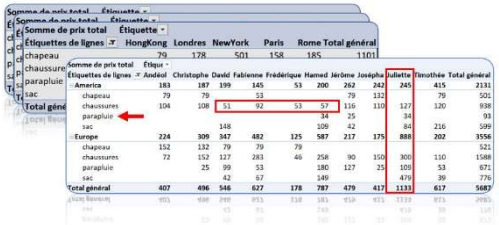
TP – Manipulation de données en tables



Objectifs :

- ⇨ Rechercher des données dans une table
- ⇨ Sélection de lignes et de colonnes d'une table
- ⇨ Trier des tables selon un ou plusieurs descripteurs
- ⇨ Fusionner les données de plusieurs tables et les agréger

I - Recherche dans une table

Prénom	Année	Langage	Intitulé
Léna	2018	Java	Jeu du pendu
Ricardo	2018	Libgdx	Platformer
Alexia	2017	Java	Tétris
Mathias	2017	Java	Spaceship
Thomas	2017	Java	FlowFree
Sabrina	2019	Java	Snake
Sami	2019	Swing	Puissance 4
Lilouanne	2020	Brython	Site mathématique
Pierre	2020	Python	Sortie d'un labyrinthe
Mathias	2020	Python	Jeu de la vie

1) Sélection

Une recherche simple dans une table consiste à rechercher l'enregistrement pour lequel un champ possède une certaine valeur.
On appelle cette opération une **sélection** car on sélectionne les lignes qui correspondent à certain critères.
Par exemple dans la liste des projets d'élèves, on pourrait rechercher les enregistrements correspondant à l'année 2017.

Pour effectuer la recherche, une simple boucle for (si on veut tous les résultats) ou while (si on veut juste le premier) suffit :

```
projets = [{'Prénom': 'Léna', 'Année': '2018', 'Langage': 'Java', …
enregistrements = []
for e in projets:
        if e['Année'] == '2017':
            enregistrements.append(e)
```

Donnera :

```
[{'Prénom': 'Alexia', 'Année': '2017', 'Langage': 'Java', 'Intitulé': 'Tétris'}, {'Prénom
': 'Mathias', 'Année': '2017', 'Langage': 'Java', 'Intitulé': 'Spaceship'}, {'Prénom': 'T
homas', 'Année': '2017', 'Langage': 'Java', 'Intitulé': 'FlowFree'}]
```

On peut également procéder par compréhension :

```
enregistrements = [e for e in projets if e['Année'] == '2017']
```

2) Projection

La projection consiste à ne prendre que certaines colonnes du tableau.
Projetons par exemple le tableau projet uniquement sur le prénom et l'intitulé du projet pour former un extrait :

```
projection = []
for e in projets:
    projection.append({'Prénom':e['Prénom'], 'Projet':e['Intitulé']})
print(projection)
```

On ne récupère que les informations qui nous intéressent

On obtient :

```
[{'Prénom': 'Léna', 'Projet': 'Jeu du pendu'}, {'Prénom': 'Ricardo', 'Projet': 'Platformer'},
{'Prénom': 'Alexia', 'Projet': 'Tétris'}, {'Prénom': 'Mathias', 'Projet': 'Spaceship'}, {'Prénom':
'Thomas', 'Projet': 'FlowFree'}, {'Prénom': 'Sabrina', 'Projet': 'Snake'}, {'Prénom': 'Sami',
'Projet': 'Puissance 4'}, {'Prénom': 'Lilouanne', 'Projet': 'Site mathématique'}, {'Prénom':
'Pierre', 'Projet': 'Sortie d'un labyrinthe'}]
```

Par compréhension :

```
projection = [{'Prénom':e['Prénom'], 'Projet':e['Intitulé']} for e in projets]
```

On remarque qu'on en a profité pour renommer le champ « Intitulé » en « Projet ».

On peut facilement procéder à une sélection et une projection en même temps.
Pour récupérer le prénom et l'intitulé de tous les projets de 2017, on peut faire :

```
selection_et_projection = []
for e in projets:
    if e['Année'] == '2017':
        selection_et_projection.append({'Prénom':e['Prénom'], 'Projet':e['Intitulé']})
print(selection)
```

Ou en compréhension :

```
selection=[{'Prénom':e['Prénom'],'Projet':e['Intitulé']} for e in projets if e['Année']=='2017']
```

Question 1 :

1) Ouvrir le fichier « Base_Question_1.py » avec Thonny et prendre connaissance des fonctions déjà programmées.
2) Utiliser la fonction `lecture_fichier_csv` pour lire le fichier « countries.csv » puis écrire le programme pour sélectionner la ligne correspondant à la France.
3) Sélectionner ensuite tous les pays du continent « AF » (Afrique). Combien y a-t-il de pays africains dans la table ?
4) Projeter les champs « Name » et « Currency_Name » (monnaie).
5) Parmi cette projection, sélectionner ensuite les pays ayant pour monnaie l'euro. Combien dénombrez-vous de pays ?

