Table of Contents

[Introduction à Ruby 2](#_Toc453243532)

[Documentation en Ruby : ri, RDoc 2](#_Toc453243533)

[Variables et objets 3](#_Toc453243534)

[Les symboles 4](#_Toc453243535)

[Lire de STDIN 5](#_Toc453243536)

[Sauvegarder son code Ruby dans un fichier et l'executer 5](#_Toc453243537)

[Arguments de la ligne de commande 6](#_Toc453243538)

[Travailler avec des fichiers sous Ruby 7](#_Toc453243539)

[Strings 8](#_Toc453243540)

[Mathématique 12](#_Toc453243541)

[Les tableaux 14](#_Toc453243542)

[Les intervales ou les séquences 17](#_Toc453243543)

[Les hashes 18](#_Toc453243544)

[Les conditions 20](#_Toc453243545)

[Des boucles avec ou sans les itérateurs 22](#_Toc453243546)

[Les méthodes 24](#_Toc453243547)

[Les blocks en Ruby 26](#_Toc453243548)

[Les Procs 27](#_Toc453243549)

[Les exceptions 28](#_Toc453243550)

[Les classes 30](#_Toc453243551)

[Les expressions régulières 36](#_Toc453243552)

[Les logs en Ruby 38](#_Toc453243553)

[Le temps en Ruby 39](#_Toc453243554)

[Les objet immutables : freeze & frozen 40](#_Toc453243555)

[Les modules 41](#_Toc453243556)

[La sérialisation des objets (*marshaling*) 42](#_Toc453243557)

# Introduction à Ruby

Ruby est un langage de programmation créé par un informaticien Japonais dans les années 90, avec l'objectif de faciliter la vie des développeurs avec une syntaxe et des outils simples et agréables à utiliser.

**Installation :**

Vérifiez si Ruby n’est pas déjà installé sur votre machine, et si vous avez une version supérieure à 2.0.0 : ruby -v

Pour l'installer sur Ubuntu, lancez la commande : sudo apt-get install ruby-full

Les commentaires :

- pour une seule ligne, le symbole diez **#**

- pour plusieurs lignes : entre **=begin**  et **=end**

# Documentation en Ruby : ri, RDoc

La documentation la plus complète : [online](http://ruby-doc.org/core-2.3.0/)

**ri** = Ruby Index; outil en ligne de commande (utilisé pour visualiser la doc de Ruby en mode offline )

ri Array

ri Array**.**sort # séparer une méthode de sa classe, ou de son module

ri Hash**#**each # fait référence à une méthode d'instance

ri Math**::**sqrt # fait référence à une méthode de classe

**RDoc** = Ruby Documentation; système qui inclue l'outil de ligne de commande rdoc

Si l'on inclue des commentaires dans nos programmes Ruby en suivant les prescriptions de RDdoc (commentaires inclus avant la définition d'une méthode, d'une classe, ...), l'outil **rdoc** scan le fichier, extrait les commentaires, les organise de manière intelligente et crée des documentations joliment formatés.

# Variables et objets

Pour exécuter du code Ruby directement dans votre console, il faut utiliser la console Ruby appelée IRB (Interactive Ruby) :

**irb**

**puts** "Hello" # afficher un message à l'écran

check = true # variable boolean (true/false)

a = 1 # => 1 # entier

b = a + 2 # => 3 # entier

msg = "Hello World" # chaîne des caractères

msg2 = msg + " ! " # => "Hello world ! "

Pour quitter l'IRB, tapez exit

Une **variable** est une sorte de "boîte" dans laquelle on range un contenu qui peut être de différentes formes : un nombre, une chaîne de caractère (ou string)...

Toutes les **variables en Ruby** sont **des objets**.

On utilise l'opérateur **=** pour ranger un objet dans une variable, par exemple :

cadeau = "bon pour un voyage autour du monde"

On peut effectuer des opérations mathématiques sur des variables contenant des nombres :

age\_de\_ruby = 2015 - annee\_de\_naissance\_ruby

On peut regrouper des variables contenant des chaînes de caractères avec l'opérateur **+** (concaténation)

msg = "Bravo, tu as reçu un " + cadeau + " !" # => "Bravo, tu as reçu un bon pour un voyage ..."

Le contenu vide : **nil**

Les variables boolean : **true**, **false**

En Ruby, toutes les valeurs sont "true", sauf les mots clés réservés "false" et "nil".

Pour attribuer une valeur à une variable seulement si elle n'a pas déjà été initialisée :

@variable **||=** "default value"

Variables **globales** :

- commencent avec le symbole **$**

- par exemple, la variable **$0** qui stocke le nom du programme qui a été lancé

Variables **locales** :

- pour chaque classe (block "class"), chaque module (block "module"), chaque méthode (block "def")

- commencent avec une lettre minuscule ou un underscore **\_** (ex: sunil, \_z, hit\_and\_run)

Variables d'**instance** (qui appartiennent à l'objet) :

- commencent avec un **@** (ex: @sign, @\_, @counter)

Variables **constantes** :

- contient des lettres majuscules, chiffres et/ou underscore

Les noms des classes et des modules en CamelCase (ex: module MyMath, PI=3.1416, class MyPune)

Conventions de **nommage en Ruby** :

- utiliser des underscore pour séparer des mots

- utiliser que des lettre majuscules pour les noms des constantes (avec des underscores si besoin)

- les noms des classes et des modules : première lettre majuscule et CamelCase (LikeThis)

- les noms des méthodes : première lettre minuscule (ou un underscore); peuvent se finir avec un des symboles "**?**", "**!**" et "**=**"

Types de variables en Ruby :

**Numeric** (Fixnum, Bignum, Float), **String**, **Array**, **Hash**, **Object**, **Symbol**, **Range, RegExp**

# Les symboles

Les symboles ont l'apparence d'une variable mais sont préfixés avec un deux-points "**:**".

Il n'est pas nécessaire de pre-déclarer un symbole ou de leur assigner une valeur.

Les symboles sont garantis d'être **uniques** (ont toujours la même valeur, par tout dans le programme).

puts "string".object\_id # => 21066960

puts **:symbol**.object\_id # =>132178

Comparaison entre symbole et string :

know\_ruby = **:yes**

if know\_ruby == **:yes**

puts 'You are a Rubyist' # => "You are a Rubyist"

end

know\_ruby = "yes"

if know\_ruby == "yes"

puts 'You are a Rubyist' # => "You are a Rubyist"

end

Dans les deux exemples, le résultat est le même, mais la première version est plus efficace.

Dans le deuxième exemple, chaque appel de "yes" crée un nouvel objet stocké séparément en mémoire.

Dans le premier exemple, les symboles sont des valeurs de référence unique, initialisés une seule fois.

On peut transformer un string dans un symbole et vice-versa :

puts "string".to\_sym.class # Symbol

puts :symbol.to\_s.class # String

# Lire de STDIN

puts "In which city do you stay?"

**STDOUT.flush**

city = **gets.chomp**

puts "The city is " + city

**STDOUT** est une constante globale, qui corresponde au flux de sortie standard réelle du programme.

Appeler la méthode **flush** n'est pas obligatoire, mais recommandé.

Pour éviter d'appeler à chaque fois la méthode flush, activez l'option STDOUT.sync = true

**gets** accepte une seule ligne d'entrée; la méthode **chomp** supprime le retour à la ligne.

# Sauvegarder son code Ruby dans un fichier et l'executer

Ici nous allons **écrire des commandes** Ruby dans un fichier, puis **exécuter ce fichier** dans la console.

Je vous propose donc de :

* créer un fichier vide test**.rb** ;
* l'ouvrir dans votre éditeur de texte à côté de votre console ;
* placer votre console dans le dossier qui contient votre fichier test.rb

Le **header de ruby** sous linux : #!/usr/bin/env ruby

Pour **l'exécuter** : ruby test.rb

# Arguments de la ligne de commande

Les arguments sont stockés automatiquement dans un array spécial appelé **ARGV**.

f = **ARGV**[0]

puts f

ruby tmp.rb 23 # => 23

**La librairie GetoptLong**

Supporte les arguments en ligne de commande qui ont la forme -char ou --string

L'ordre des arguments n'est pas important.

