DS n^{o}1 d'informatique commune

À lire attentivement avant de faire quoi que ce soit :

— les consignes de présentation et de soin données dans les autres matières s'appliquent aussi à l'informatique

— les programmes sont tous à coder en langage Python : veillez à bien respecter les règles de syntaxe ainsi que les indentations

— l'usage de commentaires dans les programmes est encouragé si ceux-ci peuvent être éclairant pour le correcteur

— la clarté des programmes sera particulièrement valorisée dans la notation

— à l'inverse, la rapidité d'exécution, bien que prise en compte, ne jouera pas une partie prépondérante dans la notation

— certaines questions demandent de coder des fonctions déjà existantes : l'utilisation de ces fonctions pré-implémentées dans Python est à proscrire

— à l'inverse, si **dans un même exercice** une question demande de coder une fonction, que cette question soit traitée ou pas, la fonction codée pourra être utilisée dans les questions suivantes **de cet exercice**

— les exercices sont indépendants, et il est encouragé de les traiter dans l'ordre

Exercice 1

On cherche ici à manipuler des listes (éventuellement vides) d'entiers relatifs pour en extraire différentes informations. On prendra bien garde dans les programmes qui suivent à traiter le cas de la liste vide.

1. Écrire une fonction `somme(L)` qui rend la somme des entiers qui composent la liste `L`. Par convention, la liste vide est de somme nulle.
2. Écrire une fonction `moyenne(L)` qui rend la valeur moyenne des entiers qui composent `L`. Par convention, cette valeur sera nulle si la liste est vide.
3. Écrire une fonction `centree(L)` qui rend une liste dont les éléments diffèrent de ceux de `L` d'une constante, mais de moyenne nulle.
4. Écrire une fonction `repartition(L)` qui rend une liste de la forme `[a,b,c]` où `a`, `b` et `c` correspondent respectivement au nombre d'éléments de `L` qui sont strictement inférieur, égal ou strictement supérieur à la moyenne de `L`.
5. Écrire une fonction `ecarttype(L)` qui détermine l'écart-type des valeurs des éléments de `L`. Par convention, cette valeur sera nulle si la liste est vide.

Exercice 2

On cherche ici à manipuler des listes (éventuellement vides) dont les éléments sont quelconques. On prendra bien garde dans les programmes qui suivent à traiter le cas de la liste vide.

1. Écrire une fonction `appartient(L,a)` qui renvoie `True` si `a` est un élément de `L` et `False` sinon.
2. Écrire une fonction `occurence(L,a)` qui revoie le nombre de fois que `a` apparaît dans la liste `L`.
3. Écrire une fonction `elements(L)` qui rend une liste qui possède les mêmes éléments que `L`, mais sans répétitions. Par exemple, l'exécution de `elements([1,3,1,2,8,3])` rendra `[1,3,2,8]`.
4. Écrire deux fonctions `union(L1,L2)` et `intersection(L1,L2)` qui prennent chacune deux listes `L1` et `L2` et rendent respectivement la liste sans répétition des éléments qui sont :
   — dans `L1` ou dans `L2` (pour `union`) ;
   — dans `L1` et dans `L2` (pour `intersection`) ;
5. Écrire une fonction `diffsym(L1,L2)` qui prend deux listes `L1` et `L2` et rend la liste sans répétition des éléments qui sont dans `L1` ou dans `L2`, mais pas simultanément dans les deux listes.
6. Écrire une fonction `elementsbis(L)` qui rend une liste de couples de la forme `(a,b)` où les valeurs de `a` sont distinctes et décrivent toutes les valeurs des éléments de `L`, et la valeur de `b` correspondante est le nombre de fois que `a` apparaît dans la liste `L`.

Exercice 3

On cherche ici à manipuler des listes (éventuellement vides) d'entiers relatifs pour les étudier vis-à-vis de l'ordre dans lequel sont rangés ses éléments. On prendra bien garde dans les programmes qui suivent à traiter le cas de la liste vide lorsque cela a un sens. On considère qu'une liste `L` est triée si ses éléments sont rangés par ordre croissant. La liste vide et les listes à un éléments sont en particulier toujours triées.

1. Écrire une fonction `triee(L)` qui rend `True` si la liste `L` est triée, et `False` sinon.
2. On suppose dans cette question seulement que tous les éléments de `L` sont entre 0 et 10.
   (a) Écrire une fonction `comptage(L)` qui rend la liste, dans l'ordre, du nombre d'occurrence de 0, 1, ... dans `L`.
   (b) En déduire une fonction `tricomptage(L)` qui rend une version triée de la liste `L` à l'aide de la fonction `comptage`.
3. On souhaite donner une méthode plus générale de tri, en triant la liste `L` de la manière suivante :
   — on crée une liste `L1` ayant les mêmes éléments que `L` ;
   — on cherche le maximum de `L`, qu'on place à la fin de `L1`
   — on cherche le deuxième plus grand élément de `L`, qu'on place à l'avant dernière place dans `L1` ;
   — on continue jusqu'à avoir rempli `L1` par les éléments de `L`

   en prenant garde à ce que, pendant toute l'exécution, les éléments de `L1` soient toujours les mêmes que ceux de `L`.
   (a) Écrire une fonction `indicemax(L)` qui rend l'indice du maximum de la liste `L`. On n'utilisera pas cette fonction sur la liste vide.
   (b) On suppose que l'on a déjà placé les i plus grands éléments de `L` à la fin de `L1` : entre quels indices (dans `L1`) se trouve le $(i+1)$-ème plus grand élément de `L` ?
   Indication : on vérifiera bien les valeurs pour $i = 0$ et $i = 1$
   (c) En déduire une écriture de la fonction `tri(L)` qui trie la liste `L` suivant la méthode décrite ci-dessous. L'algorithme prendra bien en compte les remarques précédentes, notamment :
   — la liste `L` n'est pas modifiée dans l'exécution ;
   — durant toute l'exécution, les listes `L1` et `L` ont les mêmes éléments ;

Exercice 4

On considère $n \in \mathbb{N}$ avec $n \geqslant 2$. On souhaite déterminer la liste de tous les nombres premiers inférieurs ou égaux à n.

1. On procède ici de manière naïve :
   (a) Écrire une fonction `estpremier(p)` qui rend `True` si `p` est premier et `False` sinon.
   (b) En déduire une fonction `premiersnaif(n)` qui rend la liste des nombres premiers inférieurs ou égaux à `n`.
2. On souhaite coder notre fonction par la méthode du crible d'Erathostène, dont on rappelle le principe :
   — on part de la liste de tous les entiers de 2 à n, qui sont initialement tous considérés ;
   — on part du premier entier considéré, et on supprime tous ses multiples, à part lui même, qu'on ne considèrera plus ;
   — on continue jusqu'à épuisement de la liste ;
   — la liste des nombres non barrés constitue la liste cherchée de nombres premiers.

   Coder le crible.
   Indication : on pourra utiliser la commande `i in range(m,n,p)` qui permet de parcourir tous les entiers de la forme $i = m + kp$ pour k entier tel que $m \leqslant i < n$.