Tours de Hanoï

Répondre aux trois premières questions posées dans l'énoncé proposé en cours (le cas de base est $n = 1$).

4. Calculer le nombre d'étapes nécessaires pour terminer le jeu avec n disques.

5. Initialement, la tour est : $[[2,1],[\,],[\,]]$. Donner les états des variables et les affichages lorsque l'on exécute `Hanoi(2,tour,0,1,2)`

Sudoku

On cherche à compléter une grille de Sudoku : c'est une grille de taille 9×9 que l'on doit compléter avec les chiffres de 1 à 9 de sorte que :

— chaque ligne contienne une fois et une seule chaque chiffre de 1 à 9.
— chaque colonne contienne une fois et une seule chaque chiffre de 1 à 9.
— la grille est divisée en 9 blocs de taille 3×3 qui contiennent une fois et une seule les chiffres de 1 à 9.

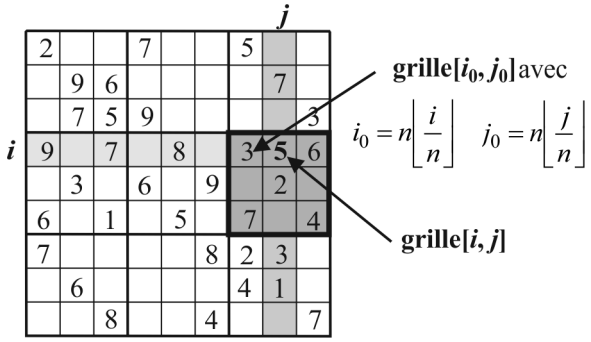
On représentera la grille par un tableau numpy de taille 9×9, les cases vides étant remplies avec le chiffre 0. Par exemple :

```
import numpy as np
GrilleTest = np.array([[6,4,5,2,3,0,0,0,0],[0,0,7,0,0,0,0,5,4],
[8,1,9,4,5,0,2,3,0],[0,0,0,9,2,0,1,6,0],[0,0,0,0,6,0,0,0,0],
[0,0,6,7,8,1,0,2,9],[0,5,8,3,4,2,6,0,7],[2,7,0,0,0,0,8,4,0],[0,0,0,0,7,0,0,9,2]])
```

0. Résoudre à la main cette grille de Sudoku.

Pour pouvoir travailler avec des tableaux éventuellement plus petits, on envisagera par la suite des tableaux de taille $n^2 \times n^2$ (pour un Sudoku normal : $n = 3$, et pour un Sudoku "réduit" : $n = 2$). On cherche donc à remplir chaque ligne, chaque colonne et chaque bloc (de taille $n \times n$) avec les chiffres compris entre 1 et n^2.

Pour les trois premières fonctions demandées, on suppose que la grille de Sudoku est valide AVANT de placer un chiffre dans la ligne i et la colonne j. Le but de ces fonctions est donc de vérifier que la grille est toujours valide APRÈS avoir rempli la case (i, j).
On utilisera des boucles `while test` où le booléen `test`, initialement vrai, devient faux dès qu'une incompatibilité est détectée avec le chiffre placé en (i, j).



1. Écrire une fonction `valideL(grille,i,j)`, qui vérifie que la ligne `i` ne contient pas deux fois le chiffre placé dans la case `(i,j)`. La fonction renvoie `True` si la $i^{\text{ème}}$ ligne reste conforme, `False` sinon.

2. Écrire de même une fonction `valideC(grille,i,j)` qui effectue la même vérification sur la colonne `j`.

3. Écrire une fonction `valideB(grille,i,j)` permettant de tester si le bloc de taille $n \times n$ contenant la case `(i,j)` reste lui aussi conforme. On pourra balayer ce bloc en remarquant que ses cases ont pour coordonnées (i_0+k, j_0+m) avec $i_0 = n\left\lfloor \frac{i}{n} \right\rfloor$ et $j_0 = n\left\lfloor \frac{j}{n} \right\rfloor$, et $(k, m) \in [\![0, n-1]\!]^2$.

4. Écrire une fonction `valide(grille,i,j)` qui renvoie `True` si la ligne `i`, la colonne `j`, et le bloc contenant la case `(i,j)` sont toujours conformes aux règles du Sudoku après remplissage de la case case `(i,j)`, et `False` sinon.

5. Écrire une fonction `libre(grille)` qui prend en argument une `grille` et qui renvoie un couple d'indices (i, j) correspondant à une case vide de la grille s'il en existe. Dans le cas contraire, la fonction renvoie la chaîne de caractères `'rempli'`.