# 14

# Traitement de données

## Extrait du programme

### Thème : Traitement de données en tables

**Contenus :**
Indexation de tables
**Capacités attendus :**
Importer une table depuis un fichier texte tabulé ou un fichier CSV.
**Commentaires :**
Est utilisé un tableau doublement indexé ou un tableau de p-uplets qui partagent les mêmes descripteurs.


**Contenus :**
Recherche dans une table
**Capacités attendus :**
Rechercher les lignes d'une table vérifiant des critères exprimés en logique propositionnelle.
**Commentaires :**
La recherche de doublons, les tests de cohérence d'une table sont présentés.

**Contenus :**
Tri d'une table
**Capacités attendus :**
Trier une table suivant une colonne.
**Commentaires :**
Une fonction de tri intégrée au système ou à une bibliothèque peut être utilisée.

**Contenus :**
Fusion de tables
**Capacités attendus :**
Construire une nouvelle table en combinant les données de deux tables.
**Commentaires :**
La notion de domaine de valeurs est mise en évidence.

## Avant propos

Les parties 3 à 6 sont fortement inspirées du document d'accompagnement « Manipulation de Tables » que vous pouvez retrouver en suivant le lien ci-dessous :
`https://cache.media.eduscol.education.fr/file/NSI/78/1/RA_Lycee_G_NSI_trtd_tables_1170781.pdf`.

# I. Indexation de tables

## a) Open data

Un contenu est dit en **open data** s'il est librement utilisable, modifiable et partageable par n'importe qui et pour n'importe quelle raison.

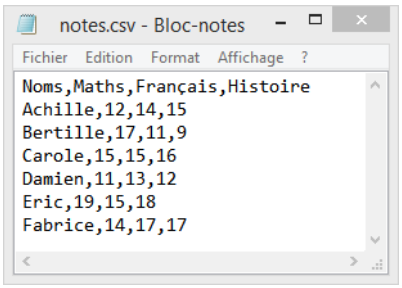
- En France, `https://www.data.gouv.fr/fr/`, est un site de partage des données publiques.
- Pour Angers, nous avons `https://data.angers.fr/pages/home/`
- Certaines données sont collaboratives, par exemple : `https://fr.openfoodfacts.org/` autour des produits alimentaires ou `https://www.openstreetmap.fr/` pour la cartographie.
  Le site Kaggle recense un nombre impressionnant de donnés collaborative : `https://www.kaggle.com/`

Les formats de données les plus courants sont **CSV** et **JSON**.

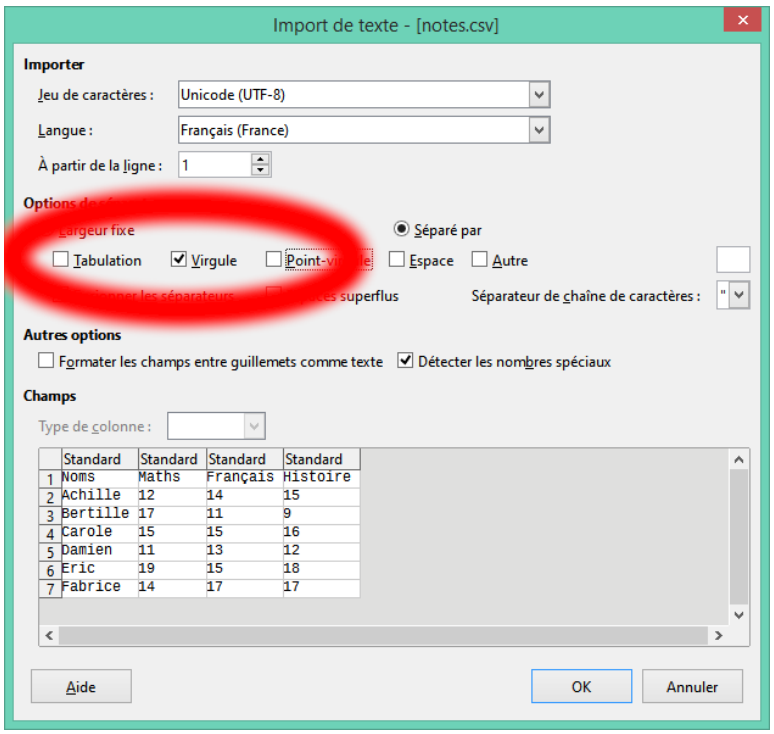
## b) Format CSV

**CSV** signifie *Comma-separated values* car à l'origine il s'agit d'un format texte qui permet de séparer les données par des virgules. Comme certains pays utilisent la virgule, par exemple pour les nombres décimaux, le format du séparateur est désormais beaucoup plus ouverts (virgule, point-virgule, tabulation, ...)
Ce format est en quelques sortes un tableur sans le cadre.

Ouvrons par exemple le fichier `notes.csv` avec le bloc-notes. Nous visualisons bien le fichier texte suivant :




Ouvrons le maintenant avec le tableur LibreOffice Calc. Le logiciel commence par nous demander quel est le séparateur que nous allons choisir parmi la tabulation, la virgule, le point-virgule, l'espace ou autre. Sélectionnons `Virgule` puis appuyons sur `OK`.



Nous obtenons le tableur suivant :

| | A | B | C | D |
|---|---|---|---|---|
| 1 | Noms | Maths | Français | Histoire |
| 2 | Achille | 12 | 14 | 15 |
| 3 | Bertille | 17 | 11 | 9 |
| 4 | Carole | 15 | 15 | 16 |
| 5 | Damien | 11 | 13 | 12 |
| 6 | Eric | 19 | 15 | 18 |
| 7 | Fabrice | 14 | 17 | 17 |

Dans la suite, nous allons voir comment, avec le langage Python, nous pouvons récupérer ces données, puis les traiter. Évidemment, dans cet exemple, les données sont peu nombreuses et on pourrait obtenir les mêmes résultats à la main facilement avec un tableur. Malheureusement, le plus souvent, lorsque l'on traite des données elles sont nombreuses (plusieurs milliers, voire plusieurs centaines de milliers, voire plus encore). L'utilisation d'un tableur devient alors difficile, quasi impossible.

