Travaux dirigés n° 3

Fonctions - Listes

*** Corrigé indicatif ***

RETOUR SUR EXPÉRIENCE:
Afin d'améliorer ce support, annotez-le directement tant au niveau du sujet que de la correction fournie.

Date intervention :

Nom de l'intervenant :

Groupe de TD/TP :

Remarques générales :

Noubliez pas de me retourner votre support annoté. Merci d'avance, R. Girard

Une liste est une collection ordonnée et modifiable d'éléments éventuellement hétérogènes. Une liste se définit par une suite d'éléments séparés par des virgules. Cette suite est entourée par des crochets.
Les opérations sur les listes dans ce TD seront : la concaténation `+`, la répétition `*`, l'accès à un élément par son indice `[]` et l'extraction de sous-liste `[:]`.
Une fonction (ou *function*) est une suite d'instructions que l'on peut nommer et qui s'utilisent comme un élément d'une expression. Une fonction s'appelle par son nom en lui passant des paramètres. Elle se termine en retournant une valeur. Dans d'autres langages de programmation, il existe la notion de procédure qui est une fonction qui ne retourne aucune valeur. Elle s'utilise comme une instruction. En python, cette distinction n'existe pas, une fonction s'utilise aussi bien dans une expression que comme une instruction. La définition d'une fonction est la suivante :

```
def nomDeLaFonction(liste de paramètres):
    ...
    bloc d'instructions
    ...
```

*Exercice 3.1

1) Créez la liste `couleur` ne contenant aucun élément. Ensuite ajouter les chaînes de caractères suivantes comme éléments à cette liste : `rouge`, `vert`, `bleue`.

```
couleur=[]
couleur=couleur+['rouge','vert','bleue']
```

2) Écrire une expression conditionnelle qui affiche 'red' si `rouge` est un élément qui appartient à la liste `couleur`.

```
if 'rouge' in couleur :
   print('red')
```

3) Créez la liste `number` qui contient les nombres de 0 à 19. Astuce : utilisez la fonction `list()` et `range()`.

En utilisant l'opérateur d'extraction de sous-liste `[:]` créer à partir de la liste `number` une liste `l1` contenant les nombres de 4 à 8 et une liste `l1` contenant tous les nombres de 13 à 19.

```
number=list(range(0,20))
l1 = number[4:9]
l2 = number[13:]
```

4) Créez une liste `unzero` qui contient 100 fois le nombre 1 suivi d'un 0 et de 100 fois le nombre 1. Afficher la longueur de la liste ainsi créée. Quel est l'indice de l'élément 0 ?

```
unzero = [1] *100 + [0] + [1] * 100
print(len(unzero))
```

0 est le 101ème élément, comme les indices commencent à 0, l'indice du 0 est 100.

5) Définissez les listes `l3 = [1, 2, 3]` et `l4 = [20, 30, 40, 50, 60, 70]`.
A partir de `l3` et `l4` et en utilisant les opérateurs de concaténation, de répétition et d'extraction créez la liste `[2, 20, 30, 40, 50, 3, 3, 3, 3, 3]`

```
[l3[1]] + l4[:4] + [l3[2]]*5
```

⋆Exercice 3.2

En premier, vous allez écrire un programme qui propose en menu l'appel des fonctions que vous allez développer par la suite. Écrivez le programme qui affiche ce menu, qui saisi le choix de l'utilisateur, qui vérifie la validité sinon il recommence jusqu'à recevoir un choix valide puis qui exécute la fonction choisie dans le menu. Lorsque l'utilisateur rentre un mauvais choix, le message d'erreur "Erreur : choix invalide, choisir une valeur [1..3]" s'affiche avant de re-inviter l'utilisateur à saisir un nouveau choix.

```
1. indoccur : occurence d'une valeur dans une liste
2. que0et1 : tous les elements ont la même valeur : soit  0, soit 1
3. Quitter
Votre choix :
```

```
# function
# here definition
# main
# Display menu
print("1. indoccur : occurence d'une valeur dans une liste")
print("2. que0et1 : tous les elements ont la  valeur  0 ou 1")
print("3. Quitter")

choix=0
while not (choix >= 1 and choix <=3):
    choix =int(input("Votre choix : ? "))
    if not (choix >= 1 and choix <=3):
        print("Erreur: choix invalide, choisir une valeur [1..3]")

if choix == 1:
        print("appel de indoccur([3,2,1,4,1,1,5,8,3,1], 1)")
    elif choix == 2:
        print("appel de que0et1([0,0,0,1,0,0,1])")
```

