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.

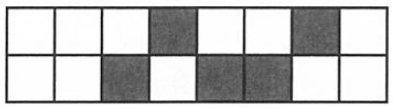
— pour les autres pixels, un pixel est noir si les deux pixels à droite et gauche du pixel juste au-dessus de ce pixel ne sont pas de la même couleur, sinon il est blanc.

a		b
	x	

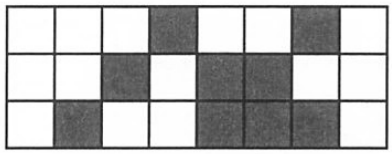
Le pixel noté "x" est donc blanc si les pixels notés "a" et "b" sont de la même couleur ou noir sinon.

1. (a) Dans cette question uniquement, on considère un dessin de cinq lignes et huit colonnes.
La première ligne est colorée comme ci-dessous :

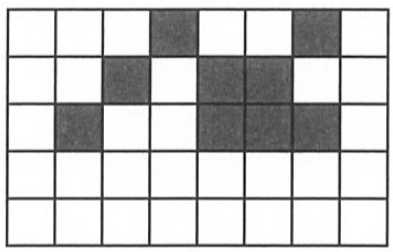
En respectant les règles de coloration, on complète la figure en ajoutant la deuxième ligne :



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.

1. (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.

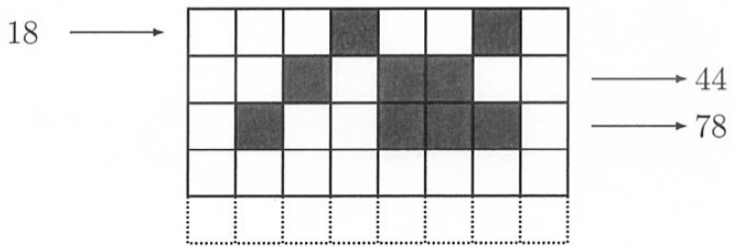
Dans la suite, on suppose qu'on dispose de la fonction Python `conversion10_2`, qui prend en paramètre un entier n compris entre 0 et 255, et qui renvoie le tableau `tab` correspondant.

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

2. 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, ...

   (a) 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.

   (b) 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
```