Exercice 3 (8 points)

Cet exercice porte sur l'algorithmique, la représentation binaire des entiers positifs et la programmation en langage Python.

Partie A : Modélisation du problème

On s'intéresse à un jeu de calcul mental appelé **Objectif somme**. Le jeu se joue sur un plateau de 5x5 cases. Chaque case contient un chiffre non nul de 1 à 9. On dispose également de nombres cibles en ligne et en colonne. Le but est de trouver les cases du tableau à vider afin d'atteindre les cibles en ligne et en colonne :

- sur chaque ligne, la somme des cases restantes doit valoir la cible de cette ligne ;
- sur chaque colonne, la somme des cases restantes doit valoir la cible de cette colonne.

De plus, il faut conserver au moins un chiffre par ligne.

Par exemple, la figure suivante représente un plateau de jeu et une solution :

	15	13	5	2	9	
13	7	9	2	3	2	L0
9	8	6	3	5	1	L1
12	7	7	3	2	7	L2
6	6	4	5	8	2	L3
4	8	6	8	8	4	L4
	C0	C1	C2	C3	C4	

	15	13	5	2	9	
13		9	2		2	L0
9	8				1	L1
12	7		3	2		L2
6		4			2	L3
4					4	L4
	C0	C1	C2	C3	C4	

Figure 1. Plateau de jeu (à gauche) et une solution (à droite).

Les lignes du plateau seront nommées de *L0* à *L4* et les colonnes de *C0* à *C4*.

Ainsi l'exemple de la figure 1, la ligne *L1* fait référence aux valeurs 8,6,3,5,1 du plateau et la colonne *C3* fait référence aux valeurs 3,5,2,8,8 du plateau.

Dans la suite, on suppose que les cibles sont nécessairement des entiers entre 1 et 45.

1. Expliquer pourquoi on fait cette hypothèse.
2. Donner la plus petite valeur de cible que la ligne `[6,4,5,8,2]` peut atteindre. Donner aussi la plus grande valeur de cible que la ligne peut atteindre.

Dans la suite on appelle *plateau* une liste de 5 listes de 5 entiers. Chacune des listes de 5 entiers représente une ligne. Les entiers de ces listes sont compris entre 0 et 9, 0 représente une case vide. Pour représenter un jeu, un plateau doit être accompagné de deux listes de 5 entiers : la liste des cibles de lignes, la liste des cibles des colonnes.

Voici une représentation en langage Python de la figure 1 :

```
plateau_ex = [[7, 9, 2, 3, 2],
              [8, 6, 3, 5, 1],
              [7, 7, 3, 2, 7],
              [6, 4, 5, 8, 2],
              [8, 6, 8, 8, 4]]

ciblesLignes_ex = [13, 9, 12, 6, 4]

ciblesColonnes_ex = [15, 13, 5, 2, 9]
```

3. Écrire une fonction `extraireLigne` qui prend en paramètre un plateau et un indice *i* (*i* compris entre 0 et 4 inclus) et renvoie la ligne *Li* du plateau. Par exemple, la valeur de retour de l'appel `extraireLigne(plateau, 0)` est `[7, 9, 2, 3, 2]`.

4. Écrire une fonction `extraireColonne` qui prend en paramètre un plateau et un indice *i* (compris entre 0 et 4 inclus) et renvoie la colonne *Ci* du plateau. Par exemple, la valeur de retour de l'appel `extraireColonne(plateau, 1)` est `[9, 6, 7, 4, 6]`.

Partie B : Simplification du problème

Dans la figure 1, la solution comporte des cases vides. Ces cases correspondent aux chiffres que l'on a éliminés. En langage Python, on représentera ces cases vides par des zéros. Ainsi, pour éliminer du plateau un chiffre, il suffira de le remplacer par 0.

5. Donner la représentation en langage Python du plateau de la solution proposée.

On se propose d'utiliser deux règles pour éliminer simplement certains chiffres du plateau.

Règle 1 : on remarque que les chiffres d'une ligne ou d'une colonne donnée du plateau doivent être inférieurs ou égaux à la cible. Par exemple, pour la ligne *L4* de la figure 1, la cible est 4, on peut alors éliminer tous les chiffres 8 et 6. En appliquant la règle 1, *L4* devient alors `[0, 0, 0, 0, 4]`.

6. Pour le jeu représenté à gauche sur la figure 1, donner en Python le plateau obtenu en appliquant la règle 1 à chaque ligne.

La fonction à compléter `regle1` ci-dessous est une implémentation de la règle **1**. Elle prend en paramètre `plateau`, `ciblesLignes` et `ciblesColonnes` décrivant un jeu comme expliqué plus haut, et elle modifie `plateau` en appliquant la règle 1 à chaque ligne et à chaque colonne.

```
 1 def regle1(plateau, ciblesLignes, ciblesColonnes):
 2     for i in range(5):
 3         tab = extraireLigne(plateau, i)
 4         cible = ...
 5         for j in range(5):
 6             if tab[j] > cible:
 7                 plateau[i][j] = 0
 8     for j in range(5):
 9         tab = extraireColonne(plateau, j)
10         cible = ...
11         for i in range(5):
12             if tab[i] > cible:
13                 plateau[i][j] = 0
```

7. Recopier et compléter les lignes 4 et 10 pour compléter le code de la fonction `regle1`.

Règle 2 : S'il n'y a qu'un seul nombre impair dans une ligne ou une colonne dont la cible est paire, on peut éliminer ce nombre impair. Par exemple, pour la ligne *L3* de la figure 1, la cible est 6 et il n'y a qu'un nombre impair : 5. On peut donc éliminer ce 5.

