Devoir surveillé n° 1 – Correction

Problème 1 : Corriger les erreurs

1. La fonction `appartient` proposée n'est pas correcte car elle ne teste que le premier élément de la liste `L`, c'est-à-dire `L[0]`. Si cet élément est égal à `e`, alors la fonction renvoie `True` comme attendu. En revanche, si `L[0]` est différent de `e`, la fonction renvoie `False`, ce qui est incorrect : ce n'est qu'après avoir parcouru tous les éléments de la liste que l'on peut affirmer que `e` ne s'y trouve pas. Le `return False` doit donc être déplacé en dehors de la boucle `for`. Par ailleurs, la fonction `appartient` proposée ne renvoie rien lorsque la liste `L` est vide. En plaçant le `return False` en dehors de la boucle, la fonction renvoie alors `False` dans un tel cas.

Une version corrigée peut être :

```
1 def appartient(e, L): # teste si e est dans L
2     for i in range(len(L)):
3         if e == L[i]:
4             return True
5     return False
```

2. On note plusieurs erreurs également dans la fonction `doublon` proposée. Une première erreur vient du `return False`, ligne 5. Comme pour la fonction `appartient`, le `return False` doit être placé en dehors des *deux* boucles, car c'est uniquement après avoir testé *tous* les couples d'éléments possibles que l'on peut affirmer qu'il n'y a pas de doublon. Par ailleurs, la sortie anticipée n'est possible que lorsqu'on trouve deux éléments égaux (d'indice distinct). Pour corriger cette sortie anticipée, on peut remplacer les lignes 4 et 5 par :

```
if i != j and L[i] == L[j]:
    return True
```

où la première condition est indispensable (sinon la fonction renverra toujours `True`). Enfin, en plaçant les `return` dans la boucle, la fonction proposée ne renvoie rien lorsque la liste est vide, alors qu'on s'attendrait plutôt à ce qu'elle renvoie `False`.

Une version corrigée peut donc être :

```
1 def doublon(L): # teste si L contient 2 fois le même élément
2     for i in range(len(L)):
3         for j in range(len(L)):
4             if i != j and L[i] == L[j]:
5                 return True
6     return False
```

Notons toutefois que cette version n'est pas optimale, car les couples d'éléments sont tous testés deux fois. On peut améliorer le temps de calcul en ne testant qu'une seule fois chaque couple :

```
1 def doublon(L): # teste si L contient 2 fois le même élément
2     for i in range(len(L)):
3         for j in range(i+1, len(L)):
4             if L[i] == L[j]:
5                 return True
6     return False
```

Problème 2 : Statistiques

1.

```
def moyenne(L):
    if len(L) == 0: return 0
    somme = 0
    for e in L:
        somme += e
    return somme/len(L)
```

2. En console (mode interactif), on peut tester la fonction précédente avec l'instruction `moyenne([3, 1, -1, 5])`. Dans l'éditeur de scripts, il faut rajouter l'instruction `print` :

```
L = [3, 1, -1, 5]
print(moyenne(L))
```

3. Une première possibilité astucieuse consiste à utiliser la propriété suivante : Si X désigne une série statistique et si $m = \overline{X}$ désigne la moyenne de X, alors la variance vérifie $\overline{(X-m)^2}$.

```
def variance(L):
    m = moyenne(L)
    Y = []
    for e in L:
        Y.append((e-m)**2)
    return moyenne(Y)
```

On peut également utiliser la version suivante :

```
def variance(L):
    if len(L) == 0: return 0
    m = moyenne(L)
    somme = 0
    for e in L:
        somme += (e-m)**2
    return somme/len(L)
```

4.

```
def n_inferieurs(L, x):
    nombre = 0
    for e in L:
        if e < x:
            nombre += 1
    return nombre
```

5.

```
def mediane_impair(L):
    p = len(L)//2
    for e in L:
        if n_inferieurs(L, e) == p:
            return e
```

6.

```
1  def mediane_pair(L):
2      p = len(L)//2
3      xp, xp1 = 0, 0 # initialisation
4      for e in L:
5          if n_inferieurs(L, e) == p-1:
6              xp = e
7          elif n_inferieurs(L, e) == p:
8              xp1 = e
9      return (xp+xp1)/2
```

7.

```
1  def mediane(L):
2      if L == []: return 0
3      if len(L)%2 == 0:
4          return mediane_pair(L)
5      else:
6          return mediane_impair(L)
```

8.

```
1  def mediane_tri(L):
2      L.sort()
3      p = len(L)//2
4      if len(L)%2 == 1:
5          return L[p]
6      else:
7          return (L[p-1]+L[p])/2
```

Problème 3 : Maximum d'une liste

1.

```
1  def indice_maximum(liste):
2      """renvoie la maximum d'une liste de nombres
3      et son indice dans la liste"""
4      if liste == []:
5          return None
6      indice = 0
7      maxi = liste[0]
8      for i in range(1, len(liste)):
9          if liste[i] > maxi:
10             indice = i
11             maxi = liste[i]
12     return maxi, indice
```

2. Il suffit de remplacer la ligne 9 par `if L[i] >= maxi` pour obtenir le plus grand indice.