II - Tri d'une table

Dans cette partie, nous allons voir comment trier une table de donnée en utilisant les fonctions built-ins de python (nous verrons les algorithmes de tri dans un prochain chapitre).

1) `sort` et `sorted`

Python possède une fonction built-in `sorted` qui prend en argument un itérable et renvoie une liste triée des valeurs de cet itérable.
On peut aussi utiliser la méthode `sort` de la classe liste qui effectue un tri en-place : si liste est une liste, alors `liste.sort()` permet de trier le tableau `liste`.

```
>>> liste = [4, 8, -2, 6, 0, 12, 3]
>>> sorted(liste)
[-2, 0, 3, 4, 6, 8, 12]
>>> liste
[4, 8, -2, 6, 0, 12, 3]
>>> liste.sort()
>>> liste
[-2, 0, 3, 4, 6, 8, 12]
>>> e = {53, -15, 6, 33, 17, 44}
>>> e
{33, 6, 44, -15, 17, 53}
>>> e.sort()
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
AttributeError: 'set' object has no attribute 'sort'
>>> sorted(e)
```

`sorted` renvoie une liste triée

et la liste d'origine n'a pas été touchée

En revanche la liste est modifiée par la méthode `sort`

Un ensemble n'a pas d'ordre

et ne peut donc pas être trié directement

Par contre on peut créer une liste triée de ses valeurs

```
[-15, 6, 17, 33, 44, 53]
```

Essayons maintenant de trier une table :

```
>>> agenda = [{"id":1, "Nom":"NAYMAR", "Prénom":"Jean"},
              {"id":2, "Nom":"ZEBLOUZE", "Prénom":"Agathe"},
              {"id":7, "Nom":"TASSION", "Prénom":"Gaëtan"},
              {"id":9, "Nom":"COVER", "Prénom":"Harry"}]

>>> sorted(agenda)
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: '<' not supported between instances of 'dict' and 'dict'
```

Il n'est pas possible de trier une telle liste.

Explication technique :
Ces fonctions de tri fonctionnent par comparaison des éléments entre eux pour les classer selon l'ordre croissant. Elles ne peuvent donc classer que des listes contenant des éléments comparables.
Des dictionnaires ne sont pas directement comparables entre eux, c'est pourquoi le tri d'une liste de dictionnaire n'est pas possible.

Pour pouvoir trier des listes complexes, les fonctions de tri proposent un argument `key` qui doit faire référence à une fonction prenant un seul argument et renvoyant une valeur.
Cette fonction sera appliquée à tous les éléments de la liste et sa valeur de retour sera utilisée pour comparer les éléments entre eux (c'est la **clé de tri**).
Pour trier une liste de dictionnaires selon une de leur clé, il suffit de créer une fonction qui renvoie la valeur de cette clé :

```
agenda = [{"id":1, "Nom":"NAYMAR", "Prénom":"Jean"},
          {"id":2, "Nom":"ZEBLOUZE", "Prénom":"Agathe"},
          {"id":7, "Nom":"TASSION", "Prénom":"Gaëtan"},
          {"id":9, "Nom":"COVER", "Prénom":"Harry"}]

def cle(dico):              # Fonction clé pour le tri
    return dico['Nom']      # renvoie la valeur du champ "Nom"

print(sorted(agenda, key=cle))
```

Les fonctions de tri peuvent également prendre un deuxième paramètre optionnel : `reverse`. S'il vaut `True`, le tri se fait selon l'ordre décroissant au lieu d'être dans l'ordre croissant (par défaut reverse vaut `False`).

2) Premier tri d'une table

Question 2 :
1) Utiliser la fonction `lecture_fichier_csv` puis trier la table selon la population décroissante. Afficher la table triée à l'aide de la procédure `affiche_table`. Que remarquez-vous ?

Notre déconvenue de la question précédente s'explique par le fait que le tri s'est fait sur des chaînes de caractères, donc selon l'ordre lexicographique et pas selon l'ordre numérique.
Pour éviter cela, il il aurait fallu ne pas oublier une étape importante dans le traitement des données en tables : la **validation**.

En effet, après avoir récupéré les données du fichier csv, on doit procéder à la **validation** des données en vérifiant leur cohérence et en transformant ces données (qui sont forcément sous forme texte (`str`)) pour leur donner le bon format (notamment transformer les chiffres en nombres).

Pour valider les données, on écrit généralement une fonction de validation qui valide un enregistrement, puis on fait une boucle (éventuellement par compréhension) pour balayer tous les enregistrements de la table.