8. Écrire une fonction `unImpair` qui prend comme paramètre une liste d'entiers, et qui renvoie `True` si la liste ne contient qu'un seul entier impair et `False` sinon.

La fonction à compléter `regle2` ci-dessous est une implémentation de la règle **2**. Elle prend en paramètre `plateau`, `ciblesLignes` et `ciblesColonnes` décrivant un jeu comme expliqué plus haut, et elle modifie `plateau` en appliquant la règle 2 à chaque ligne et à chaque colonne.

9. Recopier et compléter les lignes 4, 5, 11 et 12 pour compléter le code de la fonction **regle2** ci-dessous qui prend comme paramètre un `plateau`, une `ciblesLignes` et une `ciblesColonnes` et qui applique la règle **2**.

```
 1 def regle2(plateau, ciblesLignes, ciblesColonnes):
 2     for i in range(5):
 3         ligne = extraireLigne(plateau, i)
 4         ...
 5         if ...:
 6             for j in range(5):
 7                 if plateau[i][j] % 2 == 1:
 8                     plateau[i][j] = 0
 9     for j in range(5):
10         colonne = extraireColonne(plateau, j)
11         ...
12         if ...:
13             for i in range(5):
```

```
14                  if plateau[i][j] % 2 == 1:
15                      plateau[i][j] = 0
```

Ces règles permettent de simplifier le jeu mais pas de le résoudre dans tous les cas. Il est nécessaire d'utiliser d'autres méthodes.

Partie C : Problème sur une ligne et représentation binaire

Pour aider à la résolution du jeu **Objectif somme**, on cherche dans cette partie à résoudre le problème sur une ligne seulement. Il s'agit de trouver les nombres d'une liste de 5 entiers dont la somme est égale à un nombre cible.

Par exemple, une solution pour la liste `[6, 4, 5, 8, 2]` avec la cible 6 est de conserver le chiffre 6 uniquement. Une autre solution est de conserver le 4 et le 2.

On représente la première solution (conserver le 6) par la liste `[1, 0, 0, 0, 0]`. Cela signifie que la solution choisie est uniquement le premier élément de la liste. La liste `[1, 0, 0, 0, 0]` est appelée un **masque solution** du problème. Le masque solution correspondant à la solution avec le 4 et le 2, est alors `[0, 1, 0, 0, 1]`.

10. Expliquer pourquoi `[1, 1, 0, 0, 1]` est un masque solution pour la liste `[1, 2, 3, 5, 2]` et la cible 5. Donner tous les autres masques solutions.

11. Écrire une fonction `somme` qui prend comme paramètres une liste de 5 entiers et un masque (une liste de taille 5 de `0` et de `1`) et qui renvoie la somme des chiffres du tableau correspondant au masque. Par exemple, `somme([1, 5, 3, 4, 8], [0, 1, 1, 0, 1])` doit renvoyer 5 + 3 + 8 = 16.

On peut remarquer que les masques solutions correspondent à des nombres en écriture binaire. Par exemple, le masque `[0, 1, 0, 0, 1]` correspond à l'entier 9 car $0 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = 9$. Ainsi, on représente les masques possibles par des nombres en écriture binaire sur 5 bits.

12. Donner la représentation binaire sur 5 bits de l'entier 26 sous la forme d'une liste de taille 5.

13. Expliquer pourquoi on ne représente que les entiers compris entre 0 et 31 sur 5 bits.

14. Écrire une fonction `dec2bin` qui prend comme paramètre un entier compris entre 0 et 31 et qui renvoie sa représentation binaire sous la forme d'une liste de 5 bits. Par exemple la valeur de retour de l'appel `dec2bin(9)` est `[0, 1, 0, 0, 1]`.

Pour résoudre le problème, on se propose de générer tous les masques possibles avec la fonction `dec2bin` et de tester si ce sont des masques solutions. On stockera alors tous ces masques solutions dans une liste. On pourra utiliser la méthode `append` appliquée à une liste. Cette méthode permet d'ajouter un élément en fin de liste. Par exemple, à l'issue du code suivant, la liste `solutions` est `[1, 2]` :

```
solutions = [] # liste vide
solutions.append(1)
solutions.append(2)
```

15. Écrire une fonction `masques_solutions` qui prend comme paramètres une liste de taille 5 entiers et une cible, et qui renvoie la liste de tous les masques solutions correspondant.

Partie D : Retour au jeu “Objectif Somme”

Finalement, on vérifie qu’un plateau proposé comme solution respecte bien les contraintes sur les lignes et les colonnes.

16. Écrire une fonction `teste_solution` qui prend comme paramètres un plateau, la liste des cibles des lignes, la liste des cibles des colonnes, et qui retourne `True` si les valeurs des cases restantes du plateau vérifient bien les cibles (sur chaque ligne et sur chaque colonne), et `False` sinon.