

Nouvelle Calédonie – 2025 – sujet1 - Correction

Exercice 3 (8 points)

Partie A – Modélisation du problème

1. Cibles entre 1 et 45

Il faut garder au moins un chiffre par ligne, et chaque chiffre vaut au moins 1 : une cible vaut donc **au moins 1**.

Une ligne contient au plus 5 chiffres, chacun au plus égal à 9 : une cible vaut donc **au plus** $5 \times 9 = 45$.

2. Cibles possibles pour la ligne [6, 4, 5, 8, 2]

La plus petite cible est obtenue en ne gardant que le plus petit chiffre : **2**.

La plus grande est obtenue en gardant tous les chiffres : $6 + 4 + 5 + 8 + 2 = \mathbf{25}$.

3. Fonction extraireLigne

```
def extraireLigne(plateau, i):  
    return plateau[i]
```

4. Fonction extraireColonne

```
def extraireColonne(plateau, i):  
    colonne = []  
    for ligne in plateau:  
        colonne.append(ligne[i])  
    return colonne
```

👉 *Commentaire : version en compréhension : `return [ligne[i] for ligne in plateau]`.*

Partie B – Simplification du problème

5. Plateau de la solution

```
plateau_solution = [[0, 9, 2, 0, 2],  
                    [8, 0, 0, 0, 1],  
                    [7, 0, 3, 2, 0],  
                    [0, 4, 0, 0, 2],  
                    [0, 0, 0, 0, 4]]
```

👉 *Commentaire : on recopie la solution de droite de la figure 1 en remplaçant les cases vides par 0. On vérifie : les sommes des lignes valent 13, 9, 12, 6, 4 et celles des colonnes 15, 13, 5, 2, 9.*

6. Règle 1 appliquée à chaque ligne

Seules les lignes L3 (cible 6 : on élimine le 8) et L4 (cible 4 : on élimine les 8 et les 6) sont modifiées :

```
[[7, 9, 2, 3, 2],
 [8, 6, 3, 5, 1],
 [7, 7, 3, 2, 7],
 [6, 4, 5, 0, 2],
 [0, 0, 0, 0, 4]]
```

👉 *Commentaire : dans L0 (cible 13), L1 (cible 9) et L2 (cible 12), tous les chiffres sont inférieurs ou égaux à la cible.*

7. Fonction regle1 (lignes 4 et 10)

```
def regle1(plateau, ciblesLignes, ciblesColonnes):
    for i in range(5):
        tab = extraireLigne(plateau, i)
        cible = ciblesLignes[i]                # ligne 4
        for j in range(5):
            if tab[j] > cible:
                plateau[i][j] = 0
    for j in range(5):
        tab = extraireColonne(plateau, j)
        cible = ciblesColonnes[j]              # ligne 10
        for i in range(5):
            if tab[i] > cible:
                plateau[i][j] = 0
```

8. Fonction unImpair

```
def unImpair(liste):
    compteur = 0
    for x in liste:
        if x % 2 == 1:
            compteur = compteur + 1
    return compteur == 1
```

👉 *Commentaire : les cases vides valent 0, qui est pair : elles ne sont pas comptées.*

9. Fonction regle2 (lignes 4, 5, 11 et 12)

```
def regle2(plateau, ciblesLignes, ciblesColonnes):
    for i in range(5):
        ligne = extraireLigne(plateau, i)
```

```

        cible = ciblesLignes[i]                                # ligne 4
    if cible % 2 == 0 and unImpair(ligne):                      # ligne 5
        for j in range(5):
            if plateau[i][j] % 2 == 1:
                plateau[i][j] = 0
for j in range(5):
    colonne = extraireColonne(plateau, j)
    cible = ciblesColonnes[j]                                  # ligne 11
    if cible % 2 == 0 and unImpair(colonne):                    # ligne 12
        for i in range(5):
            if plateau[i][j] % 2 == 1:
                plateau[i][j] = 0

```

👉 *Commentaire : la capture de l'énoncé s'arrête à la ligne 13 : les lignes 14 et 15 (test de parité et mise à 0) sont les mêmes que pour les lignes.*

Partie C - Problème sur une ligne et représentation binaire

10. Masques solutions pour [1, 2, 3, 5, 2] et la cible 5

Le masque [1, 1, 0, 0, 1] garde le 1er, le 2e et le 5e élément : $1 + 2 + 2 = 5$, c'est bien la cible.

Les autres masques solutions sont :

masque	nombres gardés	somme
[0, 1, 1, 0, 0]	$2 + 3$	5
[0, 0, 1, 0, 1]	$3 + 2$	5
[0, 0, 0, 1, 0]	5	5

👉 *Commentaire : il ne faut pas oublier [0, 0, 1, 0, 1] : le 3 peut s'associer avec l'un ou l'autre des deux 2.*

11. Fonction somme

```

def somme(liste, masque):
    total = 0
    for i in range(5):
        if masque[i] == 1:
            total = total + liste[i]
    return total

```

👉 *Commentaire : variante sans test : $total = total + masque[i] * liste[i]$.*

12. Écriture binaire de 26

$26 = 16 + 8 + 2 = 1 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1$: la représentation est **[1, 1, 0, 1, 0]**.

13. Entiers de 0 à 31

Avec 5 bits, on a $2^5 = 32$ combinaisons : le plus petit entier est $00000 = 0$ et le plus grand $11111 = 16 + 8 + 4 + 2 + 1 = \mathbf{31}$.

👉 *Commentaire : c'est aussi le nombre de masques possibles : chacun des 5 chiffres est gardé ou non.*

14. Fonction dec2bin

```
def dec2bin(n):
    bits = []
    for i in range(5):
        bits.insert(0, n % 2)
        n = n // 2
    return bits
```

👉 *Commentaire : on obtient les bits du moins significatif au plus significatif : on les insère donc en tête de liste. Autre version : $[(n // 2^{**k}) \% 2 \text{ for } k \text{ in range}(4, -1, -1)]$.*

15. Fonction masques_solutions

```
def masques_solutions(liste, cible):
    solutions = []
    for n in range(32):
        masque = dec2bin(n)
        if somme(liste, masque) == cible:
            solutions.append(masque)
    return solutions
```

👉 *Commentaire : masques_solutions([1, 2, 3, 5, 2], 5) renvoie bien les 4 masques de la question 10.*

Partie D – Retour au jeu « Objectif somme »

16. Fonction teste_solution

```
def teste_solution(plateau, ciblesLignes, ciblesColonnes):
    for i in range(5):
        if somme(extraireLigne(plateau, i), [1, 1, 1, 1, 1]) != ciblesLignes[i]:
            return False
    for j in range(5):
        if somme(extraireColonne(plateau, j), [1, 1, 1, 1, 1]) != ciblesColonnes[j]:
```

```
        return False  
    return True
```

👉 *Commentaire : on peut aussi utiliser `sum(...)`. Comme les cibles valent au moins 1, une ligne dont la somme vaut sa cible contient forcément au moins un chiffre : la contrainte « au moins un chiffre par ligne » est vérifiée automatiquement.*