Si on reprend l'exemple du début, on pourrait écrire :

```
def validation(e):
    e['Prénom'] = e['Prénom'][:12]   # Limite le prénom à 12 caractères
    e['Année'] = int(e['Année'])     # Converti l'année en nombre
    if e['Année'] < 2000 or e['Année'] > 2100:
        print("Erreur dans la validation, année du projet incorrecte :",e['Année'],"\nPour :",e)
    return e

table_validee = [validation(e) for e in projets]
```

Question 2 :

2) Rajouter une étape de validation à votre programme (en transformant la population en entier et la superficie en flottant) et vérifier que le tri s'effectue maintenant correctement.

3) Tri selon plusieurs critères

Nous voudrions maintenant trier la liste des pays selon plusieurs critères : d'abord par continent, puis par population.

Question 3 :

1) Essayer de trouver deux méthodes permettant de trier la liste des pays selon ces deux critères. Si vous ne trouvez pas, vous pouvez vous référer à l'aide à la suite de cet exercice pour continuer.
2) Mettre en œuvre la méthode des tuples et vérifier qu'on obtient bien le bon résultat.
3) Mettre en œuvre la méthode du double tri en triant par ordre croissant du continent et par ordre décroissant de population.

Aides :

- Lorsqu'on compare deux tuples, la comparaison s'opère d'abord sur la première valeur du tuple et en cas d'égalité, sur la deuxième, puis la troisième …. Par exemple `(1,9)` < `(2,5)` < `(2,6)` < `(3,0)`
- On peut aussi effectuer deux tris successifs : on crée une première liste triée selon le premier critère, puis on reprend cette liste et on la trie selon le deuxième critère.

La méthode des tris successifs fonctionne car le tri de python a une propriété importante ici : il est **stable**. Cela signifie qu'en cas d'égalité de la clé de tri, l'ordre des éléments est préservé. Nous en reparlerons lorsque nous verrons les algorithmes de tri.

III – Fusion de tables

1) Jointure

Si on reprend le premier exemple avec les projets. Imaginons que nous ayons une deuxième table contenant les prénoms et les notes que les élèves ont obtenus à leur projet (voir ci-contre).

Prénom	Note
Léna	17
Ricardo	20
Alexia	15
Mathias	16
Thomas	11
Sabrina	18
Sami	20
Lilouanne	18
Pierre	15
Mathias	12

Il serait intéressant de faire la correspondance entre ces deux tables pour connaitre par exemple les notes en fonction des projets ou faire la moyenne des notes pour chaque année.
Mettre deux tables en correspondance, c'est faire une **jointure** entre les deux tables. Pour que la jointure puisse se faire, il faut que les tables aient un *descripteur commun* (le nom du descripteur ne doit pas forcément être le même, mais l'information qu'ils représentent doit être identique). C'est le cas ici de la colonne « Prénom » qui est présente dans les deux tables et représente la même information.
La *jointure* des tables se fait généralement avec une *sélection* des champs d'une part pour éviter de doublonner le descripteur commun et d'autre part parce que toutes les informations des deux tables ne sont pas forcément pertinentes simultanément.
Pour la mise en œuvre, là encore une boucle for ou une compréhension de liste permettra de parcourir les deux tables. Il faudra juste rajouter la condition que les valeurs des descripteurs commun soient les même dans les deux tables.
Par exemple pour obtenir les notes obtenues pour chaque projet, on pourrait écrire :

Sélection

```
jointure = [{'Intitulé':p['Intitulé'], 'Note':n['Note']} \
            for p in projets for n in notes if p['Prénom'] == n['Prénom']]
print(jointure)
```

Jointure

On obtient :

```
[{'Intitulé':'Jeu du pendu', 'Note':'17'}, {'Intitulé':'Platformer', 'Note
':'20'}, {'Intitulé':'Tétris', 'Note':'15'}, {'Intitulé':'Spaceship', 'Not
e':'16'}, {'Intitulé':'Spaceship', 'Note':'12'}, {'Intitulé':'FlowFree', '
Note':'11'}, {'Intitulé':'Snake', 'Note':'18'}, {'Intitulé':'Puissance 4',
 'Note':'20'}, {'Intitulé':'Site mathématique', 'Note':'18'}, {'Intitulé':
'Sortie d'un labyrinthe', 'Note':'15'}, {'Intitulé':'Jeu de la vie', 'Note
':'16'}, {'Intitulé':'Jeu de la vie', 'Note':'12'}]
```

On remarque que le projet « Spaceship » et « Jeu de la vie » apparaissent deux fois avec les notes 16 et 12 alors qu'ils n'apparaissent qu'une seule fois dans les tables.
Cela est dû au fait que deux élèves ont le même prénom et que donc lors de la jointure, la correspondance s'est faite pour chaque projet deux fois (une fois pour chaque prénom).
Cela montre que les descripteurs utilisés pour les jointures doivent avoir une propriété importante : **l'unicité de la clé**. Il ne faut pas qu'il puisse y avoir deux enregistrements avec la même valeur pour ce descripteur.
Le prénom est donc un mauvais choix car il est fréquent de trouver des prénoms identiques. C'est pourquoi dans les grandes tables de données, on préfère utiliser un **numéro d'identifiant unique** (souvent noté « Id ») pour désigner un enregistrement et faciliter les jointures.
C'est le cas des deux tables de « countries.csv » et « cities.csv » : countries.csv possède un champ « Capital_Id » qui contient un identifiant unique faisant référence au champ « Id » du fichier cities.csv.