Une option peut avoir plusieurs formes (par exemple, verbose : -v, --verbose ou --details )

Exemple d'appel

ruby testarg.rb **–**h ftp.ibiblio.org **–**u anonymous **–**p s@s.com

# Le code du script testarg

**require** **'optparse'** # include

options = {}

parser = OptionParser.new do |opts|

opts**.banner** = 'Usage: ruby testarg.rb [options]'

opts**.on**('-h', '--hostname HOST, 'Hostname'){|v| options[:hostname] = v }

opts**.on**('-u', '--username USER, 'Username'){|v| options[:username] = v }

opts**.on**('-p', '--password PASS, 'Password'){|v| options[:password] = v }

opts**.on**('--help', 'Displays Help'){puts opts; exit }

end

begin

parser.parse!

rescue OptionParser::ParseError => e

STDERR.puts("Exception encountered while parsing the arguments: #{e} !")

exit

end

# Travailler avec des fichiers sous Ruby

**Ouvrir et lire un fichier :**

**File.open**('p014constructs.rb', '**r**') **do** **|f1|** # f1 = handler du fichier overt en mode read

while line = f1**.gets**

puts line

end

**end**

**File.open**('p014constructs.rb', '**r**').each\_line **do** **|line|** # lire ligne par ligne

puts(line)

**end**

f = **File.open**('in.txt').read # lire tout le contenu du fichier

f**.gsub!**(/\r\n?/, "\n")

f**.each\_line**{|line| print "#{line}" }

f**.close**

**Ouvrir un fichier en mode écriture :**

**File.open**('test.rb', '**w**') **do** **|f2|** # f2 = handler du fichier overt en mode write

f2**.puts** "Created by Satish\nThank God!"

**end**

f = **File.new**("out.txt", "w")

f**.write**("1234567890") # => 10

f**.close**

Modes d'ouverture d'un fichier sous Ruby :

* **"r" :** read only
* **"r+" :** read/write; commence au début du fichier
* **"w" :** write only

**Méthodes utiles :**

**File.basename**("/home/gumby/work/ruby.rb") #=> "ruby.rb"

**File.basename**("/home/gumby/work/ruby.rb", "**.**rb") #=> "ruby"

**File.basename**("/home/gumby/work/ruby.rb", "**.**\*") #=> "ruby"

**File.dirname**("/home/gumby/work/ruby.rb") # => "/home/gumby/work"

**File.extname**("test.rb") # => ".rb"

**File.extname**("a/b/d/test.rb") # => ".rb"

**File.extname**(".a/b/d/test.rb") # => ".rb"

**File.extname**("foo.") # => ""

**File.extname**("test") # => ""

**File.extname**(".profile") # => ""

**File.extname**(".profile.sh") # => ".sh"

**File.exist?**('file\_name.csv') # => true / false ; vérifie si le fichier ou le répertoire existe

**File.file?**('file\_name.csv') # => true / false ; vérifie si le fichier existe

**Encodages** : ouvrir un fichier en mode lecture; encodage fichier UTF-16LE; lecture en UTF-8

File.open('file.txt', '**r**:UTF-16LE:UTF-8') do |f1| ...

# Strings

Afficher la liste des méthodes de la classe String : **String.methods.sort**

Afficher la liste des méthodes des instances de String : **String.instance\_methods.sort**

-- ignorer les méthodes héritées de la class Objet : **String.instance\_methods(false).sort**

s = 'No escape sequence\n' # affiche les caractères \n

s = "Escape sequence\n" # affiche le retour à la ligne

name = 'Jim'

s = 'Hello #{name}' # affiche les caractères #{name}

s = "Hello **#{name}**" # affiche le contenu de la variable m

s = "Hello World"

s**.size** # => 11

s**.clear** # => ''

"hello" **\*** 5 # => "hellohellohellohellohello"

"hello"**.capitalize**  # => "Hello"

"HELLO"**.downcase**  # => "hello" # marche pas bien avec les accents

"hello"**.upcase**  # => "HELLO" # marche pas bien avec les accents

"Hello"**.swapcase** # => "hELLO" # marche pas bien avec les accents

"abcdef"**.casecmp**("abcde") # => 1

"aBcDeF"**.casecmp**("abcdef") # => 0

"hello"**.reverse**  # => "olleh"

"hello"**.center**(4) # => "hello"

"hello"**.center**(20) # => " hello "

"hello"**.ljust**(4) # => "hello"

"hello"**.ljust**(20) # => "hello "

"hello"**.rjust**(4) # => "hello"

"hello"**.rjust**(20, '-') # => "---------------hello"

"hello"**.chomp** # => "hello" supprime les \r et \n en fin de chaine

"hello\n"**.chomp** # => "hello"

"hello\r\n"**.chomp** # => "hello"

"hello\n\r"**.chomp** # => "hello\n"

"hello\r"**.chomp** # => "hello"

"hello \n there"**.chomp** # => "hello \n there"

"hello"**.chomp**("llo") # => "he"

"hello\r\n\r\n"**.chomp**('') # => "hello"

"hello\r\n\r\r\n"**.chomp**('') # => "hello\r\n\r"

"hello"**.chop** # => "hell" supprime le dernier caractère

" hello "**.strip** # => "hello" supprime les espace en début et fin de chaine

"\tgoodbye\r\n"**.strip** # => "goodbye" renvoie nil si aucun caractère a été supprimé

a = "hello "

a **<<** "world" # => "hello world" concate sans créer un objet supplémentaire

a**.concat**(33) # => "hello world**!**" ajoute un objet converti en str

a**.count** "lo" # => 5 le nombre d'occurrences des lettres l et o

a**.count** "lo" , "o" # => 2 le nombre d'occurrences de "o" dans ...

a**.count** "hello", "**^**l" # => 4 le nombre de lettres différentes de l dans ..

a**.count** "a**-**e" # => 2 d, e

"hello"**.delete** "l","lo" # => "heo" supprime l'intersection de ses arguments

"hello"**.delete** "lo" # => "he"

s = "hello there"

puts s[0**..**1] # affiche "he" les caractères entre positions 0 et 1

puts s[0**,**2] # affiche "he" à partir de la position 0, 2 chars

puts s[1..3] # affiche "ell"

puts s[-3, 2] # affiche "er" à partir du 3eme dernier char, 2 chars

s = "test"

s[2..3] = "mp" # s= "temp" remplace des caractères

s[0..1] = "bu" # s= "bump"

s[0..1] = "shri" # s= "shrimp"

s[0] = "" # s= "hrimp" supprime un caractère

s[0] = "sh" # s= "shrimp"

"abcd"**.insert**(0, 'X') # => "Xabcd"

"abcd"**.insert**(3, 'X') # => "abcXd"

"abcd"**.insert**(4, 'X') # => "abcdX"

"abcd"**.insert**(-3, 'X') # => "abXcd"

"abcd"**.insert**(-1, 'X') # => "abcdX"

ex = "test1,test2,test3,test4,test5"

ex**.split**(",")**.first** # => "test1"

ex**.split**(",")**.last** # => "test5"

ex**.split**(',', 2) # => "test1" "test2, test3, test4, test5" ; split into **2 pieces**, not more

ex**.split**(',', 2).last # => "test2, test3, test4, test5"

all\_but\_first = ex**.split**(/,/)**[1..-1]** # => ["test2", "test3, "test4", "test5"]

names = "Alice,Bob,Eve"

names\_a = names**.split**(/,/) # équivalent avec .split(",")

names\_a.push("Carol", "Dave")

names = names\_a**.join**(',') # => "Alice,Bob,Eve,Carol,Dave"

"hello"**.each\_char** {|c| print c, ' ' } # => h e l l o

"hello\nworld"**.each\_line** {|s| puts s} # => "hello\n" "world"

"hello\nworld"**.each\_line**('l') {|s| puts s} # => "hel" "l "o\nworl" "d"

"hello\n\n\nworld"**.each\_line**('') {|s| p s} # => "hello\n\n\n" "world"

"hello"**.end\_with?**("ello") # => true

"hello"**.end\_with?**("heaven", "ello") # => true si au moins un match

"hello"**.end\_with?**("heaven", "paradise") # => false

"hello"**.start\_with?**("hell") # => true

"hello"**.start\_with?**("heaven", "hell") # => true

"hello"**.start\_with?**("heaven", "paradise") # => false

"hello"**.include?** "lo" # => true

"hello"**.include?** "ol" # => false

"hello"**.index**('lo') # => 3

"hello"**.index**('a') # => nil

"hello"**.index**('?e') # => 1

"hello"**.index**(**/**[aeiou]**/**, -3) # => 4 cherche à partir de la position -3

"hello"**.index**('l') # => 2

"hello"**.rindex**('l') # => 3 dernière occurrence

Substitutions avec des **expressions régulières** :

s = "My hovercraft is full of eels"

s**.sub!**(/hovercraft/, 'spaceship') # => My spaceship is full of eels

s**.sub!**(/eels/, 'tribbles') # => My spaceship is full of tribbles

# \w means "word character", \w+ means a sequence of word characters (match words separated by spaces)

s**.gsub!**(/\w+/) do |w|

w.capitalize # => My Spaceship Is Full Of Tribbles

end

"hello"**.sub**(**/**[aeiou]**/**, '\*') # => "h\*llo" première occurrence

"hello"**.gsub**(**/**[aeiou]**/**, '\*') # => "h\*ll\*" toutes les occurrences

"hello"**.gsub**(**/**([aeiou])**/**, '<**\1**>') # => "h<e>ll<o>"

