# Numérique et science informatique
# Classe de première

Lycée hoche

année scolaire 2025-2026

# Contents

# 1 Introduction

Une des utilisations pratiques de l'informatique de nos jours est le traitement de quantités importantes de **données** dans des domaines variés:
Un site de commerce en ligne peut avoir à gérer une base de données pour des dizaines de milliers d'articles , de clients, de commandes.Un hôpital doit pouvoir accéder à tous les détails de traitements de ses patients etc.

Mais si les logiciels de traitement de base de données sont des programmes hautement spécialisés pour effectuer ce genre de tâches le plus efficacement possible , il est aisé de mettre en oeuvre les opérations de base dans un langage de programmation comme **P**ython.

# 2 Le format CSV

Le format *csv* ( *comma separated values* ) est un format très pratique pour représenter des données structurées.

**Dans ce format , chaque ligne représente un enregistrement et , sur une même ligne , les différents champs de l'enregistrement sont séparés par une virgule** (d'où le nom) .

En pratique , on peut spécifier le caractère utilisé pour séparer les différents champs et on utilise souvent pour ceci le point virgule, la virgule ,une tabulation ou les deux points.

Notons enfin que **la première ligne d'un tel fichier est souvent utilisée pour indiquer le nom des différents champs**.
Dans ce cas , le premier enregistrement apparaît en deuxième ligne du fichier.

Nous allons utiliser un fichier nommé *countries.csv* (Les données sont issues du site http://www.geonames.o et ont été légèrement modifiées.)

```
iso;name;Capital;Area;Population;Continent;currency_code;currency_name
AD;Andorra;Andorra la Vella ;468.0;77.006;EU;EUR;Euro
AE;United Arab Emirates;AbuDhabi;82.880;959;AS;AED;Dirham
AF;Afghanistan;Kabul;647500.37;386;AS;AFN;Afghani
AG;Antigua and Barbuda;St.John's;443;96.286;NA;XCD;Dollar
AI;Anguilla;The Valley;102.0;13.254;NA;XCD;Dollar
AL;Albania;Tirana;28.748;866.376;EU;ALL;Lek
CA;Canada;Ottawa;99846700;37058856;NA;CAD;Dollar
CN;China;Pekin;9596960.0;1392730000;AS;CNY;Yuan
FR;France;Paris;547030.0;66987244;EU;EUR;Euro
KR;South Korea;Seoul;98480.0;51635256;AS;KRW;Won
LS;Lesotho;Maseru;303550;2108132;AF;LSL;Loti
```

# 3 Importation des données

Une façon de charger un fichier *csv* est d'utiliser la bibliothèque de même nom (ou module):

```
import csv
pays=[ ]
with open('countries.csv',newline='') as csvfile:
    spamreader = csv.reader(csvfile,delimiter= ';')
    for row in spamreader:
        pays.append(row)
```

Par ce procédé , les données sont stockées dans un tableau.

```
>>> pays[0]
['iso',
'name',
'Capital',
'Area',
'Population',
'Continent',
'currency_code',
'currency_name']
```

On s'aperçoit que lors de la lecture du fichier , la première ligne n'a pas été utilisée comme description 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.

```
import csv
pays = [ ]
with open('countries.csv',newline='') as csvfile:
    spamreader = csv.DictReader(csvfile,delimiter= ';')
    for row in spamreader:
        pays.append(dict(row))
```

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

```
>>> pays[ 0]
{iso':AD,
'name':Andorra,
'Capital':Andorra la vella,
```

```
'Area' :468.0,
'Population':77.006,
'Continent':EU,
'currency_code':EUR,
'currency_name':Euro]
}
```

# 4 Exploitation des données

Nous allons donner deux types d'utilisation simples des données que l'on vient de charger:

Tout d'abord , l'interrogation des données pour récupérer telle ou telle information, puis le tri des données.

## 4.1 Interrogation

On peut traduire en `Python` des questions simples.Par exemple:

Quelles sont les types de pays où l'on paie en euros?

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

```
['Andorra','Austria','Belgium', ... , 'Vatican','Mayotte']
```

Quelles sont les monnaies qui s'appellent Dollar?

