04- Interaction avec des fichiers

Le but de ce TP est de pouvoir interagir avec un fichier depuis un programme Python, et d'utiliser efficacement les données qui se trouvent dans un tel fichier.

On se contentera dans ce TP de gérer des fichiers textes (extension .txt en général, mais aussi parfois .csv ,nous en reparlerons dans les exercices)

1 Ouverture en mode lecture d'un fichier existant

1.1 La syntaxe de base pour ouvrir un fichier en mode lecture

Consignes :

1. Copier dans votre répertoire de travail le fichier 'releve_bac.txt' qui se trouve dans le "cahier de prépas" de la classe..
2. Si vous voulez voir à quoi ressemble ce fichier, vous pouvez l'ouvrir dans le bloc-note par exemple. Une fois votre curiosité satisfaite, refermez-le.
3. Créer un script Python dans le même répertoire de travail.
4. Taper et tester les instructions suivantes :

```
f=open('releve_bac.txt','r') #on ouvre le fichier en lecture('r' pour read)
f.close() #on referme le fichier
```

Quand vous exécutez ce script, si vous n'avez pas de message d'erreur, c'est que Python a bien pu ouvrir le fichier, et l'a refermé.

Dans la suite nous allons bien sûr insérer des instructions entre ces deux lignes de codes pour faire quelque chose du contenu de ce fichier.

En fait, techniquement, dans la variable f a été créé un objet Python, de type fichier, dont le nom est complètement indépendant de celui du fichier texte (ici notre objet Python s'appelle f).

Pour Python, un fichier est une séquence de caractères : une fois qu'un fichier a été ouvert par un programme, celui-ci maintient un marqueur (fictif) à la position courante, qui indique à tout moment où sera lu/écrit le prochain caractère. Toute opération de lecture ou d'écriture fait bouger le marqueur vers l'avant.

Voici ce que l'on peut faire une fois que l'on a ouvert le fichier en mode lecture :

1.2 Lecture globale du fichier avec la méthode `read`

On transforme **la totalité** du texte du fichier en une **chaine de caractère** Python (type `string`) selon la syntaxe :

```
mon_texte=f.read()
```

Exemple 1 : Récupérer tout le contenu du fichier "releve_bac.txt" dans une variable de type **`string`**, faire afficher ce contenu dans la console python, et déterminer le nombre de caractères de ce fichier.

En réalité, si vous comptiez à la main les caractères que vous voyez, vous ne trouveriez pas le même nombre.
En effet, dans un fichier texte, le retour à la ligne est codé par le caractère ' $\backslash n$ ', une tabulation est codé par le caractère ' $\backslash t$ '.

Cette méthode de lecture globale du fichier n'est pas très pratique lorsqu'il s'agit d'exploiter un fichier de données, où chaque ligne correspond à une donnée particulière. On utilisera donc, dans la quasi-totalité des cas que nous rencontrerons, la méthode suivante :

1.3 Lecture par lignes du fichier avec la méthode `readlines`

L'opération consiste à récupérer toutes les lignes sous forme d'une **liste de chaines de caractères**, selon la syntaxe :

```
ma_liste=f.readlines()
```

Chaque élément de la liste renvoyée est une ligne du fichier, sous forme d'une chaine de caractère.

Exemple 2 : : 1) Ecrire et exécuter le script suivant :

```
f=open('releve_bac.txt','r') #on ouvre le fichier en lecture
ma_liste=f.readlines()
f.close() #on ferme le fichier
print(ma_liste)
```

2) Ecrire des instructions à la suite de ce script permettant de :

a) Connaître le nombre de lignes du fichier.

b) Faire afficher uniquement la première ligne du fichier.

c) Faire afficher les trois derniers caractères de chaque ligne.

1.4 Instructions utiles pour traiter des fichiers de données

En sciences, on est souvent amener à traiter des fichiers de données : chaque ligne du fichier correspond aux caractéristiques d'une donnée particulière. Ces caractéristiques sont alors séparées par un séparateur (un point-virgule, une tabulation...), on parle alors de fichier **csv**. Le sigle **csv** signifie Comma-Separated Values. Ces fichiers peuvent être notamment lus par un logiciel type Excel (chaque ligne du fichier texte correspondra alors à une ligne d'un tableau).

