Math93.com

# Traitement données en tables

# Opérations sur les tables
# TD 3 - NSI

---

**Objectifs**

- Recherche dans une table.
  *Rechercher les lignes d'une table vérifiant des critères exprimés en logique propositionnelle.*
- Tri d'une table.
  *Trier une table suivant une colonne.*
- Jointures ou fusions de tables.
  *Construire une nouvelle table en combinant les données de deux tables.*

Nous avons dans le TD 1 appris à lire dans des fichiers CSV et écrire dans ces fichiers.
L'idée ici est d'apprendre à extraire des données cibles, les manipuler, les trier ou faire des statistiques. Ces opérations sont nommées **requêtes**.

## I Opérations sur les tables.

Faire les exercices sur Capytale: Opération sur les tables.

### I.1 Importation de la table depuis un fichier CSV.

- On cherche à créer une nouvelle table en extrayant d'une table, les lignes satisfaisant une condition exprimée sous la forme d'une **fonction booléenne**.
- Par exemple on reprend l'exemple du TD 2.

| Nom | Maths | NSI | Anglais |
|---|---|---|---|
| Galois | 17 | 14 | 17 |
| Gauss | 20 | 12 | 8 |
| Euler | 19 | 15 | 13 |

Voici le code qui nous permet d'importer les données à partir du fichier CSV.

```
# Dans l'éditeur PYTHON
import csv
#
fichier=open("NotesEleves.csv", encoding='utf8')
table=list(csv.DictReader(fichier,delimiter=","))
print(table)
fichier.close()

# On obtient alors :
[{'Nom': 'Galois', 'Maths': '17', 'NSI': '14', 'Anglais': '17'},
{'Nom': 'Gauss', 'Maths': '20', 'NSI': '12', 'Anglais': '8 '},
{'Nom': 'Euler', 'Maths': '19', 'NSI': '15', 'Anglais': '13'}]
# Vous pouvez aussi utiliser la table_valide créée lors du TD 1.
```

## I.2 Projection

**Définition 1** (projection)

**Une projection** est la sélection d'un ou de plusieurs **attributs (colonnes)** d'une table.

On cherche par exemple à sélectionner les **noms des élèves**, on va donc projeter sur une colonne :

| Nom | Maths | NSI | Anglais |
|---|---|---|---|
| Galois | 17 | 14 | 17 |
| Gauss | 20 | 12 | 8 |
| Euler | 19 | 15 | 13 |

La rédaction sera :

```
# Dans l'éditeur
t1=[]
for ligne in table:
    t1.append(ligne["Nom"] )
print(t1)
```

```
# On obtiendra dans la console :
['Galois', 'Gauss', 'Euler']
```

## I.3 Projection et sélection

On cherche à sélectionner **les élèves ayant plus de 10 en anglais**.
La sélection sur une colonne peut être combinée à la sélection d'un ensemble de lignes.
La rédaction sera :

```
# Dans l'éditeur
# Attention il faut convertir les chaînes en flottants pour le test
t2=[]
for ligne in table:
    if float(ligne["Anglais"])>10:
        t2.append(ligne["Nom"] )
print(t2)
```

```
# On obtiendra dans la console :
['Galois', 'Euler']
```

Remarquer que le code précédent est équivalent à :

```
# Dans l'éditeur
# Attention il faut convertir les chaînes en flottants pour le test
t2=[ligne["Nom"] for ligne in table
                 if float(ligne["Anglais"])>10]
print(t2)
```

Faire les exercices sur Capytale: Opération sur les tables.

**Exercice 1**

En utilisant les méthodes de projection et sélection :

1. construire un tableau T3 contenant les noms des élèves ayant plus de 18 en Maths **et** plus de 13 en NSI ;
2. construire un tableau T4 contenant les noms des élèves ayant plus de 10 en Anglais **ou** plus de 13 en NSI.

