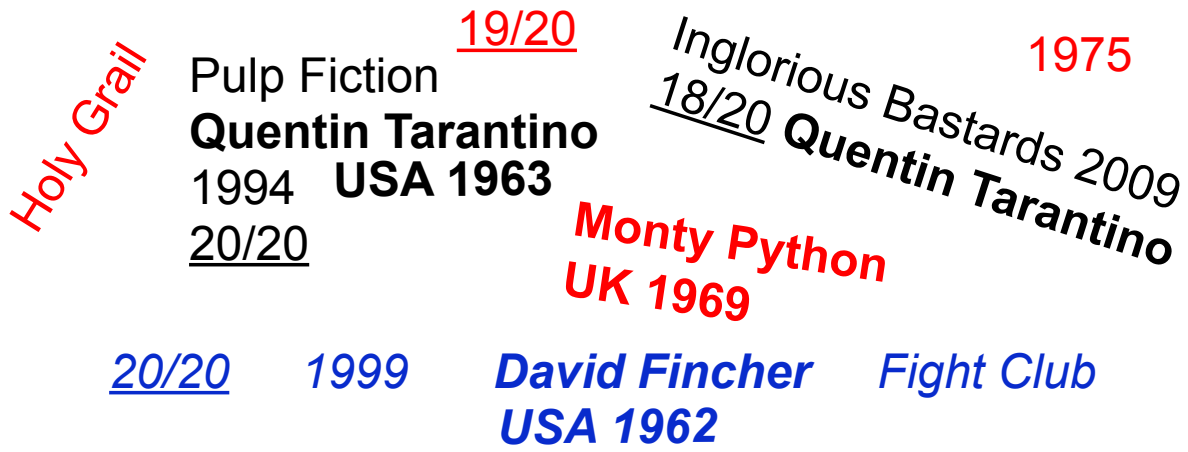
Chapitre 11 : Traitement de données

I. Introduction

1) Petit effort d'imagination et d'écriture

Imaginez que vous disposiez d'un certain nombre de données :



On vous demande d'organiser ces données de manière lisible afin de pouvoir les échanger avec une personne située à l'autre bout du monde.

Comment feriez-vous ? *Quelles informations enverriez-vous ? Comment organiseriez-vous ces informations ?*

2) Tableaux et bases de données

Il est clair que l'on va mettre ces données sous forme de tableau(x) avec des en-têtes claires et concises. De cette manière, n'importe qui pourra lire ces deux tableaux que je lui envoie. On a rajouté ici quelques données.

Clé	Titre	Réalisateur	Année de sortie	Note	Origine	Date de naissance
1	Pulp Fiction	Quentin Tarantino	1994	20	USA	1962
2	Inglorious Bastards	Quentin Tarantino	2009	18	USA	1962
3	Inglorious Bastards 2	Quentin Tarantino	2024	18	USA	1962
4	Kill Bill 1	Quentin Tarantino	2003	18	USA	1962
5	Kill Bill 2	Quentin Tarantino	2004	18	USA	1962
6	Kill Bill : Fury Road	Quentin Tarantino	2025	18	USA	1962
7	Holy Grail	Monty Python	1975	19	UK	1969
8	Fight Club	David Fincher	1999	20	USA	1963
9	Life of Brian	Monty Python	1979	17	UK	1969

Question : Quel est le problème de ce tableau ? Pourriez-vous résoudre ce problème ?

On va essayer de proposer un format **standardisé** contenant le moins de **répétitions :**

Clé	Titre	Réalisateur	Année de sortie	Note
1	Pulp Fiction	Quentin Tarantino	1994	20
2	Inglorious Bastards	Quentin Tarantino	2009	18
3	Inglorious Bastards 2	Quentin Tarantino	2024	18
4	Kill Bill 1	Quentin Tarantino	2003	18
5	Kill Bill 2	Quentin Tarantino	2004	18
6	Pulp Fiction	Martin Tarantino	2025	18
7	Holy Grail	Monty Python	1975	19
8	Fight Club	David Fincher	1999	20
9	Life of Brian	Monty Python	1979	17

Clé	Réalisateur	Origine	Date de naissance
1	Quentin Tarantino	USA	1962
2	Monty Python	UK	1969
3	David Fincher	USA	1963

