À rendre mercredi 3 octobre 2025

Devoir Maison n°2 : Dictionnaires et compagnies – *Corrigé*

Remarques

Ne recopiez pas les spécifications de l'énoncé.
L'énoncé écrit `moyenne(d:dict)->float:`, sur votre copie, écrivez `def moyenne(d):`

I Rappels de 1re année

II Exercices

Exercice n°1 Fonctions élémentaires

R1. On suppose disposer d'un dictionnaire qui contient les températures à Montélimar à 8h00.
Un exemple et extrait de ce dictionnaire est donné ici :

```
{'J1':-10,'J2':-9,'J3':-4,'J4':0,'J5':-1,'J6':4,'J7':-5,'J8':1,'J9':-2}
```

Écrire une fonction `moyenne(d:dict)->float` qui prend en argument un dictionnaire `d` du type de celui défini ci-dessus et qui renvoie la valeur moyenne des températures.

Solution:

```python
def moyenne(d:dict)->float:
    """
    renvoie la moyenne des valeurs du dictionnaire
    """
    S=0
    for c in d: # parcours des clés de d
        S=S+d[c] # on ajoute la température de la clé c
    return S/len(d)
```

R2. Étant donné un dictionnaire dont les clés sont les noms des élèves et les valeurs sont les listes des notes obtenus par l'élève. Écrire la fonction qui renvoie un autre dictionnaire dont les clés sont le nom des élèves et les valeurs la moyenne de leur note `nom_moy(d:dict)->dict`.
Par exemple :
`nom_moy({'Louise':[12,15,12],'Gabriel':[15,17,16],'Léon':[8,18,10],'Alex':[9,14,13]})` renverra `{'Louise':13,'G`
Indication : On pourra commencer par programmer une fonction intermédiaire.

Solution:

```python
def moyenne_liste(L):
    """
    renvoi la moyenne des éléments de la liste L
    """
    M=0
    for x in L:
        M=M+x
    return M/len(L)
def nom_moy(d):
    d_moy={} # dictionnaire des moyennes
    for c in d: # parcours des clés de d
        d_moy[c]=moyenne_liste(d[c]) # calcul de la moyenne des valeurs associées à c (d[c] est la liste des éléments)
    return d_moy
```

R3. Écrire la fonction `mind(d:dict)->float` qui prend en argument un dictionnaire du type du précédent, et qui renvoie la valeur minimale.
On pourra utiliser `float('inf')` qui donne « l'infini ».

Solution:

Remarques

N'initialisez pas le minimum à 0, car si le dictionnaire ne comporte que des nombres strictement positifs, c'est 0 qui sera renvoyé et non la valeur minimale.
Le minimum ne peut pas être initialisé à `d[0]` qui sous-entendrait l'existence d'une clés `0`, ce qui n'est a priori pas le cas...
Attention les éléments d'un dictionnaire sont repérés par leurs clés, et non par leurs rangs.

```python
def mind(d)
    m=float('inf') # initialisation de la valeur minimale
    for c in d: # parcours du dictionnaire
        if d[c]<m: # valeur de clé c inférieure au min local
            m=d[c] # nouveau minimum
    return m
```

R4. Écrire la fonction `cle_min(d:dict)->str` qui prend en argument un dictionnaire du type du précédent, et qui renvoie la clé de valeur minimale.

Solution:

```python
def cle_max(d):
    m=float('inf') # valeur minimale
    for c in d: # parcours du dictionnaire
        if d[c]<m: # nouveau min
            m=d[c]
            cmin=c # mise à jour de la clé
    return cmin
```

On appelle `nb_oc(L:list[int])->dict` une fonction prenant comme argument une liste d'entiers naturels `L` et renvoyant un dictionnaire `D` dont les clefs sont les éléments de `L` et la valeur associée est le nombre d'occurrences de cet élément dans `L`.

R5. Que renvoie la commande `nb_oc([4,0,0,1,2,0,4])` ?

Solution:

Remarques

Une phrase est toujours la bienvenu ! Ne « balancez » pas un résultat sans rien d'autre.

`nb_oc([4,0,0,1,2,0,4])` renvoie `{4:2,0:3,1:1,2:1}`

R6. Écrire en Python cette fonction `nb_oc(L:list[int])->dict`.

Solution:

```python
def nb_oc(L):
    D = {}
    for x in L:
        if x in D : # x a déjà été rencontré et placé dans D
            D[x] = D[x] + 1
        else : # 1ère rencontre de x
            D[x] = 1
    return D
# ou :
def nb_oc(L):
```

```
    D = { x : 0 for x in L }  # création des clés avec les éléments de L
    for x in L :
        D[x] = D[x] + 1 # toutes les clés sont déjà présentes
    return D
```

Exercice n°2 Suite de Robinson ♪ ♪

On appelle `nb_oc(L:list[int])->dict` une fonction prenant comme argument une liste d'entiers naturels `L` et renvoyant un dictionnaire `D` dont les clefs sont les éléments de `L` et la valeur associée est le nombre d'occurrences de cet élément dans `L`.

R1. Que renvoie la commande `nb_oc([4,0,0,1,2,0,4])` ?

Solution:
`nb_oc([4,0,0,1,2,0,4])` renvoie `{4:2,0:3,1:1,2:1}`

R2. Écrire en Python cette fonction `nb_oc(L:list[int])->dict`.

