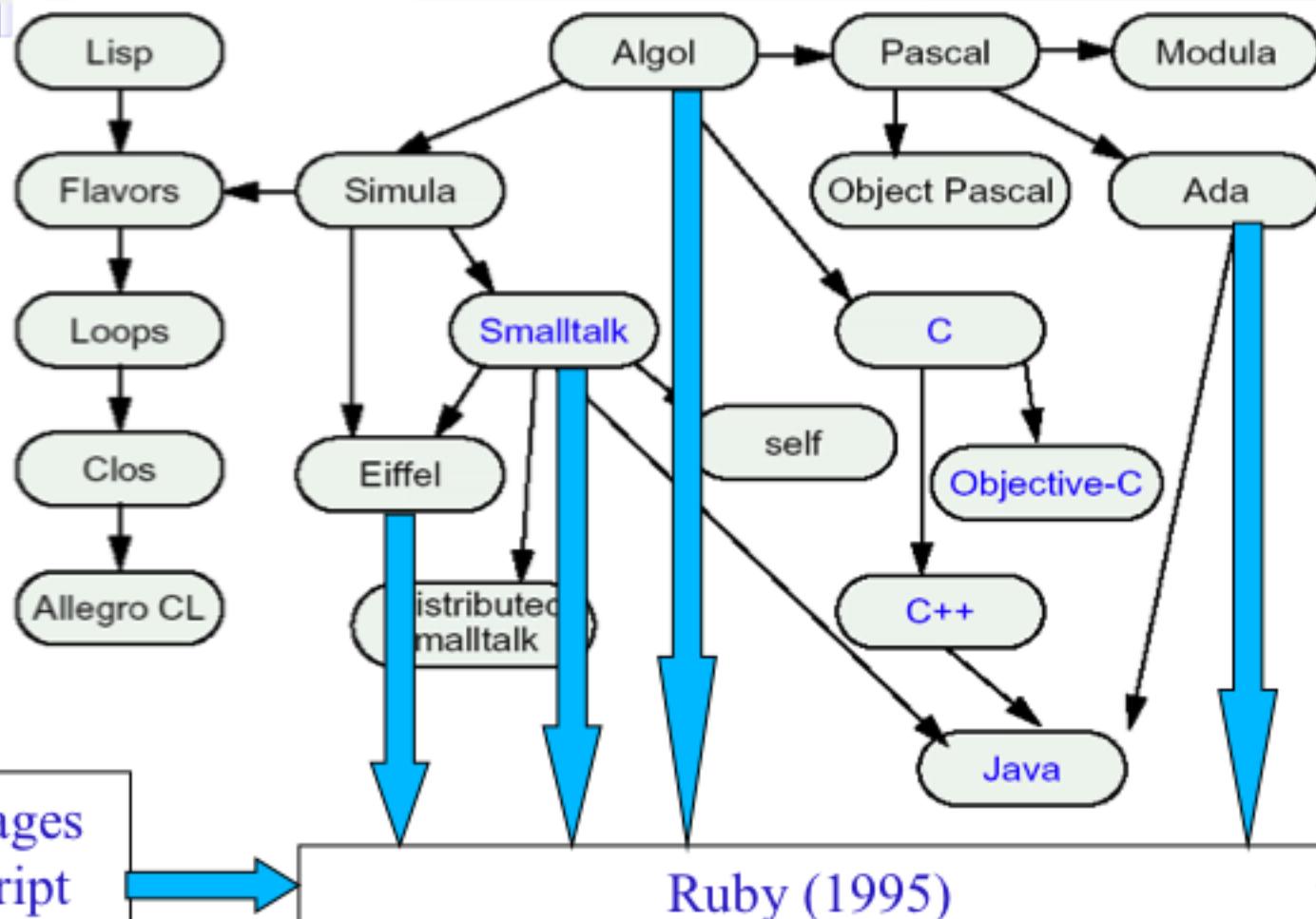
- A - Introduction à Ruby
 - 1. Pourquoi Ruby
 - 2. Premiers éléments de syntaxe
 - 1- les principes
 - 2- classes/instances
 - 3- transmission de messages
 - 4- variables et affectation
 - 5- instanciation
 - 3. Utiliser Ruby
 - Ligne de commande
 - Interactive Ruby Shell
 - Utilisation d'un IDE



Plan

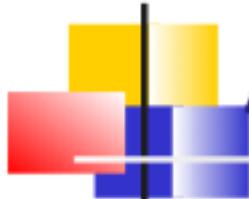
- B - Les bases du langages
 - 1-Les chaînes de Caractères
 - 2-Les nombres
 - 3-Les Conditionnelles
 - 4-Les Itérations

A-Historique des langages



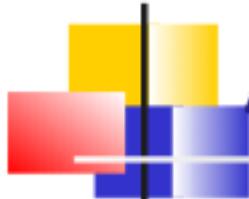
4
Langages
de Script

Perl



A-Les Langages orientés objets

- *Simula*, créé en 1967, est le premier langage orienté objet.
- *Smalltalk* ensuite, le nouveau langage majeur orienté objet.
- *C++* développé par Bjarne Stroustrup chez AT&T dans les années 1980.
- *Objective C* développé par Stepstone a connu son heure de gloire avec les stations NeXT.
- En 1995, Sun a proposé un nouveau langage Java en liaison avec l'Internet.



A-Les Langages purs

Attributs complexes
Métaclasses

Kee
Art
Kool
Smeci
Crl

Représentation des
connaissances

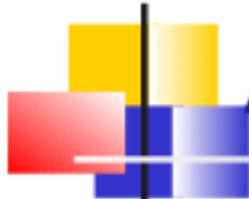
Tout objet
Liens Dynamiques
Métaclasses

Attributs simples
Métaclasses

Smalltalk
Clos
Ruby

Domaines de prédilection

Prototypage
Génération d'application
Simulation



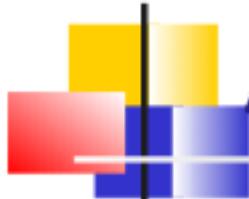
A-Les Langages Hybrides

Type objet et type simple
Pas de métaclasser utilisateur

C++
Eiffel
Simula
Objective C
Java

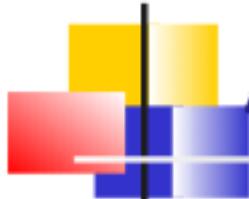
Domaines de préférence

Génie logiciel industriel
Applications à fort taux de maintenabilité
Client/Serveur



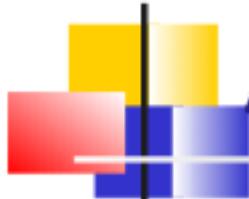
A-Langages Hybrides

- Langage Simula
 - langage compilé ancien dont la syntaxe est hérité d'Algol.
- Objective C
 - Ce langage est une extension de C, fortement inspiré de Smalltalk 80.
- Langage Eiffel
 - Conçu pour les applications scientifiques et la gestion, ce langage compilé intègre, comme Smalltalk et C++, les exceptions.



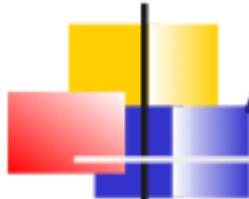
A-Les Langages Hybrides

- Langage C++
 - C++ est un sur-ensemble de C. Il rajoute un ensemble de fonctionnalités nouvelles pour C.
 - Voir cours M2.
- Langage Java
 - Voir Inf 320B



A1-Pourquoi Ruby

- D'après Ruby, un langage script orienté objet., Armin Roehrl, Stefan Schmiedl, Clemens Wyss, Stephen Jarvis, Valerie Chavez et Olivier Allain
- Ruby, un langage script orienté objet.
 - Yukihiro "Matz" Matsumoto est l'inventeur de Ruby. Un grand merci à lui!
 - Certains transparents ont été repris de l'exposé réalisé par Matz (JAOO 2001).

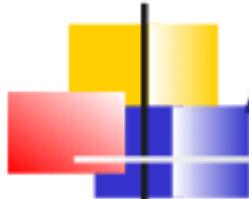


A1-Pourquoi Ruby

- Ruby = Smalltalk - Syntaxe inhabituelle
 - + script perl
 - + gestion des erreurs de Python
 - + itérateurs utilisés dans le langage CLU.
 - + bien d'autres outils intéressants.
- Ruby > (Smalltalk + Perl)/2.

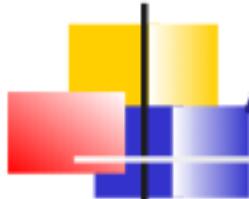
Les langages “jouets” ne sont pas bons. Ils peuvent résoudre des petits problèmes, mais les langages réels résolvent les mêmes problèmes d'une meilleure façon.

matz



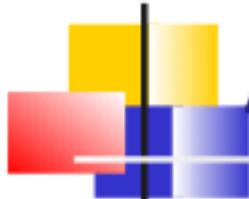
A1-Pourquoi Ruby

- il est productif
 - Les programmes Ruby sont courts et lisibles.
- il est sans surprise
 - Vous pouvez vous concentrer sur la programmation.
- il est libre
 - Gratuit, mais aussi libre d'être modifier/copier/utiliser.
 - Possibilité d'apprendre plus en lisant ses sources.



A1-Pourquoi Ruby

- il est suffisamment rapide
 - Rapide pour un interpréteur,
 - Remplacer les parties lentes par des extensions en C. (90/10)
- il est portable et multi-plateforme
 - Linux, Windows, DOS, ...
 - Le même fichier source s'exécute sur n'importe quelle plateforme
- il est facile à apprendre
 - L'autre alternative ce serait Smalltalk !



