Programmation 2 Spring, 2026

TD 9: Sets, Tuples

1. Une fois les lignes de code suivantes exécutées, quelles valeurs sont stockées dans `output_set` ?

```
input_list = [3,1,4,1,5,9,2,6,5,3,5,9]
output_set = set([])
for i in input_list:
    output_set.add(i)
```

Solution.

`{1, 2, 3, 4, 5, 6, 9}`

2. Écrivez un code en une seule ligne pour supprimer les doublons d'une liste.

   `Example:` Pour `[1, 1, 2, 3, 4, 2]`, il faut calculer `[1,2,3,4]`.

Solution.

```
A = [1,1,2,3,4,2]

B = list(set(A))

print(B)
```

3. Écrivez une fonction `myenumerate` qui prend une liste et renvoie une liste de *tuples* contenant (index, item) pour chaque élément de la liste. Faites-le de deux manières :

   i) avec une boucle, et

   ii) avec une compréhension de liste.

   `Example:` Pour `[1, 1, 2, 8]`, il faut retourner `[(0,1), (1,1), (2,2), (3,8)]`.

Solution.

Boucle:

```
def myenumerate(A):
    B = []
    for i in range(0, len(A)):
        B.append((i,A[i]))
    return B

print(myenumerate([1,1,2,8]))
```

Compréhension de liste:

```
def myenumerate(A):
    return [ (i, A[i]) for i in range(0, len(A)) ]

print(myenumerate([1,1,2,8]))
```

4. Écrivez une fonction `mytriplets` qui prend un nombre n en argument et renvoie une liste de *toutes* les triplets tels que la somme des deux premiers éléments du triplet soit égale au troisième élément. Chaque nombre utilisé dans un triplet doit être supérieur à 0, et inférieur ou égal à n.

   Veuillez noter que (a, b, c) et (b, a, c) représentent le même triplet.

   Donnez deux codes : le premier utilisant des boucles, le second utilisant des compréhensions de liste.

   `Example:` Pour `n = 5`, il va retourner `[(1,1,2), (1,2,3), (1,3,4), (1,4,5), (2,2,4), (2,3,5)]`.

   Solution.

```
def mytriplets(n):
    A = []
    for i in range(1, n+1):
        for j in range(i, n+1):
            if i+j <= n:
                A.append((i,j,i+j))

    return A

print(mytriplets(8))
```

```
def mytriplets(n):
  return [(i,j,i+j) for i in range(1, n+1) for j in range(i,n+1) if i+j <= n]

print(mytriplets(8))
```

5. La fonction `zip`, déjà présente en Python, prend deux itérables en entrée et renvoie un itérateur contenant une paire du premier et du deuxième itérables.

   Le i-ième tuple est : $\Big($**i-ième élément du premier itérable, i-ième élément du deuxième itérable**$\Big)$.

   `Example:`

```
A = [3,5,1, "hello"]
B = [2,9]

C = zip(A,B)

print(type(C))

D = set(C)
print(D)
```

$\Downarrow$

```
<class 'zip'>
{(3, 2), (5, 9)}
```

   Supposons que nous ayons deux listes, A et B, qui donnent les coordonnées x et y d'un ensemble de points. Créez **une liste** avec les coordonnées (x, y) sous forme de tuple à l'aide de la fonction `zip`.

   Solution.

```
A = [3.234, 13, 1.1, 56.1]
B = [67, 12.12, 123, 62]

C = list(zip(A,B))
print(C)
```

6. (**Sudoku TP.**) La grille de Sudoku est stockée dans une liste de listes nommée `data`, où chaque liste contient une ligne de la grille. Voici un exemple :

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

Un 0 dans une cellule indique une valeur manquante.

Étant donné l'input `data` et une ligne i ne contenant qu'un seul zéro, comment trouver la valeur manquante ?

