1 Importer un fichier CSV en Python

Les données suivantes sont enregistrées dans le fichier **temperature.csv**. On veut les importer pour les traiter en Python. Il y a plusieurs possibilités...

```
annee,mois,jour,temp_deg
2021,1,1,2.3
2021,1,2,4.3
2021,1,3,5.2
2021,1,4,-4.5
2021,1,5,-2.6
```

Voici l'importation des données avec la bibliothèque **csv** de Python. L'import est sous forme d'une liste de dictionnaires sous la forme **clé : valeur**.

```
import csv

fichier  = open("temperature.csv")
data = csv.DictReader(fichier, delimiter = ",")
table = list(data)

# quelques sorties avec l'objet table :
print("nombre d'éléments: ", len(table))
print("table complète : ", table)
print("premier objet :" , table[0])
print("attribut du premier objet :" , table[0]["jour"])
print("attribut du premier objet :" , table[4]["temp_deg"])

# un exemple de parcours :
print("parcours :")
for elem in table :
    print(elem)

# transformer une str en int ou float :
val = float(table[3]["temp_deg"])
print(val)

# récupérer la liste des champs, ça peut servir
champs = list(table[0].keys())
print(champs)
```

Voici le contenu de l'objet **table** de manière compréhensible :

```
[
OrderedDict([('annee', '2021'), ('mois', '1'), ('jour', '1'), ('temp_deg', '2.3')]),
OrderedDict([('annee', '2021'), ('mois', '1'), ('jour', '2'), ('temp_deg', '4.3')]),
OrderedDict([('annee', '2021'), ('mois', '1'), ('jour', '3'), ('temp_deg', '5.2')]),
OrderedDict([('annee', '2021'), ('mois', '1'), ('jour', '4'), ('temp_deg', '-4.5')]),
OrderedDict([('annee', '2021'), ('mois', '1'), ('jour', '5'), ('temp_deg', '-2.6')])
]
```

2 Exporter un fichier au format CSV en Python

Voici comment exporter une table de données écrite sous forme d'une liste de dictionnaires en Python. En pratique, on peut regrouper sous forme de fonctions les imports-exports dans un fichier CSV.

```
import csv

# les données à exporter sont dans la variable table
table =  [ { 'nom' :  'dupond' , 'age' : 18} ,
           {'nom' : 'durand', 'age' : 22} ,
           {'age' : '30', 'nom' : 'dubois'}
         ]
print(table)



fichier  = open('age.csv', 'w', newline = '')
champs = ['nom', 'age']
writer = csv.DictWriter(fichier, fieldnames = champs, delimiter = ";")
writer.writeheader()
writer.writerows(table)
fichier.close()

"""
on peut aussi ajouter les enregistrements un par un avec :
writer.writerow({'age' : '30', 'nom' : 'dubois'})
"""
```

Quelques exercices sur les deux premières parties : on utilisera le fichier **CSV** des températures de la partie précédente. Il est conseillé répondre à chaque exercice dans un fichier dédié.

1. Sur le fichier de températures, on s'est rendu compte qu'il y a un biais sur la sonde de température : elle affiche 0,2 °C en trop. Proposer un algorithme qui importe les températures, les corrige, et sauvegarde au format CSV dans le fichier **temp_corr.csv**. Il ne doit pas modifier le fichier initial !

2. Sur le fichier de températures, on souhaite ajouter un champ **temp_fahr** pour disposer aussi de la température en degré Fahrenheit. Proposer un algorithme qui importe les températures et ajoute ce champ, calcule automatiquement les valeurs, et sauvegarde le tout dans le fichier **temp_fahr.csv**.

3. Sur le fichier de températures, on souhaite, pour un traitement ultérieur simplifié, ne retenir uniquement les données sur les jours impairs. Proposer un algorithme qui importe les températures, sélectionne automatiquement les bonnes « lignes », et sauvegarde le tout dans le fichier **temp_impair.csv**.

4. Écrire une fonction stats_csv(fichier) qui prend comme paramètre le nom d'un fichier (c'est à dire que fichier est un nom de fichier CSV) et qui affiche le nombre de lignes (on dit plutôt **enregistrements**) et le nombre de colonne (on dit plutôt **champs**) de la table stockée dans le fichier, et tester cette fonction.

5. Sur le fichier de températures, on souhaite, pour un traitement statistique ultérieur (moyenne, minimum, maximum ... et même graphique) extraire seulement les températures en °C au format **flottant**, et sous forme d'un tableau Python (list). Proposer un algorithme qui renvoie ces données.

3 Validation des données

Quelques généralités à retenir sur les CSV

Les données tabulées sont omniprésentes. Un moyen simple de diffuser des données est de les stocker dans des fichiers **au format CSV**. La bibliothèque standard Python possède un module **csv** permettant de charger des fichiers CSV comme des **tableaux de dictionnaires** Python et de sauvegarder de tels tableaux dans des fichiers au format CSV. Le module **csv** charge les données comme des **chaîne de caractères**. Il est donc nécessaire d'écrire du **code de validation** afin de convertir les valeurs vers le type approprié et de vérifier qu'elles respectent les contraintes attendues.

Le format **CSV** est très libre, et l'utilisation de la fonction **DictReader** de la bibliothèque Python **csv** provoque presque jamais d'erreurs, même lorsque le fichier est « malformé ». Elle pourra en revanche renvoyer des dictionnaire incorrects. Voici quelques points à bien retenir.

- La première ligne d'un fichier **CSV** est toujours utilisée comme ligne d'en-tête : elle doit contenir les noms des différents champs. Si ce n'est pas le cas, les données de la première lignes servent comme clé, ce qui peut être gênant !
- Si la ligne d'en-tête comporte moins de champs que les autres lignes, alors les champs supplémentaires sont associés à la clé **None** (sans guillemets) comme un tableau.

