TP-Cours – Les fichiers csv

Objectifs :

- ⇨ Apprendre à lire et écrire dans un fichier texte en python
- ⇨ Organiser des données en tables
- ⇨ Découvrir un format de stockage de données en tables (csv)
- ⇨ Lire et écrire des fichiers csv en python

I – Lecture et écriture de fichiers en python

Python permet de lire et écrire des fichiers très simplement grâce à la fonction builtin `open` et l'objet file.
Python permet de lire aussi bien des fichiers texte que des fichiers binaires, mais nous ne verrons dans ce cours que les fichiers texte (vous pouvez vous référer à la documentation officielle pour davantage d'informations).

1) Lecture d'un fichier texte

Pour pouvoir manipuler un fichier, il est d'abord nécessaire de l'ouvrir avec la fonction `open`.
Celle-ci prend en argument le nom du fichier (avec le chemin complet, sinon il est cherché par défaut dans le répertoire courant) et le mode d'ouverture. Elle renvoie un objet file qui sert par la suite à gérer les opérations sur le fichier.

Le tableau ci-contre donne les principaux modes d'ouverture.

Mode	Nom	Effet
`'r'`	Read	Ouvre le fichier en lecture seule (par défaut)
`'a'`	Append	Ecriture en se plaçant à la fin du fichier si il existe déjà
`'w'`	Write	Ecriture en écrasant le contenu éventuel du fichier s'il existe déjà
`'x'`	Create	Ecriture en créant le fichier – provoque une erreur si le fichier existe déjà

Si le fichier à ouvrir n'existe pas et qu'on l'ouvre en lecture (`mode = 'r'`), python lève une exception `FileNotFoundError`.
On peut au préalable tester l'existence du fichier avec la fonction `isfile` du module `os.path` :

```
import os.path
if os.path.isfile("test.txt"):
    fichier = open("test.txt", "r")
```

Une fois le fichier ouvert, on peut utiliser les méthodes suivantes sur l'objet `file` retourné par la commande `open` pour réaliser des opérations :

Méthode	Action
`read()`	Lit l'intégralité du fichier en une seule fois et renvoi une chaîne avec le contenu du fichier
`readline()`	Lit une seule ligne. Renvoi un str ou une chaîne vide (`""`) si la fin du fichier est atteinte
`write()`	Ecrit la chaîne donnée en argument dans le fichier et renvoi le nombre de caractères effectivement écrits. Si on veut écrire des sauts de ligne (`'\n'`) il faut les inclure dans la chaîne)
`seek(0)`	Ramène le curseur de lecture/écriture au début du fichier.
`close()`	Ferme le fichier

Lorsqu'on lit ou écrit dans un fichier, le système d'exploitation gère un curseur qui correspond au nombre d'octet depuis le début du fichier et lui permet de savoir où il en est. On peut utiliser ce curseur en python avec les méthodes `tell()` et `seek()`, mais dans le cas d'un fichier texte, seule la méthode `seek(0)` qui ramène au début du fichier fonctionne de manière simple et correcte.

Important : A partir du moment où on a ouvert avec succès un fichier, il est indispensable de le refermer avec la méthode `close()` avant la fin du programme pour libérer les ressources correspondantes. En particulier, si on a écrit des choses dans le fichier et que l'on n'appelle pas `close()` avant de quitter, les dernières données ne seront probablement pas enregistrées correctement dans le fichier.

Question 1 :

1) Créez un fichier « test.txt » dans votre répertoire de travail avec un éditeur de texte (notepad ou notepad++). Ecrivez 4 ou 5 lignes de texte dans le fichier et sauvegardez-le.
2) Ecrire un programme python pour lire et afficher une par une les lignes de votre fichier jusqu'à arriver à la fin du fichier. Que remarquez-vous au niveau de l'affichage ?

NB : Si on souhaite se débarrasser des caractères '\n' à la fin des lignes, on peut utiliser la méthode replace de la classe `str`. Par exemple `ma_chaine.replace('a','b')` permet de remplacer tous les « a » de la chaîne `ma_chaine` par des « b ».

3) Un objet fichier est *itérable*. Comment cette information peut-elle simplifier l'écriture du programme précédent ? Mettre en œuvre cette modification.
4) Ecrire (et tester) une fonction `lecture_fichier` qui prend en argument le nom du fichier et renvoi un tableau contenant l'ensemble des lignes du fichier (une ligne par case du tableau) ou `None` si le fichier n'existe pas.