Table « countries.csv »

ISO	Name	Capital_Id	Area	Population	Cont	Cur	Cur_Name
AD	Andorra	3041563	468	84000	EU	EUR	Euro
AE	United Arab Emirates	292968	82880	4975593	AS	AED	Dirham
AF	Afghanistan	1138958	647500	29121286	AS	AFN	Afghani
AG	Antigua and Barbuda	3576022	443	86754	NA	XCD	Dollar
AI	Anguilla	3573374	102	13254	NA	XCD	Dollar
AL	Albania	3183875	28748	2986952	EU	ALL	Lek
AM	Armenia	616052	29800	2968000	AS	AMD	Dram

Table « cities.csv »

Id	Name	Latitude	Longitude	C_ISO	Population
3040051	les Escaldes	42.50729	1.53414	AD	15853
3041563	Andorra la Vella	42.50779	1.52109	AD	20430
290594	Umm Al Quwain City	25.56473	55.55517	AE	62747
291074	Ras Al Khaimah City	25.78953	55.9432	AE	351943
291580	Zayed City	23.65416	53.70522	AE	63482
291696	Khawr FakkÄ n	25.33132	56.34199	AE	40677
292223	Dubai	25.07725	55.30927	AE	2956587
292231	Dibba Al-Fujairah	25.59246	56.26176	AE	30000

Identifiant unique permettant la jointure

Question 4 :
Importer les données des deux tables « countries.csv » et « cities.csv » puis fusionner les tables pour obtenir une table unique contenant les champs suivants : 'ISO', 'Nom' (du pays), 'Superficie', 'Population totale', 'Capitale' (nom de la capitale), 'Pop. capitale' (population de la capitale), 'Continent'.

2) Agrégation de données

On souhaite maintenant faire de l'**agrégation de données**, c'est-à-dire créer des informations nouvelles à partir des informations des tables de données.

Les agrégations peuvent être très simples, comme le calcul de la moyenne, du nombre d'éléments ou du maximum[1], ou plus complexes en croisant les informations de plusieurs tables (grâce à des jointures).

Reprenons l'exemple des notes de projets et calculons la moyenne :

```
n = [int(note['Note']) for note in notes]  # Crée une liste simple contenant les notes
print("Il y a", len(n), "notes de projet. La plus élevée est", max(n), "et la plus basse", min(n),
      "avec une moyenne de", sum(n)/len(n))
```

On aura alors `n = [17, 20, 15, 16, 11, 18, 20, 18, 15, 12]`
et l'affichage donnera : `Il y a 10 notes de projet. La plus élevée est 20 et la plus basse 11 avec une moyenne de 16.2`

Question 5 :
1) En se servant des données validées de countries.csv, déterminer la superficie moyenne des pays.
2) Déterminer la plus forte et la moins forte densité de population (la densité de population s'obtient en divisant la population du pays par sa superficie).

TRAVAIL A FAIRE POUR LA PROCHAINE SEANCE :

Travail sur le cours :
Faire une fiche sur ce chapitre avec les définitions des termes suivants : Sélection, Projection, Validation, Jointure ainsi qu'un mémo sur les fonctions de tri de python (`sort` et `sorted`).

Travail de programmation :
En utilisant les méthodes vues précédemment, créer et afficher la liste triée des 10 pays ayant le plus faible ratio [population dans la capitale] / [population totale].

Aide : Une façon d'arriver au résultat est de commencer par créer à partir de la jointure réalisée à la question 4 une liste de tuples pour chaque pays dont le premier élément est le ratio de ce pays et le deuxième son code ISO (qui permet d'identifier ce pays et de faire la jointure ensuite). On peut ensuite trier cette liste (puisqu'elle contient des tuples et non des dictionnaires, il n'y a même pas besoin de fonction clé), puis faire la jointure sur le code ISO de la liste triée avec la liste de départ pour récupérer toutes les informations comme le nom du pays et de sa capitale. Enfin, il suffit de créer un extrait de cette dernière jointure contenant les 10 premiers éléments. Bien sûr il y a d'autres façons de procéder. A vous de choisir celle qui vous convient le mieux.

[1] On pourra ici se servir des fonctions built-ins `max`, `min` et `sum` (qui fait la somme des nombres d'un tableau).