Théorie des langages rationnels
TP 1 - Un peu de *Python*

Adrien Pommellet

18 janvier 2023

L'objectif de ce TP est de réviser quelques notions de Python qui vous seront utiles pour implémenter plus tard les algorithmes que nous introduirons en cours.

Les questions marquées d'une étoile `(*)` seront **notées** et font l'objet d'un rendu dont **la rédaction doit obligatoirement respecter les règles suivantes** :

— Les réponses au TP doivent être regroupées dans un unique fichier nommé `thlr_1_code.py` (tout autre fichier sera ignoré). Inutile d'uploader les bibliothèques externes que l'on vous fournit telle que `thlr_automata.py`.

— Il est impératif que le fichier soit interprétable, sans la moindre erreur de syntaxe. Si ce n'est pas le cas, le rendu entier sera ignoré. Testez votre code sous Linux directement depuis la console avec l'instruction `python3 thlr_1_code.py` depuis un répertoire contenant ses éventuelles dépendances, sans passer par un IDE.

— Les noms de fonctions donnés dans l'énoncé doivent être respectés.

— La réponse à la question numéro `i` et les éventuelles sous-fonctions utilisées doivent être placées entre des balises de la forme `#[Qi]` et `#[/Qi]` (sans espace après `#`, avec un `/` et non un \). Ainsi, si l'on vous demande d'écrire une fonction `f`, votre réponse doit être de la forme :

```
#[Qi]
def f():
  # Insert your code here
#[/Qi]
```

— Testez vos fonctions sur des exemples pour vous assurer de leur correction ; n'incluez toutefois pas ces tests et ces exemples entre les tags `#[Qi]` et `#[/Qi]`.

— Certaines fonctions sont interdépendantes : une erreur sur la première fonction peut compromettre les suivantes, soyez donc vigilants.

Le non-respect de ces critères peut mettre en défaut le système de notation automatique et conduire à une note de 0 pour le rendu dans son intégralité, sans contestation possible.

1 Quelques rappels

1.1 Commandes Unix

Voilà un petit rappel des commandes et outils incontournables :

`mkdir` *`dir`*	crée un répertoire nommé *`dir`* (MaKe DIRectory)
`ls`	affiche le contenu du répertoire courant (LiSt)
`cd` *`dir`*	va dans le répertoire pointé par *`dir`* (Change Directory)
`cd ..`	retourne dans le répertoire parent
`cd`	retourne dans votre *`HOME`*
`cp` *`src dest`*	copie un fichier (CoPy)
`cp -R` *`src dest`*	copie un répertoire (CoPy)
`mv` *`src dest`*	déplace/renomme un fichier ou un répertoire (MoVe)
`cat` *`file`*	sort le contenu de *`file`* sur la sortie standard

À tout moment, vous pouvez invoquer le manuel en utilisant la commande `man` pour MANual. Par exemple, pour savoir comment utiliser la commande `mkdir`, faites : `man mkdir`.

Essayez d'organiser proprement votre espace de stockage pour vous y retrouver au moment de réviser ou de soumettre vos résultats. Créez par exemple un répertoire `tps` (`mkdir tps`), allez y (`cd tps`), et ajoutez un répertoire `thlr` (`mkdir thlr`). Vous pourrez organiser ce répertoire par TP. **Pensez à y conserver vos résultats d'une séance à l'autre !**

1.2 Utiliser Python

Python est un langage de programmation interprété utilisant le format `.py`. L'instruction `python` *`filename.py`* permet de démarrer l'interprétation du fichier `filename.py` : toutes les instructions (hormis les définitions) dans le corps du fichier sont alors exécutées dans leur ordre de lecture. Si un nom de fichier n'est pas spécifié, une invite par ligne de commande est ouverte que l'on peut quitter avec `Ctrl+D`. On travaillera sur des fichiers que l'on conservera et non directement via la console.

Dans le doute, n'hésitez pas à rechercher les propriétés d'une structure de données ou d'un mot clé dans la documentation officielle de Python.

2 Listes et ensembles en Python

L'utilisation de structures de données telles que les listes ou les ensembles sera nécessaire à la manipulation de langages ; il est donc bénéfique de les voir ou les revoir.

2.1 Utilisation de listes

Une liste en Python représente une série de valeurs non triées avec possiblement des répétitions. Il existe de multiples manières de manipuler les listes que nous allons explorer dans cette sous-partie.

Question 1. Exécutez la suite d'instructions :

```
l = list()
l = [2, 3]
l.append(4)
l = [0, 1] + l
l = l + [5]
```

En affichant avec `print` après chaque instruction la valeur de la liste `l`. Décrivez les effets de chaque opération sur la liste `l`.

Question 2. Affichez les éléments suivants :

```
len(l)
l[0]
l[1:]
l[1:3]
l.pop()
```

Déduisez-en l'effet des différentes instructions.

2.2 Manipulation d'ensembles

Un ensemble ou `set` en Python est une structure de données qui vise à reproduire le comportement d'un ensemble mathématique et diffère donc d'une liste par certains aspects.

