INTRODUCTION AU TRAITEMENT DES DONNEES

1. Le format CSV

Le traitement des données occupe une place de plus en plus importante dans la société actuelle. Les capacités informatiques permettent une analyse de données en quantités très importantes, que ce soit pour des usages légaux mais douteux (ciblage des utilisateurs, reconnaissance faciale, notation sociale…), aussi bien que des usages plus nobles (épidémiologie, amélioration des services rendus aux usagers, science collaborative…).

Exemple de données :

Les données sont souvent structurées en tables. Un format standard pour les données, généralisé par l'usage des tableurs, est le format CSV : comma separated values (valeurs séparées par des virgules). Ce format est particulièrement simple : chaque ligne représente un **enregistrement** (une ligne de la table) , et chaque colonne, séparée des autres par une virgule, représente un **champ** (un **attribut**, un **descripteur**) de cet élément. Toutes les lignes ont le même nombre de champs. L'extension de ces fichier est « .csv », mais on peut tout aussi bien les enregistrer au format texte. Les données sont structurées dans une **base de données**

En France, comme la virgule est utilisée pour délimiter la partie décimale de la partie entière, on utilise plutôt comme séparateur des tabulations ; voire des points virgules. L'extension « .tsv » est parfois utilisée lorsque le séparateur est la tabulation. Il y aussi des habitudes suivant les systèmes d'exploitation, sous Windows on trouve souvent des « ; » et sous Linux des « , » comme séparateur.

2. Indexation de tables

Les fichiers csv étant des fichiers texte, on les ouvre de la même manière. **Dans ce chapitre uniquement**, on transformera les fichiers csv en liste de dictionnaires. En effet, ce n'est nullement une obligation. Par ailleurs, ces dictionnaires sont ordonnés à partir de la version 3.6 de Python, c'est-à-dire que ce ne sont pas tout à fait les dictionnaires que l'on a vus jusqu'à présent.

Code/exemple :

```
import csv
entree_csv = open('films.csv','r', encoding = 'utf-8')
      #on retrouve les fonctions d'ouvertures de fichiers, ici en lecture
      #avec un encodage en utf-8
lecteur = csv.DictReader(entree_csv)  # on crée un lecteur du
                        # dictionnaire fait à partir du fichier csv
```

On peut bien sûr écrire un fichier csv :

Code/exemple :

```
import csv
sortie_csv = open('films_preferes.csv','w', encoding = 'utf-8')
      #on retrouve les fonctions d'ouvertures de fichiers, ici en écriture
      #avec un encodage en utf-8, et création éventuelle du fichier
      #s'il n'existe pas déjà

scribe = csv.DictWriter(sortie_csv,['en-tête1','en-tête2','en-tête3'])
      # on crée un « écrivain » qui va pouvoir écrire dans le dictionnaire
      # crée à partir du fichier csv.
scribe.writeheaders()
scribe.writerows(table)
```

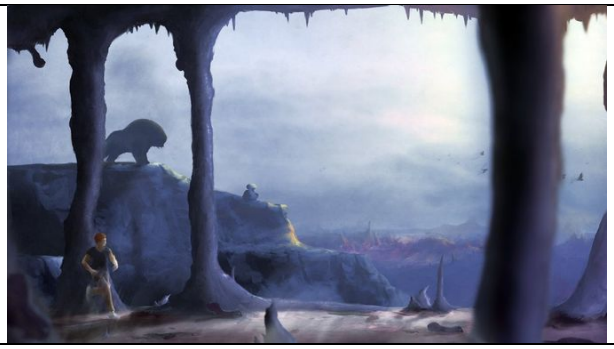
Dans ce cours, les données seront sous forme de liste de **dictionnaires ordonnés**. Les dictionnaires ordonnés sont un nouveau type de variable (type `OrderedDict`) . Les méthodes usuelles `.keys()` , `.values()` et `.items()` fonctionnent de la même manière qu'avec les dictionnaires standard.

