Informatique Pour Tous

Sujet d'Approfondissement 3 – Matrices et vecteurs en fichier texte ou str

A / Jeu de décryptage

Une variable de type *str* est une chaîne de caractères. Elle a quelques points communs avec les listes, on peut notamment :

- En déterminer la longueur à l'aide de len(). Ex :

```
toto = "lapin"
len(toto) # renvoie 5
```

- Extraire un ou plusieurs caractères de la chaîne. Ex :

```
print(toto[0]) # affiche "l"
print(toto[-3 :]) # affiche "pin"
```

En revanche, attention : il y a plusieurs différences notables, particulièrement liées à la portée des variables. Les types *str* suivent la même logique que les *bool, int* et *float*. Notamment :

- On ne peut pas modifier 1 seul caractère sans méthode spécifique (ex : .replace(), .strip() , …)
  Ex : toto[0] = 's' # on imagine qu'on veut passer « lapin» en « sapin », mais renvoie une erreur.

- Contrairement aux listes, les méthodes et fonctions s'appliquant aux *str* **renvoient** une sortie (locale) mais n'affectent pas la variable en argument elle-même !
  Illustration :
  - # Avec des listes :

```
L = [1, 2, 5]

L.append(6) # la méthode affecte la variable en argument avec une
            # portée globale
print(L) # affiche [1, 2, 5, 6]
```

  - # Avec des *str*:

```
mot = "coucou"

mot.replace("c","d") # cette ligne calcule le résultat, mais comme
                     # il n'est pas affecté, il est perdu.
print(mot)           # affiche "coucou" : inchangé
mot =  mot.replace("c","d") # ici on écrase l'ancienne valeur de la
                            # variable par la nouvelle
print(mot)           # Là, ça affiche bien "doudou"
```

Vous avez ci-dessus un exemple d'utilisation de la méthode `.replace()`. On souhaite reproduire cette fonction (il est donc hors sujet de faire appel à cette méthode).

Q1 – Écrire une fonction *remplacer_caractere(texte, c1, c2)* qui prend en arguments une chaîne de caractères *texte,* ainsi que deux caractères *c1* et *c2*, et qui renvoie la valeur qu'aurait *texte* si toutes les occurrences de *c1* étaient remplacées par *c2*.

Aide : on rappelle que `"text1" + "text2"` donne une concaténation `"text1text2"`.

Test :

```
print(remplacer_caractere ("coucou", "c", "d")  # doit afficher "doudou"
```

Challenge : quel(s) appel(s) (en une seule ligne !) doit-on faire pour passer de `"PTSI"` à `"PT"` ?

Vous trouverez un texte encrypté dans le fichier *fichier_crypte.txt*. Nous souhaitons ici lire ce fichier et **afficher** (pas de return, fonction *procédurale*) le texte décrypté. Pour décrypter, voici l'algorithme :

- Chaque ligne correspond à un caractère
- Attention, il faut convertir les 'O' (O majuscules) en '0' (chiffre zéro), car chaque lettre doit correspondre à un nombre. Pour ce faire, vous utiliserez la fonction écrite en **Q1**.
- La fonction permettant de passer d'un *int* ainsi lu en *str* (fonction de décryptage) est chr(a) qui prend en argument un nombre entier, et renvoie le caractère décrypté. Dans l'aide de Python, on a : La fonction Python chr() est une fonction intégrée utilisée pour obtenir n'importe quel caractère dont la valeur du code Unicode est connue . La valeur du code Unicode sous la forme d'un entier est passée en argument dans la fonction. La fonction renvoie le caractère ou la chaîne correspondant.

