TP n° 05 – Recherches, manipulation de liste

I Manipulations de listes

Exercice 1. Utilisez une boucle `for` pour afficher tous les termes de la liste `L=[2,8,-7,3]` :

```
2
8
-7
3
```

Exercice 2. Soit `L1` la liste suivante : `L1=[12,-3,8,1.2,7]`.

1. Ajouter `10` à la fin de la liste. Afficher la nouvelle liste.
2. Afficher l'élément `1.2` de la liste.
3. Retirer l'élément `12` de la liste.
4. Afficher le dernier élément de la liste.
5. Modifier l'élément en position 1 de la liste par l'élément `77`.
6. Créer la liste `L2=[-3,77,1.2,7,10,0,1,2,3,...,100]`.
7. Créer la liste des 30 premiers termes de `L2`.
8. Créer la liste des 25 premiers termes de `L2` sans les 3 premiers termes, suivis de 30 zéros.

Exercice 3. Ecrire une fonction `renverser` qui à une liste renvoie la liste renversée.
Remarque : On reprogramme ainsi la méthode `reverse` déjà implémentée dans Python.

```
>>> L=[2,8,-1,7]
>>> renverser(L)
[7,-1,8,2]
```

II Recherches

Exercice 4. Recherche dans une liste
Programmer une fonction `position(liste,element)` qui a comme entrée une liste et un élément et qui renvoie la position de l'élément dans la liste et qui renvoie `None` si l'élément n'est pas dans la liste.
Remarque : On reprogramme ainsi la méthode `index` déjà implémentée dans Python

Exercice 5. Recherche du maximum dans une liste de nombres.

1. Ecrire une fonction `maximum(liste)` qui renvoie le maximum d'une liste de nombres non triée. Montrer que la complexité de l'algorithme obtenu est linéaire.

   ```
   >>>L=[2,8,-7,3]
   >>>maximum(L)
   8
   ```

2. Ecrire une fonction `positionMax(liste)` qui renvoie le maximum et la position de ce maximum pour une liste de nombres :

   ```
   >>>positionMax(L)
   (1,8)
   ```

3. Que se passe-t-il si le maximum apparait plusieurs fois dans le tableau ?
   Modifier la fonction pour que toutes les positions apparaissent.

   ```
   >>>L2=[1,-1,1,0,1]
   >>>positionMax2(L2)
   [0,2,4]
   ```

Exercice 6. Pour les listes, il existe une fonction `max` et une méthode `index`. On en rappelle les syntaxes respectives dans l'annexe.
Testez `max` et `index` sur des exemples pour comprendre ce qu'elles renvoient.
A l'aide de `max` et `index`, déterminer la première position du maximum de la liste `L`.

Exercice 7. Recherche d'un mot dans une chaîne de caractères.

1. Ecrire une fonction `estIci(motif,texte,i)` qui a comme entrée deux listes (ou deux chaînes de caractères) `motif` et `texte` et un entier `i` et qui renvoie `True` si `motif` est dans `texte` à la position `i` et `False` sinon.

```
>>>estIci('le','Bonjour le monde',8)
True
>>>estIci('le','Bonjour le monde',9)
False
```

2. Ecrire une fonction `recherche(motif,texte)` qui a comme entrée deux listes (ou deux chaînes de caractères) et qui renvoie `True` si `motif` est dans `texte` et `False` sinon.

```
>>>recherche('le','Bonjour le monde')
True
>>>recherche('bonjour','Bonjour le monde')
False
```

3. Déterminer la complexité de l'algorithme de recherche d'un motif dans une liste.
Indication : On se placera dans le pire des cas. La complexité dépendra de la longueur de `motif` et de celle de `liste`

Remarque. Liste en compréhension
Une liste en compréhension est une liste dont le contenu est défini par filtrage du contenu d'une autre liste. Sa construction se rapproche de la notation ensembliste en mathématiques. Exemples :

en langage ensembliste :	$\mathcal{S}_1 = \{3n+2 \mid n \in \mathbb{N},\ n < 20\}$	$\mathcal{S}_2 = \{n \mid n \in \mathbb{N},\ n \leq 50,\ n^2 - 17n + 60 < 0\}$
en Python :	`S1=[3*n+2 for n in range(20)]`	`S2=[n for n in range(51) if n**2-17*n+60<0]`

Exercice 8.
Utilisez une liste en compréhension pour créer la liste suivante :
On considère la suite $u_n = 4n^2 + 7n - 2$. Créer la liste des termes de la suite qui sont des multiples de 3 pour $n \leq 20$.

Annexe

En Python, il y a deux façons de faire des opérations sur les objets qu'on manipule : les fonctions et les méthodes.
La différence est, pour vous, surtout d'ordre syntaxique.

Syntaxe fonction :	Syntaxe méthode :
`fonction(objet ,parametres)`	`objet.methode(parametres)`

Pour les listes, on trouve des fonctions et des méthodes. Certaines modifient la liste et ne renvoient rien, d'autres renvoient un résultat sans modifier la liste.

Liste non-exhaustive des méthodes pour les listes.
- **append(élément)** : modifie la liste en ajoutant élément à la fin de la liste
- **remove(élément)** : modifie la liste en supprimant élément de la liste
- **reverse()** : modifie la liste en inversant les valeurs de la liste
- **count(élément)** : renvoie le nombre d'occurrences d'élément dans la liste
- **index(élément)** : renvoie la position de élément dans la liste

Liste non-exhaustive des fonctions pour les listes :
- **del liste[index]** : modifie la liste en éliminant l'item en position **index**
- **len** : renvoie la longueur de la liste
- **max** : renvoie le maximum d'une liste de nombres

Correction TP n° 05 – Recherches, manipulation de liste

Solution 1.

```
for i in L:
    print(i)
```

ou

```
for i in range(len(L)):
        print(L[i])
```

Solution 2.