Solution:

```
def nb_oc(L):
    D = {}
    for x in L:
        if x in D : # x a déjà été rencontré et placé dans D
            D[x] = D[x] + 1
        else : # 1ère rencontre de x
            D[x] = 1
    return D
```

R3. Déterminer la complexité de votre fonction `nb_oc(L:list[int])->dict`. Justifier.

Solution: On parcourt une fois la liste : $len(L)$ itérations. Au sein de ces itérations, il y a un test d'appartenance au dictionnaire, qui est de coût en $O(1)$ constant indépendant de la taille du dictionnaire. Ainsi la complexité est linéaire en $O(len(L))$.

R4. Si votre fonction `nb_oc` n'était pas de complexité linéaire, la réécrire afin qu'elle le soit.

Solution: rien à faire

Soit A une liste d'entiers naturels. On définit la suite de Robinson $(L_n)_{n\in\mathbb{N}}$ associée à la suite A par récurrence comme suit :

- $L_0 = A$.
- Si L_n est construite, alors :
  - on détermine $T_n = \texttt{nb_oc}\,(L_n)$.
  - on détermine I_n la liste des clefs non nulles de T_n classée par ordre strictement croissant.
  - si $I_n = [i_1, \ldots, i_r]$, alors $L_{n+1} = [T\,[i_r]\,, i_r, \ldots, T\,[i_1]\,, i_1]$.

Par exemple si $A = [4, 4, 1, 2]$, alors :

- $L_0 = [4, 4, 1, 2]$
- $L_1 = [2, 4, 1, 2, 1, 1]$ (il y a deux « 4 », un « 2 » et un « 1 » dans la liste L_0).
- $L_2 = [1, 4, 2, 2, 3, 1]$ (il y a un « 4 », deux « 2 » et trois « 1 » dans la liste L_1).

R5. On donne $A = [2, 0, 4, 1, 3, 3, 2, 3, 1, 1]$. Déterminer L_3 et L_{2024}.

Solution:

Remarque
Un minimum de justification est appréciée !

1. — $L_0 = A = [2, 0, 4, 1, 3, 3, 2, 3, 1, 1]$
   — $T_0 = nb_oc(L_0) = 0 : 1, 1 : 3, 2 : 2, 3 : 3, 4 : 1$
   — $I_0 = [1, 2, 3, 4]$
   — $L_1 = [1, 4, 3, 3, 2, 2, 3, 1]$
2. — $L_1 = [1, 4, 3, 3, 2, 2, 3, 1]$
   — $T_1 = nb_oc(L_1) = 1 : 2, 2 : 2, 3 : 3, 4 : 1$
   — $I_1 = [1, 2, 3, 4]$
   — $L_2 = [1, 4, 3, 3, 2, 2, 3, 1]$
3. Par récurrence immédiate : $\forall n \geq 1\ L_n = [1, 4, 3, 3, 2, 2, 3, 1]$

R6. On donne $B = [2, 4, 1, 1, 1, 2]$, on suppose que $L_1 = B$, donner toutes les solutions possibles pour L_0.

Solution: La liste B proposée ne peut pas être la liste L_1 d'une suite de Robinson puisque par construction dans une telle liste, les valeurs $i_r, \ldots, i_2, i_1$ dans les cases d'indice impair devraient aller en décroissant.

R7. On donne $C = [2, 4, 1, 0]$. Si l'on suppose que $L_1 = C$, donner toutes les solutions possibles pour L_0.

Solution: Les listes L_0 correspondant à une telle liste L_1 sont toutes les listes constituées de deux « 4 »et d'un « 0 »(et de rien d'autre). Il y a donc trois possibilités : $[0, 4, 4]$, $[4, 0, 4]$ ou $[4, 4, 0]$.

R8. Proposer alors une fonction `rob(A:list[int], n:int)->list[int]` qui prend en arguments une liste d'entiers naturels `A` et un entier naturel `n` et qui renvoie l'élément L_n de la suite de Robinson associée à `A`.

Solution: J'ai besoin de deux fonctions annexes à programmer : le maximum d'une liste (pour pouvoir créer `L`), et la fonction qui inverse une liste.

```python
def max(L):
    M=L[0]
    for i in range(1,len(L)):
        if L[i]>M:
            M=L[i]
    return M
def inverse(L):
    inv = []
    n = len(L)
    for i in range(n):
        inv.append(L[n-1-i]) # inv[0] est L[n-1] ; inv[1] est L[n-2] ...
    return inv
def rob(A,n):
    L = A # L0=A (pas de problème de copie de liste, car L n'est pas modifiée, mais préférer L = copy(A))
    for k in range(n): # on part de L0, donc de n=0
        T = nb_oc(L) # dictionnaire des occurrences
        I = [] # liste des clés
        r = max(L)
        for i in (1,r): # permet d'ajouter les clés par ordre croissant (elles sont dans un ordre quelconque selon l'apparition de la valeur dans L)
            if i in T: # ajout des clés présentes dans T
                I.append(i) # ajoute à i
        I_inv = inverse(I) # on inverse I pour obtenir les indices dans l'ordre décroissant
        # création de L_(n+1) à partir de L_n :
        L = [] # L_(n+1)
        for i in I_inv :
            L.append(T[i]) # occurrence de l'entier i
            L.append(i) # ajoute l'entier i
    return L
```