"hello"**.gsub**(**/**.**/**) {|s| s.**ord**.**to\_s** + ' '} # => "104 101 108 108 111 "

"hello"**.gsub**(**/**(?<foo>[aeiou])**/**, '{\k<foo>}') # => "h{e}ll{o}" nom du groupe

'hello'**.gsub**(**/**[eo]**/**, 'e' **=>** 3, 'o' **=>** '\*') # => "h3ll\*"

"hello"**.tr**('el', 'ip') # => "hippo"

"hello"**.tr**('aeiou', '\*') # => "h\*ll\*"

"hello"**.tr**('aeiou', 'AA\*') # => "hAll\*"

"hello"**.tr**('a-y', 'b-z') #=> "ifmmp"

"hello"**.tr**('^aeiou', '\*') #=> "\*e\*\*o"

"hello^world"**.tr**("\\^aeiou", "\*") #=> "h\*ll\*\*w\*rld" \\^ == le caractère ^

"hello-world"**.tr**("a\\-eo", "\*") #=> "h\*ll\*\*w\*rld"

"hello\r\nworld"**.tr**("\r", "") #=> "hello\nworld"

"hello\r\nworld"**.tr**("\\r", "") #=> "hello\r\nwold"

"hello\r\nworld"**.tr**("\\\r", "") #=> "hello\nworld"

"X['\\b']"**.tr**("X\\", "") #=> "['b']"

"X['\\b']"**.tr**("X-\\]", "") #=> "'b'"

"a8"**.upto**("b6") {|s| print s, ' ' } # => a8 a9 b0 b1 b2 b3 b4 b5 b6

"9"**.upto**("11").to\_a # => ["9", "10", "11"]

"07"**.upto**("11").to\_a # => ["07", "08", "09", "10", "11"]

**Comparer** deux strings : par **contenu** (==, eql), par **référence** (equal)

s1 = 'Jonathan'

s2 = 'Jonathan'

s3 = s1

if s1 **==** s2

puts 'Both Strings have identical content' # match

else

puts 'Both Strings do not have identical content'

end

if s1**.eql?(**s2)

puts 'Both Strings have identical content' # match

else

puts 'Both Strings do not have identical content'

end

if s1**.equal?(**s2)

puts 'Two Strings are identical objects'

else

puts 'Two Strings are not identical objects' # match

end

if s1**.equal?(**s3)

puts 'Two Strings are identical objects' # match

else

puts 'Two Strings are not identical objects'

end

**Encodage :** spécifie l'encodage d'un programme Ruby dans l'en-tête du fichier : # encoding: utf-8

puts "λ".encoding # => UTF-8

"\xc2\xa1"**.force\_encoding**("UTF-8")**.valid\_encoding?** # => true

"\xc2"**.force\_encoding**("UTF-8")**.valid\_encoding?** # => false

"\x80"**.force\_encoding**("UTF-8")**.valid\_encoding?** # => false

# Mathématique

**Constantes :**

EPSILON

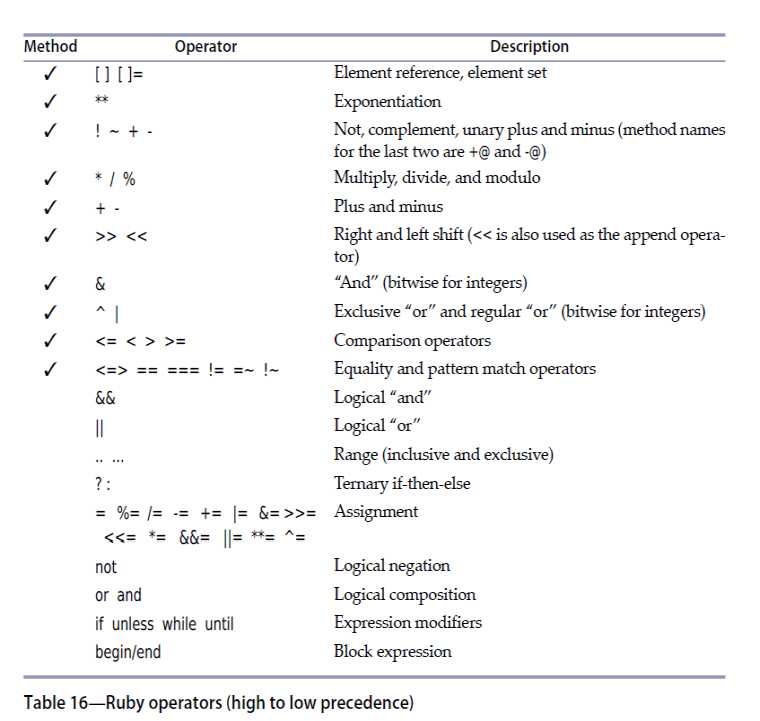
INFINITY

MAX # largest integer in a double precision float number (usually 1.7976931348623157e+308)

MIN # smallest positive integer in a double precision float (usually 2.2250738585072014e-308)

NAN # expression representing a value which is not a number

Float::MAX # => 1.79769313486232e+308



**Operations :**

a = 5

b = a + 2 # => 7

b = a - 2 # => 3

b = a \* 2 # => 10

b = a \*\* 2 # => 25 pow

b = a / 2 # => 2 integer division

b = a % 2 # => 1 modulo

b = a / 2.0 # => 2.5 float division

b = a**.to\_f** / 2 # => 2.5

b = a / 2**.to\_f** # => 2.5

b = a**.fdiv**(2) # => 2.5

b = **Math.sqrt**(9) # => 3.0 module Math

6543.21**.modulo**(137) # => 104.21

42.0**.divmod**(5) # => [8, 2.0]

(-34.56)**.abs** # => 34.56

1.2**.ceil** # => 2

2.0**.ceil** # => 2

(-1.2)**.ceil** # => -1

(-2.0)**.ceil** # => -2

1.2**.floor** # => 1

2.0**.floor** # => 2

(-1.2)**.floor** # => -2

(-2.0)**.floor** # => -2

1.4**.round** # => 1

1.5**.round**  # => 2

(-1.5) **.round** # => -2

1.234567**.round**(2) # => 1.23

1.234567**.round**(5) # => 1.23457

1.723456**.to\_i** # => 1 convert to integer

1**.to\_f** # => 1.0 convert to float

1**.to\_s** # => "1" convert to string

1**.to\_a** # => [1] convert to array

**Afficher seulement deux décimales :**

puts format("%.2f", 9790/6) # => 1631.00

puts format("%.2f", 9790/6.0) # => 1631.67

**Les nombres au hasard :**

**rand**() # => une valeur flottante au hasard entre 0 et 1

**rand**(10) # => une valeur entière au hasard entre 0 et 9

**rand**(3..15) # => une valeur entière au hasard entre 3 et 15 (**INCLUS**)

**rand**(3...15) # => une valeur entière au hasard entre 3 et 14 (**EXCLUS**)

# Les tableaux

Les **tableaux** permettent de ranger des données de façon ordonnée, qu'on retrouve à l'aide d'un index.

Pour créer un tableau :

on utilise les crochets **[ ]** ou on crée une nouvelle instance de la classe array avec **Array.new**(...)

tableau\_vide = **Array.new**(3) # => [nil, nil, nil]

tableau\_bool = **Array.new**(3, true) # => [true, true, true]

tableau\_num = **Array.new**(3, 0) # => [0, 0, 0]

tableau\_num = **Array.new**(3) {|index| index \*\* 2 } # => [0, 1, 4]

tableau\_de\_hash = **Array.new**(4) {Hash.new } # => [{}, {}, {}, {}]

tableau\_de\_tableaux = **Array.new**(3) {Array.new(2) } # => **matrice** 3x2 [[nil, nil], [nil, nil], [nil, nil]]

[1, 2, [[3, 4], [5, 7, 8]], [9, 10]]**.flatten** # => [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] ; existe aussi **.flatten!**

arr = [1, 2, 3, 4, 5]

arr**.map**{|a| 2 \* a } # => [2, 4, 6, 8, 10] ; arr = [1, 2, 3, 4, 5]

arr**.map!**{|a| a \*\* 2 } # => [1, 4, 9, 16, 25] ; arr = [1, 4, 9, 16, 25]

mon\_tableau = **[**1, 5, 7, 39, 5, 15, 20**]** # tableau de 3 entiers

mon\_tableau[1] # => 5 (index à partir de 0)

mon\_tableau**[-1]**  # => 20

mon\_tableau**.at**(3) # => 39 ; équivalent avec mon\_tableau[3]

mon\_tableau**[2..4]**  # => [7, 39, 5]

mon\_tableau**.take**(3) # => [1, 5, 7] ; les 3 premiers éléments

mon\_tableau**[**10**]**  # => nil ; sans aucune exception

mon\_tableau**.fetch**(10) # => soulève l'exception IndexError

mon\_tableau**.size** # => 7; pareil avec les alias **.length** et **.count**

mon\_tableau**.first** # => 1

mon\_tableau**.last**  # => 20

mon\_tableau**.max**  # => 39

mon\_tableau**.min**  # => 1

mon\_tableau**.sort**  # => [1, 5, 5, 7, 15, ...] ; ne modifie pas la variable

mon\_tableau**.sort!**  # => [1, 5, 5, 7, 15, ...] ; **modifie** la variable

mon\_tableau**.empty?**  # => false

mon\_tableau**.include?**(5) # => true

mon\_tableau**.count**(2) # => 0

mon\_tableau**.count**(5) # => 2

mon\_tableau**.count {**|x| x % 2 == 0 **}**  # => 1 (l'élément 20)

mon\_tableau**.index**(7) # => 2

mon\_tableau**.index**(17) # => nil

mon\_tableau**.rindex**(5) # => 4 (le dernier index)

