DS d'informatique n°1 : Algorithmes de Pronote – Corrigé

Noté sur 100 pts ±5 pts pour le soin et la clarté,
puis la note est divisée par 5 pour faire une note sur 20.

L'outil Pronote a été une révolution dans les établissements d'enseignement public. Les élèves et leurs familles ont maintenant accès à leurs notes à tout moment ainsi qu'aux calculs automatisés des moyennes, des rangs et plus encore. Sur le principe, ces algorithmes semblent simples. Voyons un peu cela...

Partie A : Notes des élèves sur un devoir

On suppose que les élèves ont composé un devoir et ont chacun une note. On dispose ainsi d'une liste qui contient toutes les notes (une par élève), typiquement :

```
[ 14, 3.75, 12.5, 7.25, ...  ]
```

On s'intéresse tout d'abord à la moyenne et l'écart-type de la classe. Pour rappel, étant donné n nombres $x_1, \cdots, x_n$, leur écart-type est donné par

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i-\overline{x})^2} \qquad \text{avec } \overline{x} \text{ la moyenne des nombres } x_1, \cdots, x_n$$

1) (4 pts) Écrire une fonction `moyenne` qui à une liste de notes renvoie la valeur moyenne de cette liste.

```
def moyenne(L):
    s=0
    for v in L:
        s = s+v  # on calcule la somme des notes de L

    return s / len(L) # on divise par le nombre de notes
```

2) (2 pt) Quel est l'écart-type de la liste `[ 8, 11, 11 ]` ?

La moyenne vaut $\frac{8+11+11}{3} = \frac{30}{3} = 10$. L'écart-type vaut donc

$$\begin{aligned} &\sqrt{\frac{1}{3}\left[(8-10)^2+(11-10)^2+(11-10)^2\right]} \\ =&\sqrt{\frac{1}{3}(4+1+1)} \\ =&\boxed{\sqrt{2}} \end{aligned}$$

3) (5 pts) Un informaticien en herbe a écrit la fonction `ecartType` suivante, qui à une liste de notes renvoie l'écart-type de cette liste. Cependant, il a fait une ou plusieurs erreurs. Recopier la fonction corrigée en soulignant chaque endroit que vous avez modifié.

```python
def ecartType(L):
    n = len(L)
    M = moyenne(L)  # Moyenne --> moyenne
    L2 = [ ( L[k]-M )**2 for k in range(n) ]  # L(k) --> L[k] et
       range(L) --> range(n)
    S = sum(L2)/n  # L --> L2
    return S**(1/2)  # parenthèses autour de 1/2
```

Avant de rendre les copies, le professeur doit savoir quelle est la meilleure note, voire les trois meilleures.

4) (5 pts) Écrire une fonction `major` qui à une liste de notes renvoie la meilleure note.

```python
def major(L):
    m = L[0]  # sera la valeur du maximum de L
    for v in L:
        if v > m : # si on trouve une valeur plus grande que m
            m = v  # m est mis à jour
    return m
```

5) (8 pts) Écrire une fonction `podium` qui à une liste de notes (ayant au moins trois notes) renvoie une liste contenant les 3 meilleures notes. *Indication* : on pourra utiliser l'instruction `L.remove(a)`, qui retire la première occurrence de la valeur `a` dans la liste `L` (et produit une erreur si `a` n'est pas dans `L`).

```python
def podium(L):
    L2 = L.copy()    # évite de modifier L, donc les effets de bords
    P = []           # liste qui contiendra les trois meilleures notes
    for k in range(3):
        m = major(L2) # on récupère la meilleure note actuelle de L2
        P.append(m)   # on l'ajoute à P
        L2.remove(m)  # et on la retire de L2
    return P
```

On souhaite enfin vérifier que toutes les notes sur un même devoir sont "valides". Une note x est valide si $x \in [0, 20]$ et si x est ou bien un entier, ou bien admet comme partie décimale 0.25, 0.5 ou 0.75.

6) (4 pts) On suppose que `x` est un flottant. Expliquer ce que fait l'instruction `int(x)`.

> L'instruction `int(x)` retourne un entier qui correspond à la troncature de l'écriture décimale de `x`.
> (Ce n'est pas la partie entière de `x`, cf TP correspondant...)

