Licence 3 d'Informatique — Langages de *script* — TP 7

Début : jeudi 23 novembre 2023

Cette nouvelle série d'exercices en Ruby va mêler quelques échauffements de base utilisant des procédures de tri et quelques exercices plus ambitieux, inspirés par des applications réelles et situés à la lisière entre des exemples de programmation et une problématique inspirée des tâches de maintenances liées aux systèmes d'exploitation. Pour ne pas ralentir la lecture de l'énoncé, les descriptions des méthodes utiles des classes `File` et du module `Find` ont été regroupées dans le § 5. Vous aurez aussi besoin du fichier d'amorce for-lc-7.rb, que comme de coutume vous pourrez trouver sur le serveur moodle, dans le répertoire « Travaux pratiques Ruby 2023 > for-lc-7 », qui est une partie du cours « Programmation fonctionnelle, scripts et XML ». Vous y trouverez aussi le fichier source Ruby blocks-etc.rb et le fichier exécutable `mk-sandbox`, dont les utilités respectives vous seront expliquées aux §§ 3.1 & 4, ainsi que le présent fichier sous forme numérique : lc-7.pdf.

1 *Intrada :* tris de tableaux

Nous rappelons l'existence des méthodes `sort` et `sort!` de Ruby, utilisées pour trier un tableau [1], de classe `Array`. Par défaut, la relation d'ordre utilisée est `<=>`, prédéfinie pour les nombres d'une part et les chaînes de caractères d'autre part — classes `Float`, `Integer` et `String` — et analogue aux méthodes `compareTo` du langage Java [2] : c'est-à-dire que l'évaluation de l'expression `x <=> y` — où `x` et `y` sont tous deux des nombres ou tous deux des chaînes de caractères [3] — retourne `0` si `x` est égal à `y`, `-1` (resp. `1`) si `x` est strictement inférieur (resp. supérieur) à `y`. Pour utiliser une autre relation d'ordre avec les méthodes `sort` et `sort!` de Ruby, présenter à ces méthodes un bloc à deux arguments formels réalisant une relation d'ordre selon le même principe. Rappelons que dans le contexte d'une procédure de tri, la valeur retournée par une telle fonction appliquée à deux éléments `x` et `y` est `0` si `x` et `y` sont égaux, `-1` (resp. `1`) si `x` est à *gauche* (resp. à *droite*) de `y`.

$\Longrightarrow$ Donner une méthode `sort_both_integers_strings`, telle que l'évaluation de l'expression :

`sort_both_integers_strings(`a_0`)`

1. Bien voir que la méthode `sort` retourne un *nouveau* tableau, formé des mêmes éléments que le tableau d'origine, mais triés, alors que la méthode `sort!` *modifie physiquement* le tableau d'origine pour en construire une version triée qui est alors retournée.

2. Des méthodes analogues — `CompareTo` — existent aussi dans le langage C#.

3. Nous ne nous préoccuperons pas ici des autres cas de figure où l'opérateur prédéfini `<=>` de Ruby peut s'appliquer, tout en rappelant qu'il retourne un résultat différent de la valeur `nil` pour deux objets de *même classe*, ce qui exclut — par exemple — un cas de figure comme « `false <=> true` » (rappelons que les classes respectives de ces deux objets sont `FalseClass` et `TrueClass`).

— où $\texttt{a}_0$ est un objet de classe `Array` dont les éléments sont soit des entiers relatifs (de classe `Integer`), soit des chaînes de caractères (de classe `String`) — retourne un tableau composé des éléments du tableau $\texttt{a}_0$, mais triés en utilisant l'ordre suivant :

— apparaissent d'abord les entiers relatifs, ainsi que les chaînes convertibles en un entier relatif, ces éléments étant triés suivant l'ordre numérique croissant ;

— puis les autres chaînes de caractères, disposées suivant l'ordre lexicographique croissant, utilisé dans un dictionnaire.