⋆Exercice 3.3

Écrivez une fonction `indoccur` qui prend en paramètres une liste `l` et une valeur `v` et retourne comme résultat la liste des **indices** des occurrences de la valeur `v` dans la liste `l`.
Par exemple l'appel : `indoccur([3,2,1,4,1,1,5,8,3,1], 1)` doit retourner la liste `[2,4,5,9]` car 2,4,5,et 9 sont les indices où `1` apparaît dans la liste donnée.

On parcourt la liste `l` par ses indices et chaque fois qu'on rencontre la valeur `v` on ajoute l'indice correspondant dans la liste résultat.

```
def indoccur(l,e):
    res=[]
    for i in range(len(l)):
        if l[i]==e:
            res = res + [l[i]]
    return res
```

```
# Correction alternative avec enumerate
def indoccur_alt(l,e):
        res = []
        for i,a in enumerate(l):
        # enumerate(l): Retourne une sequence dont chaque element est une paire (indice, element) de la liste l
                if a == e:
                        res.append(i)
        return res
```

***Exercice 3.4**

Définissez en Python une fonction `que0et1` qui prend en paramètre un liste et qui retourne le booléen `True` si et seulement si les éléments de la liste donnée ne sont que des 0 et des 1.
Par exemple l'appel : `que0et1([0,0,0,1,0,0,1])` doit retourner True.

On suppose le résultat `True` par défaut et on le rend `False` si et seulement si en examinant successivement tous les éléments de la liste on en rencontre (au moins) un différent de 0 et différent de 1.
Les indices n'ont ici pas d'importance, on peut donc appliquer la boucle for directement sur la collection de valeurs formée par la liste.

```
def que0et1(l):
    res=True
    for n in l:
        if n != 1 and n != 0 :
            res=False
    return res
```

***Exercice 3.5**

Définissez en Python une fonction `elem_ind_pair` qui prend en paramètre une liste et qui retourne comme résultat une nouvelle liste (donc on ne modifie pas celle donnée en paramètre !) formée avec les éléments d'indice pair de la liste donnée.
Par exemple l'appel : `elem_ind_pair([3,7,9,1,2,0,4])` doit retourner la liste `[3,9,2,4]`

On ajoute dans la liste résultat, au départ vide, tous les éléments d'indice pair de la liste donnée. Il suffit pour cela de faire un parcours de la liste à partir de l'indice 0 avec un pas de 2.

```
def elem_ind_pair(l):
    res=[]
    for i in range(0,len(l),2):
      res+[ l[i] ]
    return res
```

***Exercice 3.6**

Définissez une fonction `est_triee` qui prend en paramètre une liste et qui retourne le booléen `True` si et seulement les valeurs de la liste sont triées (de gauche à droite) en ordre croissant.

On suppose la liste triée et on vérifie que pour tout indice i (depuis 0 jusqu'à l'avant dernier) on a bien `l[i] <= l[i+1]` ou en d'autre terme, s'il existe au moins un indice `i` tel que `l[i] > l[i+1]` alors la liste n'est pas triée en ordre croissant.

```
def est_triee(l):
    res=True
    for i in range(len(l)-1):
        if l[i] > l[i+1] :
            res=False
    return res
```

⋆Exercice 3.7

On a vu en cours qu'une liste en python est une collection ordonnée d'éléments de valeurs quelconques. En particulier les éléments d'une liste peuvent être des listes. Ainsi la liste `[[1,2,3], [4,5]]` est formée de deux éléments qui sont les listes `[1,2,3]` et `[4,5]`.
Écrivez une fonction `listesom` qui prend en paramètres une liste $l1$ de listes de nombres et qui retourne comme résultat la listes constituées des sommes des éléments de chacune des listes de $l1$.
Par exemple l'appel : `listesom([[1,2,3], [4], [5,6]])` doit retourner la liste `[6,4,11]`

