

Nous allons maintenant étudier comment filtrer des lignes et des colonnes puis nous verrons comment trier une table en fonction d'une colonne.



— Exercice 7 —

Proposez une fonction `filtrer_ligne` qui prend en argument un tableau, un critère (Nom, Science ...) ainsi qu'une valeur et renvoie une liste de dictionnaire filtrée. Seules les lignes correspondantes à votre critère d'égalité (par simplicité) doivent apparaître.

```
def filtrer_ligne(données : list, critere : str, valeur : str):  
    ...  
    return tableau_filtré
```

Si vous avez fini avant : Faites la même chose en 1 ligne !



Filtrer suivant une colonne s'appelle aussi une projection.



— Exercice 8 —

On suppose pour l'instant que l'on cherche à faire notre projection sur un seul critère (le 'Nom' par exemple).

- ❖ Compléter la fonction `filtrer_colonne` ci-dessous. Celle-ci prend en argument un tableau et un critère (Nom, Science ...) et renvoie un tableau filtré. Seules les colonnes correspondantes à votre critère doivent apparaître.

```
def filtrer_colonne(données : list, liste_criteres : str):  
    tableau_filtré = []  
    for dico in données:  
        dico_filtré = {}  
        for ..... in ..... :  
            ..... # ici  
        tableau_filtré.append(dico_filtré)  
    return tableau_filtré
```

- ❖ Testez votre code grâce à un print : `print(filtrer_colonne(données, 'Nom'))` .

Conserver une seule colonne est assez inutile... On suppose à présent que l'utilisateur peut donner une **liste** de critères. `liste_criteres` sera donc de type list.

- ❖ Modifier le programme précédent afin de prendre en compte une sélection sur une liste de critères.
Les modifications ne concerneront que la boucle interne indiquée par # ici

Si vous avez fini avant : Faites la même chose en 1 ligne !



Finalement, le tri d'une table suivant une colonne peut se faire grâce aux algorithmes de tri.

Dans le cadre de ce chapitre, nous allons utiliser l'instruction **sorted** propre à Python. **sorted** possède un argument très pratique appelé **key** qui permet de préciser selon quel critère une liste doit être triée (cela est une fonction de variables à trier). Un autre argument d'intérêt est **reverse** qui est un booléen permettant d'indiquer si on souhaite que l'ordre soit croissant (False) ou décroissant (True).



— Exercice 9 —

❖ Recopiez le code ci-dessous :

```
def tri_table(données: list, critere: str, décroissant = False ):
    return sorted(données, key = lambda table: table[critere],
reverse = décroissant)
```

❖ Testez votre code en lançant tri_table sur votre table et en classant par 'Science'. Cela fonctionne-t-il ?

On a l'impression que le tableau est correctement trié : sorted doit regarder ma liste et la trier suivant le critère ['Science']. C'est bon, on a tout compris.

Non, on n'a rien compris. C'est même le drame car un nouveau mot-clé est apparu : c'est le mot clé lambda .

"Lambda, c'est quoi ???"

Réponse : Lambda est appelé "fonction anonyme" (ou encore fonction poubelle).

Lorsque l'on a besoin d'une fonction pour une utilisation unique, à la volée, dans un code, il est souvent fastidieux de créer une fonction complète. C'est là que la lambda entre en scène. Par exemple, pour calculer un carré d'un nombre d'une manière ponctuelle :

Version classique

```
def carré(x: list):
    return str(x**2)
print('Carré de 4 = ' + carré(4))
```

Version lambda

```
carré = lambda x: str(x**2)
print('Carré de 4 = ' + carré(4))
```

```
print('le carré de 4 est ' + (lambda x: str(x**2))(4))
```

En résumé :

C'est exactement la même chose qu'une fonction, seule la syntaxe change:

- ❖ lambda au lieu de def ;
- ❖ pas de nom de fonction ;
- ❖ pas de parenthèses ;
- ❖ pas de mot clé return.

4) Fusion et jointure de table

Définition : La fusion de deux tables consiste à ajouter des enregistrements d'une table B à ceux d'une table A, ces deux tables ayant exactement la **même** structure.