### I.4 Construction d'une nouvelle table

— **L'opération de projection** est notamment utilisée pour **construire un nouvelle table** contenant seulement **une partie des colonnes** de la table d'origine. Pour cela, plutôt que d'extraire seulement la valeur d'un attribut, on peut construire un nouveau dictionnaire contenant les valeurs des attributs qui nous intéressent. Par exemple ici si on ne veut garder que les noms et les notes de Maths on écrira l'instruction suivante :

```
# Dans l'éditeur Python
i4a=[]
for ligne in table:
    dico={"Nom":ligne["Nom"],"Maths":float(ligne["Maths"])}
    i4a.append(dico)

#Equivalent à
i4a= [ {"Nom":ligne["Nom"],"Maths":float(ligne["Maths"])}
              for ligne in table ]
```

```
# On obtient dans la console
[{'Nom': 'Galois', 'Maths': 17.0}, {'Nom': 'Gauss', 'Maths': 20.0},
{'Nom': 'Euler', 'Maths': 19.0}]
```

— Plus généralement on peut au passage effectuer des opérations sur les lignes au moment de faire une sélection ou une projection.
Par exemple, si on veut ajouter 10% aux notes d'anglais on obtient :

```
# Dans l'éditeur Python
i4b=[]
for ligne in table:
    newNote=float(ligne["Anglais"])*1.1
    dico={"Nom":ligne["Nom"],"Anglais":newNote}
    i4b.append(dico)

#Equivalent à

i4b= [ {"Nom":ligne["Nom"],"Anglais":float(ligne["Anglais"])*1.1}
              for ligne in table ]
```

```
# On obtient dans la console
[{'Nom': 'Galois', 'Anglais': 18.700000000000003},
{'Nom': 'Gauss', 'Anglais': 8.8},
{'Nom': 'Euler', 'Anglais': 14.3}]
```

## Exercice 2

En utilisant les méthodes de projection et sélection :

1. construire un tableau T5 contenant les noms des élèves et les notes en Maths augmentées de 1 points, sans dépasser 20 ;

**Aide**

$min(20+1,20)=20$

2. construire un tableau T6 contenant les noms des élèves et une nouvelle clé nommée "Moyenne" de valeur la moyenne des 3 notes.

# II Trier une table

## II.1 La fonction sorted(tableau)

**sorted(tableau, key=None, reverse=False)**

En Python, la fonction **sorted(tableau)** permet de trier un tableau.

— Cette fonction dispose d'un argument optionnel nommé **key** permettant de précisez selon quel critère une liste doit être triée.

La valeur du paramètre key devrait être **une fonction** qui prend un seul argument et renvoie une clef à utiliser à des fins de tri. Cette technique est rapide car la fonction clef est appelée exactement une seule fois pour chaque enregistrement en entrée.

— Un troisième argument optionnel **reverse** (un booléen) permet de préciser si on veut le résultat par ordre croissant par défaut) ou décroissant (reverse=True).

*Remarque : ne confondez pas avec la procédure de tri des listes "**List.sort()** "qui va modifier la liste elle-même.*

```
# Dans l'éditeur Python
t=['c','a','14','2','1']
tri=sorted(t)
print(tri) # tri de caractères
print(t) # t n'est pas modifié
```

```
# On obtient dans la console
['1', '14', '2', 'a', 'c']
['c', 'a', '14', '2', '1']
```

## II.2 Trier des données en fonction d'une clé

On rappelle la construction de notre table.

```
[{'Nom': 'Galois', 'Maths': '17', 'NSI': '14', 'Anglais': '17'},
{'Nom': 'Gauss', 'Maths': '20', 'NSI': '12', 'Anglais': '8 '},
{'Nom': 'Euler', 'Maths': '19', 'NSI': '15', 'Anglais': '13'}]
```

On voudrait trier la table selon les noms par ordre alphabétique.

1. Pour cela il faut créer une fonction qui va renvoyer la valeur de la **clé 'Nom'** de chaque **dictionnaire** de la liste.
2. Dans **sorted()**, on va passer la fonction créée en paramètre **key**.

```
# Dans l'éditeur Python
# On va trier par rapport à la clé 'Nom' de la ligne qui est un dictionnaire
def nom(ligne):
    return ligne['Nom']

tri_nom=sorted(table,key=nom,reverse=False)
print(tri_nom)
```

```
# On obtient dans la console
[{'Nom': 'Euler', 'Maths': '19', 'NSI': '15', 'Anglais': '13'},
{'Nom': 'Galois', 'Maths': '17', 'NSI': '14', 'Anglais': '17'},
{'Nom': 'Gauss', 'Maths': '20', 'NSI': '12', 'Anglais': '8 '}]
```

**Propriété 1**

Pour résumer, on considère que la variable **table** est une liste de dictionnaires.
Alors la fonction $f$ associée à l'argument de tri **key** de la table sera de la forme :

```
# Fonction f associée à l'argument key pour la fonction sorted()
def f(ligne):
    return ligne[attribut] # attribut = clé du dictionnaire

Table_triee=sorted(table, key=f ,reverse=False)
```

## Exercice 3

1. Effectuer un tri selon les notes en Maths (ordre décroissant).
   *Attention, convertissez les notes en flottants dans la fonction sinon ...*
2. Effectuer un tri selon les notes en NSI (ordre décroissant).
3. Effectuer un tri selon les notes en Anglais (ordre décroissant).

