MGL7460 : Laboratoire #1
Introduction à Ruby

15 septembre 2016

Le but de ce laboratoire est de vous familiariser avec l'utilisation du langage Ruby, principalement avec l'utilisation des blocs et du style fonctionnel (notamment, l'utiisation des méthodes du module `Enumerable`).

---

1 Pour vous connecter à la machine Unix

- Connectez vous au poste de travail Windows — code MS et NIP standards UQAM.
- Lancez l'application `x2go` (voir sur le bureau).
- Définissez une nouvelle connection en indiquant :
  - Pour le *host* : `malt.labunix.uqam.ca`
  - Pour le type de session : par défaut le type indiqué est «`KDE`» ; **il faut plutôt aller dans le menu et sélectionner `Gnome`.**

Si vous préférez utiliser votre portable, vous pouvez simplement vous connecter avec `ssh` :

```
$ ssh malt.labunix.uqam.ca
```

2 Pour obtenir le code à compléter

Pour obtenir les fichiers pour ce laboratoire :

```
$ git clone http://www.labunix.uqam.ca/~tremblay/git/LaboRuby.git
```

3 Utilisation d'Unix, édition de fichiers et exécution de commandes

Pour une brève introduction à Unix (ou un bref rappel), consultez le document suivant :
`http://www.labunix.uqam.ca/~tremblay/INF5171/Labos/intro-unix.txt`

Tel qu'indiqué dans ce document, un éditeur graphique simple à utiliser est `gedit`. Mais, évidemment, vous pouvez aussi utiliser `pico`, `nano`, `vim` ou `emacs` si vous les connaissez, ces éditeurs étant tous disponibles sur `malt`.

Pour ce type de laboratoire où je vous fournis un `makefile`, je vous suggère d'avoir (au moins) deux (2) fenêtres, toujours ouvertes, et d'alterner entre ces deux fenêtre (édition, tests, etc) :

a. Une fenêtre où vous faites l'édition du fichier à modifier (`array1.rb` ou `array2.rb`) ;

b. Une fenêtre où vous lancez l'exécution de tests avec la commande `make`.

Donc, évitez de lancer/fermer `gedit` de façon répétitive.

4 Utilisation des méthodes du module `Enumerable` et approche fonctionnelle

> *Learn to use* `Enumerable`*. You will not be a rubyist until you do.*
>
> *«Ruby QuickRef», R. Davis (*`http://www.zenspider.com/Languages/Ruby/QuickRef.html`*)*

Pour cette première partie, divers fichiers vous sont fournis :

- `array1.rb` : Une extension de la classe `Array`, qui définit diverses méthodes pouvant être appelées sur un tableau, **méthodes que vous devez mettre en oeuvre.**
- `array1_spec.rb` : Des tests unitaires, écrits avec `MiniTest`, pour chacune des méthodes que vous devez mettre en oeuvre.
- `makefile` : Un fichier qui permet d'automatiser l'exécution des tests unitaires, et ce en définissant une (1) cible par méthode — en plus d'une cible globale `tests` pour exécuter tous les tests.

  Pour tester une méthode spécifique, il suffit de changer la variable `TEST` dans le `makefile` — voir page suivante.

Plus spécifiquement, chaque méthode à mettre en oeuvre est précédée d'un commentaire qui donne une brève **description** de ce que fait la méthode. Cette description est suivie d'un **petit exemple**, simple. Pour des exemples plus détaillés et plus complets, il faut consulter les tests pour la méthode, dans le fichier `array_spec.rb`.

Voici la liste des méthodes à compléter :

a. `pairs`
b. `partitionner`
c. `trier`
d. `compacter`
e. `juste_des_1`
f. `le_max`
g. `index_du_max`
h. `inverse`
i. `nb_inversions`
j. `to_chaine`
k. `partitionner_bis`

Exemple : la première méthode à mettre en oeuvre, `pairs`

- Fichier `array.rb`

```
# Retourne les elements pairs du tableau.
#
# Exemple:
#   [10, 23, 30].pairs == [10, 30]
#
def pairs
end
```

- Fichier `array_spec.rb`

```
describe "#pairs" do
  it "retourne [] quand []" do
    [].pairs.must_equal []
  end

  it "retourne le tableau quand singleton pair" do
    [10].pairs.must_equal [10]
  end

  it "retourne [] quand singleton impair" do
    [11].pairs.must_equal []
  end

  it "retourne le tableau quand juste des pairs" do
    a = 100.times.map { 2 * Integer(rand) }
    a.pairs.must_equal a
  end

  it "retourne [] quand juste des impairs" do
    a = 100.times.map { 2 * Integer(rand) + 1 }
    a.pairs.must_equal []
  end
end
```

- Fichier `makefile` :

```
# Pour executer un (1) test specifique, changer la variable qui suit
# par un autre nom de methode.  Si vous etes pret a executer *tous les
# tests*, indiquer "TEST=tests"
TEST=pairs

################################################################
....

pairs:
        ruby array_spec.rb -n /pairs/
....
```

5 Mise en oeuvre de méthodes avec `yield`

Comme pour la première partie, vous devez compléter certaines méthodes, définies cette fois dans le fichier `array2.rb`, alors que les tests sont dans le fichier `array2_spec.rb`.

À la différence de la première partie, les méthodes à définir **vont devoir utiliser** `yield`, donc vont devoir appeler un bloc reçu en argument, plutôt que simplement faire un appel à une meéthode en lui passant un bloc.

Voici la liste des méthodes à compléter :

a. `select_`

b. `map_`

c. `inverse_each`

d. `each_prefixe`

e. `map2`

Notez que, pour plusieurs de ces méthodes (mais pas toutes), vous devrez utiliser `each` ou `each_index` et ce dans un style **impératif**.