Exemple : quelques jeux vidéo et leur année de sortie.

```
from collections import OrderedDict
data_jeux_video = [\
   OrderedDict([('titre', 'league_of_legend'), ('annee', 2009)])
   OrderedDict([('titre', 'mmoprpgporgpppmm'), ('annee', 0)])
   OrderedDict([('titre', 'wow'), ('annee', 2004)])
   OrderedDict([('titre', 'diablo2'), ('annee', 2000)])
   OrderedDict([('titre', 'doom'), ('annee', 1993)])
   OrderedDict([('titre', 'warcraft3'), ('annee', 2002)])
   OrderedDict([('titre', 'tetris'), ('annee', 1984)])
   OrderedDict([('titre', 'monkey_island'), ('annee', 1990)])
   OrderedDict([('titre', 'minecraft'), ('annee', 2011)])
   OrderedDict([('titre', 'baldurs_gate'), ('annee', 1998)])
   OrderedDict([('titre', 'portal'), ('annee', 2007)])
   OrderedDict([('titre', 'heroes_mm3'), ('annee', 1999)])
   OrderedDict([('titre', 'prince_of_persia'), ('annee', 1989)])
   OrderedDict([('titre', 'another world'), ('annee', 1991)])]

>>> data_jeux_video[0].keys()
odict_keys(['titre', 'annee'])
>>> data_jeux_video[0].values()
odict_values(['league_of_legend', 2009])
>>> data_jeux_video[0].items()
odict_items([('titre', 'league_of_legend'), ('annee', 2009)])
```

Another World en 1991



Another World en 2015

Dans une base de données, il peut y avoir plusieurs enregistrements possédant les mêmes « valeurs importantes », comme par exemple Nom et Prénom. Pour différencier ces enregistrements, on crée un **identifiant unique** associé à chacun d'entre eux.

3. Recherche dans une table

Pour rechercher des enregistrements dans une table, on utilise les instructions habituelles en Python.

Exemple de construction d'une nouvelle table par extraction des données pertinentes :

On veut extraire de la table de données tous les jeux sortis avant 2000, avec leur date.

```
def extraction(donnees,an):
    """
    extrait les jeux vidéos sortis avant une année donnée:
    @param données : liste de dictionnaires ordonnés
                     ayant un champ 'annee'
    @param an : entier, compris de préférence entre -6000
```

```
                    et l'année courante
    @return avant_date : liste de dictionnaires ordonnés extraits
                    de "données" dont le valeur du champ annee est
                    strictement inférieure au paramètre an
    """
    avant_date = []
    for ligne in donnees:
        if ligne['annee'] < an:
            avant_date.append(ligne)
    return avant_date

avant_2000 = extraction(data_jeux_video,2000)

>>>avant_2000
     [OrderedDict([('titre', 'mmoprpgporgpppmm'), ('annee', 0)]),
      OrderedDict([('titre', 'doom'), ('annee', 1993)]),
      OrderedDict([('titre', 'tetris'), ('annee', 1984)]),
      OrderedDict([('titre', 'monkey_island'), ('annee', 1990)]),
      OrderedDict([('titre', 'baldurs_gate'), ('annee', 1998)]),
      OrderedDict([('titre', 'heroes_mm3'), ('annee', 1999)]),
      OrderedDict([('titre', 'prince_of_persia'), ('annee', 1989)]),
      OrderedDict([('titre', 'another world'), ('annee', 1991)])]
```

Cet exemple nous montre aussi l'importance de tester la cohérence et la validité des données. Dans cette table il y a un champ de données invalide : l'année 0 pour le jeu « mmoprpgporgpppmm ». La valeur était peut-être manquante ou inconnue au moment de la collecte des données. On pourrait se dire qu'il aurait été plus logique de donner `"inconnu"` comme valeur au champ `année`. On se serait alors retrouvé face à un autre souci, celui de la cohérence. En effet `0` est un entier et `"inconnu"` une chaîne de caractères. Les champs de données sont toujours du même type, et ceci de manière rigoureuse dans les langages spécialisés (`5.0` est un flottant de type différent de `5` qui est entier).

Remarque : l'extraction des données peut se faire par compréhension. L'exemple précédent devient :

```
def extraction(donnees,an):
    """
    extrait les jeux vidéos sortis avant une année donnée:
    @param données : liste de dictionnaires ordonnés
                    ayant un champ 'annee'
    @param an : entier, compris de préférence entre -6000
                    et l'année courante
    @return avant_date : liste de dictionnaires ordonnés extraits
                    de "données" dont le valeur du champ annee est
                    strictement inférieure au paramètre an
    """
    avant_date = [ligne for ligne in donnees if ligne['annee'] < an]
    return avant_date
```

4. Tri d'une table

Il est souvent nécessaire de trier les données suivant un critère précis. Nous avons vu quelques algorithmes de tris que l'on pourrait appliquer. Les fonctions intégrées sont plus rapides, et aussi bien plus faciles d'utilisation. On dispose de deux outils :

- La méthode `.sort()`, qui modifie l'objet trié.
- La fonction `sorted()`, qui ne modifie pas l'objet trié, et dont le résultat doit être affecté dans une variable.