Dans Python, une fois récupérée chaque ligne du fichier, on pourra séparer les caractéristiques qui la composent grâce à la méthode **`split`**.

Exemple 3 : Voici un exemple d'utilisation de **`split`**, toujours avec la liste issue du relevé des notes du bac :

```
f=open('releve_bac.txt','r') #on ouvre le fichier en lecture ('r' pour read)
ma_liste=f.readlines()
f.close() #on ferme le fichier
liste_notes=[]
for ligne in ma_liste:
    couple=ligne.split(';') #on créée une liste en séparant la chaine suivant le caractère ';'
    liste_notes.append(float(couple[1])) #on transforme la chaine couple[1] en nombre flottant
```

Calculer alors la moyenne obtenue par cet élève (en supposant que toutes les matières ont le même coefficient).

2 Exercices de lecture d'un fichier

Exercice 1 : Copier et placer dans votre répertoire de travail le fichier : « courbe_mystère.csv ».

Ce fichier contient 600 lignes construites selon le modèle suivant :

688.64;0.0
680.1754833863976;35.706105780659676
655.0698019258382;68.96639993920128

Chaque ligne contient une chaine de caractères correspondants à deux valeurs flottantes séparées par un point-virgule, représentant respectivement l'abscisse et l'ordonnée d'un point.

1. Ouvrir le fichier en lecture, et créer une liste nommée `total` contenant toutes les lignes du fichier.
2. En parcourant tous les éléments de la liste `total` grâce à une boucle, construire la liste LX des abscisses et la liste LY des ordonnées contenues dans ce fichier. Le type des éléments de LX et de LY doit être flottant.
3. Une fois le script ci-dessus exécuté, faire afficher les listes LX et LY, et vérifier (grâce à Python, pas à la main !) qu'elles sont bien toutes les deux de longueur 600 .
4. Représenter les points sur une figure, dans un repère orthonormé.

On rappelle que pour faire des tracés, il faut importer le module `matplotlib.pyplot`, et qu'ensuite l'instruction de base est : `plot(liste des abscisses,liste des ordonnées)`.

Remarque : pour forcer Python à utiliser la même unité en abscisses et en ordonnées, vous utiliserez l'instruction : `plt.axis('equal')`

Exercice 2 : 1. Importer dans votre répertoire de travail le fichier 'chute.csv'.

Ce fichier contient les relevés effectués lors de la chute d'un mobile.

Voici les premières lignes de ce fichier :

t(s)	x(m)	y(m)
0	0	0.25
0.04	0.2	0.30216
0.08	0.4	0.33864
0.12	0.6	0.35944

Chaque ligne (hormis la première qui contient l'intitulé des trois colonnes) contient une chaine de caractères correspondants à trois valeurs flottantes séparées par une tabulation (on rappelle qu'une tabulation est codée par '\t'), représentant respectivement le temps du relevé, l'abscisse et l'ordonnée du mobile au temps correspondant.

2. En utilisant les mêmes méthodes que dans l'exercice précédent, effectuer le tracé de la trajectoire du mobile au cours du temps.

3 Ouverture en mode écriture d'un fichier

Si on souhaite créer un fichier et écrire des choses dans ce fichier, on l'ouvre alors en mode écriture, selon la syntaxe :

```
f=open('nouveau_fichier.txt','w') #on ouvre le fichier en écriture ('w' pour write)
```

Attention, l'option ' w ' créée automatiquement un nouveau fichier texte dans lequel écrire, mais surtout va écraser un fichier texte déjà existant portant le même nom.

Remarque(pas au programme) : Si on veut modifier un fichier existant, il faut ouvrir le fichier en ajout d'écriture, avec la syntaxe :**`f=open ('mon_fichier_déjà_là.txt','a')`**

Pour écrire ensuite dans ce fichier, on utilise la méthode **`write()`**, selon la syntaxe :

```
f.write('blablabla')
```

Si on effectue plusieurs write de suite, les différents textes seront écrits les uns à la suite des autres.

Exemple 4 : Tester les instructions suivantes :

```
f=open('mon_nouveau_fichier.txt','w')
f.write('Voici les nombres impairs compris entre 1 et 100:\n')
for i in range(1,101,2):
    f.write(str(i)+'\n')
f.close()
```

Utilisez ensuite l'explorateur pour découvrir le fichier que vous avez créé.