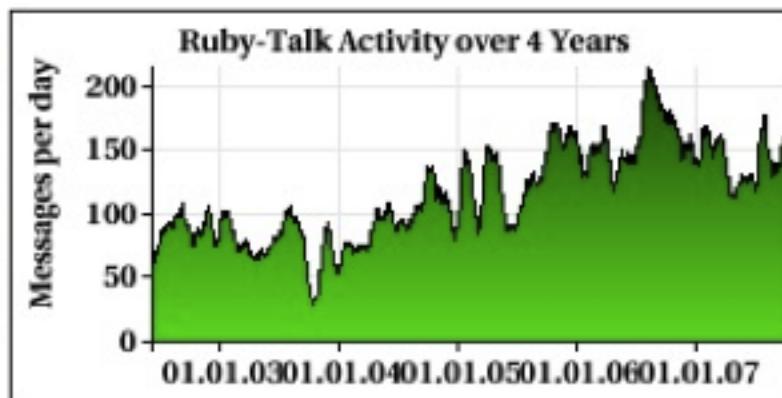
A1-Ressources en ligne.

- Communauté: Mailingliste & Newsgroup
 - comp.lang.ruby
- Sites web :
 - <http://www.ruby-doc.org/>
 - <http://ruby.on-page.net/> ✓
 - <http://www.ruby-lang.org/fr/>
 - <http://tryruby.org/levels/1/challenges/0> ✓
- « Programming Ruby , The Pragmatic Programmer's Guide », Addison Wesley, 2000

Croissance de Ruby

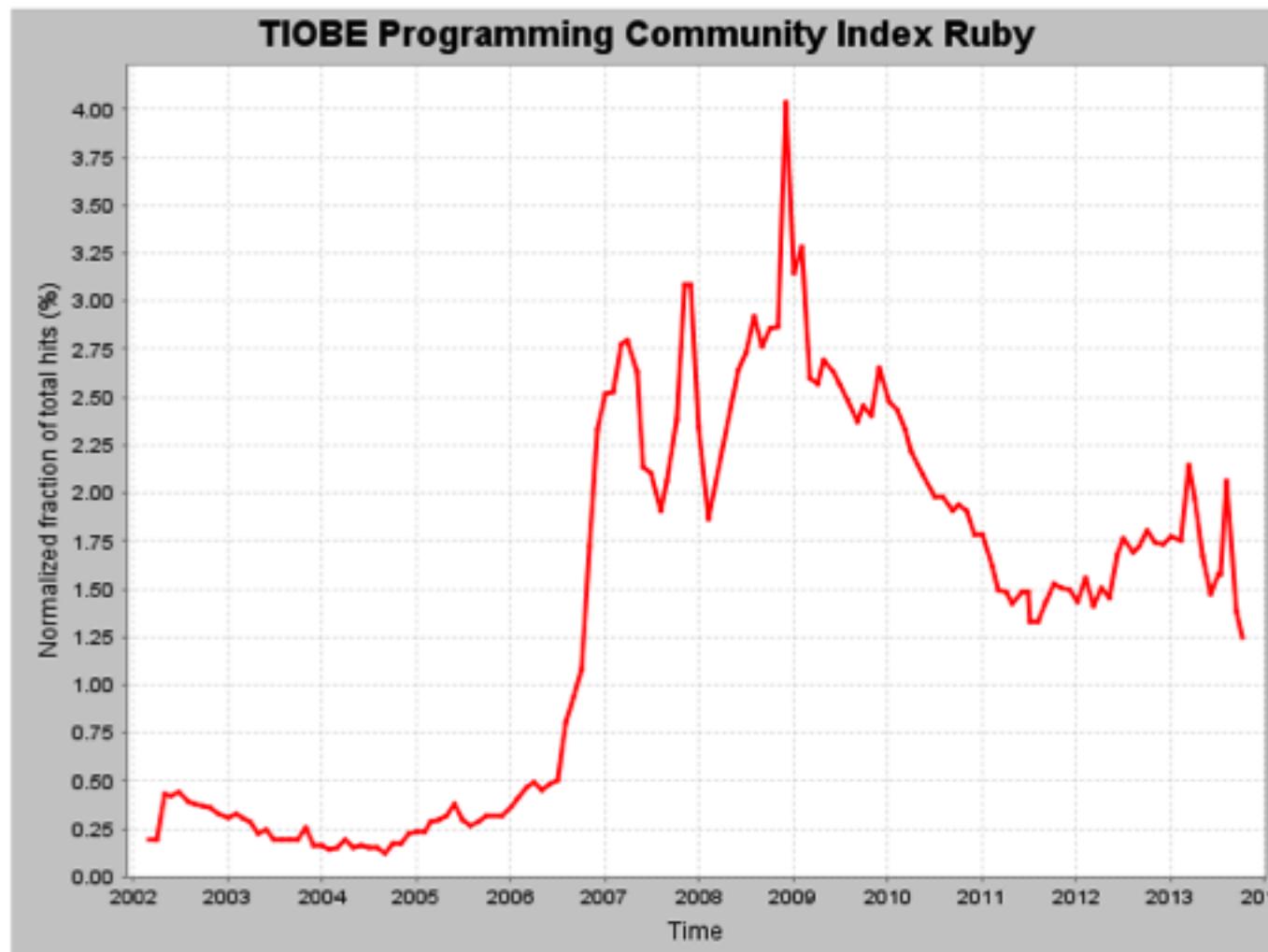
Depuis sa publication en 1995, Ruby a attiré des programmeurs passionnés des quatre coins du monde. En 2006, Ruby rassemblait une masse critique d'utilisateur et gagnait une réelle reconnaissance. Des groupes d'utilisateurs existent dans les plus grandes villes du monde, et les conférences à propos de Ruby affichent complet.

Ruby-Talk, la toute première [liste de diffusion](#) recevant les discussions à propos du langage Ruby, atteint aujourd'hui une moyenne de deux cent nouveaux messages par jour.



L'index TIOBE, qui mesure la croissance des langages informatiques, place Ruby à la dixième place du classement des langages les plus utilisés au monde. Concernant cette évolution, leur prédition est la suivante : « il y a des chances que Ruby entre dans le top 10 dans moins de six mois. » La plus grande part de cette accélération semble revenir à la popularité de certains logiciels écrits en Ruby, notamment le framework web Ruby on Rails².

- Highest Rating (since 2002): 4.034% (8th position, December 2008)
- Lowest Rating (since 2002): 0.124% (27th position, August 2004)
- Paradigms: Object-Oriented
- Type system: Dynamically typed



TIOBE Programming Community Index for November 2013



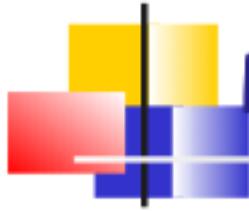
November Headline: Programming languages of Microsoft are gaining popularity

It is interesting to see that 3 out of 4 languages in the top 20 that are gaining popularity are languages defined by Microsoft. These are C#, SQL Server language Transact-SQL and Visual Basic.NET. It might be a coincidence but this happened in the same month in which Windows Mobile gained market share in comparison to the Android and iOS mobile phone operating systems. The other language that is doing well is JavaScript. That might come as no surprise. JavaScript is the lingua franca of websites nowadays.

The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. The popular search engines Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings. Observe that the TIOBE index is not about the best programming language or the language in which most lines of code have been written.

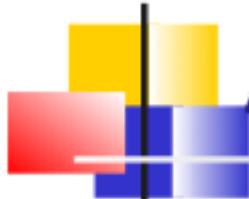
The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system. The definition of the TIOBE index can be found [here](#).

Position Nov 2013	Position Nov 2012	Delta in Position	Programming Language	Ratings Nov 2013	Delta Nov 2012	Status
1	1	=	C	18.155%	-1.07%	A
2	2	=	Java	16.521%	-0.93%	A
3	3	=	Objective-C	9.406%	-0.98%	A
4	4	=	C++	8.369%	-1.33%	A
5	6	↑	C#	6.024%	+0.43%	A
6	5	↓	PHP	5.379%	-0.36%	A
7	7	=	(Visual) Basic	4.396%	-0.64%	A
8	8	=	Python	3.110%	-0.95%	A
9	23	↑↑↑↑↑	Transact-SQL	2.521%	+2.05%	A
10	11	↑	JavaScript	2.050%	+0.77%	A
11	15	↑↑↑	Visual Basic .NET	1.969%	+1.20%	A
12	9	↓↓↓	Perl	1.521%	-0.86%	A
13	10	↓↓↓	Ruby	1.303%	-0.44%	A
14	14	=	Pascal	0.715%	-0.17%	A
15	13	↓↓	Lisp	0.706%	-0.25%	A
16	19	↑↑↑	MATLAB	0.656%	+0.04%	B
17	12	↓↓↓↓	Delphi/Object Pascal	0.649%	-0.36%	A-
18	17	↓	PL/SQL	0.605%	-0.03%	A-
19	24	↑↑↑↑	COBOL	0.585%	+0.11%	B
20	20	=	Assembly	0.532%	-0.05%	B



Plan

- A - Introduction à Ruby
 - 1. Pourquoi Ruby
 - 2. Premiers éléments de syntaxe
 - 1- les principes
 - 2- classes/instances
 - 3- transmission de messages
 - 4- variables et affectation
 - 5- instanciation
 - 3. Utiliser Ruby
 - Ligne de commande
 - Interactive Ruby Shell
 - Utilisation d'un IDE



A2-Premiers éléments de Syntaxe

- **Les Commentaires**

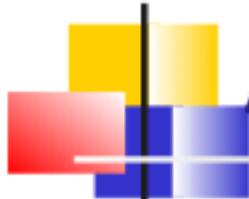
- Les fichiers Sources Ruby doivent commencer par des commentaires qui listent les noms de classes, les copyrights, les informations sur l'auteur,

