TP 05 – RECHERCHE SÉQUENTIELLE

Le but du TP est de programmer un algorithme de recherche d'un élément dans une liste (et ses variantes : recherche du minimum, du maximum, du second maximum). On va souvent faire des recherches *par balayage*, c'est-à-dire des recherches où l'on passe en revue tous les termes de notre liste afin de déterminer l'information souhaitée. On utilisera pour cela pour une boucle `for`.

I Recherche dans une liste par balayage

Le but du TP est de programmer un algorithme de recherche d'un élément dans une liste (et ses variantes : recherche du minimum, du maximum, du second maximum). On va souvent faire des recherches *par balayage*, c'est-à-dire des recherches où l'on passe en revue tous les termes de notre liste afin de déterminer l'information souhaitée. On utilisera pour cela pour une boucle `for`.

Entrée [1]:
```
#Balayage des éléments de manière directe
L = [1,4.2, 'Info']
for e in L:
    print(e)
```

Out [1]:
```
1,4.2, 'Info'
```

Entrée [2]:
```
#Balayage des éléments via la position
L = [1,4.2, 'Info']
for k in range(len(L)):
    print(L[k])
```

Out [2]:
```
1,4.2, 'Info'
```

Entrée [3]:
```
#Balayage des éléments via la position
L = [1,4.2, 'Info']
for k in range(len(L)):
    print(k)
    print(L[k])
```

Out [3]:
```
0,1,1, 4.2, 2, 'Info'
```

I.1 Recherche d'un élément

Pour effectuer une **recherche séquentielle** (méthode naïve) d'un élément `x` dans une liste `L`, on procède de la manière suivante.

- On parcourt tous les éléments de `L` un par un grâce à une boucle, en introduisant une variable `e` qui va prendre successivement toutes les valeurs de la liste.
  - Si l'élément de l'étape en cours correspond à l'élément `x`, on renvoie `True`.
- Une fois tous les éléments testés, si aucun ne correspond à l'élément `x`, on renvoie `False`.

À noter : Python sait déjà effectuer cette tâche, via la commande `x in L`, que l'on n'utilisera pas...

Entrée [4]:
```
L=[4,8,15,16,23,42]  #(Qui a la ref?)
print(8 in L)
print(2 in L)
```

Out [4]:
```
True
False
```

Exercice 1 Décrire (sur papier) l'algorithme lorsque l'on cherche si l'élément `x=10` appartient à la liste `L=[1,4,5,10,3]`.

`e`	Test `e=x` ?	Arrêt du programme ? Oui/Non
`1`	Non	On continue...
`4`	Non	On continue...
`5`	Non	On continue...
`10`	Oui	Arrêt et affiche `True`
`3`	x	x

Exercice 2 Décrire (sur papier) l'algorithme lorsque l'on cherche si l'élément `x=7` appartient à la liste `L=[4, 8, 15, 16, 23, 42]`.

`e`	Test `e=x` ?	Arrêt du programme ? Si oui, affichage.
`4`	Non	On continue...
`8`	Non	On continue...
`15`	Non	On continue...
`16`	Non	On continue...
`23`	Non	On continue...
`42`	Non	Arrêt et affiche `False`

Exercice 3

1. Écrire une fonction `recherchenaive` qui prend en argument une liste `L` et un élément `x` et qui renvoie `True` si l'élément `x` est dans la liste et `False` sinon. *Vérification : L'évaluation de la fonction en* `([1,4,5,10,3],10)` *doit renvoyer* `True` *et l'évaluation de la fonction en* `([4, 8, 15, 16, 23, 42],7)` *doit renvoyer* `False`.

Entrée [5]:
```
def recherchenaive(L,x):
    for e in L: #création d'une variable e qui parcourt la liste
        if e==x: #si e correspond à x
            return(True) #la fonction renvoie True
    return(False) #si aucun match, la fonction renvoie False
```

Tester cette fonction.

Entrée [6]:
```
recherchenaive([1,4,5,10,3],10)
```

Out [6]:
```
True
```

Entrée [7]:
```
recherchenaive([4, 8, 15, 16, 23, 42],7)
```

Out [7]:
```
False
```

2. Si la liste `L` est de taille n (c'est-à-dire contient n éléments), alors le nombre maximal de tests d'égalité effectués est

$$n$$

Exercice 4

1. Écrire une fonction `nbocc` qui prend en argument une liste `L` et un élément `x` et qui renvoie le nombre d'occurences de `x` dans la liste. *Vérification : L'évaluation de la fonction en* `([1,2,0,2,1,1,1],1)` *doit renvoyer* `4`.

