Partie 1

Listes

On propose l'**implémentation** suivante pour les listes chainées :

```
def creer_liste():
  """exemple:
  L = creer_liste()
  """
  return ()

def liste_vide(L):
  """exemple:
  liste_vide(L)
  """
  return L == ()

def inserer_tete(L,e):
  """exemple:
  L = inserer_tete(L,e)
  """
  return (e,L)

def tete(L):
  """exemple:
  L = (1,(2,()))
  tete(L) -> 1
  """
  return L[0]

def queue(L):
  """exemple:
  L = (1,(2,()))
  queue(L) -> (2,())
  """
  return L[-1]

def inserer(L,element_recherche,element_a_inserer):
  """exemple:
  L = (1,(2,()))
  inserer(L,2,3) -> (1,(3,(2,())))
  """
  Liste_des_elements = []
  # recherche
  while element_recherche != tete(L):
    Liste_des_elements.append(tete(L))
    L = queue(L)
  # insertion
  L = inserer_tete(L,element_a_inserer)
  for i in range(len(Liste_des_elements)-1,-1,-1):
    L = inserer_tete(L,Liste_des_elements[i])
  return L
```

```
48
49 def elements_liste(L):
50     """exemple:
51     L2 = elements_liste(L)
52     """
53     # à completer
54     #
```

1.1 Exercice 1 : parcours scolaire

Le `parcours` scolaire est un type abstrait qui s'apparente à une Liste. Les éléments sont disposés dans cette Liste sous la forme :

```
1 ('Terminale', ('Premiere', ('Seconde', ())))
```

L'interface propose les fonctions suivantes :

`creer_liste, liste_vide, inserer, tete, queue.`

1. Quelles instructions, utilisant l'interface, vont créer la Liste `parcours_lycee` de la manière suivante : `('Terminale', ('Premiere', ('Seconde', ())))` ?

2. Quelle instruction va permettre de connaitre la dernière classe visitée lors du parcours scolaire ?

3. Que retourne `queue(parcours)` ? *(choisir)*

- la première classe du parcours scolaire
- le parcours scolaire sans la dernière année

4. Que retourne le script suivant ?

```
1 while not liste_vide(parcours):
2     print(tete(parcours))
3     parcours = queue(parcours)
```

5. On souhaite utiliser ce type abstrait pour décrire le parcours universitaire. Quelle serie d'instructions va créer la structure de données du parcours qui ira de `Licence 1` à `Master 2` ? Cette liste s'appelera `parcours_univ`

6. On souhaite maintenant créer une liste unique appelée `scolarite`, issue de la jonction des deux listes, `parcours_lycee` et `parcours_univ`. Ecrire le script correspondant.

1.2 Exercice 2 : composition d'un train

On cherche à implémenter la construction d'un train pour voyageurs, motrice et wagons, à l'aide d'une liste chainée.

Le train suit l'ordre suivant :

- motrice
- wagon_1
- wagon_2
- wagon_3

1. Qu'est ce qui est affiché par le programme suivant ?

```
L1 = creer_liste()
L1 = inserer(L1,'motrice')
print(L1)
L1 = inserer(L1,'wagon_1')
print(L1)
L1 = inserer(L1,'wagon_2')
print(L1)
L1 = inserer(L1,'wagon_3')
print(L1)
```

2. Quelle instruction de l'interface de liste faut-il utiliser pour consulter la nature du dernier wagon du train ?

3. La fonction `elements_liste` va retourner les éléments de la liste chainée de la manière suivante : `['wagon_3', 'wagon_2', 'wagon_1','motrice']`

```
elements_liste(L1)
# affiche ['wagon_3', 'wagon_2', 'wagon_1','motrice']
```

Compléter le script de cette fonction.

```
def elements_liste(L):
  Liste_des_elements = []
  while not liste_vide(L):
    element = ... (L)
    L = ... (L)
    Liste_des_elements.append(...)
  return ...
```

4. INSERTION : On souhaite modifier la liste L1 et intercaler `wagon_bar` entre le `wagon_1` et le `wagon_2`.

- Quelle instruction utilisant la fonction `insere` de l'interface de liste va modifier L1 de cette façon ?
- Quelle sera l'objet retourné ?
- Expliquer en détail les différentes parties de cette fonction.

