Algorithmes simples sur les listes

Dans tous les exercices qui suivent on fait l'hypothèse que l'on travaille sur une liste `L` qui contient un certain nombre d'entiers.

0. Ecrire un programme qui recherche la valeur maximale d'une liste.
1. Ecrire un programme qui calcule la somme des éléments de la liste.
2. Ecrire un programme qui calcule la moyenne des éléments de la liste.
3. Ecrire un programme qui calcule l'"amplitude" de la liste, c'est-à-dire l'écart entre la valeur la plus faible et la valeur la plus haute.
4. Ecrire un programme qui détermine si une valeur donnée appartient à la liste (en faisant un parcours de la liste, sans utiliser `in`)
5. Ecrire un programme qui détermine si une valeur donnée apparaît plus d'une fois dans la liste.
6. Ecrire un programme qui affiche tous les éléments de la liste qui n'apparaissent qu'une fois.
7. Ecrire un programme qui détermine si une liste est incluse dans une autre.
8. Ecrire un programme qui trie une liste dans l'ordre croissant.

[1]:
```
# Exercice 0
L = [0,7,8,5,6,21,8,45,3]
max = 0
for val in L:
    if max < val:
        max = val
print(max)
```

```
45
```

[16]:
```
# Version avec une liste d'entiers aléatoire
import random
# Tirage au sort d'un entier pour la taille
taille = random.randint(5,20)
# Tirage au sort d'une liste d'entiers de longueur "taille"
# (tirage avec remise)
liste = random.choices(range(20),k=taille)
print(liste)
```

```
[13, 13, 5, 5, 10, 2, 3, 15, 3, 12, 16]
```

[17]:
```
# Exercice 0 : version sous forme de fonction
def valmax(l):
    max = 0
    for val in l:
        if max < val:
            max = val
    return max
```

[18]:
```
print(valmax(liste))
```

```
16
```

[19]:
```
# Exercice 2
somme  = 0
for x in L:
    somme+=x
moyenne = somme/len(L)
print("moyenne =",moyenne)
```

```
moyenne = 11.444444444444445
```

[20]:
```
# Exercice 2: version sous forme de fonction
def moyenne(l):
    somme  = 0
    for x in l:
        somme += x
    return somme/len(L)
print("La moyenne vaut %f" % moyenne(L))
```

```
La moyenne vaut 11.444444
```

[21]:
```
# Exercice 3 : on définit la fonction valmin()
# sur le modèle de valmax() plus haut
def valmin(L):
    min=1000
    for val in L:
        if min>val:
            min=val
    return min
```

[23]:
```
# Test unitaire de la fonction
# Avant d'utiliser la fonction, on la teste sur des données qu'on connaît
# pour garantir qu'elle fonctionne.
print(valmin([0,1,2])) # ça devrait être 0
print(valmin([1,2,0])) # ça devrait être 0
print(valmin([9,8,7,6,1,2,3])) # ça devrait être 1
```

```
0
0
1
```

[24]:
```
# Exercice 3 : version où on utilise les fonctions valmin() et valmax()
# Très facile à écrire, mais 2 parcours de la liste
amp = valmax(L)-valmin(L)
print("Amplitude = ", amp)
```

```
Amplitude =  45
```

[26]:
```
# Exercice 3 : essayons de faire un seul parcours
# Pour l'initialisation, au lieu de choisir des valeurs arbitraires
# (0 pour le min, 1000 pour le max), on prend la première valeur de
# la liste, ce qui explique que la boucle commence à i = 1
min = max = L[0]
i = 1
while i < len(L):
    if L[i] > max:
        max = L[i]
    elif L[i] < min:
        min = L[i]
    i += 1
print("Amplitude : ", max - min)
```

```
Amplitude :  45
```

La complexité de la version proposée ci-dessus est légèrement plus favorable que celle de la version précédente. Soit n la longueur de la liste. On rappelle que la version précédente fait $2 \times n$ comparaisons (dans tous les cas) et $2 \times n$ incrémentations du compteur de la boucle. La version présente fait exactement n incrémentations (la moitié donc), et quant au nombre de comparaisons on sait qu'il est inférieur à $2 \times n$ car si le premier test est vérifié (l'entier courant est strictement plus grand que la max), alors le second test n'est pas fait (même dans le cas extrême où le min et le max sont les mêmes, si l'entier est *strictement* plus grand que le max, il ne peut pas être *strictement* plus petit que le min).

Pour se faire une idée du nombre exact de comparaisons, on peut essayer de dérouler l'algorithme pour les listes suivantes, et déterminant combien de comparaisons sont effectuées dans chaque cas:

```
L1 = [0,1,2,3,4]
L2 = [4,3,2,1,0]
L3 = [4,0,3,1,2]
```

[ ]:
```

```