```python
# Ch2-Prog.py

def plus_proches1(L):
    n=len(L)
    paire=[0,1]
    dmin=abs(L[0]-L[1])
    for i in range(0,n):
        for j in range(0,n):
            if i!=j and dmin>abs(L[i]-L[j]):
                dmin=abs(L[i]-L[j])
                paire=[i,j]
    return(paire)

def plus_proches2(L):
    n=len(L)
    paire=[0,1]
    dmin=abs(L[0]-L[1])
    for i in range(0,n-1):#toutes les paires [...,n] seront testées grâce à la seconde boucle
        for j in range(i+1,n):#toutes les paires [i,j] avec j<i ont déjà été testées sous la forme [j,i] dans les itérations précédentes
            if dmin>abs(L[i]-L[j]):
                dmin=abs(L[i]-L[j])
                paire=[i,j]
    return(paire)


#-------------------------------------------------------------------------------------------------------------#

def tri_bulles(L):
    j=len(L)-1
    while j>0:
        tri=True
        for i in range(0,j):
            if L[i]>L[i+1]:
                L[i],L[i+1]=L[i+1],L[i]
                tri=False
        j=j-1
        if tri:
            j=0
 #il n'est pas nécessaire que la fonction effectue un renvoi puisque les modifications sont appliquées directement à la liste L


#-------------------------------------------------------------------------------------------------------------#

def recherche(motif, texte):
    n=len(texte)
    m=len(motif)
    for j in range(0,n-m+1):
        i=0
        while i<m and texte[j+i]==motif[i]:
            i=i+1
        if i==m:
            return(j)

def distances(motif):
    m=len(motif)
    dic={char(i):m for i in range(0,256)}#on initialise le tableau en considérant qu'aucun caractère n'apparait dans le motif
                                        #char(i) permet de convertir le nombre i en caractère ASCII
    for i in range(0,m-1):#le caractère de rang m-1 est exclu car, s'il n'apparait pas avant dans le motif, sa distance reste m
        dic[motif[i]]=m-1-i#pour les caractères apparaissant dans le motif (sauf
```

```
le dernier), on modifie la distance
56|     return(dic)
57|
58| def boyer_moore(texte,motif):
59|     m=len(motif)
60|     n=len(texte)
61|     d=distances(motif) # on effectue le pré-traitement du motif
62|     s=m-1  # on commence par le dernier caractère du motif
63|     sol=[] # on crée une liste afin de stocker les rangs des occurences du motif
dans le texte
64|     while s<n:
65|         i=m-1
66|         while i>=0 and motif[i]==texte[s]:# tant qu'il y a correspondance entre
les caractères du motif et du texte
67|             s=s-1                         # on recule d'un cran afin de comparer
le caractère précédent du motif
68|             i=i-1                         # au caractère précédent du texte
69|         if i<0:                           # si le motif est trouvé (i=-1 car m
passages dans la boucle précédente)
70|             sol.append(s+1)               # on stocke dans la liste sol le rang
dans le texte du premier terme du motif
71|         s=s+max(d[texte[s]],m-i)
72|     return(sol)
73|
74|
```