Définitions importantes :

- Un **enregistrement** est _____
  _____
- Un **descripteur** est _____.
- Une clé **primaire** est _____;
- Une clé **étrangère** est _____

Rem : une clé primaire permet d'identifier de manière unique un enregistrement. C'est utile dans le cas d'homonymes (par exemple : Pulp Fiction a deux versions…)

Exemple :

- Un enregistrement s'écrit :
  { 'clé': 4, 'titre': 'Fight Club', 'réalisateur': 'David Fincher', 'année': 1999, 'note': 20 }
- 3 est la clé primaire de l'enregistrement
  { 'clé': 3, 'titre': 'Holy Grail', 'réalisateur': 'Monty Python', 'année': 1975, 'note': 19 }
- Il n'existe pas de clé étrangère ici car les deux tableaux ne sont pas reliés entre eux. Pour en introduire une, il faut donc changer le premier tableau :

Clé	Titre	Réalisateur	Année de sortie	Note
1	Pulp Fiction	1	1994	20
2	Inglorious Bastards	1	2009	18
3	Holy Grail	2	1975	19
4	Fight Club	3	1999	20
5	Life of Brian	2	1979	17

2 est ici la clé étrangère des enregistrements 3 et 5. Cela permet également de standardiser les relations entre tables.

Ce type d'organisation s'appelle "base de données relationnelles" et sera vue en Terminale.

"Ok, non mais c'est cool. Mais moi, j'ai **jamais envoyé de tableaux à personne**."

Réponse : Déjà, ce n'est pas tout à fait vrai. Vous avez sans doute déjà envoyé des centaines — voire des milliers — de tableaux, sans même le savoir. Et, puis, même si vous n'avez jamais envoyé de tableaux, vous en avez déjà reçu beaucoup !

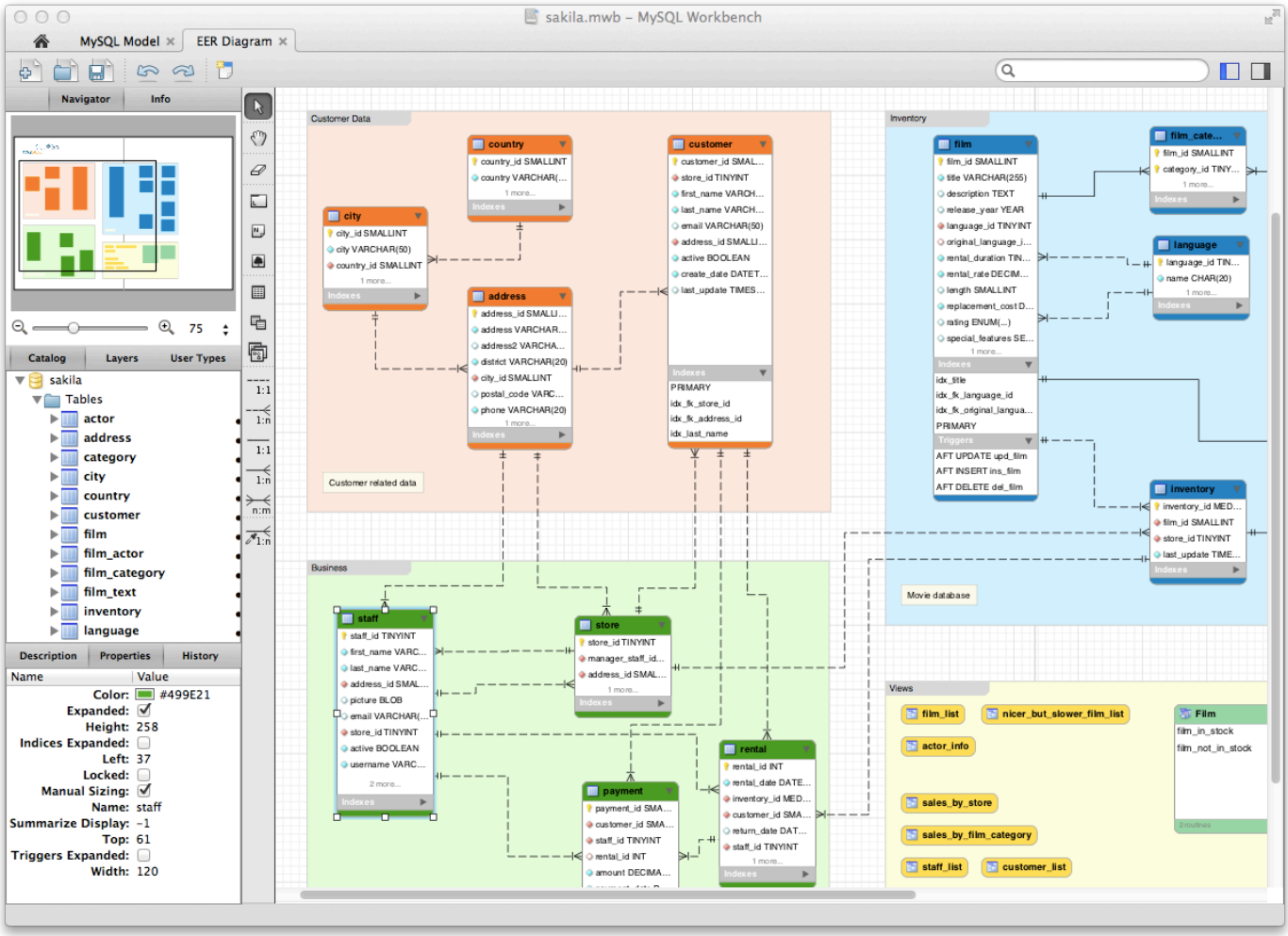
En effet, dans les chapitres précédents, nous avons dit que les serveurs web étaient pratiquement toujours clients de serveurs gérant uniquement des bases de données et stockaient leurs informations importantes dans ces bases. Ces bases de données contiennent ni plus ni moins que de gigantesques tableaux standardisés bourrés d'informations…