# III Jointures (ou fusions) de tables

## III.1 Réunion de tables

Une première opération naturelle est de réunir dans une même table les données de deux (ou plusieurs) autres tables qui ont les **même structures** (même attributs, i.e. **noms des colonnes**).
L'opération est facile dans ce cas, il suffit d'utiliser l'opérateur + sur les deux tables.

On va par exemple utiliser la table précédente puis une table nouvelle table3.

| Nom | Maths | NSI | Anglais |
|---|---|---|---|
| Galois | 17 | 14 | 17 |
| Gauss | 20 | 12 | 8 |
| Euler | 19 | 15 | 13 |

**Table 1**

| Nom | Maths | NSI | Anglais |
|---|---|---|---|
| Abel | 15 | 13 | 16 |
| Oresme | 16 | 11 | 7 |
| Al-Kashi | 19 | 10 | 13 |

**Table 2**

| Nom | Maths | NSI | Anglais |
|---|---|---|---|
| Galois | 17 | 14 | 17 |
| Gauss | 20 | 12 | 8 |
| Euler | 19 | 15 | 13 |
| Abel | 15 | 13 | 16 |
| Oresme | 16 | 11 | 7 |
| Al-Kashi | 19 | 10 | 13 |

**Table 3**

On va donc ici :

1. Créer un fichier **NotesEleves2.csv** avec les données de la table2.
2. Importer les données de **NotesEleves.csv** et **NotesEleves2.csv** dans les variables table1 et table2.
3. Fusionner les deux dans une table3 par l'opération : table3 = table1 + table2.
4. Exporter les données de la table3 dans un nouveau fichier CSV : **NotesEleves3.csv**
5. Vérifier le résultat obtenu.

```
# Dans l'éditeur Python
# Etape 1 : création du fichier CSV NotesEleves2.csv
# Etape 2 : importation
fichier=open("NotesEleves.csv", encoding='utf8')
# souvent utile de préciser l'encodage
table1=list(csv.DictReader(fichier,delimiter=","))
# on peut aussi préciser le séparateur
fichier.close()
#
fichier=open("NotesEleves2.csv")
table2=list(csv.DictReader(fichier))
fichier.close()
# Etape 3 : Fusionner les deux dans une table3
table3=table1+table2
# Etape 4 : Exporter les données de la table3 dans NotesEleves3.csv
with open("NotesEleves3.csv","w") as sortie:
        objet=csv.DictWriter(sortie,['Nom','Maths','NSI','Anglais'])
        # Pour écrire la ligne d'en têtes
        objet.writeheader()
        # On peut donner la table directement
        objet.writerows(table3)
```

### III.2 Fusions (jointures) de deux tables

Nous pouvons effectuer des opérations plus délicates et considérer des tables ayant des attributs différents, mais **au moins un attribut commun**.
Par exemple :

| Nom | Maths | NSI | Anglais |
|---|---|---|---|
| Galois | 17 | 14 | 17 |
| Gauss | 20 | 12 | 8 |
| Euler | 19 | 15 | 13 |

**Table 1**

| Nom | Prénom | Année | e-mail |
|---|---|---|---|
| Galois | Evariste | 2001 | e.galois@yahoo.fr |
| Gauss | Carl | 2000 | c.gauss@yahoo.fr |
| Euler | Leonhard | 2005 | l.euler@yahoo.fr |

**Table 4**

L'idée est donc de créer une table qui fusionne les deux tables 1 et 4, pour les cas où le champ **"Nom"** est le même :

| Nom | Prénom | Année | e-mail | Maths | NSI | Anglais |
|---|---|---|---|---|---|---|
| Galois | Evariste | 2001 | e.galois@yahoo.fr | 17 | 14 | 17 |
| Gauss | Carl | 2000 | c.gauss@yahoo.fr | 20 | 12 | 8 |
| Euler | Leonhard | 2005 | l.euler@yahoo.fr | 19 | 15 | 13 |

