TP9

Manipulation de fichiers

Le langage Python est particulièrement plébiscité par la communauté scientifique pour traiter de grandes quantités de données (par exemple des mesures accumulées par un capteur sur une longue période, ou encore dans le cadre de l'instruction d'une intelligence artificielle). Ces données sont en général stockées dans des fichiers adaptés. Nous nous concentrerons ici sur le cas où les fichiers sont de simples fichiers contenant du texte.

Ces objets sont censés être très volumineux, et n'ont pas vocation à être ouverts littéralement. En revanche, on va pouvoir accéder au contenu de ces fichiers à l'aide de Python, afin d'en manipuler les données.

Dans ce qui suit, tous les exemples utilisent un fichier *source* censé se trouver dans le dossier de travail courant, contenant les données que nous allons manipuler.
Ce fichier sera un gros fichier *csv*, pioché dans l'OpenData français. On place le chemin vers ce fichier dans une variable python :

```
fichier_source = 'U:/fr-esr-sise-effectifs-d-etudiants-inscrits-esr-public.csv
'
```

Remarque 1

Un fichier de type « csv » est un simple fichier texte dont chaque ligne contient des données séparées par des virgules (ou des points-virgules, ou encore des tabulations).

I Lire dans un fichier

I.1 - Première étape : l'ouverture du fichier en lecture

```
source = open(fichier_source,'r',encoding="utf-8")
```

Cette instruction ouvre le fichier en créant un objet Python, et affecte cet objet à la variable `source`. Le second argument, `r'` pour *read*, indique que le fichier est ouvert en « lecture ». On ne pourra pas en modifier le contenu.

I.2 - Dernière étape : la fermeture du fichier

Ce type d'objet occupe en général une place mémoire importante, c'est pourquoi, avant d'en apprendre plus sur un tel objet, il ne faudra pas oublier, suite aux manipulations de cet objet, de fermer cet objet avec l'instruction :

```
source.close()
```

On aurait alors un script de la forme :

```
source = open(fichier_source,'r')
# des instructions
source.close()
```

I.3 - Les étapes entre la première et la dernière...

L'objet Python créé par `open` est un objet de type incompréhensible :

```
>>> source = open(fichier_source,'r')
>>> type(source)
<class _io.TextIOWrappe>
```

Mais on ne se laisse pas impressionner par un nom aussi compliqué : on peut se figurer cet objet comme un *curseur*, qui pointe au début de la première ligne du fichier, et auquel on peut demander de lire des lignes et d'en retourner le contenu sous forme de chaînes de caractères. Il faut comprendre que, chaque fois que ce curseur nous retourne une ligne, il passe à la ligne suivante. Pour revenir sur des lignes déjà lues, il faudra fermer cet objet et le rouvrir ou bien utiliser la méthode `seek()` avec l'argument 0 pour revenir au début de la première ligne, comme dans `source.seek(0)`.
Cet objet qui sert à lire les lignes d'un fichier possède un certain nombre de *méthodes*. L'instruction `dir(source)` renvoie en les méthodes :
`'close', 'newlines', 'read', 'readline', 'readlines', 'seek', 'write', 'writelines'.`

Remarque 2 (*Faire une boucle sur les lignes du fichier*)

L'objet créé par `open()` est aussi un *itérable* dont chaque terme est une ligne du fichier. On peut le parcourir dans une boucle `for`, comme on le ferait pour une liste, ou une chaîne de caractères, ou encore l'objet `range()`.