Solution.

```
zeros = [j for j in range(0, 9) if data[i][j] == 0]

if len(zeros) == 1:
    somme_ligne = sum(data[i])
    valeur_manquante = 45 - somme_ligne
    data[i][zeros[0]] = valeur_manquante
```

7. (**Sudoku TP.**) La grille de Sudoku est stockée dans une liste de listes nommée `data`, où chaque liste contient une ligne de la grille. Voici un exemple :

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

Un 0 dans une cellule indique une valeur manquante.

Étant donné l'input `data` et une colonne j qui ne contient qu'un seul 0, comment déterminer la valeur manquante dans cette colonne ?

Solution.

```
colonne = [data[i][j] for i in range(0, 9)]

zeros = [i for i in range(0, 9) if data[i][j] == 0]

if len(zeros) == 1:
    somme_colonne = sum(colonne)
    valeur_manquante = 45 - somme_colonne
    data[zeros[0]][j] = valeur_manquante
```

8. (**Sudoku TP.**) La grille de Sudoku est stockée dans une liste de listes nommée `data`, où chaque liste contient une ligne de la grille. Voici un exemple :

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

Un 0 dans une cellule indique une valeur manquante.

Notez qu'il y a des blocs de taille 3×3 :

```
data[0]:   [6, 1, 3, 4, 7, 8, 5, 2, 9]
data[1]:   [7, 4, 8, 5, 2, 9, 1, 6, 0]
data[2]:   [2, 5, 9, 1, 6, 3, 4, 7, 8]
           -------+-------+-------
data[3]:   [0, 2, 4, 6, 9, 5, 7, 3, 1]
data[4]:   [3, 7, 1, 2, 8, 4, 6, 9, 0]
data[5]:   [9, 6, 5, 7, 3, 1, 2, 8, 4]
           -------+-------+-------
data[6]:   [5, 3, 6, 8, 1, 0, 9, 4, 0]
data[7]:   [1, 8, 7, 9, 4, 2, 3, 5, 6]
data[8]:   [4, 9, 2, 3, 5, 6, 8, 1, 7]
```

Étant donné l'entrée `data` et une cellule (i, j), déterminez si **le bloc contenant** (i, j) ne contient qu'un seul 0 à la position (i, j), et le cas échéant, trouvez la valeur manquante.

Solution.

```python
block_row = (i // 3) * 3
block_col = (j // 3) * 3

block_sum = 0
zero_count = 0
zero_position = None

for r in range(block_row, block_row + 3):
    for c in range(block_col, block_col + 3):
        if data[r][c] == 0:
            zero_count += 1
            zero_position = (r, c)
        else:
            block_sum += data[r][c]

if zero_count == 1 and zero_position == (i, j):
     data[i][j] = 45 - block_sum
```

9. (**Sudoku TP.**) Le tableau de Sudoku est stocké dans une liste de listes nommé `data`, oú chaque liste contenant une ligne de tableau. Voici une exemple:

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

Un 0 dans une cellule indique une valeur manquante.

Étant donné l'input `data` et une cellule (i, j), comment déterminer si cette cellule n'a qu'un seul candidat possible après avoir examiné les chiffres présents dans sa ligne, sa colonne et son bloc ?

Il faut utiliser un `set`.

Solution.

Pour cellule (i, j) dans la grille, détermine l'ensemble des chiffres de 1 à 9 qui n'apparaissent ni dans la ligne i, ni dans la colonne j, ni dans le bloc contenant (i, j). Si cet ensemble contient exactement un chiffre, alors ce chiffre est l'unique candidat pour `data[i][j]`.

```
possibles = {1, 2, 3, 4, 5, 6, 7, 8, 9}

for col in range(0, 9):
    possibles.discard(data[i][col])

for ligne in range(0, 9):
    possibles.discard(data[ligne][j])

bloc_i, bloc_j = (i // 3) * 3, (j // 3) * 3
for di in range(0, 3):
    for dj in range(0, 3):
    possibles.discard(data[bloc_i + di][bloc_j + dj])

if len(possibles) == 1:
    data[i][j] = possibles.pop()
```