Voici le corps (à trous : à copier-coller puis compléter) de la fonction de décryptage :

```
def decrypt() :
    file = ...      # ouvrir le fichier fichier_crypte.txt qui doit être dans le même
                    # répertoire que le fichier python.
    lignes = file.readlines()
    phrase = ...    # à initialiser « vierge », c'est ce qu'on renvoie à la fin
    for element in lignes :
        valeur = ... # remplacer les O par 0 dans element, à l'aide d'un appel à
                     # votre fonction précédente. À cette ligne on a converti en int
        ... # appel de chr() pour décoder le caractère, puis mise de ce caractère à
            # la suite de la phrase
    print(phrase)           # affichage du résultat une fois le texte lu et décrypté
```

Q2 – ⌨ Copier-coller le code donné, puis exécuter la fonction (sans argument) et découvrir le message crypté.

B / Matrice en *str* (assez difficile car peu guidé – si bloqués, passer à l'exo C)

Soit le *str* suivant : `Matrice = "1,1\t2,2\t3,3\n4,4\t5,5\t6,6\n7,7\t8,8\t9,9"`

La commande `print(Matrice)` affiche :

```
1,1 2,2 3,3
4,4 5,5 6,6
7,7 8,8 9,9
```

On souhaite « ranger » automatiquement les valeurs de cette Matrice dans une variable `M` de type liste de listes :

`M` ⟵ `[[1.1, 2.2, 3.3], [4.4, 5.5, 6.6], [7.7, 8.8, 9.9]]`

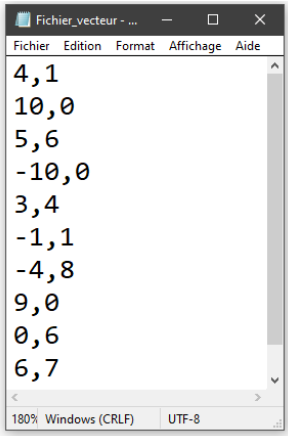
Vous remarquerez que pour les valeurs en *str*, le séparateur entre la partie entière et les décimales est une virgule, alors que ça doit être un point en *Python* ! Dans cette partie (et pour la suite), vous avez le droit d'utiliser la méthode `.replace()`.

Remarque: `readlines()` ne s'applique pas aux *str*. Ici on utilisera uniquement des `split()` (sur "\n" pour les lignes, "\t" pour les colonnes)

Q3 – ⌨ Écrire le script, s'appuyant sur deux boucles *for* imbriquées, permettant de faire ce « rangement ».

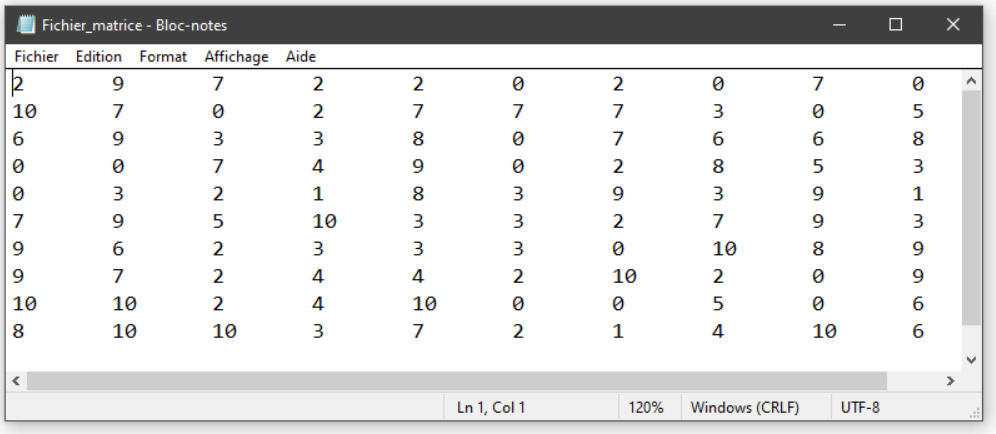
C / Calcul d'un produit matriciel

On imagine qu'il y a un capteur communicant qui envoie une série de mesures, présentes dans le fichier *Fichier_vecteur.txt.* Vous avez un exemple de fichier, mais on ne peut pas simplement *copier-coller* les valeurs, car on suppose que ce fichier est changé au cours du temps par le capteur communicant. Ce vecteur compte *N* lignes. Ici *N* est donné avec 10 lignes, mais cette valeur peut changer au cours du temps, elle devra donc être variable.