Par exemple, le code ci-dessous donne une manière de compter le nombre de lignes du fichier :

```
source = open(fichier_source,'r')
nombre_lignes = 0 #initialisation du compteur
for ligne in source : # on fait une boucle sur les lignes du fichier
    nombre_lignes += 1
print("le fichier contient", nombre_lignes, "lignes")
source.close()
```

il affiche :
`le fichier contient 333603 lignes.`

I.4 - Les méthodes `read()`, `readlines()`, et `readline()`

- La méthode `read()` :
  Cette méthode permet de lire d'un coup l'intégralité du fichier. Elle retourne une unique chaîne de caractères formée de tous les caractères des lignes, à partir de la position du curseur.

```
source = open(fichier_source, 'r')
print("le fichier contient ", len(source.read()), "caracteres")
# affiche la longueur de la chaine de caracteres totale
# le curseur pointe a present a la fin du fichier
print("le fichier contient ",len(source.read()),"caracteres")
# affichera 0 car le curseur est a la fin du fichier
source.seek(0) # replace le curseur au debut du fichier
print("le fichier contient ",len(source.read()),"caracteres")
source.close()
```

affiche :

```
le fichier contient 316265155 caractères
le fichier contient 0 caractères
le fichier contient 316265155 caractères
```

Remarque 3

Si le fichier est très volumineux, cette méthode peut engorger rapidement les capacités de mémoire de Python !

- La méthode `readlines()` (bien noter le "s") :
cette méthode retourne une grande liste dont chaque élément est une ligne du fichier, sous forme d'une chaîne de caractères. Ici aussi, ce ne sont que les lignes situées après le curseur qui sont retournées.

```
source = open(fichier_source, 'r')
print("le fichier contient", len(source.readlines()), "lignes")
# affiche le nombre de lignes
source.close()
```

renvoie `le fichier contient 333603 lignes`

Remarque 4

Ici aussi, la liste obtenue peut être encombrante en cas de fichier de départ très volumineux.

- La méthode `readline()` : cette méthode renvoie la ligne courante (vers laquelle pointe le curseur) sous forme d'une unique chaîne de caractères, puis fait passer le curseur à la ligne suivante.

```
with open(fichier_source,'r') as source :
    print('ligne 1 :', source.readline())
    # affiche le contenu de la premiere ligne
    print('ligne 2 :',source.readline())
    # affiche le contenu de la deuxieme ligne
```

Exercice 1

Écrire les instructions Python permettant d'afficher le nombre de lignes du fichier contenant l'expression 'Bayonne'.
Indication : l'instruction `'Pyrénées' in "C'est joli les Pyrénées-atlantiques"` *renvoie* `True`.

II Pour traiter les données : manipulation des chaînes de caractères.

L'outil principal utilisé ici est la chaîne de caractères, de type `string`.

Imaginons que la première ligne d'un fichier (récupérée avec l'instruction `source.readline()` est la longue chaîne de caractères abrégée ci-dessous :

```
"numéro INSEE, nom, page wikipedia, surface(m²), longitude
(...)
code région, nom région \n"
```

⋆ On remarque tout d'abord le *caractère d'échappement* `\n` en fin de chaîne, qui indique un retour à la ligne.
Il y a parfois en début et/ou en fin de ligne un espace ou un caractère d'échappement, qui

nous importunent. Pour les enlever, on dispose de la méthode `strip()`, déclinable à droite : `rstrip()`, et à gauche : `lstrip()` :
Le curseur pointe à présent sur la ligne suivante, donc l'instruction `source.readline().strip()` retourne la chaîne de caractères (abrégée) ci-dessous :

```
'2114, Brasles, fr:Brasles, 7453763.000000000000000,
    (...)
1.3, 2, AISNE,22, PICARDIE'
```

où l'on constate la disparition du caractère de fin de ligne.

⋆ Ici, au sein de chaque ligne, les données sont séparées par une virgule.
En pratique, on veut souvent séparer et extraire les informations, ou une partie d'entre elles, contenues dans les lignes lues. La méthode `split()` des chaînes de caractères découpe la chaîne selon le caractère qu'on lui donne en paramètre, et retourne les morceaux dans une liste.

