TP 10 – Recherche dans une liste

Soit `L` une liste composée d'entiers et `x` un nombre entier. On cherche à écrire une fonction `recherche(L,x)` qui renvoie `True` ou `False` selon que `x` appartienne à la liste `L` ou non, sans utiliser de commande prédéfinie dans `Python`.

I Recherche séquentielle

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`.

I.1 Exemples à la main

1. Recherchons si l'élément `x=10` appartient à la liste `L=[1,4,5,10,3]`.

`e`	`e=x` ?	Arrêt du programme ? Si oui, affichage.
`1`	Non	On continue...
`4`	Non	On continue...
`5`	Non	On continue...
`10`	Oui	Arrêt et affiche `True`
`3`	x	x

2. Recherchons si l'élément `x=7` appartient à la liste `L=[4, 8, 15, 16, 23, 42]`.

`e`	`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`

I.2 Algorithme

Grâce à la recherche séquentielle, on peut coder un algorithme déterminant si un élément donné est dans une liste donnée.

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.

Entrée [1]:
```
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 [2]:
```
recherchenaive([1,4,5,10,3],10)
```

Out [2]:
```
True
```

Entrée [3]:
```
recherchenaive([4, 8, 15, 16, 23, 42],7)
```

Out [3]:
```
False
```

I.3 Des variantes

a) Recherche de la place d'un élément dans une liste

Grâce à la recherche séquentielle, on peut aussi déterminer le plus petit indice 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.

1. Déterminons 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

2. Déterminons le plus petit indice où se trouve l'élément `x=7` dans la liste `L=[4, 8, 15, 16, 23, 42]`.

k	L[k]	L[k]=x ?	Arrêt du programme ? Si oui, affichage.
0	4	Non	On continue...
1	8	Non	On continue...
2	15	Non	On continue...
3	16	Non	On continue...
4	23	Non	On continue...
5	42	Non	Arrêt et affichage d'une erreur

② 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.

Entrée [4]:
```
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
```

Tester cette fonction.

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

Out [5]:
```
3
```

Entrée [6]:
```
indice([4, 8, 15, 16, 23, 42],7)
```

Out [6]:
```
Non trouvé !
```

b) Recherche du nombre d'occurence d'un élément dans une liste

Grâce à la recherche séquentielle, on peut aussi compter le nombre de fois qu'un élément est présent dans une liste, en rajoutant une variable qui fait office de compteur.

1. Déterminons le nombre de fois que l'élément `x=1` est présent dans la liste `L=[1,2,0,2,1,1,1]`.

e	e=x ?	Compteur	Affichage.
1	Oui	1	
2	Non	1	
0	Non	1	
2	Non	1	
1	Oui	2	
1	Oui	3	
1	Oui	4	Arrêt et affichage de 4

2. Déterminons le nombre de fois que l'élément `x=7` est présent dans la liste `L=[1,2,0,2,1,1,1]`.

`e`	`e=x` ?	`Compteur`	Affichage.
`1`	Non	`0`	
`2`	Non	`0`	
`0`	Non	`0`	
`2`	Non	`0`	
`1`	Non	`0`	
`1`	Non	`0`	
`1`	Non	`0`	Arrêt et affichage de `0`

③ 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.

Entrée [7]:
```
def nbocc(L,x):
    compteur = 0
    for e in L:
        if e==x:
            compteur=compteur+1
    return(compteur)
```

Tester cette fonction.

Entrée [8]:
```
nbocc([1,2,0,2,1,1,1],1)
```

Out [8]:
```
4
```

Entrée [9]:
```
nbocc([1,2,0,2,1,1,1],4)
```

Out [9]:
```
0
```

c) 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.

1. Déterminons le maximum de la liste `L=[1,2,3,-1]`.

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

2. Déterminons le maximum de la liste `L=[4,5,-1,10]`.

e	maxi	Affichage.
4	4	
5	5	
-1	5	
10	10	Arrêt et affichage de 10

④ Écrire une fonction `maximum` qui prend en argument une liste `L` et qui renvoie le maximum de cette liste.

Entrée [10]:
```
def maximum(L):
    maxi = L[0]
    for e in L:
        if e>maxi:
            maxi=e
    return(maxi)
```

Tester cette fonction.

Entrée [11]:
```
maximum([1,2,3,-1]
```

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

Entrée [12]:
```
maximum([4,5,-1,10])
```

Out [12]:
```
10
```

⑤ Écrire une fonction `minimum` qui prend en argument une liste `L` et qui renvoie le minimum de cette liste.

Entrée [13]:
```
def minimum(L):
    mini = L[0]
    for e in L:
        if e<mini:
            mini=e
    return(mini)
```

Tester cette fonction.

Entrée [14]:
```
minimum([1,2,-1,3]
```

Out [14]:
```
-1
```

Entrée [15]:
```
minimum([10,2,-1,3])
```

Out [15]:
```
-1
```

II Pour aller plus loin

⑥ Écrire une fonction `ListePositions`, qui prend en argument une liste `L` et un élément `x` et qui renvoie la liste (éventuellement vide) de toutes les positions de l'élément `x` dans la liste `L`.

Entrée [16]:
```
def ListePositions(L,x):
    P=[]
    for k in range(len(L)):
        if L[k]==x:
            P.append(k)
    return(P)
```

⑦ 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 [17]:
```
def selectionner(L,a,b):
    M=[]
    for e in L:
        if a<=e and e<=b:
            M.append(e)
    return(M)
```

⑧ 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 [18]:
```
def contientpositif(L):
    for e in L:
        if e>0:
            return(True)
    return(False)
```

⑨ 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 [19]:
```
def surpprimer(L,x):
    S=[]
    for e in L:
        if not(e==x):
            S.append(e)
    return(S)
```

⑩ 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 [20]:
```
def nbplusmoins(L):
    m=0
    p=0
    for e in L:
        if e>=0:
            p=p+e
        else:
            m=m+e
    return(p,m)
```