Python – Tableaux

Modification de tableaux

Pour tous les exercices suivants, on modifiera directement le tableau passé en paramètre. Puisque les listes sont mutables en Python, il n'est pas nécessaire de mettre de **return**. Néanmoins, pour des besoin de tests, il peut être pratique d'afficher la valeur du tableau à la fin de la fonction.

Exercice 1 : Compléter la fonction `ajoute(tab, a)` qui ajoute `a` à tous les éléments du tableau de nombres `tab`.

```
def ajoute(tab, a):
    for i in range(len(tab)):
        tab[i] = ...
```

```
>>> tab = [2, 5, -1, 7]
>>> ajoute(tab, 100)
>>> tab
[102, 105, 99, 107]
```

Exercice 2 : Écrire une fonction `affine(tab, m, p)` qui applique la fonction $x \mapsto mx + p$ à tous les éléments du tableau de nombres `tab`.

```
>>> tab = [2, 5, -1, 7]
>>> affine(tab, -2, 4)
>>> tab
[0, -6, 6, -10]
```

Exercice 3 : Écrire une fonction `remplace(tab, a, b)` remplace toutes les occurrences de `a` par `b` dans le tableau `tab`.

```
>>> notes = [17, 5, 10, 13, 5, 11]
>>> remplace(notes, 5, 10)
>>> notes
[17, 10, 10, 13, 10, 11]
>>> phrase = ['Vive', 'la', 'spécialité', 'SPC']
>>> remplace(phrase, 'SPC', 'NSI')
['Vive', 'la', 'spécialité', 'NSI']
```

Exercice 4 : Écrire une fonction `interverti(tab, a, b)` qui prend un tableau `tab` et deux valeurs `a` et `b`, et qui remplace toutes les occurrences de `a` par `b` et celles de `b` par `a` dans `tab`.

```
>>> tab = [7, 9, 1, 45, 28, 16, 9, 28, 28]
>>> interverti(tab, 1, 28)
>>> tab
[7, 9, 28, 45, 1, 16, 9, 1, 1]
>>> phrase = ['chien', '<', 'chat']
>>> interverti(phrase, 'chien', 'chat')
>>> phrase
['chat', '<', 'chien']
```

Exercice 5 : Écrire une fonction `normalise(tab, a, b)` qui prend deux nombres `a` et `b` tels que `a < b` et qui remplace tous les éléments de `tab` inférieurs à `a` par `a` et tous ceux supérieurs à `b` par `b`.

```
>>> tab = [15, -7, 1, 9, 11]
>>> normalise(tab, 0, 10)
>>> tab
[10, 0, 1, 9, 10]
```

Exercice 6 : Écrire une fonction `echange(tab, i, j)` qui permute la valeur d'indice `i` avec celui d'indice `j` dans `tab`.

```
>>> tab = [15, -7, 1, 9, 11]
>>> echange(tab, 0, 2)
>>> tab
[1, -7, 15, 9, 11]
```

Exercice 7 : Écrire une fonction `melange(tab)` qui fait un mélange aléatoire du tableau `tab`. Un algorithme simple pour faire cela consiste à parcourir les indices `i` du tableau et d'échanger la valeur en position `i` avec celle d'indice `j` tiré au hasard entre `0` et `i` inclus. Pensez à rajouter "`from random import randint`" au début de votre fichier. Pour tirer un entier entre `a` et `b` inclus, la commande est `randint(a, b)`.

```
>>> tab = list(range(10))
>>> melange(tab)
>>> tab
[1, 6, 5, 9, 2, 0, 7, 8, 4, 3]
>>> melange(tab)
>>> tab
[5, 1, 7, 0, 2, 8, 4, 9, 3, 6]
>>> melange(tab)
>>> tab
[5, 6, 1, 4, 2, 0, 8, 9, 7, 3]
```

Exercice 8 : Écrire une fonction `miroir(tab)` qui inverse l'ordre des éléments de `tab` Le premier devenant le dernier et ainsi de suite. Pour rappel le dernier élément a pour indice `n-1`.

```
>>> tab = [1, 5, 2 ,9, -1, 8]
>>> miroir(tab)
>>> tab
[8, -1, 9, 2, 5, 1]
```

Tableaux à deux dimensions

Pour toute cette partie, on considérera des tableaux à 2 dimensions où tous les sous-tableaux ont la même taille. On suppose que les tableaux sont au minimum de taille 1×1.

