Fichiers CSV : Traitement des données en tables

1. Données en tables

Une des utilisations principales de l'informatique de nos jours est le **traitement de quantités importantes de données** dans des domaines très variés :

- un site de commerce en ligne peut avoir à gérer des bases données pour des dizaines de milliers d'articles en vente, de clients, de commandes ;
- un hôpital doit pouvoir accéder efficacement à tous les détails de traitements de ses patients ;
- un éditeur de jeux vidéo doit pouvoir stocker les informations relatives à chaque joueur et chaque élément du jeu ;
- etc …

De tels traitements requierent souvent des logiciels de **gestion de base de données (SGDB)**, qui sont des programmes hautement spécialisés pour effectuer ce genre de tâches le plus efficacement et sûrement possible. L'étude plus détaillée des SGBD est au programme de terminale. Il est cependant facile de mettre en œuvre les opérations de bases sur certaines structures dans un langage de programmation comme Python.

Il est en effet possible que de simples jeus de données soient **organisés en tables**. En informatique, une **table de données** correspondent à une liste de **p-uplets** nommés qui partagent les mêmes **descripteurs**.

Exemple

Par exemple on peut considérer la table suivante :

Nom	Rang	Couleur du sabre
Luke Skywalker	Apprenti/Maitre Jedi	Vert
Anakin Skywalker	Apprenti Jedi	Bleu
Yoda	Maitre Jedi	Vert
Conte Dooku	Seigneur Sith	Ecarlate
Darth Vader	Apprenti Sith	Ecarlate
Rey	Apprentie Jedi	Jaune

Chacune des lignes, appelée aussi **enregistrement**, est un **p-uplet nommé** de taille 3. Chaque enregistrement correspond donc à un **dictionnaire** en Python. La première ligne correspond à :

```
{'Nom' : 'Luke Skywalker', 'Rang' :'Apprenti/Maitre Jedi', 'Couleur du sabre' : 'Vert'}
```

2. Fichiers CSV

Le format **CSV** (pour *Comma Separated Values*, soit en français *valeurs séparées par des virgules*) 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).

Pour des raisons pratiques, il est possible en fait de spécifier le caractère de séparation entre chaque champ, qui peut donc être `:`, `;`, `/`, etc...

Nous allons dans la suite de cette partie utiliser un fichier nommé `countries.csv`, téléchargeable ici.

Exercice 1

Une fois le fichier téléchargé, vous pouvez l'ouvrir avec `Notepad ++`.

Enoncé

1. Quel est le symbole utilisé pour séparer les champs ?
2. Combien de champs différents sont présents et quesl sont leurs descripteurs ?
3. Il est aussi possible d'utiliser un **tableur** comme `LibreOffice.Calc` pour lire un fichier CSV. Vous pourrez constater que `LibreOffice.Calc` vous demande un certain nombre d'informations sur le contenu du fichier avant de l'ouvrir réellement. Quels intérêt voyez-vous à l'utilisation d'un tableur ? Quelles en sont les limites ?

Réponses

1. C'est le point-virgule
2. Il y a 8 champs dont les descripteurs sont :
   - ISO
   - Name
   - Capital_Id
   - Area
   - Population
   - Continent
   - Currency_Code
   - Currency_Name
3. Le tableur permet de visualiser rapidement les descripteurs et les données, quand elles ne sont pas trop nombreuses. Un tableur est limité a 1 000 000 de lignes, alors qu'un fichier CSV n'est pas limité (ce qui n'empêche pas des problèmes liées à la quantité de mémoire de l'ordinateur).

3. Module CSV

3.1. Lecture de base

Un fichier `csv` étant un fichier texte, il est tout à fait possible de lire les données grâce aux méthodes classiques des fichiers(voir ici) :

```
>>> with open('countries.csv','r',encoding='utf8',newline='') as file:
        contenu = file.readlines()
        print(contenu)
```

Le résultat est peu lisible, et difficilement exploitable ainsi. Mais il est tout à fait possible de reconstituer correctement chaque enregistrement, en utilisant des méthodes de chaines de caractères (*on se limite aux 5 premiers enregistrelments pour des raisons de lisbilités dans l'interpréteur*) :

```
>>> formatedContent=[]
>>> for line in contenu[ :5] :
        formatedContent.append(line.replace('\r\n','').split(';'))
```