Entrée [8]:
```
def nbocc(L,x):
    compteur = 0
    for e in L:
        if e==x:
            compteur=compteur+1
    return(compteur)
```

Tester cette fonction.

Entrée [9]:
```
nbocc([1,2,0,2,1,1,1],1)
```

Out [9]:
```
4
```

I.2 Recherche de la place d'un élément dans une liste

Grâce à la recherche séquentielle, on peut aussi déterminer le plus petit indice (position) où se trouve l'élément `x` s'il est présent dans la liste. On procède de la manière suivante.

- On parcourt tous les éléments de `L` un par un grâce à une boucle, à l'aide d'une variable k qui correspond à l'indice de l'élément dans la liste que l'on regarde.
  - Si l'élément de l'étape en cours correspond à l'élément `x`, on renvoie l'indice de l'élément en cours.
- Une fois tous les éléments testés, si aucun ne correspond à l'élément `x`, on renvoie un message d'erreur.

Exercice 5 Décrire (sur papier) l'algorithme lorsque l'on cherche le plus petit indice où se trouve l'élément `x=10` dans la liste `L=[1,4,5,10,3]`.

`k`	`L[k]`	`L[k]=x` ?	Arrêt du programme ? Si oui, affichage.
`0`	`1`	Non	On continue...
`1`	`4`	Non	On continue...
`2`	`5`	Non	On continue...
`3`	`10`	Oui	Arrêt et affichage de `3`
`4`	`3`	x	x

Exercice 6 Écrire une fonction `indice` qui prend en argument une liste `L` et un élément `x` et qui renvoie le plus petit indice où se trouve l'élément `x` s'il est présent dans la liste, et renvoie le message "non trouvé" sinon. *Vérification : L'évaluation de la fonction en* `([1,4,5,10,3],10)` *doit renvoyer* `3`*, alors que l'évaluation en* `([4, 8, 15, 16, 23, 42],7)` *doit renvoyer* `Non trouvé`.

Entrée [10]:
```
def indice(L,x):
    for k in range(len(L)): #on parcourt  les indices de la liste
        if L[k]==x: #si l'élément en position k vaut x
            return(k) #on renvoie la position de l'élément
    return('Non trouvé !')  #si aucun match, message d'erreur
```

Entrée [11]:
```
indice([1,4,5,10,3],10)
```

Out [11]:
```
3
```

Entrée [12]:
```
indice([4, 8, 15, 16, 23, 42],7)
```

Out [12]:
```
Non trouvé !
```

Exercice 7 Écrire une fonction `listeindice` qui prend en argument une liste `L` et un élément `x` et qui renvoie la liste de tous les indices de position de `x` dans la liste `L` (le programme renvoie une liste vide si l'élément `x` ne se trouve pas dans `L`). *Vérification : L'évaluation de la fonction* `listeindice` *en* `([4,5,4],4)` *doit renvoyer* `[0,2]`.

Entrée [13]:
```
def listeindice(L,x):
    P=[]
    for k in range(len(L)):
        if L[k]==x:
            P.append(k)
    return(P)
```

Entrée [14]:
```
listeindice([4,5,4],4)
```

Out [14]:
```
[0, 2]
```

Exercice 8 Écrire une fonction `dernierindice` qui prend en argument une liste `L` et un élément `x` et qui renvoie le plus grand indice où se trouve l'élément `x` s'il est présent dans la liste, et renvoie le message "non trouvé" sinon. *Vérification : L'évaluation de la fonction* `dernierindice` *en* `([4,5,4],4)` *doit renvoyer* `2`.

Entrée [15]:
```
def dernierindice(L,x):
    P=listeindice(L,x)
    return(P[-1])
```

Entrée [16]:
```
dernierindice([4,5,4],4)
```

Out [16]:
```
2
```

I.3 Recherche d'un minimum/maximum d'une liste

Grâce à la recherche séquentielle, on peut aussi déterminer le maximum d'une liste. On procède de la manière suivante.

- On initialise une variable `maxi` au premier élément de la liste.
- On parcourt tous les éléments de `L` un par un grâce à une boucle.
  - Si l'élément de l'étape en cours est plus grand (strictement) que le maximum temporaire, on actualise la valeur du maximum.
  - Sinon, on ne fait rien.