Les grandes bases de données sont organisées grâce à des langages de gestion de base de données (comme MySQL ou PostgreSQL). Grâce à l'utilisation de nombreuses clés étrangères, il devient possible de retrouver très rapidement un enregistrement donné.

Toutefois, très souvent, pour afficher une page dans un navigateur, il suffit de charger une toute petite partie de la base de données. Le serveur web vous transmet donc des **petits tableaux** qui sont construits **sous la forme de fichiers texte** simplifié : les deux standards dominant à l'heure actuelle sont le format JSON (JavaScript Object Notation) et le CSV (Comma Separated Value).

Exemple de base MySQL :

On remarque l'extrême importance des clés étrangères et les nombreuses relations entre tableaux.



Cette année, nous allons nous intéresser tout particulièrement au format CSV.

3) Fichiers CSV

Définition :

Le format CSV est couramment utilisé pour importer ou exporter des données d'une feuilles de calcul d'un tableur.

Exemple :

prenom, nom, date_naissance
Poussin, Piou, 2012
Crazy, Frog, 2005
René, LaTaupe, 2009

constitue un tableau au format csv . Pour vous en convaincre, suivez les étapes ci-dessous :

- Recopier les quatre lignes ci-dessus dans un **éditeur de texte** (VSCodium)
- Enregistrer ce texte sous le nom : testCSV.csv.
- Ouvrez testCSV.csv avec Excel ou LibreOffice.

Vous devriez vous apparaitre la belle feuille de calcul ci-contre :

prenom	nom	date_naissance
Poussin	Piou	2012
Crazy	Frog	2005
René	LaTaupe	2009

On vient de voir qu'un fichier CSV était un fichier texte comme dans l'exemple ci-dessus.

Question : comment faire pour représenter un fichier CSV en Python ?