2) Utilisation de la clause `with`

Pour être sûr qu'un fichier ouvert sera bien fermé quoi qu'il arrive (oubli, erreur lors de l'exécution du code, …), tout en rendant le code bien lisible, python propose une syntaxe alternative avec le mot-clé `with` :

```
with open("monfichier.txt") as fichier:
    # traitement utilisant le fichier
    # dés qu'on sort de ce bloc, le fichier est automatiquement fermé
# Suite du programme
```

On remarque le « `:` » à la fin de la ligne du `with` qui indique qu'on a ensuite un bloc de code indenté. La fermeture est automatique et l'instruction `close()` n'apparaît donc pas dans le code (du coup on ne risque pas de l'oublier !).

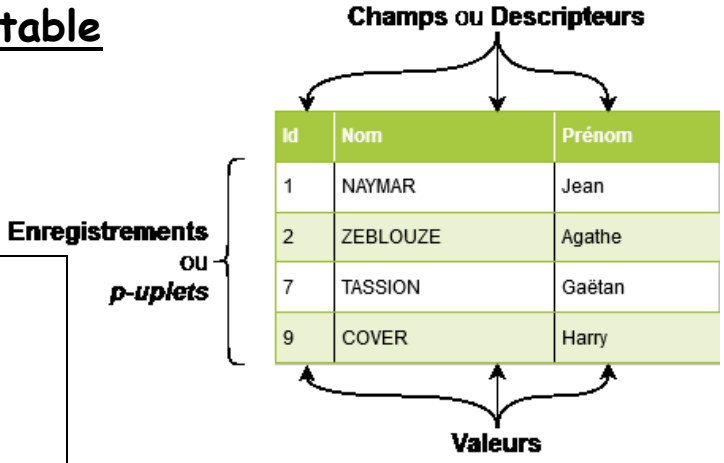
3) Ecriture d'un fichier texte

L'écriture est très similaire à la lecture. Elle se fait à l'aide de la méthode `write` vue plus haut. Là encore, l'utilisation d'un bloc `with` est conseillée pour être sûr que le fichier soit bien fermé (et donc toutes les écritures bien réalisées).
La méthode `write` renvoyant le nombre de caractères réellement écrits, on peut comparer cela au nombre de caractères que l'on souhaitait écrire pour détecter des éventuelles erreurs lors de l'écriture du fichier qui peuvent avoir lieu pour différentes raisons (disque plein, erreurs d'écriture, …).

II – Organisation des données en table

1) Définition

On appelle données en tables, toute collection de données présentées sous forme de tableau.



Id	Nom	Prénom
1	NAYMAR	Jean
2	ZEBLOUZE	Agathe
7	TASSION	Gaëtan
9	COVER	Harry

Définition :

De nombreuses données se présentent naturellement sous la forme de table de données, comme les contacts d'un agenda par exemple ou tous les tableaux présentés dans un livre d'histoire-géographie.

Enfin les tableurs (comme Calc ou Excel) présentent également les données sous la forme de tables.

2) Représentation en mémoire

Les enregistrements d'une table étant des p-uplets nommés, on les représentera naturellement en python avec des ..... Une table de donnée étant alors un ..... .

L'exemple précédent donnerait donc en mémoire :

```
agenda = [{"id":1, "Nom":"NAYMAR", "Prénom":"Jean"},
          {"id":2, "Nom":"ZEBLOUZE", "Prénom":"Agathe"},
          {"id":7, "Nom":"TASSION", "Prénom":"Gaëtan"},
          {"id":9, "Nom":"COVER", "Prénom":"Harry"}]
```

On a alors par exemple `agenda[2]["Nom"]` qui vaut ..... .

et `agenda[` ..... qui vaut `9`.

III – Le format csv

1) Description du format

Comme on l'a vu, les tableurs présentent naturellement leurs données en tables. Pour sauvegarder les données, ils proposent tous le format csv qui est un format simplifié standardisé permettant l'échange de données.

Question 2 :

1) A l'aide d'un tableur (Calc, Excel, …) ouvrir le fichier « countries.csv ». Quels sont les différents descripteurs de ce fichier ?
2) Ouvrir maintenant le même fichier à l'aide d'un éditeur de texte (Le bloc-note de windows ou notepad++ par exemple). Comment sont les données sont-elles enregistrées ?

csv vient de **C**omma **S**eparated **V**alues ou Valeurs Séparées par des Virgules.
Comme on l'a vu à la question 2, les valeurs des différents champs sont séparées par des « ; » et chaque enregistrement correspond à une ligne du fichier. En fait le format csv accepte différents séparateur pour les différents champs, les plus commun étant « `;` », « `,` » et tabulation (`"\t"` en python).

