Fichiers

Skander Zannad et Judicaël Courant

Lycée La Martinière-Monplaisir

2013-10-29

1 Fichiers

Fichier : suite (généralement finie) d'octets [1].

Contient : tout type d'information.

Comment interpréter les données : pas de règle *a priori*.

Donc c'est celui qui lit/écrit qui choisit. [2]

Mais il y a des conventions, qu'il vaut mieux respecter.

Comment lire/écrire des données dans des fichiers depuis python ?

---

1. Sous Unix, il y a des fichiers spéciaux qui sont une suite infinie d'octets, notamment `/dev/zero` et `/dev/random`
2. En général c'est un programme, qui suit les choix de son programmeur.

2 The Good

Simple sur le plan conceptuel :

1. On accède au fichier (on dit qu'on l'*ouvre*) via un objet qui le représente (on dit que c'est un *objet fichier* ou un objet *de type fichier*).
2. On a des fonctions pour lire (un octet ou une ligne de texte ou tout ce qui reste à lire) ou écrire (un octet ou un texte).
3. L'accès est par défaut séquentiel : il faut imaginer une tête de lecture positionnée au début du fichier qui se déplace au fur et à mesure de la lecture ou de l'écriture.
4. On ferme le fichier.

Pour ouvrir un fichier, on utilise la fonction python `open(f, m)` où `f` est le nom du fichier et `m` est le *mode d'ouverture* :

- 'r' pour une ouverture en lecture seule.
- 'w' pour une ouverture en écriture seule (fichier tronqué à 0 octets s'il existe, créé s'il n'existe pas).
- 'a' pour une ouverture en ajout (tête d'écriture positionnée en fin de fichier, fichier créé s'il n'existe pas)

Exemple en lecture :

```
>>> f = open('/etc/passwd', 'r')
>>> x = f.readline()
>>> y = f.readline()
>>> print(x)
root:x:0:0:root:/root:/bin/bash

>>> print(y)
daemon:x:1:1:daemon:/usr/sbin:/bin/sh

>>> r = f.read()
>>> f.close()
```

Remarques :

1. `readline`, `read` et `close`
   - Ne sont pas des vraies fonctions
   - Sont attachées à l'objet `f`
   - Sont appelées des *méthodes* de l'objet.
2. `read` lit tout depuis la position courante jusqu'à la fin.
3. Il n'est pas nécessaire de tout lire pour fermer.
4. Lecture et écriture sur un fichier fermé donnent une erreur.
5. `readline` retourne toute une ligne, retour chariot inclus.
   Ici `x` vaut `'root:x:0:0:root:/root:/bin/bash\n'`

Écriture : même principe.

```
f = open('monfichier.txt', 'w')
f.write("Bonjour,\n")
f.write("Comment")
f.write("ça va ?")
f.close()
```

Contenu du fichier `monfichier.txt` après l'exécution :

```
Bonjour,
Commentça va ?
```

3 The Bad

Ne pas oublier `f.close()` !

- Le nombre de fichiers ouverts simultanément est limité.
- Les écritures n'ont pas lieu immédiatement :
  - Python attend d'en avoir suffisamment avant de transmettre à l'OS.
  - L'OS écrit quand ça lui semble approprié.

Un `f.close()` ferme le fichier et force le transfert à l'OS.

Autre façon de forcer l'écriture : `f.flush()`

Une instruction `with` a été introduite en python pour s'assurer de la fermeture des fichiers (en lecture comme en écriture) :

```
with open('monfichier.txt', 'w') as f:
    f.write("Bonjour,\n")
    f.write("Comment")
    f.write("ça va ?")
```

- à la sortie du bloc, python fait automatiquement un `f.close()`
- en cas d'erreur dans le bloc, `f` est fermé correctement.

4 The Ugly

Texte : comporte éventuellement des retours à la ligne.

Codage du retour à la ligne :

En mémoire (chaîne de caractères) : un caractère spécial (`\n`)

Dans les fichiers : Dépend du système

- **Unix :** Pareil : (`\n`)
- **Mac OS (avant 1999) :** Un autre caractère (`\r`)
- **Microsoft :** Deux caractères (`\n` puis `\r`).

Bizarrerie :

- Ayant peut-être des raisons en 1980 ou 1984.
- Corrigée par Apple dans Mac OS X
- Mais pas par Microsoft...

Sous Windows : traduction nécessaire dans le passage fichier/mémoire et vice-versa.

Problème corrigé par Microsoft au niveau du système, en proposant deux modes d'ouverture des fichiers (pour chaque mode déjà existant) :

Mode binaire Lit (resp. écrit) exactement ce qu'il y a dans le fichier (resp. la mémoire).

Mode texte Traduit les automatiquement les retours à la ligne lors des lectures/écritures.

Solution bancale car :

1. Source de bugs (ouverture d'un fichier dans le mauvais mode).
2. Complique la portabilité des programmes entre OS.
3. Pas viable à l'heure d'Internet (comment lire un fichier texte Microsoft sous une autre plateforme et vice-versa ?).

Solution Python :

1. On ajoute un caractère au mode pour demander une ouverture en texte ou en binaire : 't' ou 'b'.
2. Pour faciliter la portabilité, sous Unix, on ajoute aussi ce caractère mais il n'a aucune importance (il est ignoré).
3. En lecture en mode texte, à partir de la version 3, python comprend tous les types de retours à la ligne.

