

## Exercice 4 (4 points).

Cet exercice porte sur la notion de chaînes de caractères, de tableau et sur la programmation de base en Python.

On appelle palindrome un texte dont l'ordre des lettres reste le même, qu'on le lise de la droite vers la gauche ou de la gauche vers la droite.

Par exemple, un mot à une lettre est un palindrome, "BOB" est un palindrome, tout comme "LAVAL". Le mot "" est considéré comme un palindrome.

On souhaite programmer une fonction `palindrome` qui prend en paramètre une chaîne de caractères `txt`. Cette fonction renvoie `True` si la chaîne de caractères `txt` est un palindrome, `False` sinon.

On rappelle que :

- La fonction `len` prend une chaîne de caractères en paramètre et renvoie sa longueur, c'est-à-dire le nombre de lettres constituant la chaîne de caractères.

**Exemple :** `len("bonjour")` vaut 7.

- Si `txt` est une chaîne de caractères, `txt[i]` est la lettre de `txt` d'indice *i*. Attention, la première lettre a pour indice 0.

**Exemple :** si `txt = "bonjour"` alors `txt[2]` désigne "n".

1. On donne ci-dessous une implémentation récursive incomplète pour la fonction `palindrome`.

L'opération + à la ligne 8 permet de concaténer deux chaînes de caractères.

**Exemple :** Si `txt1 = "bon"` et `txt2 = "jour"`, l'instruction `txt1 + txt2` renvoie la chaîne de caractères "bonjour".

```
1 def palindrome(txt):
2     """ Str -> Bool """
3     -----
4     -----
5     taille = len(txt)
6     interieur = ""
7     for i in range(1, taille - 1):
8         interieur = interieur + txt[i]
9     return (txt[0] == txt[taille - 1]) and (palindrome(intérieur))
```

- (a) Choisir parmi les propositions ci-dessous celle qui convient pour compléter la fonction `palindrome` (ligne 3 et 4) :

**Proposition 1 :**

```
1 if len(txt) <= 2:
2     return True
```

**Proposition 3 :**

```
1 if len(txt) < 2:
2     return True
```

**Proposition 2 :**

```
1 if len(txt) < 2:
2     return False
```

**Proposition 4 :**

```
1 if len(txt) <= 2:
2     return False
```

- (b) Lors de l'appel de `palindrome("bonjour")`, indiquer les valeurs, à la ligne 9, de : `txt[0]`, `txt[taille - 1]` et `interieur`.
2. Proposer deux tests pour cette fonction qui permettent de tester deux cas de figure différents en justifiant ce choix.
  3. Écrire une version non récursive de la fonction `palindrome`.
  4. On étudie dans cet exercice des chaînes de caractères utilisant uniquement les lettres "A", "T", "C" et "G".

**Exemple :** "AA", "CAT" et "CCGATACG".

On associe à chacune de ces lettres une autre lettre appelée lettre complémentaire selon le tableau suivant :

Lettre	"A"	"T"	"G"	"C"
Lettre complémentaire	"T"	"A"	"C"	"G"

On obtient le complémentaire d'un mot en remplaçant chacune de ses lettres par sa lettre complémentaire.

**Exemple :** Le complémentaire à "GAATTC" est "CTTAAG".

- (a) Écrire une fonction en Python, nommée `complementaire`, qui prend en paramètre une chaîne de caractères `txt` écrite uniquement avec les lettres "A", "C", "G" et "T". Cette fonction renvoie la chaîne de caractères complémentaire de `txt`.

**Exemple :** l'appel `complementaire("GAATTC")` renvoie "CTTAAG".

- (b) Une chaîne de caractères `txt` est dite palindromique si la concaténation de `txt` avec son complémentaire est un palindrome.

**Exemples :**

"GAATTC" est palindromique car "GAATTC" + "CTTAAG" = "GAATTCCCTTAAG" est un palindrome.

"GAAT" n'est pas palindromique car "GAAT" + "CTTA" = "GAATCTTA" n'est pas un palindrome.

Déterminer si la chaîne de caractères "GATCGT" est palindromique.

- (c) Écrire une fonction `est_palindromique` prenant comme paramètre une chaîne de caractères `txt`. Cette fonction doit renvoyer `True` s'il s'agit d'une séquence palindromique, `False` sinon.

