Eléments de correction (interros python)

Exercice 1. Etant donné une liste L et un objet `v` écrivez une fonction d'entête `def indiceDe(L,V):` qui renvoie le premier indice `i` où `v` figure dans `L`. Si `v` n'est pas dans `L`, on renverra `None`. (Pas de `if v in L` autorisé ici bien sûr)

Solution. Le plus simple est d'utiliser une boucle `for` pour parcourir (par indices) la liste `L`, et de s'arrêter prématurément dès lors que la valeur recherchée est rencontrée :

```
def indiceDe(L, v):
    for i in range(len(L)):
        if L[i] == v: # == et non = bien sûr
            return i
```

Rien à ajouter, car si on parvient à la dernière instruction de la fonction sans rencontrer l'instruction `return` (qui conduit à sortir définitivement de ladite fonction) alors on sort de la fonction et la valeur de retour par défaut est celle qu'on attend dans ce cas, à savoir `None`.

Exercice 2. écrire une fonction d'entête `indicesDe(L, v)` qui admet pour arguments une liste `L`, une valeur `v` et qui renvoie la liste (éventuellement vide) de tous les indices où `v` figure dans `L`. Par exemple :

```
>>> indicesDe([2,1,6,1,3,1], 1)
    [1,3,5]

>>> indicesDe([2, 7, 4], 3)
    []
```

Solution.

```
def indicesDe(L, v):
    L = []
    for i in range(len(L)):
        if L[i] == v:
            L.append(i)
    return L
```

Exercice 3. (*) On dipose pour une balance de cuisine d'un nombre de poids et on a un objet de masse m. Ecrire une fonction d'entête `def nbDecomp(m, P):` qui étant donnés m et `P` renvoie le nombre de manières d'égaler la masse m avec une distribution de ces poids.

Par exemple : si $m=4$ et $P=[1,2]$, ce nombre vaut 0, et avec $m=3$ et $P=[1,2,1]$ ce nombre vaut 2 (si deux poids ont la même masse, on les considère tout de même discernables, si bien que les les décompositions $P[0]+P[1]$ et $P[1]+P[2]$ sont considérées comme différentes). On considèrera que m et les éléments de P sont tous des entiers strictement positifs.

Solution. Le plus simple (on peut sans nul doute faire plus efficace, mais un algorithme efficace n'est pas attendu ici) est une fonction récursive :

```
def nbComp(m, P):
    if m==0:
        return 1
    if len(P)==0:
        return 0
    if P[-1] <= m:
        return nbComp(m-P[-1],P[:-1])+nbComp(m, P[:-1])
    else:
        return nbComp(m, P[:-1])
```

Exercice 4. Etant donné une liste L de nombres (entiers, flottants, peu importe) écrire une fonction d'entête `def minimum(L):` qui en détermine la plus petite valeur.

Solution. A savoir faire impérativement !

```
def minimum(L):
    m = L[0]
    for i in range(1, len(L)):
        if L[i] < m:
            m = L[i]
    return m
```

Exercice 5. Ecrire une fonction d'entête `def indiceMinimum(L):` pour qu'elle renvoie non pas la plus petite valeur de `L`, mais plutôt le premier indice où celle-ci figure dans `L` (exemple : `indiceMinimum([3,1,6,2,1])` devra renvoyer 1.

Expliquer ensuite comment modifier la fonction que vous venez d'écrire pour que ce soit le dernier indice où figure le minimum qui soit renvoyé (de sorte que `indiceMinimum([3,1,6,2,1])` renvoie alors 4 et non plus 1.)

Solution.

```
def indiceMinimum(L):
    i_mini = 0
    for i in range(1, len(L)):
        if L[i] < L[i_mini]:
            i_mini = i
    return i_mini
```

Et pour obtenir le dernier indice, remplacer l'inégalité stricte par une inégalité large.

Exercice 6. (*) On suppose disposer d'un nombre illimité de pièces de monnaies dont les valeurs forment les nombres entiers d'une liste `V`. La plus petite valeur est 1 ce qui indique que tout montant entier est atteignable d'au moins une façon.

On souhaite connaître le plus petit nombre de pièces permettant d'atteindre un montant donné. Par exemple pour les valeurs de pièces 1,3,4 et un montant de 6, alors le nombre optimal de pièces vaut 2.

Ecrire donc une fonction `def minPieces(n, V):` qui renvoie le plus petit nombre de pièces dont les valeurs figurent dans `V` permettant de décomposer la somme `n`.

Solution. Une fonction récursive assez peu efficace :

```
def minPieces(n, V):
    if n == 0:
        return 0
    mini = n # il est possible de décomposer n en n pièces de valeur 1 (perfectible, sans doute)
    for v in V:
        if v <= n:
            m = minPieces(n-v, V) + 1
            if m < mini:
                mini = m
    return mini
```

Une solution meilleure en termes de complexité : on utilise de la programmation dynamique : l'idée est de calculer pour toutes les sommes jusqu'à n leur meilleure décomposition, mais de ne le faire qu'une seule fois. On peut obtenir la même chose en reprenant l'algorithme récursif précédent et la mémoïzation (le fait de garder trace de toutes les valeurs calculées pour ne pas refaire le travail si la même valeur est demandée une seconde fois) :

```
def minPieces(n, V):
    mini = [0, 1]
    for i in range(2, n):
        mini_i = i
        for v in V:
            if v <= i:
                if mini[i-v] + 1 < mini_i:
                    mini_i = mini[i-v] + 1
        mini.append(mini_i)
    return mini[n]
```

Exercice 7. On rappelle que, étant donnés `a` et `b` des entiers, tels que `b>0`, alors `a % b` (qui peut se lire « a modulo b ») est le reste de la division euclidienne de `a` par `b`.

Ecrire alors une fonction d'entête `sommeDiv(n)` qui prend en argument un naturel `n` et renvoie la somme de ses diviseurs propres (ceux qui diffèrent de `n`). Par exemple : `sommeDiv(8)` devra renvoyer 7 car les diviseurs propres de 8 sont 1,2 et 4.

Solution.

```
def sommeDiv(n):
    s = 0
    for i in range(1, n):
        if n % i == 0:
            s = s + i
    return s
```

Exercice 8. Ecrire deux fonctions d'entête `factorielle(n)` qui admettent un entier naturel `n` en paramètre, et qui renvoient la factorielle de `n`, l'une itérative (boucle for ou while comme vous préférez) et l'autre récursive.

Solution.

```
def factorielle(n):
    p = 1
    for i in range(2, n+1):
        p = p * i
    return p
```

Version récursive :

```
def factorielle(n):
    if n == 0:
        return 1
    return n * factorielle(n-1)
```

(A noter que si on invoque cette dernière fonction avec autre chose qu'un entier positif, le plantage est assuré, mais après tout, il ne nous est pas demandé de nous assurer que `n` est bien positif ici alors...)