Exercice 9 : Écrire une fonction `initialiser(m, n, val=0)` qui renvoie un tableau de taille $m \times n$ rempli avec la valeur `val` qui vaut 0 par défaut.

```
>>> initialiser(2, 3)
[[0, 0, 0], [0, 0, 0]]
>>> initialiser(2, 3, 9)
[[9, 9, 9], [9, 9, 9]]
```

Exercice 10 : Écrire une fonction `affichage_simple(tableau)` qui affiche le tableau ligne par ligne.

```
>>> affichage_simple([[-21, -89, -121], [-17, -99, -151], [-25, -36, -94]])
[-21, -89, -121]
[-17, -99, -151]
[-25, -36, -94]
```

Exercice 11 : Écrire une fonction `maximum(tableau)` qui renvoie le triplet `(maxi, i, j)` où `maxi` est la valeur maximale du tableau et `tableau[i][j]` = `maxi`. Si cette valeur apparaît plusieurs fois dans le tableau, vous pouvez décider quelle position sera utilisée pour la réponse.

```
>>> maximum([[1, 2, 4], [2, 1, 3]])
(4, 0, 2)
>>> maximum([[-21, -89, -121], [-17, -99, -151], [-25, -36, -94]])
(-17, 1, 0)
```

Exercice 12 : Compléter la fonction `diagonale1(n)` qui renvoie un tableau de taille $n \times n$ qui ne contient que des 0, sauf sur la diagonale allant du coin en haut à gauche au coin en bas à droite.

```
def diagonale1(n):
    tab = initialiser(n, n)
    for i in range(n):
        tab[...][...] = 1
    return tab
```

```
>>> affichage_simple(diagonale1(5))
[1, 0, 0, 0, 0]
[0, 1, 0, 0, 0]
[0, 0, 1, 0, 0]
[0, 0, 0, 1, 0]
[0, 0, 0, 0, 1]
```

Exercice 13 : Écrire une fonction `diagonale2(n)` qui renvoie un tableau de taille $n \times n$ qui ne contient que des 0, sauf sur la diagonale allant du coin en haut à droite au coin en bas à gauche.

```
>>> affichage_simple(diagonale2(5))
[0, 0, 0, 0, 1]
[0, 0, 0, 1, 0]
[0, 0, 1, 0, 0]
[0, 1, 0, 0, 0]
[1, 0, 0, 0, 0]
```

Exercice 14 : Écrire une fonction `diagonales(n)` qui renvoie un tableau de taille $n \times n$ qui ne contient que des 0, sauf sur les deux diagonales.

```
>>> affichage_simple(diagonales(5))
[1, 0, 0, 0, 1]
[0, 1, 0, 1, 0]
[0, 0, 1, 0, 0]
[0, 1, 0, 1, 0]
[1, 0, 0, 0, 1]
```

Exercice 15 : Écrire une fonction `que_col(n, k)` qui renvoie un tableau de taille $n \times n$ qui ne contient que des 0, sauf sur la colonne d'indice `k` qui contient des 1.

```
>>> affichage_simple(que_col(5, 3))
[0, 0, 0, 1, 0]
[0, 0, 0, 1, 0]
[0, 0, 0, 1, 0]
[0, 0, 0, 1, 0]
[0, 0, 0, 1, 0]
```

