TP n°5 : corrigé

PTSI Lycée Eiffel

décembre 2014

Activités 1 et 2 :

Une méthode possible est la suivante : on parcourt toute la liste élément par élément, et on teste à chaque fois si l'élément est égal à celui qu'on recherche. Si c'est le cas, on peut tout de suite arrêter la recherche pour retourner notre résultat (la boucle s'arrêtera automatiquement si on écrit une définition de fonction et qu'on insère un return à ce moment-là). Sinon, on ne fait rien (on continuera évidemment la boucle). En fin de boucle, si on n'a croisé aucun élément correspondant à celui recherché, il est temps de retourner un message négatif. On peut par exemple écrire le programme suivant :

```
def recherche(liste,element) :
    for i in liste :
        if i==element :
            return 'élément trouvé'
    return "l'élément n'est pas dans la liste"
```

Activités 3 et 4 :

C'est déjà dans le cours, je ne reviens pas dessus.

Activités 5 et 6 :

Il suffit d'additionner tous les éléments de la liste, puis de diviser par le nombre d'éléments. On peut bien sûr tricher à l'aide de la fonction sum, mais si on préfère faire les choses « à la main » :

```
def moyenne(liste) :
    m=0
    for i in liste :
        m=m+i
    return m/len(liste)
```

Activités 7 et 8 :

On rappelle simplement que la variance est la moyenne des carrés des écarts à la moyenne. Le programme ressemble alors fortement au précédent, on fera juste attention à calculer un seule fois la valeur de la moyenne.

```
def variance(liste) :
    m=moyenne(liste)
    v=0
    for i in liste :
        v=v+(i-m)**2
    return v/len(liste)
```

Activités 9 et 10 :

Autant réutiliser la fonction déjà écrite pour le maximum (on peut aussi en écrire une très similaire pour le minimum). À chaque étape, on calcule le maximum de la liste restante, on insère la valeur dans la liste destinée à recueillir notre tri, et on supprime la valeur correspondante dans la liste d'origine. On recommence jusqu'à ce que la liste soit vide. On profite de ce programme pour utiliser quelques petites fonctionnalités sur les listes : on fera une copie intégrale de la liste donnée en argument puisqu'on va la modifier en cours de programme à l'aide de la commande m=l[:]. Comme va insérer à chaque fois le maximum au début de la deuxième liste, on utilisera aussi le tranchage. Bien sûr, on supprimera l'élément maximal à l'aide d'un remove, qui a l'avantage de ne supprimer qu'une occurrence si l'élément apparait plusieurs fois.

```
triselection(liste) :
    m=liste[ :]
    tri=[]
    while m!=[] :
        a=maximum(m)
        tri[0 :0]=[a]
        m.remove(a)
    return tri
```

Activités 11 et 12 :

C'est un peu plus compliqué à programmer que le précédent, mais on va suivre le même genre de méthode. Pas besoin de créer une copie de la liste initiale, qu'on ne va pas modifier. On se contente de la parcourir, et, pour chaque élément croisé, on l'insère au bon endroit dans la liste tri (qui correspond comme précédemment à la liste triée en cours de construction). Pour cela, on compare l'élément aux éléments de la liste tri jusqu'à en trouver un plus grand, et on l'insère à l'endroit adéquat grâce à un tranchage.

```
def triinsertion(liste) :
    tri=[]
    for i in liste :
        j=0
        while (j<len(tri)) and (i>tri[j]) :
            j=j+1
        tri[j :j]=[i]
    return tri
```

Activités 13 et 14 :

Comme je ne sais absolument pas ce qui était attendu ici, on va sauter directement aux questions suivantes !

Activités 15 et 16 :

Il suffit tout simplement de reprendre l'algorithme programmé en tout début de TP.

Activités 17 et 18 :

C'est dans le cours !

Activités 19 :

Avec l'algorithme naïf, on effectuera un nombre d'étapes (et de comparaisons) qui est au pire égal au nombre d'éléments dans la liste, et en moyenne égal à la moitié de ce nombre si l'élément est effectivement dans la liste. Avec la dichotomie, on divise le nombre d'élément potentiellement à tester par deux à chaque étape, jussqu'à ne plus en avoir qu'un. Il faut alors environ $\log_2(n)$ étapes, où n est le nombre d'éléments de la liste. C'est beaucoup moins que par la méthode naïve.

Activités 20 et 21 :

C'est dans le cours ! Mais comme c'est demandé dans un cas précis, écrivons le programme correspondant :

```
a=-1 ; b=0
while b-a>0.01 :
        c=(a+b)/2
        if 3*c+4 >0 :
                b=c
        else :
                a=c
print a
```