**Modifier, ajouter et supprimer des éléments**

utilisateurs = ["Alice", "Bob", "John"] # tableau de 3 strings

utilisateurs[2] = "Johnny" # modifier un élément du tableau

utilisateurs **<<** "Marc" # ajouter un nouvel élément à la fin

utilisateurs**.push**("Dave") # ajouter un nouvel élément à la fin

utilisateurs**.unshift**("Luc") # ajouter un nouvel élément au début

utilisateurs**.insert**(3, "Alex") # insérer un nouvel élément à la position 3

utilisateurs**.pop**  # supprimer le dernier élément

utilisateurs**.shift**  # supprimer le premier élément

utilisateur**.delete\_at**(2) # supprimer l'élément à l'index 2

utilisateurs**.delete**("Bob") # supprimer l'entrée "Bob" du tableau

utilisateurs **-=** "Bob" # équivalent au .delete("Bob")

arr = [1, 2, 3]

arr**.concat**([ 4, 5, 6]) # => [1, 2, 3, 4, 5, 6] ; **modifie** la variable

arr**.any?**{|a| a > 3 } # => true ; s’arrête a la première valeur valide

arr**.all?**{|a| a > 3 } # => false

arr**.select**{|a| a > 3 } # => [4, 5, 6] ; existe aussi **.select!**

arr**.reject**{|a| a < 5 } # => [ 5, 6] ; existe aussi **.reject!**

arr**.drop\_while**{|a| a < 5 } # => [ 5, 6] ; ne modifie pas la variable

arr = [1, 2, 3, 4, 5, 6]

arr**.delete\_if**{|a| a < 4 } # => [4, 5, 6] ; **modifie** la variable

arr**.keep\_if**{|a| a < 4 } # => [1, 2, 3] ; **modifie** la variable

arr = ['foo', 0, nil, 'bar', 7, 'baz', nil]

arr**.compact** #=> ['foo', 0, 'bar', 7, 'baz']; existe aussi **.compact!**

arr = [2, 5, 6, 556, 6, 6, 8, 9, 0, 123, 556]

arr**.uniq** # => [2, 5, 6, 556, 8, 9, 0, 123]; existe aussi **.uniq!**

**Créer un array des mots :**

names1 = ['ann', 'richard', 'william', 'susan', 'pat']

puts names1[0] # => ann

puts names1[3] # => susan

names2 = **%w**{ann richard william susan pat}

puts names2[0] # => ann

puts names2[3] # => susan

**Convertir le contenu d'un array dans un string :**

["a", "b", "c" ]**.join** # => "abc"

["a", "b", "c" ]**.join**("-") # => "a-b-c"

**Assignation parallèle :**

a = 1, 2, 3, 4 # => a == [1, 2, 3, 4]

b = [1, 2, 3, 4] # => b == [1, 2, 3, 4]

a, b = 1, 2, 3, 4 # => a == 1, b == 2

c, = 1, 2, 3, 4 # => c == 1

**Randoms :**

a = [1, 2, 3, 4, 5, 6, 7]

a**.sample** # => 1 ou 5 ou 3 .... choix d'un élément de l'array au hasard

a**.sample**(3) # => [2, 5, 7] ou [1, 4, 7] , ....

a**.shuffle** # => [2, 6, 3, 1, 7, 4, 5] ou ... mélanger les éléments ; existe aussi **.shuffle!**

**Permutations et combinaisons :**

a = [1, 2, 3]

a**.permutation**.to\_a # => [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

a**.permutation**(1).to\_a # => [[1],[2],[3]]

a**.permutation**(2).to\_a # => [[1,2],[1,3],[2,1],[2,3],[3,1],[3,2]]

a**.permutation**(3).to\_a # => [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

a**.permutation**(0).to\_a # => [[]] # one permutation of length 0

a**.permutation**(4).to\_a # => [] # no permutations of length 4

a = [1, 2, 3, 4]

a**.combination**(1).to\_a # => [[1],[2],[3],[4]]

a**.combination**(2).to\_a # => [[1,2],[1,3],[1,4],[2,3],[2,4],[3,4]]

a**.combination**(3).to\_a # => [[1,2,3],[1,2,4],[1,3,4],[2,3,4]]

a**.combination**(4).to\_a # => [[1,2,3,4]]

a**.combination**(0).to\_a # => [[]] # one combination of length 0

a**.combination**(5).to\_a # => [] # no combinations of length 5

**Transpose :**

a = [[1,2], [3,4], [5,6]]

a**.transpose** #=> [[1, 3, 5], [2, 4, 6]]

**Product :**

[1,2,3]**.product**([4,5]) #=> [[1,4],[1,5],[2,4],[2,5],[3,4],[3,5]]

[1,2]**.product**([1,2]) #=> [[1,1],[1,2],[2,1],[2,2]]

[1,2]**.product**([3,4],[5,6]) #=> [[1,3,5],[1,3,6],[1,4,5],[1,4,6], [2,3,5],[2,3,6],[2,4,5],[2,4,6]]

[1,2]**.product**() #=> [[1],[2]]

[1,2]**.product**([]) # => []

# Les intervales ou les séquences

Sous Ruby, l'on peut spécifier des intervalles en utilisant

* deux points ".." : inclusion des deux extrémités; ex 1..5 = [1, 2, 3, 4, 5]
* trois points "..." : exclusion de l'extrémité droite; ex 1...5 = [1, 2, 3, 4]

Ces intervalles ne sont pas traduites automatiquement sous forme d'un array; ils sont des objets **Range**.

Ces objets de type Range peuvent être facilement convertis en array avec la méthode "**.to\_a**"

**(**1**..**10**).to\_a**  # => [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

digits = -1**..**9

digits.include?(5) # => true

digits.min # => -1

digits.max # => 9

digits.reject{|x| x < 5} # => [5, 6, 7, 8, 9]

Les **tests d'inclusion** dans les intervalles (utilisant l'operateur d'égalité **===** )

(1..10) **===** 5 # => true

(1..10) **===** 15 # => false

(1..10) **===** 3.14159 # => true

('a'..'j') **===** 'c' # => true

('a'..'j') **===** 'z' # => false

# Les hashes

Les **hashes** permettent de ranger des données que l'on retrouve à l'aide d'une clé.

Pour créer une table de hashage, on utilise des accolades **{ }**.

Chaque valeur rangée dans une hash est associée à une clé (qui permettra de la retrouver).

Les hashes ont l’avantage d'accepter n'importe quel objet comme clé.

**Des hashes avec des clés de différents types :**

h = {'dog' => 'canine', 'cat' => 'feline', 'donkey' => 'asinine', 12 => 'dodecine'}

puts h**.size** # => 4

puts h['dog'] # => 'canine'

puts h[12] # => dodecine'

**Des hashes avec des clés de type string :**

grades = **Hash.**new

grades**["**Dorothy**"]** = 9

grades**["**Thomas**"]** = 15

**Des hashes avec des clés de type symbole** *(conseillé)*

(deux notations :<key> => <value> ou <key>:<value> )

books = **{:**matz => "The Ruby Language", **:**black => " The Well-Grounded Rubyist"**}**

jours\_voyage = **{**paris**:** 0, toronto**:** 7, nyc**:** 3**}** # table de hashage de 3 éléments

jours\_voyage**[:**toronto**]** # => 7

jours\_voyage**[:**rio**]** = 5 # ajoute un nouvel élément

jours\_voyage**[:**rio**]** = 7 # modifie la valeur d'un élément

**Creer des hash avec des valeurs par défaut :**

h = **Hash**.new # => {}

h**.default** # => nil

h = **Hash**.new(**"cat"**) # => {}

h**.default** # => "cat"

h = { "a" => 100, "b" => 200 }

h**.default** = "Go fish"

h["a"] # => 100

h["z"] # => "Go fish"