Prendre garde aux points suivants :

— la méthode qui convertit une chaîne de caractères en un entier relatif est `to_i` :

```
'  -273  '.to_i ⟹ -273
```

— mais cette méthode retourne le nombre entier zéro si la chaîne n'est pas convertible en un entier relatif :

```
'Miam-miam bouffe-bouffe !'.to_i ⟹ 0
```

et présente quelques comportements curieux si un nombre est suivi par d'autres caractères [4] :

```
'273  15'.to_i ⟹ 273
```

aussi le plus sûr moyen de détecter qu'une chaîne est *vraiment* convertible en un nombre est l'utilisation d'une expression régulière (de classe `Regexp`). Ne pas oublier d'y utiliser les marqueurs de début (`\A`) et de fin (`\z`) de chaîne ; pensez aussi à permettre l'occurrence de caractères d'espacement (`\s`) avant et après le nombre.

2 *Adagio lugubre* : découverte de la classe `File`

Dans toute la suite de ce § 2, ainsi que dans le § 4, « $\texttt{path}_0$ » désignera une chaîne de caractères représentant l'accès à un *directory*, c'est-à-dire un *pathname* selon la terminologie anglo-saxonne. Les méthodes de la classe `String` de Ruby peuvent s'appliquer à un tel objet.

⟹ Définir une fonction `look_4_pdf_files`, telle que l'évaluation de l'expression :

```
look_4_pdf_files(path₀)
```

retourne un tableau de toutes les adresses (*paths*) des fichiers en format PDF [5] présents dans l'arborescence de ce *directory*, c'est-à-dire directement présents dans le *directory* $\texttt{path}_0$ ou appartenant à un sous-répertoire — à quelque niveau que ce soit — de $\texttt{path}_0$. Peu importe l'ordre des éléments dans le résultat. Pour réaliser cette fonctionnalité, vous aurez besoin de la méthode `find` du module `Find`, décrite au § 5.2.

4. Les mêmes inconvénients existent avec la méthode `to_f` de conversion d'une chaîne de caractères en un nombre flottant de classe `Float`.

5. ***P**ortable **D**ocument **F**ormat*, le format d'Adobe.

```
class File
  #
  def self.whole_size(path)
    if File.directory?(path)
      size_accumulator = 0
      Dir.glob(File.join(path,'**','*')) do |path_0|
        size_accumulator += File.size(path_0) if File.file?(path_0)
      end
      size_accumulator
    else
      File.size(path)
    end
  end
  #
end
```

Figure 1 : Fichier for-lc-7.rb (début).

Indication Penser à utiliser une expression régulière pour chercher le suffixe d'un fichier en format PDF.

Vérifiez maintenant que la méthode `File.size`, lorsqu'elle est appliquée à un nom de répertoire, retourne la taille de la structure d'accès aux fichiers et sous-répertoires correspondants, plutôt que la taille cumulée de tous ses composants[6]. Pour cette fonctionnalité de calcul de la taille totale d'un répertoire, le fichier for-lc-7.rb vous offre la méthode `File.whole_size`, reproduite dans la figure 1.

$\Longrightarrow$ Écrire une fonction `sort_by_size`, telle que l'évaluation de l'expression :

$$\texttt{sort\_by\_size}(path_0)$$

retourne le tableau de tous les sous-répertoires [7], trié par ordre croissant suivant la place occupée par ces divers sous-répertoires.

3 *Intermezzo*

3.1 Découverte des exceptions

Vous pouvez à nouveau considérer le fichier blocks-etc.rb qui vous a été démontré en cours. Que sont les noms d'exceptions, tels que `ZeroDivisionError`, `RuntimeError`, `StandardError`, *etc.* ? et quelle est leur organisation ? (Penser à utiliser les méthodes `class` et `superclass`.) Puis faites quelques essais qui vous montreront que la clause « `rescue Exception` » permet de récupérer toute erreur. (Vous pouvez par exemple essayer l'application d'une fonction qui n'existe pas.)