- En utilisant #

```
# StringReplace
# $Id: stringreplace.rb, v 1.0 10/15/01 20:05:17$
# Copyright © 2001, Jason Wong
# You can redistribute and/or modify it under ...
```

- En Utilisant un Bloc (=begin, =end)

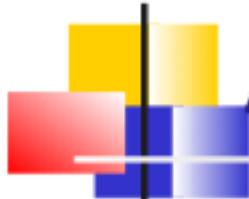
```
=begin
    Block of comments here
=end
```



A2-Premiers éléments de Syntaxe

- la factorielle d'un nombre

```
# Sauver ce source en fact.rb
def fact(n)
    if n == 0
        1
    else
        n * fact(n-1)
    end
end
print fact(ARGV[0].to_i), "\n"
```



A2-Premiers éléments de Syntaxe

```
% ruby fact.rb 1
```

```
1
```

```
% ruby fact.rb 5
```

```
120
```

- Est-ce que ça marche si on passe 40 en argument ? Cela devrait donner un overflow...

```
% ruby fact.rb 40
```

```
??????
```



A2

9

1

9

120

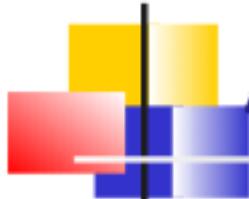
E

a

9

$$40! = 815915283247897734345611269596115894272000000000$$

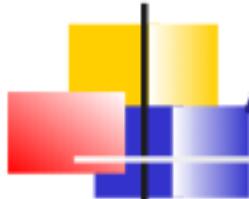
en
erflow...



A21-Principes

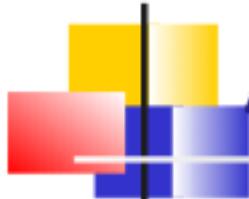
- Caractéristiques de Ruby / POO

- Toutes les entités manipulées sont des objets
- Tout objet est instance d'une classe
- Toute les classes sont des objets
- Un objet est accessible uniquement par envoi de message
- Toutes les procédures sont des méthodes.



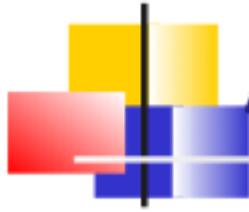
A22-Classes/Instances

- Tout objet est instance d'une classe
 - la classe spécifie la structure de l'objet et son comportement
 - le nom des variables d'état (les champs)
 - Ruby/Smalltalk : variables d'instances
 - le texte des procédures définissant le comportement
 - Ruby/ Smalltalk : méthodes
 - l'objet est responsable des valeurs des variables d'état (les attributs)
 - de l'extérieur on n'accède à l'état de l'objet que par un envoi de message
 - ie que si l'objet autorise un accès



A22-Interface d'un objet

- L'ensemble des messages qu'il sait traiter
 - on ne peut communiquer avec un objet que par son interface (les services qu'il offre)
- Encapsulation des données en Ruby
 - l'interface d'un objet est donnée par les méthodes de sa classe
 - les VI ne peuvent être manipulées de l'extérieur que par le biais de messages adressés à l'objet
 - les données propres à un objet ne sont manipulées que par l'objet lui-même
 - les opérations sur un objet ne dépendent pas des choix de réalisation mais de l'interface de l'objet



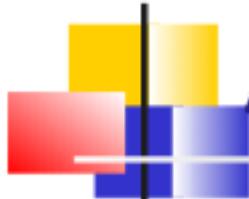
A22-Exemple

- Définition d'une classe.

```
class Humain

    def identifie
        print "Je suis une personne.\n"
    end

    def tarif_train(age)
        if age < 12
            print "Tarif réduit.\n";
        else
            print "Tarif normal.\n";
        end
    end
end
```



A23-Transmission de message

- L'opération de base en Ruby est la transmission de message à un objet (appelé le receveur du message)
 - un message est formé d'un sélecteur et d'éventuels arguments
 - un envoi de message
 - retourne un résultat (un objet)
 - caractère fonctionnel
 - de plus il peut opérer des effets de bord sur l'environnement
 - aussi un caractère procédural

A23-Interprétation des messages

objet abstrait
(classe)

Classe Compte

structure

numéro
solde
titulaire

comportement
initialize
donneTonSolde
dépose
retire
afficheToi

dépose : 2000

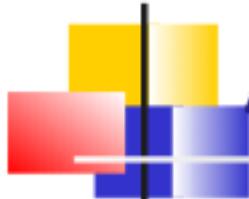
objets concrets
(instance)

instance de .
numéro 8765
solde 5000
titulaire Després

unCompte

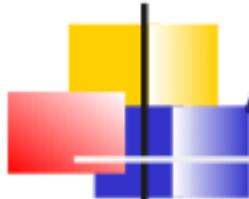
instance de .
numéro 2345
solde 3 milliards
titulaire Jacoboni

unAutreCompte



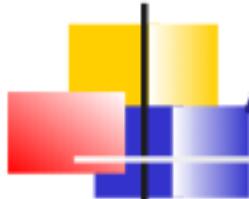
A23-Interprétation des messages

- Un message est envoyé à un objet
 - le message fournit le nom de la procédure (méthode) à exécuter
 - le corps de la méthode est recherché dans le catalogue des méthodes de la classe
 - le corps est exécuté dans l'environnement de l'objet qui a reçu le message, ie chaque nom de variable d'instance dans le corps de la méthode est interprété comme désignant le champ de l'objet qui a reçu le message
- Le résultat est retourné à l'expéditeur du message



A23-Les messages : classification

- d'après le nombre d'arguments
 - message unaire : pas d'argument
 - message binaire : un argument
 - message n-aires : plusieurs arguments
- syntaxe : `objetReceveur.sélecteur`
 - si l'objet receveur est une instance
le msg est associé à une méthode d'instance
 - si l'objet receveur est une classe
le msg est associé à une méthode de classe



A23-Les messages unaires

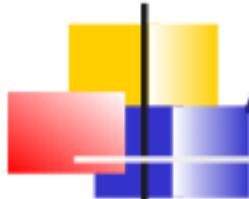
- Pas d'argument

- messages unaires associés à des méthodes d'instance

- "hEllO".downcase → "hello"
 - 1256.class → Fixnum alors que
 - "1256".class → String
 - "1234".reverse → "4321" alors que
 - 1234.reverse → GROSSE ERROR (quoique)

- messages unaires associés à des méthodes de classe

- Time.now → Tue Nov 11 22:36:57 Paris, Madrid 2003
 - String.new → ""



A23-Les messages binaires

- Un et un seul argument

- Utilisation

- arithmétique

- $21 + 3 \rightarrow 24$
 - $61 / 20 \rightarrow 3$

- comparaison

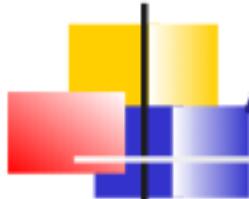
- "b" > "a" → true
 - 'abc' == :abc → false

- concaténation

- 'Une' + ' table' → "Une table"

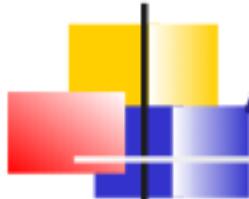
Sucre Syntaxique

```
irb(main):029:0> 21.+(3)
=> 24
irb(main):030:0> 'une'.+(' table')
=> "une table"
irb(main):031:0> █
```



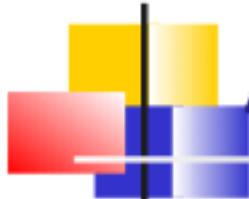
A23-Les messages naires

- Un ou plusieurs arguments
 - le sélecteur est suivi de parenthèses donnant la liste des paramètres
- Convention :
 - une méthode de modification d'une variable d'instance porte le nom de cette variable d'instance
 - Suivre cette convention
 - simplifie le codage (utilisation du Coding Assistant)
 - augmente
 - la lisibilité (par la standardisation)
 - la confusion pour les tout débutants



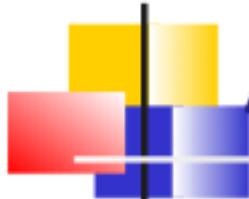
A24-Les Variables dans Ruby

- Les variables **Locales** sont les variables définies par exemple dans les méthodes.
 - Elles ne sont pas accessibles en dehors de la méthode où elles sont définies.
- Les variables **d'Instances** sont accessibles dans toutes les méthodes pour un objet particulier.
 - Leur nom est précédé de @. Elles sont uniques pour chaque objet et inaccessibles entre les objets.



A24-Les Variables dans Ruby

- Les variables de *Classes* sont des variables partagées par toutes les instances de la classe.
 - On peut les utiliser même sans création d'instance.
 - Leur nom est préfixé par @@.
 - Les variables de classes ne peuvent pas être partagées entre les classes.
- Au contraire des trois premières catégories, les variables *Globales* ne sont pas définie dans une classe.
 - Elles sont accessibles dans toutes les classes et leur nom est préfixé par \$



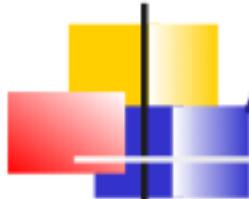
A24-Les Variables Locales

- Son nom commence par une minuscule ou un blanc souligné (_). Les variables locales ne sont pas initialisées.

```
ruby> foo
```

```
ERR: (eval):1: undefined local variable or method `foo' for
main(Object)
```

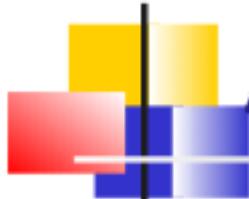
- La première affectation que l'on fait à une variable locale agit un peu comme une déclaration.
 - Si vous vous référez à une variable locale non initialisée, l'interpréteur ruby croit à une tentative d'invoquer un méthode de ce nom, d'où le message d'erreur de l'exemple ci-dessus.
- En général, la portée d'une variable locale est dans :
 - `proc{ ... }`
 - `loop{ ... }`
 - `def ... end`
 - `class ... end`
 - `module ... end`
 - ...le programme entier si aucun des cas précédents ne s'est appliqué.



