EXERCICE 1 (4 points)

Écrire une fonction `couples_consecutifs` qui prend en paramètre une liste de nombres entiers `tab` non vide, et qui renvoie la liste (éventuellement vide) des couples d'entiers consécutifs successifs qu'il peut y avoir dans `tab`.

Ne pas oublier d'ajouter au corps de la fonction une documentation et une ou plusieurs assertions pour vérifier les pré-conditions.

Exemples :

```
>>> couples_consecutifs([1, 4, 3, 5])
[]
>>> couples_consecutifs([1, 4, 5, 3])
[(4, 5)]
>>> couples_consecutifs([1, 1, 2, 4])
[(1, 2)]
>>> couples_consecutifs([7, 1, 2, 5, 3, 4])
[(1, 2), (3, 4)]
>>> couples_consecutifs ([5, 1, 2, 3, 8, -5, -4, 7])
[(1, 2), (2, 3), (-5, -4)]
```

EXERCICE 2 (4 points)

Soit une image binaire représentée dans un tableau à 2 dimensions. Les éléments `M[i][j]`, appelés pixels, sont égaux soit à `0` soit à `1`.

Une composante d'une image est un sous-ensemble de l'image constitué de pixels de même valeur (soit `1`, soit `0`) qui sont côte à côte, soit horizontalement soit verticalement.

Par exemple, les composantes de

M =

0	0	1	0
0	1	0	1
1	1	1	0
0	1	1	0

sont

M =

0	0	1	0
0	1	0	1
1	1	1	0
0	1	1	0

On souhaite, à partir d'un pixel égal à 1 dans une image M, donner la valeur `val` à tous les pixels de la composante à laquelle appartient ce pixel.

La fonction `propager` prend pour paramètre une image `M` (représentée par une liste de listes), deux entiers `i` et `j` et une valeur entière `val`. Elle met à la valeur `val` tous les pixels de la composante du pixel `M[i][j]` s'il vaut `1` et ne fait rien s'il vaut `0`.

Par exemple, `propager(M,2,1,3)` donne

M =

0	0	1	0
0	3	0	1
3	**3**	3	0
0	3	3	0

Compléter le code récursif de la fonction `propager` donnée ci-dessous

```
def propager(M, i, j, val):
    if M[i][j] == ...:
        M[i][j] = val

    # l'élément en haut fait partie de la composante
    if i-1 >= 0 and M[i-1][j] == ...:
        propager(M, i-1, j, val)

    # l'élément en bas fait partie de la composante
    if ... < len(M) and M[i+1][j] == 1:
        propager(M, ..., j, val)

    # l'élément à gauche fait partie de la composante
    if ... and M[i][j-1] == 1:
        propager(M, ..., ..., val)

    # l'élément à droite fait partie de la composante
    if ... and ...:
        propager(..., ..., ..., ...)
```

Exemple :

```
>>> M = [[0, 0, 1, 0], [0, 1, 0, 1], [1, 1, 1, 0], [0, 1, 1, 0]]
>>> propager(M, 2, 1, 3)
>>> M
[[0, 0, 1, 0], [0, 3, 0, 1], [3, 3, 3, 0], [0, 3, 3, 0]]
```