## Exercice 5 (4 points).

Cet exercice porte sur les tableaux à deux dimensions et la programmation python en général.

### Partie A

Dans cette partie, on s'intéresse à un dessin pixelisé en noir et blanc dans lequel chaque ligne est obtenue séquentiellement à partir de la ligne juste au-dessus d'elle.

Au départ, on dispose donc de la ligne du haut et on déduit les lignes en-dessous les unes après les autres.

Les règles sont les suivantes :

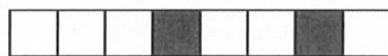
- pour chaque ligne, le pixel complètement à droite et le pixel complètement à gauche sont blancs.
- pour les autres pixels, un pixel est noir si les deux pixels à droite et gauche du pixel juste au-dessus de ce pixel ne sont pas de la même couleur, sinon il est blanc.

a		b
	x	

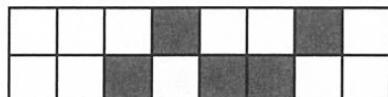
Le pixel noté "x" est donc blanc si les pixels notés "a" et "b" sont de la même couleur ou noir sinon.

1. (a) Dans cette question uniquement, on considère un dessin de cinq lignes et huit colonnes.

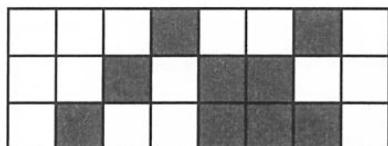
La première ligne est colorée comme ci-dessous :



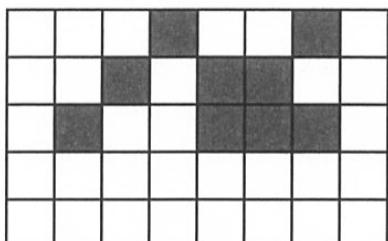
En respectant les règles de coloration, on complète la figure en ajoutant la deuxième ligne :



De même, en ajoutant la troisième ligne :



Recopier et colorier les deux dernières lignes du dessin ci-dessous en respectant les règles.



Dans la suite de cet exercice :

- les lignes sont numérotées du haut vers le bas, en commençant à 0 ;
- les colonnes sont numérotées de la gauche vers la droite, en commençant à 0.

(b) On considère le pixel de la ligne 4 et de la colonne 1.

- Indiquer sur quelle ligne et sur quelle colonne se trouve le pixel situé sur la ligne juste au-dessus, à sa gauche.
  - Même question, pour le pixel situé sur la ligne juste au-dessus, à sa droite.
- (c) Plus généralement, on considère le pixel de la ligne  $li$  et de la colonne  $co$  et situé ni sur la première ligne, ni sur la colonne tout à gauche, ni sur la colonne tout à droite.
- Indiquer sur quelle ligne et sur quelle colonne se trouve le pixel situé sur la ligne juste au-dessus, à sa gauche.
  - Même question, pour le pixel situé sur la ligne juste au dessus, à sa droite.

2. On dispose d'un tableau `image` à deux dimensions ( $m$  lignes et 8 colonnes) qui contient uniquement des 0 ou des 1. `image[li][co]` vaut 1 si le pixel de la ligne  $li$  et la colonne  $co$  est noir et 0 sinon.

Pour chaque valeur de  $li$ , `image[li][0] = 0` et `image[li][7] = 0`.

- (a) On suppose que le tableau `image` a été rempli en respectant les règles de coloration jusqu'à la ligne  $li-1$  et contient des 0 partout ailleurs. On suppose de plus que  $0 < co < 7$ .  
On cherche à connaître quelle valeur la case `image[li][co]` doit prendre si on respecte les règles de coloration. Donner les conditions portant sur les valeurs de la ligne  $li-1$  qui doivent être satisfaites pour que `image[li][co]` prenne la valeur 1.
- (b) On suppose toujours que le tableau `image` a été rempli en respectant les règles de coloration jusqu'à la ligne  $li-1$  et contient des 0 partout ailleurs.