Question 3. Exécutez la suite d'instructions :

```
l1 = [0, 0, 1, 1, 2, 3, 3, 3]
X = set(l1)
Y = {0, 3, 2, 1, 1, 2, 3, 1}
```

Affichez ensuite les ensembles `X` et `Y`. Que dire de l'ensemble `X` par rapport à la liste `l1` ? Plus généralement, que dire des ensembles par rapport aux listes ?

Question 4. Exécutez la suite d'instructions :

```
Y.add(5)
Y.remove(5)
```

En affichant après chaque instruction la valeur de l'ensemble `Y`. Décrivez les effets de chaque opération sur l'ensemble `Y`.

Question 5. Essayez de définir l'ensemble `Z = {[0, 1], [0, 3]}`. Est-ce possible ? Pourquoi ?

Question 6. Que fait l'instruction `list(Y)` ?

3 Quelques fonctions

Cette section est l'occasion de revoir la définition et l'utilisation de fonctions en Python, en particulier sur des listes et des ensembles.

3.1 Généralités

Question 7. Introduisez la fonction suivante :

```
def test(E):
  if E:
    print("Cas 1.")
  else:
    print("Cas 2.")
  if 1 in E:
    print("Vu.")
```

Puis testez-la sur l'ensemble `Y` défini précédemment et sur un ensemble vide `set()`. Que fait cette fonction ? Que pouvez-vous en conclure sur le sens des instructions `if E:` et `1 in E` ?

Question 8. Introduisez la fonction suivante :

```
def something(E):
  x = 0
  for y in E:
    x += y
  return x
```

Puis testez-la sur des ensembles d'entiers (par exemple `X` et `Y`). Que fait cette fonction ? Notez bien la syntaxe de la boucle `for`.

3.2 Quelques opérations ensemblistes

Habituez-vous à manipuler des ensembles en implémentant les fonctions classiques suivantes, sans utiliser celles incluses par défaut en Python bien sûr.

Question 9 (⋆). Définissez une fonction `union(E, F)` qui renvoie l'union de deux ensembles `E` et `F` sans modifier `E` et `F`.

Question 10 (⋆). Définissez une fonction `intersection(E, F)` qui renvoie l'intersection de deux ensembles `E` et `F` sans modifier `E` et `F`.

Question 11 (⋆). Définissez une fonction `subtraction(E, F)` (et non pas *substraction*) qui renvoie la soustraction ensembliste de `F` à `E` sans modifier `E` et `F`, c'est-à-dire les éléments dans `E` mais pas dans `F`.

Question 12 (⋆). Définissez une fonction `diff(E, F)` qui renvoie la différence symétrique des ensembles `E` et `F` sans modifier `E` et `F`, c'est-à-dire les éléments dans `E` ou `F` mais pas dans les deux à la fois.

4 Un peu de récursion

Passons aux choses sérieuses : nous souhaitons désormais écrire une fonction `power_set(E)` qui renvoie une liste contenant les sous-ensembles d'un ensemble `E`. Par exemple :

```
>>> power_set({0, 2, 6})
[set(), {0}, {2}, {0, 2}, {6}, {0, 6}, {2, 6}, {0, 2, 6}]
```

Pour commencer, nous souhaitons écrire une fonction `sublists(l)` qui renvoie une liste de toutes les sous-listes d'une liste `l`, c'est-à-dire toute liste que l'on peut obtenir en retirant zéro ou plusieurs éléments de `l`. Par exemple :

```
>>> sublists([0, 1, 1, 2])
[[], [0], [1], [0, 1], [1], [0, 1], [1, 1], [0, 1, 1], [2], [0, 2], [1, 2],
[0, 1, 2], [1, 2], [0, 1, 2], [1, 1, 2], [0, 1, 1, 2]]
```

Pour ce faire, nous allons utiliser une fonction *récursive* (c'est-à-dire, qui fait appel à elle-même dans sa propre définition) `sublists(l)` qui effectue les opérations suivantes :

– si `l` est vide, renvoyer la liste contenant la liste vide `[[]]` ;
– sinon, puisque `l` est non vide, appliquer la fonction `sublists` à sa queue `l[1:]` et pour toute sous-liste `s` ainsi obtenue, ajouter `s` et `[l[0]] + s` à la liste désirée.

Notre intuition est la suivante : une sous-liste de `l` peut contenir la tête `l[0]` de `l` ou non. Dans le premier cas, si on lui retire `l[0]`, on obtient une sous-liste de `l[1:]` ; dans le second cas, c'en est déjà une.

Question 13 ($\star$). Écrivez une fonction `sublists(l)` qui renvoie une liste contenant toutes les sous-listes possibles de la liste `l`. L'ordre de la liste renvoyée importe peu.

Question 14 ($\star$). En réutilisant la fonction `sublists(l)` de la manière la plus simple, écrivez une fonction `power_set(E)` qui renvoie une liste contenant tous les sous-ensembles possibles de l'ensemble `E`.