Exercice 16 : Compléter la fonction `que_col_paires(n, k)` qui renvoie un tableau de taille $n \times n$ qui ne contient que des 0, sauf sur les colonnes d'indice pair qui contient des 1.

```
def que_col_paires(n):
    tab = initialiser(n, n)
    for i in range(n):
        for j in range(n):
            if ...%2 == 0:
                tab[i][j] = 1
    return tab
```

```
>>> affichage_simple(que_col_paires(5))
[1, 0, 1, 0, 1]
[1, 0, 1, 0, 1]
[1, 0, 1, 0, 1]
[1, 0, 1, 0, 1]
[1, 0, 1, 0, 1]
```

Exercice 17 : Écrire une fonction `addition(n)` qui renvoie un tableau de taille $n \times n$ dont chaque case contient la somme de l'indice de la ligne et celui de la colonne.

```
>>> affichage_simple(addition(5))
[0, 1, 2, 3, 4]
[1, 2, 3, 4, 5]
[2, 3, 4, 5, 6]
[3, 4, 5, 6, 7]
[4, 5, 6, 7, 8]
```

Exercice 18 : Écrire une fonction `damier(n)` qui renvoie un damier de taille $n \times n$ composé de 0 et de 1. On pourra s'inspirer de tableau obtenu avec `addition(n)` et comparer la parité des sommes avec la case voulue sur le damier.

```
>>> affichage_simple(damier(5))
[0, 1, 0, 1, 0]
[1, 0, 1, 0, 1]
[0, 1, 0, 1, 0]
[1, 0, 1, 0, 1]
[0, 1, 0, 1, 0]
```

Exercice 19 : Écrire une fonction `transposer(tab)` qui prend `tab` de taille $m \times n$ et renvoie `tab2` de taille $n \times m$ tel que `tab[i][j] = tab2[j][i]`. Il faudra peut-être commencer par déterminer les dimensions de `tab`.

```
>>> transposer([[10, 11, 12], [20, 21, 22]])
[[10, 20], [11, 21], [12, 22]]
```

On considère la fonction ci-dessous qui permet de faire l'affichage d'un tableau à 2 dimensions, en indiquant les indices :

```
def affichage_avance(tableau, label=""):
    maxi, _, _ = maximum(tableau)
    m = len(tableau)
    n = len(tableau[0])
    plus_grande_valeur = max(maxi, m-1, n-1)
    k = len(str(plus_grande_valeur))+1
    k2 = max(k, len(label))
    print(label.rjust(k2) + "|" +\
          "".join([str(i).rjust(k) for i in range(n)]))
    print("-"*k2 + "+" + "-"*(k*n))
    for i in range(m):
        ligne = str(i).rjust(k2)+"|"+\
                "".join([str(tableau[i][j]).rjust(k) for j in range(n)])
        print(ligne)
```

```
>>> affichage_avance([[21, 89, 121], [17, 99, 151], [25, 36, 94]], "Tableau")
Tableau|   0   1   2
-------+------------
      0|  21  89 121
      1|  17  99 151
      2|  25  36  94
```

Vous pouvez utiliser cette fonction pour l'exercice suivant.

Exercice 20 : Écrire une fonction `table_addition(n)` qui affiche un tableau d'addition de $a+b$ avec a et b allant de 0 à $n-1$.

```
>>> table_addition(4)
 +| 0 1 2 3
--+--------
 0| 0 1 2 3
 1| 1 2 3 4
 2| 2 3 4 5
 3| 3 4 5 6
```

Pour aller plus loin

Exercice 21 : La fonction `affichage_avance` ne fonctionne pas bien si le tableau ne contient que des valeurs négatives. Modifier la fonction, ou en écrire une nouvelle qui permet de gérer les nombres négatifs. Vous pouvez par exemple écrire une nouvelle fonction `taille_maximum` qui renvoie la longueur de l'élément le plus long à écrire dans le tableau. Par exemple `-12` est plus long à écrire que `99`.