Recopier et compléter la fonction Python `remplir_ligne` qui prend en paramètre le tableau `image` et un entier  $li$ . Cette fonction affecte à chaque case de la ligne  $li$  de `image`, la valeur correspondant à la couleur du pixel, en respectant les règles de coloration (dans le code de la fonction, les trois points (...) peuvent correspondre à une ou plusieurs lignes de programme).

```
1 def remplir_ligne(image, li) :  
2     image[li][0] = 0  
3     image[li][7] = 0  
4     for ... in range(..., ...) :  
5         ...
```

(c) La première ligne du tableau `image` est remplie correctement.

Écrire une fonction `remplir` qui prend en paramètre le tableau `image`. Cette fonction modifie les valeurs des cases du tableau `image` des lignes restantes.

## Partie B

À un tableau de taille 8 contenant des 0 ou des 1, on associe l'entier dont la représentation binaire est donnée par ce tableau.

Par exemple, le tableau [0, 0, 0, 1, 0, 0, 1, 0] représente le nombre binaire 00010010, correspondant à 18 en base 10.

- (a) Soit le tableau  $[0, 0, 1, 0, 1, 1, 0, 0]$ . Déterminer la représentation en base 10 de l'entier correspondant.
- (b) Écrire la fonction Python `conversion2_10` qui prend en paramètre un tableau `tab` de taille 8 composé de 0 et 1, et qui renvoie l'entier correspondant, écrit en base 10.

**Exemple :** `conversion2_10([0, 0, 0, 1, 0, 0, 1, 0])` vaut 18.

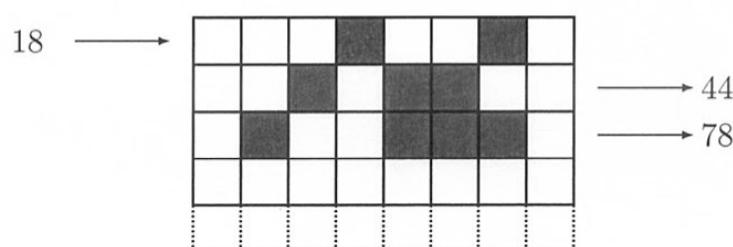
- (c) Réciproquement, donner le tableau associé à l'entier 78 dont l'écriture en base 10 est 78.

Dans la suite, on suppose qu'on dispose de la fonction Python `conversion10_2`, qui prend en paramètre un entier  $n$  compris entre 0 et 255, et qui renvoie le tableau `tab` correspondant.

**Exemple :** `conversion10_2(18)` vaut  $[0, 0, 0, 1, 0, 0, 1, 0]$ .

- À partir d'un entier  $n$ , on construit une liste de  $k$  entiers en utilisant l'image étudiée dans la partie A, de la façon suivante :

- l'entier  $n$  fournit la première ligne de l'image sous forme d'un tableau de taille 8, représentant son écriture binaire ;
- chaque ligne suivante de l'image est obtenue selon les règles appliquées dans la partie A et permet de déterminer un entier de la liste.



Par exemple, l'entier 18 fournit la liste 44, 78, ...

- On rappelle que pour chaque ligne de l'image, le pixel complètement à gauche et le pixel complètement à droite restent blancs. En déduire une ou des précondition(s) sur l'entier  $n$ .
- Recopier et compléter la fonction `generer` qui prend en paramètres deux entiers  $n$  et  $k$ , et renvoie le tableau `tab`, de taille  $k$ , contenant les  $k$  entiers obtenus à partir de  $n$  (dans le code de la fonction, les trois points (...) peuvent correspondre à une ou plusieurs lignes de programme).

```

1 def generer(n, k):
2     tab = [None for i in range(k)]
3     image = [[0 for j in range(8)] for i in range(k+1)]
4     ...
5     return tab

```

## Exercice 2 (4 points)

Cet exercice porte sur la programmation et les algorithmes de tri.

Au service des urgences d'un hôpital, le triage consiste à classifier ou à déterminer le degré de priorité des patients. Il implique une réévaluation périodique et systématique de ce degré pour les patients en attente.

Dans le système informatique, chaque patient obtient un identifiant à son arrivée en salle d'attente ainsi qu'une priorité dépendant de la gravité potentielle de ses symptômes. Le patient ayant la priorité 1 est le premier qui doit être pris en charge et deux patients ne peuvent pas avoir la même priorité.