On peut alors récupérer les **descripteurs** :

```
>>> formatedContent[0]
['ISO',
 'Name',
 'Capital_Id',
 'Area',
 'Population',
 'Continent',
 'Currency_Code',
 'Currency_Name']
```

Et le premier enregistrement :

```
>>> formatedContent[1]
['AD', 'Andorra', '3041563', '468', '84000', 'EU', 'EUR', 'Euro']
```

Cependant cette solution n'est pas des plus efficace, car **le lien entre descripteurs et valeurs n'est pas direct**.

3.2. Lire un fichier CSV, premiers traitements

Dans la suite du cours, il sera nécessaire d'adapter au fur et à mesure un fichier python !

Le module `csv` est un des modules présents dans toute installation Python. Comme tout module, il est disponible par import direct :

```
import csv
```

Une fois importé, il offre de nombreuses possibilités, décrites dans la doc Python, dont la méthode `DictReader` qui permet der récupérer les enregistrements sous la forme d'un **dictionnaire ordonné** :

```
countries = []
with open('countries.csv', 'r', encoding='utf8', newline='') as file :
    reader = csv.DictReader(file, delimiter=";") # L'objet reader est un itérable. On précise le delimiteur à
l'appel de DictReader
    for line in reader :
        countries.append(line)
```

La variable `countries` fait donc référence à une liste de dictionnaires, ayant tous les mêmes descripteurs. Une fois le programme exécuté, on peut donc tester dans le shell les comandes suivantes :

```
>>> countries[:5]
[{    [{'ISO': 'AD',
  'Name': 'Andorra',
  'Capital_Id': '3041563',
  'Area': '468',
  'Population': '84000',
  'Continent': 'EU',
  'Currency_Code': 'EUR',
  'Currency_Name': 'Euro'},
 {'ISO': 'AE',
  'Name': 'United Arab Emirates',
  'Capital_Id': '292968',
  'Area': '82880',
  'Population': '4975593',
  'Continent': 'AS',
  'Currency_Code': 'AED',
  'Currency_Name': 'Dirham'},
 {'ISO': 'AF',
  'Name': 'Afghanistan',
  'Capital_Id': '1138958',
  'Area': '647500',
  'Population': '29121286',
  'Continent': 'AS',
  'Currency_Code': 'AFN',
  'Currency_Name': 'Afghani'},
 {'ISO': 'AG',
  'Name': 'Antigua and Barbuda',
  'Capital_Id': '3576022',
  'Area': '443',
  'Population': '86754',
  'Continent': 'NA',
```

```
  'Currency_Code': 'XCD',
  'Currency_Name': 'Dollar'},
 {'ISO': 'AI',
  'Name': 'Anguilla',
  'Capital_Id': '3573374',
  'Area': '102',
  'Population': '13254',
  'Continent': 'NA',
  'Currency_Code': 'XCD',
  'Currency_Name': 'Dollar'}]
```

On peut bien entendu connaître le nombre d'enregistrements contenus dans la variable countries :

```
>>> len(countries)
243
```

On peut alors interroger cette variable à partir des descripteurs. Par exemple, nous pouvons chercher à obtenir la liste des pays dont la devise est en euro :

```
>>> [c['Name'] for c in countries if c['Currency_Code'] == 'EUR']
['Andorra',
 'Austria',
 'Aland Islands',
 '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',
 'French Southern Territories',
 'Vatican',
 'Mayotte']
```

Exercice

Enoncé

1. Écrire une compréhension de listes permettant de lister les codes de toutes les monnaies qui s'appellent `'Dollar'`.
2. Écrire une compréhension de listes permettant de lister les pays du continent Nord-Américain, sous la forme `Nom, Superficie, Population`

Réponses

1. Le code :

```
>>> [c['Currency_Code'] for c in countries if c['Currency_Name'] == 'Dollar']
['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']
```

