TP 3- Proposition de solutions

Solution 1

```
def nb_neg(L):
    compteur = 0
    for e in L:
        if e<=0:
            compteur = compteur+1
    return compteur

def pos_neg(L):
    pos = []
    for i,e in enumerate(L):
        if e<=0:
            pos.append(i)
    return pos
```

Solution 2 Listes définies par compréhension

On retiendra l'utilisation de listes définies par compréhension.

```
L1=list(range(0,1000,3))
L1=[3*k for k in range(1000//3+1)]
L1=[j for j in range(1001) if j%3==0]

L2=list(range(1000,-1,-2))
L3=[1/k**2 for k in range(1,13)]
S3= sum(L3)
f3=lambda n:sum([1/k**2 for k in range(1,n+1)])
S4=sum([np.cos(k)/k for k in range(1,1000,2)])
pair=lambda L:[e for e in L if e%2==0]
nb_beg=lambda L:sum([1 for e in L if e<=0])
```

Solution 3 Minimum - Tri

Minimum d'une liste

```
def min(L):
    '''
    Entree : L une liste de nombres reels
    Sortie : le minimum
    '''
    valeur=L[0]
    for i in range(1,len(L)):
        if L[i]<valeur:
            valeur=L[i]
    return valeur
```

Position d'un minimum

```
def pmin(L):
    '''
    Entree : L une liste de nombres reels
    Sortie : une position et la valeur du minimum
    '''
    position=0
    for i in range(1,len(L)):
        if L[i]<L[position]:
            position=i
    return position,L[position]
```

Tri d'une liste

```
def tri(L):
    '''
    Entree : L une liste de nombres reels
    Sortie : T la liste triee
             (methode par selection)
    '''
    LL,T=list(L),[] # copie de L, liste vide
    while len(LL)>0:
        p,v=pmin(LL)
        T.append(v)
        LL.pop(p)
    return T
```

Description du déroulement du tri appliqué à la liste `[5,-2,-4,3,-4,-1]` :

boucle	L	pmin(L)	T
X	[5,-2,-4,3,-4,-1]	X	[]
1	[5,-2,3,-4,-1]	2,-4	[-4]
2	[5,-2,3,-1]	3,-4	[-4,-4]
3	[5,3,-1]	1,-2	[-4,-4,-2]
4	[5,3]	2,-1	[-4,-4,-2,-1]
5	[5]	1,3	[-4,-4,-2,-1,3]
6	[]	0,5	[-4,-4,-2,-1,3,5]

La liste ordonnée associée est `[-4,-4,-2,-1,3,5]`.

Solution 4 Extrema - Médiane

Position du min et du max

```
def extremum(L):
    pmin,pmax=0,0
    for i in range(1,len(L)):
        if L[i]>L[pmax]:
            pmax=i
        elif L[i]<L[pmax]:
            pmin=i
    return pmin,pmax
```

Recherche de la médiane

```
1  def mediane(L):
2      LL=list(L) # copie de L
3      while len(LL)>2:
4          pmin,pmax=extremum(LL)
5          if LL[pmin]==LL[pmax]:
6              break
7          else:
8              LL.pop(pmin)
9              if pmin<pmax:
10                 LL.pop(pmax-1)
11             else:
12                 L.pop(pmax)
13     return (LL[0]+LL[-1])/2
```

Attention ! (lignes 9 à 12) L'élimination de deux éléments d'une liste se fait en deux temps ; ainsi, le retrait du premier élément impacte la place des autres éléments dans la liste modifiée. Si l'on peut, il vaut mieux retirer les éléments par position décroissante.

Solution 5 Triangle de Pascal

Triangle de Pascal

```
def Pascal(n):
    L=[1]
    print('n = 0 : [1]')
    for i in range(1,n+1):
        LL=[1]
        for j in range(1,len(L)):
            LL.append(L[j]+L[j-1])
        LL.append(1)
        L=list(LL)
        print('n =',i,':',L)
    return L
```

Coefficient binomial

```
def binome(k,n):
    assert type(k)==int and k>=0,'ENC'
    if k>n:
        return 0
    else:
        L=Pascal(n)
        return L[k]
```

Solution 6 La fonction suivante, retourne une liste contenant les éléments de L sans répétition.

```
def f(L):
    T=[L[0]]
    for e in L:
        if not(e in T): T.append(e)
    return T
```

Solution 7 **Ordre lexicographique**

```
def lexico(L1,L2):
    '''
    Entrees : deux listes L1,L2
    Sortie : True si L1<=L2, False sinon
            (ordre lexicographique)
    '''
    i=0
    while i<len(L1) and i<len(L2) and L1[i]==L2[i]:
        i=i+1
    if i==len(L1):
        return True
    elif i==len(L2):
        return False
    else:
        return L1[i]<L2[i]
```

ou

```
def lexico(L1,L2):
    for i in range(min(len(L1),len(L2))):
        if L1[i]!=L2[i]:
            return L1[i]<L2[i]
    return len(L1)<=len(L2)
```

Solution 8 **Tri par comptage**
L'appel `stat([[2,1,1,0,1,4],4)` retourne `[1,3,1,0,1]`.

Dénombrement par élément

```
def stat(L,n):
    '''
    Entree : L une liste de nombres dans [[0,n]]
    Sortie : C une liste de longueur n+1
            C[k] est le nb d occurences de k dans L
    '''
    C=(n+1)*[0]
    for e in L: C[e]=C[e]+1
    return C
```

La fonction `stat` crée une liste de longueur $n+1$ et parcourt la liste `L` une fois. Le nombre d'affectation est de $n+1+$`len(L)`.

Tri par comptage

```
def tri_comptage(L,n):
    '''
    Entree : L une liste de nombres dans [[0,n]]
    Sortie : T la liste triee
            (methode par comptage)
    '''
    C=stat(L,n)
    T=[]
    for i,e in enumerate(C): T=T+e*[i]
    return T
```

Solution 9 **Tri par comparaison**
L'appel `comparaison([2,0,1,4,2,3,1,0])` retourne `[4,0,2,7,5,6,3,1]`.

Repérage des positions

```
def comparaison(L):
    '''
    Entree : L une liste de nombres reels
    Sortie : C une liste de longueur len(L)
           C[k] est la position de L[k]
           dans la liste triee
    '''
    C=len(L)*[0]
    for i,e in enumerate(L):
        for j in range(0,i):
            if L[j]<=e: C[i]=C[i]+1
        for j in range(i+1,len(L)):
            if L[j]<e: C[i]=C[i]+1
    return C
```

La fonction `comparaison` effectue notant n le nombre éléments de `L` :

- n affectations lors de la création de `C`
- Pour chaque élément de `L`, elle réalise $n-1$ tests
- le nombre global d'affectations est $\frac{(n-1)n}{2}$ car l'incrément +1 est utiliser pour reformer toutes les positions possibles 0, 1, 2, $n-1$ soit $\sum_{k=0}^{n-1} 1$ incréments.

Au total, le nombre d'opérations est : $n+n(n-1)+\frac{(n-1)n}{2}$

Tri par comparaisons

```
def tri_comparaison(L):
    '''
    Entree : L une liste de nombres reels
    Sortie : T la liste triee
            (methode par comparaison)
    '''
    C=comparaison(L)
    T=len(L)*[0]
    for i in range(len(L)): T[C[i]]=L[i]
    return T
```