Application : nous allons reprendre le fichier de températures de la page 1, et nous allons créer une fonction **valide(ligne)** qui vérifiera chaque ligne de la **table**. Compléter cette fonction pour que la **table_valide** respecte les contraintes suivantes. Tester cette fonction, puis vérifier son efficacité en modifiant les fichier CSV.

1. les années doivent être au format entier ;
2. les mois doivent être au format entier, avec des valeurs comprises entre 1 et 12 ;
3. les jours doivent être des entiers, avec des valeurs comprises entre 1 et 31 ;
4. les températures en °C doivent être des flottants compris entre −80 et 100.

```
def valide(ligne) :
        annee = ligne[annee]
        ...
        if mois < 1 or mois > 12 :
                exit("mois invalide")
        ...
        return ...


table_valide = [valide(ligne) for ligne in table]
```

Remarques :

- la dernière ligne permet de créer automatiquement le tableau de dictionnaire par **compréhension**. C'est un moyen rapide créer des tableaux ou des dictionnaires.
- Pour vérifier la question 1, on utilisera **try** ... **except** pour éviter l'arrêt du programme...

4 Fusion de deux tables

Notions introduites dans cette partie :

Quand des tables possèdent un champ commun, il est possible de combiner et fusionner les données de ces tables. Cette notion s'appelle simplement la **fusion** et de **jointures** de tables.

4.1 Un exemple de fusion de deux tables : réunir les prénoms sur deux années

Nous disposons de fichiers qui contiennent les prénoms donnés par les parents à leurs enfants. Les fichiers disponibles sont **année**, sur le modèle ci-dessous.

prenoms2005.csv

```
annaiss, sexe, prenom , nombre
2005 , F , Emma, 1523
2005 , M , Mathéo, 2835
2005, F , Léa , 3410
2005, M, François, 26
```

prenoms2006.csv

```
annaiss, sexe, prenom , nombre
2006 , F , Emma, 1235
2006 , M , Mathéo, 2456
2006, F , Léa , 1623
2006, M, Damien, 56
```

1. Pour un traitement global sur les prénoms, il est cohérent de vouloir réunir ces tables étant donné qu'elles ont les mêmes champs.

   Proposer un algorithme qui :

   (a) importe les données ;

   (b) fusionne les données ;

   (c) exporte la nouvelle table obtenue dans un nouveau fichier **prenomsFusion.csv**

   Le fichier créé doit donc contenir la réunion de ces deux fichiers, avec les mêmes champs.

   Écrire la fonction `fusion_2_tables` qui renvoie `table_fusion`, la table obtenue par fusion.

```
def fusion_2_tables(tab1, tab2):
    fusion = []
    ...
    return fusion

# tests
# table1 contient les données de 2005
# table2 contient les données de 2006.
table_fusion = fusion_2_tables(table1, table2)
print(table_fusion)
```

2. La réunion des tables à la question précédente donne encore des résultats séparés par année. On souhaite créer un fichier CSV qui ferait le total pour chaque prénom, sur le modèle ci-dessous, où pour « Emma » on donne le nombre total de fois qu'il a été donné sur les années fusionnées.

```
prenom , nombre
Emma, 2758
Mathéo, 5291
Léa, 5033
François, 26
Damien, 56
```

(a) Donner la méthode algorithmique qui permet d'arriver à cette fusion avec somme.

(b) **Écrire** et tester la fonction `reunir_prenoms(tab)`, qui prend comme argument la table fusionnée obtenue à la question précédente, et qui renvoie `tab_somme`, avec le regroupement par prénom et la somme correspondante comme ci-dessus.

(c) Exporter `prenoms_reunis` au format CSV avec comme nom de fichiers `prenomsReunis.csv`.

```
def reunir_prenoms(tab):
        tab_somme = []

        return tab_somme

prenoms_reunis = reunir_prenom(table_fusion)
print(prenoms_reunis)
```

5 Tri dans une table

On suppose que l'on a importé dans données à partir d'un fichier CSV avec la méthode `DictReader`. Les données ont donc la forme d'une liste de dictionnaires, comme par exemple :

```
t1 = [   { 'prenom' : 'Denis', 'nombre' : 5 } ,
         { 'prenom' : 'Fabien', 'nombre' : 3 } ,
         { 'prenom' : 'Emma', 'nombre' : 203 } ,
         { 'prenom' : 'Mathéo', 'nombre' : 18 } ,
      ]
```

Notion de tri

On appelle **tri** le fait de ranger les données d'une table selon un champ, dans l'ordre croissant ou décroissant.

Écrire la fonction **triTable(table, champ)** qui prend comme argument une `table` de donnée sous forme de liste de dictionnaires, ainsi que `champ` qui est le nom d'un des descripteurs de la table, et renvoie LA table rangée selon l'ordre **croissant** de ce descripteur.

Vous pourrez utiliser comme méthode le tri par **insertion**.

```
def triTableInsertion(liste, champ):
      # initialisation
      liste_inter = []
      liste_triee = []

      # traitement

      # sortie
      return liste_triee
```

Tests...
la commande :

```
print(triTableInsertion(t1, 'nombre'))
```

doit renvoyer :

```
[
        {'prenom': 'Fabien', 'nombre': 3},
        {'prenom': 'Denis', 'nombre': 5},
        {'prenom': 'Mathéo', 'nombre': 18},
        {'prenom': 'Emma', 'nombre': 203}
]
```

Enfin vous pourrez tester la commande :

```
print(triTableInsertion(t1, 'prenom'))
```