2. Le code :

```
>>> [c['Name'] for c in countries if c['Continent']=='NA']
    ['Antigua and Barbuda',
 'Anguilla',
 'Aruba',
 'Barbados',
 'Saint Barthelemy',
 'Bermuda',
 'Bahamas',
 'Belize',
 'Canada',
 'Costa Rica',
 'Cuba',
 'Curacao',
 'Dominica',
 'Dominican Republic',
 'Grenada',
 'Greenland',
 'Guadeloupe',
 'Guatemala',
 'Honduras',
 'Haiti',
 'Jamaica',
 'Saint Kitts and Nevis',
 'Cayman Islands',
 'Saint Lucia',
 'Saint Martin',
 'Martinique',
 'Montserrat',
 'Mexico',
 'Nicaragua',
 'Panama',
 'Saint Pierre and Miquelon',
 'Puerto Rico',
 'El Salvador',
 'Sint Maarten',
 'Turks and Caicos Islands',
 'Trinidad and Tobago',
 'United States',
 'Saint Vincent and the Grenadines',
 'British Virgin Islands',
 'U.S. Virgin Islands']
```

3.3. Les données numériques

Comment faire pour lister les pays dont la superficie est inférieure à 300 km² ? On pourrait penser à écrire une compréhension de liste identique à celles écrites précédfemment :

```
>>> [(c['Name'],c['Area']) for c in countries if c['Area']<300]
    ---------------------------------------------------------------------------

    TypeError                                 Traceback (most recent call last)

    <ipython-input-13-6debe1b6bf11> in <module>
    ----> 1 [(c['Name'],c['Area']) for c in countries if c['Area']<300]


    <ipython-input-13-6debe1b6bf11> in <listcomp>(.0)
    ----> 1 [(c['Name'],c['Area']) for c in countries if c['Area']<300]


    TypeError: '<' not supported between instances of 'str' and 'int'
```

Mais nous avons un problème : les données extraites d'un fichier `csv` sont **toutes sous la forme de chaines de caractères**. Il faut donc **transtyper** certaines données pour parvenir au résultat escompté. D'après les 5 premiers enregistrements, la superficie semble être sous la forme d'un nombre entier, donc on teste :

```
>>> [(c['Name'],c['Area']) for c in countries if int(c['Area'])<300]


    ---------------------------------------------------------------------------

    ValueError                                Traceback (most recent call last)

    <ipython-input-14-d2574c3771d7> in <module>
    ----> 1 [(c['Name'],c['Area']) for c in countries if int(c['Area'])<300]


    <ipython-input-14-d2574c3771d7> in <listcomp>(.0)
    ----> 1 [(c['Name'],c['Area']) for c in countries if int(c['Area'])<300]


    ValueError: invalid literal for int() with base 10: '6.5'
```

Mais encore une fois un problème apparâit : **les données ne sont pas toutes transtypables sous la forme entière**, chose que nous ne pouvions savoir avant de traiter. Il faut donc les convertir en flottant :

```
>>> [(c['Name'],c['Area']) for c in countries if float(c['Area'])<300]
[('Anguilla', '102'),
     ('American Samoa', '199'),
     ('Aruba', '193'),
     ('Saint Barthelemy', '21'),
     ('Bermuda', '53'),
     ('Cocos Islands', '14'),
     ('Cook Islands', '240'),
     ('Christmas Island', '135'),
     ('Guernsey', '78'),
     ('Gibraltar', '6.5'),
     ('Jersey', '116'),
     ('Saint Kitts and Nevis', '261'),
     ('Cayman Islands', '262'),
     ('Liechtenstein', '160'),
     ('Monaco', '1.95'),
     ('Saint Martin', '53'),
     ('Marshall Islands', '181.3'),
     ('Macao', '254'),
     ('Montserrat', '102'),
     ('Norfolk Island', '34.6'),
     ('Nauru', '21'),
     ('Niue', '260'),
     ('Saint Pierre and Miquelon', '242'),
     ('Pitcairn', '47'),
     ('San Marino', '61.2'),
     ('Sint Maarten', '21'),
     ('Tuvalu', '26'),
```

```
     ('Vatican', '0.44'),
     ('British Virgin Islands', '153'),
     ('Wallis and Futuna', '274')]
```

Exercice

Enoncé

Écrire une compréhension de liste donnant le nom, le continent et la superficie des pays de plus de 100 millions d'habitants