II. Dictionnaires Python

1) Introduction

Utiliser des listes semble être une bonne idée mais, très rapidement, on risque de se retrouver avec des données comme cela (cf Pronote) :

```
[['Prénom;Né;Sexe;Classe;Entrée;Sortie;Option 1;Option 2;Option
3;Option 4;Option 5'], ['Eloise;2A03;G;1G03;02/09/2019;;ANGLAIS
LV1;ITALIEN LV2;MATHEMATIQUES;Numérique et sciences informatiques;
PHYSIQUE-CHIMIE'], ['Valentin;2003;G;1G03;02/09/2019;;ANGLAIS
LV1;ITALIEN LV2;MATHEMATIQUES;Numérique et sciences informatiques;
PHYSIQUE-CHIMIE'], ['Léo;2002;G;1G 08;02/09/2019;;ANGLAIS
LV1;ALLEMAND LV2;Numérique et sciences informatiques;LITT. ANGLAIS;
MATHEMATIQUES'], ['Thibaud;2003;G;1G03;02/09/2019;;ANGLAIS
LV1;ESPAGNOL LV2;Numérique et sciences informatiques;MATHEMATIQUES;
PHYSIQUE-CHIMIE'], ['Julia;2003;F;1G11;02/09/2019 ;;ANGLAIS
LV1;ESPAGNOL LV2;LITT. ANGLAIS;Numérique et sciences informatiques;
PHYSIQUE- CHIMIE'], ['Julien;2002;G;1G06;02/09/2019;;ANGLAIS
LV1;ESPAGNOL LV2;Numérique et sciences informatiques;MATHEMATIQUES;
LITT. ANGLAIS'], ['Sylvain;2003;G;1G09;02/09/2019;;ANGLAIS
LV1;ESPAGNOL LV2;Numérique et sciences informatiques;LI……..]
```

Imaginons : on souhaite accéder à l'Option 3 des élèves ayant anglais comme Option 1 …
Je vais devoir dire : "quand le 6ème élément de la sous-liste est égal à 'ANGLAIS', lis le 8ème élément STP" ➠ **Ignoble**.
Il serait tellement plus facile de dire : "quand l'Option 1 est égal à ANGLAIS, lis l'Option 3".

L'inconvénient des listes dans ce cas est qu'il est pénible d'y rechercher des éléments. En effet, elles sont basées sur la notion **d'indice** et non sur la notion de **contenu** alors que les bases de données sont précisément basées sur la notion de contenu.

Pour faire une chose pareille, il faut nous appuyer sur une autre structure de données disponible en Python et basée sur les **contenus** : les dictionnaires.

2) Manipulation de dictionnaires

Un dictionnaire Python est composé de deux parties : la clé et la valeur. Étudions un exemple de dictionnaire pour mieux comprendre :

```
monDico = {"nom": "LaTaupe", "prenom": "René", "naissance": 2009}
maListe = ["LaTaupe", "René", 2009]
```

Syntaxe :

- un dictionnaire se caractérise par des accolades {} ;
- les clés sont ici "nom", "prenom" et "naissance". Les clés sont de types **`str`** ou **`int`**.
- la valeur associée à la clé "nom" est "LaTaupe", à la clé "prenom" est "René" et à la clé "naissance", 2009. Les valeurs peuvent être de n'importe quel type (y compris liste ou dictionnaire !)

Initialisation :

Il est possible d'initialiser un dictionnaire vide puis de le remplir avec les 4 lignes suivantes :

```
monDico = {}      # monDico = dict() est une autre syntaxe possible.
monDico["nom"] = "LaTaupe"
monDico["prenom"] = "René"
monDico["naissance"] = 2009
```

❐ — Exercice **1** —

Dans VSCodium, tapez le code suivant et exécutez-le :

```
monDico = {"nom": "LaTaupe", "prenom": "René", "naissance": 2009}
print(f"Le type de mon Dico est {type(monDico)}")
print(monDico)
```

Quel est le type affiché ? _____

Qu'avons-nous fait dans la première ligne ? _____

❐ — Exercice **2** —

Dans VSCodium, créez un fichier appelé **dicoTest.py**, tapez le code suivant et exécutez-le:

```
monDico = {"nom": "LaTaupe", "prenom": "René", "naissance": 2009}

print( f'Bonjour je suis {monDico["prenom"]} {monDico["nom"]} !' )
```

Quel est le résultat attendu après l'exécution de ce programme ?

_____

_____

Vérifiez votre réponse.

- En réutilisant la même syntaxe, ajoutez la phrase : "Je suis né en ……" .
- Testez votre programme en changeant le nom de René LaTaupe en Sergent LaTaupe.

Les dictionnaires sont des objets mutables (=pouvant être modifiés).

Rappel :

- les listes sont aussi des objets mutables et sont obtenus avec des crochets [] ;
- les tuples sont des objets immutables obtenus à l'aide de parenthèses ().

