Manipulations avancées de listes en Python

Romain Tavenard

1 Travail à préparer chez vous avant la séance

1. Proposez un algorithme permettant de répondre à l'énoncé suivant (2 approches sont envisageables : la simulation ou le calcul exact) :

   On s'intéresse à un jeu de dés dont la règle est la suivante : on lance un premier dé qui nous donne une valeur N. On relance ensuite N dés et l'on compte la somme S des valeurs de ces dés. Pour chaque valeur possible de S, quelle est la probabilité de cette somme ?

2 Construction de liste

Il existe plusieurs façons de créer une liste. La première est d'utiliser la syntaxe `l = [1, 7, 2]`. Une autre façon est de partir d'une liste vide puis de la peupler progressivement au fil de l'exécution du programme. Par exemple :

```
l = []
for i in range(n):
    l.append(i ** 2)
```

Il existe une troisième syntaxe plus concise que la précédente mais qui lui est strictement équivalente : l'utilisation de listes en compréhension. L'exemple précédent devient par exemple :

```
l = [i ** 2 for i in range(n)]
```

Cette syntaxe se rapproche de la notation ensembliste mathématique correspondante :

$$l = \{i^2 | i \in [0, n[\}$$

2. Écrivez deux fonctions équivalentes qui prennent en entrée une liste et retournent la liste contenant les carrés des nombres de la liste passée en argument.

3. Si l'on souhaite maintenant construire une liste correspondant à la notation mathématique suivante :

$$l = \{i^2 | i \in [0, n[, i\text{pair}\}$$

on peut encore utiliser les listes en compréhension :

```
l = [i ** 2 for i in range(n) if i % 2 == 0]
```

4. Écrivez deux fonctions équivalentes qui prennent en entrée une liste et retournent la liste ne contenant que les nombres positifs de la liste passée en argument.

3 Copie de liste

La copie de liste en Python est un exercice délicat. En effet, si l'on crée une première liste `liste1` puis que l'on écrit

```
liste2 = liste1
```

le contenu de `liste1` ne sera pas recopié dans `liste2`, les deux objets ne feront qu'un. Cela signifie notamment que si l'on modifie l'une des deux listes, la modification sera répercutée sur l'autre liste.

5. Pour vous en convaincre, mettez en œuvre ce scénario et affichez le contenu des deux listes après modification de l'une des deux.

Ainsi, si l'on veut copier le contenu d'une liste dans une autre, on devra utiliser une astuce syntaxique. Deux approches sont possibles. La première utilise une liste en compréhension, la deuxième la fonction `list` qui effectue une copie de la liste passée en argument.

6. Mettez en œuvre ces deux approches et vérifiez que le problème observé lors de la manipulation précédente ne se pose plus.

4 Tri de liste

Nous avons vu, au fil des TD précédents, qu'il était possible d'obtenir les extrema d'une liste à l'aide des fonctions `min` et `max`. Sachez qu'il existe également une fonction de tri de liste : la fonction `sorted`. Lorsque l'on souhaite trier une liste en utilisant une relation d'ordre classique, sa syntaxe est relativement simple. Essayez, pour vous en convaincre, d'exécuter les deux lignes suivantes et analysez l'utilisation qui y est faite de la fonction `sorted` :

```
print(sorted([1, 7, 32, 2, 9, 5]))
print(sorted([1, 7, 32, 2, 9, 5], reverse=True))
```

Le tutoriel Python spécifique au tri de listes donne un aperçu de ce que l'on peut faire avec cette fonction `sorted` : https://docs.python.org/3/howto/sorting.html. Un paramètre utile de cette fonction est le paramètre `key`. Celui-ci spécifie une fonction que l'on appliquera aux éléments de notre liste avant de les classer. Ainsi, lorsque l'on souhaite effectuer un tri qui soit insensible à la casse, il est possible d'appliquer la fonction `str.lower` (*ie*, la méthode `lower` de la classe `str`) aux éléments avant d'effectuer le tri :

```
sorted(ma_liste_de_str, key=str.lower)
```

Ce paramètre peut également être utilisé lorsque l'on souhaite trier une liste de tuples. Pour reprendre l'exemple du tutoriel précédemment cité, supposons que l'on souhaite trier la liste suivante :

```
student_tuples = [
    ('john', 'A', 10),
    ('jane', 'B', 12),
    ('dave', 'B', 10)]
```

Il s'agit d'une liste de tuples dans laquelle le premier élément de chaque tuple est le prénom de l'élève, le second son groupe de TD et le troisième son âge. Si l'on applique le tri directement sur cette liste, le tri sera effectué d'abord sur le premier élément du tuple puis, en cas d'égalité, sur le deuxième et enfin, sur le troisième. Or, il se peut que l'on souhaite trier les étudiants par âge.

7. Pour cela, codez une fonction `troisieme_element` qui retourne le troisième élément d'un tuple passé en argument.

Ensuite, on utilise la fonction de tri comme suit :

```
sorted(student_tuples, key=troisieme_element)
```

8. On souhaite maintenant effectuer un tri d'abord sur l'âge puis sur le prénom. Pour cela, écrivez une fonction qui, à partir d'un tuple de 3 éléments retourne un nouveau tuple constitué des troisième et deuxième élément du tuple fourni en argument. Utilisez cette fonction pour trier la liste d'étudiants d'abord sur l'âge puis sur le prénom.

5 Listes de listes

Dans certains cas, il est utile de manipuler des données sous forme de matrice. Or le type matrice n'existe pas dans la librairie standard de Python (nous verrons dans la suite qu'une librairie spécialisée existe, toutefois, permettant d'effectuer des manipulations avancées de matrices).

Si l'on souhaite donc représenter des matrices à partir de listes, l'astuce à utiliser sera de construire des listes de listes. Si l'on veut représenter la matrice suivante :

$$M = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

on pourra représenter dans une première liste la première ligne, dans une seconde la deuxième ligne et enfin la liste de ces lignes sera notre matrice :

```
ligne1 = [1, 2]
ligne2 = [3, 4]
matrice = [ligne1, ligne2]
```

ou, écrit de manière plus concise :

```
matrice = [[1, 2], [3, 4]]
```

Ainsi, pour accéder à l'élément d'indice (i, j) de cette matrice, on devra écrire `matrice[i][j]` (on demande le j-ème élément de la i-ème ligne de la matrice).

9. Soit M une matrice dont les lignes correspondent à des points dans un espace à deux dimensions. Écrivez une fonction qui trie cette matrice (on pourra utiliser un appel à `sorted` de manière à obtenir les points les plus proches de l'origine aux premières lignes de M. Vous utiliserez la matrice suivante pour tester votre fonction :

$$M = \begin{pmatrix} 0.5 & 0.5 \\ 5 & 1 \\ 0.5 & 0.3 \\ 0.7 & 1 \end{pmatrix}$$

10. Écrivez une fonction qui prend en entrée une matrice (telle que celle de la manipulation précédente) et un vecteur (une liste de taille 2, dans le cas de l'exercice précédent) et retourne la ligne de la matrice correspondant au point le plus proche du vecteur fourni.

6 Exercice de synthèse

11. Écrivez une fonction qui prenne les mêmes arguments qu'à la manipulation précédente et retourne la liste des distances du vecteur à chacune des lignes de la matrice.