CH5 – TDT

Traitement de données en table

Cours / TD

PARTIE 1 : Le format CSV

1. Définition du format CSV

DÉFINITION 1 : Le format CSV (pour Comma Separated Values, soit en français valeurs séparées par des virgules) est un format très utilisé pour représenter des données structurées, notamment pour importer ou exporter des données à partir d'une feuille de calculs d'un tableur. C'est un fichier texte dans lequel chaque ligne correspond à une ligne du tableau

✔ EXEMPLE 1 : un exemple de feuille de calcul

	A	B	C	D	E	F	G	H
1	Nom	Prénom	NSI	Physique	Maths			
2	Bertrand	Paul	18	16	15			
3	Tranchant	Tom	1	3	5			
4	Gauchard	Zoé	14	13	16			
5								
6								
7								

Exemple d'enregistrement

Exemple de champ

Et voici le fichier CSV correspondant :

```
Nom,Prénom,NSI,Physique,Maths
Bertrand,Paul,18,16,15
Tranchant,Tom,1,3,5
Gauchard,Zoé,14,13,16
```

Dans ce format, chaque ligne représente un enregistrement c'est à dire une structure de données, de types éventuellement différents auxquelles on accède grâce à un nom et, sur une même ligne, les différents champs de l'enregistrement sont réparés par une virgule (d'où le nom).

En python en enregistrement peut être représenté par un dictionnaire :

```
{'Nom':'Baron','Prénom':'Paul','NSI':'18','Physique ':'16','Maths':'15'}
```

Et un fichier csv par une liste de dictionnaires, dont les clés sont les noms des colonnes :

```
ma_table= \
   [{'Nom':'Baron','Prénom':'Paul','NSI':'18','Physique ':'16','Maths':'15'},
    {'Nom':'Taillant','Prénom':'Greg','NSI':'1','Physique':'3','Maths':'5'},
    {'Nom':'Gourdy','Prénom':'Zoé','NSI':'14','Physique ':'13','Maths':'16'}]
```

2. Importation d'un fichier CSV

L'objectif de cette partie est d'apprendre à importer un fichier csv en Python. C'est à dire à lire un fichier csv en vue de le manipuler.

PROPRIÉTÉ 1 : Importation d'un fichier CSV

La bibliothèque csv implémente des classes pour lire des données tabulaires au format CSV.

La fonction reader() du module csv renvoie un objet de type csv.reader qui est itérable. Chaque élément de cet objet est une liste.

La fonction DictReader du module csv renvoie un objet de type csv.DictReader itérable. Chaque élément de cet objet est un dictionnaire.

✔ EXEMPLE 2 : Utilisation de la fonction Reader

```
import csv                  # le module pour les fichiers csv
file = open("test.csv" , "r")   # ouvrir le fichier

# initialisation d un lecteur de fichier delimiter est facultatif
csv_en_liste = csv.reader(file , delimiter = ",")

for ligne in csv_en_liste :    # parcours du lecteur avec une boucle
        print(ligne)               # affichage ligne à ligne

file.close ()              # fermeture du fichier
```

✔ EXEMPLE 3 : Utilisation de la fonction DictReader

```
import csv                  # le module pour les fichiers csv
file = open("test.csv" , "r")   # ouvrir le fichier

# initialisation d un lecteur de fichier avec creation automatique de dictionnaire
csv_en_dico = csv.DictReader(file , delimiter = ",")

for ligne in csv_en_dico :     # parcours du lecteur avec une boucle
  print(dict(ligne))        # affichage ligne a ligne

file.close ()              # fermeture du fichier
```

EXERCICE 1 : Quelle est la différence entre ce qu'on obtient avec la fonction Reader et la fonction DictReader ?

✍ EXERCICE 2 : L'objectif de cet exercice est l'écriture d'une fonction csv_list_de_list(nom) où nom est une chaine de caractère qui renvoie une liste de la liste obtenue avec la fonction reader.
a) par extension
b) par compréhension
Testez votre fonction avec le fichier test.csv précédent.

