N. S. I. Term. - DS03 (jeudi 16 novembre 2023)

Exercice 1.

On considère le dictionnaire suivant, représentant les données relatives à un élève :

```
eleve1 = {'Nom' : 'BERNARD', 'Prénom' : 'Claude', 'Genre' : 'M',
          'Date de naissance' : ''}
```

Q1. Écrire une instruction, utilisant la fonction `print`, pour afficher, en utilisant les données enregistrées dans le dictionnaire `eleve1`, le prénom et le nom de cet élève, sous la forme : `Claude BERNARD`.

Q2. Écrire une instruction, pour que la date de naissance de cet élève, non renseignée initialement, soit mise à la valeur 06/12/2007 (sous la forme d'une chaîne de caractères).

Q3. Écrire une instruction pour supprimer du dictionnaire toute information relative au genre de l'élève (clé et valeur).

Q4. Écrire une instruction pour que l'information sur la classe de l'élève soit ajoutée au dictionnaire. L'élève sera supposé être scolarisée dans la classe nommée T5.

Q5. À l'aide de quelle(s) instruction(s), grâce à un parcours par les couples (clé, valeur), peut-on afficher toutes les informations sur l'élève.

Exercice 2. Pratique des tris

Q1. Pour les tris vus en classe (tri-bulle, tri-sélection, tri-insertion), l'algorithme comporte toujours deux boucles imbriquées, la boucle intérieure effectuant l'opération caractéristique du tri (redressement des inversions, sélection ou insertion). Combien de fois au maximum doit-on répéter l'étape caractéristique du tri pour trier une liste de longueur 5 ? De façon générale, combien de répétitions de l'opération élémentaire du tri sont nécessaires pour trier une liste L ?

Q2. Lors d'un tri-bulle, quel est l'état de la liste suivante après le premier parcours de la liste :

```
L1 = [19, 20, 7, 41, 11, 13, 10, 59, 24, 53]
```

Q3. Lors d'un tri-sélection dans lequel à chaque étape on recherche un maximum, que l'on repositionne à l'aide d'un unique échange, donner les états de la liste L suivante à la fin de chacun des trois premières étapes. Il y a trois états de la liste à donner.

```
L1 = [19, 20, 7, 41, 11, 13, 10, 59, 24, 53]
```

Q4. Lors d'un tri-insertion effectué comme en classe, donner l'état de la liste suivante après les cinq premières étapes du tri. Il y a un seul état de la liste à donner.

```
L1 = [19, 20, 7, 41, 11, 13, 10, 59, 24, 53]
```

Exercice 3. Diverses fonctions réalisant des opérations utiles aux tris

Q1. Écrire une fonction `inversions(L)` prenant en argument une liste `L` d'entiers ou de flottants et renvoyant le nombre de fois où deux valeurs consécutives dans la liste ne sont pas rangées dans l'ordre croissant.

Q2. Modifier la fonction précédente en une fonction `positions_inversions(L)`, construisant et renvoyant la liste des indices `i` tels que `L[i] > L[i+1]`.

Q3. Écrire une fonction `echange(L, i, j)` échangeant dans la liste `L` les éléments en position `i` et `j`. On utilisera une assertion (`assert`) pour vérifier que les indices i et j sont bien des positions positives valides.

Q4. Écrire une fonction `minL(L)` renvoyant le minimum d'une liste L non vide d'entiers ou de flottants.

Q5. Écrire une fonction `posminL(L)` renvoyant la position de la première occurrence du minimum d'une liste L non vide d'entiers ou de flottants.

Q6. Écrire une fonction `selectionminL(L, i)` renvoyant la valeur et la position de la première occurrence du minimum des éléments **situés en position `i` et au-delà** d'une liste L non vide d'entiers ou de flottants.

Q7. Écrire une fonction `position_insertion(L, x)`, prenant en entrée une liste croissante, L, d'entiers ou de flottants, et une valeur x, et renvoyant la plus petite position i telle que $x < L[i]$ si une telle position existe et `len(L)` sinon.

On impose d'utiliser une boucle `while`.

On devra avoir, par exemple :

```
>>> position_insertion([2, 6, 6, 8], 3)
1
>>> position_insertion([2, 6, 6, 8, 10], 6)
3
>>> position_insertion([2, 6, 8], 8)
3
```

Exercice 4.

La fonction `tri_bulles` prend en paramètre une liste `T` d'entiers non triés et renvoie la liste triée par ordre croissant.