Remarque : le tri utilisé par `.sort()` et `sorted()` s'appelle Timsort. Il a été inventé spécialement pour le Python

Code/exemple : avec la liste composée uniquement des titres des jeux vidéo

```
>>> liste = ['league_of_legend', 'mmoprpgporgpppmm', 'wow', 'diablo2',
'doom','warcraft3','tetris','monkey_island','minecraft','baldurs_gate','p
ortal','heroes_mm3','prince_of_persia','another world']
>>> liste.sort()      #tri dans l'ordre lexicographique
>>> liste
['another  world',  'baldurs_gate',  'diablo2',  'doom',  'heroes_mm3',
'league_of_legend',  'minecraft',  'mmoprpgporgpppmm',  'monkey_island',
'portal', 'prince_of_persia', 'tetris', 'warcraft3', 'wow']

#On recommence avec la même liste :
>>> liste = ['league_of_legend', 'mmoprpgporgpppmm', 'wow', 'diablo2',
'doom','warcraft3','tetris','monkey_island','minecraft','baldurs_gate','p
ortal','heroes_mm3','prince_of_persia','another world']
>>> sorted(liste)
['another  world',  'baldurs_gate',  'diablo2',  'doom',  'heroes_mm3',
'league_of_legend',  'minecraft',  'mmoprpgporgpppmm',  'monkey_island',
'portal', 'prince_of_persia', 'tetris', 'warcraft3', 'wow']
     #le tri est bien le même
#Mais la liste n'est pas modifiée :
>>> liste)
['league_of_legend',      'mmoprpgporgpppmm',      'wow',      'diablo2',
'doom','warcraft3','tetris','monkey_island','minecraft','baldurs_gate','p
ortal','heroes_mm3','prince_of_persia','another world']
#On l'enregistre dans une autre variable
>>> liste_triee = sorted(liste)
>>> liste_triee
['another  world',  'baldurs_gate',  'diablo2',  'doom',  'heroes_mm3',
'league_of_legend',  'minecraft',  'mmoprpgporgpppmm',  'monkey_island',
'portal', 'prince_of_persia', 'tetris', 'warcraft3', 'wow']
```

Lorsque l'on trie une table de données, on ne veut pas forcément un tri suivant le premier attribut (la première colonne de la table). Il est alors nécessaire de définir une **clé** de tri, avec le paramètre `key`. On peut aussi préciser l'ordre dans lequel on veut trier (croissant ou décroissant) avec le paramètre `reverse`. Ces paramètres sont acceptés aussi bien par `sort()` que `sorted()`.

Exemple : tri des jeux vidéo par année de sortie

```
def cle_annee(ligne):
    """
    Renvoie la valeur du champ 'annee' d'un enregistrement de la table
    """
    return ligne['annee']

jeux_tries = sorted(data_jeux_video, key = cle_annee)

>>> jeux_tries
[OrderedDict([('titre', 'mmoprpgporgpppmm'), ('annee', 0)]),
 OrderedDict([('titre', 'tetris'), ('annee', 1984)]),
 OrderedDict([('titre', 'prince_of_persia'), ('annee', 1989)]),
 OrderedDict([('titre', 'monkey_island'), ('annee', 1990)]),
 OrderedDict([('titre', 'another world'), ('annee', 1991)]),
 OrderedDict([('titre', 'doom'), ('annee', 1993)]),
 OrderedDict([('titre', 'baldurs_gate'), ('annee', 1998)]),
 OrderedDict([('titre', 'heroes_mm3'), ('annee', 1999)]),
 OrderedDict([('titre', 'diablo2'), ('annee', 2000)]),
 OrderedDict([('titre', 'warcraft3'), ('annee', 2002)]),
```

```
 OrderedDict([('titre', 'wow'), ('annee', 2004)]),
 OrderedDict([('titre', 'portal'), ('annee', 2007)]),
 OrderedDict([('titre', 'league_of_legend'), ('annee', 2009)]),
 OrderedDict([('titre', 'minecraft'), ('annee', 2011)])]
```

5. Fusion des tables
   a. Union de tables

Si l'on dispose de tables de données comportant les mêmes champs, alors on peut les réunir en ajoutant les éléments de l'une à l'autre.