**Table 5**

- Étape 1 : Créer un fichier **NotesEleves4.csv** avec les données de la table4.
- Étape 2 : Importer les données de **NotesEleves.csv** et **NotesEleves4.csv** dans les variables table1 et table4.
- Étape 3 : créer une fonction fusion(tableA,tableB) qui va créer un nouveau dictionnaire représentant le ligne de la table fusionnée recherchée.

```
# Dans l'éditeur PYTHON
# Etape 3
def fusion(ligneA,ligneB):
    '''In : ligneA la table 1 et ligneB de la table 4
       Out : le dictionnaire fusion'''
    return {"Nom":ligneA["Nom"],"Prénom":ligneB["Prénom"],
    "année":ligneB["Année"],"e-mail":ligneB["e-mail"],
    "Maths":ligneA["Maths"],"NSI":ligneA["NSI"],
    "Anglais":ligneA["Anglais"]}
```

- Étape 4 : on va créer la jointure de deux façons :
  - Étape 4a : par une double boucles ;
  - Étape 4b : et par une double compréhension qui s'apparente en fait plus au fonctionnement des requêtes SQL qui seront abordées en terminale.

```
# Dans l'éditeur PYTHON
# Etape 4a
jointure1=[]
for ligneA in table1:
    for ligneB in table4:
        if ligneA["Nom"]==ligneB["Nom"]:
            jointure1.append(fusion(ligneA,ligneB))
# Etape 4b
jointure2=[fusion(ligneA,ligneB)
                        for ligneA in table1 for ligneB in table4
                        if ligneA["Nom"]==ligneB["Nom"]]
```

— Étape 5 : Exporter les données de la jointure dans un nouveau fichier CSV : **jointure1-4.csv**

### III.3 Fusions (jointures) de deux tables : cas particuliers

| Nom | Maths | NSI | Anglais |
|---|---|---|---|
| Galois | 17 | 14 | 17 |
| Gauss | 20 | 12 | 8 |
| Euler | 19 | 15 | 13 |
| Pythagore | 20 | 5 | 10 |
| Descartes | 20 | 20 | 20 |

**Table 1**

| Nom | Prénom | Année | e-mail |
|---|---|---|---|
| Galois | Evariste | 2001 | e.galois@yahoo.fr |
| Gauss | Carl | 2000 | c.gauss@yahoo.fr |
| Euler | Leonhard | 2005 | l.euler@yahoo.fr |
| Descartes | René | 2004 | rene.des@yahoo.fr |
| Descartes | René | 2004 | descartes@bbox.fr |
| Turing | Alan | 2005 | alan.turing@yahoo.fr |

**Table 4**

L'idée est donc de créer une table qui fusionne les deux tables 1 et 4 pour les cas où le champ **"Nom"** est le même :

| Nom | Prénom | Année | e-mail | Maths | NSI | Anglais |
|---|---|---|---|---|---|---|
| Galois | Evariste | 2001 | e.galois@yahoo.fr | 17 | 14 | 17 |
| Gauss | Carl | 2000 | c.gauss@yahoo.fr | 20 | 12 | 8 |
| Euler | Leonhard | 2005 | l.euler@yahoo.fr | 19 | 15 | 13 |
| Descartes | René | 2004 | descartes@bbox.fr | 20 | 20 | 20 |
| Descartes | René | 2004 | rene.des@yahoo.fr | 20 | 20 | 20 |

**Table 5**

**On remarque que :**

- **Pythagore et Turing n'apparaissent pas dans la table fusionnée ;**
- **Descartes apparait deux fois.**

Faire les exercices sur Capytale: Opération sur les tables.

## IV Exercices

### Exercice 4

On considère les tables suivantes :

| nom | couleur 1 |
|---|---|
| pie | noire |
| aigle | brun |
| chouette | brun |

**Table Oiseaux 1**

| nom | couleur 1 |
|---|---|
| perruche | jaune |
| perroquet | bleu |

**Table Oiseaux 2**

| nom | couleur 2 |
|---|---|
| pie | blanc |
| aigle | noir |
| chouette | brun clair |
| perruche | vert |
| perroquet | rouge |

**Table couleur 2**

1. Pour chaque table, écrire le contenu d'un fichier CSV correspondant : **Oiseaux1.csv**, **Oiseaux2.csv** et **couleur2.csv**.
2. Construire à la main la réunion des tables **Oiseaux 1** et **Oiseaux 2**. On note cette table **Oiseaux 3**.
3. Calculer **à la main** la jointure des tables **Oiseaux 3** et **couleur 2**. On la nommera **bilan_oiseaux**.
4. Écrire un programme en Python qui charge les trois fichiers CSV **Oiseaux1.csv**, **Oiseaux2.csv** et **couleur2.csv** et qui réalise la réunion **Oiseaux 3** et la jointure **bilan_oiseaux** des questions précédentes.