7) (8 pts) Écrire une fonction `validation` qui à un flottant `x` renvoie un booléen selon que la note `x` soit valide ou non.

```python
def validation(x):
    if x>20 or x<0:
        return False
    else:
        y = 4*x # doit être un entier pour que x soit valide
        if y==int(y): # on teste si c'est effectivement un entier
            return True
        else:
            return False
```

Note : on verra plus tard que ce code n'est pas tout à fait exact dans des cas particuliers... Par exemple, si on donne comme argument `x = 0.75+0.3-0.1-0.1-0.1`*, le résultat est incorrect. Il faut se méfier quand on cherche à tester une égalité avec des flottants.*

Partie B : Moyenne d'un élève sur une matière

On regarde maintenant, dans une matière donnée, toutes les notes obtenues par un(e) unique élève. Ces notes sont stockées dans une liste `M`. On cherche à calculer la moyenne de l'élève pour cette matière. Cependant, toutes les notes n'ont pas le même coefficient. On dispose ainsi d'une liste `Coeff` qui a la même taille que la liste `M` : pour chaque indice `i`, la note `M[i]` est pondérée par le coefficient `Coeff[i]`.

8) (2 pts) Déterminer la moyenne pondérée de l'élève lorsque

`M = [ 12, 16, 6 ]` `Coeff = [ 3, 1, 4 ]`

La moyenne pondérée vaut :

$$\frac{12\times 3+16\times 1+6\times 4}{3+1+4}=\frac{36+16+24}{8}=\frac{76}{8}=\boxed{9,5}$$

9) (8 pts) Écrire une fonction `moyMatiere` qui à une liste de notes `M` retourne la moyenne pondérée de l'élève.

```
1  def moyMatiere(M):
2      s = 0  # somme pondérée des notes
3      c = 0  # somme des coefficients
4      n = len(M)
5      for k in range(n):
6          s = s + M[k]*Coeff[k]
7          c = c + Coeff[k]
8      return s/c
```

Pour calculer `c`*, on pouvait aussi écrire* `c = sum(Coeff)`*, mais cette rédaction permet de mieux comprendre la réponse de la question 11.*

10) (5 pts) Quel problème présente la fonction `moyMatiere` telle qu'elle est implémentée ? On discutera en termes de variable locale et de variable globale.

La fonction `moyMatiere` fait appel à une variable globale `Coeff`. C'est un problème car cela nécessite que la variable `Coeff` soit bien définie (et de même taille que `M`) pour que l'instruction `moyMatiere(M)` s'exécute normalement. Pour résoudre ce problème, on peut donc inclure `Coeff` parmi les arguments de `moyMatiere`, en remplaçant la première ligne du code ci-dessus par

```
def moyMatiere(M,Coeff):
```

On considère à présent que l'élève a pu avoir une ou plusieurs absences. En cas d'absence au devoir d'indice `i`, on note `M[i]="abs"` : la note de ce devoir ainsi que son coefficient ne sont pas pris en compte pour le calcul de la moyenne de l'élève. Par exemple, si

`M = [ 12, 16, "abs" ]` `Coeff = [ 3, 1, 4 ]`

alors la moyenne de l'élève est 13.

11) **(10 pts)** Écrire une fonction `moyMatiere2` similaire à `moyMatiere` et qui tient compte des absences.

```
def moyMatiere2(M):
    s = 0  # somme pondérée des notes
    c = 0  # somme des coefficients
    n = len(M)
    for k in range(n):
        if M[k]!="abs":
            s = s + M[k]*Coeff[k]
            c = c + Coeff[k]

    if c==0:   # l'élève a toujours été absent

        print("pas de note, calcul de moyenne impossible")
        return None
    else:
        return s/c
```

Le professeur a également fait 10 devoirs maison pendant l'année. Ces devoirs sont notés par des lettres majuscules, allant de `A` à `E`. Ainsi, pour chaque élève, on dispose également d'une chaine de caractères `DM` de taille 10 qui contient les lettres associés à ces devoirs.

