TP : p-uplets et fichiers textes

Thème types construits

1 Traitement d'un fichier texte

Un fichier texte contient une séquence de caractères, chaque caractère étant codé par une série d'octets selon un certain encodage. Nous allons manipuler dans ce TP trois fichiers textes fournis dans l'archive `materiel.zip` qu'il faut donc déballer.

- `communes69.csv` contient les informations du recensement 2015 pour les 296 communes du Rhône. Chaque ligne correspond à une commune avec les informations *nom,population municipale, population à part, population totale* séparées par une virgule. L'extension csv signifie *Comma Separated Values* et désigne les fichiers textes ainsi formatés.

  Voici les trois premières lignes :

```
Affoux,340,3,343
Aigueperse,246,1,247
Albigny-sur-Saône,2793,37,2830
```

- `airports-reduit.csv` contient des informations sur les 57302 aéroports recensés par `https://ourairports.com/`. Chaque ligne correspond à un aéroport avec les informations *nom;latitude;longitude;altitude (en pieds); code ISO du pays* séparées par un point virgule. Voici les trois premières lignes :

```
Total Rf Heliport;40.07080078125;-74.93360137939453;11;US
Aero B Ranch Airport;38.704022;-101.473911;3435;US
Lowell Field;59.94919968;-151.695999146;450;US
```

- `photos.txt` contient la sortie standard d'une commande du `shell` (en l'occurrence `ls -l`) listant le contenu d'un répertoire contenant 123 photos. Chaque ligne regroupe diverses informations concernant un fichier contenues dans des champs séparés par des espaces. Le nom du fichier figure dans le dernier champ et sa taille en octets dans le cinquième. Voici les trois premières lignes :

```
-rw-r--r-- 1 frederic frederic 2422388 mai 14 15:06 DSC00001.JPG
-rw-r--r-- 1 frederic frederic 2532569 juil. 13 14:00 DSC00002.JPG
-rw-r--r-- 1 frederic frederic 3159257 juil. 13 15:10 DSC00003.JPG
```

L'objectif des exercices suivants est d'extraire les informations de chacun de ces fichiers sous la forme de **tableaux de tuples** :

```
>>> len(tab_tuple_communes), tab_tuple_communes[0:2]
(296, [('Affoux', '340', '3', '343'), ('Aigueperse', '246', '1', '247')])

>>> len(tab_tuple_airports), tab_tuple_airports[0:2]
```

```
(57302,
 [('Total Rf Heliport', '40.07080078125', '-74.93360137939453', '11', 'US'),
  ('Aero B Ranch Airport', '38.704022', '-101.473911', '3435', 'US')])

>>> len(tab_tuple_photos), tab_tuple_photos[0:2]
(123, [('-rw-r--r--','1','frederic','frederic','2422388','mai','14','15:06',
    'DSC00001.JPG'),
  ('-rw-r--r--','1',frederic','frederic','2532569','juil.','13','14:00','
     DSC00002.JPG')])
```

On peut remarquer qu'il faut connaître la signification des différents champs constituant une ligne pour l'exploiter. Cette information est en général contenue dans une ligne d'en-tête qui a été enlevée de chaque fichier. En effet, on ne dispose pas encore d'un type de séquence à champs nommés, qui permettrait d'accéder à ses éléments par des descripteurs de type texte plutôt que par des index numériques. En Python, ces tuples à champs nommés sont représentés par les dictionnaires que nous introduirons dans le prochain chapitre.

Exercice 1

1. Dans un IDE Python, ouvrir le script `TP-puplets-eleves.py` fourni dans l'archive `materiel.zip` Tous les codes du TP seront écrits dans ce script, en séparant si possible les différents exercices en cellules.

2. Saisir puis exécuter les instructions suivantes dans l'éditeur, afin d'afficher dans la console le contenu du fichier `communes69.csv`. On fournit en Annexe un memento de manipulation des fichiers textes.

```
# ouverture du fichier
f = open('communes69.csv')
# lecture du fichier ligne par ligne
for ligne in f:
    print(ligne)
#fermeture du fichier
f.close()
```

3. Saisir puis exécuter les instructions suivantes dans la console :

```
>>> f = open('communes69.csv')
>>> ligne1 = f.readline()
>>> ligne1
'Affoux,340,3,343\n'
>>> ligne1 = ligne1.rstrip()
>>> ligne1
'Affoux,340,3,343'
>>> tuple_ligne = tuple(ligne1.split(','))
>>> tuple_ligne
('Affoux', '340', '3', '343')
>>> f.close()
```