5. Supprimer le wagon de queue à l'aide d'une instruction de l'interface.

1.3 Exercice 3 : separation d'une liste

1. Compléter la fonction `separe` qui sépare les éléments d'une liste en deux listes selon s'ils sont inférieurs (strictement) ou supérieurs (et égal) à une valeur `v` :

```
def separe(L,v):
  L_inf = creer_liste()
  L_sup = creer_liste()
  while not liste_vide(L):
    # à completer
    # utilise les fonctions tete, queue et insere_tete
    #
    #
  return L_inf, L_sup
```

2. Compléter alors le programme au niveau du repère ICI. Le programme affiche les 2 listes une fois celles-ci séparées. La liste L contiendra les valeurs à séparer.

```
L = creer_liste()
for elem in [1,3,20,18,16,11,101,12,15,2,5,4,2,8,1]:
    L = inserer_tete(L,elem)

v = 12
# ICI, appel de la fonction separe
L_inf, L_sup = separe(L,v)
print(L_inf)
print(L_sup)
# (1, (3, (11, (2, (5, (4, (2, (8, (1, ()))))))))))
# (20, (18, (16, (101, (12, (15, ()))))))
```

Partie 2

Tableaux statiques

On propose l'implémentation suivante pour les tableaux statiques :

```
T = (['lundi','mardi','mercredi','jeudi','vendredi'],5)

def taille(T):
  """exemple:
  > taille(T)
  > 5
  """
  return T[1]

def element(T,i):
  """exemple:
  > element(T,3)
  > 'jeudi'
  """
  return T[0][i]

def remplacer(T,i,e):
  """exemple:
  > remplacer(T,2,'jour des enfants')
  > T
  > (['lundi','mardi','jour des enfants','jeudi','vendredi'],5)
  """
  # à completer
```

2.1 Exercice 1 : Utiliser l'interface du tableau

{{< img src="../images/array.png" caption="Tableau de notes" >}}

1. Soit le tableau qui implémente les notes de Kyle dans les matières C1, C2, C3 :

```
T = ([6, 7, 8], 3)
```

a. Ecrire l'instruction qui retourne la note de Kyle dans la matière C3.

b. Kyle n'a pas eu 8, mais 12/20 dans la matière C3. Ecrire l'instruction qui modifie sa note 8 en 12, à partir d'une fonction de l'interface des tableaux.

c. Compléter le script donné plus haut pour cette fonction.

d. Ecrire le script d'une fonction `moyenne` qui calcule la moyenne des notes du tableau T.

2. Soit le tableau suivant qui implémente les notes de Kyle, Sean, Quentin et Zinedine dans les matières C1, C2, C3 :

```
T = ([[6, 7, 8],
      [10, 0, 10],
      ['?', '?', '?']
      ['?', '?', '?']], (4,3))
```

a. Recopier le tableau T et remplacer les '?' par les bonnes valeurs (utiliser l'image plus haut). Que signifie le tuple `(4,3)` placé comme 2e élément du tableau ?

b. Utiliser l'instruction `taille` de l'interface pour donner le nombre d'élèves dans le tableau. Puis le nombre de colonnes (matières) :

```
n_eleves = taille(...)
n_matieres = taille(...)
```

c. Modifier la fonction `remplacer` pour que celle-ci modifie la note dans la matière voulue pour un élève donné.

Par exemple : Pour modifier la note de la colonne 2 de l'élève au rang 0 (premiere ligne) :

Si on veut lui mettre 12, on fera : `remplacer(T,0,2,12)`

```
remplacer(T,0,2,12)
T
# affiche
([[6, 7, 12],
 [10, 0, 10],
 ['?', '?', '?']
 ['?', '?', '?']], ..)
```

3. Quelle instruction, utilisant la fonction `moyenne` va retourner la moyenne des notes de Zinedine ?

4. `moyenne_matiere` Programmer la fonction `moyenne_matiere` qui va calculer la moyenne sur une matière pour la colonne `j`.

Pour programmer cette fonction `moyenne_matiere(j)`, il faudra :

- Faire la somme $M[0][j] + M[1][j] + M[2][j] + M[3][j]$. Utiliser un **accumulateur `s` dans une boucle bornée**.
- Diviser `s` par le nombre de notes
- retourner la valeur

2.2 Exercice 2 : moyenne glissante et tableau statique

Les courbes de données issues du monde réel sont souvent *bruitées*. Pour simuler ce type de données, nous allons utiliser une liste de valeurs cumulées aléatoires.