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

['XCD' , 'XCD' ,'USD', ...,'USD' , 'ZWL']

On obtient une liste (qui pourrait être assez longue avec beaucoup de répétitions)
Pour supprimer les répétitions, on peut utiliser la commande **set** pour transformer la liste en un ensemble **set**:

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

```
{'AUD','BDD' , ... , 'USD','XCD','ZWL'}
```

## 4.2 Tris-La méthode de listes *sort*-La fonction *sorted*

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.

- **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 l'argument supplémentaire **key** qui est une fonction renvoyant la valeur utilisée pour le tri:

  Par exemple , si on veut trier les pays par leur superficie , **on doit spécifier la clé** 'Area'.Pour ceci , on définit une fonction appropriée:

```
    def cle_superficie(p):
        return p['Area']
```

  Ainsi , pour classer les pays par superficie décroissante , on effectue:

```
>>>  pays.sort(key=cle_superficie , reverse = True)
```

  Un petit problème surgit.Si on récupère les cinq premiers noms , le résultat est surprenant

```
>>> [(p['name'] ,p['Area']) for p in pays  [:5]]

[('Canada','9984670'),
('South Korea' , '98480'),
('United states' , 9629091'),
('Netherlands Antilles'), '968')]
```

  On ne s'attend pas à trouver la Corée du sud parmi eux. La raison est que lors de l'import , tous les champs sont considérés comme des chaînes de caractère, 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 , puis par 962.

  Pour remédier à cela , on modifie la fonction de clé

```
    def cle_superficie(p):
        return float(p['Area'])
```

- **Tri selon plusieurs critères**

  Suppososons 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 cle_combinee(p):
        return (p['Continent'],p['name'])
```

```
>>> [(p['Continent'] , p['name'] ) for p in  sorted(pays, key = cle_combinee)]
```

[('AF', 'Lesotho'),
('AS', 'Afghanistan'),
('AS', 'China'),
('AS', 'South Korea'),
('AS', 'United Arab Emirates'),
...
('NA', 'Antigua and Barbuda'),
('NA', 'Canada')]

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.
A 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:

Ainsi:

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


def cle_continent(p):
    return int(p['Continent'])
```

```
>>> pays.sort(key=cle_population , reverse = True)
>>> pays.sort(key = cle_continent)
>>> [(p['Continent'] , p['name'],p['population'] ) for p in pays]
```

[('AF', 'Lesotho', '2108132'),
('AS', 'United Arab Emirates', '959'),
('AS', 'South Korea', '51.635.256'),
('AS', 'Afghanistan', '386'),
('AS', 'China', '1.392730'),
('EU', 'Albania', '866376'),_ ('EU', 'Andorra', '77006'),
('EU', 'France', '66987244'),
('NA', 'Antigua and Barbuda', '96286'),
('NA', 'Canada', '37058856'),
('NA', 'Anguilla', '13254')]

Pour que cela soit possible, la fonction de tri de `Python` vérifie une propriété très importante : la **stabilité**.
Ceci signifie que lors d'un tri , si plusieurs enregistrements 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.

### 4.3 conclusion

Nous l'avons vu , il est assez facile d'écrire en python des commandes simples pour exploiter un ensemble de données.Cependant , une utilisation plus poussée va vite donner lieu à des programmes fastidieux.Nous présenterons plus loin la bibliothèque **Pandas** qui permet une gestion plus efficace de ce genre de traitement.

## 5 Manipulation de tables avec la bibliothèque Pandas

Notons pour commencer qu'il n'y a rien de magique dans la bibliothèque *Pandas* et que les commandes présentées précédemment continuent d'illustrer les traitements de départ.Seulement, avec *Pandas* la présentation des données permet de rendre le tout plus efficace.

### 5.1 Prise en main

On commence par charger le module:

```
>>> import pandas
```

La lecture d'un fichier csv se fait aisément grâce à la commande :

```
pays= pandas.read_csv("countries.csv", delimiter = ";", keep_default_na=False)
```

où on spécifie explicitement le caractère utilisé pour délimiter les champs du fichier, ici un point virgule.