### c) Importer une table

Avant tout importation, il faut savoir quel est le séparateur qui a été utilisé. Ici il s'agit de la virgule.

```
# -*- coding : utf-8 -*-
import codecs #pour lire en UTF8, n'est pas toujours nécessaire


#ouverture du fichier
fichier = codecs.open("notes.csv", 'r', 'utf-8')

#on lit la première ligne qui contient les descripteurs mais pas les données
fichier.readline()
#on est maintenant à la deuxième ligne

#on crée la table de données
donnees = []

#on lit le fichier pour transférer les données dans la table
for ligne in fichier:
    #on enlève le retour à la ligne \r et la nouvelle ligne \n
    traitement = ligne.replace('\r\n','')
    #on récupère chaque données de la ligne séparées par une virgule sous la forme d'un tableau
    traitement = traitement.split(',')
    #on rajoute ce tableau à notre table
    donnees.append(traitement)

#fermeture du fichier
fichier.close()

#pour vérification
print(donnees)
```

Le résultat obtenu est une table ne contenant que les données, pas les descripteurs (la première ligne n'a été que lue mais non traitée).

```
[['Achille', '12', '14', '15'], ['Bertille', '17', '11', '9'], ['Carole', '15', '15', '16'], ['Damien', '11', '13', '12'], ['Eric', '19', '15', '18'], ['Fabrice', '14', '17', '17']]
```

### d) Le module `csv`

Le module `csv` permet de faire la même chose tout en résolvant certains petits problèmes délicats facilement (par exemple si une donnée est une chaîne de caractères contenant une virgule, il ne faut pas prendre cette virgule comme un séparateur...)

**Remarque**
Aucune connaissance n'est exigible sur l'utilisation de ce module.

```
1  # -*- coding : utf-8 -*-
2  import codecs #pour lire en UTF8, n'est pas toujours nécessaire
3  import csv
4
5
6  donnees = []
7
8  with codecs.open('notes.csv', 'r', 'utf-8') as fichier:
9      #on lit la première ligne qui contient les descripteurs mais pas les données
10     fichier.readline()
11     #on est maintenant à la deuxième ligne
12
13     csvnotes = csv.reader(fichier, delimiter=',')
14     for ligne in csvnotes:
15         donnees.append(ligne)
16
17 #pour vérification
18 print(donnees)
```

**Remarque**
Il est à noter que la commande `with codecs.open('notes.csv', 'r', 'utf-8') as fichier:` n'est pas spécifique au module `csv`. On aurait très bien pu l'utiliser dans la version précédente. Un des intérêts de cette méthode est qu'il n'y a pas besoin de refermer le fichier comme on a pu le faire précédemment **ligne 25**.

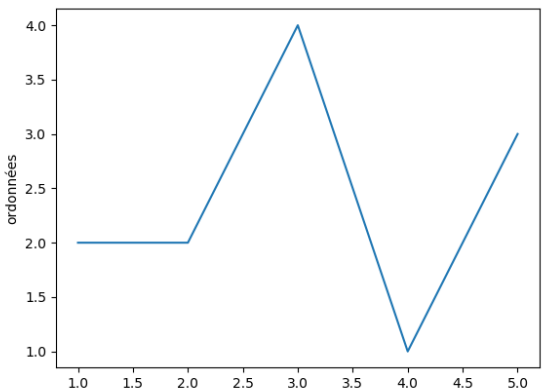
### e) Le module `matplotlib`

Le module `matplotlib` permet de faire de nombreux types de graphiques, courbes, histogrammes, ...
Pour un ensemble de cours et de tutoriels sur ce module : `https://matplotlib.org/`

**Remarque**
Aucune connaissance n'est exigible sur l'utilisation de ce module.

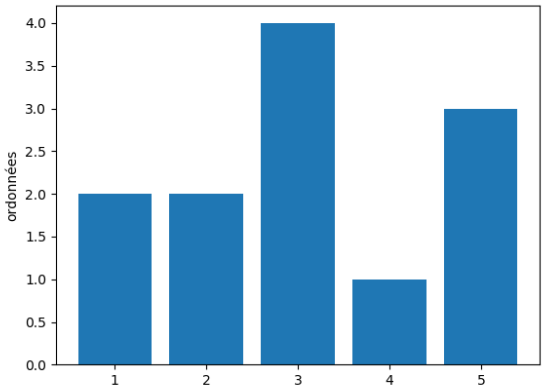
#### Tracer une courbe

```
1  import matplotlib.pyplot as plt
2
3  #liste des abscisses
4  listeX = [1, 2, 3, 4, 5]
5  #liste des ordonnées
6  listeY = [2, 2, 4, 1, 3]
7  #on trace la courbe avec plot()
8  plt.plot(listeX, listeY)
9  #une étiquette pour l'axe des ordonnées
10 plt.ylabel('ordonnées')
11 #ne pas oublier d'afficher le résultat
12 plt.show()
```



### Tracer un diagramme en bâtons

```
import matplotlib.pyplot as plt

#liste des abscisses
listeX = [1, 2, 3, 4, 5]
#liste des ordonnées
listeY = [2, 2, 4, 1, 3]
#on trace le diagramme avec bar()
plt.bar(listeX, listeY)
#une étiquette pour l'axe des ordonnées
plt.ylabel('ordonnées')
#ne pas oublier d'afficher le résultat
plt.show()
```




## II. Exercices

Tous les exercices 1 à 4 sont des traitements du fichiers `notes.csv`.

| | A | B | C | D |
|---|---|---|---|---|
| 1 | Noms | Maths | Français | Histoire |
| 2 | Achille | 12 | 14 | 15 |
| 3 | Bertille | 17 | 11 | 9 |
| 4 | Carole | 15 | 15 | 16 |
| 5 | Damien | 11 | 13 | 12 |
| 6 | Eric | 19 | 15 | 18 |
| 7 | Fabrice | 14 | 17 | 17 |

**Exercice 1 (*)**
Écrire un programme qui calcule la moyenne de chaque élèves.

**Exercice 2 (*)**
Écrire un programme qui calcule la moyenne dans chaque matières.

**Exercice 3 (*)**
Écrire un programme qui affiche le diagramme en bâtons des notes obtenus dans chaque matière par un élève choisi par l'utilisateur.

**Exercice 4 (**)**
Écrire un programme qui rajoute une ligne *« moyenne par matière »* et une colonne *« moyenne par élève »* au fichier `notes.csv` avec les données calculées aux exercices précédents.

**Exercice 5 (Titanic - **)**


Pour cet exercice on travaillera avec le fichiers `titanic_kaggle.csv` qui est une version légèrement modifiée d'un fichier provenant du site **Kaggel** : `https://www.kaggle.com/c/titanic`

Ce fichier `titanic.csv` contient les descripteurs suivants :

- `PassengerId`, un numéro d'identifiant des passager ;
- `Survived`, 0 pour Non, 1 pour Oui ;
- `pclass`, billet 1re, 2e ou 3e classe ;
- `Name`, Nom et Prénoms du passager ;
- `sex`, le genre, male ou female ;
- `age`, l'âge en années ;
- `sibsp`, nombre de frères, sœurs, demi-frères, demi-soœurs, époux, épouse à bord ; (les maîtresses et fiancées sont ignorées)
- `parch`, nombre de parents ou d'enfants à bord ; (bien que certains enfants voyageaient avec une nurse parch=0)
- `ticket`, numéro du ticket ;
- `cabin`, numéro de la cabine ;
- `Embarked`, port d'embarquement ; C = Cherbourg, Q = Queenstown, S = Southampton.

