Partie 1

Partie 1 : Tableaux 2D - Jeux de reflexion

L'interface des tableaux à 2D présente les fonctions :

- `taille`, paramètres : T : tableau, retourne la taille du tableau sous forme de tuple : (3,3) par exemple
- `element` ; paramètres : T : tableau, i et j deux entiers : retourne l'élément à la place i,j
- `remplacer` : paramètres : T : tableau, car : le caractère de remplacement, i et j deux entiers

1.1 Exercice 1.1 : Grille de Morpion

Exemple 1 : Parcours par indice (tableau 2D)

```
# Afficher la grille avec les indices
n = taille(morpion)[0]
for i in range(n):
    for j in range(n):
        print(element(morpion,i,j), end=' ')
    print()
# Affiche toutes les valeurs ligne par ligne:
X | O | X |
O | X |   |
  |   | O |
```

Exemple 2 : Accéder à un élément précis

```
# Afficher et modifier un élément
print("Case [0][1] :", element(morpion,0,1))  # Affiche 'O'
remplacer(morpion,'X',1,2) # Modifier la case
```

Exemple 3 : Compte le nombre de X dans une ligne, colonne ou diagonale

```
# Vérifier si la ligne 0 contient trois symboles identiques
indices_X = [0,0,0]
indices_Y = [0,1,2]
compteur = 0
for i in range(len(indices_X)):
    x = indices_X[i]
    y = indices_X[i]
    if grille[x][y] == 'X':
        compteur +=1
print("il y a ", compteur, "cases X")
```

Questions :

1. Utiliser les exemples proposés pour programmer les fonctions de l'interface relative aux tableaux statiques. *Utiliser le plus souvent possible ces fonctions pour les questions suivantes.*
2. Le joueur O joue en `position [1][2]` (ligne 1, colonne 2). Ecrire l'instruction qui modifie la grille.
3. Quelles sont les coordonnées `morpion[i][j]` pour chacune des cases ? Dessinez sur votre cahier une matrice pour placer les coordonnées `[i][j]` dans chaque case.

4. Ecrire une fonction `compte(grille,car,liste_X,liste_Y)` qui parcourt les cases de coordonnées (`liste_X[0], liste_Y[0]), (liste_X[1], liste_Y[1]), (liste_X[2], liste_Y[2])` et retourne le nombre de fois qu'apparait le caractère `car`

5. Ecrire une fonction `verifie(grille,car,liste_X,liste_Y)` qui verifie si les cases de coordonnées `(liste_X[0], liste_Y[0]), (liste_X[1], liste_Y[1]), (liste_X[2], liste_Y[2])` contiennent au moins un symbole `car`, et retourne les coordonnées de la première case contenant le caractère `car`.

6. Ecrire les instructions qui verifient si la première ligne contient trois symboles identiques (condition de victoire).

7. Ecrire une fonction `victoire` qui prend pour paramètre une matrice `grille` comme la grille de morpion (3*3). `grille` contient dans chaque case `'X'`, `'O'`, ou `' '`. Cette fonction retournera `True` si l'une des lignes/colonne/diagonale est remplie avec trois `'X'`.

1.2 Exercice 1.2 : Grille de Sudoku (9×9)

Voici une grille de Sudoku partiellement remplie :

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

Les cases vides sont représentées par 0.

Exemple : Compter les cases vides

```
# Compter les 0 dans toute la grille
compteur = 0
for i in range(len(sudoku)):
    for j in range(len(sudoku[i])):
        if sudoku[i][j] == 0:
            compteur += 1
print("Nombre de cases vides :", compteur)
```

Exemple : Afficher une ligne

```
# Afficher la ligne 0
for j in range(len(sudoku[0])):
    print(sudoku[0][j], end=' ')
print()
```

Exemple : Afficher une colonne

```
# Afficher la colonne 2
for i in range(len(sudoku)):
    print(sudoku[i][2], end=' ')
```

```
print()
```

Exemple : Modifier une case

```
# Remplacer le 0 en position [0][2] par 4
sudoku[0][2] = 4
print("Nouvelle valeur :", sudoku[0][2])
```