Exemple : on ajoute quelques jeux

```
jeux_intelligents =
      [OrderedDict([('titre', 'the witness'), ('annee', 2016)]),
       OrderedDict([('titre', 'myst'),('annee', 1993)])]
```

On utilise les instructions sur les listes

```
>>> data_jeux_video + jeux_intelligents
```

renvoie

```
      [OrderedDict([('titre', 'league_of_legend'), ('annee', 2009)]),
       OrderedDict([('titre', 'mmoprpgporgpppmm'), ('annee', 0)]),
       OrderedDict([('titre', 'wow'), ('annee', 2004)]),
       OrderedDict([('titre','diablo2'), ('annee', 2000)]),
       OrderedDict([('titre', 'doom'), ('annee', 1993)]),
       OrderedDict([('titre', 'warcraft3'), ('annee', 2002)]),
       OrderedDict([('titre', 'tetris'), ('annee', 1984)]),
       OrderedDict([('titre', 'monkey_island'), ('annee', 1990)]),
       OrderedDict([('titre', 'minecraft'), ('annee', 2011)]),
       OrderedDict([('titre', 'baldurs_gate'), ('annee', 1998)]),
       OrderedDict([('titre', 'portal'), ('annee', 2007)]),
       OrderedDict([('titre', 'heroes_mm3'), ('annee', 1999)]),
       OrderedDict([('titre', 'prince_of_persia'), ('annee', 1989)]),
       OrderedDict([('titre', 'another world'), ('annee', 1991)]),
       OrderedDict([('titre', 'the witness'), ('annee', 2016)]),
       OrderedDict([('titre', 'myst'), ('annee', 1993)])]
```

Ou bien :

```
      for ligne in jeux_intelligents:
          data_jeux_video.append(ligne)
```

qui renvoie le même résultat.

   b. Jointure de tables

La jointure crée une nouvelle table à partir de deux (ou plus) tables préexistantes.

Exemple : on souhaite rajouter le créateur des jeux, on dispose des deux tables `data_jeux_video` et `createurs`. Les deux ont en commun le descripteur `'titre'`, qui va servir d'identifiant (rappel : on aurait du créer un entier, identifiant unique, pour chaque jeu).

Voici la liste des créateurs qui est incomplète, tout en comportant un jeu absent de la table `data_jeux_video` :

```
>>> createurs
[OrderedDict([('titre', 'the witness'), ('createur', 'Jonathan Blow')]),
OrderedDict([('titre', 'myst'), ('createur', 'Robyn et Rand Miller')]),
OrderedDict([('titre',  'heroes_mm3'),  ('createur',  'Robyn  et  Rand
Miller')]),
OrderedDict([('titre', 'prince_of_persia'), ('createur', 'Jordan Mechner
')]),
OrderedDict([('titre', 'diablo2'), ('createur', 'Blizzard North')]),
 OrderedDict([('titre',     'warcraft3'),     ('createur',     'Blizzard
Entertainment')]),
OrderedDict([('titre', 'monkey_island'), ('createur', 'LucasArt')]),
OrderedDict([('titre', 'another world'), ('createur', 'Eric Chahi')]),
```

```
OrderedDict([('titre', 'tetris'), ('createur', 'Alekseï Pajitnov')]),
OrderedDict([('titre', 'wow'), ('createur', 'Blizzard Entertainment')]),
OrderedDict([('titre', 'lemmings'),('createur', 'DMA Design')]),]
```

A partir de cette liste et de la liste des jeux, on peut réaliser plusieurs types de jointures.

- On ne conserve que les enregistrements présents dans les deux listes ;
- On conserve tous les enregistrements, en complétant les données absentes absents par des champs de valeur fixée (une chaîne vide par exemple `''`, ou `None`)
- On conserve tous les enregistrements d'une des listes, en complétant comme on peut avec la deuxième. Il y aura éventuellement des champs à compléter comme ci-dessus.

Exemple : premier type de jointure (dite jointure interne)

```
jeux_createurs = []
for jeu in data_jeux_video:
   for crea in createurs:
      if jeu['titre'] == crea['titre']:
         jeux_createurs.append(OrderedDict([('titre',jeu['titre']),\
           ('annee',jeu['annee']), ('createur',crea['createur'])]) )
```

Renvoie :

```
>>> jeux_createurs

[OrderedDict([('titre', 'wow'), ('annee', 2004), ('createur', 'Blizzard
Entertainment')]),
OrderedDict([('titre', 'diablo2'), ('annee', 2000), ('createur',
'Blizzard North')]),
OrderedDict([('titre', 'warcraft3'), ('annee', 2002), ('createur',
'Blizzard Entertainment')]),
OrderedDict([('titre', 'tetris'), ('annee', 1984), ('createur', 'Alekseï
Pajitnov')]),
OrderedDict([('titre', 'monkey_island'), ('annee', 1990), ('createur',
'LucasArt')]),
OrderedDict([('titre', 'heroes_mm3'), ('annee', 1999), ('createur',
'Robyn et Rand Miller')]),
OrderedDict([('titre', 'prince_of_persia'), ('annee', 1989), ('createur',
'Jordan Mechner ')]),
OrderedDict([('titre', 'another world'), ('annee', 1991), ('createur',
'Eric Chahi')]),
OrderedDict([('titre', 'the witness'), ('annee', 2016), ('createur',
'Jonathan Blow')]),
OrderedDict([('titre', 'myst'), ('annee', 1993), ('createur', 'Robyn et
Rand Miller')])]
```