1. A l'aide de différentes fonctions, déterminer le nombre :
   - de survivants
   - de décédés
   - de 1re classe
   - de 2e classe
   - de 3e classe
   - de femmes survivantes en 1re classe
   - d'hommes décédés en 3e classe
2. J'ai toujours entendu dire « les femmes et les enfants d'abord ». Cet ordre a-t-il été respectée sur le Titanic ?
3. Écrire un programme qui retourne un fichier `csv` contenant le tableau suivant donnant les taux de survie des 1re, 2e et 3e classes pour les hommes et les femmes :

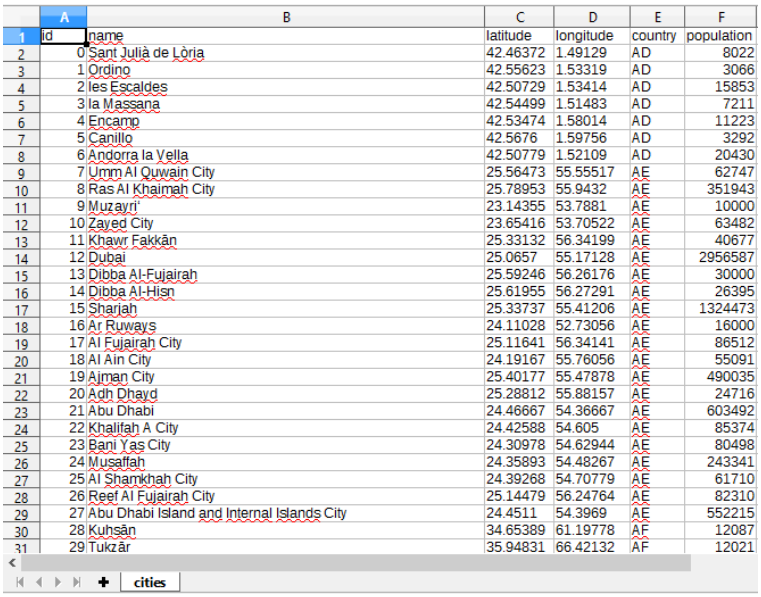
| | 1re | 2e | 3e |
|---|---|---|---|
| Femmes | | | |
| Hommes | | | |

4. Écrire un programme retournant les taux de survie pour les 1re, 2e et 3e classes sous la forme d'un diagramme en bâtons.
5. Écrire un programme retournant les taux de survie pour chaque classe d'âge (0-9 ans, 10-19 ans, 20-29 ans, ...) sous la forme d'un diagramme circulaire.

# III. Manipulation de Tables

Il existe des modules pour le traitement de données issues de tables, comme `pandas`. Le parti pris pour la suite est de ne pas les utiliser.

Nous travaillerons avec les fichiers `cities.csv` et `countries.csv` qui comme leurs noms l'indiques sont au format **CSV**.

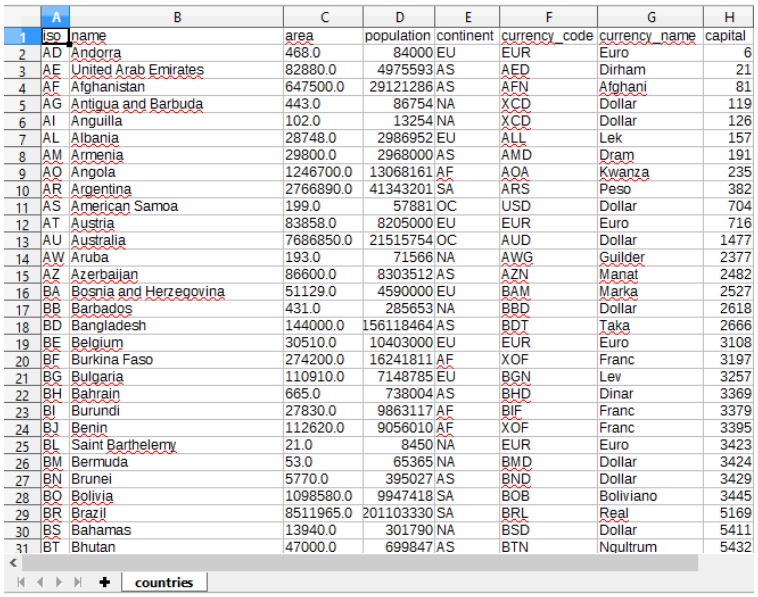


| | id | name | latitude | longitude | country | population |
|---|---|---|---|---|---|---|
| 2 | 0 | Sant Julià de Lòria | 42.46372 | 1.49129 | AD | 8022 |
| 3 | 1 | Ordino | 42.55623 | 1.53319 | AD | 3066 |
| 4 | 2 | les Escaldes | 42.50729 | 1.53414 | AD | 15853 |
| 5 | 3 | la Massana | 42.54499 | 1.51483 | AD | 7211 |
| 6 | 4 | Encamp | 42.53474 | 1.58014 | AD | 11223 |
| 7 | 5 | Canillo | 42.5676 | 1.59756 | AD | 3292 |
| 8 | 6 | Andorra la Vella | 42.50779 | 1.52109 | AD | 20430 |
| 9 | 7 | Umm Al Quwain City | 25.56473 | 55.55517 | AE | 62747 |
| 10 | 8 | Ras Al Khaimah City | 25.78953 | 55.9432 | AE | 351943 |
| 11 | 9 | Muzayri' | 23.14355 | 53.7881 | AE | 10000 |
| 12 | 10 | Zayed City | 23.65416 | 53.70522 | AE | 63482 |
| 13 | 11 | Khawr Fakkān | 25.33132 | 56.34199 | AE | 40677 |
| 14 | 12 | Dubai | 25.0657 | 55.17128 | AE | 2956587 |
| 15 | 13 | Dibba Al-Fujairah | 25.59246 | 56.26176 | AE | 30000 |
| 16 | 14 | Dibba Al-Hisn | 25.61955 | 56.27291 | AE | 26395 |
| 17 | 15 | Sharjah | 25.33737 | 55.41206 | AE | 1324473 |
| 18 | 16 | Ar Ruways | 24.11028 | 52.73056 | AE | 16000 |
| 19 | 17 | Al Fujairah City | 25.11641 | 56.34141 | AE | 86512 |
| 20 | 18 | Al Ain City | 24.19167 | 55.76056 | AE | 55091 |
| 21 | 19 | Ajman City | 25.40177 | 55.47878 | AE | 490035 |
| 22 | 20 | Adh Dhayd | 25.28812 | 55.88157 | AE | 24716 |
| 23 | 21 | Abu Dhabi | 24.46667 | 54.36667 | AE | 603492 |
| 24 | 22 | Khalifah A City | 24.42588 | 54.605 | AE | 85374 |
| 25 | 23 | Bani Yas City | 24.30978 | 54.62944 | AE | 80498 |
| 26 | 24 | Musaffah | 24.35893 | 54.48267 | AE | 243341 |
| 27 | 25 | Al Shamkhah City | 24.39268 | 54.70779 | AE | 61710 |
| 28 | 26 | Reef Al Fujairah City | 25.14479 | 56.24764 | AE | 82310 |
| 29 | 27 | Abu Dhabi Island and Internal Islands City | 24.4511 | 54.3969 | AE | 552215 |
| 30 | 28 | Kuhsān | 34.65389 | 61.19778 | AF | 12087 |
| 31 | 29 | Tukzār | 35.94831 | 66.42132 | AF | 12021 |