Exercice 9 Décrire (sur papier) l'algorithme lorsque l'on cherche le maximum de la liste `L=[1,2,3,-1]`.

`e`	`maxi` (temporaire)	Affichage.
`1`	`1`	
`2`	`2`	
`3`	`3`	
`-1`	`3`	Arrêt et affichage de `3`

Exercice 10 Écrire une fonction `maximum` qui prend en argument une liste `L` et qui renvoie le maximum de cette liste. *Vérification : L'évaluation de la fonction* `maximum` *en* `[-3,-5,-10,-1]` *doit renvoyer* `-1`.

Entrée [17]:
```
def maximum(L):
    maxi = L[0]
    for e in L:
        if e>maxi:
            maxi=e
    return(maxi)
```

Tester cette fonction.

Entrée [18]:
```
maximum([-3,-5,-10,-1])
```

Out [18]:
```
-1
```

Exercice 11 Écrire une fonction `minimum` qui prend en argument une liste `L` et qui renvoie le minimum de cette liste. *Vérification : L'évaluation en* `[1,2,-1,3]` *doit renvoyer* `-1`.

Entrée [19]:
```
def minimum(L):
    mini = L[0]
    for e in L:
        if e<mini:
            mini=e
    return(mini)
```

Tester cette fonction.

Entrée [20]:
```
minimum([1,2,-1,3])
```

Out [20]:
```
-1
```

Exercice 12 Écrire une fonction `indiceminimum` qui en argument une liste `L` et qui renvoie la (première) position du minimum de cette liste. *Vérification : L'évaluation en* `[-3,-5,-10,-1]` *doit renvoyer* `3`.

Entrée [21]:
```
def indiceminimum(L):
    mini = L[0]
    imini = 0
    for k in range(len(L)):
        if L[k]<mini:
            mini=L[k]
            imini=k
    return(k)
```

Entrée [22]:
```
indiceminimum([-3,-5,-10,-1])
```

Out [22]:
```
3
```

Exercice 13 Écrire une fonction `secmaximum` qui prend en argument une liste `L` et qui renvoie la valeur du deuxième maximum de la liste. Si la liste est de taille inférieure ou égale à 1, cette fonction doit renvoyer `None`. *Vérification : L'évaluation en* `[3,1,10,5,7]` *doit renvoyer* `7` *alors que l'évaluation en* `[1]` *doit renvoyer* `None`.

Entrée [23]:
```
def secmaximum(L):
    if len(L):
        return('None')
    maxi=maximum(L)
    M=[x for x in L if x !=maxi]
    return(maximum(M))
```

Tester cette fonction.

Entrée [24]:
```
secmaximum([3,1,10,5,7])
```

Out [24]:
```
7
```

Entrée [25]:
```
secmaximum([10,2,-1,3])
```

Out [25]:
```
'None'
```

II Moyenne et variance

Exercice 14 Écrire une fonction `somme` qui prend en argument une liste `L` et qui calcule la somme de tous les éléments. *Vérification : L'évaluation en* `[1,2,3,4,5,6]` *doit renvoyer* `21`.

Entrée [26]:
```
def somme(L):
    c = 0
    for e in L:
        c=c+e
    return(c)
```

Entrée [27]:
```
L = [1,2,3,4,5,6]
somme(L)
```

Out [27]:
```
21
```

Soit $L = [x_1, x_2, \ldots, x_n]$ une liste de réels. On définit la *moyenne* de la liste par :

$$m = \frac{1}{n}\sum_{k=1}^{n} x_k.$$

La *variance* de L est alors :

$$V = \frac{1}{n}\sum_{k=1}^{n} (x_k - m)^2.$$

Exercice 15 Écrire une fonction `moyenne` qui prend en argument une liste d'entiers et qui renvoie la moyenne des valeurs d'une liste. *Vérification : L'évaluation en* `[0,20]` *doit renvoyer* `10`.

Entrée [28]:
```
def moyenne(L):
    s = 0
    for e in L:
        s = s + e
    s = s / len(L)
    return s
```

Entrée [29]:
```
moyenne([0,20])
```

Out [29]:
```
10
```

Exercice 16 Écrire une fonction `variance` qui prend en argument une liste d'entiers et qui renvoie la variance des valeurs d'une liste. *Vérification : L'évaluation en* `[0,20]` *doit renvoyer* `100`.

