INFO-H-100 - Programmation
TP 9 — Boucles imbriquées et polynômes

Lorsque l'exercice demande d'écrire une fonction, écrivez la fonction demandée et testez-la avec plusieurs valeurs pertinentes.

Cette séance d'exercice s'effectue sur papier ! Veuillez ne pas ouvrir vos sessions sur les ordinateurs.

Ex. 1. Pour les exercices suivants, nous considérons qu'une matrice est représentée par une liste de listes de longueurs égales. Écrire une fonction `trace` à valeur réelle qui calcule la trace d'une matrice A de dimension $n \times n$ donnée en paramètre.

$$\texttt{trace}(A) = \sum_{i=1}^{n} A_{ii}$$

Ex. 2. Écrire une fonction qui compte le nombre de valeurs strictement négatives dans une matrice de nombres.

Ex. 3. Écrire une fonction qui détermine la somme des nombres positifs dans une matrice de nombres.

Ex. 4. Écrire les fonctions `symmetric` et `skew_symmetric` qui testent respectivement la symétrie et l'antisymétrie d'une matrice carrée.

Pour rappel, une matrice A_{ij} ($i, j \in [0, n[$) est dite symétrique si elle est égale à sa transposée, c'est-à-dire si $\forall i, j : 0 \leq i, j < n : A_{ij} = A_{ji}$ et antisymétrique si $\forall i, j : 0 \leq i, j < n : A_{ij} = -A_{ji}$.

Ex. 5. Écrire une fonction qui effectue le produit de la matrice $A(l \times m)$ par la matrice $B(m \times n)$.

Ex. 6. Écrire une fonction qui, pour un n donné, affiche le triangle "Nord-Est" comme dans l'exemple ci-dessous où n vaut 5 :

```
* * * * *
  * * * *
    * * *
      * *
        *
```

Ex. 7. Écrire une fonction qui, pour un n donné, affiche le triangle "Nord-Est" numéroté comme dans l'exemple ci-dessous où n vaut 5 :

```
1 2 3 4 5
  1 2 3 4
    1 2 3
      1 2
        1
```

Vous pouvez, vous-même, inventer un très grand nombre de variations sur le thème de ces triangles, par exemple, en en changeant l'orientation ou la numérotation. Vous pouvez aussi, par exemple, écrire une fonction qui affiche un noeud-papillon de taille n donnée comme ci-dessous (pour $n = 7$) :

```
*           *
* *       * *
* * *   * * *
* * * * * * *
* * *   * * *
* *       * *
*           *
```

Ex. 8. Écrire une fonction `p_eval` qui évalue un polynôme en un x donné.

```
>>> polynomial = [4,2,7,1]
>>> x = 2
>>> print(p_eval(polynomial,x))
44
```

Ex. 9. Écrire une fonction `p_add` qui additionne deux polynômes donnés et retourne le polynôme résultant.

```
>>> polynomial1 = [4,2,7,1]
>>> polynomial2 = [1,1,1]
>>> print(p_add(polynomial1,polynomial2))
[5,3,8,1]
```

Ex. 10. Écrire une fonction `p_derivative` qui retourne la dérivée d'un polynôme.

```
>>> polynomial = [4,2,7,1]
>>> print(p_derivative(polynomial))
[2, 14, 3]
```

Ex. 11. Écrire une fonction `p_sum_polys` qui retourne la somme d'une liste de polynômes.

```
>>> polynomials = []
>>> polynomials.append([1,2,3,4,5])
>>> polynomials.append([5,4,3,2])
>>> polynomials.append([5,2,3])
>>> polynomials.append([8,8,8,8,8,8])
>>> polynomials.append([51])
>>> print(p_sum_polys(polynomials))
[70, 16, 17, 14, 13, 8]
```

INFO-H-100 - Programmation
TP 9 — Boucles imbriquées et polynômes
Corrections

Solution de l'exercice 1:

```
def trace(matrix):
    trace = 0
    for i in range(len(matrix)):
        trace += matrix[i][i]
    return trace
```

Solution de l'exercice 2:

```
def count_negative_values(matrix):
    count = 0
    for line in matrix:
        for item in line:
            if item < 0:
                count += 1
    return count
```

Solution de l'exercice 3:

```
def sum_positive_values(matrix):
    total = 0
    for line in matrix:
        for item in line:
            if item > 0:
                total += item
    return total
```

Solution de l'exercice 4:

```
def symmetric(matrix):
    symm = True
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            if matrix[i][j] != matrix[j][i]:
                symm = False
    return symm
```

Comme alternative, voici une autre manière de programmer la fonction `symmetric`, en ne parcourant que les éléments nécessaires et en quittant la boucle dès qu'on a trouvé une paire d'éléments non symmétriques.

```
def symmetric(matrix):
    symm = True
    i = 0
    while (i < len(matrix)-1) and (symm == True):
        j = i+1
        while (j < len(matrix)) and (symm == True):
            symm = (matrix[i][j] == matrix[j][i])
            j+=1
        i+=1
    return symm
```

```
def skew_symmetric(matrix):
    skew_symm = True
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            if matrix[i][j] != -matrix[j][i]:
                skew_symm =  False
    return skew_symm
```

Solution de l'exercice 5:

```
def multiply_matrices(mat_a, mat_b):
    mat_res = []
    for l in range(len(mat_a)):
        new_line = []
        for n in range(len(mat_b[0])):
            value = 0
            for m in range(len(mat_a[l])):
                value += mat_a[l][m] * mat_b[m][n]
            new_line.append(value)
        mat_res.append(new_line)
    return mat_res
```

Solution de l'exercice 6:

```
def triangle_north_est(n):
    for i in range(n, 0, -1):
        for j in range(n - i):
            print(' ', end=' ')
        for j in range(i):
            print('*', end=' ')
        print()
```

Solution de l'exercice 7:

```
def triangle_north_est(n):
    for i in range(n, 0, -1):
        for j in range(n - i):
            print(' ', end=' ')
        for j in range(i):
            print(j + 1, end=' ')
        print()
```

Solution de l'exercice 8:

```
def p_eval(polynomial, x):
    value = 0
    for i in range(len(polynomial)):
        value += polynomial[i] * (x ** i)
    return value
```

Solution de l'exercice 9:

```
def p_add(poly1, poly2):
    if len(poly1) > len(poly2):
        new_poly = poly1
        poly_to_add = poly2
    else:
        new_poly = poly2
        poly_to_add = poly1
    for i in range(len(poly_to_add)):
        new_poly[i] += poly_to_add[i]
    return new_poly
```

Solution de l'exercice 10:

```
def p_derivative(polynomial):
    derivative = []
    for i in range(1, len(polynomial)):
        value = polynomial[i] * i
        derivative.append(value)
    return derivative
```

Solution de l'exercice 11:

```
def p_sum_polys(polynomials):
    poly_res = polynomials[0]
    for i in range(1, len(polynomials)):
        poly_res = p_add(poly_res, polynomials[i])
    return poly_res
```