Pour modéliser la salle d'attente en Python, chaque patient est représenté par un tuple composé de son identifiant d'arrivée et de sa priorité. Ainsi, la variable `attente` implantée ci-dessous représente une salle d'attente de trois patients où, à cet instant, le patient identifié 47 sera le premier à être pris en charge, puis le patient 45 et finalement le patient 49.

```
attente = [(45, 2), (47, 1), (49, 3)]
```

1. Écrire l'instruction qui permet d'insérer dans la liste `attente`, définie ci-dessus, un nouveau patient identifié 50 avec une priorité de 4.
2. Pour optimiser le traitement informatisé de prise en charge des patients, on veut que la salle d'attente soit ordonnée par priorité. On utilise alors la fonction `tri(attente)` donnée ci-dessous qui renvoie la salle d'attente triée dans l'ordre croissant des priorités.

```
def tri(attente) :  
    for i in range(len(attente)) :  
        pos = i  
        mini = attente[i][1]  
        for j in range(i, len(attente)) :  
            if attente[j][1] < mini :  
                pos = j  
                mini = attente[j][1]  
        temp = attente[i]  
        attente[i] = attente[pos]  
        attente[pos] = temp
```

- a. Quel algorithme de tri est ici implanté ?
  - ✓ le tri fusion
  - ✓ le tri par insertion
  - ✓ le tri par sélection
  - ✓ le tri rapide
- b. Quelle est la complexité en temps des tris par insertion et par sélection ?
  - ✓ constante :  $O(1)$
  - ✓ logarithmique :  $O(\log n)$
  - ✓ linéaire :  $O(n)$
  - ✓ quadratique :  $O(n^2)$

3. On considère dans cette question que la salle d'attente est triée par ordre croissant des priorités.
- a. Quand un patient est pris en charge, il faut l'enlever de la salle d'attente. La fonction `quitte(attente)` supprime le patient de priorité 1 de la liste `attente` passée en paramètre. Cette fonction renvoie la nouvelle salle d'attente. On aurait par exemple :

```
>>> quitte([(47, 1), (45, 2), (49, 3)])
[(45, 2), (49, 3)]
```

Recopier et compléter la ligne 14 en définissant une liste en compréhension.

```
13     def quitte(attente) :
14         return [.....]
```

- b. La fonction `quitte(attente)` ne met pas à jour les priorités des patients. Écrire une fonction `maj(attente)` qui met à jour, dans une nouvelle liste, les priorités des patients suite à la prise en charge d'un patient et renvoie cette nouvelle salle d'attente. On aurait par exemple :

```
>>> maj([45, 2), (49, 3])
[(45, 1), (49, 2)]
```

#### 4.

- a. Écrire la fonction `priorite(attente, p)` qui renvoie l'ordre de priorité d'un patient `p` de la liste `attente`. On aurait par exemple :

```
>>> priorite([(45, 2), (47, 1), (49, 3)], 49)
3
```

- b. La fonction `revise(attente, p)` renvoie une nouvelle salle d'attente non triée où le patient `p` dont la priorité médicale a évolué reçoit la priorité 1 et les priorités des autres patients sont mises à jour. On aurait par exemple :

```
>>> revise([(45, 2), (47, 1), (49, 3)], 49)
[(45, 3), (47, 2), (49, 1)]
```

Recopier et compléter les lignes 28, 29 du code ci-dessous.

```
23 def revise(attente, p) :
24     nouvelle = []
25     n = priorite(attente, p)
26     for (patient, prio) in attente :
27         if patient == p :
28             nouvelle.append( ..... )
29         elif ..... :
30             nouvelle.append((patient, prio + 1))
31         else :
32             nouvelle.append((patient, prio))
33     return nouvelle
```

## EXERCICE 1 (4 points)

Thèmes abordés : systèmes d'exploitation linux

L'entreprise capNSI gère les contrats de ses clients en créant pour chacun d'eux un sous-dossier dans le dossier Contrats sur leur ordinateur central. Le système d'exploitation de cet ordinateur est une distribution linux. Quelques commandes de bases pour ce système d'exploitation sont rappelées en annexe 1 en fin de sujet.