Entrée [30]:
```
def variance(L):
    m=moyenne(L)
    s = 0
    for e in L:
        s = s + (e-m)**2
    s = s / len(L)
    return s
```

Entrée [31]:
```
variance([0,20])
```

Out [31]:
```
100
```

III Comptage dans une liste à l'aide d'un dictionnaire

Exercice 17

1. Ecrire une fonction `dico` prenant en argument une liste `L` et renvoyant un dictionnaire dont les clefs sont les éléments distincts de `L` et les valeurs associés sont le nombre d'occurrences de cet élément dans `L`. Pour ajouter une nouvelle valeur au dictionnaire, on pourra utiliser la commande `nom_dicto[nvclef]=nvvaleur`. *Vérification : L'évaluation en* `[5, 13, 2, 9, 5, 1, 13]` *doit renvoyer* `{5:2 , 13:2 , 2:1 , 9:1 , 1:1}`.

Entrée [32]:
```
def dico(L):
    d={}
    for e in L:
        nvclef=e
        nvvaleur=nbocc(L,e)
        d[nvclef]=nvvaleur
    return(d)
```

Entrée [33]:
```
dico( [5, 13, 2, 9, 5, 1, 13])
```

Out [33]:
```
{5: 2, 13: 2, 2: 1, 9: 1, 1: 1}
```

2. Ecrire une fonction `identiques` prenant en argument deux listes `L1` et `L2`, et renvoyant `True` si ces deux listes ont les mêmes éléments avec pour chacun le même nombre d'occurrences (pas forcément à la même place), et `False` sinon. *Vérification : L'évaluation en* `([1,2],[2,1])` *doit renvoyer* `True`.

Entrée [34]:
```
def identiques(L1,L2):
    return(dico(L1)==dico(L2))
```

Entrée [35]:
```
identiques([1,2],[2,1])
```

Out [35]:
```
True
```

IV Recherche d'un mot dans une chaîne de caractères

Exercice 18

1. Ecrire puis tester une fonction `is_mot_position` prenant en arguments un mot `m`, un indice de position `i` et une chaîne de caractères `s` et renvoyant `True` si le mot `m` apparaît dans `s` à partir de la i-ième position (c'est-à-dire si la mot apparaît et que sa première lettre tombe exactement en position i), `False` sinon.

   Par exemple :

   - `is_mot_position (''petit'',12,''il était un petit homme'')` doit renvoyer `True`,
   - `is_mot_position (''petit'',13,''il était un petit homme'')` doit renvoyer `False`
   - `is_mot_position (''petit'',12,''il était un peti'')` doit renvoyer `False`

Entrée [36]:
```
def is_mot_position(m,i,s):
    if i+len(m)>len(s):
        return(False)
    for j in range(len(m)):
        if s[i+j] != m[j]:
            return(False)
    return(True)
```

2. Ecrire puis tester une fonction `is_mot` prenant en arguments un mot `m` et une chaîne de caractères `s` et renvoyant `True` si le mot `m` apparaît dans `s` `False` sinon.

Entrée [37]:
```
def is_mot(m,s):
    for i in range(len(s)-len(m)+1):
        if is_mot_position(m,i,s):
            return(True)
    return(False)
```

3. Ecrire puis tester une fonction `recherche_mot` prenant en arguments un mot `m` et une chaîne de caractères `s` et renvoyant l'indice de la première occurrence du mot `m` dans la chaîne `s` si ce mot y figure, et `None sinon`.

Entrée [38]:
```
def is_mot(m,s):
    for i in range(len(s)-len(m)+1):
        if is_mot_position(m,i,s):
            return(i)
    return(None)
```

4. Ecrire puis tester une fonction `occurrences_mot` prenant en arguments un mot `m` et une chaîne de caractères `s` et renvoyant le nombre d'occurrences du mot `m` dans la chaîne `s`.

Entrée [39]:
```
def occurrences_mot(m,s):
    compteur = 0
    for i in range(len(s)-len(m)+1):
        if is_mot_position(m,i,s):
            compteur=compteur+1
    return(compteur)
```

V Exercices en vrac

Exercice 19 Écrire une fonction `selectionner`, qui prend en argument une liste `L` et deux entiers `a` et `b` (avec $a \leqslant b$) et qui renvoie la liste (éventuellement vide) des éléments de `L` qui sont compris (au sens large) entre `a` et `b`.