❐ — Exercice **3** —

Nous allons modifier le fichier **dicoTest.py.**
En suivant la méthode d'initialisation proposée page 8, rajoutez à votre dictionnaire monDico une clé "lieu_naissance" à laquelle on affectera la valeur "Copacabana".

Le code ressemblera donc à :

```
monDico = {"nom": "LaTaupe", "prenom": "René", "naissance": 2009}
# instruction à rajouter :
# …… …… …… …… …… …… …… …… ……
print( f'Bonjour je suis {monDico["prenom"]} {monDico["nom"]} ! Je suis né à {……} .' )
```

Remarque : Il est maintenant clair qu'on peut accéder aux valeurs contenues dans un dictionnaire en tapant : nomDuDictionnaire["clé"] . C'est exactement ce que l'on souhaite faire pour lire du CSV.

❐ — Exercice **4** —

Nous allons encore modifier le fichier **dicoTest.py.**

À la suite de René, créez un nouveau dictionnaire qui va s'appeler mesFruits :

```
mesFruits = {"poire": 3, "pomme": 4, "orange": 2}
```

Les dictionnaires sont mutables, donc modifiables.

- J'aime bien les oranges. Soustraire 2 oranges à mesFruits en utilisant :

```
mesFruits["orange"] = mesFruits["orange"] - 2
```

- Raccourcir cette instruction en évitant la répétition de `mesFruits["orange"]`

- Afficher ensuite le nombre d'orange contenu dans mesFruits :

```
print("Il y a {……} oranges dans le bol!" )
```

❖ Normalement, vous avez du trouver qu'il n'y a plus d'oranges dans le bol… Supprimons (=**del**ete) donc l'entrée `orange`. Pour faire cela, nous utilisons l'instruction **`del`** :

```
del mesFruits["orange"]
```

Il est possible de parcourir un dictionnaire à l'aide d'une boucle **for**. Par rapport à une liste, ce parcours peut se faire selon les **clés** ou les **valeurs**.

Commençons par parcourir les **clés** à l'aide de la méthode **`keys()`**.

❒ — Exercice **5** —

❖ À la suite de votre fichier **dicoTest.py ,** rajoutez :

```
monBol = {"poire": 3, "pomme": 4, "orange": 2}
print("liste des fruits :")
for fruit in monBol.keys():
   print(fruit)
```

❖ Modifier la boucle for afin d'afficher à chaque passage dans la boucle la clé et la valeur associée. On doit obtenir :

```
poire 3
pomme 4
orange 2
```

❖ Essayer de supprimer l'appel à la méthode **`keys()`** . Que remarquez-vous ?

_____

Remarque : La méthode **`keys()`** _____

Les **valeurs** d'un dictionnaire peuvent être parcourues en utilisant la méthode **`values()`**.

❒ — Exercice **6** —

❖ À la suite de votre travail précédent, dans le fichier **dicoTest.py**, ajoutez :

```
print("quantité de fruits :")
for nombre in monBol.values():
   print(nombre)
```

❖ Essayez de modifier la boucle afin d'obtenir le même affichage que dans le **"Exercice 5"**. Est-ce possible ?

_____

Enfin, les **clés** et les **valeurs** (appelées **champs**) peuvent être parcourues simultanément en utilisant la méthode **`items()`**.

❐ — Exercice **7** —

❖ Finalement, dans le fichier **dicoTest.py**, rajoutez à la suite de votre travail **:**

```
print("liste des fruits et quantité :")
for (fruit, nombre) in monBol.items():
   print(fruit, nombre)
```

❖ Remarquez la syntaxe de la boucle for. Avez-vous déjà vu cette syntaxe ? Rappelez dans quel contexte.

_____

_____

Remarque importante : un dictionnaire Python a exactement le format **JSON** proposé en Javascript.

III. Manipulation de fichiers CSV

1) Lecture, importation et exportation de fichiers CSV

Comme vous l'avez deviné, en Python, nous allons lire les fichiers CSV grâce à des dictionnaires. De cette manière, nous allons pouvoir faire des recherches sur les contenus plutôt que sur des numéros d'indices sans aucun sens.