**Méthodes utiles des hashes :**

h = **{**"a" => 100, "b" => 200, "c" => 300**}**

h.**size** # => 3

h["a"] # => 100

h**.fetch**("a") # => 100

h["d"] # => nil

h**.fetch**("d") # => soulève une exception KeyError

h.**entries** # => [["a", 100], ["b", 200], ["c", 300]]

h**.keys** # => ["a", "b", "c"]

h**.values** # => [100, 200, 300]

h**.key**(200) # => "b" # la clé associé à cette valeur

h.**values\_at**("a", "c") # => [100, 300]

h**.clear**  # => {}

h**.delete**("a") # => supprime l'élément "a"; retourne 100

h**.delete\_if**{|key, value| key >= "b" } #=> {"a"=>100} ;

h**.shift** # => supprime une entrée du hash (au hasard)

h**.each** # => #<Enumerator: {"a"=>100, ...}:each>

h**.each\_key** # => #...:each\_key>

h**.each\_value** # => #...:each\_value>

h**.empty?** # => false

h**.key?**("a") # => true ; équivalent avec .has\_key?

h**.value?**(500) # => false ; équivalent avec .has\_value?

h**.invert** # => {100=>"a", 200=>"b", 300=>"c"}

h = { "a" => 100, "b" => 200, "c" => 300 }

h**.reject**{|k,v| k < "b"} # => {"b" => 200, "c" => 300} ; existe aussi **.reject!**

h**.reject**{|k,v| v > 100} # => {"a" => 100}

h**.select**{|k,v| k > "a"} # => {"b" => 200, "c" => 300}; existe aussi **.select!**

h**.select**{|k,v| v < 200} # => {"a" => 100}

h = { "c" => 300, "a" => 100, "d" => 400, "c" => 300 }

h**.to\_a** # => [["c", 300], ["a", 100], ["d", 400]]

# Les conditions

Comparaisons : == (égalité), != (différence), >, >=, <, <=

**if** a **>** 10

puts "a est supérieur à 10"

**else**

puts "a est inférieur à 10"

**end**

**if** trajet\_minutes **>** 120

puts "Prends un film avec toi"

**elsif** trajet\_minutes < 5

puts "Pas le temps de t'asseoir..."

**else**

puts "Tu as le temps de lire au moins quelques chapitres d'un bouquin !"

**end**

**if** emails.size **==** 1

puts "J'ai un email"

**else**

puts "J'ai plus ou moins d'un email"

**end**

On peut combiner des conditions à l'aide des signes && (pour "ET") et || (pour "OU")

**if** a **!=** 10 **&&** a **!=** 10

puts "ok"

**end**

**if** **(**a **==** 12 **||** a **==** 10**)** **&&** a **!=** "hello"

puts "ok"

**end**

Les booléens sont des variables utiles pour tester si une condition est vraie ou fausse.

mon\_boolean = (a > 10)

**if** mon\_boolean

puts "ok"

**end**

La condition **unless** (l'inverse de if) :

**unless** ARGV.length == 2

puts "Usage: program.rb 23 45"

exit

**end**

Le **conditionnel ?:**

age = 15

puts (13...19).include?(age) **?** "teenager" **:** "not a teenager" # => teenager

age = 23

person = (13...19).include?(age) ? "teenager" : "not a teenager"

puts person # => "not a teenager"

Le **"statement modifier"**

puts "Enrollments will now Stop" **if** participants > 2500 # affiche le msg si condition valide

Le **switch**

year = 2000

leap = **case**

**when** year % 400 == 0 **then** true # => true

**when** year % 100 == 0 **then** false

**else** year % 4 == 0

**end**

puts leap

puts **case** a

**when** 1..5

"It's between 1 and 5"

**when** 6

"It's 6"

**when** String

"You passed a string"

**else**

"You gave me #{a} -- I have no idea what to do with that."

**end**

**case** grade

**when** "A", "B"

puts 'You pretty smart!'

**when** "C", "D"

puts 'You pretty dumb!!'

**else**

puts "You can't even use a computer!"

**end**

# Des boucles avec ou sans les itérateurs

Une boucle permet d'effectuer des actions répétitives de manière simple à l'aide du mot-clé **each**, par exemple pour parcourir un tableau ou encore refaire plusieurs fois la même action.

* **Boucle while** (accepte les mots clés **next** et **break**)

x = 5

**while** x >= 0 # pour une boucle infinie : while 1

puts x

x -= 1 # équivalent avec x = x - 1

**end**

# => ... 5 4 3 2 1 0

* **Boucle for**

x = 5

**for** i **in** 1..x **do**

puts i

**end**

# => ... 1 2 3 4 5

* **Parcourir un tableau**

utilisateurs = ["Alice", "Bob", "John", "Hector"]

**for** user **in** utilisateurs **do** # => ranger chaque élément du tableau dans la variable "user"

**puts** user

**end**

Parcourir un tableau **avec les itérateurs** :

utilisateur**.each** **do** **|**user**|** # => ranger chaque élément du tableau dans la variable "user"

**puts** user

**end**

jours\_ouvres = ["lundi", "mardi", "mercredi", "jeudi", "vendredi"]

i = 5

jours\_ouvres**.each** **do** **|**jour**|**

**if** jour **==** "vendredi"

puts jour + " : Bon weekend !"

**elsif** jour **==**"lundi"

puts jour + " : Bon courage !"

**else**

puts jour + " : Weekend dans **#{i}** jours !"

**end**

i -= 1

**end**

* **Parcourir une hashe**

counts = [ a:5, b:8, c:4, d:18]

counts**.each** **do |**key, value**|** # => ranger chaque clé et valeur du hash dans ces variables **puts** "#{key} #{value"}

**end**

counts**.each** **{ |**key, value**| puts** "#{key} #{value"} **}**

counts**.each\_key** **{ |**key**| puts** key **}**

counts**.each\_value** **{ |**value**| puts** value **}**

* **Boucles de répétition :** **times**

7.**times** **do** # => répéter cette action 7 fois

**puts** "bla"

**end**

7.**times** **do |**i**|** # => i contiendra successivement les valeurs [0, 6]

**puts** "Hello **#{i}**"

i**.times** **do** # deuxième boucle (répétée "i" fois)

puts "World"

**end**

**end**

* **Autres types de boucles de répétition :** **upto**, **downto**, **step**

3**.upto**(5) **do** |x| # x aura successivement les valeurs [3, 4, 5]

x**.upto**(x + 2) do |y| # y aura successivement les valeurs [x, x+1, x+2]

**print** y, " " # affichage sans retour à la ligne

end

**print** "\n"

**end**

# =>

3 4 5

4 5 6

5 6 7

La syntaxe courte de **upto** :

0**.upto**(2) **{**|x| puts "Number: #{x}" **}** # => ... 0 1 2

La syntaxe courte de **downto** :

5**.downto**(3) **{**|x| puts "Number: #{x}" **}** # => ... 5 4 3

La syntaxe courte de **step** :

0**.step**(6, 2) **{**|x| puts "Number: #{x}" **}** # => ... 0 2 4 6

12**.step**(6, -2) **{**|x| puts "Number: #{x}" **}** # => ... 12 10 8 6

# Les méthodes

Une **méthode** (aussi appelée **fonction**) est une série d'actions; définie avec les mots clés **def** et **end**

Une méthode peut prendre zéro, un ou plusieurs paramètres d'entrée.

La convention de Ruby :

- utiliser des parenthèses pour les méthodes avec paramètres

- n'utiliser pas des parenthèses pour les méthodes sans paramètres

Les méthodes renvoient la **dernière expression exécutée** avant la fin de la méthode.

**def** hello # méthode sans argument

'Hello'

**end**

puts hello # => Hello

**def** hello1**(**name**)** # méthode avec un argument entre parenthèses

'Hello ' + name

**end**

puts(hello1('satish')) # => Hello satish

**def** hello2 name2 # méthode avec un argument sans parenthèses

'Hello ' + name2

**end**

puts(hello2 'talim') # => Hello talim

Ruby permet de définir des **valeurs par default** pour les arguments d'une méthode :

**def** mtd(arg1="Dibya", arg2="Shashank", arg3="Shashank")

"#{arg1}, #{arg2}, #{arg3}."