A24-Les Variables d'instances

- Les variables d'instance n'ont pas à être déclarées, en ruby.
 - Ceci donne une structure d'objets très souple.
 - En fait, chaque variable d'instance est dynamiquement ajoutée à l'objet au moment de sa première invocation.

```
class InstTest
    def set_foo(n)
        @foo = n          # Une VI @foo
    end
    def set_bar(n)
        @bar = n          # Une autre VI @bar
    end
end
```

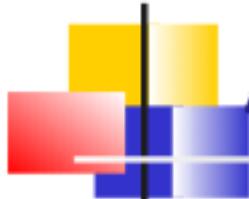


A24-Les Variables d'instances

- Il est possible de "déclarer" en début de classe les variables d'instances.
 - Gain en lisibilité, donc obligation de le faire dans le cadre de ce cours

```
class Compte
    @numero      # Le Numéro du compte
    @titulaire   # Le nom du Titulaire
    @solde       # Le solde du compte
....
```

```
end
```

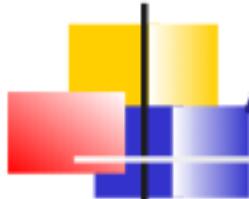


A24-Les Variables d'instances

- Il est possible d'utiliser le « **coding assistant** » pour générer les méthodes d'accès en lecture écriture.
- Exemple dans la classe Compte

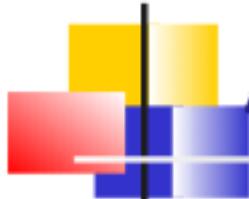
```
# Génération des méthodes numero et numero=
  attr :numero, true
# Génération des méthodes titulaire et titulaire=
  attr :titulaire, true
# Génération de la méthodes solde (false => uniquement en lecture)
  attr :solde, false
```

.....



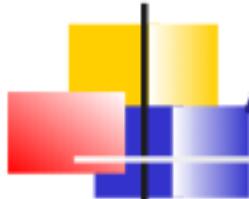
A24-Accès aux Variables d'instances

- Autres formes du « Cod. Ass »
 - **attr attribut [,writable]**
 - **attr_reader attribut [,attribut]**
 - **attr_writer attribut [,attribut]**
 - **attr_accessor attribut [,attribut]**
 - Équivalent à utiliser **attr attribut, true** sur chacun des attributs de la liste.
- En pratique préférez l'utilisation de
 - **attr_reader attribut** pour *getter*
 - **attr_accessor attribut** pour *getter et setter*



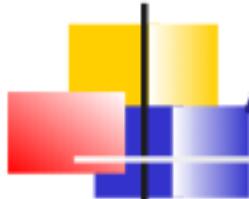
A24-Variables Globales prédéfinies

- Il existe un certain nombre de variables globales prédéfinies:
 - **\$!** Dernier message d'erreur
 - **\$@** Lieu de l'erreur
 - **\$_** Dernière chaîne lue par `gets`
 - **\$..** Numéro de la dernière ligne lue par l'interpréteur
 - **\$&** Dernière chaîne trouvée par `regexp`
 - **\$~** Dernière trouvaille de `regexp`, en tableau
 - **\$n** La *n*ième trouvaille du dernier `regexp` (équivaut à `$~[n]`)
 - **\$=** Flag de sensibilité à la casse (minuscules/majuscules) des recherches par `regexp`
 - **\$/** Séparateur des articles en entrée
 - **\$** Séparateur des articles en sortie
 - **\$0** Le nom du script ruby
 - **\$*** Les arguments de la ligne de commande
 - **\$\$** L'identifiant du process de l'interpréteur
 - **\$?** Code retour du dernier processus fils



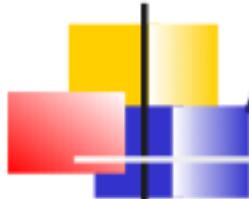
A24-Variables et affectations

- On peut affecter un objet comme valeur d'une variable
 - le symbole de l'affectation est =
 - l'affectation n'est pas un envoi de message
 - l'affectation associe à un nom de variable une valeur (un objet)
 - les variables en Ruby ne sont pas typées
 - n'importe quel objet peut être affecté à n'importe quelle variable
 - convention syntaxique :
 - les variables globales commencent par une majuscule
 - les variables locales commencent par une minuscule



A25-Instanciation

- Crédit d'un objet à partir de sa classe en 2 temps
 - 1) allocation par le système d'une zone mémoire dont la taille est connue
 - langages non typés : nombres d'attributs + 1 pointeur (sur la classe)
 - cette allocation se fait en Ruby en envoyant un message à la classe de l'objet que l'on veut créer
 - en exécutant "une méthode de classe"
 - 2) initialisation des variables d'instances
 - par des messages adressés à l'objet
 - les méthodes d'initialisation doivent être programmées

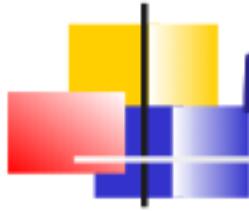


A24-Variables Classe/instance

- Ex : un JukeBox stats sur les chansons

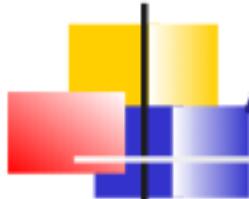
```
class Chanson
    @@lecture=0          # Ca c'est une variable de classe
    @nom ; @artiste ; @duree ; @lecture # Les variables d'instances
    def initialize(nom,artiste,duree)
        @nom,@artiste,@duree=nom,artiste,duree
        @lecture=0
    end
    def play()
        @lecture+=1
        @@lecture+=1
        puts "Cette chanson #@nom lecture : #@lecture fois,
              Total des lectures : #@@lecture."
    end
end
```

Exécution
s1=Chanson.new("Le Diner","Bénabar",180)
s2=Chanson.new("Le Fou Rire","Bénabar",216)
s1.play() ; s2.play()
s1.play() ; s1.play()
s2.play()



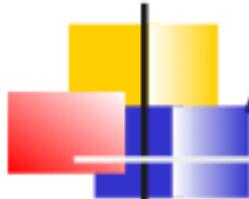
Plan

- A - Introduction à Ruby
 - 1. Pourquoi Ruby
 - 2. Premiers éléments de syntaxe
 - 1- les principes
 - 2- classes/instances
 - 3- transmission de messages
 - 4- variables et affectation
 - 5- instanciation
 - 3. Utiliser Ruby
 - Ligne de commande
 - Interactive Ruby Shell
 - Utilisation d'un IDE



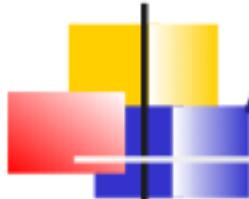
A3-Utiliser Ruby

- Ruby est disponible gratuitement sur de nombreuses plates-formes, MacOS X, Unix, Windows, Linux
- Il existe au moins trois façons d'utiliser Ruby sur un système.
 - Ligne de commande
 - Interactive Ruby Shell (IRB)
 - Environnement de Développement Intégré (IDE)
 - Freeride, Netbeans, Eclipse, etc ...