6. Cette vérification est importante, surtout si vous travaillez sur votre propre matériel.
7. . . . au sens large, c'est-à-dire que $path_0$ appartient lui-même à ce tableau.

```
$s = "l'été"                          # Exemple d'une chaîne comportant des lettres accentuées.
$s.encoding              ⟹ ??  # Codage utilisé pour la chaîne $s.
Encoding.class           ⟹ ??  # Est-ce que la classe Encoding est bel et bien disponible ?
Encoding.find('UTF-8')   ⟹ ??  # Est-ce que le résultat est bien différent de la valeur nil,
                                      # c'est-à-dire que le codage UTF-8 est bel et bien disponible ?
Encoding.find('locale')  ⟹ ??  # Quel est le codage utilisé par défaut sur votre installation ?
```

Figure 2 : Fichier for-lc-7.rb (suite).

⟹ Vous pouvez vous apercevoir que l'accès à un élément d'un tableau dont on donne l'indice, s'il est effectué au moyen de la méthode `at` de la classe `Array` — ou de la notation entre crochets droits — retourne `nil` si l'élément correspondant n'existe pas. C'est assez simple de s'en rendre compte :

```
[][0] ⟹ nil          [].at(0) ⟹ nil
```

Ajouter une méthode `protected_i` à la classe `Array`, admettant un argument entier et retournant l'élément de rang correspondant dans le tableau s'il existe, et déclenche l'exception `OutOfBounds` sinon. On pourra aussi admettre des indices négatifs, comptés à partir de la fin du tableau, comme le fait la méthode prédéfinie `at` :

```
[23,11,2023][-1] ⟹ 2023          [23,11,2023].at(-3) ⟹ 23
```

Exemples :

```
[23,11,2023].protected_i(2)     ⟹ 2023
[23,11,2023].protected_i(-1)    ⟹ 2023
[23,11,2023].protected_i(2023)  ⟹ OutOfBounds raised
[23,11,2023].protected_i(-2022) ⟹ OutOfBounds raised
```

⟹ Puis donner une fonction `outofboundsnb`, qui appliquée à un tableau `a` et à un tableau d'entiers *`index_a`*, accède aux éléments du tableau `a` en considérant les indices donnés dans le tableau *`index_a`*. Ces accès utiliseront la méthode `protected_i` précédente, et la fonction `outofboundsnb` retourne le nombre de fois où l'exception `OutOfBounds` a été déclenchée. Exemples :

```
outofboundsnb([23,11,2023],[2,1,-1])        ⟹ 0
outofboundsnb([23,11,2023],[2023,1,2])      ⟹ 1
outofboundsnb([23,11,2023],[2023,0,-2021]) ⟹ 2
```

3.2 Quelques notions complémentaires sur les chaînes de caractères

Considérez les évaluations de la figure 2 — présentes dans le fichier for-lc-7.rb —, visant à vous introduire à l'utilisation d'Unicode pour le traitement des chaînes de caractères [8] et à vérifier que votre version de Ruby utilise le codage UTF [9]-8 pour les chaînes de caractères. Si votre fichier source commence par la ligne suivante :

8. Là aussi, cette vérification est importante si vous travaillez sur votre propre matériel.
9. ***U**nicode **T**ransformation **F**ormat.*

```
class String
  #
  ACCENTED_LETTER_MAPPING = {
    'A' => [192,193,194,195,196,197],'C' => [199],'E' => [200,201,202,203],
    'I' => [204,205,206,207],'N' => [209],'O' => [210,211,212,213,214,216],'
    U' => [217,218,219,220],'Y' => [221],'a' => [224,225,226,227,228,229,230],
    'c' => [231],'e' => [232,233,234,235],'i' => [236,237,238,239],'n' => [241],
    'o' => [242,243,244,245,246,248],'u' => [249,250,251,252],'y' => [253,255],
    'AE' => [306],'ae' => [346],'OE' => [188],'oe' => [189]
  }
  #
  def removeaccents
    str = String.new(self)
    ACCENTED_LETTER_MAPPING.each do |letter,accents|
      packed = accents.pack('U*')
      rxp = Regexp.new("[#{packed}]",nil)
      str.gsub!(rxp,letter)
    end
    str
  end
  #
end
```