Le séparateur est le point virgule.
Les descripteurs sont :

- `id` : un numéro d'identification
- `name` : le nom d'une ville
- `latitude` : la latitude de la ville
- `longitude` : la longitude de la ville
- `country` : le pays où se situe la ville suivant un code à deux lettres
- `population` : le nombre d'habitants dans la ville



| | iso | name | area | population | continent | currency_code | currency_name | capital |
|---|---|---|---|---|---|---|---|---|
| 2 | AD | Andorra | 468.0 | 84000 | EU | EUR | Euro | 6 |
| 3 | AE | United Arab Emirates | 82880.0 | 4975593 | AS | AED | Dirham | 21 |
| 4 | AF | Afghanistan | 647500.0 | 29121286 | AS | AFN | Afghani | 81 |
| 5 | AG | Antigua and Barbuda | 443.0 | 86754 | NA | XCD | Dollar | 119 |
| 6 | AI | Anguilla | 102.0 | 13254 | NA | XCD | Dollar | 126 |
| 7 | AL | Albania | 28748.0 | 2986952 | EU | ALL | Lek | 157 |
| 8 | AM | Armenia | 29800.0 | 2968000 | AS | AMD | Dram | 191 |
| 9 | AO | Angola | 1246700.0 | 13068161 | AF | AOA | Kwanza | 235 |
| 10 | AR | Argentina | 2766890.0 | 41343201 | SA | ARS | Peso | 382 |
| 11 | AS | American Samoa | 199.0 | 57881 | OC | USD | Dollar | 704 |
| 12 | AT | Austria | 83858.0 | 8205000 | EU | EUR | Euro | 716 |
| 13 | AU | Australia | 7686850.0 | 21515754 | OC | AUD | Dollar | 1477 |
| 14 | AW | Aruba | 193.0 | 71566 | NA | AWG | Guilder | 2377 |
| 15 | AZ | Azerbaijan | 86600.0 | 8303512 | AS | AZN | Manat | 2482 |
| 16 | BA | Bosnia and Herzegovina | 51129.0 | 4590000 | EU | BAM | Marka | 2527 |
| 17 | BB | Barbados | 431.0 | 285653 | NA | BBD | Dollar | 2618 |
| 18 | BD | Bangladesh | 144000.0 | 156118464 | AS | BDT | Taka | 2666 |
| 19 | BE | Belgium | 30510.0 | 10403000 | EU | EUR | Euro | 3108 |
| 20 | BF | Burkina Faso | 274200.0 | 16241811 | AF | XOF | Franc | 3197 |
| 21 | BG | Bulgaria | 110910.0 | 7148785 | EU | BGN | Lev | 3257 |
| 22 | BH | Bahrain | 665.0 | 738004 | AS | BHD | Dinar | 3369 |
| 23 | BI | Burundi | 27830.0 | 9863117 | AF | BIF | Franc | 3379 |
| 24 | BJ | Benin | 112620.0 | 9056010 | AF | XOF | Franc | 3395 |
| 25 | BL | Saint Barthelemy | 21.0 | 8450 | NA | EUR | Euro | 3423 |
| 26 | BM | Bermuda | 53.0 | 65365 | NA | BMD | Dollar | 3424 |
| 27 | BN | Brunei | 5770.0 | 395027 | AS | BND | Dollar | 3429 |
| 28 | BO | Bolivia | 1098580.0 | 9947418 | SA | BOB | Boliviano | 3445 |
| 29 | BR | Brazil | 8511965.0 | 201103330 | SA | BRL | Real | 5169 |
| 30 | BS | Bahamas | 13940.0 | 301790 | NA | BSD | Dollar | 5411 |
| 31 | BT | Bhutan | 47000.0 | 699847 | AS | BTN | Ngultrum | 5432 |



Le séparateur est le point virgule.
Les descripteurs sont :

- `iso` : un code d'identification à 2 lettres
- `name` : le nom du pays
- `area` : la superficie du pays
- `population` : le nombre d'habitants dans le pays
- `continent` : le continent où se situe le pays suivant un code à deux lettres
- `currency_code` : un code à trois lettres pour la monnaie officiel du pays
- `currency_name` : le nom de la monnaie officiel du pays
- `capital` : le numéro d'identification de la capital donné dans le fichier `cities.csv`

# IV. Indexation de tables

## a) Importer une table (rappels)

1. Première méthode : on fait tout à la main, étape après étape.
   Cette étape, un peu plus laborieuse, nous permet néanmoins de comprendre comment est créé le fichier **CSV**.

```
# -*- coding : utf-8 -*-
import codecs #pour lire en UTF8, n'est pas toujours nécessaire

#ouverture du fichier
fichier = codecs.open("countries.csv", 'r', 'utf-8')

#on crée la table de données
pays = []

#on lit le fichier pour transférer les données dans la table
for ligne in fichier:
    #on enlève le retour à la ligne \r et la nouvelle ligne \n
    traitement = ligne.replace('\r\n','')
    #on récupère chaque données de la ligne séparées par une virgule sous la
    forme d'un tableau
    traitement = traitement.split(';')
    #on rajoute ce tableau à notre table
    pays.append(traitement)

#fermeture du fichier
fichier.close()

#pour vérification on affiche les descripteurs
print(pays[0])
#pour afficher les données du premier pays
print(pays[1])
```

2. Deuxième méthode : on peut utiliser le module `csv` qui permet de faire le travail directement.
   C'est beaucoup plus rapide, mais aucune connaissance n'est exigible sur ce module.

```
# -*- coding : utf-8 -*-
import codecs #pour lire en UTF8, n'est pas toujours nécessaire
import csv

pays = []

with codecs.open('countries.csv', 'r', 'utf-8') as fichier:
    csvpays = csv.reader(fichier, delimiter=';')
    for ligne in csvpays:
        pays.append(ligne)

#pour vérification on affiche les descripteurs
print(pays[0])
#pour afficher les données du premier pays
print(pays[1])
```

Dans les deux cas, nous obtenons le même résultat dans la console :

```
['iso', 'name', 'area', 'population', 'continent', 'currency_code', 'currency_name'
    , 'capital']
['AD', 'Andorra', '468.0', '84000', 'EU', 'EUR', 'Euro', '6']
```