✍ EXERCICE 3 : Écrire une fonction impor(nom) qui reçoit un paramètre nom de type chaine de caractère qui est le nom du fichier csv sans l'extension et qui retourne une liste de dictionnaire qui contient les informations du fichier csv.
Vous testerez votre fonction sur le fichier précédent.

PROPRIÉTÉ 2 : La commande with open(...) as ...
Il existe en Python une commande qui permet d'ouvrir un fichier csv et qui ferme le fichier à la fin de l'indentation.

```
with open(nom_du_fichier ,'r') as sortie
```

Ici on ouvre le fichier "nom_du_fichier" en lecture et on stocke cette ouverture dans la variable sortie.
On pourra remplacer le "r" par "w" pour écrire.

3. Export d'un fichier CSV

Maintenant nous savons ouvrir un fichier csv en Python. L'objectif de se chapitre est de comprendre comment on exporte un fichier csv à partir de Python.
Une situation classique serait l'ouverture d'un fichier csv avec Python(import) on obtient une table puis on la modifie et on enregistre les modifications dans un csv(export).
Voyons dans l'exemple suivant comment créer un fichier csv correspondant à une table donnée.

✔ EXEMPLE 3 : Conversion d'une table en fichier csv

La fonction suivante va permettre de créer un fichier csv portant le nom sortieCSV.csv correspondant à la table précédente.

```
import csv

table_exemple=[ {'nom': 'Dupont', 'prenom': 'Jean-Claude', 'age': '32'},
                {'nom': 'Duteil', 'prenom': 'Paul', 'age': '41'},
                {'nom': 'Claudon', 'prenom': 'Goery', 'age': '37'},
                {'nom': 'Tonton', 'prenom': 'Pierre', 'age': '54'},
                {'nom': 'Penard', 'prenom': 'Bob', 'age': '18'},
                {'nom': 'Herpoix', 'prenom': 'Stephane ', 'age': '55'},
                {'nom': 'Salicorne', 'prenom': 'Bruno', 'age': '15'},
                {'nom': 'Poiteau', 'prenom': 'Maxe', 'age': '33'},
                {'nom': 'Clanget', 'prenom': 'Gilles', 'age': '54'},
                {'nom': 'Luillier', 'prenom': 'Martin', 'age': '34'},
                {'nom': 'Clanget', 'prenom': 'Justine', 'age': '14'},
                {'nom': 'Gillier', 'prenom': 'Paul', 'age': '16'}]

def vers_csv(nom_de_la_table ,ordre , fichier_sortie_csv):
    # nom_de_la_table: chaine de caracteres --> table a utiliser
    # ordre: liste contenant l ordre dans lequel les valeurs doivent etre affichees
    # fichier_sortie_csv est le nom du fichier a creer

    table=eval(nom_de_la_table) # evaluation du nom de la table

    # Ouverture du fichier en ecriture
    with open( fichier_sortie_csv+'.csv',"w",newline='') as fic:
        # initialisation d un lecteur de fichier
        dic=csv.DictWriter(fic ,fieldnames=ordre)
        # Ecriture des cles sur la premiere ligne
        dic.writeheader()
        # Ecriture des differentes lignes
        for ligne in table:
            dic.writerow(ligne)
    return None

ordre = ['nom','age','prenom'] # Definition de l ordre des cles
vers_csv('table_exemple',ordre ,'sortieCSV') # Appel fonction
```

✍ EXERCICE 4 : L'objectif de cet exercice est d'obtenir un outil qui permet d'insérer un nouvel enregistrement dans un fichier csv.

a) Ouvrir le fichier csv fichier.csv . Repérer le nom et l'ordre des clé.

b) Écrire une fonction ajout(fichier) qui va proposer à l'utilisateur(input) l'implémentation d'un nouvel enregistrement.