2) La bibliothèque csv

Une bibliothèque python permet de simplifier la manipulation des fichiers csv : la bibliothèque `csv`.
Elle possède essentiellement deux fonctions dont on donne ici une description simplifiée :

```
csvReader = csv.DictReader(Objet_Fichier [,delimiter=delimiteur])
csvWriter = csv.DictWriter(Objet_Fichier, fieldnames,
            [,delimiter=delimiteur] [,lineterminator = '\n'])
```

On doit donc fournir à ces fonctions un objet File (récupéré avec la fonction `open`) et on peut préciser le délimiteur utilisé dans le fichier (par défaut il utilise la virgule « , ») ainsi que le terminateur de ligne (« `\r\n` » par défaut).

3) Lecture

La fonction DictReader retourne un objet DictReader qui est un itérable renvoyant un *dictionnaire ordonné* (OrderedDict) pour chaque ligne du fichier).

Exemple d'utilisation :

```
with open('monFichier.csv') as fichier:
   reader = csv.DictReader(fichier, delimiter=';')   # Objet DictReader (itérateur)
   contenu = []
   for ligne in reader:
      contenu.append(dict(ligne))     # Conversion du type OrderedDict en dictionnaire
```

4) Ecriture

La création de l'objet DictWriter pour écrire un fichier csv nécessite un argument supplémentaire `fieldnames` qui est une liste contenant les descripteurs de la table (ou un dictionnaire dont les clés sont les descripteurs). Il est utile dans le cas de l'écriture de spécifier le terminateur de ligne pour éviter des sauts de lignes en doublon.
On peut alors utiliser les méthodes :

`writeheader()`	Ecrit la première ligne du fichier csv avec les descripteurs des champs.
`writerow(`*`ligne`*`)`	Ecrit une ligne dans le fichier csv ou *`ligne`* est un dictionnaire ayant les mêmes clés que `fieldnames`.
`writerows(`*`contenu`*`)`	Ecrit l'ensemble des lignes dans la liste de dictionnaires *`contenu`*.

Exemple d'utilisation :

```
with open("Agenda.csv", "w") as fichier:
    csvWriter = csv.DictWriter(fichier, agenda[0],delimiter=";",lineterminator='\n')
    csvWriter.writeheader()          # Ecrit l'en-tête avec les descripteurs
    csvWriter.writerows(agenda)      # Ecrit les données
```

Question 3 :

1) En utilisant la bibliothèque csv, écrire une fonction `lecture_fichier_csv` qui prend en argument un nom de fichier et renvoie `None` si le fichier n'existe pas et une liste de dictionnaires représentant la table de donnée du fichier s'il existe. Tester votre fonction avec le fichier « countries.csv ».
2) Ecrire une procédure `pays_du_continent` qui prend en argument une chaîne représentant le code du continent et affiche tous les pays (du fichier countries.csv) de ce continent. Tester votre procédure avec le code `"OC"` pour Océanie ou `"EU"` pour Europe.
3) Ecrire une procédure `fichier_pays_du_continent` qui prend en argument un code de continent et crée un fichier csv dont le nom est le code du continent suivi de « .csv » et qui contienne une table de tous les pays de ce continent. Tester votre procédure pour différents code de continent.

TRAVAIL A FAIRE POUR LA PROCHAINE SEANCE :

1) Aller sur la page internet https://www.data.gouv.fr/fr/datasets/indicateurs-de-suivi-de-lepidemie-de-covid-19/ et télécharger le fichier « table-indicateurs-open-data-france.csv » qui contient les dernières données sur l'évolution du covid-19 en France. Ouvrir le fichier avec un éditeur de texte et examiner sa structure (en particulier, essayer d'identifier le délimiteur des données).
2) En reprenant le travail fait pendant ce chapitre, écrire une fonction qui récupère les données du fichier « table-indicateurs-open-data-france.csv » et les stocke en mémoire.
3) En vous servant de la fiche de référence sur l'utilisation de la bibliothèque `pyplot`, écrire une procédure qui prend en argument le nom d'un champ du fichier précédent et trace l'évolution de ce champ en fonction du temps en utilisant la bibliothèque `pyplot`. On simplifiera en créant une liste de valeurs de 0 à (nombre de données -1) pour la liste des valeurs des abscisses et on ne testera la procédure que pour le champ `'taux_occupation_sae'` (le seul à avoir des valeurs pour chaque enregistrement).

Si vous le souhaitez, vous pouvez améliorer le programme en mettant à zéro les valeurs non disponibles (Valeur `'NA'`) pour pouvoir tracer les autres champs (`'R'` et `'tx_incid'`) et en traçant les trois courbes sur le même graphique avec des couleurs différentes.