12) **(5 pts)** Écrire une fonction `comptage` qui à une chaine de caractères `S` et à un caractère `c` retourne le nombre d'occurrences de `c` dans `S`.

```
def comptage(S,c):
    N = 0 # compte le nombre d'occurrences
    for v in S:
        if v==c:
            N = N+1
    return N
```

Un bon professeur cherche à encourager les élèves qui fournissent un travail sérieux et régulier. Un élève est un "chouchou" du professeur s'il a obtenu au moins six `A` sur les dix devoirs maison. Dans ce cas, sa moyenne est augmentée de 10% (sans pour autant dépasser 20).

13) **(8 pts)** Écrire une fonction `moyFinal` qui à une liste de notes `M` et une liste de lettres `DM` retourne la moyenne.

```
def moyFinal(M,DM):
    moy = moyMatiere2(M) # calcule la moyenne sans compter les DM

    if comptage(DM,"A")>=6: # Si au moins 6 DM avec A
        moy = moy * 1.1

    if moy>20:
        moy = 20 # on ramène la note à 20 si elle dépasse 20

    return moy
```

Partie C : rang d'un élève sur une matière

On suppose qu'on a stocké les moyennes des élèves pour une matière donnée dans une liste `MOY`. De plus, on assigne à chaque élève un indice : `MOY[i]` est ainsi la moyenne de l'élève d'indice `i`.

14) **(5 pts)** Écrire une fonction `position(L,a)` qui retourne la position de la première occurrence de la valeur `a` dans la liste `L`.

```
def position(L,a):
    n = len(L)
    for i in range(n):
        if L[i]==a:
            return i
    return None # Si a n'est pas dans la liste, on ne retourne rien
```

On suppose qu'on dispose d'une fonction `tri` qui prend en argument une liste `L` et retourne une liste de même taille avec les valeurs triées par ordre croissant.

15) **(11 pts)** Écrire une fonction `mediane(MOY)` qui retourne la médiane des notes contenues dans `MOY`, ainsi que les notes inférieures à cette médiane. Rappel : la médiane est le plus petit réel m tel qu'au moins 50% des notes soient inférieures à m.

```
def mediane(MOY):
    T = tri(MOY) # T contient les notes par ordre croissant
    n = len(T)   # nombre de notes

    if n%2==0:
        mediane = T[n//2-1] # cf plus bas
    else:
        mediane = T[(n+1)//2-1] # cf plus bas

    L = []  # liste des notes inférieures à la médiane
    for v in T:
        if v <= mediane:
            L.append(v)

    return mediane, L
```

Déterminer la médiane va dépendre de la parité de n.

- Si n est pair, les $\frac{n}{2}$ premières notes de `T` sont `T[0],...,T[n//2-1]`. La médiane vaudra donc bien `T[n//2-1]`.
- Si n est impair, il faut les $\frac{n+1}{2}$ premières notes de `T` (pour en avoir plus de 50%), càd `T[0],...,T[(n+1)//2-1]`. Dans ce cas, la médiane vaut donc `T[(n+1)//2-1]`.

Note : on fait une division entière avec "`//2`" pour que le résultat soit un entier. Sinon le slice ne fonctionne pas.

16) **(10 pts)** Écrire une fonction `rang(MOY,i)` qui retourne le rang de l'élève numéro `i` dans la liste des moyennes `MOY`. En cas d'ex-aequo, on retournera toujours le rang le plus bas. Par exemple les instructions `rang( [ 14, 11, 11 ] , 2 )` et `rang( [ 14, 11, 11 ] , 1 )` retourneront 3.

```
def rang(MOY,i):
    note = MOY[i] # la note dont on cherche le rang

    T = tri(MOY) # on trie MOY par ordre croissant
    T = T[::-1]  # on trie MOY par ordre décroissant
    # la note d'indice 0 de T correspond alors au rang 1,
    # la note d'indice 1 de T correspond alors au rang 2, etc.

    n = len(T)
    for i in range(n):
        if T[i] >= note:
            j = i
    # à ce stade, j est le plus grand indice tel que T[j] == note
    # ainsi, on retournera le rang le plus bas en cas d'ex aequo

    rang = j+1 # décalage de 1 entre rang et indice, cf plus haut

    return rang
```

Note : il y avait d'autres manières de faire, certaines plus concises, mais également moins claires.