DS n°1 d'informatique commune

Exercice 1

1.
```
def somme(L) :
    s=0
    for l in L :
        s+=l
    return s
```

2.
```
def moyenne(L):
    if L==[] :
        return 0
    return somme(L)/len(L)
```

3.
```
def centree(L):
    if L==[] :
        return L
    m=moyenne(L)
    return [l-m for l in L]   # on retranche la moyenne de L à chaue élément de L
```

4.
```
def repartition(L) :
    R=[0,0,0]
    m=moyenne(L)
    for l in L :
        if l<m :
            R[0]+=1
        if l==m :     # on pourrait faire un elif mais les conditions sont incompatibles
            R[1]+=1
        if l>m :     # idem
            R[2]+=1
    return R
```

5.
```
def ecarttype(L) :
    M=centree(L)
    M2=[m**2 for m in M]
    return somme(M2)**0.5
```

Exercice 2

1.
```
def appartient(L,a) :
    for l in L :
        if l==a :
            return True
    return False         # on met bien le return False hors de la boucle for
```

2.
```
def occurence(L,a):
    o=0
    for l in L :
        if l==a :
            o+=1
    return o
```

3.
```
def elements(L):
    E=[]
    for l in L :
        if not appartient(E,l):
            E.append(l)
    return E
```

4.
```
def union(L1,L2):
    U=elements(L1)
    for l in L2 :
        if not appartient(U,l):
            U.append(l)
    return U

def intersection(L1,L2):
    E=elements(L1)
    N=[]
    for e in E :
        if appartient(L2,e):
            N.append(e)
    return N
```

5.
```
def diffsym(L1,L2):
    U=union(L1,L2)
    N=intersection(L1,L2)
    D=[]
    for u in U :
        if not appartient(N,u) :
            D.append(u)
    return D
```

6.
```
def elementsbis(L) :
    return [[e,occurence(L,e)] for e in elements(L)]
```

Exercice 3

1.
```
def triee(L) :
    for i in range(len(L)-1):
        if L[i]>L[i+1]:
            return False
    return True         # on met bien le return True hors de la boucle for
```

2. (a)
```
def comptage(L):
    N=[0]*11
    for l in L :
        N[l]+=1
    return N
```

(b)
```
def tricomptage(L):
    N=comptage(L)
    T=[]
    for i in range(11):
        for j in range(N[i]) :
            T.append(i)
    return T
```

3. (a)
```
def indicemax(L):
    i=0
    M=L[0]
    for j in range(len(L)) :
        if L[j]>M :
            i=j
            M=L[j]
    return i
```

(b) Il se trouve entre les éléments d'indice `0` et `len(L)-1-i` (au sens large). C'est-à-dire que c'est un élément de `L1[:len(L)-i]`.

Pour $i = 0$, cela veut dire entre les indices `0` et `len(L)-1`, ce qui donne bien tous les éléments de `L1`.

Pour $i = 1$, cela veut dire entre les indices `0` et `len(L)-2`, ce qui donne bien tous les éléments de `L1` sauf le dernier (dont on sait que c'est le plus grand à ce stade de l'algorithme).

(c)
```
def tri(L) :
    L1=L.copy()
    for i in range(len(L)):
        j=indicemax(L1[:len(L)-i])   #l'indice du (i+1)ème plus grand élément dans L1
        L1[len(L)-i-1],L1[j]=L1[j],L1[len(L)-i-1]    # on échange les éléments d'indice j
              # et l'élément en (i+1)ème position en partant de la fin
    return L1
```

Exercice 4

1. (a)
```
def estpremier(p) :
    for i in range(2,p):   # on peut écrire range(2,int(p**0.5)+1) pour accélérer
        if p%i==0 :
            return False
    return True
```

(b)
```
def premiersnaif(n) :
    L=[]
    for p in range(2,n+1):
        if estpremier(p) :
            L.append(p)
    return L
```

2.
```
def eratosthene(n):
    L=[True]*(n+1)
    L[0]=False
    L[1]=False
    for i in range(2,n+1):
        if L[i]:
            for j in range(2*i,n+1,i) :
                L[j]=False
    P=[]
    for i in range(n+1):
        if L[i] :
            P.append(i)
    return P
```