Chapitre 3 : Exemples de boucles imbriquées

Il est possible de placer plusieurs boucles *while* ou *for* à l'intérieur d'une boucle *while* ou *for* : on parle de boucles imbriquées (la 1e boucle est dite externe et les autres internes).

Exemple :

```
L=[]
for i in range(4):
    for j in range(3):
        L=L.append([i,j])
print(L) [[0, 0], [0, 1], [0, 2], [1, 0], [1, 1], [1, 2], [2, 0], [2, 1], [2, 2], [3, 0], [3, 1], [3, 2]]
#par compréhension
print([[i,j] for i in range(4) for j in range(3)])
```

I- Recherche des deux points les plus proches Y34MNJ

On cherche une paire de points dont la distance euclidienne dans le plan est minimale dans un ensemble fini de points. On peut proposer un algorithme pour lequel on teste toutes les paires. Si l'ensemble contient n éléments, il y aura alors $\frac{n(n-1)}{2}$ paires à tester.

```
def distance(point1,point2):#point=>[x,y]
    x1,y1=point1
    x2,y2=point2
    return ((x1-x2)**2+(y1-y2)**2)**0.5
def plus_proches(liste_points):
    paire=[0,1]#initialisation
    d=distance(liste_points[0],liste_points[1])#init
    for i in range(len(liste_points)-1):
        for j in range(i+1,len(liste_points)):
            if d>distance(liste_points[i],liste_points[j]):
                d=distance(liste_points[i],liste_points[j])
                paire=[i,j]
    return paire
```

II- Exemple d'algorithme de tri : Le tri à bulles (LLA546)

Cherchons à trier une liste d'entiers dans l'ordre croissant. L'algorithme de tri à bulles consiste :

- à parcourir la liste et si deux éléments consécutifs sont rangés dans le désordre, on les échange ;
- si à la fin du parcours au moins un échange a eu lieu, on recommence l'opération ;
- sinon, la liste est triée, on arrête

```
def bulle(L):
    j=len(L)-1
    while j>0:
        modification="False"
        for i in range(j):#j exclue
            if L[i+1]<L[i]:
                L[i],L[i+1]=L[i+1],L[i]
                modification="True"
        j=j-1#plus grande valeur restante est mise à la fin
        if modification=="False":
            return L#ou j=0
    return L
```

III- Recherche textuelle (PMBXK5)

Pour déterminer la présence ou l'absence d'un mot dans un texte, le principe est le suivant :

- On cherche la présence du premier caractère
- Si on le trouve, on vérifie si les caractères suivants du motif coïncident avec le mot
- Si tous les caractères coïncident, le motif est trouvé, sinon on reprend la recherche du 1e caractère

```
def recherche(chaine,mot):
    n=len(chaine)
    m=len(mot)
    for i in range(n-m+1):
        j=0
        while j<m and chaine[i+j]==mot[j]:
            j=j+1
        if j==m:
            return print("mot est à l'index {}".format(i))
    print("le mot n'est pas dans la liste")
```

Commenté [AM3]: Exemple : L = [12,17,14,11,8]
-1e passage :
[12,17,14,11,8] → [12,14,17,11,8] → [12,14,11,17,8] → [12,14,11,8,17]
On pourra remarquer que la plus grande valeur se retrouve à la fin (et est donc triée) au 1e tour
-2e passage :
[12,14,11,8,17] → [12,11,14,8,17] → [12,11,8,14,17]
On pourra remarquer que les deux plus grands éléments sont rangés à leur place définitive.
-3e passage :
[12,11,8,14,17] → [11,12,8,14,17] → [11,8,12,14,17]
-4e passage :
[11,8,12,14,17] → [8,11,12,14,17]
-5e passage : aucune modification
Si la liste est triée, alors on a $n-1$ comparaisons.
Si la liste est décroissante alors on a autant de permutations que de comparaisons : $\frac{n(n-1)}{2}$

Commenté [AM4]: On présente ici un algorithme naïf c'est-à-dire un algo dont la mise en œuvre d'une idée simple

Commenté [AM1]: Nous disons que la complexité de cet algorithme est de l'ordre de n^2 :
$T(n) = (n-1) + (n-2) + \cdots = n * \frac{n-1}{2}$

Commenté [AM5]: Dans le pire cas (mot absent), la boucle for est exécutée $n-m+1$ *fois*. Si en plus, la boucle while est parcourue m fois alors le nombre de comparaison est au plus de $m(n-m+1)$

Commenté [AM6]: Avec le slicing on peut l'économie d'une boucle :

```
def recherche1(chaine,mot):
    n=len(chaine)
    m=len(mot)
    for i in range(n-m+1):
        if chaine[i:i+m]==mot:
            return print("le mot recherché est à la position {}".format(i))
    print("le mot n'est pas dans la liste")
```

Commenté [AM2]: Cette méthode permet de faire passer l'élément le plus grand à la fin de la liste