c) Tester cette fonction avec Danlta Alphonse qui a eu respectivement 15 16 et 11

```
import csv

# creation d'une liste de dictionnaire a partir d »un fichier csv (exo3)
def impor(nom):
…
…
…
…
…
…
…
def ajout(fichier):
        # creation d'une liste de dictionnaire avec l'appel de la fonction import
        ….
        # saisie par l'utilisateur du Nom, Prenom, et des notes NSI, Physique, Math dans trois fenêtres successives
                #(exemple : lieu=input('saisir le lieu')
        …
        …
        …
        …
        …

        # Ajout des entrees utilisateur dans la liste de dictionnaire
        …

        # Definition de l ordre
        ordre = …

        # Ecriture dans le fichier csv
        …                                                    # Ouverture du fichier a ecrire
        …                                                    # initialisation d'un lecteur de fichier
        …                                                    # Ecriture des cles sur la premiere ligne
        …                                                    # Ecriture des differentes lignes
```

PARTIE 2 : Opérations sur les tables

1. Sélectionner des enregistrements suivant certains critères

✔ EXEMPLE 1 : Ici nous utiliseront le principe des listes par compréhension où nous ajouterons une comparaison

Lors de la création de notre liste par compréhension, on ajoute seulement les enregistrements dont le champ nom correspond à Clanget

```
resultat= impor('test')
select_nom = [p for p in resultat if p['nom'] == 'Clanget']
print (select_nom)
```

```
*** Console de processus distant Réinitialisée ***
[{'age': '54', 'prenom': 'Gilles', 'nom': 'Clanget'}, {'age': '14', 'prenom': 'Justine', 'nom': 'Clanget'}]
```

✍ EXERCICE 1 :

a) Écrire une fonction un_critere(CSV,champ,valeur) qui extrait les enregistrements dont le champ champ est valeur.

Tester cette fonction avec un_critère('test','nom','Clanget')

```
*** Console de processus distant Réinitialisée ***
[{'nom': 'Clanget', 'age': '54', 'prenom': 'Gilles'}, {'nom': 'Clanget', 'age': '14', 'prenom': 'Justine'}]
```

b) Écrire une fonction criteres_ou(CSV,champ1,valeur1,champ2,valeur2) qui extrait les enregistrements dont le champ champ1 est valeur1 ou dont le champ champ2 est valeur2.

Tester cette fonction avec criteres_ou('test','nom','Clanget','prenom','Paul')

```
*** Console de processus distant Réinitialisée ***
[{'age': '41', 'nom': 'Duteil', 'prenom': 'Paul'}, {'age': '54', 'nom': 'Clanget', 'prenom': 'Gilles'}, {'age': '14', 'nom': 'Clanget', 'prenom': 'Justine'}, {'age': '16', 'nom': 'Gillier', 'prenom': 'Paul'}]
```

c) Écrire une fonction criteres_et_non(CSV,champ1,valeur1,champ2,valeur2) qui extrait les enregistrements dont le champ champ1 est valeur1 et qui ne contient pas dans le second champ champ2 la valeur2.

Tester cette fonction avec criteres_et_non('test','nom','Clanget','prenom','Justine')

```
*** Console de processus distant Réinitialisée ***
[{'prenom': 'Gilles', 'age': '54', 'nom': 'Clanget'}]
```

d) Écrire une fonction criteres_ou_non(CSV,champ1,valeur1,champ2,valeur2) qui extrait les enregistrements dont le champ champ1 est valeur1 ou qui ne contient pas dans le second champ champ2 la valeur2.

Tester cette fonction avec criteres_ou_non('test','nom','Clanget','prenom','Justine')

```
*** Console de processus distant Réinitialisée ***
[{'age': '32', 'nom': 'Dupont', 'prenom': 'Jean-Claude'}, {'age': '41', 'nom': 'Duteil', 'prenom': 'Paul'}, {'age': '37', 'nom': 'Claudon', 'prenom': 'Goery'}, {'age': None, 'nom': 'Tonton', 'prenom': 'Pierre54'}, {'age': '18',
'nom': 'Penard', 'prenom': 'Bob'}, {'age': '55', 'nom': 'Herpoix', 'prenom': 'Stephane'}, {'age': '15', 'nom': 'Salicorne', 'prenom': 'Bruno'}, {'age': '33', 'nom': 'Poiteau', 'prenom': 'Maxe'}, {'age': '54', 'nom': 'Clanget',
'prenom': 'Gilles'}, {'age': '34', 'nom': 'Luillier', 'prenom': 'Martin'}, {'age': '14', 'nom': 'Clanget', 'prenom': 'Justine'}, {'age': '16', 'nom': 'Gillier', 'prenom': 'Paul'}]
>>>
```

e) Écrire une fonction un_critere_depasse(CSV,champ,valeur_a_depasser) qui extrait les enregistrements dont le champ « champ » a une valeur qui dépasse valeur_a_depasser.