Attention, l'argument dans **`write`** doit toujours être une chaine de caractères.

Tout objet d'un type différent doit préalablement être converti en une chaine de caractères. (par exemple **`str(42)`** convertit l'entier 42 en la chaine de caractères '42')

4 Exercices de lecture et d'écriture dans un fichier

Exercice 3 : 1. Importer dans votre répertoire de travail le fichier **`candidats.txt`**.

Le fichier **`candidats.txt`** contient des lignes de la forme :

1234 DUPONT Jean Admissible 2 **ou** 4321 DUPOND Jeanne Eliminé -

(Il s'agit des candidats au concours commun des Mines en 2013 pour la filière MP).

Des tabulations séparent les quatre champs, ces champs étant :

- Le numéro de candidat
- Le nom et le prénom sont séparés par un espace et sont dans un même champ.
- Le troisième champ est le texte «Admissible» ou «Eliminé»
- Le quatrième champ est le numéro de la vague d'oral si le candidat est admissible, un tiret sinon.

2. Compter, à l'aide de Python bien sûr, le nombre total de candidats.
3. Compter le nombre de candidats admissibles.
4. Compter le nombre de candidats attribués à chaque série d'oral (il y en a 4).
5. Créer quatre nouveaux fichiers texte, contenant la liste des candidats de chaque série d'oral (uniquement nom et prénom).

Exercice 4 : On cherche à calculer une valeur approchée de l'intégrale d'une fonction donnée par des points dont les coordonnées sont placées dans un fichier « integrale.csv ».

Le fichier « integrale.csv ». contient une quinzaine de lignes selon le modèle suivant :

0.0;1.00988282142

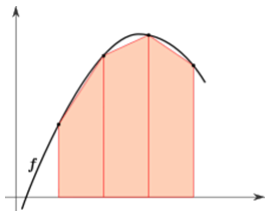
0.1;1.07221264497

Chaque ligne contient une chaine de caractères correspondants à deux valeurs flottantes séparées par un point-virgule, représentant respectivement l'abscisse et l'ordonnée d'un point. Les points sont donnés par abscisses croissantes.

1. Ouvrir le fichier en lecture, et construire la liste LX des abscisses et la liste LY des ordonnées contenues dans ce fichier. Le type des éléments de LX et de LY doit être flottant.
2. Représenter les points sur une figure.

Les points précédents sont situés sur la courbe représentative d'une fonction f. On souhaite déterminer une valeur approchée de l'intégrale I de cette fonction sur le segment où elle est définie.

Pour cela on va utiliser la méthode des trapèzes, consistant à ajoutant les aires de tous les trapèzes délimités par l'axe des abscisses et le segment reliant deux points consécutifs.



3. Ecrire une fonction **`trapeze`**, d'arguments deux listes **`liste_abs`** et **`liste_ord`** de même longueur (correspondant aux abscisses et aux ordonnées des points successifs), qui renvoie l'aire obtenue en appliquant la méthode des trapèzes.
4. Utiliser cette fonction pour déterminer une valeur approchée de notre intégrale I.

En plus : Remarques générales pour l'organisation de vos fichiers
Je vous conseille de toujours travailler avec un fichier que vous aurez au préalable rangé dans le même répertoire que votre script Python, ainsi vous pourrez utiliser, comme ci-dessus, la syntaxe simple suivante :

`f=open('chute.csv','r')` (on dit que l'on indique le chemin relatif du fichier)

Si toutefois votre fichier est rangé ailleurs, vous devez alors préciser le chemin absolu du fichier (c'est-à-dire en partant de la racine de l'arborescence) :

```
f=open('C :/doc/PSI/chute.csv','r')
```

(Attention sous Windows la syntaxe d'écriture des chemins se fait avec des \, contrairement à Python qui utilise des /, donc à transformer si besoin)

De plus Python n'accepte pas les noms de dossiers ou de fichiers comportant des espaces (comme beaucoup d'autres langages).
Une bonne pratique en informatique consiste à ne jamais utiliser d'espaces pour nommer les dossiers ou les fichiers. Utilisez des underscores à la place (exemple : dossier_travail)