Figure 3 : Fichier for-lc-7.rb (fin).

```
# coding: utf-8
```

— c'est la déclaration par défaut pour les versions modernes de Ruby —, les chaînes de caractères présentes dans le fichier sont lues suivant ce codage. Puis considérez les définitions de la figure 3 — également jointes au fichier for-lc-7.rb — et plus particulièrement la méthode `removeaccents` ajoutée à la classe `String`[10]. Voyez ce que donne le résultat de l'évaluation :

```
$s.removeaccents ⟹ ??
```

4 *Allegro nervoso :* sauvegardes de répertoires

Nous désirons sauvegarder tous les répertoires — à quelque niveau que ce soit — d'un *directory* $\texttt{path}_0$, avec les contraintes suivantes[11] :

— un sous-répertoire doit être *renommé* si son nom contient des caractères d'espacement, qui sont à remplacer par le caractère de soulignement (« _ »), ou des lettres accentuées, en

10. Cette méthode `removeaccents` utilise une table de hachage `ACCENTED_LETTER_MAPPING` (de classe `Hash`), ainsi que la méthode `pack` des tableaux (la classe `Array`), hors du programme de l'unité *Programmation fonctionnelle et scripts*. Mais n'hésitez pas à demander à votre gentil animateur si vous voulez en savoir plus à son sujet.

11. Les exercices de ce § 4 tirent leur origine de difficultés qui sont réellement survenues dans une entreprise lors de l'installation de procédures de sauvegarde. Comme on le voit parfois au cinéma, « ce film s'inspire d'éléments réels ».

dehors du codage ASCII [12] traditionnel, auquel cas ces accents doivent être supprimés et la lettre accentuée originale doit être restituée sans accent ; si le nom `d` obtenu correspond à un autre sous-répertoire déjà existant ou à un autre fichier déjà existant, alors les renommages `d-1`, `d-2`, ... — où « `-` » est bel et bien le trait d'union — sont à essayer jusqu'à trouver un nom qui ne correspond pas à une ressource existante ;

— si le *pathname* absolu d'un sous-répertoire contient strictement plus de 200 caractères, le répertoire correspondant sera ignoré durant l'opération de sauvegarde.

Dans un premier temps, nous allons simuler ces opérations et montrer ce qu'elles impliqueraient en situation réelle. Puis nous réaliserons effectivement les renommages.

$\Longrightarrow$ Donner une fonction `superscan`, telle que l'évaluation de l'expression `superscan(`$path_0$`)` retourne un tableau comprenant d'abord des tableaux à deux éléments répertoriant tous les renommages à opérer — c'est l'ancien nom qui est donné d'abord, suivi du nouveau —, ensuite les noms des sous-répertoires ignorés durant l'opération de sauvegarde :

```
superscan(...)  ⟹ [["...","..."],["...","..."],...,"...","...",...]
```

Indication Ruby fournit la méthode `ascii_only?` pour savoir si les caractères d'une chaîne appartiennent tous au codage ASCII original. On pourra bien évidemment utiliser la méthode `removeaccents` de la classe `String` (*cf.* figure 3).

$\Longrightarrow$ Puis définir une fonction `scan_and_rename`, telle que l'évaluation de l'expression :

`scan_and_rename(`$path_0$`)`

réalise réellement les opérations de renommages décrites ci-dessus, et retourne un tableau comprenant les noms des sous-répertoires ignorés durant l'opération de sauvegarde, ou pour lesquels l'opération de renommage a échoué. Pour ce faire, vous aurez besoin des formes de traitement d'exceptions, afin de récupérer une exception consécutive à un renommage qui vient d'échouer.

Indication Pour vos essais, il vous faut utiliser un répertoire vide, quelque part dans l'arborescence de vos fichiers ; une bonne place, si vous utilisez le système d'exploitation `Linux`, étant le répertoire /tmp. Créez-y un sous-répertoire for-lc-7-ruby. Puis installez-y le fichier mk-sandbox, assurez-vous qu'il est exécutable au moyen de la commande :

```
ls -l mk-sandbox
```

s'il ne possède pas le droit « `x` », ajoutez-le au moyen de la commande :

```
chown a+x mk-sandbox
```

Il ne vous reste plus qu'à l'exécuter :

```
./mk-sandbox
```

ce qui va vous créer tout un tas de sous-répertoires vides, dont quelques-uns ne respectent pas les conventions de notation qui en assureraient la sauvegarde. Vous pouvez donc à présent utiliser le répertoire /tmp/for-lc-7-ruby pour vos tests.