- lecture de la première ligne uniquement avec `ligne1 = f.readline();`
- suppression du caractère de fin de ligne avec `ligne1 = ligne1.rstrip();`

- découpage de la ligne en une liste de champs selon le séparateur `','` et conversion en `tuple` avec `tuple_ligne = tuple(ligne1.split(','))`

4. Écrire la fonction dont on donne le prototype ci-dessous :

```
def fichier_vers_tab_tuple(fichier, separateur):
    """Prend en paramètres :
    - fichier de type str représentant un chemin d'accès à un
        fichier texte
    - separateur de type str désignant le séparateur de champs sur
        une ligne du fichier
    Retourne un tableau de tuples, chaque tuple contenant les diffé
        rents champs
    d'une ligne du fichier"""
```

Deux assertions sont fournies dans `TP-puplets-eleves.py` pour tester la fonction.

Exercice 2

1. Exécuter le code suivant dans la console. Quel problème peut-on remarquer ?

```
>>> tab_tuple_photos = fichier_vers_tab_tuple('photos.txt', ' ')
>>> tab_tuple_photos[0:2]
[('-rw-r--r--', '1', 'frederic', 'frederic', '2422388', 'mai',
'', '', '14', '15:06', 'DSC00001.JPG'),
('-rw-r--r--','1','frederic','frederic','2532569','juil.','13','
    14:00','DSC00002.JPG')]
```

2. Une solution est d'utiliser une expression régulière (ou regex) comme séparateur et de remplacer la fonction `str.split` par `re.split` du module `re`. Une expression régulière, est une formule décrivant un ensemble de chaînes de caractères.

Écrire une fonction de signature `fichier_vers_tab_tuple2(fichier, separateur, regex)` avec un paramètre booléen `regex` qui permet de découper selon une expression régulière.

```
>>> ligne = 'un deux trois'
>>> ligne.split(' ')
['un', 'deux', '', 'trois']
>>> import re
>>> sep = '\s+' #expression régulière 1 espace ou plus
>>> re.split(sep,ligne)
['un', 'deux', 'trois']
```

2 Recherches dans un tableau de tuples

Exercice 3

Pour traiter les informations d'un tableau de tuples, on peut utiliser le *tuple unpacking* mais attention tous les champs de tuple sont de type chaînes de caractères `str` et il faut d'abord les convertir dans le type correspondant au traitement souhaité.

```
>>> commune1, commune2 = tab_tuple_communes[:2]
>>> commune1, commune2
commune1, commune2
(('Affoux', '340', '3', '343'), ('Aigueperse', '246', '1', '247'))
>>> nom, pop_int, pop_ext, pop_tot = commune1
>>> pop_int, pop_ext
('340', '3')
>>> pop_int + pop_ext #calcul incorrect de la population totale
'3403'
>>> int(pop_int) + int(pop_ext) #calcul correct de la population totale
    après conversion en entiers
343
```

Les fonctions doivent être testées sur les assertions fournies dans `TP-puplets-eleves.py`.
Pour chaque fonction le paramètre `tab` est un tableau de tuples rassemblant tous les enregistrements contenus dans le fichier `communes69.csv`.

1. Écrire une fonction `population_minimale(tab)` qui retourne un tuple constitué de la population minimale et un tableau des noms des communes atteignant ce minimum.
2. Écrire une fonction `population_cumul(tab)` qui retourne un tuple constitué des cumuls des populations municipale, externe et totale des communes.
3. Écrire une fonction `population_moyenne(tab)` qui retourne un tuple constitué des moyennes des populations municipale, externe et totale des communes.
4. Écrire une fonction `population_maximale(tab)` qui retourne un tuple constitué de la population maximale et un tableau des noms des communes atteignant ce maximum. Tester puis critiquer le résultat.
5. Écrire une fonction `population_lyon(tab)` qui retourne la population totale de la ville de Lyon regroupant neuf communes d'arrondissement.

Exercice 4

Avec `fichier_vers_tab_tuple(airports-reduit.csv, ',')` on peut extraire un tableau de tuples rassemblant tous les enregistrements contenus dans le fichier `airports-reduit.csv`.
Les fonctions doivent être testées sur les assertions fournies dans `TP-puplets-eleves.py`.