Réponse

```
>>> [(c['Name'],c['Continent'],c['Area']) for c in countries if float(c['Population'])>10**8]
[('Bangladesh', 'AS', '144000'),
('Brazil', 'SA', '8511965'),
('China', 'AS', '9596960'),
('Indonesia', 'AS', '1919440'),
('India', 'AS', '3287590'),
('Japan', 'AS', '377835'),
('Mexico', 'NA', '1972550'),
('Nigeria', 'AF', '923768'),
('Pakistan', 'AS', '803940'),
('Russia', 'EU', '17100000'),
('United States', 'NA', '9629091')]
```

3.4. Trier les données

Les résultats precédents sont encore peu lisibles, il serait préférable que les données obtenues soient triées. On va utiliser la fonction *built-in* `sorted` :

```
>>> sorted([(c['Name'],c['Area']) for c in countries if float(c['Area'])<300])
    [('American Samoa', '199'),
     ('Anguilla', '102'),
     ('Aruba', '193'),
     ('Bermuda', '53'),
     ('British Virgin Islands', '153'),
     ('Cayman Islands', '262'),
     ('Christmas Island', '135'),
     ('Cocos Islands', '14'),
     ('Cook Islands', '240'),
     ('Gibraltar', '6.5'),
     ('Guernsey', '78'),
     ('Jersey', '116'),
     ('Liechtenstein', '160'),
     ('Macao', '254'),
     ('Marshall Islands', '181.3'),
     ('Monaco', '1.95'),
     ('Montserrat', '102'),
     ('Nauru', '21'),
     ('Niue', '260'),
     ('Norfolk Island', '34.6'),
     ('Pitcairn', '47'),
     ('Saint Barthelemy', '21'),
     ('Saint Kitts and Nevis', '261'),
     ('Saint Martin', '53'),
     ('Saint Pierre and Miquelon', '242'),
     ('San Marino', '61.2'),
     ('Sint Maarten', '21'),
     ('Tuvalu', '26'),
     ('Vatican', '0.44'),
     ('Wallis and Futuna', '274')]
```

On obtient bien une liste triée, mais par ordre alphabétique, ce qui est peu pertinent. En effet, pour trier une table, il faut préciser **selon quels critères**.

On va donc passer un argument supplémentaire à la fonction `sorted` : une **clé de tri**. Dans Python, la clé **doit être une fonction** permettant d'extraire la valeur à trier. On va donc créer une fonction qui va renvoyer la valeur voulue selon l'enrtegistrement passé en

paramètre. Ici on va passer un **tuple** `(Nom;Superficie)` et on veut donc trier selon l'élément d'indice 1 :

```
def areaKey(c):
    return float(c[1])
```

Puis on va passer cette fonction à la fonction `sorted` :

```
>>> sorted([(c['Name'],c['Area']) for c in countries if float(c['Area'])<300],key = areaKey)
[('Vatican', '0.44'),
     ('Monaco', '1.95'),
     ('Gibraltar', '6.5'),
     ('Cocos Islands', '14'),
     ('Saint Barthelemy', '21'),
     ('Nauru', '21'),
     ('Sint Maarten', '21'),
     ('Tuvalu', '26'),
     ('Norfolk Island', '34.6'),
     ('Pitcairn', '47'),
     ('Bermuda', '53'),
     ('Saint Martin', '53'),
     ('San Marino', '61.2'),
     ('Guernsey', '78'),
     ('Anguilla', '102'),
     ('Montserrat', '102'),
     ('Jersey', '116'),
     ('Christmas Island', '135'),
     ('British Virgin Islands', '153'),
     ('Liechtenstein', '160'),
     ('Marshall Islands', '181.3'),
     ('Aruba', '193'),
     ('American Samoa', '199'),
     ('Cook Islands', '240'),
     ('Saint Pierre and Miquelon', '242'),
     ('Macao', '254'),
     ('Niue', '260'),
     ('Saint Kitts and Nevis', '261'),
     ('Cayman Islands', '262'),
     ('Wallis and Futuna', '274')]
```

Fonctions anonymes

Exercice

Enoncé

1. Ecrire un code qui donne la liste des 5 états ayant la plus grande superficie, sous la forme `(Nom, Superficie)`
2. Ecrire un code qui donne la liste des 5 états ayant la plus petite superficie parmi les 20 états ayant la plus grande population, sous la forme `(Nom, population)`.
3. Écrire les instructions permettant de d'afficher les 8 pays possédant la plus grande densité de population, dans l'ordre inverse de densité, sous la forme `(Pays, population,superficie, densité)`.

