Cette partie d'examen permettra de valider les 2 premières compétences et intervenir sur la 3ème :

- de mettre en œuvre une stratégie cohérente de résolution du problème posé ;
- de concevoir, de construire et de représenter l' (les) algorithme(s) correspondant(s) ;
- de justifier la démarche algorithmique et les choix mis en œuvre ; (en partie)

Les algorithmes demandés devront être exprimés en pseudo code en respectant le canevas donné.

1°) assignation : a ← 4
2°) échange : a ⇆ b
3°) division réelle : a / b
entière : a // b
4°) modulo : a modulo b ou a %b
5°) test égalité : a = b
6°) test si :
```
si a<b alors
    i ← i+1
sinon
    j ← j+1
fsi
```
7°) tantque :
```
tq a<b faire
    ⋮
ftq
```
8°) repeter :
```
repeter
    ⋮
jusqu'a ce que a>=b
```
9°) pour :
```
pour i:0 -> m faire
    ⋮
fpour
pour i:m -> 0 par pas de -1 faire
    ⋮
fpour
```
10°) indice matrice : m.i.j
11°) taille matrice : m.taille ou Taille de m
12°) Fonction :
```
fonction f(a,b)
    ⋮
retour a+b
fonction
```
13°) Procedure :
```
procedure p1(a,b)
    ⋮
fprocedure
```
14°) Nouveau vecteur : V ← nouveau vecteur taille n
Nouvelle Matrice : M ← nouvelle Matrice de taille n sur m

1 Enoncé 1

Créez une fonction qui retourne un vecteur qui contiendra la somme de v1 additionnée au vecteur v2 décalé de d positions vers la droite.

Exemple :

v1	1	8	5	6	8

v2	4	2	6	7

d = 2

La fonction devra retourner un nouveau vecteur v3

v1	1	8	5	6	8	
v2			4	2	6	7
v3	1	8	9	8	14	7

Fonction : **int[] sommeVD(int[] v1, int[] v2, int d)**

Hypothèse de départ : 0<= d <= taille de v1

Variables d'entrée de la fonction

- v1,v2 sont des vecteurs d'entiers
- d : désigne un décalage positif tel quel 0<= d <= taille de v1

Variable de sortie de votre fonction :

- v3 vecteur d'entier

Variables locales (nom, type et description)

Pseudo-code de votre algorithme :

2 Réalisez une fonction « int cmpBit0Bit1(int n) »

Cette fonction va devoir renvoyer :

- 0 : si le nombre de bits à 0 est égal au nombre de bits à 1
- -1 : si le nombre de bits à 0 est supérieur au nombre de bits à 1
- +1 : si le nombre de bits à 1 est supérieur au nombre de bits à 0

Hypothèse de départ :

« n » représente un nombre entier.

On peut obtenir son nombre de bits en faisant : cpt ← nombre de bits de n

Variable d'entrée de la fonction :

- « n » : un entier

Variable de sortie de votre fonction :

- « res » un entier qui contiendra -1 , 0 ou 1

Variables locales (nom, type et description)

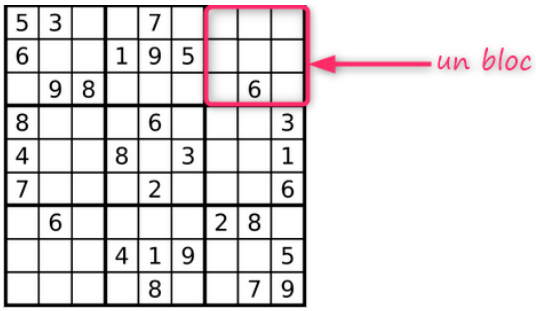
Pseudo-code de votre algorithme :

3 Créez une fonction qui valide un coup de Sudoku

Le jeu de Sudoku consiste à remplir une grille de 9x9 cases avec une série de chiffres (1 à 9) tous différents, qui ne se trouvent jamais plus d'une fois sur une même ligne, dans une même colonne ou dans un même bloc (on distincte 9 bocs de 3x3 dans la grille).

On va considérer ici une matrice de 9x9 d'entiers où le zéro représente une case vide.

On vous demande d'écrire une fonction qui reçoit en paramètre la grille 9x9, le chiffre (1 à 9) à insérer et sa position (x, y) d'insertion.



Votre fonction devra :

- Si le coup est valable : (case vide et pas de conflit)
  - renvoyer « true »
  - mettre à jour la grille en rajoutant le chiffre
- Si le coup n'est pas valable : (case occupée ou au moins un conflit)
  - renvoyer « false »

Fonction : **boolean ajoutValeur(int[][] jeu, int chiffre, int x, int y)**

Hypothèses de départ :

« jeu » : La matrice sera une matrice d'entiers de 9 sur 9 comme ci-dessus avec des 0 dans les cases vides. On suppose que cette matrice est correcte càd qu'elle ne possède pas de conflit.

« chiffre » : un entier entre 1 et 9

« x », « y » : une position x et y où 0 <= x <9 et 0 <= y <9

Variables d'entrée de la fonction (nom, type et description)

Jeu matrice d'entiers 9 sur 9 comme indiqué ci-dessus

chiffre la valeur que l'on veut encoder

x et **y** la position de la case. Sachant que (0,0) désigne la case supérieure gauche et (8,8) la case inférieure droite

Variable de sortie de votre fonction : (nom, type et description)

Un booléen qui donnera vrai si le coup est valable, faux sinon. La matrice sera ajustée si le coup est valable.

Variables locales utilisées et description :

Pseudo-code de votre algorithme :