1. `L1.append(10)`
2. `print L1[3]`
3. en utilisant la position de l'élément : `del(L1[0])`
   en utilisant la valeur de l'élément : `L1.remove(12)`
4. `print L1[-1]`
5. `L1[1]=77`
6. `L2=L+range(101)`
7. `L3=L2[:30]`
8. `L4=L2[3:25]+30*[0]`

Solution 3.

```
def renverser(liste):
    n=len(liste)
    listeRenversee=[]
    for i in range(n):
        listeRenversee.append(liste[n-i-1])
    return(listeRenversee)
```

Solution 4.

```
def f(L,a):
# on commence au début de la liste
    i=0
    # tant que on n'est pas au bout de la liste et que le terme de la liste n'est pas a, on avance
    while i<len(L) and L[i]!=a:
        i=i+1
    # si on est sorti de la liste, on renvoie None
    if i= =len(L):
        return(None)
    # sinon on renvoie la position
    else:
        return(i)
```

Solution 5.

1. ```
   def maximum(liste):
       n=len(liste)
       # le maximum est initialis\' e avec le premier terme de la liste
       max=liste[0]

       # on parcourt la liste
       for i in range(n):
       # si le i\` eme terme de la liste est sup\' erieur au maximum, il devient la nouvelle vale
           if liste[i]>max:
               max=liste[i]
       return(max)
   ```
   ```

Dans cette fonction, on compte le nombre d'opérations :
— deux affectations avant la boucle
— n itérations de la boucle dans laquelle on effectue :
  — un test
  — au pire des cas, une affectation.

Au total, on a comme nombre d'opérations : $2 + 2n = O(n)$.

On a donc bien une complexité linéaire.

2.
```
def positionMax(liste):
    n = len(liste)
    indexMax = 0
    max = liste[0]
    for i in range(1,n):
        if liste[i] > max :
            max = liste[i]
            indexMax = i
    return indexMax,max
```

3. Dans le cas de la liste `L2`, seule la position du premier maximum est donnée.

```
def positionMax(liste):
    n = len(liste)
    # la liste des positions des maxima est initilis\' ee \` a 0
    indexMax = [0]
    max = liste[0]
    for i in range(1,n):
     # si le i\` eme terme de la liste est sup\' erieur au maximum, il devient la nouvelle va
        if liste[i] > max :
            max = liste[i]
            # la liste des positions est r\' einitialis\' ee \` a [i]
            indexMax = [i]
        # si on retombe sur le maximum, on ajoute l'index \` a la liste des positions
        elif liste[i]==max:
            indexMax.append(i)
    return indexMax
```

Solution 6.
```
>>> L.index(max(L))
1
```

Solution 7. 1.
```
def estIci(motif,texte,i):
    k = 0
    p = len(motif)
    # on parcourt le texte \` a partir de l'index i tant que on ne d\' epasse pas la longueur
    while k<p and motif[k] == texte[k+i]:
        k = k+1
    # si on a parcouru p termes, alors, motif est dans texte \` a l'index i
    if k==p:
        resultat=True
    else:
        resultat=False
    return resultat
```

2.
```
def recherche(motif,texte):
    n = len(texte)
```

```
    p = len(motif)
    # si motif est plus long que texte, on renvoie False
    if p>n:
        return False
    else:
    # par d\' efaut, Resultat est False
        resultat=False
        i=0
        # on cherche motif dans texte \` a toutes les positions i
        while i <= n-p and resultat==False:
            resultat=estIci(motif,texte,i)
            i = i + 1
        return resultat
```

3. Le pire des cas s'obtient avec un texte type : `texte='aaaaa...aaaaa'` et un motif du type : `'aa..aab'`. On note n la longueur du texte et p celle du motif.
Compté grossièrement : dans `estIci(motif,texte,i)`, on effectue p tests. On répète $n-p$ fois cette opération dans `recherche(motif,texte)`. Donc on s'attend à une complexité en : $\boxed{O((n-p)p).}$
On obtient $O(np)$ si $n >> p$

   Plus précisément, avec le pire des cas :
   dans `estIci(motif,texte,i)`, on a :
   — 2 affectations avant la boucle
   — on répète p fois :
      — 2 tests
      — 1 affectation
      — 1 addition
   — 1 test
   — 1 affectation

   $\boxed{\text{Au total, on a comme nombres d'opérations : } 2+4p+2=4p+4=O(p).}$
   `recherche(motif,texte)`, on a :
   — 2 affectations
   — dans la boucle else : 2 affectations
   — on répète $n-p$ fois :
      — 2 tests
      — 1 affectation
      — 1 calcul de `estIci`
      — 1 afffectation
      — 1 addition

   Au total, on a comme nombres d'opérations :
   $\boxed{2+2+(n-p)\times(5+4p+4)=4+(n-p)(4p+9)=O((n-p)p).}$

Solution 8.

```
L1=[4*n**2+7*n-2 for n in range(21) if (4*n**2+7*n-2)%3==0]
>>>[9, 90, 243, 468, 765, 1134, 1575]
```