Numpy coté MATRIX

Exercice 1 : Manipulations de base

Répondre à chaque question avec **une seule ligne de code** :

1. Créer un tableau A qui contient tous les multiples de 3 entre 0 et 100.
2. Créer un tableau B qui contient les sinus des carrés des éléments de A multipliés par $\pi/2$
3. Déterminer le maximum du tableau B et le garder dans une variable.
4. Compter le nombre de fois que ce maximum est présent dans B.

Exercice 2

Les fonctions demandées peuvent ne faire qu'une seule ligne de code.

Aide :
np.triu(A) Renvoie le tableau A dont les éléments en-dessous de la diagonale ont été annulés.
np.tril(A) Renvoie le tableau A dont les éléments au-dessus de la diagonale ont été annulés.
Un test A==B renvoie un tableau de booléens.
Pour savoir si ce tableau ne contient que des True on utilise (A==B).all()

1. Ecrire une fonction « est_diagonale(M) » qui renvoie True si M est diagonale, False sinon.
2. Ecrire une fonction « est_triangulaire_sup(M) » qui renvoi True si la matrice M est triangulaire supérieure, False sinon.

Exercice 3

Soit la matrice de taille 8

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Faire trois constructions différentes de cette matrice, dont une avec des boucles(s) et une autre sans boucle et sans entrer tous les coefficients un à un. (Il peut être utile d'utiliser la fonction concatenate())

Exercice 4 :

Pour cet exercice, on prend n = 1000000. On pourra augmenter ou diminuer cette valeur en fonction de la machine.

1. Calculer Σi de 0 à n sans utiliser Numpy
2. Chronométrer le temps nécessaire pour le calcul précédent en utilisant la méthode « time.clock() »
3. Utiliser un tableau Numpy et la méthode sum pour effectuer le même calcul
4. Comparer le temps de calcul des deux méthodes.

Exercice 5

1. Définir une matrice aléatoire a de taille 50 x 50

2. Déterminer la valeur $Max_{i,j}|a_{i,j+1}-a_{i,j}|$

Exercice 6

1. Définir une matrice aléatoire A de flottants, de taille 50 x 50
2. Compter le nombre de valeurs inférieures à 0.5 (On pourra utiliser $print(True+True) \rightarrow 2$)
3. Remplacer toutes les valeurs inférieures à 0.5 par des 0 et autres par 1

Exercice 7

On s'intéresse au système linéaire suivant :

$$\begin{cases} x_1 + 2x_2 + x_3 + x_4 = 0 \\ x_2 + 2x_3 + x_4 = 0 \\ x_1 + x_3 + 2x_4 = 1 \\ 2x_1 + x_2 + x_4 = 0 \end{cases}$$

1. Vérifier qu'il n'y a qu'une solution à ce système
2. En utilisant np.linal.solve, déterminer cette solution.
3. Vérifier le résultat obtenu en utilisant un produit matriciel.

Exercice 8 : RLE pour le challenge

L'algorithme de compression RLE (Run Length Encoding) est un algorithme utilisé dans de nombreux formats de fichiers (BMP, TIFF,...). Il est basé sur l'identification de la répétition consécutive de mêmes éléments.
Le principe consiste à parcourir la séquence de données à compresser et de donner séquentiellement le nombre de répétitions consécutives d'un élément donné suivie de la description de cet élément.

La séquence de données AAAAAACCCCAABAAAAAADDDD soumise à l'algorithme RLE identifie :

- une séquence de 6 caractères A → 6A,
- une séquence de 4 caractères C → 4C,
- une séquence de 2 caractères A → 2A,
- une séquence de 1 caractère B → 1B,
- une séquence de 6 caractères A → 6A,
- une séquence de 4 caractères D → 4D

et donne la séquence compressée suivante : 6A4C2A1B6A4D.

Définissez une fonction RLEcompress(tab) admettant pour argument un tableau de caractères, et retournant la version compressée de ce tableau.

Proposez ensuite la fonction de décompression RLEuncompress().