TP9

PARCOURIR UNE LISTE

Corrigé

Exercice 1 **Somme et moyenne**

1. On souhaite écrire une fonction qui calcule la somme des éléments d'une liste numérique. On va proposer deux versions.

   (a) Version 1

```
def somme1(liste):
    n = len(liste)        # longueur de la liste
    somme = 0
    for i in range(n) : # i = position
        somme = somme + liste[i]
    return somme
```

   Compléter ce que fait cette fonction pour $liste = [2, 7, 1]$.

i	somme
Initialisation	$somme = 0$
$i = 0$	$somme = somme + liste[0] = 0 + 2 = 2$
$i = 1$	$somme = somme + liste[1] = 2 + 7 = 9$
$i = 2$	$somme = somme + liste[2] = 9 + 1 = 10$

   (b) Version 2. On complètera aussi avec $liste = [2, 7, 1]$.

```
def somme2(liste):
    somme = 0
    for element in liste : # element parcourt directement la liste
        somme = somme + element
    return somme
```

element	somme
Initialisation	$somme = 0$
$element = 2$	$somme = somme + element = 0 + 2 = 2 = 2$
$element = 7$	$somme = somme + element = 2 + 7 = 9$
$element = 1$	$somme = somme + element = 9 + 1 = 10$

   Dans la première fonction, la boucle for parcourt les indices i et dans la deuxième, la boucle for parcourt les éléments.

2. En déduire une fonction `moyenne(liste)` qui renvoie la moyenne des éléments d'une liste numérique.

```
def moyenne(liste):
    somme = somme1(liste) # ou somme2
    return somme/len(liste)
```

Exercice 2 **Déterminer si une valeur est dans une liste ou non.**

L'instruction `valeur in liste` permet de savoir si une valeur est dans une liste ou non.

1. À l'aide de la fonction randint de la bibliothèque random, créer une liste aléatoire de 10 nombres entre 1 et 20.

```
from random import randint
L = [ randint(1,20) for k in range(10)]
print(L)
```

2. Déterminer si 10 est dedans et si oui, à quelle position.

```
i = 0  # position
while i < len(L) and L[i] != 10 :
    i = i+1
if i < len(L) : # si 10 est dedans, la boucle s'est arrêtée
                # avant de dépasser la fin de la liste
    print(True, ", position = " , i)
else :
    print(False)
```

3. Seconde version : avec une fonction et for

```
def appartient(x,L) :
    '''appartient(x : nombre, L: liste) -> bool
    True si x appartient à L et False sinon (pas de position) '''
    for i in range(len(L)) :
        if L[i] == x :
            return True  # arrêt de la fonction
    # si on arrive après la boucle, x n'appartient pas à L
    return False

# version 2
def appartient(x,L) :
    '''appartient(x : nombre, L: liste) -> bool
    True si x appartient à L et False sinon (pas de position) '''
    for element in L :
        if element == x:
            return True  # arrêt de la fonction
    # si on arrive après la boucle, x n'appartient pas à L
    return False

>>> L = [4, 16, 10, 10, 4, 7, 10, 17, 1, 17]
>>> appartient(10,L)
True

>>> L = [16, 14, 9, 16, 17, 19, 8, 16, 15, 15]
>>> appartient(10,L)
False
```

Exercice 3 Minimum et maximum

1. Commençons par le maximum. Compléter le tableau suivant permettant de déterminer le maximum de $L = [4, 1, 7, 8, 3]$.

Étape	**élément regardé**	**max provisoire** M
Initialisation	L[0] = 4	M = 4
i=1	L[1] = 1	M = 4 car $1 \leqslant 4$
i=2	L[2] = 7	M = 7 car $7 > 4$
i=3	L[3] = 8	M = 8 car $8 > 7$
i=4	L[4] = 3	M =8 car $3 \leqslant 8$

2. Écrire alors une fonction `maximum(L)` qui déterminer le maximum d'une liste numérique non vide L.

```
def maximum(L):
    assert L != [] # on vérifie que L est non vide
    M = L[0]  # initialisation, M = maximum
    for i in range(1, len(L)) : # on parcourt la liste
        if L[i] > M : # si on tombe sur un élément > M, on change M
            M = L[i]
    return M
```

Remarque : si L[i] $\leqslant$ M, on ne change rien, donc pas besoin de mettre un else.

```
# version 2
def maximum(L):
    assert L != [] # on vérifie que L est non vide
    M = L[0]  # initialisation, M = maximum
    for element in L :
        if element > M :
            M = element
    return M
```

3. Programmer de même le minimum.

```
def minimum(L):
    assert L != [] # on vérifie que L est non vide
    m = L[0]  # initialisation, m = minimum
    for i in range(1, len(L)) :
        if m > L[i] :
            m = L[i]
    return m

#version 2
def minimum(L):
    assert L != [] # on vérifie que L est non vide
    m = L[0]  # initialisation, m = minimum
    for element in L :
        if element < m :
            m = element
    return m
```