### b) Stocker les données sous la forme d'un dictionnaire

On s'aperçoit que lors de la lecture du fichier, la première ligne n'a pas été utilisée comme descriptions des champs et le premier enregistrement, concernant **Andorre**, apparaît dans `pays[1]`. Plus gênant, le lien entre les valeurs du tableau `pays[1]` et le nom des enregistrements, contenus dans `pays[0]` , n'est pas direct.
Pour y remédier, nous allons utiliser `DictReader` qui retourne un dictionnaire pour chaque enregistrement, la première ligne étant utilisée pour nommer les différents champs.

```
# -*- coding : utf-8 -*-
import codecs #pour lire en UTF8, n'est pas toujours nécessaire
import csv

pays = []

with codecs.open('countries.csv', 'r', 'utf-8') as fichier:
    csvpays = csv.DictReader(fichier, delimiter=';')
    for ligne in csvpays:
        pays.append(dict(ligne))

#pour vérification on affiche le premier pays
print(pays[0])
```

Cette fois, on obtient un tableau de $p$-uplets représentés sous forme de dictionnaire.

```
{'iso': 'AD', 'name': 'Andorra', 'area': '468.0', 'population': '84000', 'continent
    ': 'EU', 'currency_code': 'EUR', 'currency_name': 'Euro', 'capital': '6'}
```

# V. Rechercher dans une table

Nous avions déjà effectué quelques recherches simples dans la première partie de ce cours. Cette fois-ci nous allons effectuer des recherches un peu plus complexe comme ... quels sont les pays qui utilisent l'euro comme monnaie ?

1. Première méthode : on interroge chaque ligne du tableau et on enregistre les réponses positives dans un tableau.

```
reponse = []

for p in pays:
    if p['currency_code'] == 'EUR':
        reponse.append(p['name'])

#on affiche le résultat
print(reponse)
```

2. Deuxième méthode : on peut aussi effectuer exactement la même recherche en écrivant les instructions sur une seule ligne grâce à la méthode par compréhension.
(Cette méthode est conseillée puisque nous n'avons pas vraiment le droit d'utiliser la méthode `append()`.

```
reponse = [p['name'] for p in pays if p['currency_code'] == 'EUR']

#on affiche le résultat
print(reponse)
```

Dans les deux cas, nous obtenons la réponse suivante :

```
['Andorra', 'Austria', 'Belgium', 'Saint Barthelemy', 'Cyprus', 'Germany', 'Estonia
    ', 'Spain', 'Finland', 'France', 'French Guiana', 'Guadeloupe', 'Greece', '
    Ireland', 'Italy', 'Kosovo', 'Lithuania', 'Luxembourg', 'Latvia', 'Monaco', '
    Montenegro', 'Saint Martin', 'Martinique', 'Malta', 'Netherlands', 'Saint Pierre
     and Miquelon', 'Portugal', 'Reunion', 'Slovenia', 'Slovakia', 'San Marino', '
    Vatican', 'Mayotte']
```

Demandons maintenant quelles sont les monnaies qui s'appellent Dollar. On peut exécuter par compréhension avec la ligne suivante :

```
reponse = [p['currency_code'] for p in pays if p['currency_name'] == 'Dollar']

#on affiche le résultat
print(reponse)
```

Nous obtenons le résultat suivant :

```
['XCD', 'XCD', 'USD', 'AUD', 'BBD', 'BMD', 'BND', 'BSD', 'BZD', 'CAD', 'AUD', 'NZD'
    , 'AUD', 'XCD', 'USD', 'FJD', 'USD', 'XCD', 'USD', 'GYD', 'HKD', 'JMD', 'AUD', '
    XCD', 'KYD', 'XCD', 'LRD', 'USD', 'USD', 'XCD', 'NAD', 'AUD', 'AUD', 'NZD', 'NZD
    ', 'NZD', 'USD', 'USD', 'SBD', 'SGD', 'SRD', 'USD', 'USD', 'USD', 'TTD', 'AUD',
    'TWD', 'USD', 'XCD', 'USD', 'USD', 'ZWL']
```

On obtient une liste contenant 54 monnaies, mais avec beaucoup de répétitions. En effet, par exemple, beaucoup de pays ont pour monnaie national le dollar américain (USD), comme par exemple l'Équateur, Les USA, ... Il faudrait donc supprimer ces répétitions. C'est ce que nous pouvons faire en transformant notre tableau en un **ensemble**[1] avec la fonction `set()`.

```
reponse = set([p['currency_code'] for p in pays if p['currency_name'] == 'Dollar'])

#on affiche le résultat
print(reponse)
```

avec le résultat :

```
{'BBD', 'NZD', 'FJD', 'TTD', 'XCD', 'AUD', 'BZD', 'ZWL', 'NAD', 'SGD', 'BND', 'CAD'
    , 'TWD', 'BSD', 'JMD', 'KYD', 'SBD', 'GYD', 'BMD', 'HKD', 'SRD', 'USD', 'LRD'}
```

Pour transformer l'**ensemble** en un **tableau** (une liste en réalité), on utilise la fonction `list()`.

```
reponse = list(set([p['currency_code'] for p in pays if p['currency_name'] == '
    Dollar']))
```

avec le résultat :

```
['BBD', 'NZD', 'FJD', 'TTD', 'XCD', 'AUD', 'BZD', 'ZWL', 'NAD', 'SGD', 'BND', 'CAD'
    , 'TWD', 'BSD', 'JMD', 'KYD', 'SBD', 'GYD', 'BMD', 'HKD', 'SRD', 'USD', 'LRD']
```

1. Un ensemble est une autre structure, différente du tuple, du tableau ou du dictionnaire. Un ensemble est une structure proche de la structure mathématique que l'on appelle également « ensemble ». Il n'y a pas de notion d'ordre et les éléments appartenant à un ensemble n'apparaissent qu'une seule fois. Cette structure n'est pas étudiée en Première.

# VI. Tri d'une table

Pour exploiter les données, il peut être intéressant de les trier. Une utilisation possible est l'obtention du classement des entrées selon tel ou tel critère.

## a) Tri selon un unique critère

On ne peut pas directement trier le tableau pays car cela ne veut rien dire. Il faut indiquer selon quels critères on veut effectuer ce tri. Pour cela, on appelle la fonction `sorted()` ou la méthode `sort()` avec les arguments suivants :

- l'argument supplémentaire `key` qui est une fonction renvoyant la valeur utilisée pour le tri ;
- l'argument optionnel `reverse` pour faire le tri dans l'ordre décroissant, s'il s'agit d'un nombre ou dans l'ordre inverse alphabétique s'il s'agit d'une chaîne de caractères.

```
1 def  superficie(p):
2     return p['area']
3
4 pays.sort(key=superficie, reverse=True)
5
6 #on affiche le résultat
7 print(pays)
```

