éduSCOL
Informer et accompagner les professionnels de l'éducation

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE DE L'ÉDUCATION NATIONALE ET DE LA JEUNESSE

VOIE GÉNÉRALE

2DE | 1RE | TLE

Numérique et Sciences informatiques

ENSEIGNEMENT
SPÉCIALITÉ

MANIPULATION DE TABLES

1. Introduction

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 (voire plus) d'articles en vente, de clients, de commandes, un hopital doit pouvoir accéder efficacement 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 facile de mettre en œuvre les opérations de base dans un langage de programmation comme Python.
Nous allons en illustrer quelque unes.

2. Le format `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 réparés par une virgule (d'où le nom). En pratique, on peut spécifier le caractère utilisé pour séparer les différents champs et on utilise fréquemment un point-virgule, une tabulation ou deux points pour cela. 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 apparaîssant en deuxième ligne du fichier.

Dans la suite, nous allons utiliser un fichier nommé `countries.csv` qui contient quelques données sur les différents pays du monde. En voici les premières lignes :

```
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
```

Les champs sont clairement séparés par des points-virgules. Les données sont issues du site `http://www.geonames.org` et ont été légèrement simplifiées.

La signification des différents champs est transparente, à part le dernier champ, nommé `capital` et dont les valeurs sont des numéros d'identifiants de villes que l'on trouvera dans un autre fichier nommé `cities.csv`. Nous verront comment gérer deux fichiers un peu plus tard.

3. Importation des données

Une façon de charger un fichier `csv` en Python est d'utiliser la bibliothèque du même nom. Voici une portion de code permettant de charger le fichier `countries.csv` avec des points-virgules comme délimitations.

```
1 import csv
2
3 pays = []
4
5 with open('countries.csv', newline='') as csvfile:
6     spamreader = csv.reader(csvfile, delimiter=';')
7     for row in spamreader:
8         pays.append(row)
```

Dans ce cas, les résultats sont stockés sous forme d'un tableau de tableau. On a par exemple :

```
1 >>> pays[0]
2 ['iso',
3  'name',
4  'area',
5  'population',
6  'continent',
7  'currency_code',
8  'currency_name',
9  'capital']
10 >>> pays[1]
11 ['AD', 'Andorra', '468.0', '84000', 'EU', 'EUR', 'Euro', '6']
```

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 [1].

```
1 pays = []
2
3 with open('countries.csv', newline='') as csvfile:
4     spamreader = csv.DictReader(csvfile, delimiter=';')
5     for row in spamreader:
6         pays.append(dict(row))
```

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

```
1 >>> pays[0]
2 {'iso': 'AD',
3  'name': 'Andorra',
4  'area': '468.0',
5  'population': '84000',
6  'continent': 'EU',
```

1. On convertit `row` en dictionnaire à la ligne 8 par souci de lisibilité, car sinon c'est un `OrderedDict` qui est utilisé et son affichage est moins plaisant.

```
 'currency_code': 'EUR',
 'currency_name': 'Euro',
 'capital': '6'}
```

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 question simples. Par exemple, quels sont les pays où l'on paye en euro ?

```
>>> [p['name'] for p in pays if p['currency_code'] == 'EUR']
['Andorra',
 'Austria',
 'Belgium',
 ...,
 'Vatican',
 'Mayotte']
```

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

```
>>> [p['currency_code'] for p in pays if p['currency_name'] == 'Dollar']
['XCD',
 'XCD',
 'USD',
 ...,
 'USD',
 'ZWL']
```

On obtient une liste de longueur 54… mais avec beaucoup de répétitions. Pour les supprimer, on peut la transformer en un ensemble (*set* en anglais). On obtient encore 23 codes de monnaie :

```
>>> set([p['currency_code'] for p in pays if p['currency_name'] == 'Dollar'])
{'AUD',
 'BBD',
 ...,
 'USD',
 'XCD',
 'ZWL'}
```

4.2. Tri

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. Une autre utilisation vient du fait que, comme présenté dans la partie algorithmique du programme, la recherche dichotomique dans un tableau trié est bien plus efficace que la recherche séquentielle dans un tableau quelconque.

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 [2].

2. Rappel : la méthode `sort` trie la liste *en place*, alors que la fonction **`sorted`** renvoie une **nouvelle** liste correspondant la liste trié, la liste initiale étant laissée intacte.

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

```
def clé_superficie(p):
    return p['area']
```

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

```
pays.sort(key=clé_superficie, reverse=True)
```

Mais un petit problème demeure. Si on récupère les noms des 5 premiers pays ainsi classés, le résultat est étonnant :

```
>>> [(p['name'], p['area']) for p in pays[:5]]
[('Canada', '9984670'),
 ('South Korea', '98480'),
 ('United States', '9629091'),
 ('Netherlands Antilles', '960'),
 ('China', '9596960')]
```

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 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 clé_superficie(p):
    return float(p['area'])
```

On a alors le résultat espéré :

```
>>> [(p['Country'], p['Area'])
      for p in sorted(pays, key=clé_superficie, reverse=True)[:5]]
[('Russia', '17100000.0'),
 ('Canada', '9984670.0'),
 ('United States', '9629091.0'),
 ('China', '9596960.0'),
 ('Brazil', '8511965.0')]
```

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

Ainsi,

```
[(p['continent'], p['name']) for p in sorted(pays, key=clé_combinée)]
[('AF', 'Algeria'),
 ('AF', 'Angola'),
 ...,
 ('AF', 'Zambia'),
 ('AF', 'Zimbabwe'),
 ('AS', 'Afghanistan'),
 ('AS', 'Armenia'),
 ...,
```

```
 ('SA', 'Uruguay'),
 ('SA', 'Venezuela')]
```

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.

Ainsi :

```
>>> def clé_population(p):
...     return int(p['population'])
...
>>> pays.sort(key=clé_population, reverse=True)
>>> pays.sort(key=clé_continent)
>>> [(p['name'], p['continent'], p['population']) for p in pays]
[('Nigeria', 'AF', '154000000'),
 ('Ethiopia', 'AF', '88013491'),
 ...,
 ('Seychelles', 'AF', '88340'),
 ('Saint Helena', 'AF', '7460'),
 ('China', 'AS', '1330044000'),
 ('India', 'AS', '1173108018'),
 ...,
 ('French Guiana', 'SA', '195506'),
 ('Falkland Islands', 'SA', '2638')]
```

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.

5. 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. Dans un autre document, nous présenterons la bibliothèque `pandas` qui permet une gestion plus efficace de ce genre de traîtement.
De plus, pour pouvoir exploiter les capitales des pays, nous allons devoir utiliser des données présentes dans un fichier supplémentaire. Nous verrons comment le faire facilement à l'aide de la bibliothèque `pandas`.