Informatique — MPSI/PCSI

Correction du TP n° 4

Exercice 4.1 (Mutable *or not* ?)

1. Comme l'opération `M=L` recopie sur `M` l'adresse du pointeur de `L`, alors modifier `M[2]` ou `L[2]` revient au même et donc `L[2]` renvoie `666`.
2. L'opération `[e for e in L]` permet de recopier les éléments de `L` sans les lier. Ainsi, si `M` est modifiée, les éléments de `L` ne le sont pas et donc `L[2]` renvoie bien `2`.
3. La recopie des éléments par compréhension ne passe pas au-dessus de la notion de pointeur. Ainsi, lorsque les éléments d'une liste sont des éléments mutables, alors ce sont leurs pointeurs qui sont recopiés et non leurs valeurs. On observe ici que la deuxième liste de `L` et la liste `B` ont maintenant `666` en dernier élément.
4. Lorsque la recopie est faite avec la fonction `deepcopy` de la bibliothèque `copy`, tous les éléments mutables sont eux-mêmes recopiés et aucun pointeur n'est lié. Ainsi, modifier `M` n'a aucun effet sur `L` ou `B` (ni `A`, ni `C`).
5. Comme c'est trois fois le pointeur de `L` qui est saisi pour constituer la liste `M`, modifier `L[2]` induit la modification de chaque sous-liste de `M`. Donc `[e[2] for e in M]` renvoie `[666, 666, 666]`.
   Comme pour toute séquence (liste, tuple, chaine de caractères), l'opération `*n` est une opération de réplication n fois. Or répliquer un élément mutable ne recopie que son pointeur. Ainsi, pour toute liste `L`, on a

   ```
   >>> [L,L,L]==3*[L]
   True
   ```

   Attention, `3*L` concatène 3 fois la liste `L` et ne recopie donc aucun pointeur au rang 1.
6. Comme par réplication, `[[]]*n` fait n fois référence au même pointeur et donc modifier une des n séquences les modifie toutes. Ainsi, `L` renvoie `[[2], [2], [2]]`.

Exercice 4.2 (Mutation)

1. Pour nettoyer la liste `L` de tous ses `a`, il est possible d'utiliser la fonction `pop` qui permet de supprimer un élément. Par contre on veillera à ce l'on puisse continuer de parcourir la liste (initialement de longueur n) même si on a déjà supprimé $k > 0$ valeurs. En partant de la fin, on peut écrire :

   ```
   def Nettoyer(L,a):
       n=len(L)
       for i in range(n):
           if L[n-1-i]==a:
               L.pop(n-1-i)
   ```

2. Pour permuter deux éléments d'une séquence, il suffit de sauvegarder de façon temporaire un des deux objets pour réaliser la permutation. Tenant compte du fait que l'on agit sur la liste elle-même (pointeur), il n'est pas nécessaire de renvoyer quoi que ce soit.

   ```
   def Permuter(L,i,j):
       tmp=L[i]
       L[i]=L[j]
       L[j]=tmp
   ```

   La version suivante utilise le *swap* natif des objets en Python.

   ```
   def Permuter(L,i,j):
       L[i],L[j]=L[j],L[i]
   ```

3. Pour inverser une liste de longueur n, il suffit d'utiliser un éléments de plus puis, en partant de l'avant dernier, copier un à un les termes à la fin puis les supprimer selon le principe décrit par les schémas ci-dessous.



C'est ce que fait le code suivant.

```
def Inverser(L):
    n=len(L)
    for k in range(n-1):
        L.append(L[n-2-k])
        L.pop(n-2-k)
```

où `L.pop(n-2-k)` est (quasiment) équivalent à `del(L[n-2-k])`. Notez, que nous aurions aussi pu utiliser le *swap* de la fonction `Permuter` :

```
def Inverser(L):
    n=len(L)
    for k in range(n//2):
        Permuter(L,k,n-1-k)
```

ou étendre la liste avec son inverse obtenue par *reverse slicing* puis supprimer la portion initiale (en ne gardant en place que le dernier élément) :

```
def Inverser(L):
    n=len(L)
    L.extend(L[n-1::-1])
    del(L[:n-1])
```

où `L.extend(L[n-1::-1])` est (quasiment) équivalent à `L=L+L[n-1::-1]`. Notons toutefois que cette solution coute temporairement le double en RAM pour l'objet `L`, ce qui parfois peut être un problème.

4. Pour vérifier qu'une séquence est égale à son inverse, il suffit de comparer successivement tous les éléments jusqu'au milieu. Dans le cas d'une séquence de nombre impairs d'éléments n, on fera ainsi $\left\lfloor \frac{n}{2} \right\rfloor$ comparaisons, $n/2$ sinon. Le code suivant convient.

```
def palindrome(L):
    n=len(L)
    k=0
    while k < n//2 and L[k]==L[n-1-k]:
        k+=1
    return(k==n//2)
```