Pratique du langage python

1. On dispose de nombres (entiers ou flottants) rangés dans une liste L.
   a) Écrire une fonction `moyenne(L)` qui renvoie la valeur moyenne m de ces nombres.
   b) Écrire de même une fonction `eqmoy(L)` qui renvoie l'écart quadratique moyen e de ces nombres. On rappelle que :
   $$e = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - m)^2$$
   où m est la valeur moyenne des n nombres x_0,..., x_{n-1}.
   c) Toujours avec une liste L de n nombres, écrire une fonction `eltMin(L)` qui renvoie le plus petit élément ; une fonction `eltMax(L)` qui renvoie le plus grand élément.
   d) Écrire une fonction `eltIndiceMin(L)` qui renvoie le plus petit élément de L et l'indice dans L du premier plus petit élément rencontré
   e) Écrire une fonction `present(L,x)` qui teste si un nombre x est présent dans une liste de nombres et qui envoie `True` ou `False` selon le cas.
   f) Écrire une fonction `nombreFoisPresent(L,x)` qui renvoie le nombre de fois où x est présent dans L (et 0 si $x \notin L$).
2. On travaille maintenant avec des entiers naturels. On considère un entier $n \in \mathbb{N}$.
   a) Écrire une fonction qui teste si $n > 0$ et, dans l'affirmative, calcule la somme des entiers de 1 à n.
   b) Écrire une fonction qui calcule la factorielle de n.
   c) Écrire une fonction qui teste si $n > 0$ et, le cas échéant, calcule le produit des nombres pairs compris entre 1 et n.
   d) Écrire une fonction qui teste si n est premier.
3. Écrire une fonction `sommeRiemann(eps)` qui renvoie la somme :
   $$S = \sum_{n=1}^{\infty}\frac{1}{n^2}$$
   On arrête la boucle lorsque le dernier terme de la somme est inférieur à eps > 0. On pourra tester cette fonction pour eps $= 10^{-10}$, ce qui s'écrit `1e-10`.
4. Écrire une fonction `wallis(eps)` qui renvoie une valeur approchée de π donnée par la formule de Wallis :
   $$\pi = 2\prod_{n=1}^{\infty}\frac{4n^2}{4n^2-1}$$
   On réfléchira à la condition d'arrêt de la boucle.

Révisions sur les tris

Dans les exercices suivants, on souhaite trier une liste L de n éléments de même nature sur lesquels est défini un ordre total.

1. **Le tri fusion**

   Le tri fusion fait partie des algorithmes de type "diviser pour résoudre". Il utilise une fonction `fusion(L1,L2)` qui prend en paramètres deux listes `L1` et `L2`, de longueurs respectives n_1 et n_2, triées séparément (par ordre croissant par exemple) et qui renvoie une liste `L` de longueur $n_1 + n_2$ dont tous les éléments sont triés (par ordre croissant).
   a) Écrire la fonction `fusion(L1,L2)`. On fera attention aux cas où $n_1 = 0$ ou bien $n_2 = 0$ (ce qui correspond à des listes vides).

b) Reproduire dans votre éditeur Python puis compléter le code ci-dessous implémentant le tri fusion d'une liste L :

```
def triFusion(L) :
    n = len(L)
    if n <= 1 :
        return L
    else :
        ... # code à compléter
        return fusion(L1,L2)
```

2. **Le tri par insertion**

Ce tri fonctionne grâce à une boucle qui examine successivement tous les éléments de L allant de l'indice 1 jusqu'au dernier élément. Lors de l'étape examinant l'élément d'indice i, on sauvegarde `L[i]` dans une variable `temp` et on applique l'algorithme suivant :

α) Faire `j = i`.

β) Tant que `j > 0` et que `L[j-1] > temp`, décaler l'élément d'indice `j-1` d'une alvéole vers la droite puis diminuer `j` d'une unité.

γ) Placer `temp` dans `L[j]`.

```
def triInsertion(L) :
    n = len(L)
    for i in range(1,n) :
        temp = L[i]
        j = i
        while ( j > 0 and L[j-1] > temp) :
            ... # compléter le code
```

Remarque : cette fonction ne renvoie rien.

Aurait-on pu dans la boucle while inverser les deux termes de la condition, c'est à dire écrire :
`while ( L[j-1] > temp and j > 0)` ?

3. **Le tri par comptage.**

Le tri par comptage prend en argument une liste `L` **d'entiers** à trier dont on connaît un minorant m et un majorant M. L'idée est de compter le nombre d'occurence dans `L` de chaque élément de `L` compris entre m et M.

Ce comptage se fait grâce à un dictionnaire dont les clés sont les éléments `e` de `L` et dont les valeurs sont le nombre de fois que `e` est présent dans `L`.

a) Écrire une fonction `comptage(L)` qui renvoie le dictionnaire décrit ci-dessus.

b) L'étape suivante consiste à construire une seconde liste `L2` de même taille que `L` : le plus petit élément de `L` est placé dans `L2` autant de fois qu'il apparaît dans `L`. On passe ensuite à l'élément suivant et ainsi de suite, jusqu'à atteindre le plus grand élément.

Cette étape est réalisée par une fonction `triComptage(L,m,M)` qui renvoie `L2`, écrite en partie ci-dessous et dont on complétera le code manquant.

```
def TriComptage(L,m,M) :
    D = comptage(L)
    L2 = []
    for e in range(m,M+1) :
        ... # compléter le code
    return L2
```