## Exercice 5

On considère les tables Membre (idm est le numéro d'identification de la personne) , Prêt et Livre (idl est le numéro d'identification du livre) ci-dessous :

| prénom | idm |
|---|---|
| Bertrand | 12 |
| Lucie | 24 |
| Jean | 42 |
| Sara | 7 |

**Table Membre**

| idl | idm |
|---|---|
| 12 | 42 |
| 11 | 7 |
| 3 | 42 |
| 4 | 7 |
| 8 | 12 |

**Table Prêt**

| idl | titre |
|---|---|
| 1 | Fondation |
| 2 | Les Robots |
| 3 | Dune |
| 4 | La Stratégie Ender |
| 5 | 1984 |
| 6 | Santiago |
| 7 | Hypérion |
| 8 | Farenheit 451 |
| 9 | Les Monades urbaines |
| 10 | Star Wars - épisode IX |
| 11 | L'étoile et le fouet |
| 12 | Ubik |

**Table Livre**

1. Pour chaque table, écrire le contenu d'un fichier CSV correspondant. : **membre.csv**, **pret.csv** et **livre.csv**.
2. Construire à la main la jointure des tables **Membre** et **Prêt**. On la nomme **jointure 1**.
3. Construire à la main la jointure des tables **jointure 1** et **Livre**. On la nomme **jointure_finale**.
4. Écrire un programme en Python qui charge les trois fichiers **membre.csv**, **pret.csv** et **livre.csv** et qui construit les deux jointures **jointure 1** et **jointure_finale** des questions précédentes. On les sauvegardera dans un fichier csv.

# V Mini Projet

## Exercice 6

On trouve sur le site https://www.data.gouv.fr/ des fichiers CSV donnant les prénoms des enfants nés en France par année. Voici les listes des prénoms de ceux nés en 2003 et 2004.

— lien fichier 2003 : Prenoms2003.csv

— lien fichier 2004 : Prenoms2004.csv

**ATTENTION : quand un affichage est demandé, votre code doit fournir aussi une liste contenant les 10 couples (occurence,prenom) classés dans l'ordre décroissant du nombre d'occurence. Respectez rigoureusement le nom de liste demandé.**

1. Importer ces fichiers CSV via Python puis créer deux tables de dictionnaires comportant ces données : **Table_Prenoms2003** et **Table_Prenoms2004**.
2. Effectuer la réunion de ces deux tables dans un fichier CSV nommé : **Prenoms2003-2004.csv**
3. Donner la liste **prenomsFrequents2003** des 10 prénoms les plus fréquents avec les occurrences pour la période 2003.
   On proposera un affichage explicite de la forme :
   Prénom 2003 n°1 = _PRENOMS_RARES - Occurrence = 16870
4. Donner la liste **prenomsFrequents2004** des 10 prénoms les plus fréquents avec les occurrences pour la période 2004.
5. On cherche maintenant à réunir les données en créant une table des prénoms des enfants nés en 2003 ou 2004. On fera la somme des nombres pour chaque prénom.
   Créer une table de jointure avec somme des occurrences.
6. Donner les listes **prenomsFrequents20032004Garcon** et **prenomsFrequents20032004Fille** des 10 prénoms les plus fréquents pour la période 2003-2004, d'une part pour les filles, d'autre part pour les garçons.
7. Faire une étude globale pour les prénoms de votre année de naissance à partir du fichier csv qui regroupe tous les prénoms de 1900 à 2018 disponible sur :
   — www.math93.com
   — ou ici nom_naissance2018.csv

```
Prenom 2003 n 1 =  _PRENOMS_RARES  - Occurrence = 16870
Prenom 2003 n 2 =  _PRENOMS_RARES  - Occurrence = 15655
Prenom 2003 n 3 =  LEA  - Occurrence = 8986
Prenom 2003 n 4 =  LUCAS  - Occurrence = 8290
Prenom 2003 n 5 =  THEO  - Occurrence = 7858
Prenom 2003 n 6 =  HUGO  - Occurrence = 7388
Prenom 2003 n 7 =  MANON  - Occurrence = 6923
```

↩ **Fin du devoir** ↪