EXEMPLE

- L'instruction `"hé ho ça va pas non ?".split(' ')` donne la liste `['hé', 'ho', 'ça', 'va', 'pas', 'non', '?']`,
- tandis que `"hé ho ça va pas non ?".split('va')` donne `['hé ho ça ', ' pas non ?']`,
- et enfin, en ce qui concerne notre exemple :

```
source.readline().strip().split(',')
```

donne la liste :

```
['2292', ' Étampes-sur-Marne', 'fr:Étampes-sur-Marne', '
2207607.000000000000000', '3.418950315195310', ' 49.031578363504998', '
    Commune simple', '110', '1.2', '2', ' AISNE', '22', ' PICARDIE']
```

Remarque 5

Bien entendu, les chaînes de caractères (de type *string*) ont bien d'autres méthodes, que vous pouvez découvrir grâce à `dir('')`.

III Écrire des données dans un fichier

Pour écrire dans un fichier, on commence par l'ouvrir en écriture de l'une des deux façons suivantes :

⋆ `NomDeNotreChoix = open('nom_du_fichier','w')`

l'option `'w'` a pour effet de créer le fichier `nom_du_fichier` dans le répertoire courant. Si ce fichier existe déjà, son contenu sera alors effacé. L'objet `f` pointe alors au début du fichier.

⋆ `AutreNomDeNotreChoix = open('nom_du_fichier','a')`

l'option `'a'` *(pour append)* permet, dans le cas où le fichier existe déjà, de faire pointer l'objet f sur la ligne suivant la dernière ligne du fichier. Sinon, comme avec 'w', le fichier est créé.

Remarque 6

Attention : ici la fermeture avec `close()` est cruciale, car c'est à cet instant que le fichier va réellement être écrit sur le disque !

On peut écrire dans ce fichier des chaînes de caractères grâce à plusieurs méthodes.

III.1 - Les méthodes `write()` et `writelines()`

⋆ la méthode `write()` qui prend comme argument une chaîne de caractères, écrit cette chaîne de caractère à l'endroit pointé par le curseur, et positionne le curseur à la fin de la chaîne nouvellement inscrite.
Ainsi l'exécution de :

```
nouveau = open('fichier_test','w')
nouveau.write("les premiers mots du fichier")
nouveau.write("les mots suivants dans le meme fichier")
nouveau.close()

# le code qui suit sert a lire ce qu'on a ecrit juste avant
source = open('fichier_test','r')
print(source.read())
```

renvoie :
`les premiers mots du fichierles mots suivants dans le même fichier`
tandis que :

```
nouveau = open('fichier_test','a')
nouveau.write("\nd'autres mots du fichier a la ligne\n")
# ici on rajoute le caractere d'echappement \n
# pour passer a la ligne suivante
nouveau.write('les mots suivants dans le meme fichier')
nouveau.close()
# le code qui suit sert a lire ce qu'on a ecrit juste avant
source = open('fichier_test','r')
print(source.read())
source.close()
```

renvoie :
`les premiers mots du fichierles mots suivants dans le même fichier`
`d'autres mots du fichier à la ligne`
`les mots suivants dans le même fichier`

Remarque 7

En vérité, le caractère d'échappement `\n` est une indication à destination de la fonction `print()` ou tout autre méthode d'affichage du contenu du fichier.

⋆ La méthode `writelines()` qui prend comme argument une liste de chaînes de caractères, et écrit chacune de ces chaînes à la suite les unes des autres. Ici aussi, on termine en fermant le fichier, toujours avec `close()`.

```
nouveau = open("les_paroles",'w')
nouveau.writelines(["Encore des mots toujours des mots\n",
"les memes mots\n",
"Je n'sais plus comment te dire\n",
"Rien que des mots\n",
"que je ne cesserai jamais de lire\n",
"Des mots faciles, des mots fragiles"])
nouveau.close()
```

`\newpage`

```
source = open('les_paroles','r')
print(source.read())
source.close()
```

renvoie :

```
Encore des mots toujours des mots
les mêmes mots
Je n'sais plus comment te dire
Rien que des mots
que je ne cesserai jamais de lire
Des mots faciles, des mots fragiles
```

Exercice 2

Écrire une fonction `supprime(fichier,mot)` qui supprime toutes les chaînes de caractères `mot` du fichier nommé `fichier`

Exercice 3

Écrire une fonction `fusion(i,j,fichier)` qui remplace la colonne i par la fusion (ou plus précisément la concaténation) des colonnes i et j du fichier de type csv nommé `fichier`.