Toutefois, pour nous aider dans cette tâche, nous **choisissons** de répéter les en-têtes pour chaque élément. Ainsi :

Nom	Français	Science	Histoire
Erwann	16	12	15
Céline	14	16	13

Le tableau ci-contre se notera en Python :
Table = **[**
{**'Nom'**: 'Erwann', **'Français'**: '16', **'Science'**: '12', **'Histoire'**: '15'},
{**'Nom'**: 'Céline', **'Français'**: '14', **'Science'**: '16', **'Histoire'**: '13'}
]

Table[0] renvoie le premier enregistrement :
{'**Nom**': 'Erwann', '**Français**': '16', '**Science**': '12', '**Histoire**': '15'}

Convention : les fichiers CSV lus en Python se présentent toujours sous la forme d'une **liste** de dictionnaires dont les clés sont les en-têtes du tableau et les valeurs sont les contenus.

Nous allons tout d'abord utiliser une bibliothèque pour importer des fichiers CSV. Cette bibliothèque s'appelle… **csv** !

❐ — Exercice **8** —

- Pour ouvrir des fichiers en Python, on utilise la commande suivante :

```
donnéesDuFichier = open( nomDuFichier, modeDOuverture)
```

avec `nomDuFichier` qui est un "string"
et `modeDOuverture` qui vaut **'r'** si l'on veut lire le fichier et **'w'** si on veut écrire un nouveau fichier. *Pour plus d'informations : "Google open python3"*

- Créez un dossier appelé NSI_CSV et téléchargez le fichier exemple.csv disponible sur mon site internet : bouillotvincent.github.io .
- Dans VSCodium, créez un nouveau fichier appelé csvReader.py et, à l'aide de l'instruction **open**, ouvrez le fichier exemple.csv.
- En ajoutant un **print** à votre code, affichez le contenu de exemple.csv.
- Que pensez-vous de votre résultat ?

On ne va pas se mentir,
`<_io.TextIOWrapper name='exemple.csv' mode='r' encoding='UTF-8'>`
n'est pas super lisible et pas vraiment ce que l'on attendait…

❐ — Exercice **9** —

On va importer la bibliothèque **csv** pour pouvoir lire ce fichier que l'on vient d'ouvrir.

- Dans votre fichier csvReader.py, importez la bibliothèque csv.
- Recopiez la fonction importCSV(fichier, separateur) qui va nous permettre d'importer des fichiers CSV.

```
def importCSV(fichier : str, separateur = ";"):
    tCSV = csv.DictReader(open(fichier,'r'), delimiter = separateur)
```

- ❖ Complétez la fonction pour afficher le tableau importé à l'aide d'un **print**, puis testez cette fonction en appelant **table = importCSV('exemple.csv')**.
- ❖ Que constatez-vous ? _____

Caramba… Encore raté : `<csv.DictReader object at 0x10335ea90>`
Toujours pas très lisible…

Pourriez-vous brièvement expliquer l'origine de cette erreur ? Dis autrement : que fait DictReader ?

_____

_____

❐ — Exercice **10** —

Finalement, pour lire le contenu de notre objet csv.DictReader, nous allons simplement parcourir la table `"tCSV"` à l'aide d'une boucle for.

- ❖ Dans la fonction importCSV, ajoutez une boucle for sur une variable que l'on appellera ligne et qui va parcourir la table `"tCSV".` À chaque passage dans la boucle affichez à l'aide de **print** la valeur de la variable ligne.
- ❖ Est-ce que tout fonctionne comme on le souhaitait initialement (p12) ? Pourquoi ?
  _____

❐ — Exercice **12** —

Afin de conserver un certain ordre, la bibliothèque CSV utiliser une structure Python hybride entre liste et dictionnaire qui est appelée un `OrderedDict`.

On peut transformer cette structure en dictionnaire Python simplement en appelant dict() sur celle-ci.

- ❖ Toujours dans la même fonction, afficher `dict(ligne)`.
- ❖ Est-ce que tout fonctionne comme on le souhaitait initialement (p12) ?
  _____
- ❖ Finalement, modifiez votre fonction afin remplir une **liste** que l'on va appeler tableau et qui contiendra tous les dictionnaires que vous avez créé ligne par ligne.