On parcourt la liste l1, et pour chacune des listes l qui la compose, on calcule la somme des éléments de l qu'on ajoute dans la liste résultat.

```
def listesom(l1):
    res=[]
    for l in l1:
        som=0
        for n in l:
            som = som+n
        res = res + [[som]]
    return res
```

⋆Exercice 3.8

Écrivez en Python une fonction `singleton` qui prend en paramètre une liste quelconque `l` et qui retourne comme résultat une liste formées à partir des éléments de `l`, chacun placé seul dans une liste.
Par exemple l'appel : `singleton([1,2,3,4])` doit retourner la liste `[[1], [2], [3], [4]]`

```
def singleton(l):
    res=[]
    for e in l:
        res = res + [[e]]
    return res
```

⋆Exercice 3.9

Écrivez une fonction `fusion` qui prend en paramètres deux listes (pas forcément de mêmes longueurs) de valeurs **triées en ordre croissant** et qui retourne comme résultat une nouvelle liste **triée en ordre croissant** résultant de la fusion des deux listes données en paramètres.
Par exemple l'appel : `fusion([3,7,9,10], [1,3,4,8,9,12,15])` retourne comme résultat :
`[1,3,3,4,7,8,9,9,10,12,15]`

Le principe est le suivant : on parcours les listes `l1` et `l2` et on construit la liste résultat en insérant à chaque étape la plus petite des deux valeurs courantes de `l1` et `l2`. On passe à l'étape suivante en avançant dans `l1` (respectivement dans `l2`) si c'est la valeur de `l1` (respectivement de `l2`) qui a été insérée. Lorsqu'on arrive au bout d'une des deux listes, il reste à insérer les valeurs restant dans l'autre.

Attention de bien gérer les indices !

```python
def fusion(t1,t2):
    res=[]
    i1=0     # indice pour parcourir le tableau t1
    i2=0     # indice pour parcourir le tableau t2

    # tant qu'on n'est pas au bout d'un des tableaux
    while i1 < len(t1) and i2 < len(t2) :
        # on insère la plus petite des valeurs t1[i1] et t2[i2]
        if t1[i1] < t2[i2]:
            res = res + [t1[i1]]
            i1=i1+1
        else:
            res = res + [t2[i2]]
            i2=i2+1
    # si on est arrivé au bout du tableau t1, on insère les éléments restant de t2
    if i1 >= len(t1):
        res = res + t2[i2:]
    # sinon, on est arrivé au bout du tableau t2 et on insère les éléments restant de t1
    else:
        res = res + t1[i1:]
    return res
```

Exercice 3.10

Écrivez une fonction `carre01` qui prend en paramètre un entier n et qui retourne comme résultat la liste représentant un tableau carré de n lignes et n colonnes rempli de 1 sauf le tour qui est rempli de 0. Si $n < 2$ la fonction doit retourner une liste vide.

Par exemple l'appel `carre01(5)` doit renvoyer la liste :
`[[0,0,0,0,0], [0,1,1,1,0], [0,1,1,1,0], [0,1,1,1,0], [0,0,0,0,0]]` qu'on peut voir sous la forme :

```
 [[0,0,0,0,0],
  [0,1,1,1,0],
  [0,1,1,1,0],
  [0,1,1,1,0],
  [0,0,0,0,0]]
```

```python
def carre01(n):
    res=[]
    if n > 1:
        res = res + [ [0]*n ]
        for i in range(1,n-1):
            res =res+ [[0]+[1]*(n-2)+[0]]
        res =res + [ [0]*n ]
    return res
```

ou bien

```python
def exo5(n):
    if n>2:
        return [ [0]*n ] + (n-2) * [[0]+[1]*(n-2)+[0]] + [[0]*n]
    else :
        return []
```