Nous constatons en regardant les 3 premiers pays qu'il y a un petit souci :

```
1 [{'iso': 'CA', 'name': 'Canada', 'area': '9984670.0', 'population': '33679000', '
     continent': 'NA', 'currency_code': 'CAD', 'currency_name': 'Dollar', 'capital':
     '5872'},
2 {'iso': 'KR', 'name': 'South Korea', 'area': '98480.0', 'population': '48422644', '
     continent': 'AS', 'currency_code': 'KRW', 'currency_name': 'Won', 'capital': '
     26314'},
3 {'iso': 'US', 'name': 'United States', 'area': '9629091.0', 'population': '
     310232863', 'continent': 'NA', 'currency_code': 'USD', 'currency_name': 'Dollar'
     , 'capital': '41896'},
4 ...]
```

La Corée du Sud apparaît en deuxième position, entre le Canada et les USA. La raison est que lors de l'import, tous les champs sont considérés comme des chaînes de caractères, et le tri utilise l'ordre du dictionnaire. Ainsi, de même que « aa » arrive avant « b », « 10 » arrive avant « 2 ». Cela apparaît ici en regardant les superficies qui commencent par 998, puis par 984, par 962, etc. Pour remédier à cela, on modifie la fonction de clé :

```
def  superficie(p):
    return float(p['area'])

pays.sort(key=superficie, reverse=True)

#on affiche le résultat
print(pays)
```

Les trois premiers pays obtenu sont alors :

```
[{'iso': 'RU', 'name': 'Russia', 'area': '17100000.0', 'population': '140702000', '
    continent': 'EU', 'currency_code': 'RUB', 'currency_name': 'Ruble', 'capital': '
    36497'},
{'iso': 'CA', 'name': 'Canada', 'area': '9984670.0', 'population': '33679000', '
    continent': 'NA', 'currency_code': 'CAD', 'currency_name': 'Dollar', 'capital':
    '5872'},
{'iso': 'US', 'name': 'United States', 'area': '9629091.0', 'population': '
    310232863', 'continent': 'NA', 'currency_code': 'USD', 'currency_name': 'Dollar'
    , 'capital': '41896'},
...]
```

### b) Tri selon plusieurs critères

Supposons maintenant que l'on veut trier les pays selon deux critères : tout d'abord le continent, puis le nom du pays. On peut faire cela en définissant une fonction de clé qui renvoie une paire (continent, pays) :

```
def  continent_nom(p):
    return p['continent'], p['name']

pays.sort(key=continent_nom)

#on affiche le résultat
print(pays)
```

Nous obtenons le résultat suivant :

```
[{'iso': 'DZ', 'name': 'Algeria', 'area': '2381740.0', 'population': '34586184', '
    continent': 'AF', 'currency_code': 'DZD', 'currency_name': 'Dinar', 'capital': '
    12101'},
 {'iso': 'AO', 'name': 'Angola', 'area': '1246700.0', 'population': '13068161', '
    continent': 'AF', 'currency_code': 'AOA', 'currency_name': 'Kwanza', 'capital':
    '235'},
{'iso': 'BJ', 'name': 'Benin', 'area': '112620.0', 'population': '9056010', '
    continent': 'AF', 'currency_code': 'XOF', 'currency_name': 'Fran
c', 'capital': '3395'},
...,
{'iso': 'ZW', 'name': 'Zimbabwe', 'area': '390580.0', 'population': '13061000', '
    continent': 'AF', 'currency_code': 'ZWL', 'currency_name': 'Dollar', 'capital':
    '49682'},
{'iso': 'AF', 'name': 'Afghanistan', 'area': '647500.0', 'population': '29121286',
    'continent': 'AS', 'currency_code': 'AFN', 'currency_name': 'Afghani', 'capital'
    : '81'},
...,
{'iso': 'VE', 'name': 'Venezuela', 'area': '912050.0', 'population': '27223228', '
    continent': 'SA', 'currency_code': 'VES', 'currency_name': 'Bolivar Soberano', '
    capital': '49144'}]
```

Cependant, dans ce tri, les deux critères ont été utilisés pour un ordre croissant. Supposons maintenant que l'on veuille trier les pays par continent et, pour chaque continent, avoir les pays par population décroissante. La méthode précédente n'est pas applicable, car on a utilisé une unique clé (composée de deux éléments) pour un tri croissant. À la place, nous allons procéder en deux étapes :

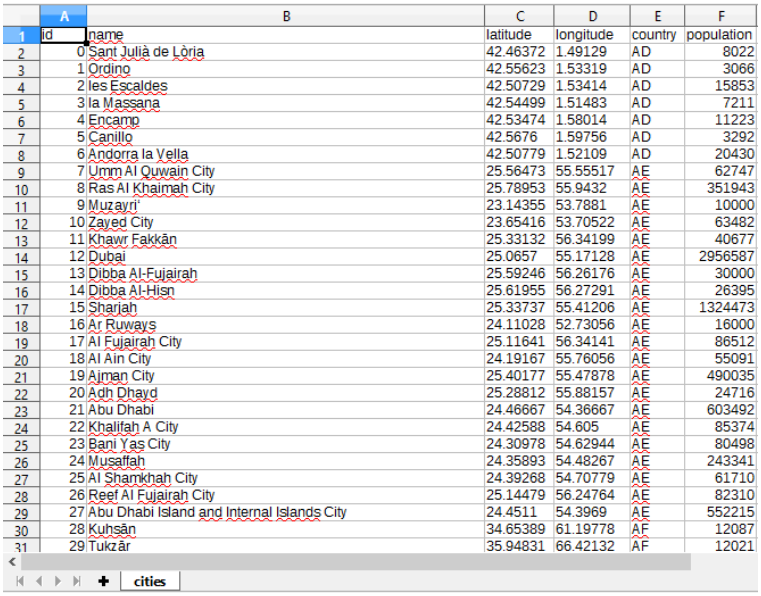
1. trier tous les pays par populations décroissantes ;
2. trier ensuite le tableau obtenu par continents croissants.

```
def population(p):
    return int(p['population'])

def  continent(p):
    return p['continent']

pays.sort(key=population, reverse=True)
pays.sort(key=continent)
```

Pour que cela soit possible, la fonction de tri de Python vérifie une propriété très importante : la **stabilité**. Cela signifie que lors d'un tri, si plusieurs enregistrement ont la même clé, l'ordre initial des enregistrements est conservé. Ainsi, si on a trié les pays par ordre décroissant de population puis par continent, les pays d'un même continent seront regroupés en conservant l'ordre précédent, ici la population décroissante.

