

DEVOIR DE SYNTHESE SEMESTRE 2**Matière : INFORMATIQUE****Classes : MP1, PC1, PT1 - Durée : 2 h**

Les algorithmes demandés doivent être écrits en langage Python. On sera très attentif à la rédaction et notamment à l'indentation du code.

Exercice 1 : Tracé des données d'un fichier (4 points)

Des données expérimentales sont disponibles sous forme d'un fichier texte nommé «data.txt» dans lequel les informations sont présentées sous forme de colonnes. La première colonne du fichier correspond au temps (unité : s), la deuxième au débit molaire d'un gaz A (unité : mol/s) enregistré au cours d'une première expérience, et la troisième colonne au débit molaire de A (unité : mol/s) enregistré au cours d'une seconde expérience.

Extrait du fichier **data.txt**

1 ^{ère} ligne du fichier	00.0 , 1.9231E-04 , 1.9230E-04
2 ^{ème} ligne du fichier	10.0 , 1.8293E-04 , 1.9108E-04
...	20.0 , 1.7401E-04 , 1.8991E-04
	30.0 , 1.6552E-04 , 1.8878E-04
	40.0 , 1.5745E-04 , 1.8771E-04
	50.0 , 1.4977E-04 , 1.8667E-04

1) Donner le code permettant de :

- ouvrir le fichier «data.txt» qui contient les données expérimentales.
- créer trois listes T, A1 et A2 correspondant respectivement aux nombres réels de la première, deuxième et troisième colonne en tenant compte de la séparation des données.
- déterminer le nombre total de points expérimentaux n .

2) Donner le code permettant de tracer simultanément sur un même graphe les deux courbes représentant les débits molaires de A en fonction du temps au cours des deux expériences en appelant la première courbe "expérience_1" et la deuxième "expérience_2".

Exercice 2 : Recherche d'un mot ou d'une sous-chaîne (6 points)

On se propose d'écrire un code permettant de rechercher un mot ou une suite de caractères m (non vide) dans une chaîne de caractères ch , avec $len(m) < len(ch)$.

1) Ecrire une fonction **position_car(ch, c, d)** qui prend en argument une chaîne ch , un caractère c et un entier $d < len(ch)$ et retourne le premier indice $p \geq d$ tel que $ch[p] = c$ s'il existe et $len(ch)$ sinon.

2) Ecrire une fonction **verifier_mot(ch, m, d)** qui prend en arguments deux chaînes non vides ch et m , un entier d et retourne *True* ou *False* selon que $ch[d], \dots, ch[d+len(m)-1] = m$ ou pas.

Nota :

ch[d+len(m)-1] ne doit pas dépasser la fin de la chaîne *ch*, ce qui garantit que l'on n'aura pas atteint la fin de la chaîne *ch* avant d'avoir fini de tester.

3) Ecrire une fonction **chercher(ch, m, d)** qui, en utilisant les deux fonctions précédentes, retourne un couple (*res, pos*) où *res* est un booléen *True* ou *False* selon que *m* est sous-chaîne de *ch* ou pas, *pos* est un entier donnant l'indice à partir duquel on trouve *m* dans *ch* si elle y figure, *len(ch)* sinon.

4) Ecrire une fonction **chercher_tous(ch, m)** qui prend en arguments deux chaînes non vides *ch* et *m* et retourne la liste (éventuellement vide) des indices du début de chaque occurrence de *m* dans *ch*.

Exemple :

```
>>> ch = "012xyz55xyz349xy00"
>>> m = "xyz"
>>> chercher_tous(ch, m)
[3, 8]
```

Problème (10 points)

Dans ce problème, on se propose d'étudier en Python quelques propriétés associées à des matrices carrées particulières en utilisant les tableaux *numpy*.

1) Ecrire une fonction **Puissance(M, p)** qui prend en entrée une matrice carrée *M* et un entier strictement positif *p* et renvoie M^p .

Indication : Utiliser la commande **dot** du module *numpy*.

2) Une matrice carrée *M* d'ordre *n* est celle d'un projecteur si on a : $M^2 = M$.

Ecrire une fonction **EstProjecteur(M)** qui prend comme paramètre une matrice carrée *M* et retourne *True* si *M* est la matrice d'un projecteur et *False* sinon.

3) On dit qu'une matrice carrée *M* d'ordre *n* est nilpotente s'il existe un entier naturel *k*, appelé indice de nilpotence, tel que M^k est égale à la matrice nulle d'ordre *n* avec $k \leq n$. L'indice de nilpotence est le plus petit $k \leq n$.

Exemple :

$$u = \begin{pmatrix} 3 & 9 & -9 \\ 2 & 0 & 0 \\ 3 & 3 & -3 \end{pmatrix} \quad u^3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

u est bien nilpotent d'indice 3.

Ecrire une fonction **IndNilpotence(M)** qui prend comme paramètre une matrice carrée *M*, supposée non nulle, et retourne l'indice de nilpotence s'il existe et 0 sinon.

Nota : On peut utiliser la commande *numpy A.all()* pour tester si tous les éléments de la matrice *A* sont tous *True*.

Exemple :

```
>>> A = np.array([[False, True, True], [False, True, True], [True, False, True]])
>>> A.all()
False
```

4) On dit qu'une matrice carrée *M* d'ordre *n* est circulante si chaque ligne d'indice *i* est la permutation circulaire vers la droite (d'une case) de la ligne d'indice *i* - 1.

Exemple : $M = \begin{pmatrix} 1 & a & b & c \\ c & 1 & a & b \\ b & c & 1 & a \\ a & b & c & 1 \end{pmatrix}$ est une matrice circulante.

Ecrire une fonction **EstCirculante(M)** qui prend comme paramètre une matrice carrée *M* et retourne *True* si *M* est circulante et *False* sinon.

5) Soit *L* une liste de nombres définie par : $L = [c_0, c_1, c_2, \dots, c_{n-1}]$. On se propose de générer une matrice circulante carrée *C* à partir de la liste *L*, tel que :

- la 1^{ère} ligne de *C* est égale à : $[c_0, c_1, c_2, \dots, c_{n-1}]$.

- la 2^{ème} ligne de *C* est égale à : $[c_{n-1}, c_0, c_1, \dots, c_{n-2}]$ et ainsi de suite.

$$C = \begin{pmatrix} c_0 & c_1 & c_2 & \dots & c_{n-1} \\ c_{n-1} & c_0 & c_1 & \dots & c_{n-2} \\ c_{n-2} & c_{n-1} & c_0 & \dots & c_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_1 & c_2 & c_3 & \dots & c_0 \end{pmatrix}$$

Ecrire une fonction **GenCirculante(L)** qui prend comme paramètre une liste *L* et retourne une matrice circulante à partir de la liste *L*.

6) Sachant que le produit de deux matrices circulantes est une matrice circulante. Ecrire une fonction **PordCirculant(A, B)** qui prend comme paramètre deux matrices carrées *A* et *B* de même ordre, supposées circulantes, et retourne leur produit matriciel en exploitant la propriété des matrices circulantes.