Réponses

1. Le code :

```
>>> sorted([(c['Name'],c['Area']) for c in countries],key = lambda x : float(x[1]),reverse =True)[:5]
[('Russia', '17100000'),
     ('Canada', '9984670'),
     ('United States', '9629091'),
     ('China', '9596960'),
     ('Brazil', '8511965')]
```

1. Le code :

```
>>> sorted( sorted(countries,key = lambda x: float(x['Population']),reverse=True )[:20],key=lambda x
:x['Area'])[:5]
        [{'ISO': 'EG',
  'Name': 'Egypt',
  'Capital_Id': '360630',
  'Area': '1001450',
  'Population': '80471869',
  'Continent': 'AF',
  'Currency_Code': 'EGP',
  'Currency_Name': 'Pound'},
 {'ISO': 'ET',
  'Name': 'Ethiopia',
  'Capital_Id': '344979',
  'Area': '1127127',
  'Population': '88013491',
  'Continent': 'AF',
  'Currency_Code': 'ETB',
  'Currency_Name': 'Birr'},
 {'ISO': 'BD',
  'Name': 'Bangladesh',
  'Capital_Id': '1185241',
  'Area': '144000',
  'Population': '156118464',
  'Continent': 'AS',
  'Currency_Code': 'BDT',
  'Currency_Name': 'Taka'},
 {'ISO': 'IR',
  'Name': 'Iran',
  'Capital_Id': '112931',
  'Area': '1648000',
  'Population': '76923300',
  'Continent': 'AS',
  'Currency_Code': 'IRR',
  'Currency_Name': 'Rial'},
 {'ISO': 'RU',
  'Name': 'Russia',
  'Capital_Id': '524901',
  'Area': '17100000',
  'Population': '140702000',
  'Continent': 'EU',
  'Currency_Code': 'RUB',
  'Currency_Name': 'Ruble'}]
```

2. Le code :

```
>>> sorted([(c['Name'],c['Area'], float(c['Population'])/float(c['Area'])) for c in countries], key =
lambda x : x[2], reverse=True)[:8]
[('Monaco', '1.95', 16905.128205128207),
 ('Singapore', '692.7', 6786.5872672152445),
 ('Hong Kong', '1092', 6317.478021978022),
 ('Gibraltar', '6.5', 4289.846153846154),
 ('Vatican', '0.44', 2093.181818181818),
 ('Sint Maarten', '21', 1782.3333333333333),
 ('Macao', '254', 1768.4960629921259),
 ('Maldives', '300', 1318.8333333333333)]
```

3.5. Fusions de tables

Le fichier `cities.csv` téléchargeable ici contient une table des principales villes au niveayu mondial.

Exercice : exploration du fichier `cities.csv`

Enoncé

1. Quels sont les descripteurs de ce fichier ?
2. Ecrire un code sauvant dans une variable `cities` les enregistrements du fichiers `cities.csv`.
3. Ecrire une compréhension de liste donnant les villes françaises sous la forme `(Nom, population)`

Réponses

1. `Id;Name;Latitude;Longitude;Country_ISO;Population`
2. Le code :

```
cities = []
with open('cities.csv', 'r', encoding='utf8', newline='') as file :
    reader = csv.DictReader(file, delimiter=";")
    for line in reader :
        cities.append(line)
```