Le tri à bulles est un tri en place qui commence par placer le plus grand élément en dernière position en parcourant la liste de gauche à droite et en échangeant au passage les éléments voisins mal ordonnés (si la valeur de l'élément d'indice `i` a une valeur strictement supérieure à celle de l'indice `i + 1`, ils sont échangés). Le tri place ensuite en avant-dernière position le plus grand élément de la liste privée de son dernier élément en procédant encore à des échanges d'éléments voisins. Ce principe est répété jusqu'à placer le minimum en première position.

Exemple : pour trier la liste `[7, 9, 4, 3]` :

- première étape : 7 et 9 ne sont pas échangés, puis 9 et 4 sont échangés, puis 9 et 3 sont échangés, la liste est alors `[7, 4, 3, 9]`
- deuxième étape : 7 et 4 sont échangés, puis 7 et 3 sont échangés, la liste est alors `[4, 3, 7, 9]`
- troisième étape : 4 et 3 sont échangés, la liste est alors `[3, 4, 7, 9]`

Compléter le code Python ci-dessous qui implémente la fonction `tri_bulles`.

```
def tri_bulles(T):
    '''
    Renvoie le tableau T trié par ordre croissant
    '''
    n = len(T)
    for i in range(...,...,-1):
        for j in range(i):
            if T[j] > T[...]:
                ... = T[j]
                T[j] = T[...]
                T[j+1] = temp
    return T
```

Exemples :

```
>>> tri_bulles([])
[]
>>> tri_bulles([7])
[7]
>>> tri_bulles([9, 3, 7, 2, 3, 1, 6])
[1, 2, 3, 3, 6, 7, 9]
>>> tri_bulles([9, 7, 4, 3])
[3, 4, 7, 9]
```

Exercice 5.

On considère l'algorithme de tri de tableau suivant : à chaque étape, on parcourt le sous-tableau des éléments non rangés et on place le plus petit élément en première position de ce sous-tableau.

Exemple avec le tableau :

```
t = [41, 55, 21, 18, 12, 6, 25]
```

Etape 1 : on parcourt tous les éléments du tableau, on permute le plus petit élément avec le premier. Le tableau devient

```
t = [6, 55, 21, 18, 12, 41, 25]
```

Etape 2 : on parcourt tous les éléments **sauf le premier**, on permute le plus petit élément trouvé avec le second. Le tableau devient :

```
t = [6, 12, 21, 18, 55, 41, 25]
```

Et ainsi de suite.

La code de la fonction `tri_selection` qui implémente cet algorithme est donné ci-dessous.

```
def tri_selection(tab):
    N = len(tab)
    for k in range(...):
        imin = ...
        for i in range(... , N):
            if tab[i] < ... :
                imin = i
        ... , tab[imin] = tab[imin] , ...
```

Compléter le code de cette fonction de façon à obtenir :

```
>>> liste = [41, 55, 21, 18, 12, 6, 25]
>>> tri_selection(liste)
>>> liste
[6, 12, 18, 21, 25, 41, 55]
```

On rappelle que l'instruction

```
a, b = b, a
```

échange les contenus de `a` et de `b`.

Exercice 6.

La fonction `tri_insertion` suivante prend en argument une liste `tab` et trie cette liste en utilisant la méthode du tri par insertion. Compléter cette fonction pour qu'elle réponde à la spécification demandée.

On rappelle le principe du tri par insertion : on considère les éléments à trier un par un, le premier élément constituant, à lui tout seul, une liste triée de longueur 1. On range ensuite le second élément pour constituer une liste triée de longueur 2, puis on range le troisième élément pour avoir une liste triée de longueur 3 et ainsi de suite… A chaque étape, le premier élément de la sous-liste non triée est placé dans la sous-liste des éléments déjà triés de sorte que cette sous-liste demeure triée.
Le principe du tri par insertion est donc d'insérer à la n-ième itération, le n-ième élément à la bonne place.

```
def tri_insertion(tab):


n = len(tab)
for i in range(1, n):
    valeur_insertion = tab[...]
    # la variable j est utilisée pour déterminer où placer la
valeur à insérer
    j = ...
    # tant qu'on a pas trouvé la place de l'élément à insérer
    # on décale les valeurs du tableau vers la droite
    while j > ... and valeur_insertion < tab[...]:
        tab[j] = tab[j-1]
        j = ...
    tab[j] = ...
```

Exemple :

```
>>> liste = [9, 5, 8, 4, 0, 2, 7, 1, 10, 3, 6]

>>> tri_insertion(liste)

>>> liste
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```