Exemple :

On dispose de deux tables :

Nom	Francais	Science	Histoire
Erwann	16	12	15
Celine	14	16	13
Julien	14	17	15

Nom	Francais	Science	Histoire
Mélanie	11	11	17
Syrine	15	17	11

La fusion de ces deux tables (ayant la même structure) va nous donner une unique table. Vous pouvez aisément voir laquelle !



— Exercice 10 —

Fusion : cette étape est à faire en autonomie.

Nous allons travailler en nous basant sur une librairie que nous avons déjà créé (dans notre cas, csv_reader.py). Pour faire cela, placez-vous dans le même répertoire que celui où vous avez créé csv_reader.py .

- ❖ Créez un nouveau fichier appelé jointure.py et depuis csv_reader, importez la fonction import_CSV.
- ❖ Créez une fonction `verifie_clef` qui vérifie si les attributs de vos deux tables sont strictement égaux. `verifie_clef` prendra en argument d'entrée : deux tables `table1` et `table2` formatées de manière classique (tableau de dictionnaires). `verifie_clef` renverra `True` si les attributs des deux tables sont égaux et `False` sinon.
- ❖ Créez une fonction `fusion` qui effectue la fusion de vos deux tables si celles-ci ont la même structure et renvoie "Tables non fusionnables" sinon. `fusion` prendra en argument d'entrée deux tables `table1` et `table2` formatées de manière classique (tableau de dictionnaires). `fusion` renverra une table fusionnée `table_fusionnée`.
- ❖ Finalement, la table fusionnée sera exportée au format csv grâce à l'importation de la fonction `export_CSV` de notre bibliothèque `csv_reader`.

❖ On testera le code sur les fichiers `exemple.csv` et `exemple3.csv` à chaque étape de sa conception afin d'éviter d'être débordé(e) à la fin du développement. N'hésitez pas à modifier les fichiers csv sachant que ceux-ci sont de simples fichiers texte. Affichez la table fusionnée dans un tableur pour vérifier la correction de votre fusion.

Définition : La jointure de deux tables consiste à ajouter des enregistrements d'une table B à ceux d'une table A selon une clé commune aux deux tableaux.

Rem : lors d'une jointure, on ne va pas laisser de "cases vides". On va supprimer les éléments dont certaines informations sont manquantes.

Exemple :

On dispose de deux tables :

Nom	Francais	Science	Histoire
Erwann	16	12	15
Celine	14	16	13
Julien	14	17	15

Nom	Age	Courriel
Julien	16	julien@nsi.fr
Erwann	15,5	erwann@nsi.fr

Ces deux tables ont clairement une clé en commun. Notez que cela ne serait pas du tout une bonne clé primaire si on travaillait sur tout le lycée. La jointure de ces deux tables va nous donner une unique table :

Nom	Francais	Science	Histoire	Age	Courriel
Erwann	16	12	15	15,5	erwann@nsi.fr
Julien	14	17	15	16	julien@nsi.fr

On voit que l'information sur Céline est perdue sinon, on aurait une table avec des données manquantes.



— Exercice 11 —

Jointure : cette étape est un peu particulière.

Nous allons réutiliser le fichier `jointure.py` créé à l'étape 10. On testera le code sur les fichiers `exemple.csv` et `exemple_age.csv` à chaque étape de sa conception afin d'éviter d'être débordé(e) à la fin du développement. N'hésitez pas à modifier les fichiers csv sachant que ceux-ci sont de simples fichiers texte. Affichez la table fusionnée dans un tableur pour vérifier la correction de votre fusion.

- ❖ Dans l'exemple ci-dessus, identifiez les étapes importantes à effectuer pour réaliser la jointure de deux tables. On essaiera d'imaginer un algorithme simple que l'on fera tourner à la main.
- ❖ Créez une fonction `jointure` qui effectue la jointure de deux tables selon une clé donnée. Si la clé proposée n'est pas un descripteur du tableau, on renverra "Aucun descripteur NOM_DU_DESCRIPTEUR trouvé dans ces tables". `jointure` prendra en argument d'entrée deux tables `table1` et `table2` formatées de manière classique (liste de dictionnaires) ainsi qu'un descripteur de table. `jointure` renverra une table jointe `tableJointe`.
- ❖ Finalement, la table jointe sera exportée au format csv grâce à l'importation de la fonction `export_CSV` de notre bibliothèque `csv_reader`.