3. Le code :

```
[c for c in cities if c['Country_ISO']=='FR']
```

Un des descripteurs est commun à la fois au fichier `countries.csv` et au fichier `cities.csv`. Il s'agit, dans `countries.csv` de `Capital_Id`, qui correspond au descripteur `Id` de `cities.csv`. Il est donc possible de joindre des données des deux tables, comme dans l'exemple ci-dessous :

```
>>> [(co['Name'],co['Population'],co['Area'],ci['Name'], ci['Population']
) for co in countries for ci in cities if int(co['Population'])> 50*10**6 and co['Capital_Id']==ci['Id'] ]

    [('Bangladesh', '156118464', '144000', 'Dhaka', '10356500'),
     ('Brazil', '201103330', '8511965', 'Brasília', '2207718'),
     ('Democratic Republic of the Congo', '70916439', '2345410', 'Kinshasa', '7785965'),
     ('China', '1330044000', '9596960', 'Beijing', '11716620'),
     ('Germany', '81802257', '357021', 'Berlin', '3426354'),
     ('Egypt', '80471869', '1001450', 'Cairo', '7734614'),
     ('Ethiopia', '88013491', '1127127', 'Addis Ababa', '2757729'),
     ('France', '64768389', '547030', 'Paris', '2138551'),
     ('United Kingdom', '62348447', '244820', 'London', '7556900'),
     ('Indonesia', '242968342', '1919440', 'Jakarta', '8540121'),
     ('India', '1173108018', '3287590', 'Delhi', '10927986'),
     ('Iran', '76923300', '1648000', 'Tehran', '7153309'),
     ('Italy', '60340328', '301230', 'Rome', '2318895'),
```

```
('Japan', '127288000', '377835', 'Tokyo', '8336599'),
('Myanmar', '53414374', '678500', 'Nay Pyi Taw', '925000'),
('Mexico', '112468855', '1972550', 'Mexico City', '12294193'),
('Nigeria', '154000000', '923768', 'Abuja', '590400'),
('Philippines', '99900177', '300000', 'Manila', '1600000'),
('Pakistan', '184404791', '803940', 'Islamabad', '601600'),
('Russia', '140702000', '17100000', 'Moscow', '10381222'),
('Thailand', '67089500', '514000', 'Bangkok', '5104476'),
('Turkey', '77804122', '780580', 'Ankara', '3517182'),
('United States', '310232863', '9629091', 'Washington, D.C.', '601723'),
('Vietnam', '89571130', '329560', 'Hanoi', '1431270')]
```

Exercice

Enoncé

1. Ecrire une compréhension donnant les pays dont la population de la capitale est supérieure à 100 000.
2. Ecrire un code donnant les pays dont la capitale possède une latitude supérieure à 60°.
3. Ecrire un code donnant les pays dont la capitale possède une latitude inférieure à -50°.

Réponses

1. Le code :

```
>>> [co['Name'] for co in countries for ci in cities if co['Capital_Id']==ci['Id'] and
int(ci['Population'])> 10**5]
```

(La sortie est bien trop longue pour être affichée ici)

2. Le code :

```
>>> [(co['Name'],co['Population'],co['Area'],ci['Name'], ci['Population']) for co in countries for ci in
cities if co['Capital_Id']==ci['Id'] and float(ci['Latitude'])> 60]
[('Aland Islands', '26711', '1580', 'Mariehamn', '10682'),
('Finland', '5244000', '337030', 'Helsinki', '558457'),
('Faroe Islands', '48228', '1399', 'Tórshavn', '13200'),
('Greenland', '56375', '2166086', 'Nuuk', '14798'),
('Iceland', '308910', '103000', 'Reykjavík', '118918'),
('Svalbard and Jan Mayen', '2550', '62049', 'Longyearbyen', '2060')]
```

3. Le code :

```
>>> [(co['Name'],co['Population'],co['Area'],ci['Name'], ci['Population']) for co in countries for ci in
cities if co['Capital_Id']==ci['Id'] and float(ci['Latitude'])<- 50]
[('Falkland Islands', '2638', '12173', 'Stanley', '2213'),
('South Georgia and the South Sandwich Islands',
'30',
'3903',
'Grytviken',
'2')]
```

Pour les cracks : Attention !

La technique de fusion montrée ci-dessus n'est vraiment pas très efficace ! En effet il s'agit d'un parcours double de boucle : à chaque tour de la boucle parcourant `countries`, on parcoure **toute la liste** `cities`.

C'est absolument horrible !

En effet, si `countries` est une liste de longueur $1\ 000\ 000$ et que `cities` est une liste de taille $1\ 000$, alors le nombre de comparaison sera de $1\ 000\ 000 \times 1\ 000 = 1\ 000\ 000\ 000$!