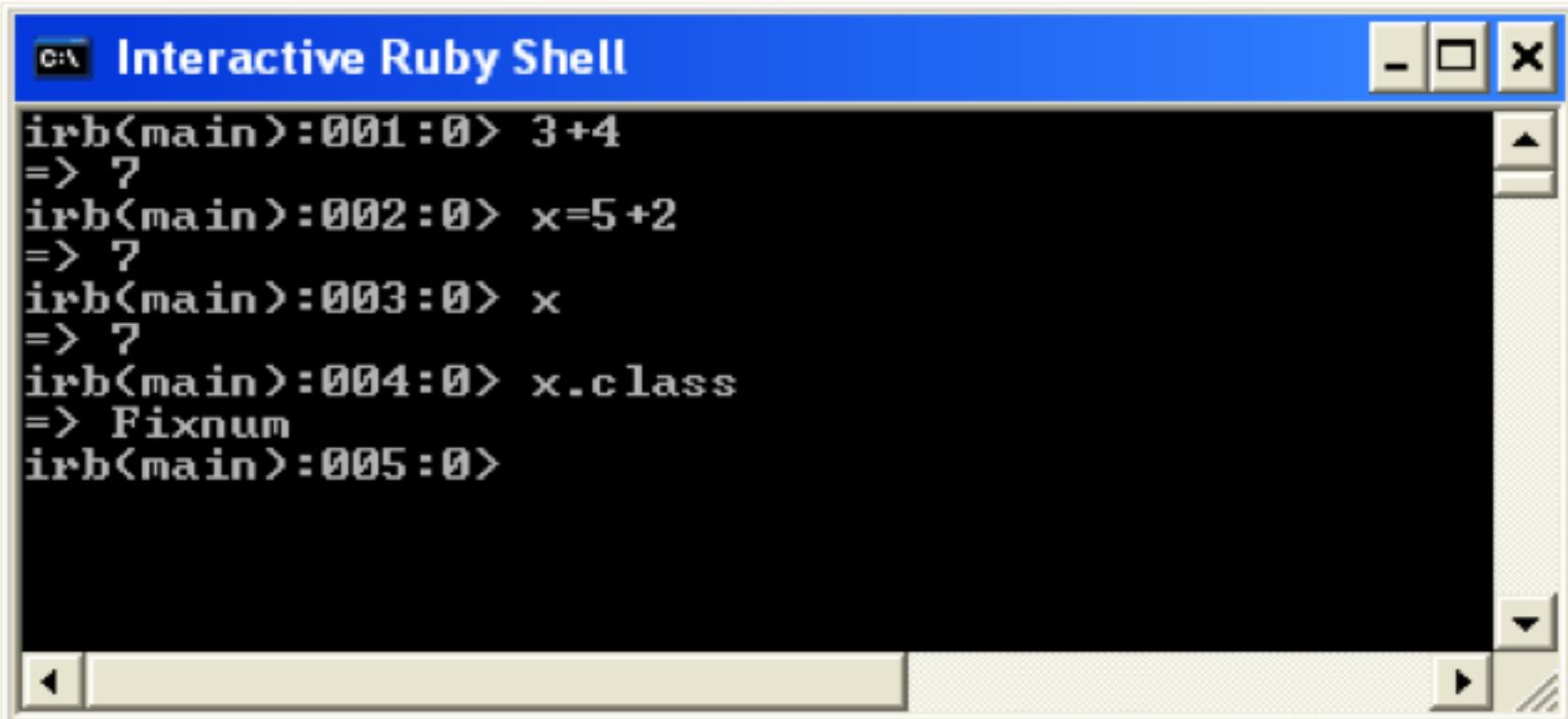


A31-Ruby sur la ligne de commande

- Directement depuis la ligne de commande,
 - % ruby -e 'print "hello world\n"'
 - hello world
- Plus classiquement, un programme ruby peut être stocké dans un fichier.
 - % cat > test.rb
 - print "hello world\n"
 - ^D
 - % ruby test.rb
 - hello world



A32-Interactive Ruby Shell (IRB)



The screenshot shows a window titled "Interactive Ruby Shell". The window contains the following Ruby session:

```
irb(main):001:0> 3+4
=> 7
irb(main):002:0> x=5+2
=> 7
irb(main):003:0> x
=> 7
irb(main):004:0> x.class
=> Fixnum
irb(main):005:0>
```

A33-Un IDE (FreeRide)

The screenshot shows the FreeRIDE IDE interface. The title bar reads "FreeRIDE - The Free Ruby IDE". The menu bar includes File, Edit, View, Run, Tools, Help. The toolbar contains icons for New, Open, Save, Run, Stop, and others. The Source View window on the left shows a file tree with a selected file: "C:/Program Files/FreeRIDE/freeride/examples/Helloworld/hello.rb". The code editor window on the right displays the contents of this file:

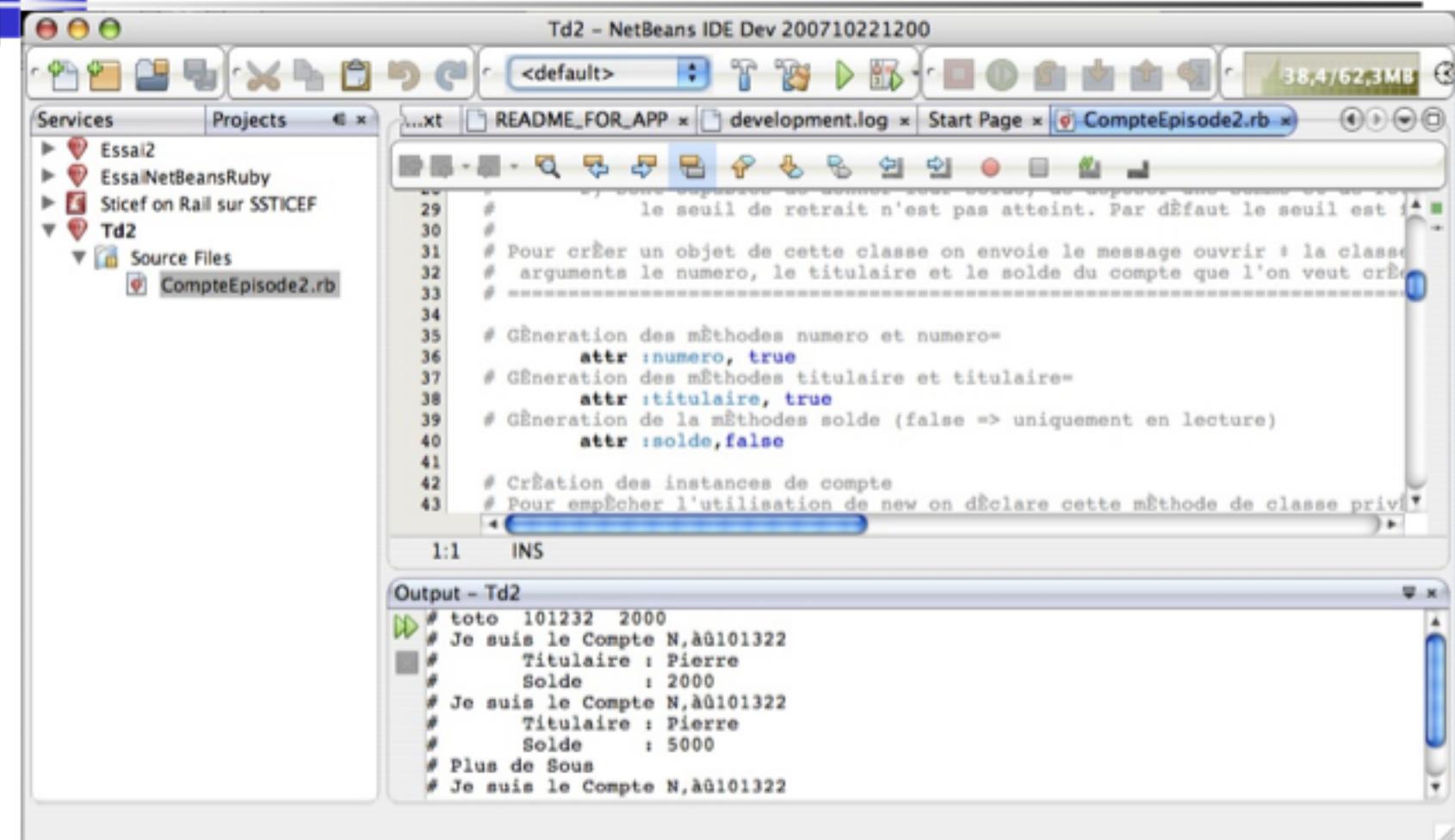
```
19 # This module creates a namespace called MyFreeRIDEPlugin in which
20 # you can place any number of related FreeRIDE plugins without having to
21 # worry about name conflicts with other FreeRIDE plugins.
22 module MyFreeRIDEPlugin
23
24   # This module creates a namespace called HelloWorld. The purpose is
25   # to prevent name conflicts with other plugins that I might also put
26   # into MyFreeRIDEPlugin.
27 module HelloWorld
28
29   # This class defines the hello world plugin
30
31   class MyHelloWorld
32     extend FreeBASE::StandardPlugin
33
34     # initialize this plugin
35     def MyHelloWorld.start(plugin)
36       # Create an instance of the our command object
37       the_cmd_object = HelloCommand.new()
38
39       # Add the command to the system
40       # at the database address: "/system/ui/commands/Examples/Hello"
41       # and give it the menu text "Hello World"
42       plugin["/system/ui/commands"].manager.add('Examples/Hello', 'Hello')
43
44       # This code is executed whenever our command is invoked
45       # this could obviously have been defined above
46   end
47 end
```

The Output View window at the bottom shows the results of running the script:

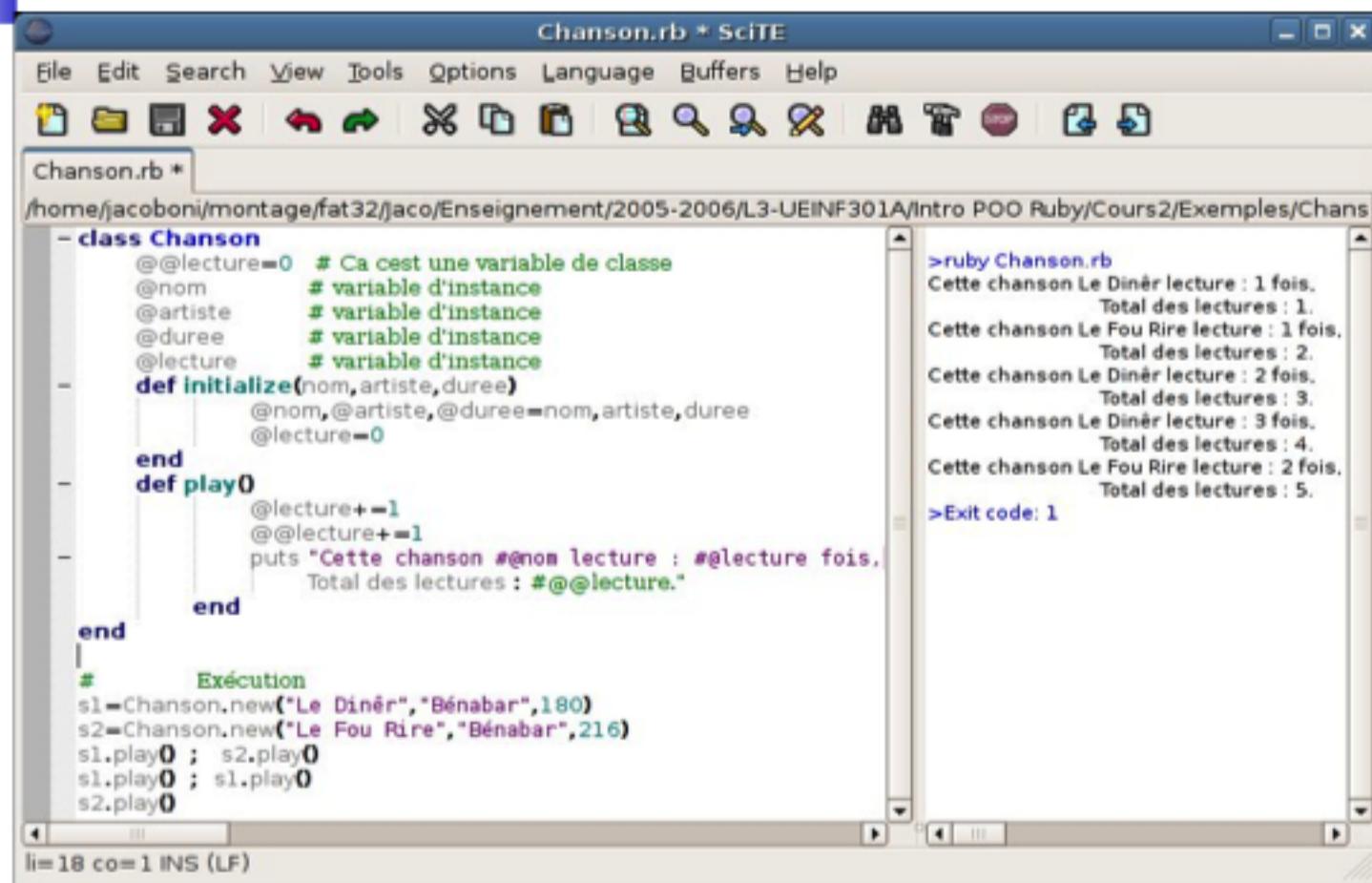
```
ruby c:/Documents and Settings/jacoboni/Mes documents/Enseignement/2003-2004/L2/Intro POO Ruby/Cours2/TD2/voyelleEnMajuscule.r
Test de la version Destructive
Avant :
vive Ruby
abcdefghijklmnopqrstuvwxyz

Le résultat de l'appel de voyelleEnMajuscule
vive RÙBÝ
AbcdEfghIjklnOpqrstUvwxyz
```