Pour ces exercices, on utilisera un « vrai » ensemble de données. Les fichiers sont « films.csv », « realisateurs.csv » et « distribution.csv ». Les données restent de taille modeste (un peu moins de 5000 films et réalisateurs, un peu plus de 100 000 acteurs). Mais certains programmes, notamment de jointure, peuvent prendre un peu de temps lors de l'exécution.

Le code suivant est à insérer soit en totalité, soit en partie, au début des exercices

```
import csv
from collections import OrderedDict

films = []
reals = []
distrib = []
```

```
def load_file(filename):
    container = []
    csv_file = open(filename,'r', encoding ='ISO-8859-15') #ou 'utf-8'
    lecteur = csv.DictReader(csv_file, delimiter=';')

    for ligne in lecteur:
        container.append(ligne)
    csv_file.close()
    return container

films = load_file('films.csv')
reals = load_file('realisateurs.csv')
distrib = load_file('distribution.csv')
```

1. Préliminaire facultatif pour la présentation des résultats : écrire une fonction pour afficher soit une liste entière, soit les n (passé en paramètre) premières lignes, soit les n dernières lignes. On pourra considérer que si $n = 0$, on affiche la liste entière, si $n < 0$ on affiche les dernières lignes. Cette fonction servira à afficher les résultats dans les exercices.
   Donnez également le nombre d'enregistrements de chaque table.
2. Écrire une fonction qui renvoie la liste tous les films sortis avant 1995 ou après 2015. On peut utiliser une des deux méthodes suivantes, ou bien même tester les deux méthodes :
   - En créant directement une seule liste, avec une seule condition
   - En créant deux listes, une par condition, puis en les réunissant.

   Trier les résultats :
   a. par année de sortie.
   b. par titre.
3. Le but est de calculer la durée moyenne des films.
   a. Faire ce calcul.

   Pour la suite, on va agréger les données, c'est à dire rassembler les données suivant des catégories d'un attribut bien déterminé.

   b. Compter le nombre de films par catégories suivantes de durée (en min) : $[1;85]$, $[86;120]$, $[121;145]$ et $[146;+\infty[$. Comparer avec le nombre total de films. Conclure, éventuellement en modifiant le code pour fouiller les données.
   c. Corriger le a pour avoir un résultat aussi correct que possible.
4. Agrégation de données : trouver la durée moyenne des films par année de sortie. On tiendra compte des anomalies constatées à l'exercice 4. Dans un deuxième temps, trier les résultats obtenus de manière à afficher la durée moyenne par année croissante. Conclure.
5. Écrire une fonction qui trie les films par année puis par recette. Donner les deux films les plus rentables sortis chaque année.
6. *À faire en dernier* : on a l'envie débile de trier les films par le dernier chiffre de l'année de sortie… écrire une fonction réalisant ce tri.

Les exercices suivant peuvent demander un certain temps d'exécution.

7. Créer une table de données réalisant la jointure films/acteurs.
   a. Créer une table de données comportant les champs suivants : identifiant du film, titre, acteur.

   Comme vous pouvez vous en douter, il y a plusieurs acteurs dans un film, et les acteurs ont (souvent) joué dans plusieurs films. Créer deux fonctions :
   b. La première renvoie la liste de tous les acteurs ayant joué dans un film donné.
   c. La deuxième renvoie la liste de tous les films dans lesquels a joué un acteur donné.
8. Créer une table de données réalisant la jointure film/réalisateur. Les données sont incomplètes : certains films n'ont pas de réalisateur connu. On mettra une chaîne vide `''` dans ce cas.
   Donner tous les films sans réalisateur connu.

9. Créer une table de données réalisant la jointure film/acteurs. On souhaite avoir toutes les données, y compris incomplètes. C'est-à-dire que si un film n'a pas d'acteurs, on veut qu'il soit présent, et de même pour les acteurs (qui peuvent être associés à des identifiants de films absents de la liste des films)
   a. Modifier le code pour obtenir un seul enregistrement par film. Les acteurs seront regroupés dans une seule chaîne de caractères, et séparés par des virgules.
   b. Extraire les films sans acteurs.
   c. Et les acteurs sans films.