**Lancez-vous dans la programmation avec Ruby**

Ruby est un langage de programmation qui a été créé par un informaticien japonais dans les années 1990, avec un objectif en tête : faciliter la vie des développeurs avec une syntaxe et des outils simples et agréables à utiliser.

**Installez vos outils**

Nous en avons besoin de deux :

Un éditeur de texte pour écrire le code Ruby

La console pour exécuter le code Ruby

**Installez Ruby**

Sur Mac OS il est installé de base, dans le terminal il suffit de taper : ruby –v afin d’avoir la version.

Sur Linux, si c’est une Debian faire : apt-get install ruby-full

Si c’est une CentOS faire : yum install ruby

Sur Windows installer le .exe

**Travail dans la console avec Interactive Ruby (IRB)**

Pour exécuter du code Ruby directement dans votre console,  il vous faut utiliser la **console Ruby appelée IRB (Interactive Ruby)**  en tapant la commande suivante :  irb

Pour quitter il faut taper : exit ou quit

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

age = 15

nom = "Qeyss"

On peut regrouper des variables contenant des chaînes de caractères à l'aide de l'opérateur + , on dit qu'on fait de la **concaténation**. Par exemple la variable student :

student = nom + " est un étudiant en Master "

Contiendra la phrase : " Qeyss est un étudiant en Master "

**Rangez vos données dans des tableaux et des hashes.**

Vous pouvez regrouper vos données  de deux façons :

* dans des **tableaux** : pour ordonner une série d'éléments à l'aide d'index ;
* dans des **tables de hashage** : pour retrouver des éléments facilement à l'aide de clés.

**Les tableaux**

* Les tableaux permettent de ranger des données de façon ordonnée que l'on retrouve à l'aide d'un **index**.
* Pour créer un tableau, on utilise les crochets, par exemple :

mes\_escales\_tour\_du\_monde = [ "Paris", "Toronto", "NYC", "Rio", "Sydney", "Hong-Kong", "Berlin" ]

On peut appliquer des **méthodes** à une variable contenant un tableau de la façon suivante :

nom\_du\_tableau.nom\_de\_la\_methode

La method *size* permet de connaître le nombre d'éléments d'un tableau. Par exemple, pour vérifier combien d'escales je vais faire dans mon tour du monde, je peux taper :

mes\_escales\_tour\_du\_monde.size

* Pour accéder à un élément dans un tableau,  on précise la position (ou index) de l'élément entre crochets. Par exemple,

mes\_escales\_tour\_du\_monde[2]

* Pour ajouter un élément dans un tableau, on utilise deux chevrons <<. Si je veux prolonger mon tour du monde à Londres, je taperai donc :

mes\_escales\_tour\_du\_monde << "Londres"

* Pour modifier l'élément d'un tableau, on lui affecte la valeur souhaitée avec le signe =. Je peux donc remplacer Berlin par Barcelone si je parle mieux l'espagnol que l'allemand :

mes\_escales\_tour\_du\_monde[6] = "Barcelone"

**Les tables de hashage**

* Les tables de hashage (ou 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 cette table est associée à une clé qui permettra de la retrouver :

table\_de\_hashage = { clé\_1: valeur\_1, clé\_2: valeur\_2, ... }

Par exemple, pour enregistrer le nombre de jours à passer dans chaque escale, je peux créer une table de hashage jours\_voyage :

jours\_voyage = { paris: 0, toronto: 7, nyc: 3 }

* Pour accéder à la valeur d'un élément dans un hash, on précise sa clé entre crochets:

jours\_voyage[:toronto]

qui renverra 7

* Pour ajouter ou modifier un élément dans un hash, on utilise l'opérateur = Par exemple en ajoutant :

jours\_voyage[:rio] = 5

Mon hash jours\_voyage contiendra la durée de mon séjour à Rio en plus :

**{ paris : 0, toronto : 7, nyc : 3, rio : 5 }**

**Travaillez avec des fichiers Ruby**

On peut aussi travailler dans un fichier créé à partir d’un éditeur de texte.

Un fichier Ruby doit avoir l’extension .rb

* Pour exécuter un fichier Ruby à partir de la console, placez votre console dans le répertoire contenant le fichier. Par exemple si tu as un fichier voyages.rb dossier ruby dans Documents tu feras cd Documents/ruby puis taper :

ruby voyages.rb

**Les boucles**

* 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.
* Exemple de boucle pour parcourir un tableau :

villes = ["Alger", "Dakar", "Tokyo", "Amman", "Torcy"]

villes.each do |villes|

puts villes

end

Autre exemple :

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

Autre syntaxe :

11.times do

puts " Salam Qeyss"

end

**Les conditions**

On peut ajouter des conditions pour réaliser différentes actions dans Ruby selon le cas à l'aide des mots-clés if, elsif et else :

trajet\_minutes = 5

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

# Guest number

puts " Entrer un chiffre "

secret = 23

number = gets.chomp.to\_i

if number > secret

puts " C'est plus petit"

elsif number < secret

puts " C'est plus grand"

else

puts " Bravo "

end

puts " Entrer un chiffre "

secret = 23

unless chiffre == secret # à moins que le chiffre soit égal à 23

puts " Vous n’avez pas trouvé le chiffre magique"

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

puts " Entrer un chiffre "

number = gets.chomp.to\_i

if number > 10 && number < 15

puts " le nombre est entre 11 et 14"

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

train = true

if train

puts "Prenons le train !"

else

puts "Prenons la voiture !"

end

Pour vérifier une condition d'égalité, il faut utiliser deux signes égal (==). Faites attention à ne pas confondre avec le signe égal tout seul (=) qui sert à affecter une valeur à une variable :

age = 18 #range la valeur 18 dans la variable age et renvoie donc 18

age == 18 #booléen qui renvoie True si age vaut 18, False sinon

Le ! avant une condition l’inverse

If chiffre.even # si le chiffre est pair

If !chiffre.even # si le chiffre est impair

# mot inversé

puts " Entrer un mot "

mot = gets.chomp.downcase

mot\_inverse = mot.reverse

if mot == mot\_inverse

puts " C'est un palyndrome "

else

puts " Ce n'est pas un palyndrome "

end

Les méthodes

Ce sont des fonctions que l’on applique sur des objets à l’aide d’un point visible ou invisible.