A33-Un IDE (NetBeans)



A33-Un Autre IDE (SciTe)



The screenshot shows the SciTE IDE interface with the file "Chanson.rb" open. The code defines a class "Chanson" with methods for initializing a song and playing it. The "play" method increments a class variable "@lecture" and prints a message indicating the number of times the song has been played. The code also includes a section for execution and creates two instances of the class.

```
Chanson.rb * SciTE
File Edit Search View Tools Options Language Buffers Help
Chanson.rb *
/home/jacoboni/montage/fat32/jaco/Enseignement/2005-2006/L3-UEINF301A/Intro POO Ruby/Cours2/Exemples/Chans
- class Chanson
  @@lecture=0 # Ca cest une variable de classe
  @nom
  @artiste
  @duree
  @lecture
- def initialize(nom,artiste,duree)
  @nom,@artiste,@duree=nom,artiste,duree
  @lecture=0
end
- def play()
  @lecture+=1
  @@lecture+=1
  puts "Cette chanson #{@nom} lecture : #{@lecture} fois.
  Total des lectures : @@lecture."
end
#
# Exécution
s1=Chanson.new("Le Dinér","Bénabar",180)
s2=Chanson.new("Le Fou Rire","Bénabar",216)
s1.play(); s2.play()
s1.play(); s1.play()
s2.play()

>ruby Chanson.rb
Cette chanson Le Dinér lecture : 1 fois.
Total des lectures : 1.
Cette chanson Le Fou Rire lecture : 1 fois.
Total des lectures : 2.
Cette chanson Le Dinér lecture : 2 fois.
Total des lectures : 3.
Cette chanson Le Dinér lecture : 3 fois.
Total des lectures : 4.
Cette chanson Le Fou Rire lecture : 2 fois.
Total des lectures : 5.
>Exit code: 1
```

li=18 co=1 INS (LF)

Mais vous, les veinards, vous avez

The screenshot shows the Hop3X IDE interface. The title bar reads "Hop3x Etudiant - V2.013 - jeudi 7 novembre 2013 - 14:19:14 - (Pierre Jacoboni)/(2013-Cours-RUBY)". The menu bar includes Fichier, Edition, Outils, Assistance, and Langage Ruby. The toolbar has icons for file operations like Open, Save, and Run. The left sidebar is an "Explorateur" showing a folder TD1 containing "CompteEpisode1.rb" and "TestCompte.rb". The main area is an "Éditeur" window titled "CompteEpisode1.rb" with the following code:

```
1  ##
2  # Auteur Pierre Jacoboni
3  # Version 0.1 : Date : Thu Nov 07 14:22:52 CET 2013
4  #
5  #
6  load "CompteEpisode1.rb"
7
8  ##### TESTS #####
9  # Pour tester que new ne marche Pas enlevez les commentaires
10 #compte.new()
11 #Ruby2.6: private method `new' called for Compte:Class(NameError)
12 #
13 #Compte.new(101232,'toto',2000)
14 #Ruby2.6: private method `new' called for Compte:Class(NameError)
15 #
16 monCompte = Compte.ouvrir(101232,'toto',2000)
17 print(monCompte.inspect(),"\n")
18 monCompte.depose(3000)
19 print(monCompte.inspect(),"\n")
20 monCompte.retre(15000)
21 print(monCompte.inspect(),"\n")
22 print monCompte.donneTonSolde()
23
24
25 #<Compte:0x6cef60 #solde=2000, #titulaire="toto", #numero=101232>
26 #<Compte:0x6cef60 #solde=5000, #titulaire="toto", #numero=101232>
27 #<Compte:0x6cef60 #solde=-10000, #titulaire="toto", #numero=101232>
28 #-10000
29
30
31
```

The right side of the editor has a large watermark "Hop3X". Below the editor is a "Console" window showing the execution results:

```
Exécution : TD1 (jeudi 7 novembre - 14:23:37)
Exécution de ruby -KU -Chop3xEtudiant/data/workspace//TD1/ TestCompte.rb

#<Compte:0x007f8c91856b58 @numero=101232, @titulaire="toto", @solde=2000>
#<Compte:0x007f8c91856b58 @numero=101232, @titulaire="toto", @solde=5000>
#<Compte:0x007f8c91856b58 @numero=101232, @titulaire="toto", @solde=-10000>
-10000

Appuyer sur Entrée pour continuer ...
Fin
```

Hop3x Etudiant - V2.013 - jeudi 7 novembre 2013 - 14:19:14 - (Pierre Jacoboni)/(2013-Cours-RUBY)

Fichier Edition Outils Assistance Langage Ruby

Générer la rdoc ⌘MD

- Ligne de commande pour rdoc
- Ligne de commande pour visualiser la doc
- Nouvelle Interface Glade

Ouvrir Glade

Ligne de commande Glade

```
1
2
3
4
5
6 load "CompteEpisode1.rb"
7
8 #<----- TESTS -----
9 # Pour tester que new ne marche Pas enlevez la ligne suivante
10 # Compte.new()
11 # Ruby2:6: private method `new' called for Compte(<Class>)
12
13 # Compte.new(101232, 'toto', 2000)
14 # Ruby2:6: private method `new' called for Compte:<Class>(NameError)
15
16 monCompte = Compte.ouvrir(101232, 'toto', 2000)
17 print(monCompte.inspect(), "\n")
18 monCompte.depose(3000)
19 print(monCompte.inspect(), "\n")
20 monCompte.retire(15000)
21 print(monCompte.inspect(), "\n")
22 print monCompte.donneTonSolde()

23
24
25 #<Compte:0x6cef60 @solde=2000, #titulaire="toto", #numero=101232>
26 #<Compte:0x6cef60 @solde=5000, #titulaire="toto", #numero=101232>
27 #<Compte:0x6cef60 @solde=-10000, #titulaire="toto", #numero=101232>
28 -10000
29
30
31
```

Editeur file.rb TestCompte.rb

Hop3X

Console

Exécution : TD1 (jeudi 7 novembre - 14:23:37)

Exécution de ruby -KU -Chop3xEtudiant/data/workspace//TD1/ TestCompte.rb

```
#<Compte:0x007f8c91856b58 @numero=101232, @titulaire="toto", @solde=2000>
#<Compte:0x007f8c91856b58 @numero=101232, @titulaire="toto", @solde=5000>
#<Compte:0x007f8c91856b58 @numero=101232, @titulaire="toto", @solde=-10000>
-10000
```

Appuyer sur Entrée pour continuer ...
Fin

The screenshot shows a Mac OS X desktop with a window titled "class Compte - RDoc Doc...". The URL in the address bar is "file:///localhost/Users/jacoboni/hop3xEtudiant/data/workspace/TD1/doc/Compte.html". The window contains the following content:

Home Classes Methods

Search

Defined In

hop3xEtudiant/data/workspace/TD1

Parent

Object

Methods

- #new
- #ouvrir
- #dépose
- #donneTonSolde
- #retire

Pages

created.rid

Class and Module Index

Compte

class Compte

Définition RUBY de la classe Début de classe

Public Instance Methods

❶ #dépose(uneSomme)

Les méthodes qui suivent sont publiques puisque on ne précise rien de particulier. Pour déposer une somme on incrémente la variable d'instance @solde

```
# File hop3xEtudiant/data/workspace/TD1/CompteEpisode1.rb, line 48
def dépose (uneSomme)
    @solde += uneSomme
end
```

❷ #donneTonSolde()

Pour accéder au solde il suffit de retourner la variable d'instance @solde

❸ #retire(uneSomme)