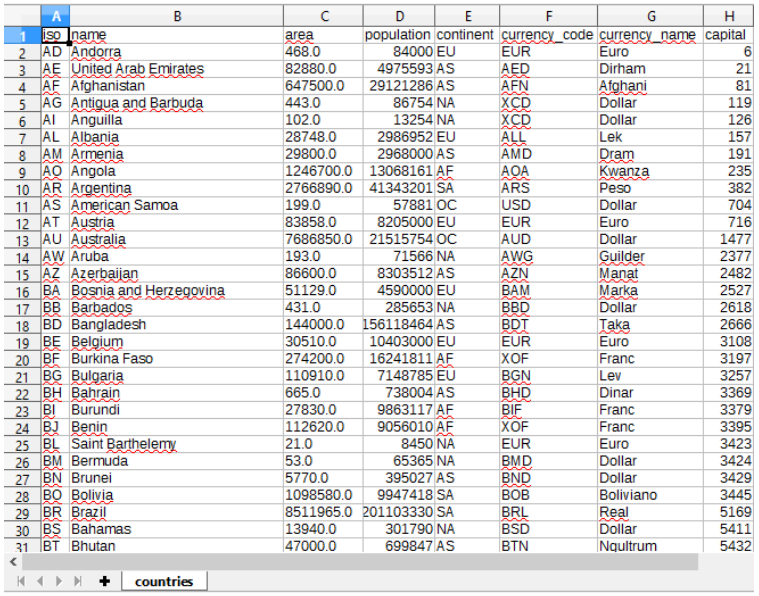
# VII. Fusion de tables

On dispose de deux tables comportant un champ en commun. Par exemple la table `pays` et la table `villes` issues des deux fichiers **CSV** présentés en début de chapitre :



| id | name | latitude | longitude | country | population |
|---|---|---|---|---|---|
| 0 | Sant Julià de Lòria | 42.46372 | 1.49129 | AD | 8022 |
| 1 | Ordino | 42.55623 | 1.53319 | AD | 3066 |
| 2 | les Escaldes | 42.50729 | 1.53414 | AD | 15853 |
| 3 | la Massana | 42.54499 | 1.51483 | AD | 7211 |
| 4 | Encamp | 42.53474 | 1.58014 | AD | 11223 |
| 5 | Canillo | 42.5676 | 1.59756 | AD | 3292 |
| 6 | Andorra la Vella | 42.50779 | 1.52109 | AD | 20430 |
| 7 | Umm Al Quwain City | 25.56473 | 55.55517 | AE | 62747 |
| 8 | Ras Al Khaimah City | 25.78953 | 55.9432 | AE | 351943 |
| 9 | Muzayri' | 23.14355 | 53.7881 | AE | 10000 |
| 10 | Zayed City | 23.65416 | 53.70522 | AE | 63482 |
| 11 | Khawr Fakkān | 25.33132 | 56.34199 | AE | 40677 |
| 12 | Dubai | 25.0657 | 55.17128 | AE | 2956587 |
| 13 | Dibba Al-Fujairah | 25.59246 | 56.26176 | AE | 30000 |
| 14 | Dibba Al-Hisn | 25.61955 | 56.27291 | AE | 26395 |
| 15 | Sharjah | 25.33737 | 55.41206 | AE | 1324473 |
| 16 | Ar Ruways | 24.11028 | 52.73056 | AE | 16000 |
| 17 | Al Fujairah City | 25.11641 | 56.34141 | AE | 86512 |
| 18 | Al Ain City | 24.19167 | 55.76056 | AE | 55091 |
| 19 | Ajman City | 25.40177 | 55.47878 | AE | 490035 |
| 20 | Adh Dhayd | 25.28812 | 55.88157 | AE | 24716 |
| 21 | Abu Dhabi | 24.46667 | 54.36667 | AE | 603492 |
| 22 | Khalifah A City | 24.42588 | 54.605 | AE | 85374 |
| 23 | Bani Yas City | 24.30978 | 54.62944 | AE | 80498 |
| 24 | Musaffah | 24.35893 | 54.48267 | AE | 243341 |
| 25 | Al Shamkhah City | 24.39268 | 54.70779 | AE | 61710 |
| 26 | Reef Al Fujairah City | 25.14479 | 56.24764 | AE | 82310 |
| 27 | Abu Dhabi Island and Internal Islands City | 24.4511 | 54.3969 | AE | 552215 |
| 28 | Kuhsān | 34.65389 | 61.19778 | AF | 12087 |
| 29 | Tukzār | 35.94831 | 66.42132 | AF | 12021 |

cities



| iso | name | area | population | continent | currency_code | currency_name | capital |
|---|---|---|---|---|---|---|---|
| AD | Andorra | 468.0 | 84000 | EU | EUR | Euro | 6 |
| AE | United Arab Emirates | 82880.0 | 4975593 | AS | AED | Dirham | 21 |
| AF | Afghanistan | 647500.0 | 29121286 | AS | AFN | Afghani | 81 |
| AG | Antigua and Barbuda | 443.0 | 86754 | NA | XCD | Dollar | 119 |
| AI | Anguilla | 102.0 | 13254 | NA | XCD | Dollar | 126 |
| AL | Albania | 28748.0 | 2986952 | EU | ALL | Lek | 157 |
| AM | Armenia | 29800.0 | 2968000 | AS | AMD | Dram | 191 |
| AO | Angola | 1246700.0 | 13068161 | AF | AOA | Kwanza | 235 |
| AR | Argentina | 2766890.0 | 41343201 | SA | ARS | Peso | 382 |
| AS | American Samoa | 199.0 | 57881 | OC | USD | Dollar | 704 |
| AT | Austria | 83858.0 | 8205000 | EU | EUR | Euro | 716 |
| AU | Australia | 7686850.0 | 21515754 | OC | AUD | Dollar | 1477 |
| AW | Aruba | 193.0 | 71566 | NA | AWG | Guilder | 2377 |
| AZ | Azerbaijan | 86600.0 | 8303512 | AS | AZN | Manat | 2482 |
| BA | Bosnia and Herzegovina | 51129.0 | 4590000 | EU | BAM | Marka | 2527 |
| BB | Barbados | 431.0 | 285653 | NA | BBD | Dollar | 2618 |
| BD | Bangladesh | 144000.0 | 156118464 | AS | BDT | Taka | 2666 |
| BE | Belgium | 30510.0 | 10403000 | EU | EUR | Euro | 3108 |
| BF | Burkina Faso | 274200.0 | 16241811 | AF | XOF | Franc | 3197 |
| BG | Bulgaria | 110910.0 | 7148785 | EU | BGN | Lev | 3257 |
| BH | Bahrain | 665.0 | 738004 | AS | BHD | Dinar | 3369 |
| BI | Burundi | 27830.0 | 9863117 | AF | BIF | Franc | 3379 |
| BJ | Benin | 112620.0 | 9056010 | AF | XOF | Franc | 3395 |
| BL | Saint Barthelemy | 21.0 | 8450 | NA | EUR | Euro | 3423 |
| BM | Bermuda | 53.0 | 65365 | NA | BMD | Dollar | 3424 |
| BN | Brunei | 5770.0 | 395027 | AS | BND | Dollar | 3429 |
| BO | Bolivia | 1098580.0 | 9947418 | SA | BOB | Boliviano | 3445 |
| BR | Brazil | 8511965.0 | 201103330 | SA | BRL | Real | 5169 |
| BS | Bahamas | 13940.0 | 301790 | NA | BSD | Dollar | 5411 |
| BT | Bhutan | 47000.0 | 699847 | AS | BTN | Ngultrum | 5432 |

countries

```python
# -*- coding : utf-8 -*-
import codecs #pour lire en UTF8, n'est pas toujours nécessaire
import csv

pays = []

with codecs.open('countries.csv', 'r', 'utf-8') as fichier:
    csvpays = csv.DictReader(fichier, delimiter=';')
    for ligne in csvpays:
        pays.append(dict(ligne))

villes = []

with codecs.open('cities.csv', 'r', 'utf-8') as fichier:
    csvvilles = csv.DictReader(fichier, delimiter=';')
    for ligne in csvvilles:
        villes.append(dict(ligne))

#pour vérification on affiche le premier pays
print(pays[0])
#puis la première ville
print(villes[0])
```