**end**

puts mtd # => Dibya, Shashank, Shashank

puts mtd("ruby") # => ruby, Shashank, Shashank

L'opérateur d'**interpolation** #{...}

Calcule séparément l'expression dans l'opérateur d'interpolation et insère le résultat dans le string.

puts "100 \* 5 = **#{**100 \* 5**}**" # => 100 \* 5 = 500

Ruby permet de définir des méthodes avec un **nombre variable de paramètres** :

**def** foo(**\***my\_string)

my\_string**.inspect**

**end**

puts foo('hello','world') # => ["hello", "world"]

puts foo() # => []

L'argument "**splat**" (\*) peut être positionné dans n'importe quelle position dans la liste des arguments :

**def** opt\_args(a, **\***x, b)

Ruby ne permet pas la définition des plusieurs méthodes ayant le même nom et des paramètres différents (processus de "overloading"), mais il permet de le simuler avec l'argument **splat** :

**def** rectangle(**\*args**)

if args.size < 2 || args.size > 3

puts 'Cette méthode accepte uniquement 2 ou 3 arguments.'

else

if args.size == 2

puts 'Traite la méthode avec 2 arguments'

else

puts 'Traite la méthode avec 3 arguments'

end

end

**end**

**!** En Ruby, les paramètres son envoyés **par référence** à une méthode :

**def** downser(string)

string**.downcase**

**end**

a = "HELLO"

downser(a) # => "hello"

puts a # => "HELLO"

**def** downser(string)

string**.downcase!**

**end**

a = "HELLO"

downer(a) # => "hello"

puts a # => " hello "

**Les méthodes Bang (!)**

Des méthodes qui se finissent par un point d'exclamation.

Elles permettent de modifier la valeur de l'objet appelant.

Les méthodes sans point d'exclamation créent un nouvel objet reflétant les résultats de l'action.

Exemples des méthodes avec ou sans bang :

- sort / sort**!** : pour trier un array

- upcase / upcase**!** : pour les strings

- chomp / chomp**!** : pour les strings

- reverse / reverse**!** : pour les strings et les arrays

**Les méthodes qui se finissent par "?"**

Par convention, les méthodes qui se finissent par un point d'interrogation renvoient la réponse à la question posé par l'invocation de la méthode.

Exemples :

- empty**?** : renvoie "true" si l'array n'a aucun élément

- nonzero**?** : renvoie "nil" si le nombre invoqué est zero, sinon elle renvoie le nombre invoqué

# Les blocks en Ruby

Les blocks sont les morceaux de code qui se trouvent entre accolades **{}** ou entre les mots clés **do..end**

Les blocks peuvent être associés avec les invocations des méthodes.

**def** call\_block # appelle un block avec le mot clé yield

puts 'Start of method'

**yield**

**yield**

puts 'End of method'

**end**

call\_block **{**puts 'In the block'**}**  # block utilisé comme source dans l'appel de la méthode

# =>

Start of method

In the block

In the block

End of method

Si aucun block n'a été fourni et la méthode appelle yield => une exception sera levée

**def** try

if **block\_given?** # vérifie si la méthode a été appelée avec un block

yield

else

puts "no block"

end

**end**

try # => "no block"

try{puts "hello" } # => "hello"

try do puts "hello" end # => "hello"

Les variables d'un block :

x = 10

5.times **do** **|x|**

puts "x inside the block: #{x}" # x est ici une variable locale dans le block

**end**

puts "x outside the block: #{x}"

# =>

x inside the block: 0

x inside the block: 1

x inside the block: 2

x inside the block: 3

x inside the block: 4

x outside the block: 10

# Les Procs

Les blocks ne sont pas des objets, mais ils peuvent être convertis en objets de classe **Proc**.

Ceci est réalisable en appelant la méthode **lambda** de la classe Objet.

Un block créé avec lambda, se comporte comme une méthode Ruby.

La méthode "**call**" de l'objet ainsi créé permet d'invoquer le block.

prc = **lambda{**puts 'Hello' **}**

prc**.call** # => Hello

toast = **lambda** **do**

'Cheers'

**end**

puts toast**.call** # => Cheers

On ne peut pas passer des méthodes comme paramètre à une méthode, et on ne peut pas retourner une méthode. Mais, on peut passer des procs comme paramètre d'une méthode, et on peut également les retourner.

**def** **some\_mtd**(some\_proc)

puts 'Start of mtd'

some\_proc**.call**

puts 'End of mtd'

**end**

say = **lambda{**puts 'Hello' **}**

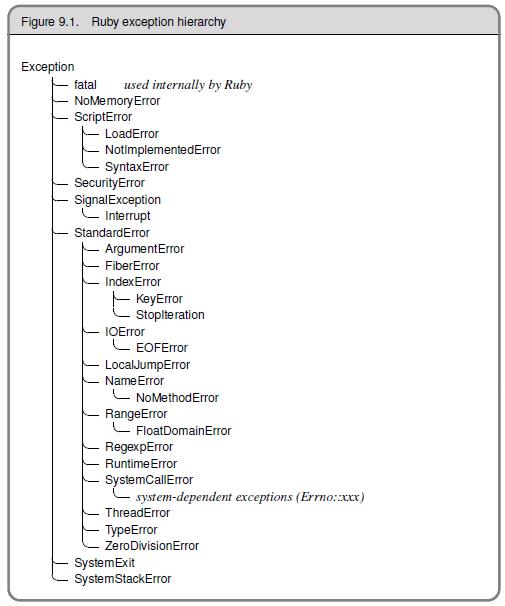
**some\_mtd**(say)

a\_Block = **lambda{**|x| "Hello #{x}!" **}**

puts a\_Block**.call** 'World' # => Hello World!

# Les exceptions

Voici l'hiérarchie des exceptions en Ruby :



Pour soulever des exceptions en Ruby il faut utiliser le mot clé **raise**.

**def** raise\_exception

puts 'I am before the raise.' # => I am before the raise.

**raise** 'An error has occured' # => An error has occured ...

puts 'I am after the raise' # => ne s'execute pas

**end**

raise\_exception

**Exemple :** déclencher une exception d'un type précis

**def** inverse(x)

**raise** **ArgumentError**, 'Argument is not numeric' **unless** x.is\_a? Numeric

1.0 / x

**end**

puts inverse(2) # => 0.5

puts inverse('not a number') # => exception (Argument is not numeric(ArgumentError)

**Exemple :** traiter une exception

**def** raise\_and\_rescue

**begin**

puts 'I am before the raise.'

**raise** 'An error has occured.'

puts 'I am after the raise.' # ne s'exécute jamais

**rescue**

puts 'I am rescued.'

**end**

puts 'I am after the begin block.'

**end**

raise\_and\_rescue

# =>

I am before the raise.

I am rescued.

I am after the begin block.

Pour traiter plusieurs exceptions dans un même bloc begin-end :

**begin**

# -

**rescue** OneTypeOfException

# -

**rescue** AnotherTypeOfException

# -

**else**

# Other exceptions

**end**

Pour interroger le type de l'exception rencontrée :

**begin**

raise 'A test exception.'

**rescue** Exception => e

puts **e.**message

puts **e.**backtrace.inspect

**end**

# =>

A test exception.

["p046excpvar.rb:3"]

# Les classes

Les **classes** sont une notion essentielle de beaucoup de langages de programmation, elles permettent de représenter un type d'objet en modélisant ses caractéristiques : ainsi, les classes sont des "plans" qui servent à créer des objets spécifiques (appelés instances de classes).

Pour connaître la classe d'un objet de Ruby : nom\_de\_l\_objet**.class**

Voici quelques exemples de cette méthode :

10**.class** # => Fixnum (nombre entier) ;

78.5**.class** # => Float (nombre décimal) ;

"Bonjour"**.class** # => String (chaînes de caractères) ;

Pour créer une classe et lui donner des attributs, on utilise les mots-clé **class** et **attr\_accessor** :

**class** Eleve

**attr\_accessor** :prenom, :**nom**, :langage\_prefere

**end**

Attention, toutes les définitions de classe doivent commencer par une majuscule.

Si vous ne respectez pas cela, votre classe ne sera pas prise en compte.

Pour créer une instance de classe, et initialiser ses attributs de l'instance :

bob = Eleve**.new** # crée une nouvelle instance de la classe Eleve sans initialiser ses attributs

bob.prenom = "Bob"

bob.nom = "Lenon"

bob.langage\_prefere = "Ruby !!!"

puts bob**.prenom** + " " + bob**.nom**

L'accès aux attributs à l'intérieur d'une classe :

* attributs précédés par un "**@**" : pour accéder aux **attributs d'instance**
* attributs précédés par "**@@**" : pour accéder aux **attributs de classe**

**L'accès aux attributs** à l'extérieur d'une classe :

Par défaut, on n'a pas accès aux attributs à l'extérieur de la classe;

Solutions : définir soi-même les méthodes *reader* et *getter* ou utiliser les méthodes "accessor" de Ruby

* mot clé **attr\_reader** : donne les droits d'accéder aux variables d'instance
* mot clé **attr\_writer** : donne les droits de modifier les variables d'instance
* mot clé **attr\_accessor** : donne les droits d'accéder et de modifier les variables d'instance