Tester cette fonction avec un_critere_depasse('test','age',50)

```
*** Console de processus distant Réinitialisée ***
[{'prenom': 'Pierre', 'age': '54', 'nom': 'Tonton'}, {'prenom': 'Stephane', 'age': '55', 'nom': 'Herpoix'}, {'prenom': 'Gilles', 'age': '54', 'nom': 'Clanget'}]
```

2. Sélectionner des colonnes

✔ EXEMPLE 2 : Ici nous utiliseront toujours le principe des listes par compréhension mais l'exemple est plus complexe, concentrez-vous sur les lignes qui suivent

Lors de la création de notre liste par compréhension, on ajoute seulement les champs nom et age de chaque enregistrement.

```
import csv

def impor(nom):
    file = open(nom+'.csv',"r")   # ouvrir le fichier nom.csv en lecture

# initialisation d un lecteur de fichier avec creation automatique de dictionnaire
    csv_en_dico = csv.DictReader(file , delimiter = ",")

    list_dico = []
    for ligne in csv_en_dico : # parcours du lecteur avec une boucle
        list_dico.append(dict(ligne))
    file.close () # fermeture du fichier
    return(list_dico)

#########################################################################
resultat=impor('test')
# Selection des colonnes qui ont pour cle 'nom' ou 'age'
select_colonne= [{cle:ligne[cle] for cle in ligne if cle in ['nom','age']} for ligne in resultat]
print (select_colonne)
```

✍ EXERCICE 2 : Écrire une fonction projection qui reçoit en paramètres une table et une liste d'attributs, puis qui retourne uniquement ces attributs.

```
import csv

def impor(nom):
    file = open(nom+'.csv',"r")                # ouvrir le fichier nom.csv en lecture

# initialisation d un lecteur de fichier avec creation automatique de dictionnaire
    csv_en_dico = csv.DictReader(file , delimiter = ",")
    list_dico = []
    for ligne in csv_en_dico : # parcours du lecteur avec une boucle
        list_dico.append(dict(ligne))
    file.close () # fermeture du fichier
    return(list_dico)

table_exemple=impor('test')
liste_attributs = ['prenom','age']

def projection (table:list,liste_attributs=list):
    # a completer
    …
    return resultat

resultat = projection(table_exemple , liste_attributs)
print(resultat)
```

```
*** Console de processus distant Réinitialisée ***
[{'prenom': 'Jean-Claude', 'age': '32'}, {'prenom': 'Paul', 'age': '41'}, {'prenom': 'Goery', 'age': '37'}, {'prenom': 'Pierre', 'age': '54'}, {'prenom': 'Bob', 'age': '18'}, {'prenom': 'Stephane', 'age': '55'
}, {'prenom': 'Bruno', 'age': '15'}, {'prenom': 'Maxe', 'age': '33'}, {'prenom': 'Gilles', 'age': '54'}, {'prenom': 'Martin', 'age': '34'}, {'prenom': 'Justine', 'age': '14'}, {'prenom': 'Paul', 'age': '16'}]
```

3. Tri d'une table sur une colonne

PROPRIÉTÉ 1 :
Une table étant représentée par une liste, on peut la trier en utilisant la fonction sorted ou la méthode .sort(), avec l'argument supplémentaire key qui est une fonction renvoyant la valeur utilisée pour le tri.

Rappel : la méthode .sort() trie la liste en place, alors que la fonction sorted() renvoie une nouvelle liste correspondant la liste triée, la liste initiale étant laissée intacte.