Exercice 4

Le fichier `liste_entiers.txt` contient une liste d'entiers (un seul par ligne). Faire la somme (vous devez trouver 53015853930180100603592).

Exercice 5

Le fichier `liste_couples.txt` contient des lignes de la forme `x;y` où `x` et `y` sont des flottants. Tracer le graphe correspondant aux `x` en abscisse et `y` en ordonnée.
Petit rappel d'utilisation de matplotlib :

```
import matplotlib.pyplot as plt
plt.plot(X,Y) #pour tracer
plt.show() #pour afficher
```

Exercice 6

Écrire une fonction `creation(f,n)` prenant en entrée un fichier `f` ouvert en écriture, et un entier `n`, et écrivant dans le fichier les entiers de 1 à n (un entier par ligne). En fin de fonction, le fichier est refermé.
Par exemple, le script suivant crée un fichier `essai.txt` contenant les entiers de 1 à 1000 :

```
h=open('essai.txt','w')
creation(h,1000)
```

Exercice 7

On dispose de données écrite dans un fichier nommé "data.csv".
La première ligne contient le texte : `largeur;longueur;hauteur`.
Chaque ligne suivante du fichier contient trois valeurs, qui représentent des flottants, séparées par des virgules. Par exemple `25.3, 37.5, 13.8`.

1. Ecrire un code permettant de lire les lignes de données et de stocker l'ensemble des lignes dans une liste dont chaque élément est une liste constituée par les trois valeurs de type `float` d'une ligne.
2. Ecrire une fonction calculant la hauteur moyenne
3. Modifier le code pour ne récupérer qu'une ligne de données sur dix (la première ligne de données, la onzième, ...)

Exercice 8 (*Numérotation*)

Écrire une fonction `numerotation` prenant en entrée un tableau `t` et renvoyant un couple `(t', d)` tel que :

- ⋆ t' contient les mêmes éléments que t mais sans doublon ;
- ⋆ d est un dictionnaire associant à chaque élément de t' son indice dans t'.

Cette fonction sert régulièrement car elle permet d'associer à n'importe quelle liste de n'importe quel type d'éléments des numéros qui les identifient de manière unique. Par exemple nous pourrons numéroter les sommets d'un graphe afin de créer la matrice qui indique comment sont reliés ces sommets.

Exercice 9 (*compression zip très simplifiée*)

1. Écrire une fonction `compression` qui prendra en entrée un tableau de mots `t` et qui renverra un couple formé de :
   - ⋆ un dictionnaire tel que celui obtenu dans l'exercice précédent, qui permet de numéroter les éléments le `t` ;
   - ⋆ un tableau d'entiers obtenu en remplaçant chaque élément de t par son numéro.
2. Écrire une fonction `decompression` réciproque de la précédente.
3. *Mise en pratique* : À présent, on veut écrire et lire les données compressées dans un fichier. Nous allons donc travailler la lecture et l'écriture dans un fichier.
   (a) Écrire une procédure `tab_vers_fichier` permettant d'enregistrer le contenu d'un tableau dans un fichier. On convient d'écrire un élément par ligne.
   (b) Écrire une procédure analogue `dico_vers_fichier`. On convient d'utiliser une ligne par entrée du dictionnaire, chaque ligne étant de la forme `mot:numéro`.
   (c) Écrire une fonction `tab_de_mots_of_fichier` qui prend un fichier enregistré dans un fichier .txt et renvoie la liste de mots correspondants.
   (d) Rassembler tous les morceaux pour obtenir une procédure prenant en entrée un fichier texte et créant un fichier contenant le texte compressé. Vérifier si le fichier obtenu est plus léger que le fichier source.
   (e) Écrire la fonction de décodage correspondante.

   Le fichier `ltdme80j-p.txt` vous permettra de tester vos programmes.