Rechercher dans une table

Recherches simples

Dans un tableau de de tableaux ou tableau de p-uplets

On peut parcourir un tableau de tableau ou un tableau de p-uplets avec l'instruction `for … in …` pour chercher un élément avec le mot clé `in`. Les recherches sont donc très simples.

Reprenons notre tableau de tableaux des pays et cherchons dans quel pays se trouve 'Rome' :

```
>>> pays
[['France', 'Paris', '68'],
 ['Espagne', 'Madrid', '48'],
 ['Italie', 'Rome', '60']]

for ligne in pays:
    if 'Rome' in ligne:
        print('Rome est en ', ligne[0])
```

Pour comprendre ce qu'il se passe, ici `ligne` itère sur chaque ligne de `pays` prenant tour à tour les valeurs des tableaux `['France', 'Paris', '68']`, `['Espagne', 'Madrid', '48']` puis `['Italie', 'Rome', '60']`. Pour chaque tableau, l'instruction conditionnelle `if 'Rome' in ligne` vérifie si `'Rome'` est présent ou pas.

Dans un tableau de dictionnaires

Un tableau de dictionnaires est aussi itérable avec l'instruction `for … in …` mais par défaut le mot clé `in` vérifie l'existence d'un élément **dans les clés du dictionnaire, pas dans les valeurs**. Il faut donc préciser que l'on recherche l'élément recherché parmi les valeurs du dictionnaire avec la méthode `.values()`.

Cherchons dans quel pays se trouve 'Rome'.

```
pays = [{'Capitale': 'Paris', 'Pays': 'France', 'Population (ml)': '68'},
        {'Capitale': 'Madrid', 'Pays': 'Espagne', 'Population (ml)': '48'},
        {'Capitale': 'Rome', 'Pays': 'Italie', 'Population (ml)': '60'}]

for ligne in pays:
    if 'Rome' in ligne.values():
        print('Rome est en ', ligne['Pays'])
```

ou alors on peut préciser la clé dans laquelle on veut faire la recherche, par exemple ici on peut chercher 'Rome' dans les valeurs associées à la clé `'Capitale'` :

```
for ligne in pays:
    if ligne['Capitale']  == 'Rome':
        print('Rome est en ', ligne['Pays'])
```

Exercice corrigé

On a importé un tableau de dictionnaires des codes postaux avec :

```
with open('laposte_hexasmal.csv', 'r', encoding='utf-8-sig') as f:
    codes = list(csv.DictReader(f, delimiter=';'))
```

1. Ecrire une fonction `coord_gps` qui prend en paramètre un nom de commune et renvoie les coordonnées gps de cette commune.

   Exemple :

   ```
   >>> coord_gps('MANOSQUE')
   '43.835211125, 5.791029867'
   ```

2. Ecrire une fonction `chercher_communes` qui prend en paramètre un code postal et renvoie le tableau des communes qui ont ce code postal.

   Exemple :

   ```
   >>> chercher_communes('04000')
   ['ENTRAGES',
   'LA ROBINE SUR GALABRE',
   'LA ROBINE SUR GALABRE',
   'LA ROBINE SUR GALABRE',
   'DIGNE LES BAINS',
   'DIGNE LES BAINS',
   'LA JAVIE',
   'LA ROBINE SUR GALABRE']
   ```

Réponse

Recherches conditionnelles

Pour un tableau de tableaux, de p-uplets ou de dictionnaires, le mot clé `in` ne suffit plus pour tester la présence d'un élément avec des conditions. Il faut tester les conditions sur chacun des tableaux, p-uplets ou dictionnaires.

Reprenons notre tableau de tableaux de pays et cherchons les pays qui ont plus de 50 millions d'habitants :

```
>>> pays
[['France', 'Paris', '68'],
 ['Espagne', 'Madrid', '48'],
 ['Italie', 'Rome', '60']]

for ligne in pays:
    if int(ligne[2]) > 50:     # convertir la population en entier
        print(ligne[0], "a plus de 50 millions d'habitants")
```

Exercice corrigé

On a importé un tableau de dictionnaires des codes postaux avec :

```
with open('laposte_hexasmal.csv', 'r', encoding='utf-8-sig') as f:
    codes = list(csv.DictReader(f, delimiter=';'))
```

Ecrire une fonction `tout_departement` qui prend en paramètre un numéro de département et renvoie un tableau avec toutes les communes de ce département.

```
Exemple :
``` py
>>> tout_departement('04')
['LE CAIRE',
'PIERREVERT',
'STE CROIX DU VERDON',
'ENTRAGES',
'LA MOTTE DU CAIRE',
'SIMIANE LA ROTONDE',
...]
```
```

Réponse

Tests de cohérence et recherche de doublons

Jusqu'ici on a fait l'hypothèse que tous les champs du fichier sont remplis et corrects sans vérifier leur cohérence. Ce n'est pas toujours le cas et un fichier mal renseigné, ou avec des valeurs vides peut ensuite générer des problèmes. Pour l'éviter, on peut faire des **tests de cohérence** et des **recherches de doublons**.

Prenons l'exemple tableau de tableaux des pays qui aurait été importé avec des données peu fiables :

```
>>> pays
[['France', 'Paris', '68.0'],
 ['Espagne', '48'],
 ['Italie', 'Rome', '60'],
 ['Italie', 'Rome', '61']]
```

Ces données vont créer plusieurs problèmes :

- La donnée de population de la France, `'68.0'`, va lever une erreur quand on va la convertir en nombre entier :

  ```
  >>> int(pays[0][2])
  Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
  ValueError: invalid literal for int() with base 10: '68.0'
  ```

  - La capitale de l'Espagne n'est pas renseignée, c'est la valeur suivante, `'48'` qui sera utilisée à la place. Si on ne veut pas renseigner cette capitale, il faudrait écrire `..., ['Espagne', '', '48'], ...` ou `..., ['Espagne', None, '48'], ...` pour respecter les descriteurs de colonne. Ici on peut noter l'avantage d'utiliser un tableau de dictionnaire où il suffirait d'omettre la clé d'une valeur qui n'est pas renseignée.

- Il y a un doublon sur l'Italie, quelle valeur utiliser dans un programme pour la capitale, `Rome` ou `Roma`, et pour la population ? C'est source d'erreurs.

Exercice corrigé

On a importé un tableau de dictionnaires des codes postaux avec :

```
with open('laposte_hexasmal.csv', 'r', encoding='utf-8-sig') as f:
    codes = list(csv.DictReader(f, delimiter=';'))
```

1. Ecrire une fonction `coherence` qui vérifie que tous les noms de communes sont renseignés et qu'aucun n'est laissé vide.

   Exemple :

   ```
   >>> coherence()
   True
   ```

2. Ecrire une fonction `doublons` qui affiche les doublons.

Réponse