La méthode gets

Elle permet de récupérer une valeur tapée au clavier, ex :

puts "Quel est ton nom ?"

nom = gets

puts " Ton nom est " + nom

La méthode gets prend en compte l’appui de la touche entrée et donc tout ce qui sera mis après sera forcément à la ligne. Pour éviter cela on ajoute à la suite de gets, .chomp

Ex :

puts "Quel âge as-tu ?"

age = gets.chomp

puts " Tu as donc " + age + "ans."

Convertir un string en integer :

"55".to\_i

puts "Quel est ton âge ? "

age = gets.chomp.to\_i

Convertir un integer en string :

6.to\_s

La méthode lenght

Elle affiche la longueur d’un string

Puts "paradis".length

7

La méthode upcase

Pour mettre en majuscule

qeyss.upcase

La méthode downcase

Pour mettre en miniscule

ANTHO.downcase

La méthode reverse :

Va retourner à l’envers le string

" Hi brothers".reverse

srehtorb ih

Méthode event

Afin de savoir si un nombre est pair ou pas.

Le ? après une méthode demande un retour booléen

2.even ?

true

On peut mettre une variable dans un string soit par une concaténation ou bien par cette syntaxe #{} , ex :

nom = "Ali"

puts " Bonjour #{nom}, comment ça va ?."

Cela fonctionne qu’avec des double quote.

La POO qu’est-ce que c’est ?

C’est la modélisation en programmation des objets de la vie courante ainsi que leurs interactions.

Chaque objet à ses caractéristiques propres

**Les classes**

Les classes sont des « plans » ou « modèles »qui servent à créer des objets spécifiques (appelés instances de classes).

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

class Eleve

attr\_accessor :prenom, :nom

end

Le nom d’une classe doit obligatoirement commencer par une majuscule.

Pour créer une instance de classe, on utilise la méthode new selon la syntaxe suivante :nom\_de\_l\_objet = nom\_de\_la\_classe.new. Puis on spécifie les attributs de l'instance comme ceci :

one = Eleve.new

one.prenom = "Zak"

one.nom = "Madani"

puts one.prenom + "  " + one.nom

Pour connaître la classe d'un objet de Ruby, il existe une méthode class qui s'appelle ainsi : nom\_de\_l\_objet.class  
Voici quelques exemples de cette méthode :

10.class renvoie Fixnum ( nombre entier)

78.5.class renvoie Float

" Bonjour ".class renvoie String, etc.

Toutes ces choses sont des objets des classes Fixnum, Float et String..

**Les méthodes**

* Une méthode (aussi appelée fonction) est une série d'actions. Elle se définit à l'intérieur d'une classe en commençant par le mot-clé def et se termine par end.
* Une méthode peut prendre un ou plusieurs paramètres d'entrée, ou aucun paramètre. Exemples de méthodes dans une classe :

Méthode sans paramètre

class Student

attr\_accessor :prenom, :nom

def nomComplet

prenom + " " + nom

end

end

one = Student.new

one.prenom = " Qeyss"

one.nom = " Madani"

puts one.nomComplet

#méthode avec paramètre

def aime\_le(langage)

if langage == langage\_prefere

"Oui :)"

else

"Non :("

end

end

end

On applique une méthode à une instance de classe de la façon suivante : nom\_de\_l\_objet.nom\_de\_la\_methode. Si on crée une instance de la classe Eleve ci-dessus, on pourra ainsi écrire :

jc = Eleve.new

jc.prenom = "John"

jc.nom = "Cleese"

jc.langage\_prefere = "Python"

puts "Est-ce que " + jc.nom\_complet + " aime le Ruby ?\n" + jc.aime\_le("Ruby")

Résultat:

Est-ce que John Cleese aime le Ruby ?

Non :(

**Manipulez une instance de classe**

Ma correction du TP :

class User

attr\_accessor :nom, :amis

def est\_ami\_avec(nom)

amis.each do |ami|

if ami.nom == nom

puts " oui, c'est son ami"

end

end

end

end

Qeyss = User.new

Qeyss.nom = "Qeyss"

Samia = User.new

Samia.nom = "Samia"

Nunu = User.new

Nunu.nom = "Nunu"

Qeyss.amis = [Samia,Nunu]

Samia.amis = [Qeyss,Nunu]

Nunu.amis = [Qeyss,Samia,Nunu]

puts " Qeyss a " + Qeyss.amis.size.to\_s + " amis" # Il faut convertir le integer en string

puts Qeyss.est\_ami\_avec("Nunu")

Regroupez des classes avec l’héritage

Un peu de maths

Pour faire puissance on écrit ainsi :

5\*\*2

Pour obtenir un nombre aléatoire on écrit

Rand() 🡪 ça va renvoyer un nombre entre 0 et 1 ( exclu)

Si je veux un nombre entre 0 et 9 il faut écrire ainsi :

puts rand(10)

ss