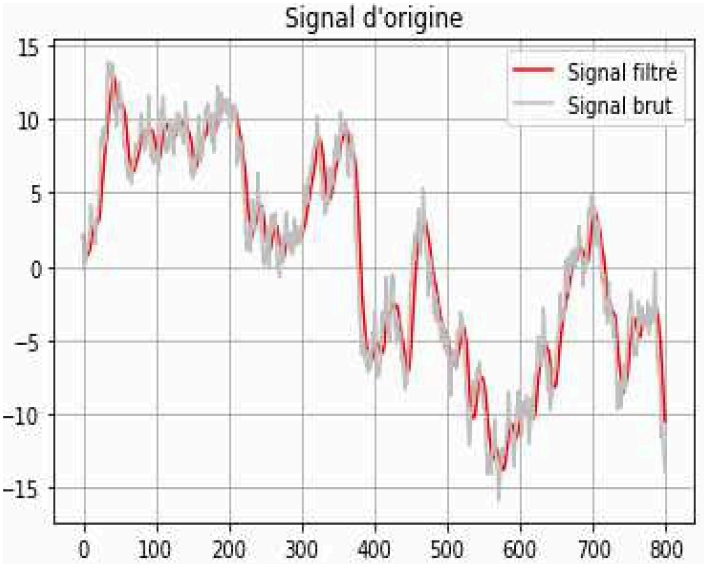

FIGURE 1 – courbes de donnees brutes et données filtrees

Etapes du traitement : Fonction `lissage`

- On fait une copie par valeur de `signal2` (liste des valeurs bruitées) dans `signal_filtre` (liste dont les termes seront remplacés par la valeur moyennée) :

```
signal_filtre = signal2.copy()
```

- On choisit une certaine largeur de liste pour les valeurs dont on fait la moyenne. Par exemple largeur = 10.
- Au rang i, dans la liste bruitée `signal2` : On prélève les valeurs entre les rangs $i - largeur//2$ et $i + largeur//2$ que l'on stocke dans un tableau STATIQUE appelé `signal`. Ce tableau conserve la même *taille* pendant tout l'exercice. (ici, largeur = 10) :

```
signal = signal2[i-largeur//2:i+largeur//2]
```

- On calcule la moyenne de valeurs de `signal` avec la fonction `moyenne`, comme par exemple celle vue dans l'exercice précédent.

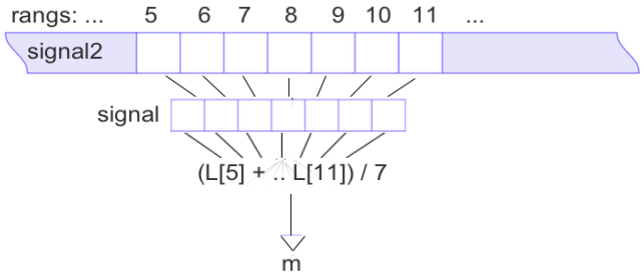

FIGURE 2 – moyenne sur l'ensemble des valeurs de `signal`

```
signal_filtre[i] = moyenne(signal)
```

- On place la valeur moyenne dans `signal_filtre[i]`

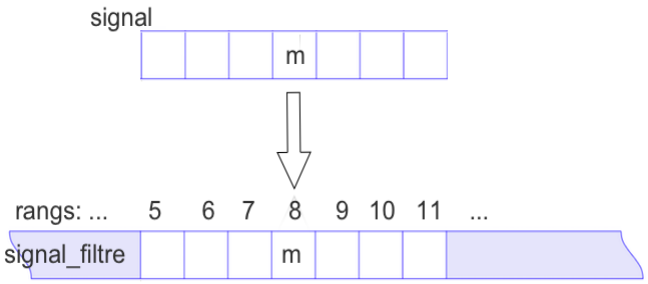

FIGURE 3 – signal filtre contient 10 valeur identiques contigües, il n'y a plus de bruit

- On répète l'opération pour tous les index `i` compris entre `largeur//2,len(signal2)-largeur//2)` :

```
for i in range(largeur//2,len(signal2)-largeur//2):
```

Puis on affiche les graphique de `signal2` (inchangé) et celle du signal après traitement (courbe lissée).

Nous allons suivre ces étapes :

1. Ecrire le script de la fonction `moyenne`

2. Créer une fonction `lissage`, qui prend en paramètres une liste `L` et une largeur `v`, et retourne une liste de valeurs filtrées par une moyenne glissante sur `v` valeurs.

3. Placer les valeurs filtrées dans une liste `signal_apres_traitement` : `signal_apres_traitement = lissage(signal2,10)`

4. Afficher les 2 courbes, `signal2` et `signal_apres_traitement`

Correction

```
def moyenne(signal):
    s = 0
    b = len(signal)
    for elem in signal:
        s += elem
    return s / b



def lissage(L,largeur):
    signal_filtre = L.copy()
    for i in range(largeur//2,len(L)-largeur//2):
        signal = L[i-largeur//2:i+largeur//2]
        # signal est le petit tableau de dimension largeur
        # dont on fait la moyenne glissante
        # puis on stocke dans signal_filtre
        signal_filtre[i] = moyenne(signal)
    return signal_filtre

signal_apres_traitement = lissage(signal2,10)
```