Pour retirer une somme on décrémente la variable d'instance @solde

Public Class Methods

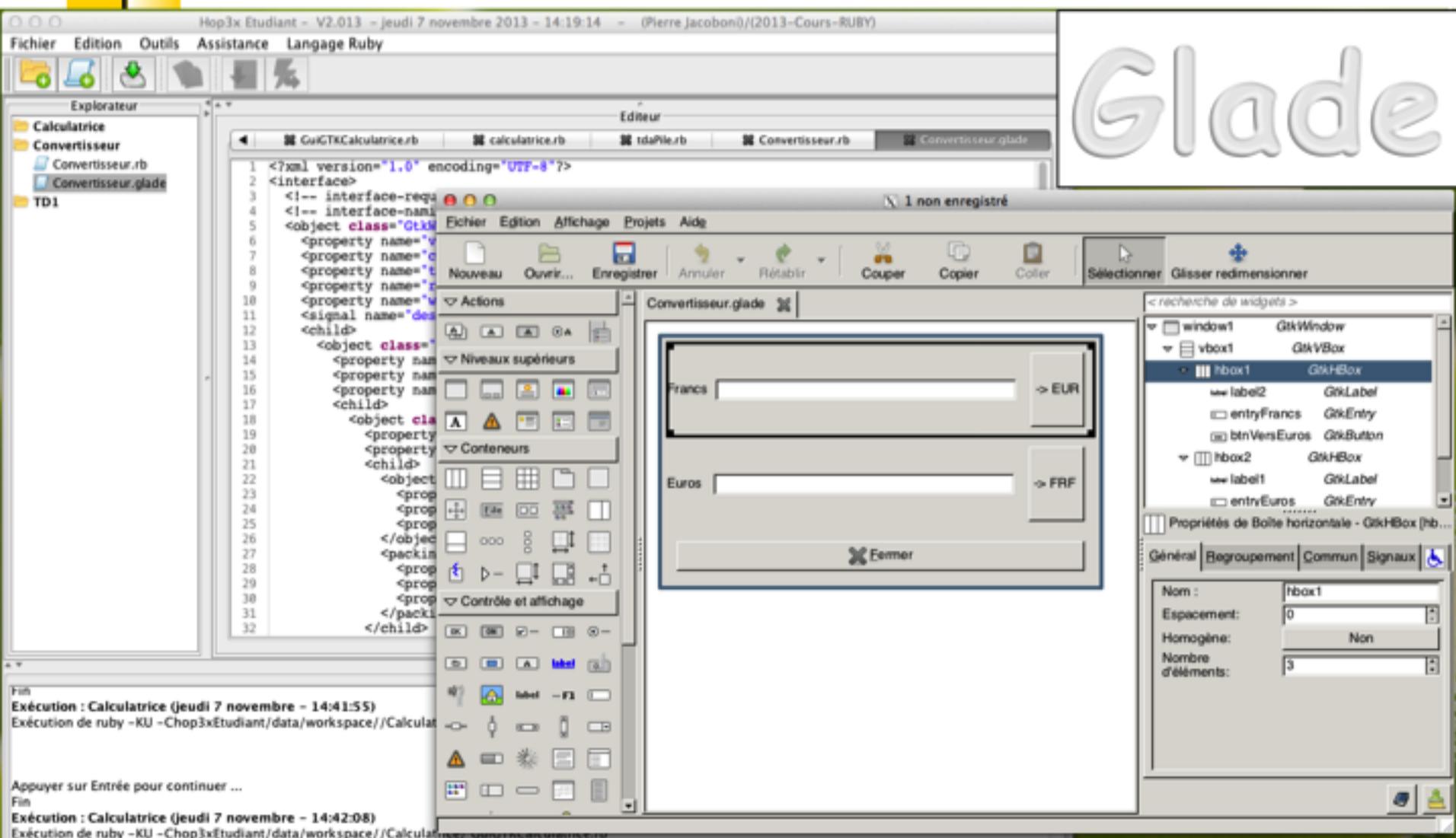
#new(unNombre, unNom, uneSomme)

La méthode d'initialisation d'un compte à trois arguments La méthode initialize est toujours privée même si on ne le spécifie pas. (elle ne peut donc pas être utilisée de l'extérieur)

#ouvrir(unNombre, unNom, uneSomme)

Pour ouvrir un compte on ne peut utiliser que la méthode de classe ouvrir avec 3 arguments
Pour spécifier que c'est une méthode de classe on précise le nom de la classe devant le nom de la méthode donc ici : **compte.ouvrir (...)**

La méthode ouvrir appelle la méthode privée new La méthode privée new appellera la méthode privée d'instance initialize qui initialisera les





Explorateur

Calculatrice
 GuIGTKCalculatrice.rb
 calculatrice.rb
 tdaPile.rb
Convertisseur
 Convertisseur.rb
 Convertisseur.glade
TD1
 CompteEpisodel.rb
 TestCompte.rb

TestCompte.rb

GuIGTKCalculatrice.rb

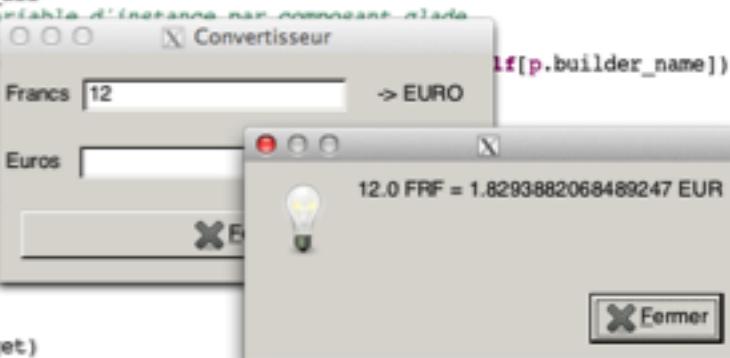
Convertisseur.rb

Convertisseur.glade

```

1  ##
2  # Auteur Pierre Jacoboni
3  # Version 0.1 : Date : Mon Jul 01 10:17:02 CEST 2013
4  #
5  require 'gtk2'
6
7  class Builder < Gtk::Builder
8
9  def initialize
10    super()
11    self.add_from_file(__FILE__.sub(".rb",".glade"))
12
13    self['window1'].set_window_position Gtk::POS_CENTER
14    self['window1'].signal_connect('destroy') { Gtk.main_quit }
15    self['window1'].show_all
16      # Creation d'une variable d'instance non nommante relata
17    self.objects.each{ |o| o.set_name(o.name) }
18    instance_varia
19    puts "#{$p.bui Francs [12] > EURO
20  }
21
22    self.connect_signal Euros [
23      puts handler
24      method(handler)
25    }
26    #taux=6.55957
27
28 end
29
30 def on_window1_destroy(widget)
31   puts "Quitter"
32   Gtk.main_quit

```



Console

```

from Convertisseur.rb:2:in `call'
from Convertisseur.rb:72:in `main'
from Convertisseur.rb:72:in `<class:Builder>'
from Convertisseur.rb:7:in `<main>'

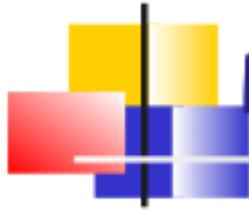
```

Appuyer sur Entrée pour continuer ...

Fin

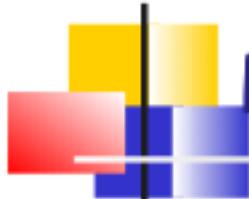
Exécution : Convertisseur (jeudi 7 novembre - 15:15:39)

Exécution de ruby -KU -Chop3xEtudiant/data/workspace//Convertisseur/ Convertisseur.rb



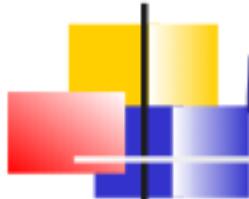
Plan

- A - Introduction à Ruby
 - 1. Pourquoi Ruby
 - 2. Premiers éléments de syntaxe
 - 1- les principes
 - 2- classes/instances
 - 3- transmission de messages
 - 4- variables et affectation
 - 5- instantiation
 - 3. Utiliser Ruby
 - Ligne de commande
 - Interactive Ruby Shell
 - Utilisation d'un IDE



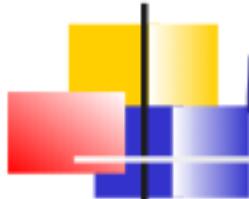
Plan

- B - Les bases du langages
 - 1-Les chaînes de Caractères
 - 2-Les nombres
 - 3-Les Conditionnelles
 - 4-Les Itérations



B1-Les Chaînes de caractères

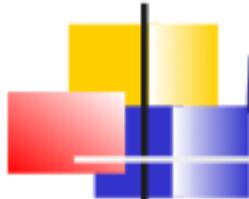
- Dans Ruby, Les chaînes de caractères sont des «containers» homogènes de caractères.
- Une chaîne de caractère peut être entre guillemets ("...") ou entre apostrophes (' ... ').
 - ruby> "abc" → "abc"
 - ruby> 'abc' → "abc"
- Dans certains cas les effets sont différents.
 - Une chaîne entre guillemets permet d'utiliser des séquences d'échappement en mettant un anti-slash devant les caractères voulus, et l'évaluation d'expressions incluses dans la chaîne et entourées de #{}.
 - Une chaîne entre apostrophes ne permet pas tout ça et ne dit rien que ce qu'elle montre.