Explorons le fichier *countries.csv* :

La bibliothèque *Pandas* propose quelques commandes utiles:

- **pays.head()** : affiche les premières entrées de la table.
- **pays.sample(7)**: affiche 7 enregistrements au hasard.
- **pays.columns** : retourne la liste des champs.
- **pays.dtypes** : affiche la liste des champs avec , à chaque fois , le type de données correspondant.
- **pays.describe()**

```
>>> pays.dtypes
iso               object
name              object
Capital           object
Area             float64
Population       float64
Continent         object
currency_code     object
currency_name     object
dtype: object
```

```
>>> pays.describe()
Area    Population
count  1.100000e+01  1.100000e+01
mean   1.009467e+07  1.434472e+07
std    2.990211e+07  2.503560e+07
min    2.874800e+01  1.392730e+00
25%    2.725000e+02  8.664600e+01
50%    9.848000e+04  8.663760e+02
75%    5.972652e+05  1.958349e+07
max    9.984670e+07  6.698724e+07
```

On remarque en particulier que *Pandas* a reconnu que les champs latitude et population correspondent à des données numériques et les traitent comme tels.

On peut également ne conserver que les champs qui nous intéressent.
**pays[ ['name' ,'Capital','Area']]**

## 5.2 Dataframes et series

**Dataframes**

Les tables lues dans les fichiers csv sont stockées par *Pandas* sous forme de *dataframes*.On parle également de p-uplets nommés.

Un exemple de Dataframe:

```
df = pandas.DataFrame(
{
"A": ["A0", "A1", "A2", "A3"],
"B": ["B0", "B1", "B2", "B3"],
"C": ["C0", "C1", "C2", "C3"],
"D": ["D0", "D1", "D2", "D3"],
},
index=[0, 1, 2, 3],
)

En sortie
>>>df
    A   B   C   D
0   A0  B0  C0  D0
1   A1  B1  C1  D1
2   A2  B2  C2  D2
3   A3  B3  C3  D3
```

Pour afficher la ligne numéro 2 (attention le premier enregistrement est sur la ligne numéro 0):

```
>>> df.loc[2]
A    A2
B    B2
C    C2
D    D2
Name: 2, dtype: object
```

Ainsi pour le fichier "countries.csv", par exemple, l'enregistrement numéro 5 s'obtient en exécutant:

```
>>> pays.loc[5]
iso                  AL
name            Albania
Capital          Tirana
Area             28.748
Population      866.376
Continent            EU
currency_code       ALL
currency_name       Lek
Name: 5, dtype: object
```

et son nom s'obtient comme un dictionnaire:

```
>>> pays.loc[5]['name']
'Albania'
```

Pour sélectionner les trois premières lignes du dataframe

```
>>> df.loc[0:2]
    A   B   C   D
1   A1  B1  C1  D1
2   A2  B2  C2  D2
3   A3  B3  C3  D3
```

Terminons ici en donnant la définition d'une *série*

**Série**

Une série est ce que l'on obtient à partir d'un dataframe en ne sélectionnant qu'un seul champ.

Par exemple,pour obtenir la série en sélectionnant le champs 'nom' dans le fichier *countries.csv'*:

```
>>>import pandas as pd
>>>pays=pd.read_csv('countries.csv', delimiter=";", keep_default_na=False)
>>>pays['name']
                     name
0                 Andorra
1     United Arab Emirates
2             Afghanistan
3      Antigua and Barbuda
4                Anguilla
5                 Albania
6                  Canada
7                   China
8                  France
9             South Korea
10                Lesotho
Name: name, dtype: object
```

Pour afficher les séries des champs 'nom' et 'currency_code':

```
>>>pays[['name','currency_code']]
                    name currency_code
0                Andorra           EUR
1   United Arab Emirates           AED
2            Afghanistan           AFN
3    Antigua and Barbuda           XCD
4               Anguilla           XCD
5                Albania           ALL
6                 Canada           CAD
7                  China           CNY
8                 France           EUR
9            South Korea           KRW
10               Lesotho           LSL
```