Si vous avez fini avant : Faites la même chose en 2 lignes : une pour importer tCSV et une pour renvoyer le résultat.

❐ — Exercice **13** —

La fonction ci-dessous permet d'exporter une table au format CSV.

❖ Recopiez le code ci-dessous dans votre programme csvReader.py.

```
def exportCSV(tableau : list, fichier : str):
    header = tableau[0].keys()
    fichierCSV = csv.DictWriter(open(fichier, 'w'), fieldnames =
header)
    fichierCSV.writeheader()
    for ligne in tableau:
        fichierCSV.writerow(ligne)
    return None
```

❖ Identifiez ce que fait chaque ligne de ce programme et écrivez-le en commentaires dans votre programme.

❖ Pour vérifier votre code, exportez votre table dans le fichier que l'on nommera "exemple2.csv" , puis ouvrez ce fichier avec un tableur. Votre export a-t-il fonctionné ?

Conclusion :

❖ Sur un plan connaissances pures, on sait maintenant importer et exporter des fichiers CSV grâce à la librairie csv, ce qui va nous permettre de réaliser des opérations sur ces fichiers.

❖ Sur un second plan, vous remarquerez que l'on a beaucoup travaillé pour faire fonctionner la librairie comme on le souhaitait. On peut se demander si on n'aurait pas été plus rapide en créant directement notre propre lecteur et écriveur de fichiers CSV (voir exercice 4!)

2) Opérations sur les tables

❐ — Exercice **14** —

On se propose de faire quelques tests dans le programme principal (PAS les fonctions) de csvReader.py .

Si vous voulez comprendre ce qu'il se passe dans la suite, je vous conseille très fortement **d'écrire** les résultats des questions suivantes sur le cours ou une feuille que vous garderez en face de vous :

❖ Qu'affiche **print(table[0])** ? _____

❖ À quoi vous attendez-vous si vous tapez **print(table[1])** ?

_____

- Qu'affiche **print(table[0]['Nom'])** ? _____
- Comment feriez-vous pour afficher la note de Céline en Science ?

_____

- Quel résultat renvoie **print(table[0:1]['Nom'])** ? _____
  Que pouvez déduire de cela ?

_____

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

❐ — Exercice **15** —

Proposez une fonction filtrerLigne 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 apparaitre.

```
def filtrerLigne(tableau : list, critere : str, valeur : str):
    … … … … … … … … … … …
    return filtrerTableau
```

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

Filtrer suivant une colonne s'appelle aussi une **projection**.

❐ — Exercice **16** —

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

- Compléter la fonction filtrerColonne 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 apparaitre.

```
def filtrerColonne(tableau : list, listeCriteres : str):
    filtrerTableau = []
    for dico in tableau:
        dicoFiltrer = {}
        for …… in …… :
            …… # ici
        filtrerTableau.append(dicoFiltrer)
    return filtrerTableau
```

❖ Testez votre code grâce à un print : print(filtrerColonne(table, 'Nom')) .

Conservez une seule colonne est assez inutile… On suppose à présent que l'utilisateur peut donner une **liste** de critères. `listeCriteres` 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 un objet 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 **17** —

❖ Recopiez le code ci-dessous :

```
def triTable(tableau: list, critere: str, decroissant = False ):
    return sorted(tableau, key=lambda f: f[critere],
reverse=decroissant)
```

❖ Testez votre code en lançant triTable 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" .
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 tranquille

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

Version lambda rapide

```
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.

3) 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 **18** —

Fusion : cette étape est à faire en autonomie.

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

- Créez un nouveau fichier appelé jointure.py et depuis csvReader, importez la fonction importCSV.
- Créez une fonction `verifieCle` qui vérifie si les descripteurs de vos deux tables sont strictement égales. `verifieCle` prendra en argument d'entrée : deux tables table1 et table2 formatées de manière classique (liste de dictionnaires). `verifieCle` renverra True si les descripteurs 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 (liste de dictionnaires). `fusion` renverra une table fusionnée tableFusion.
- Finalement, la table fusionnée sera exportée au format csv grâce à l'importation de la fonction exportCSV de notre bibliothèque csvReader.
- *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 **19** —

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 exempleAge.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 exportCSV de notre bibliothèque csvReader.

Projet

On va passer sur cela cette année…