**class** Song

**def** initialize(name)

@name = name

end

**def** **name** # => getter

@name

end

**def** **name=**(name) # => setter

@name = name

end

**end**

song = Song**.new**("Brazil")

puts song**.name** # => Brazil

song**.name =** "Fight"# => new value

puts song**.name** # => Fight

**class** Song

**def** initialize(name, artist)

@name = name

@artist = artist

end

**attr\_reader** :name, :artist

# reader only

**end**

song = Song**.new**("Brazil", "Ivete Sangalo")

puts song**.name**

puts song**.artist**

**Le niveau de protection des méthodes** :

* par défaut, les méthodes définies dans une classe sont **publiques**
* **protected** : méthodes qui peuvent être appelées par les instances de la classe courante et de ses sous-classes (l'accès est limitée à la "famille")
* **private** : méthodes qui peuvent être appelées seulement dans le corps de la classe courante

**Exemples** :

**class** ClassAccess

def m1 # this method is public

end

**protected**

def m2 # this method is protected

end

**private**

def m3 # this method is private

end

**end**

ca.m1

ca.m2 # access violation runtime error

ca.m3 # access violation runtime error

**class** Person

**def** **initialize**(age)

@age = age

end

**def** age

@age

end

**def** compare\_age(c)

if c.age > age

"Older."

else

"Younger or the same age"

end

end

**protected** :age

**end**

chris = Person.new(25)

marcos = Person.new(34)

puts chris.compare\_age(marcos) # => Older

puts chris.age # => exception

**Méthodes aux noms réservés :**

* "**initialize**" : pour initialiser les attributs d'instance

**class** PostSurUnBlog

**attr\_accessor** :titre, :contenu, :auteur

**def** **initialize**(titre, contenu, auteur) # initialise les attributs d'instance

**@**titre = titre

**@**contenu = contenu

**@**auteur = auteur

**end**

**end**

my\_post = PostSurUnBlog**.new**("Mon premier post", "Voici son contenu...", "Moi")

* "**method\_missing**" : pour intercepter les appels des méthodes non-définies

**class** Dummy

def **method\_missing**(m, \*args, &block)

puts "There's no method called #{m} here -- please try again."

**end**

**end**

Dummy.new.anything # => There's no method called anything here -- please try again

**Méthodes propres à l'objet Class :**

Pour voir la **liste complète des méthodes d'une classe** :

Eleve**.instance\_methods**

Pour voir seulement la liste de méthodes qu'on a créées :

Eleve**.instance\_methods**(false)

Pour vérifier si notre classe a accès a une méthode :

Eleve**.respond\_to?**("age") # => false

Eleve**.respond\_to?**("nom\_complet") # => true

**Définir ses propres méthodes**

Définies à l'intérieur d'une classe en commençant par le mot-clé **def** et se termine par **end**.

**class** Eleve

**attr\_accessor** :prenom, :nom, :pays, :langage\_prefere

**def** nom\_complet # méthode sans paramètre

prenom + " " + nom

**end**

**def** aime\_le(langage) # méthode avec paramètre

**if** langage == langage\_prefere

"Oui :)"

**else**

"Non :("

**end**

**end**

**end**

Un **exemple** plus complet de classe et méthodes :

**class** MegaGreeter

**attr\_accessor** :names

**def** **initialize**(names **=** "World") # créer l'objet; **valeur par default** pour la variable "names"

@names = names

**end**

**def** say\_hi

**if** **@**names**.nil?** # vérifie si la variable est définie

puts "..."

**elsif** **@**names**.respond\_to?("each")** # vérifie si la variable est itérable

**@**names**.each**  **{** |name| puts "Hello #{name}!" }

**else**

puts "Hello #{@names}!"

**end**

**end**

**def** say\_bye

**if** @names.nil?

puts "..."

**elsif** **@**names**.respond\_to?("join")** # Join the list elements with commas

puts "Goodbye **#{@names.join(", ")}**. Come back soon!"

**else**

puts "Goodbye #{@names}. Come back soon!"

**end**

**end**

**end**

**if \_\_FILE\_\_ == $0**  # vérifie si c'est le programme principal qui a été lancé

mg = MegaGreeter**.new**

mg.say\_hi

mg.say\_bye

**mg.names =** "Zeke" # change le nom avec "Zeke"

mg.say\_hi

mg.say\_bye

**mg.names =** ["Albert", "Brenda", "Dave"] # change le nom avec une liste de noms

mg.say\_hi

mg.say\_bye

**end**

Le résultat est :

Hello World! ..... Goodbye World. Come back soon!

Hello Zeke! ..... Goodbye Zeke. Come back soon!

Hello Albert! Hello Brenda! Hello Dave! ..... Goodbye Albert, Brenda, Dave. Come back soon!

La condition "**if \_\_FILE\_\_ == $0**" permet d'utiliser le même fichier comme une **librairie**, sans exécuter le code dans ce contexte; mais si le fichier est utilisé comme un exécutable, alors exécute le code.

**L'héritage :**

Pour économiser votre code et le rendre plus maintenable (c'est-à-dire plus facile à mettre à jour), vous pouvez faire **hériter** des classes qui ont des propriétés communes à partir d'une classe-mère.

Pour faire hériter une classe d'une classe-mère, on utilise le symbole **<** dans la définition de la classe :

**class** nom\_de\_la\_classe **<** nom\_de\_la\_classe\_mere

...

**end**

**Exemple** : des classes des animaux

Pour explorer cette notion d'héritage, nous allons partir d'une classe Animal :

**class** Animal

attr\_accessor :nom

**def** **initialize**(nom)

**@**nom = nom # @nom = variable d'instance; accessible à toutes les méthodes de la classe

end

def parler

puts "Je suis un animal qui s'appelle #{nom}"

end

**end**

mon\_chien = Animal**.new**("Bob le chien")

mon\_chat = Animal**.new**("Adeline le chat")

mon\_chat**.parler** # => Je suis un animal qui s'appelle Adeline le chat

mon\_chien**.parler** # => Je suis un animal qui s'appelle Bob le chien

On crée une **classe Chat** qui hérite de la classe Animal, et qui contient une méthode pour miauler

class Chat **< Animal**

def parler # *override* la méthode "parler" de la classe mère

puts "#{nom} : Miaou !"

end

end

On crée une **classe Chien** qui hérite de la classe Animal, et qui contient une méthode pour aboyer

class Chien **< Animal**

def parler

puts "#{nom} : Ouaf !"

end

end

On crée un chien et un chat

mon\_chien = Chien**.new**("Bob le chien")

mon\_chat = Chat**.new**("Adeline le chat")

mon\_chat.parler # => Adeline le chat : Miaou !

mon\_chien.parler # => Bob le chien : Ouaf !

Dans le cas d'un override, on peut également exécuter le bloc de la méthode de la classe mére : avec le mot clé **super**.

**Composer des objets plus complexes :**

La composition consiste à faire interagir des classes qui représentent des objets très différents entre eux.

Pour faire de la composition, on crée **un attribut** dans une classe dans lequel on pourra préciser des informations d'une autre classe.

Par exemple, si on veut répertorier les examens qu'a passés un élève, on va faire interagir une classe **Eleve** avec une classe **Examen** en ajoutant un **attribut examens** dans la classe Eleve. Lorsque l'on instanciera un nouvel élève, cet attribut examens pourra ainsi contenir un tableau d'objets de la classe Examen correspondant aux examens qu'il a passés.

**class** Examen # on crée la classe Examen

**attr\_accessor** :sujet, :note

def initialize(sujet, note)

@sujet = sujet

@note = note

end

**end**

**class** Eleve # on crée la classe Eleve

**attr\_accessor** :nom, :examens # attr "examens" : relie la classe Eleve à la classe Examen

**def** initialize(nom, examens)

@nom = nom

@examens = examens

**end**

**def** moyenne

total = 0

examens**.each** **do** |examen|

total += examen.note # on ajoute examen.note à total

**end**

total /= examens.size

return total

**end**

**end**

# on crée deux examens

**crypto** = **Examen.new**("cryptologie",20)

**maths** = **Examen.new**("maths", 20)

# on les relie à un objet de la classe Eleve

eleve\_brilliant = **Eleve.new**("Alan", **[crypto, maths])**

# on affiche les examens d'Alan

puts "Examens de #{eleve\_brilliant.nom}"

puts "============================"

eleve\_brilliant.examens.each **do** |examen|

puts examen.nom + " : " + examen.note

**end**

puts "Moyenne générale : " + eleve\_brilliant.moyenne

# Les expressions régulières

/regex/ crée un nouvel objet de la classe Regexp.

On peut l'assigner à une variable pour utiliser plusieurs fois la même expression régulière, ou on peut utiliser le littéral regex directement.

Pour tester si un regex correspond à un string, on peut utiliser

* l'opérateur **=~** : retourne l'index de la position où le regex à été trouvé ou nil
* la méthode **match()**: retourne un objet de type MatchData ou nil

puts "success" if subject **=~** /regex/

puts "success" if /regex/**.match**(subject)

m1 = "The future is Ruby" **=~** /Ruby/ # => 14 (l'index ou il a été trouvé)

m2 = /Ruby/**.match**("The future is Ruby") # objet de type MatchData

Le résultat des tests sur les expressions régulières est stocké temporairement (dans la méthode courante ou jusqu'au prochain appel d'un regex) dans une variable spéciale **$~** .