12. ***A**merican **S**tandard **C**ode for **I**nformation **I**nterchange.*

5 Bréviaire

Rappelons que vous pouvez vérifier par vous-mêmes que `File` est une classe et `Find` un module au moyen du résultat des évaluations suivantes [13] :

`File.class` $\Longrightarrow$?? `Find.class` $\Longrightarrow$??

5.1 Classe `File`

Voici quelques méthodes de la classe `File` du langage Ruby :

`File.basename(`*`path`*`)` retourne le nom de la composante finale d'un *pathname*, sans prendre en compte les étapes successives qui permettent d'y arriver :

`File.basename('/tmp/for-lc-7-ruby/viruses')` $\Longrightarrow$ `"viruses"`

`File.directory?(`*`path`*`)` retourne `true` si *`path`* correspond à une ressource réelle, et que cette ressource est un *directory* ; retourne `false` sinon ;

`File.dirname(`*`path`*`)` retourne la concaténation des étapes successives permettant d'arriver à la composante finale d'un *pathname*, sans inclure cette dernière :

`File.dirname('/tmp/for-lc-7-ruby/viruses')` $\Longrightarrow$ `"/tmp/for-lc-7-ruby"`

`File.exist?(`*`path`*`)` retourne `true` si *`path`* correspond à une ressource réelle ; retourne `false` sinon ;

`File.expand_path(`*`path`*`)` traite les symboles spéciaux utilisés dans les *pathnames* — par exemple, le caractère « `~` » pour le *home directory* d'un utilisateur — et retourne le *pathname* absolu correspondant — l'évaluation suivante ne fonctionne bien évidemment que sur le réseau des machines de travaux pratiques du Département d'Informatique de l'Université — :

`File.expand_path('~AD\jmhuffle')` $\Longrightarrow$ `"/home/AD/jmhuffle"`

`File.file?(`*`path`*`)` retourne `true` si *`path`* correspond à une ressource réelle, et que cette ressource est bel et bien un fichier ; retourne `false` sinon ;

`File.join(`*`path`*$_0$`)` concatène les étapes successives de *`path`* avec le nom *`path`*$_0$ pour obtenir le chemin complet d'accès à *`path`*$_0$ à partir de *`path`* :

`File.join('/tmp/for-lc-7-ruby','viruses')` $\Longrightarrow$ `"/tmp/for-lc-7-ruby/viruses"`

`File.rename(`*`path`*$_0$`)` renomme la ressource de nom *`path`* en utilisant le nom *`path`*$_0$: si l'opération est effectivement réalisée, la valeur `0` est retournée, sinon l'exception `SystemCallError` est déclenchée ; dans un tel cas, utiliser la méthode `inspect` pour obtenir le libellé de l'erreur.

`File.split(`*`path`*$_0$`)` retourne un tableau de deux valeurs : toutes les étapes permettant d'arriver à la composante finale d'un *pathname*, puis cette composante finale :

`File.split('/tmp/for-lc-7-ruby/viruses')` $\Longrightarrow$ `["/tmp/for-lc-7-ruby","viruses"]`

5.2 Module `Find`

L'utilisation du module `Find` est signalée par la directive suivante, à placer en début de votre fichier :

```
require 'find'
```

et il fournit la méthode suivante, inspirée de la commande éponyme du système Unix :

13. Attention ! une opération préliminaire est néccessaire pour le chargement du module `Find` : *cf.* § 5.2.

`Find.find(`*`path`*`)` explore récursivement le répertoire *`path`* ainsi que tous ses fichiers et sous-répertoires, et applique à chaque ressource le bloc à un argument formel qui suit. Le résultat retourné par la méthode `find` est `nil`. L'expression suivante permet d'afficher les noms de tous les fichiers et sous-répertoire d'une arborescence *`path_0`* :

```
Find.find(path_0) do |path_1| puts path_1 end ⟹ nil
```