TP opérations sur les colonnes d'un vecteur d'entiers

On veut écrire un algorithme qui transforme un vecteur d'entiers $(x_1, \dots, x_n)$ en $(d, 0, \dots, 0)$ par opérations sur les colonnes.

Les opérations en question sont de type $C_i + a * C_j \to C_i$, $C_i \leftrightarrow C_j$, $C_i \to -C_i$.

On choisit d'encoder un n-uplet $(x_1, \dots, x_n)$ par une liste [x1,x2,]. On pourrait aussi choisir le type vecteur ou matrice (avec une seule ligne). Voir pour une documentation les commandes vector? et matrix?

Ex 1. Choisir un encodage des opérations puis définir une fontion qui prend un vecteur u et une opération op et qui rend le vecteur transformé par l'opération.

Ecrire de même une fontion qui prend un vecteur et une liste d'opérations et rend le vecteur transformé par la composée des opérations.

Tester ces fonctions.

Ex 2. Ecrire une fonction qui prend un vecteur d'entiers et rend une liste d'opérations telle que le vecteur transformé par la liste d'opérations soit de type $(d, 0, \dots, 0)$.

Tester cette fonction.

Corrigé

1. On choisit d'encoder la première opération par la liste [i,j,a], la seconde par [i,j], la troisième par [i]. La longueur de la liste permet de distinguer les types d'opérations.

Rq. On aurait pu aussi indiquer dans l'encodage de l'opération le type de l'opération, par exemple ['ajout',i,j,a], ['echange',i,j], ['signe',i].

Documentation sur les listes dans Python : faire une recherche sur internet python listes' ; voir des exemples d'instruction dans les feuilles de TP de 2016-17, regarder cette documentation (utilisations des listes en Python)

Un cours sur la programmation en Python, un autre spécialisé à Sagemath.

Attention : l'indexation des éléments d'une liste commence à 0, ainsi l[i] désigne le i+1ème élément de la liste l.

```
def opere(o,l):
    if len(o)==1:l[o[0]-1]=-l[o[0]-1];return(l)
    elif len(o)==2:c=l[o[0]-1];l[o[0]-1]=l[o[1]-1];l[o[1]-1]=c;\
   return(l)
    else:l[o[0]-1]=l[o[0]-1]+o[2]*l[o[1]-1];return(l)
```

```
#test
l=[-4,6,3]
print l
opere([3,1,1],l)
opere([1,3],l)
opere([1],l)
```

[-4, 6, 3]

[-4, 6, -1]

[-1, 6, -4]

[1, 6, -4]

```
#la variable globale l semble modifiée par la fonction opere où l \
   est pourtant une variable locale.
#en fait ce n'est pas la variable l qui est modifié mais ce vers \
   quoi elle pointe. Test ci-dessous :
def modif(l):
    l=0
    return(l)
modif(l)
print l
```

0

[1, 6, -4]

```
#correction de la fonction opere pour que l'argument reste inchangé
def opere(o,ll):
    l=deepcopy(ll)
    if len(o)==1:l[o[0]-1]=-l[o[0]-1];return(l)
    elif len(o)==2:c=l[o[0]-1];l[o[0]-1]=l[o[1]-1];l[o[1]-1]=c;\
   return(l)
    else:l[o[0]-1]=l[o[0]-1]+o[2]*l[o[1]-1];return(l)
```

```
def oopere(oo,l):
    if oo==[]:return(l)
    else:return(oopere(oo[1:],opere(oo[0],l)))
#definition récursive !
#oo[1:] est la liste oo privée de son premier élément oo[0]
```

```
#test
l=[-4,6,3]
oo=[[3,1,1],[1,3],[1],[2,1,-6],[3,1,4]]
print l,'devient',oopere(oo,l)
```

[-4, 6, 3] devient [1, 0, 0]

2. L'algorithme retenu est : repérer dans la liste l le plus petit élément en valeur absolue, l'utiliser comme pivot ("algorithme du pivot") s'il divise tous les autres, le diminuer suivant l'algorithme d'Euclide et utiliser une programmation récursive sinon.

```
#essai
l=[-4,6,-3,0,4]
lplus=[abs(x) for x in l if x!=0] #liste définie par "compréhension"
#taper lplus. suivi de la touche <tab> pour voir les méthodes \
   disponibles pour l'objet lplus. choix d'une telle méthode suivi \
   de ? pour voir la documentation sur cette méthode.
lplus.index?
i=lplus.index(min(lplus))
print l,i,l[i]
```

```
Docstring :
L.index(value, [start, [stop]]) -> integer -- return first index of
value. Raises ValueError if the value is not present.
```

[-4, 6, -3, 4] 2 -3

```
def listeop(l):
    lplus=[abs(x) for x in l]
    if max(lplus)==0:return([])
    else:
        aplus=min(x for x in lplus if x!=0)
        i0=lplus.index(aplus)
        a=l[i0]
        I=[i for i in range(len(l)) if l[i]%a != 0]
        if I==[]:#pivot suivant l[i0]
            op=[[i+1,i0+1,-l[i]/a] for i in range(len(l)) if i!=i0\
]+[[1,i0+1]] #la dernière opération est l'identité si i0=0
            print 'pivot : l=',l,'i0=',i0+1,'op=',op #pour contrôle \
de l'exécution du programme
            return(op)
        else:#division euclidienne de l[j] par l[i0]
            j=min(I)
            op=[j+1,i0+1,-l[j]//a]
            l1=opere(op,l)
            print 'euclide : l=',l,'i0=',i0+1,j+1,'op=',op #pour \
contrôle de l'exécution du programme
            return([op]+listeop(l1))
```

```
#test
l=[0,-4,6,3]
#lcopy=deepcopy(l)
oo=listeop(l)
print
print 'l=',l
print 'liste op =',oo
print 'l transformé =',oopere(oo,l)
```

euclide : l= [0, -4, 6, 3] i0= 4 2 op= [2, 4, 1]
pivot : l= [0, -1, 6, 3] i0= 2 op= [[1, 2, 0], [3, 2, 6], [4, 2, 3], [1, 2]]

l= [0, -4, 6, 3]

liste op = [[2, 4, 1], [1, 2, 0], [3, 2, 6], [4, 2, 3], [1, 2]]

l transformé = [-1, 0, 0, 0]