Exemple : Extraire un bloc 3×3

```
# Extraire le bloc en haut à gauche (lignes 0-2, colonnes 0-2)
bloc = []
for i in range(0, 3):  # lignes 0, 1, 2
    ligne_bloc = []
    for j in range(0, 3):  # colonnes 0, 1, 2
        ligne_bloc.append(sudoku[i][j])
    bloc.append(ligne_bloc)
print(bloc)
```

Exemple : Vérifier les doublons dans une ligne

```
# Vérifier si la ligne 1 contient des doublons (sans compter les 0)
ligne = sudoku[1]
nombres = []
for valeur in ligne:
    if valeur != 0:
        if valeur in nombres:
            print("Doublon détecté !")
        else:
            nombres.append(valeur)
```

Questions avec parcours par indice :

1. Programmer les fonctions qui permettent d'afficher les chiffres par ligne, par colonne. Chacune de ces fonctions retournera une liste de 9 chiffres.
2. Créez une fonction `bloc` qui extrait l'un des 9 blocs 3×3, n'importe où dans la grille. (par exemple, le bloc en haut à gauche : lignes 0-2, colonnes 0-2), à partir des indices `i,j` de départ (coin supérieur gauche du bloc). Cette fonction devra retourner une liste de 9 chiffres, ceux constituant le bloc.
3. Programmer une fonction `remplacer`, qui devra affecter une nouvelle valeur dans une case où il y a un 0.
4. Programmer une fonction `nb_zeros` qui devra compter le nombre de cases vides (valeur 0) dans toute la grille.
5. Utilisez les fonctions de votre interface pour affichez tous les chiffres de la première ligne (ligne d'indice 0).
6. Affichez tous les chiffres de la cinquième colonne (colonne d'indice 4).
7. Remplacez le 0 en position [0][2] par le chiffre 4.
8. Ecrire une fonction `doublons`, qui prend en paramètre une grille de sudoku `G`, ainsi qu'un numero de ligne `i`, et qui retourne un booléen `True`/`False`. selon si la ligne `i` contient des *doublons*. (sans compter les 0).
9. Appeler la fonction `doublons` pour vérifier si la première ligne contient des *doublons*.

Partie 2

Exercice 2 : moyenne glissante et tableau statique

Les courbes de données issues du monde réel sont souvent *bruitées*. Pour simuler ce type de données, nous allons utiliser une liste de valeurs cumulées aléatoires.

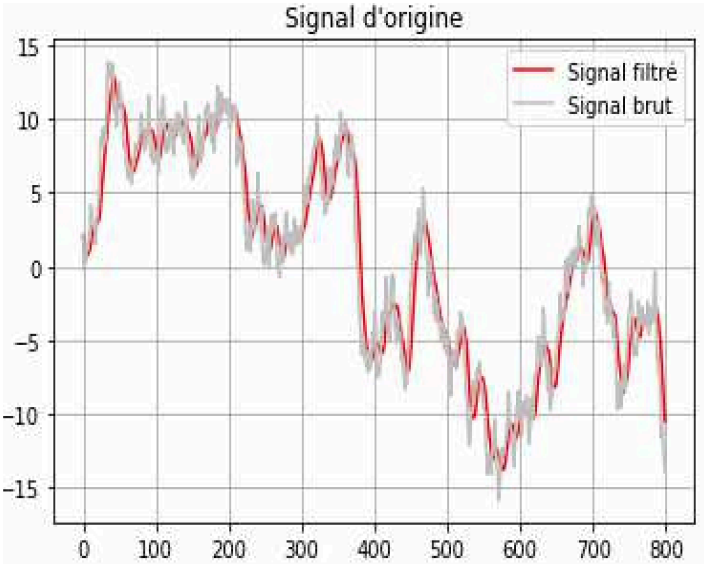

FIGURE 1 – courbes de donnees brutes et données filtrees

Etapes du traitement : Fonction `lissage`

- On fait une copie par valeur de `signal2` (liste des valeurs bruitées) dans `signal_filtre` (liste dont les termes seront remplacés par la valeur moyennée) :

```
1 signal_filtre = signal2.copy()
```