Autres variables spéciales read-only :

* **$&** : stocke le texte qui match l'expression régulière;
* **$1**, **$2**, ... : stocke le texte qui match le premier groupe, le seconde groupe, etc;
* **$+** : stocke le texte correspondant au groupe ayant le numéro de capture le plus élevé (qui a effectivement participé au match)
* **$`**, **$'** : stockent le texte dans la chaîne sujet vers la gauche et vers la droite du match du regex

Les modifiants associés aux expressions régulières :

* **/i** : insensible à la case des lettres
* **/m** : le "." identifie également les retours à la ligne
* **/x** : ignorer les espaces blancs entre les token regex
* **/o** : force toute substitution de type # {...} à s'exécuter une seule fois dans un regex. Dans le cas contraire, les substitutions seront effectuées à chaque fois que le littéral génère un objet Regexp.

Caractères utiles dans les expressions régulières :

* **\?** : considère les caractères suivants tel quels

(ignore les caractères spéciaux : ^, $, ?, ., /, \, [, ], {, }, (, ), +, \*

* **.** : n'importe quel caractère
* **[]** : caractères de classes; ex: /[dr]ejected/ , /[a-z]/, /[0-9]/, /[A-Fa-f0-9]/, /[**^**A-Fa-f0-9]/ (négation)
* /**\d**/ : n'importe quel chiffre; /**\D**/ : tout sauf un chiffre (la négation de \d)

/**\s**/ : n'importe quel "espace"; /**\S**/ sa négation

/**\w**/ : n'importe quel chiffre, lettre ou \_; /**\W**/ sa négation

Autres méthodes regex:

* **chercher et remplacer** (seulement la première occurrence "**sub**", toutes les occurrences "**gsub**"):

result = subject**.gsub**(/before/, "after")

result = subject**.gsub**(/**(**before**)**/, "**\1** than after") # capture les groups

* **splitting & collecting** :

**scan** = récupère tous les matches de regex dans un array;

**split** = sépare le string en fonction du délimiteur

subject = "that was before"

myarray = subject**.scan**(/\w/) # => ['t', 'h', 'a', 't', 'w', 'a', 's', 'b', 'e', 'f', 'o', 'r', e']

myarray = subject**.split**(/\s/) # => ['"that", "was", "before"]

**Exemple** d'objet MatchData :

string = "My phone number is (123) 555-1234."

phone\_re = /\((\d{3})\)\s+(\d{3})-(\d{4})/

m = phone\_re.match(string)

if m

print "The entire part of the string that matched: "

puts m[0] # : (123) 555-1234

puts "The three captures: "

m**.captures**.**each\_with\_index** do |match, index|

puts "Capture #{index}: #{match}"

end

puts "Here's another way to get at the first capture:"

puts "Capture #1: #{m[1]}"

end

# =>

The three captures:

Capture 0: 123

Capture 1: 555

Capture 2: 1234

Here's another way to get at the first capture:

Capture #1: 123

# Les logs en Ruby

La classe **Logger** dans la librairie standard de Ruby permet de sauvegarder des messages de log dans des fichiers ou des streams.

Il est possible de changer le niveau de verbosité des logs d'un programme; d'habitude le niveau est réglé sur **Logger::INFO** ou **Logger::WARN**.

Du moins au plus sévère, les méthodes d'instance sont :

**Logger.debug**, **Logger.info**, **Logger.warn**, **Logger.error et Logger.fatal**.

Exemple : dans le setup ERROR, si le program ne peut pas résoudre un problème, il enregistre l'exception plutôt qu'arrêter l'exécution du programme et attend qu'un administrateur s'en occupe.

Si les logs sont stockés dans un fichier, Logger peut remplacer le fichier quand il devient trop grand ou une fois des temps en temps :

require '**logger**'

**Logger.new**('this\_month.log', 'monthly') # => Keep data for the current month only

**Logger.new**('application.log', 20, 'daily') # => Keep data for the past 20 days + today

**Logger.new**('application.log', 0, 100 \* 1024 \* 1024) # => Restart when the log exceeds 100 MB

**Exemple d'utilisation :**

require 'logger'

**$LOG** = **Logger.new**('log\_file.log', 'monthly')

**def** divide(numerator, denominator)

$LOG.debug("Numerator: #{numerator}, denominator #{denominator}")

**begin**

result = numerator / denominator

**rescue** Exception => e

$LOG.error "Error in division!: #{e}"

result = nil

**end**

return result

**end**

divide(10, 2)

divide(10, 0)

#=> # Logfile created on Tue Mar 18 17:09:29 +0530 2008 by /

D, [2008-03-18T17:09:29.216000 #2020] DEBUG -- : Numerator: 10, denominator 2

#=> # Logfile created on Tue Mar 18 17:09:29 +0530 2008 by /

D, [2008-03-18T17:09:29.216000 #2020] DEBUG -- : Numerator: 10, denominator 2

D, [2008-03-18T17:13:50.044000 #2820] DEBUG -- : Numerator: 10, denominator 0

E, [2008-03-18T17:13:50.044000 #2820] ERROR -- : Error in division!: divided by 0

Pour changer le niveau de logs il faut assigner la constante correspondante :

**$LOG.level** = Logger::ERROR

# => E, [2008-03-18T17:15:59.919000 #2624] ERROR -- : Error in division!: divided by 0

# Le temps en Ruby

t = **Time.now** # => 2016-06-06 14:52:26 +0200

# to get day, month and year with century, also hour, minute and second

puts t**.strftime**("%d/%m/%Y %H:%M:%S") # => 06/06/2016 14:52:26

# You can use the upper case A and B to get the full # name of the weekday and month

puts t.strftime("%A") # => Monday

puts t.strftime("%B") # => June

# You can use the lower case a and b to get the abbreviated name of the weekday and month

puts t.strftime("%a") # => Mon

puts t.strftime("%b") # => Jun

# 24 hour clock and Time zone name

puts t.strftime("at %H:%M %Z") # => at 14:52 Romance Daylight Time

# Les objet immutables : freeze & frozen

La méthode **freeze** de la classe Objet nous empêche de changer un objet en le transformant dans une constante. Une tentative de modifier un objet qui a été "figé" va soulever une exception de type TypeError.

str = 'A simple string. '

str**.freeze**

**begin**

str << 'An attempt to modify.'

**rescue** => err

puts "#{err.class} #{err}"

**end**

# => TypeError can't modify frozen string

La méthode **freeze** agit sur une référence d'objet, et non sur une variable.

Cela signifie qu'une opération résultant en un nouvel objet sera exécutée :

str = 'Original string - '

str**.freeze**

str **+=** 'attachment'

puts str # => Output is - Original string - attachment

Pour vérifier si un objet est "frozen" ou pas :

a = b = 'Original String'

b.freeze

puts a**.frozen?** # true

puts b**.frozen?** # true

a = 'New String'

puts a**.frozen?** # false

puts b**.frozen?** # true

# Les modules

Les modules en Ruby sont similaires aux classes, dans le sens qu'ils contiennent une collection des méthodes, constantes et autres définitions des modules et des classes.

Les modules sont des bons endroits de stockage.

# mytrig.rb

**module** Trig

PI = 3.1416

**def** Trig.sin(x)

# ...

**end**

**def** Trig.cos(x)

# ...

**end**

**end**

# mymoral.rb

**module** Moral

VERY\_BAD = 0

BAD = 1

**def** Moral.sin(badness)

# ...

**end**

**end**

# usemodule.rb

**require\_relative** 'mytrig' # le nom du fichier; le chemin en relatif

**require\_relative** 'mymoral'

**Trig**.sin(**Trig**::PI/4)

**Moral**.sin(**Moral**::VERY\_BAD)

# La sérialisation des objets (*marshaling*)

Ruby nous permet de sauvegarder des objets dans des fichiers et de re-charger les objets sauvegardés dans des fichiers.

# gamecharacters.rb

**class** GameCharacter

attr\_reader :power, :type, :weapons

def initialize(power, type, weapons)

@power = power

@type = type

@weapons = weapons

end

**end**

# dumpgc.rb

**require\_relative** 'gamecharacters'

gc = GameCharacter.new(120, 'Magician', ['spells', 'invisibility'])

puts "#{gc.power} #{gc.type}"

gc.weapons.each do |w|

puts w

end

**File.open**('game', 'w+') do |f|

**Marshal.dump**(gc, f)

end

# loadgc.rb

**require\_relative** 'gamecharacters'

**File.open**('game') do |f|

@gc = **Marshal.load**(f)

end

puts "#{@gc.power} #{@gc.type}"

@gc.weapons.each do |w|

puts w

end