Séance 5 : Listes, tuples et gestion mémoire

L1 – Université Côte d'Azur

Restrictions

On interdit les méthodes `append` et `insert` sur les listes sauf dans les exercices où elles sont explicitement autorisées.

Rappel : somme vectorielle

La somme de deux vecteurs du plan, de coordonnées (x_1, x_2) et (y_1, y_2) est le vecteur de coordonnées $(x_1 + y_1, x_2 + y_2)$. Plus généralement, si $\vec{x} = (x_1, \dots, x_n)$ et $\vec{y} = (y_1, \dots, y_n)$ sont des vecteurs d'un espace de dimension n, leur somme est le vecteur $(x_1 + y_1, \dots, x_n + y_n)$.

Exercice 1 — Somme vectorielle (⋆)

1. Que vaut `(1,2) + (3,4)` ?
2. Écrivez une fonction `somme_vect2(u,v)` qui prend en arguments deux vecteurs du plan $\vec{u}$ et $\vec{v}$, chacun représenté par un couple, et qui renvoie leur somme vectorielle. Par exemple, `somme_vect2((1,2), (3,4))` renvoie `(4,6)`.
3. Écrivez une fonction `somme_vect(u,v)` qui prend deux vecteurs de même taille et qui renvoie leur somme vectorielle. Le programme doit renvoyer le tuple vide si les vecteurs `u` et `v` n'ont pas le même taille.
4. Écrivez cinq tests avec `assert`.

Exercice 2 — Lettre et le néant (⋆)

Écrivez une fonction `première_lettre(s)` qui prend en argument une chaîne de caractères `s` et qui retourne dans un couple l'indice du premier caractère de `s` qui est une lettre ainsi que ce caractère. Si la chaîne `s` ne contient aucune lettre, la fonction renvoie `None`. Par exemple, `première_lettre('12ab')` renvoie `(2,'a')`. Vous pourrez utiliser la méthode `isalpha` sur les chaînes de caractères.

Exercice 3 — Comprendre les compréhensions (⋆)

Pour chacune des constructions de liste par compréhension, donnez la valeur finale de la liste puis écrivez une boucle `for` construisant une telle liste. On pourra utiliser la méthode `append`.

1. `[ x*x for x in range(4)]`
2. `[(i,10-i) for i in range(11)]`
3. `[ str(i) for i in range(11) if i%2==0]`

Exercice 4 — Moyenne (⋆)

Écrivez une fonction `moyenne(L)` prenant en argument une liste non vide L d'entiers ou de flottants et renvoyant leur moyenne.

Exercice 5 — Pierre-Feuille-Ciseaux (⋆)

On souhaite établir une correspondance entre certaines chaînes de caractères et des entiers (cf TP 1), à l'aide de deux fonctions `f` et `g` inverses l'une de l'autre. On aura par exemple

```
 f('pierre')==0    g(0)=='pierre'
f('feuille')==1    g(1)=='feuille'
f('ciseaux')==2    g(2)=='ciseaux'
```

En utilisant une variable globale correspondant à un tuple, définissez les fonctions `f` et `g` sans utiliser de branchement conditionnel. Vous pourrez utiliser la méthode `index`.

Exercice 6 — Reconnaître une liste triée (⋆)

1. Écrivez une fonction `est_triée(L)` qui prend en argument une liste d'entiers `L` et qui renvoie `True` si la liste est triée en ordre croissant. Par exemple, `est_triée([1,2,2])` renvoie `True` et `est_triée([1,5,2])` renvoie `False`.
2. Écrivez des tests avec `assert`.

Exercice 7 — Compactage (⋆⋆)

Dans cet exercice, vous pouvez utiliser la méthode `append`.

1. Écrivez une fonction `grouper(L)` qui prend en argument une liste `L` et qui renvoie la liste obtenue en remplaçant toute suite d'éléments consécutifs `x,x,x,…,x` par un seul élément `x`. Par exemple, `grouper([4,4,4,2,2,4])` renvoie `[4,2,4]`.
2. Écrivez une fonction `compacter(L)` qui prend en argument une liste `L` et qui renvoie la liste obtenue par groupage en indiquant de plus à l'aide d'un couple la taille de chaque groupe. Par exemple, `compacter([4,4,4,2,2,4])` renvoie `[(3,4),(2,2),(1,4)]`.

Tri par insertion

Le tri par insertion est un algorithme de tri qui insère un à un les éléments à trier dans une liste qui contient à la fin le résultat attendu. Par exemple, on aura

```
Étape 0 : triés : []         à trier : [5,8,7,1]
Étape 1 : triés : [5]        à trier : [8,7,1]
Étape 2 : triés : [5,8]      à trier : [7,1]
Étape 3 : triés : [5,7,8]    à trier : [1]
Étape 3 : triés : [1,5,7,8]  à trier : []
```

Exercice 8 — Tri par insertion (⋆⋆)

1. Écrivez une fonction `index_insertion(L,n)` qui prend en arguments une liste triée d'entiers `L` et un entier `n` et qui renvoie l'indice de la position à laquelle insérer `n` dans `L` afin de garder la liste triée. Par exemple, `index_insertion([1,5,6,10],2)` renvoie 1 car 2 est à l'indice 1 dans la liste `[1,2,5,6,10]`. De plus, si `n` est déjà dans la liste, on renverra le plus grand indice qui convient. Par exemple, `index_insertion([1,2,6,10],2)` renvoie 2.
2. Écrivez une fonction `insertion_triée(L)` qui prend en argument une liste d'entiers `L` et qui renvoie une nouvelle liste contenant les entiers de `L` en ordre croissant. Vous pourrez utiliser la méthode `insert` sur les listes.