- On choisit une certaine largeur de liste pour les valeurs dont on fait la moyenne. Par exemple largeur = 10.
- Au rang i, dans la liste bruitée `signal2` : On prélève les valeurs entre les rangs $i - largeur//2$ et $i + largeur//2$ que l'on stocke dans un tableau STATIQUE appelé `signal`. Ce tableau conserve la même *taille* pendant tout l'exercice. (ici, largeur = 10) :

```
1 signal = signal2[i-largeur//2:i+largeur//2]
```

- On calcule la moyenne de valeurs de `signal` avec la fonction `moyenne`, comme par exemple celle vue dans l'exercice précédent.

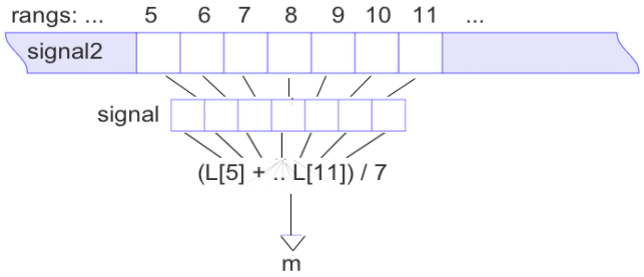

FIGURE 2 – moyenne sur l'ensemble des valeurs de `signal`

```
signal_filtre[i] = moyenne(signal)
```

- On place la valeur moyenne dans `signal_filtre[i]`

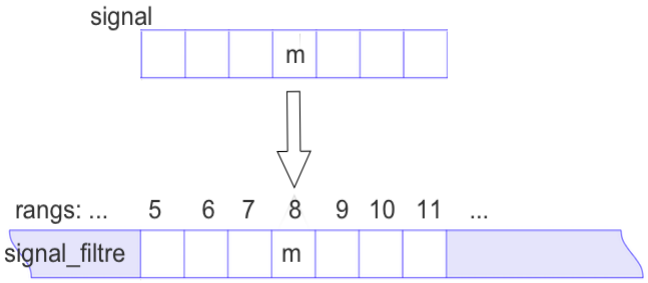

FIGURE 3 – signal filtre contient 10 valeur identiques contigües, il n'y a plus de bruit

- On répète l'opération pour tous les index `i` compris entre `largeur//2,len(signal2)-largeur//2)` :

```
for i in range(largeur//2,len(signal2)-largeur//2):
```

Puis on affiche les graphique de `signal2` (inchangé) et celle du signal après traitement (courbe lissée).

Nous allons suivre ces étapes :

1. Ecrire le script de la fonction `moyenne`

2. Créer une fonction `lissage`, qui prend en paramètres une liste `L` et une largeur `v`, et retourne une liste de valeurs filtrées par une moyenne glissante sur `v` valeurs.

3. Placer les valeurs filtrées dans une liste `signal_apres_traitement` : `signal_apres_traitement = lissage(signal2,10)`

4. Afficher les 2 courbes, `signal2` et `signal_apres_traitement`

Partie 3

Corrections

3.1 Sudoku

```
def check_ligne(G,num_ligne):
    test = True
    nombres = []
    for val in G[num_ligne]:
        if val != 0:
            if val in nombres:
                test = False
            else:
                nombres.append(val)
    return test

def check_lignes(G):
    for i in range(len(G)):
        if check_ligne(G,i) == False:
            return False
    return True
check_lignes(sudoku)
True
```

3.2 Filtrage : Exercice 2

```
def moyenne(signal):
    s = 0
    b = len(signal)
    for elem in signal:
        s += elem
    return s / b



def lissage(L,largeur):
    signal_filtre = L.copy()
    for i in range(largeur//2,len(L)-largeur//2):
        signal = L[i-largeur//2:i+largeur//2]
        # signal est le petit tableau de dimension largeur
        # dont on fait la moyenne glissante
        # puis on stocke dans signal_filtre
        signal_filtre[i] = moyenne(signal)
    return signal_filtre

signal_apres_traitement = lissage(signal2,10)
```