Dans la console représentée sur la figure ci-dessous, on peut visualiser les répertoires (ou dossiers) à la racine de l'ordinateur central avec l'instruction `ls` :

```
gestion@capNSI-ordinateur_central:~$ ls
Bureau      Documents      Modèles      Public
Téléchargements  Contrats      Images      Musique
Vidéos
```

1.

- Donner le nom de l'utilisateur et le nom de l'ordinateur correspondant à la capture d'écran précédente.
- Ecrire les instructions permettant d'afficher la liste des dossiers clients du répertoire `Contrats` en partant de la situation ci-dessous :

```
gestion@capNSI-ordinateur_central:~$
```

Après une campagne de démarchage, l'entreprise a gagné un nouveau client, Monsieur Alan Turing. Elle souhaite lui créer un sous-dossier nommé `TURING_Alan` dans le dossier `Contrats`. De plus, elle souhaite attribuer tous les droits à l'utilisateur et au groupe et seulement la permission en lecture pour tous les autres utilisateurs. La commande `chmod` permet de le faire.

2.

- Ecrire les instructions permettant de créer le sous-dossier `TURING_Alan` à partir du répertoire racine.
- Ecrire l'instruction permettant d'attribuer les bons droits au sous-dossier `TURING_Alan`.

En Python, le module `os` permet d'interagir avec le système d'exploitation. Il permet de gérer l'arborescence des fichiers, des dossiers, de fournir des informations sur le système d'exploitation. Par exemple, le code de la page suivante, exécuté dans la console, permet de créer le sous-dossier `TURING_Alan` précédent :

```
>>> import os
>>> os.mkdir("Contrats/TURING_Alan")
>>> os.chmod("Contrats/TURING_Alan", 774)
```

L'entreprise dispose d'un tableau de nouveaux clients :

```
tab_clients = [  
    ('LOVELACE', 'Ada'),  
    ('BOOLE', 'George'),  
    ('VONNEUMANN', 'John'),  
    ('SHANNON', 'Claude'),  
    ('KNUTH', 'Donald')  
]
```

Elle souhaite automatiser le formatage des tableaux des nouveaux clients. Elle souhaite également automatiser la création et l'attribution des droits des dossiers portant les noms des nouveaux clients.

3. Ecrire une fonction **formatage(tab)** qui prend en paramètre un tableau de tuples (**Nom**, **Prenom**) des nouveaux clients et renvoie un tableau de chaînes de caractères. Par exemple, **formatage(tab\_clients)** renvoie  
['LOVELACE\_Ada', 'BOOLE\_George', 'VONNEUMANN\_John',  
'SHANNON\_Claude', 'KNUTH\_Donald']
4. Ecrire une fonction **creation\_dossiers(tab)** qui prend en paramètre un tableau de chaînes de caractères et qui crée et modifie les droits des dossiers au nom de ces chaînes de caractères avec les mêmes droits que le sous-dossier **TURINGAlan**.

## Annexe 1 (exercice 1)

(à ne pas rendre avec la copie)

*Extrait des commandes de base linux*

ls      permet d'afficher le contenu d'un répertoire  
cd      se déplacer dans l'arborescence (ex cd repertoire1)  
cp      créer une copie d'un fichier (ex cp fichier1.py fichier2.py)  
mv      déplacer ou renommer un fichier ou un répertoire (ex : mv fichier.txt dossier)  
rm      effacer un fichier ou un répertoire (ex rm mon\_fichier.mp3)  
mkdir    créer un répertoire (ex mkdir nouveau)  
cat      visualiser le contenu d'un fichier  
chmod    modifier les permissions d'un fichier ou d'un dossier. Pour un fichier, le format général de l'instruction est :

chmod droits\_user droits\_group droits\_other nom\_fichier

Où droits\_user, droits\_group et droits\_other indiquent respectivement les droits de l'utilisateur, du groupe et des autres et peuvent être :

- + ajouter
- supprimer
- r read
- w write
- x execute

**Exemple :** chmod rwx +r -x script.sh

### EXERCICE 3 (4 points)

Thèmes abordés : structures de données, programmation.

Le « jeu de la vie » se déroule sur une grille à deux dimensions dont les cases, qu'on appelle des « cellules », par analogie avec les cellules vivantes, peuvent prendre deux états distincts : « vivante » (= 1) ou « morte » (= 0).