Pour la suite nous utiliserons la fonction sorted()

On peut trier les chaînes de caractères selon différents critères : ordre lexicographique : ['aaa', 'bb'] longueur : ['bb', 'aaa']

✔ EXEMPLE 3 : L'attribut key, nous permet de choisir le critère de tri, ici len pour longueur

```
print(sorted(['bb', 'aaa'])) # ordre lexicographique par défaut ['aaa ', 'bb ']
print(sorted(['aaa', 'bb'], key=len)) # longueur ['bb', 'aaa ']
```

PROPRIÉTÉ 2 :
Une fonction lambda est une fonction anonyme, c'est en quelque sorte une mini-fonction d'une ligne. On la note ainsi : **lambda entree : sortie**

✔ EXEMPLE 4 : Afin de bien comprendre la structure d'une fonction lambda, testez les deux codes ci-dessous dans une console pour x=4

```
def f(x):
    return x*2

g = lambda x: x*2

print(f(3))
print(g(3))
```

✔ EXEMPLE 5 : Dans l'exemple ci-dessous, nous allons trier la table par note de NSI, le nom de la variable **lyceen** n'a pas d'importance

```
ma_table =[{'Nom':'Baron','Prenom':'Paul','NSI':'18','Physique ':'16','Maths':'15'},
{'Nom':'Taillant','Prenom':'Greg','NSI':'1','Physique':'3','Maths':'5'},
{'Nom':'Gourdy','Prenom':'Zoé','NSI':'14','Physique ':'13','Maths':'16'}]

table_triee=sorted(ma_table, key=lambda lyceen: lyceen["NSI"])

print (table_triee)
```

✍ EXERCICE 3 :

a) A l'aide de la table que vous avez générée à partir du fichier test.csv, trier cette dernière par âge croissant, puis enregistrer dans un fichier table_age_croi.csv

```
nom,prenom,age
Clanget,Justine,14
Salicorne,Bruno,15
Gillier,Paul,16
Penard,Bob,18
Dupont,Jean-Claude,32
Poiteau,Maxe,33
Luillier,Martin,34
Claudon,Goery,37
Duteil,Paul,41
Tonton,Pierre,54
Clanget,Gilles,54
Herpoix,Stephane,55
```

b) A l'aide de la table que vous avez générée à partir du fichier test.csv, trier cette dernière par âge décroissant, puis enregistrer dans un fichier table_age_decroi.csv

```
nom,prenom,age
Herpoix,Stephane,55
Tonton,Pierre,54
Clanget,Gilles,54
Duteil,Paul,41
Claudon,Goery,37
Luillier,Martin,34
Poiteau,Maxe,33
Dupont,Jean-Claude,32
Penard,Bob,18
Gillier,Paul,16
Salicorne,Bruno,15
Clanget,Justine,14
```

✍ EXERCICE 4 : Écrire une fonction tri(table,attribut,decroit=False) qui trie une table selon un attribut de manière croissante dans le cas général et de manière décroissante s'il on écrit True comme troisième argument.

```
import csv

def impor(nom):
    file = open(nom+'.csv',"r")   # ouvrir le fichier nom.csv en lecture

# initialisation d un lecteur de fichier avec creation automatique de dictionnaire
    csv_en_dico = csv.DictReader(file , delimiter = ",")

    list_dico = []
    for ligne in csv_en_dico : # parcours du lecteur avec une boucle
        list_dico.append(dict(ligne))
    file.close () # fermeture du fichier
    return(list_dico)

def tri(table,attribut,decroit=False):
            ...
            ...
            ...


table_triee_crois = tri('test','age') # Tri de la table selon attribut croissant
print(table_triee_crois)
table_triee_decroi = tri('test','age',True) # Tri de la table selon attribut decroissant
print(table_triee_decroi)
```

4. Jointure de tables

DÉFINITION 1 : Jointure de deux tables selon un champ

La jointure de deux tables selon un champ est une table contenant les enregistrements des valeurs communes correspondant au champ de deux tables.

