Nom, prénom :

Évaluation NSI terminal

-

Diviser pour régner et Programmation dynamique

<u>**Exercice 1**</u> **: Compléter l'algorithme du Tri fusion.** **(5 points)**

```
def fusion(l1, l2):
    l = []
    i, j = (0,0)
    while ___________________ or ___________________:
        if i == len(l1):
            l.append(___________________)
            j += 1
        elif j == len(l2):
            l.append(___________________)
            i += 1
        elif l1[i] < l2[j]:
            l.append(___________________)
            i += 1
        else:
            l.append(___________________)
            j += 1
    return ___________________

def tri_fusion(lst):
    if ___________________:
        return lst
    else:
        m = len(lst)_____          # médiane
        l1 = tri_fusion(lst[:m])   # sous-liste de gauche
        l2 = tri_fusion(lst[m:])   # sous-liste de droite
        return ___________________

print(tri_fusion([38, 27, 43, 3, 9, 82, 10]))
```

Exercice 2 : Pyramide de nombres **(15 points)**

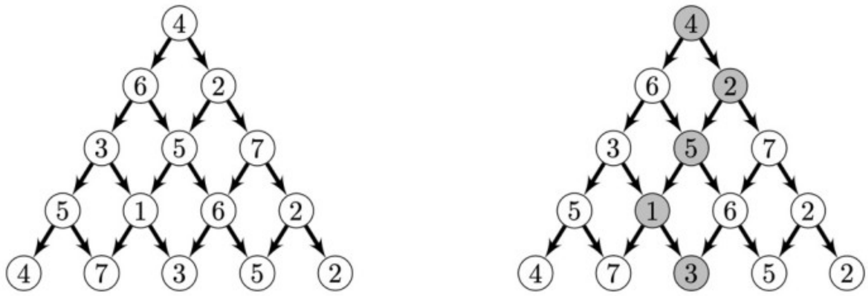

Figure 1.

La pyramide de nombre de la *Figure 1* est représentée par la liste de listes suivante :

`p = [[4],[6,2],[3,5,7],[5,1,6,2],[4,7,3,5,2]].`

L'objectif est de trouver le chemin de valeur maximale partant du sommet de la pyramide `p[0][0]` et descendant vers la base `p[len(p)-1][...]`. La valeur d'un chemin est égale à la somme des nombres qui le composent et est appelé *score*. On cherche le *score maximal* de la pyramide.
La partie droite de la *Figure 1* représente un chemin possible du sommet à la base de la pyramide, mais qui n'est pas de valeur maximale. Son score est 4 + 2 + 5 + 1 + 3 = 15.

Relation de récurrence

Chaque nombre de la pyramide possède deux sous-nombres, un sous-nombre droite et un sous-nombre gauche, représentés par des flèches sur la *Figure 1*.

Pour trouver le score maximal d'un nombre dans la sous-pyramide ayant pour sommet ce nombre, il faut calculer les score maximal de ses sous-nombres de droite et de gauche, ainsi :

- Pour un nombre qui n'est pas à la base de la pyramide, son score maximal est égale à la somme de :
  - son nombre
  - et le maximum entre :
    - le score maximal de son sous-nombre droite
    - le score maximal de son sous-nombre gauche.
- Pour un nombre de la base de la pyramide, son score maximal est égale à son nombre.

Si on note `score_max(i,j)` le score maximal possible depuis le nombre d'indice `j` du niveau `i` pour une pyramide `p`, on a alors les relations précédentes se traduisent en :

```
score_max(i, j, p) = p[i][j] + max(score_max(i+1, j, p), score_max(i+1, j+1, p))
score_max(len(p)-1, j, p) = p[len(p)-1][j]
```

La score maximal pour `p` toute entière sera alors `score_max(0,0,p)`.

Question 1 : Compléter la fonction `score_max` qui implémentent les règles précédentes.

```
def score_max(i,j,p):
    if i == ___________________:    # Base de la pyramide
        return ___________________
    else:                          # Autres étages
        return ___________________ + max(_________________________
                                                 ,_________________________)
print(score_max(_____ ,_____ ,p))
```

Si on suit à la lettre la définition de `score_max`, on obtient une résolution dont le coût est prohibitif à cause de la redondance des calculs. Par exemple `score_max(3, 1, p)` va être calculé pour chaque appel à `score_max(2, 0, p)` et `score_max(2, 1, p)`.

Pour éviter cette redondance, on décide de mettre en place des approches par programmation dynamique. Pour cela, on va construire une pyramide vide `s` de même dimension que p dans laquelle on va mémoriser les valeurs maximale déjà calculées. La valeur à l'indice `j` du niveau `i` sera égale à `score_max(i, j, p)`, c'est-à-dire à la valeur maximal nombre correspondant dans `p`.

<u>Question 2</u> : Complété la fonction `score_max_dyn` qui résout le problème récursivement.

```
def pyramide_nulle(n):
    p = []
    for i in range(1, n+1):
        p.append([0]*i)
    return p

def score_max_dyn(i, j, p, s):
    if ___________________ != 0:        # Déjà mémorisé
        return ___________________
    elif i == ___________________:   # Base de la pyramide
        s[i][j] = ___________________
    else:                    # Autres étages
        s[i][j] = ___________________ + max(________________________
                                                    ,________________________)
    return ___________________

print(score_max_dyn(___, ___, p, ________________________))
```

<u>Question 3</u> : Complété la fonction `score_max_ite` qui résout le problème itérativement.

```
def score_max_ite(p):
    n = len(p)
    s = _______________________

    for j in range(n):              # remplissage de la base
        s[_______][j] = ___________________

    for i in range(n-2, -1, -1):    # remplissage des autres étages
        for j in range(len(s[i])):
            s[i][j] = __________________ + max(_____________________
                                                       ,_____________________)

    return ______________________    # renvoie du score maximal

print(score_max_ite(p))
```