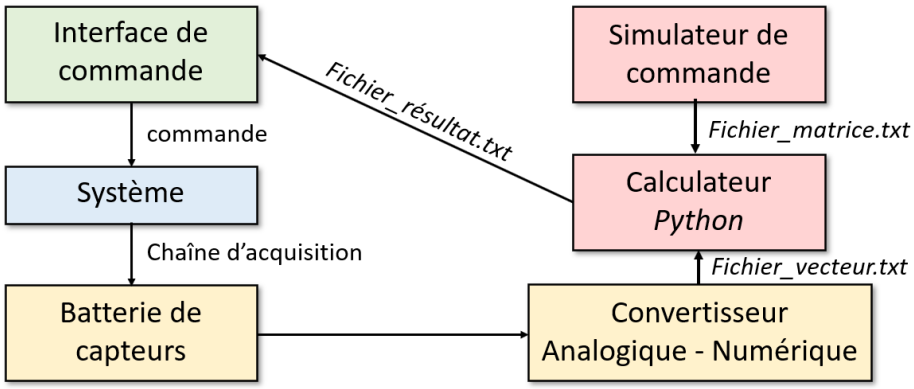


Noter que, comme dans l'exercice précédent, le séparateur de décimales est une virgule dans ce fichier *texte* alors que ça doit être un point en *Python.*

Par ailleurs, on suppose qu'un simulateur externe calcule en temps réel une matrice, à coefficients entiers, qui est donnée dans le document *Fichier_matrice.txt*. Cette matrice est une matrice *N x N* : *N* lignes, et *N* colonnes (même taille que le vecteur).



→ *Python* est ici utilisé pour calculer le produit **Matrice x Vecteur** et écrire le résultat dans un nouveau fichier *Résultat.txt* , qui sera utilisé par l'interface de commande.



Objectif – L'objectif de ce dernier exercice du TD est de lire le vecteur et créer une variable *Numpy* correspondante. De même pour la Matrice, faire le produit **Matrice x Vecteur** et enregistrer le résultat dans un nouveau fichier.

Dans ce sujet, nous n'utiliserons pas de fonctions, pour simplifier. Au début de votre code, pensez à importer la bibliothèque *Numpy*, sous l'alias classique `np`.

Q4 – Ouvrir le fichier *Fichier_vecteur.txt* en lecture seule, et stocker toutes ses données dans une variable **globale** de type *Numpy.array* qui s'appellera `Vecteur`.

Test : `print(Vecteur)  # affiche [4.1  10.  5.6  -10.  3.4  -1.1  -4.8  9.  0.6  6.7]`

Q5 – ⌨ Ouvrir le fichier *Fichier_matrice.txt* en lecture seule, et stocker toutes ses données dans une matrice carrée de type *Nympy.array*, de même taille que `Vecteur`, variable qu'on notera `Matrice`.

Rappel : pour initialiser une matrice vide, de taille *N* x *M* (*N* lignes, *M* colonnes), on tape :

```
np.zeros([N, M])
```

Pour modifier la valeur de l'élément $M_{i,j}$ (i^{ème} ligne, j^{ème} colonne d'une matrice `Mat`), on tape :

```
Mat[i,j] = … # mettre ici la valeur à insérer dans la matrice
```

➔ Ainsi, pour balayer tous les éléments d'une matrice, il faut deux boucles **for** imbriquées !