✔ EXEMPLE 6 : On considère deux tables :

Nom	Maths	Anglais	Info
Joe	16	17	18
Zoé	19	15	17
Max	14	19	13

Nom	Âge	Courriel
Joe	16	joe@info.fr
Zoé	15	zoe@info.fr

La jointure de ces deux tables est :

Nom	Âge	Courriel	Maths	Info	Anglais
Joe	16	joe@info.fr	16	18	17
Zoé	15	zoe@info.fr	19	17	15

PROPRIÉTÉ 2 : Fonction fusion

Voilà un script de la fonction fusion qui réalise la jointure de deux tables selon la clé **cle**

```
def fusion(table1,table2,cle):
    table_fusion=[]
    for ligne1 in table1:                    # Balayage des lignes de table1
        for ligne2 in table2:                # Balayage des lignes de table2
            if ligne1[cle]==ligne2[cle]:     # Si la cle correspond
                new_ligne=ligne1             # Recuperation de la ligne de table1
                for key in ligne2:           # Balayage des cles de table2
                    if key!=cle:                        # A la recherche d une nouvelle cle pour la fusion
                        new_ligne[key]=ligne2[key]      # Ajout de la cle et valeur table_fusion.append(new_ligne)
                table_fusion.append(new_ligne)          # append de la nouvelle ligne
    return table_fusion
```

EXERCICE 5 : Tester ce programme sur les deux tables suivantes :

```
table1 = [ {'Nom': 'Joe','Maths' : 16,'Anglais':17,'Info':18} ,
{'Nom': 'Zoe','Maths' : 19,'Anglais':15,'Info':17} ,
{'Nom': 'Max','Maths' : 14,'Anglais':19,'Info':13}]

table2 = [ {'Nom': 'Joe','Age' : 16,'courriel ':'joe@info.fr'},
{'Nom': 'Zoe','Age' : 15,'courriel ':'zoe@info.fr'},
{'Nom': 'teo','Age' : 16,'courriel ':'teo@info.fr'},
{'Nom': 'Max','Age' : 17,'courriel ':'max@info.fr'}]

def fusion(table1,table2,cle):
    table_fusion=[]
    for ligne1 in table1:                    # Balayage des lignes de table1
        for ligne2 in table2:                # Balayage des lignes de table2
            if ligne1[cle]==ligne2[cle]:     # Si la cle correspond
                new_ligne=ligne1             # Recuperation de la ligne de table1
                for key in ligne2:           # Balayage des cles de table2
                    if key!=cle:             # A la recherche d une nouvelle cle pour la fusion
                        new_ligne[key]=ligne2[key]  # Ajout de la cle et valeur table_fusion.append(new_ligne)
                table_fusion.append(new_ligne)      # append de la nouvelle ligne
    return table_fusion

table_fusionnee = fusion(table1 ,table2 ,'Nom')
```

✍ EXERCICE 6 : Écrire une fonction fusion2(table1,table2,cle1,cle2=None) où cle1 n'est pas forcément dans table2 mais serait équivalente à cle2. Par exemple name et nom (cle2=None dans le cas où il n'y a pas de clé équivalente).

```
table1 = [ {'Nom': 'Joe','Maths' : 16,'Anglais':17,'Info':18} ,
{'Nom': 'Zoe','Maths' : 19,'Anglais':15,'Info':17} ,
{'Nom': 'Max','Maths' : 14,'Anglais':19,'Info':13}]

table2 = [ {'name': 'Joe','Age' : 16,'courriel ':'joe@info.fr'},
{'name': 'Zoe','Age' : 15,'courriel ':'zoe@info.fr'},
{'name': 'teo','Age' : 16,'courriel ':'teo@info.fr'},
{'name': 'Max','Age' : 17,'courriel ':'max@info.fr'}]
def fusion2(table1 ,table2,cle1,cle2=None):
            …       # teste la presence de cle2
            …
   table_fusion=[]
   for ligne1 in table1:                # Balayage des lignes de table1
      for ligne2 in table2:             # Balayage des lignes de table2
                                 # à compléter (6 lignes)



   return table_fusion

table_fusionnee2 = fusion2(table1 ,table2 ,'Nom','name')
print(table_fusionnee2)
```