Séance 5: Tuples et listes

L1 – Université Nice Sophia Antipolis

Objectifs:

- Savoir construire et déconstruire un tuple
- Listes de taille fixe, notion d'index et de longueur
- Comparer les éléments d'une même liste
- Algorithme de tri insertion avec recopie

Restrictions

On garde la contrainte des TDs précédents : il est **interdit d'utiliser la boucle while**. On interdit aussi 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 $\mathbf{x} = (x_1, \ldots, x_n)$ et $\mathbf{y} = (y_1, \ldots, y_n)$ sont des vecteurs d'un espace de dimension n, leur somme est le vecteur $(x_1 + y_1, \ldots, x_n + y_n)$.

Exercice 1 (Somme vectorielle, $\star$)

1. *Que vaut* `(1,2) + (3,4)` *?*
2. *Écrivez une fonction* `vect_sum2(x,y)` *qui prend en arguments deux vecteurs du plan* $\mathbf{x}$ *et* $\mathbf{y}$*, chacun représenté par un couple, et qui renvoie leur somme vectorielle. Par exemple,* `vect_sum2( (1,2) , (3,4) )` *renvoie* `(4,6)`*.*
3. *Écrivez une fonction* `vect_dim(x)` *qui prend en argument un vecteur* $\mathbf{x}$ *et qui renvoie sa dimension. Par exemple,* `vect_dim( (1,2) )` *renvoie* 2 *et* `vect_dim( (2,3,3,7) )` *renvoie* 4*.*

□

Exercice 2 (Lettre et le néant, $\star$)
Écrivez une fonction `find_alpha(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. Par exemple,* `find_alphanum('12ab')` *renvoie* `(2,'a')`*. Si la chaîne* s *ne contient aucune lettre, la fonction renvoie* `None`*. Vous pourrez utiliser la méthode* `isalpha` *sur les chaînes de caractères.* □

Exercice 3 (Moyenne, $\star$)
Écrivez une fonction `moyenne(L)` *prenant en argument une liste non vide* L *d'entiers ou de flottants et renvoyant leur moyenne.* □

Exercice 4 (Pierre-Feuille-Ciseaux, $\star$)
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 qui est un tuple, définissez les fonctions f et g sans utiliser de branchement conditionnel. Vous pourrez utiliser la méthode `index`. □

Exercice 5 (Reconnaître une liste triée, $\star$)
Écrivez une fonction `est_triee(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_triee([1,2,2])` *renvoie* `True` *et* `est_triee([1,5,2])` *renvoie* `False`. □

Exercice 6 (Compactage, $\star\star$)
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 $x, x, x, \ldots, x$ 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

```
Etape 0 : tries : []          a trier : [5,8,7,1]
Etape 1 : tries : [5]         a trier : [8,7,1]
Etape 2 : tries : [5,8]       a trier : [7,1]
Etape 3 : tries : [5,7,8]     a trier : [1]
Etape 3 : tries : [1,5,7,8]   a trier : []
```

Exercice 7 (Tri par insertion, $\star\star$)

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]`. *Si n est déjà dans la liste L, on renverra le plus grand indice qui convient. Par exemple,* `index_insertion([1,2,6,10],2)` *renvoie 2.*
2. *Écrivez une fonction* `insertion_sorted(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.*

□