- 1 première boucle `for i in range(Nlignes) : # balaiera toutes les lignes`
- 1 seconde boucle **for** imbriquée dans la 1ère, qui pour un *i* fixé par la première boucle, balaiera toutes les colonnes : `for j in range(Ncolonnes) : # affichera la matrice (table) ci-contre`

Test : `print(Matrice)`

```
[[ 2  9  7  2  2  0  2  0  7  0]
 [10  7  0  2  7  7  7  3  0  5]
 [ 6  9  3  3  8  0  7  6  6  8]
 [ 0  0  7  4  9  0  2  8  5  3]
 [ 0  3  2  1  8  3  9  3  9  1]
 [ 7  9  5 10  3  3  2  7  9  3]
 [ 9  6  2  3  3  3  0 10  8  9]
 [ 9  7  2  4  4  2 10  2  0  9]
 [10 10  2  4 10  0  0  5  0  6]
 [ 8 10 10  3  7  2  1  4 10  6]]
```

Si *A* est une matrice carrée de dimension *N x N*, *B* un vecteur de longueur *N*, alors par définition du produit matriciel, si on note $V = A \times B$ alors :

$$\forall i \in [\![1, N]\!], \qquad V_i = \sum_{j=1}^{N} A_{ij} B_j$$

Dans le cas de $N = 2$, on obtient :

$$V = \begin{pmatrix} V_1 \\ V_2 \end{pmatrix} = A\,B = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \end{pmatrix} = \begin{pmatrix} A_{11}\,B_1 & A_{12}\,B_2 \\ A_{21}\,B_1 & A_{22}B_{\,2} \end{pmatrix}$$

Q6 – ⌨ Écrire une fonction `produit(A, B)` prenant en argument une matrice carrée *A* (type *numpy.array*) et un vecteur *B*. Cette fonction renverra le vecteur produit A x B.

Remarque : il existe en *numpy* une fonction « toute faite » de calcul matriciel : `Matrice.dot(Vecteur)`. Il est ici interdit d'utiliser cette fonction : vous devez faire le calcul terme par terme. Pour chaque position ou index *i* (première boucle *for*), il vous faudra faire la somme de *j* termes (seconde boucle *for*, imbriquée).

Test : en revanche, on s'autorise l'utilisation de la fonction pré-compilée pour tester notre fonction.

```
print(Matrice.dot(Vecteur))
print(produit(Matrice, Vecteur)) # ces 2 lignes doivent afficher la même …
   # chose, à savoir [118.8 134.  206.2 115.3  51.  132.5 240.1 119.8 231.4 257.8]
```

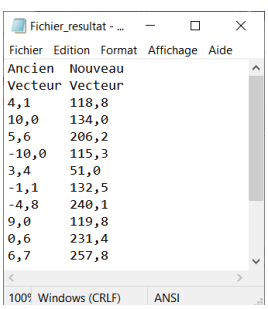
Q7 – ⌨ Ouvrir ou créer le fichier *Fichier_resultat.txt* en écriture, et écrire le résultat précédent, sous le même standard d'écriture que le standard du *Fichier_vecteur.txt*. On veut donc :

- Deux lignes d'en-tête (voir ci-dessous)
- Deux nombres par ligne (deux colonnes), séparées par **une tabulation**
  - La 1ère colonne est un simple « copier-coller » du vecteur initial. La 2ème colonne correspond au vecteur produit
  - Chaque ligne codant pour un nombre avec un seul chiffre significatif après la virgule, et le séparateur de décimale étant écrit avec une virgule, pas un point. Par exemple : « 3,4 ».
- **Aide :** la fonction `round(x, i)` renvoie l'arrondi d'un *float* `x` à `i` chiffres après la virgule. Par défaut, si un seul argument, `i` est sous-entendu ==0, et on renvoie un *int*.

```
round(3.1416, 2)    # renvoie 3.14
round(3.1416, 3)    # renvoie 3.142
round(3.1416)    # renvoie 3
round(3.8)    # renvoie 4
```

- *N* lignes (autant que dans le *Fichier_vecteur.txt*) (+ les 2 lignes d'en-tête).

Remarque : bien penser à **fermer** (.close()) le fichier à la fin du script !



Test : le fichier généré peut alors être ouvert avec un autre logiciel (*Notepad++, Bloc-notes* pour les *Windows*) :