Une cellule possède au plus huit voisins, qui sont les cellules adjacentes horizontalement, verticalement et diagonalement.

À chaque étape, l'évolution d'une cellule est entièrement déterminée par l'état de ses huit voisines de la façon suivante :

- Règle 1 : une cellule morte possédant exactement trois voisines vivantes devient vivante (elle naît) ; sinon, elle reste à l'état « morte »
- Règle 2 : une cellule vivante possédant deux ou trois voisines vivantes reste vivante, sinon elle meurt.

Voici un exemple d'évolution du jeu de la vie appliquée à la cellule centrale :

<table border="1"><tr><td>1</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td></tr></table>	1	1	0	0	0	0	0	0	1	devient par la règle 1	<table border="1"><tr><td></td><td></td><td></td></tr><tr><td></td><td>1</td><td></td></tr><tr><td></td><td></td><td></td></tr></table>					1					<table border="1"><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td></tr></table>	1	0	0	0	1	1	1	0	0	reste par la règle 2	<table border="1"><tr><td></td><td></td><td></td></tr><tr><td></td><td>1</td><td></td></tr><tr><td></td><td></td><td></td></tr></table>					1				
1	1	0																																							
0	0	0																																							
0	0	1																																							
	1																																								
1	0	0																																							
0	1	1																																							
1	0	0																																							
	1																																								
<table border="1"><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	1	1	0	0	0	0	devient par la règle 2	<table border="1"><tr><td></td><td></td><td></td></tr><tr><td></td><td>0</td><td></td></tr><tr><td></td><td></td><td></td></tr></table>					0					<table border="1"><tr><td>1</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	1	1	0	0	1	1	1	1	0	devient par la règle 2	<table border="1"><tr><td></td><td></td><td></td></tr><tr><td></td><td>0</td><td></td></tr><tr><td></td><td></td><td></td></tr></table>					0				
0	0	0																																							
1	1	0																																							
0	0	0																																							
	0																																								
1	1	0																																							
0	1	1																																							
1	1	0																																							
	0																																								

Pour initialiser le jeu, on crée en langage Python une grille de dimension 8x8, modélisée par une liste de listes.

**1. Initialisation du tableau :**

- a.** Parmi les deux scripts proposés, indiquer celui qui vous semble le plus adapté pour initialiser un tableau de 0. Justifier votre choix

Choix 1	Choix 2
1   ligne = [0,0,0,0,0,0,0,0] 2   jeu = [] 3   for i in range(8) : 4       jeu.append(ligne)	1   jeu = [] 2   for i in range(8) : 3       ligne = [0,0,0,0,0,0,0,0] 4       jeu.append(ligne)

- b.** Donner l'instruction permettant de modifier la grille jeu afin d'obtenir

```
>>> jeu
[[0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0]]
```

**2.**

- a.** Ecrire en langage Python une fonction `remplissage(n, jeu)` qui prend en paramètres un entier `n` et une grille `jeu`, et qui ajoute aléatoirement exactement n cellules vivantes dans le tableau `jeu`.

- b.** Quelles sont les préconditions de cette fonction pour la variable `n` ?

On propose la fonction en langage Python `nombre_de_vivants(i, j, jeu)` qui prend en paramètres deux entiers `i` et `j` ainsi qu'une grille `jeu` et qui renvoie le nombre de voisins **vivants** de la cellule `tab[i][j]` :

```
1 | def nombre_de_vivants(i, j, jeu) :
2 |     nb = 0
3 |     voisins = [(i-1,j-1), (i-1,j), (i-1,j+1), (i,j+1),
4 |                 (i+1,j+1), (i+1,j), (i+1,j-1), (i,j-1)]
5 |     for e in voisins :
6 |         if 0 <= ... < 8 and 0 <= ... < 8 :
7 |             nb = nb + jeu[...][...]
8 |     return nb
```

**3. Recopier et compléter les pointillés pour que la fonction réponde à la demande.**

**4. En utilisant la fonction `nombre_de_vivants(i, j, jeu)` précédente, écrire en langage Python une fonction `transfo_cellule(i, j, jeu)` qui prend en paramètres deux entiers `i` et `j` ainsi qu'une grille `jeu` et renvoie le nouvel état de la cellule `jeu[i][j]` (0 ou 1)**