B1-Les Chaînes de caractères

- Le traitement des chaînes par Ruby est plus intuitif que celui de C.
- On n'a pas à se soucier de la place occupée par une chaîne. On ne s'occupe pas du tout de la gestion de la mémoire.
 - Concaténer des chaînes est beaucoup plus pénible en C parce qu'il faut gérer explicitement l'allocation mémoire correspondante :

```
char *s = malloc(strlen(s1)+strlen(s2)+1);
strcpy(s, s1);
strcat(s, s2);
/* ... */
free(s);
```



B1-Les Chaînes de caractères

- Concaténation :

- ruby> word = "fo" + "o" → "foo"

- Répétition :

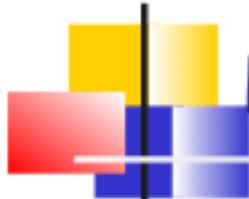
- ruby> word = word * 2 → "foofoo"

- Extraction de caractères :

les caractères sont des entiers pour ruby

- ruby> word[0] → 102 # le code ASCII pour 'f'
 - Pour obtenir "f" il aurait fallu écrire word[0,1]
 - ruby> word[-1] → 111 # le code ASCII pour 'o'
 - Pour obtenir "o" il aurait fallu écrire word[-1,1]

(Les indices positifs indiquent des positions à partir du début de la chaîne,
les indices négatifs à partir de la fin).



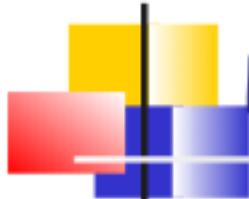
B1-Les Chaînes de caractères

- Extraction de sous-chaînes :

- ruby> herb = "parsley" → "parsley"
- ruby> herb[0,1] → "p"
- ruby> herb[-2,2] → "ey"
- ruby> herb[0..3] → "pars"
- ruby> herb[-5..-2] → "rsle"

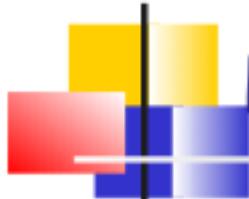
- Test d'égalité :

- ruby> "foo" == "foo" → true
- ruby> "foo" == "bar" → false



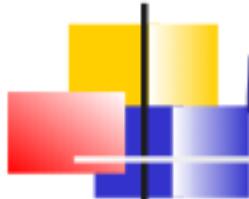
B1-Les Chaînes de caractères

- Modification d'une partie de la chaîne
 - foo="whisbone" → "whisbone"
 - foo[0,1]="f"
 - # foo="fishbone"
 - foo[5,1]="a"
 - # foo="fishbane"
 - foo[0,4]="wolf"
 - # foo="wolfbane"
- La partie à remplacer peut ne pas être de la même longueur
 - foo[1,5)="i"
 - # foo="wine"
 - foo[1,2)="edg"
 - # foo="wedge"



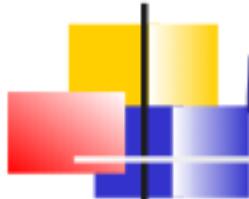
B1-Les Chaînes de caractères

- Comme tout dans Ruby les chaînes de caractères sont des objets.
 - "foofoo".class → String
- En fait, les notations [] et []= correspondent à des messages envoyé à l'objet instance de la classe String correspondant.
 - foo[2] ⇔ foo.[] (2)
 - foo[2, 1] = "d" ⇔ foo.[]= (2, 1, "d")



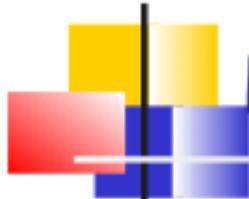
B1-Les Expressions Régulières

- Certains caractères et combinaisons de caractères ont une signification spéciale dans les formes, notamment :
 - [] Spécification d'intervalle (par ex: [a-z] indique une lettre dans l'intervalle a à z)
 - \w Lettre ou chiffre; équivaut à [0-9A-Za-z]
 - \s Caractère espace; équivaut à[\t\n\r\f]
 - \S Caractère non espace
 - \d Chiffre; équivaut à [0-9]
 - * Zero, 1 ou n occurrences de ce qui précède
 - + 1 ou n occurrences de ce qui précède
 - {m,n} Au moins m et au plus n occurrences de ce qui précède
 - ? Au plus une occurrence de ce qui précède; équivaut à {0,1}
 - | Alternative: soit ce qui précède soit ce qui suit



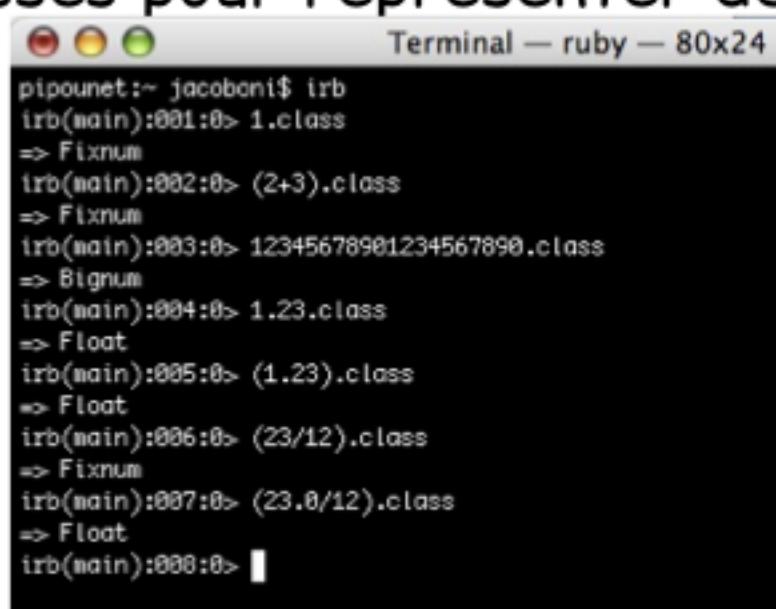
B1-Les Expressions Régulières

- Par exemple, supposons que nous voulions tester si une chaîne correspond à la description suivante:
 - "Commence par un f minuscule, suivi immédiatement par exactement une majuscule, suivie par n'importe quoi sauf des minuscules."
 - Si vous êtes un programmeur C expérimenté, vous avez sûrement déjà écrit mentalement une douzaine de lignes, pas vrai ? Reconnaissez-le, vous n'avez pas pu vous en empêcher.
 - Mais en ruby vous n'avez qu'à tester votre chaîne contre l'expression régulière suivante: `/^f[A-Z][^a-z]*$/`.

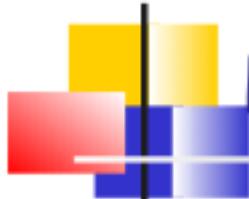


B2-Les Nombres

- Plusieurs classes pour représenter des nombres en Ruby
- Numeric
 - Integer
 - FixNum
 - BigNum
 - Float
- Chacune offre des services classiques appropriés.



```
Terminal — ruby — 80x24
pipounet:~ jacoboni$ irb
irb(main):001:0> 1.class
=> Fixnum
irb(main):002:0> (2+3).class
=> Fixnum
irb(main):003:0> 12345678901234567890.class
=> Bignum
irb(main):004:0> 1.23.class
=> Float
irb(main):005:0> (1.23).class
=> Float
irb(main):006:0> (23/12).class
=> Fixnum
irb(main):007:0> (23.0/12).class
=> Float
irb(main):008:0> |
```



B3-Les Conditionnelles

- Les structures de contrôles en Ruby sont des expressions, et "ont donc une valeur".
- `false` et `nil` sont considérés comme la valeur FAUX, le reste est VRAI.
- `if` et `unless`
 - Classique
 - Modifiée
- `case`

■ Forme Classique

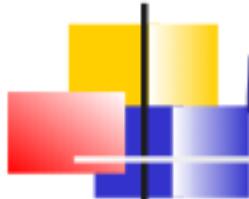
```
if expr [then]
    expr...
[elsif expr [then]
    expr...] ...
[else
    expr...]
```

Ruby utilise "elsif" pas
"else if" ou "elif"

■ Autre forme

expr if expr

- execute l'expression de gauche si expression de droite est vraie.



B32-unless

■ Forme classique

```
unless expr [then]
    expr...
[else
    expr...]
end
```

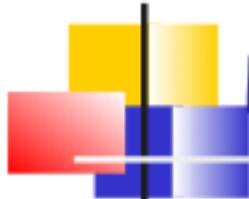
unless s'utilise pour renverser la conditionnelle cela est équivalent à :

```
if !(cond)
    ...
else
    ...
end
```

■ Autre Forme

```
expr unless expr
```

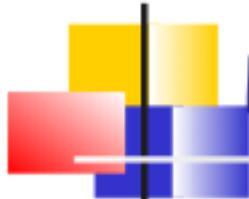
- exécute l'expression de gauche si l'expression droite est fausse.



B33-case

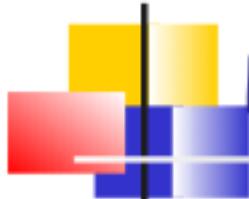
```
case expr
  [when expr [, expr]...[then]
    expr...]
  [else
    expr...]
end
```

- L'expression case est aussi une conditionnelle les comparaisons sont faites avec l'opérateur ==.
 - Attention la sémantique de cet opérateur varie selon les objets considérés



B4-Les Itérations

- Il existe des formes classiques de répétitions comme `while` et `until`.
- Il existe aussi une autre façon de réaliser des itérations en Ruby avec les itérateurs sur les collections. (cours 4)



B41-Les Itérations - while

- Forme classique

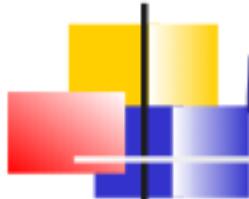
```
while expr [do]
    ...
end
```

- Execution du corps tant que la condition renvoie vrai

- Autre Forme

- ```
expr while expr
```

    - Répète l'évaluation de l'expression de gauche tant que l'expression de droite est vraie



## B42-Les Itérations - until

---

- Forme classique

```
until expr [do]
```

```
...
```

```
end
```

- Exécute le corps jusqu'à ce que la condition soit vraie

- Autre forme

```
expr until expr
```

- Répète l'évaluation de l'expression gauche jusqu'à ce que l'expression de droite soit vraie.