En termes mathématiques, si `countries` est de taille n et `cities` de taille m, alors cet algorithme aura un coût temporel en $\mathbb{O}(n \times m)$.

Pour accélérer une telle fusion de liste, il peut-être judicieux de transformer ces listes en des dictionnaires, **avec un choix judicieux de clés !**.

Par exemple, ici on pourrait transformer la liste `cities` en un dictionnaire dont la clé est l'`Id`, puis effectuer une boucle sur la liste.

```
dicCities = {}
for ci in cities :
        dicCities[ci['Id']] = ci
result = []
for co in countries :
    if int(co['Population'])> 50*10**6 :
        result.append([(co['Name'],
                        co['Population'],
                        co['Area'],
                        dicCities[co['Capital_Id']]['Name'],
                        dicCities[co['Capital_Id']]['Population']))
```

Certes le code paraît plus complexe. Mais en terme de coût temporel :

- Il n'y a qu'un seul parcours de la liste `cities` ;
- A chaque tour de boucle parcourant `countries`, on ne fera appel qu'à des actions en **temps constant** (en $\mathbb{O}(1)$) sur le dictionnaire `dicCities`.

En reprenant les données précédentes :

- on fait $1\ 000$ tours de boucles pour la construction du dictionnaire ;
- on fait $1\ 000\ 000$ de tours de boucles sur la liste `countries`.

En ordre de grandeur, on reste sur environ $1\ 000\ 000$ d'opérations, soit $1\ 000$ fois moins que précédemment !

En termes mathématiques, le coût en temps est $\mathbb{O}(max(m,n))$.

Pour être encore plus clair, voici un ordre de comparaison en temps dépendant des tailles n et m des deux listes fusionnées, le contenu d'une case donne la comparaison en temps entre $\mathbb{O}(m \times n)$ et $\mathbb{O}(max(m,n))$, avec le principe de $1\ 000$ opérations par secondes.

n\m	1	10^3	10^6	10^9
1	$0,001'' \ / \ 0,001''$	$1'' \ / \ 1''$	$16'40'' \ / \ 16'40''$	$11j13h46'40'' \ / \ 11j13h46'40''$
10^3	...	$16'40'' \ / \ 1''$	$277h46'40'' \ / \ 16'40''$	$\simeq 31\ ans / \ 11j13h46'40''$
10^6	...	...	$\simeq 31\ ans / \ 16'40''$	$\simeq 31\ 000\ années \ / \ 11j13h46'40''$
10^9	...	...	...	$\simeq 31\ 000\ 000\ années \ / \ 11j13h46'40''$

Il y a 30 millions d'années, apparaissaient seulement les grands singes, l'Himalaya n'était pas totalement formé...

Oui, direz-vous, mais les ordinateurs actuels sont 1 million de fois plus rapides !

Certes, mais alors combien de temps faudrait-il dans les deux cas pour traiter deux listes de 1 milliard de données chacune ?

3.6. Ecrire un nouveau fichier csv

Après avoir extrait, modifié et fusionné des fichiers de type `.csv`, il est normal de vouloir en créer de nouveaux à partir de données existantes. Le module `csv` propose une méthode `.DictWriter` dont l'utilisation est la suivante :

```
with open("capitales.csv", "w", encoding="utf8",newline="") as file :
    ##On definit les descripteurs du nouveau fichier
    descripteurs = ['Pays', 'Population_pays', 'Superficie', 'Densite', 'Capitale', 'Population_capitale']
    writer = csv.DictWriter(file, fieldnames=descripteurs,delimiter=";")

    writer.writeheader()##Ecrit les descripteurs en première ligne
    for co in countries :
        for ci in cities :
            if co['Capital_Id'] == ci['Id'] :
                writer.writerow({'Pays' : co['Name'],
                                 'Population_pays' : co['Population'],
                                 'Superficie' : co['Area'],
                                 'Densite' : float(co['Population'])/float(co['Area']),
                                 'Capitale' : ci['Name'],
                                 'Population_capitale' : ci['Population']

                                }
                )
```

4. TP : Explorer IMDB

Le TP se fait sur Capytale par ce lien.