Entrée [40]:
```
def selectionner(L,a,b):
    M=[]
    for e in L:
        if a<=e and e<=b:
            M.append(e)
    return(M)
```

Exercice 20 Écrire une fonction `contientpositif`, qui prend en argument une liste `L`et qui renvoie `True` si la liste contient un nombre strictement positif et `False` sinon.

Entrée [41]:
```
def contientpositif(L):
    for e in L:
        if e>0:
            return(True)
    return(False)
```

Exercice 21 Écrire une fonction `supprimer`, qui prend en argument une liste `L` et un entier `x` et qui renvoie une liste constituée des mêmes éléments que `L` sauf les entiers `x`.

Entrée [42]:
```
def surpprimer(L,x):
    S=[]
    for e in L:
        if not(e==x):
            S.append(e)
    return(S)
```

Exercice 22 Écrire une fonction `npplusmoins`, qui prend en argument une liste `L` et renvoie le couple (p, m) où p est égal à la somme des éléments positifs ou nuls de `L` et m est égal à la somme des éléments strictement négatifs de `L`.

Entrée [43]:
```
def nbplusmoins(L):
    m=0
    p=0
    for e in L:
        if e>=0:
            p=p+e
        else:
            m=m+e
    return(p,m)
```

Exercice 23 Écrire une fonction `valproche` qui prend en argument une liste `L` et un élément `x` (qui est dans la liste) et qui renvoie l'élément le plus proche de `x`, c'est-à-dire l'élément dont l'écart à `x` est le plus petit. L'écart entre deux valeurs x et y peut être quantifiée par $|x - y|$.

Entrée [44]:
```
import numpy as np
def valproche(L,x):
    y = L[0]
    ecart = np.abs(x-y)
    for e in L:
        if e != x :
            if np.abs(e-x)< ecart :
                y = e
                ecart = np.abs(x-y)
    return(y)
```

Tester cette fonction.

Entrée [45]:
```
valproche([1,3,7,13], 14)
```

Out [45]:
```
13
```

Entrée [46]:
```
valproche([1,3,7,13], 0)
```

Out [46]:
```
1
```

Exercice 24 Écrire une fonction `deuxvalproches` qui prend en argument une liste `L` et qui renvoie les deux éléments les plus proches (c'est-à-dire les deux éléments dont l'écart est le plus petit parmi tous les couples d'éléments de la la liste).

Entrée [47]:
```
def deuxvalproches(L):
    ecart = np.abs(L[0] - L[1])
    x=L[0]
    y=L[1]
    for e in L:
        for f in L:
            if e != f :
                if np.abs(e-f) < ecart :
                    x = e
                    y = f
                    ecart = np.abs(e-f)
    return(x,y)
```

Entrée [48]:
```
#version optimisée
import numpy as np
def deuxvalprochesopt(L):
    ecart = np.abs(L[0] - L[1])
    x=L[0]
    y=L[1]
    for i in range(len(L)):
        for j in range(i+1, len(L)):
            if np.abs(L[i] - L[j]) < ecart :
                ecart=np.abs(L[i] - L[j])
                x=L[i]
                y=L[j]
    return(x,y)
```

Tester cette fonction.

Entrée [49]:
```
deuxvalproches([1,6,10,9,42])
```

Out [49]:
```
(10,9)
```

Entrée [50]:
```
deuxvalproches([1,3,7,13])
```

Out [50]:
```
(1,3)
```

Exercice 25 Écrire une fonction `croissante` qui prend en argument une liste d'entiers et qui renvoie `True` si la liste est rangée dans l'ordre croissant et `False` sinon.

Entrée [51]:
```
def croissante(L):
    for k in range(len(L)-1):
        if not(L[k]<=L[k+1]):
            return(False)
    return(True)
```

Exercice 26 Écrire une fonction `tri` qui prend en argument une liste `L` et qui renvoie cette même liste mais rangée dans l'ordre croissant. On pourra utiliser l'algorithme de tri par sélection qui se déroule de la manière suivante :

- On cherche le plus petit élément de la liste et on l'échange avec l'élément en position 0.
- On cherche le second plus petit élément de la liste et on l'échange avec l'élément en position 1.
- On continue ainsi jusqu'à ce que le tableau soit entièrement trié.

Entrée [52]:
```
def tri(L):
    n = len(L)
    for i in range(n-1):
        imini = i
        for j in range (i+1, n):
            if L[j] < L[imini]:
                imini = j
        L[i], L[imini]= L [imini] , L[i]
    return(L)
```