## 5.3 Interrogations simples

**Noms des pays où on paye en euros**

On sélectionne la bonne valeur de *currency_ code*.Ainsi:

**pays[pays.currency_code == 'EUR']**

Ensuite , on ne garde que les noms des pays ainsi obtenus:

**pays[pays.currency_code == 'EUR'].name**

**Codes des monnaies appelées Dollar**
on écrit:

**pays[pays.currency_name == 'Dollar'].currency_code.unique()**

Remarque:*La méthode unique s'applique à une série et non pas à un dataframe.*

## 5.4 Tris

Les méthodes *nlargest* et *nsmallest* permettent de déterminer les plus grands et les plus petits éléments selon un critère donné.Ainsi, pour obtenir les pays les plus grands en superficie et ceux les moins peuplés , on écrit:

```
pays.nlargest(10, 'Area')
pays.nsmallest(10, 'Population')
```

Le tri d'un dataframe s'effectue à l'aide de la méthode *sort_ values* , comme :

**pays.sort_values(by = 'Population')**

On peut trier selon plusieurs critères , en spécifiant éventuellement les monotonies.Ainsi , pour classer par continent puis par superficie décroissante (avec une sélection pertinente de champs):

**pays.sort_values(by=['Continent','Area'],**
**ascending=[True,False]) [['Continent','name','Area']]**

## 5.5 Manipulation de données

Regardons quelques méthodes de manipulation de tables qu'offre la bibliothèque Pandas.

### Création d'un nouveau champ

Il est facile de créer de nouveaux champs à partir d'anciens.Par exemple , pour calculer la densité de chaque pays , il suffit d'exécuter:

```
pays['density']=pays.Population / pays.Area
```

Les séries peuvent être utilisées avec le **module numpy**.Ainsi, on peut réaliser une carte des villes utilisant la projection de Mercator en effectuant:

```
villes['projection_y']=numpy.arcsinh(numpy.tan(villes.latitude * numpy.pi / 180))
villes.plot.scatter(x='longitude' , y = 'projection_y')
```

### Fusion de tables

Dans la table des pays, la capitale est indiquée par un numéro... qui correspond au champ id de la table des villes. Pour récupérer le nom de la capitale de chaque pays, nous allons fusionner les tables en effectuant une jointure. Ainsi, nous allons faire correspondre le champ capital de pays et le champ id de villes. Cela se fait à l'aide de la fonction merge :

```
pandas.merge(pays, villes, left_on='capital', right_on='id')
```

```
import pandas

df1=pandas.read_csv('auteurs.csv', delimiter=";", keep_default_na=False)
df2=pandas.read_csv('libri-bis.csv', delimiter=";", keep_default_na=False)

pandas.merge(df1,df2,left_on='auteur_id' , right_on='auteur')
```

Cependant en procédant ainsi , il va y avoir un conflit entre les champs des deux tables.Cela apparaît en listant les champs de la table obtenue:

```
>>> pandas.merge(pays,villes,left_on='Capital' , right_on='id').columns
Index(['iso',name_x','Area','population_x','continent','currency_code','currency_name',
'Capital',id','name_y','latitude','longitude','country','population_y'],dtype='object')
```

On voit que des tables initiales contiennent toutes deux des champs *name* et *population* , d'où les suffixes _x et _y pour marquer la référence à la première table initiale ou à la seconde. Pour rendre cela plus lisible , nous allons:

1. Ne garder que les colonnes de ville qui nous intéressent, ici l'identifiant et le nom; Renommer ces colonnes pour éviter les collisions avec les champs de pays

```
villes[['id' , 'name']].rename(columns={'id':'Capital' , 'name':'capital_name'})
```

Et c'est cette nouvelle table que nous allons fusionner avec la table *pays*(dont nous ne garderons pas toutes les colonnes non plus):

```
pays_et_capitales = pandas.merge(
pays[['isi','name','capital','Continent']],
villes[[ 'id','name']].rename(
columns={'id' :'capital','name':'capital_name'}),
on='Capital')
```

La liste des pays d'Océanie et leurs capitales s'obtient alors facilement:

```
pays_et_capitales[pays_et_capitales.continent == 'OC']
```