On obtient :

```
{'iso': 'AD', 'name': 'Andorra', 'area': '468.0', 'population': '84000', 'continent': 'EU', 'currency_code': 'EUR', 'currency_name': 'Euro', 'capital': '6'}
{'id': '0', 'name': 'Sant Julià de Lòria', 'latitude': '42.46372', 'longitude': '1.49129', 'country': 'AD', 'population': '8022'}
```

On aimerait pouvoir **fusionner** ces deux tableaux en créant un nouveau tableau en réunissant des informations communes. Par exemple, on pourrait choisir de rajouter au tableau **pays** des informations sur la capitale que nous irions récupérer dans le tableau **villes**. On appelle ce nouveau tableau une **jointure**. Pour fusionner les deux tables, il faudra :

- faire correspondre le champ `capital` de `pays` et le champ `id` de `villes`.
- faire aussi attention aux conflits que l'on pourra rencontrer entre les champs des deux tables qui comportent le même nom, comme par exemple `name` et `population`.

```
# -*- coding : utf-8 -*-
import codecs #pour lire en UTF8, n'est pas toujours nécessaire
import csv

pays = []

with codecs.open('countries.csv', 'r', 'utf-8') as fichier:
    csvpays = csv.DictReader(fichier, delimiter=';')
    for ligne in csvpays:
        pays.append(dict(ligne))

villes = []

with codecs.open('cities.csv', 'r', 'utf-8') as fichier:
    csvvilles = csv.DictReader(fichier, delimiter=';')
    for ligne in csvvilles:
        villes.append(dict(ligne))

##############################################
# fusion pays avec capital
##############################################
pays_capitales = []

for p in pays:
    #recherche du code capitale
    code_capitale = p['capital']
    #recherche de la capitale dans le tableau villes
    i = 0
    trouvee = False
    while i<len(villes) and (trouvee == False) :
        if villes[i]['id'] == code_capitale:
            trouvee = True
        else :
            i += 1
    #création d'un dictionnaire contenant les informations complètes du pays
    dict_pays_capi = {}
    for clef, valeur in p.items():
        dict_pays_capi[clef]=valeur
    #on rajoute au dictionnaire les informations de la capitale
    for clef, valeur in villes[i].items():
        #attention aux clefs déjà existante
        if clef == 'name':
            dict_pays_capi['name_cap']=valeur
        elif clef == 'population':
            dict_pays_capi['population_cap']=valeur
        elif (clef != 'id') and (clef != 'country'):
            #on ne veut pas d'information redondante
            dict_pays_capi[clef]=valeur
    #On rajoute ce dictionnaire à notre tableau pays_capitales
    pays_capitales.append(dict_pays_capi)

#pour vérifier
print(pays_capitales)
```

Avec le résultat suivant :

```
1 [{'iso': 'AD', 'name': 'Andorra', 'area': '468.0', 'population': '84000', '
      continent': 'EU', 'currency_code': 'EUR', 'currency_name': 'Euro', 'capital': '6
      ', 'name_cap': 'Andorra la Vella', 'latitude': '42.50779', 'longitude': '1.52109
      ', 'population_cap': '20430'},
2 {'iso': 'AE', 'name': 'United Arab Emirates', 'area': '82880.0', 'population': '
      4975593', 'continent': 'AS', 'currency_code': 'AED', 'currency_name': 'Dirham',
      'capital': '21', 'name_cap': 'Abu Dhabi', 'latitude':'24.46667', 'longitude': '
      54.36667', 'population_cap': '603492'},
3 {'iso': 'AF', 'name': 'Afghanistan', 'area': '647500.0', 'population': '29121286',
      'continent': 'AS', 'currency_code': 'AFN', 'currency_name': 'Afghani', 'capital'
      : '81', 'name_cap': 'Kabul', 'latitude': '34.52813', 'longitude': '69.17233', '
      population_cap': '3043532'},
4 {'iso': 'AG', 'name': 'Antigua and Barbuda', 'area': '443.0', 'population': '86754'
      , 'continent': 'NA', 'currency_code': 'XCD', 'currency_name': 'Dollar', 'capital
      ': '119', 'name_cap': "Saint John's", 'latitude': '17.12096', 'longitude': '
      -61.84329', 'population_cap': '24226'},
5 {'iso': 'AI', 'name': 'Anguilla', 'area': '102.0', 'population': '13254', '
      continent': 'NA', 'currency_code': 'XCD', 'currency_name': 'Dollar', 'capital':
      '126', 'name_cap': 'The Valley', 'latitude': '18.21704', 'longitude': '-63.05783
      ', 'population_cap': '2035'},
6 ...]
```

# VIII. Exercices

Pour les exercices 6 à 9 nous utiliserons la base de données sur les héros Marvel issues du site Kaggle : `https://www.kaggle.com/csanhueza/the-marvel-universe-social-network`.

Vous disposez de deux fichiers :

- `hero-network.csv` correspondant aux rencontres entre deux super-héros.
  Les deux descripteurs sont `hero1` et `hero2`
- `edges.csv` indique dans quel comics les super-héros sont apparus.
  Les deux descripteurs sont `hero` et `comic`

**Exercice 6 (Indexation de tables)**
Récupérer les deux information contenus dans ces fichiers en les plaçant dans deux tableaux `hero_network` et `edges`

**Exercice 7 (Rechercher dans une table)**

1. Trouver tous les super-héros qu'a rencontré mon ami Iron Man (Tony Starck).
2. Combien de fois Iron Man a rencontré Fortune, Dominic ?
3. Dans quels comics Iron Man et Fortune Dominic se sont rencontrés ?

**Exercice 8 (Tri d'une table)**
Classer la table `hero-network` dans l'ordre alphabétique du `hero1` puis du `hero2`.

**Exercice 9 (Recherche)**
Dans quels comics le Venom rencontre quelqu'un de la famille de Peter Parker, lui compris ?

Pour l'exercices 10 nous utiliserons une autre base de données sur les héros Marvel issues du site Kaggle : `https://www.kaggle.com/dannielr/marvel-superheroes`.

**Exercice 10 (Fusion de tables)**
Construire une table contenant tous les comics dans lesquels Kitty Pride est apparue.