Dans certains cas, certains valeurs de tuple sont manquantes! Il faut prévoir ces cas dans le traitement.

1. Écrire une fonction `nombre_aeroports_pays(tab, pays)` dont on donne ci-dessous le prototype :

```
def nombre_aeroports_pays(tab, pays):
    """Prend en paramètres:
    - tab un tableau de tuples rassemblant tous les enregistrements
        contenus dans airports-reduit.csv
    - pays de type str correspondant au code ISO d'un pays
    Retourne le nombre d'aéroports pour ce pays"""
```

2. Un pied correspond à 1/3 de verge anglaise (yard), c'est-à-dire 0,3048 mètre. Écrire une fonction qui convertit une mesure en mètres en une mesure en pieds.

3. Écrire une fonction `filtre_altmin_aeroports(tab, altmin)` vérifiant le prototype ci-dessous :

```
def filtre_altmin_aeroports(tab, altmin):
    """Prend en paramètres:
    - tab un tableau de tuples rassemblant tous les enregistrements
        contenus dans airports-reduit.csv
    - altmin une altitude minimale en mètres
    Retourne un tuple constitué du pays et du nombre d'aéroports
        pour ce pays"""
```

4. Écrire une fonction `filtre_altmax_aeroport` vérifiant le prototype ci-dessous :

```
def filtre_altmax_aeroport(tab, latmin):
    """Prend en paramètres:
    - tab un tableau de tuples rassemblant tous les enregistrements
        contenus dans airports-reduit.csv
    - latmin une latitude minimale
    Retourne un tuple constitué de latmax, altmax, tab_nom
    où altmax est l'altitude maximale d'un aeroport de latitude >
        latmin
    latmax est sa latitude
    et tab_nom le tableau des noms des aéroports atteignant ce
        maximum"""
```

   Tester la fonction et critiquer le résultat.

5. Peut-on facilement déterminer le pays qui compte le plus d'aéroports dans `airports-reduit.csv` ?

ANNEXE

Méthode *Lecture / écriture de fichiers textes*

En Python, l'accès à un fichier texte se fait par l'intermédiaire d'un descripteur de fichier créé à l'aide de la primitive `open(nom,mode)`. Une fois que les manipulations sont terminées, il faut bien penser à fermer le descripteur de fichier avec `f.close()`.
Lors de l'ouverture d'un fichier on précise l'un des trois modes d'accès : lecture `'r'`, écriture `'w'` ou ajout `'a'`. Attention, pour l'ouverture en mode écriture, les modes `'w'` ou `'a'` créent le fichier s'il n'existe pas mais le mode `'w'` écrase le fichier s'il existe déjà.
Pour bien manipuler un fichier texte, il faut d'abord connaître la façon dont il est structuré!!!

Fonctions	Rôle
`f = open('nom_fichier.txt','w')`	accès en écriture avec création d'un nouveau fichier
`f = open('nom_fichier.txt','r')`	accès à un fichier existant en mode lecture
`f = open('nom_fichier.txt','a')`	accès en écriture à un fichier existant en mode ajout
`f.write('texte')`	ajout de 'texte' dans le fichier
`f.writelines(liste)`	ajout d'une liste de lignes dans un fichier
`f.read()`	lecture de tout le fichier
`f.read(8)`	lecture des 8 premiers caractères du fichier
`f.readline()`	lecture de la ligne courante du fichier
`f.readlines()`	lecture de toutes les lignes stockées dans une liste
`for ligne in f`	itération sur les lignes du fichier
`f.close()`	fermeture du fichier

Lecture de tout le fichier

```
f = open('fichier.txt','r')
data = f.read()
f.close()
```

Lecture ligne par ligne

```
f = open('fichier.txt','r')
for ligne in f:
    #traitement sur la ligne
f.close()
```

Lecture ligne par ligne

```
f = open('fichier.txt','r')
ligne = f.readline()
while ligne != '':
    #traitement sur la ligne
    ligne = f.readline()
f.close()
```

Capture dans une liste de toutes les lignes

```
f = open('fichier.txt','r')
listlignes = f.readlines()
f.close()
```

Écriture

```
f = open('fichier.txt','r')
f.write('Ligne ecrase tout\n')
f.close()
```

Ajout à la fin

```
f = open('fichier.txt','a')
f.write('Ligne de plus\n')
f.close()
```