5 En résumé

`open(fichier, mode)` où `mode` est : 'rt', 'rb', 'wt', 'wb', 'at', ou 'ab'.

Méthodes utiles pour la lecture en mode texte :

- `readline()` : lit une ligne, la retourne sous forme de chaîne.
- `read()` : lit tout ce qui reste, le retourne sous forme de chaîne.
- `readlines()` : lit toutes les lignes restantes, les retourne sous forme de liste de chaînes.

Méthode utile pour l'écriture en mode texte : `write(chaine)`

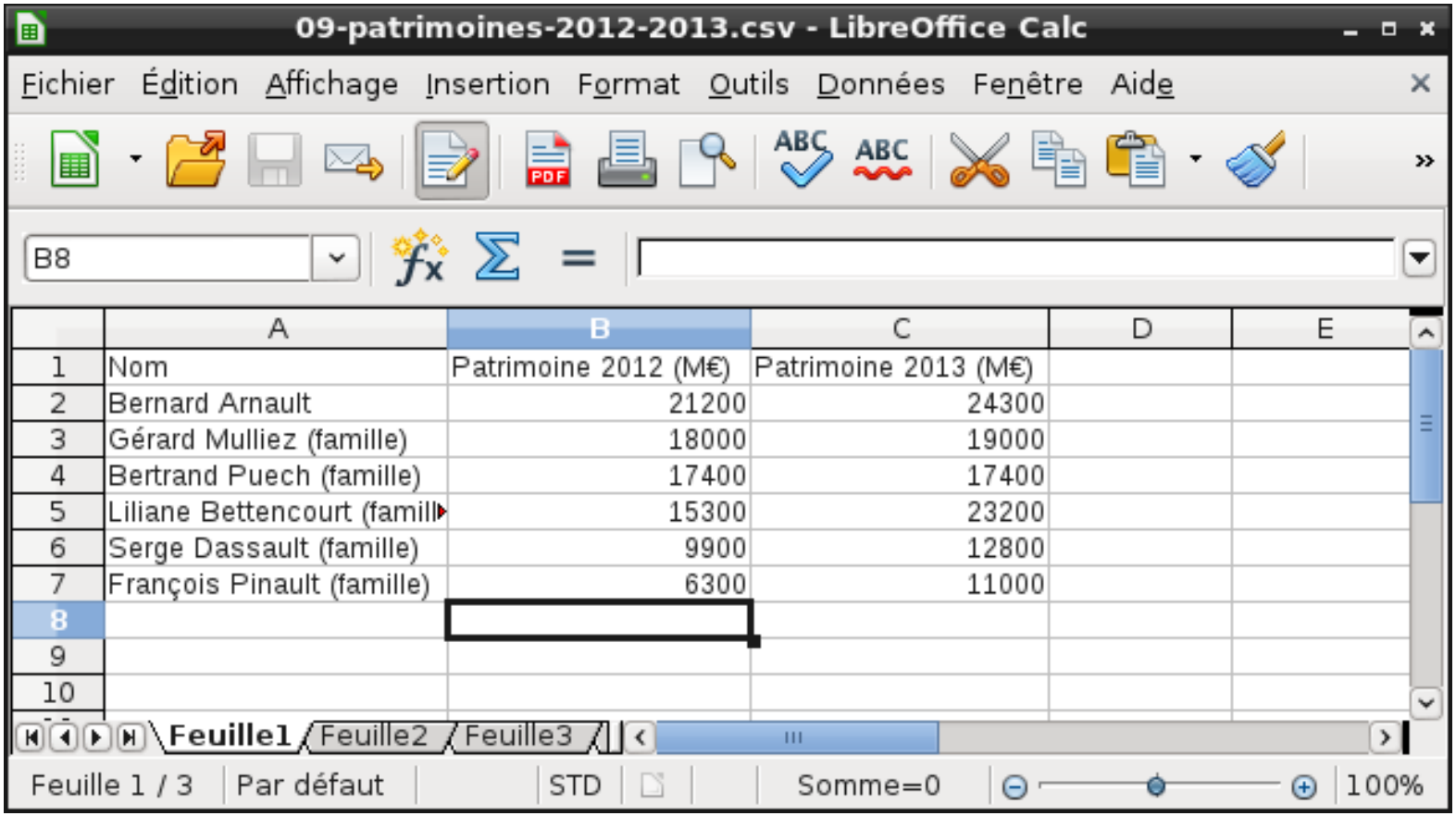
Ne pas oublier de fermer les fichiers ! (méthode `close()`)

6 Un exemple

On aimerait savoir combien une taxe de 75% sur la plus-value de leur patrimoine laisserait à certains contribuables [3].

Avertissement : les transparents suivants peuvent choquer.

3. À l'heure actuelle, une telle taxe est totalement imaginaire, une taxation à 75% des seuls revenus **au-dessus** d'un million étant encore en discussion.



	A	B	C	D	E
1	Nom	Patrimoine 2012 (M€)	Patrimoine 2013 (M€)		
2	Bernard Arnault	21200	24300		
3	Gérard Mulliez (famille)	18000	19000		
4	Bertrand Puech (famille)	17400	17400		
5	Liliane Bettencourt (famill	15300	23200		
6	Serge Dassault (famille)	9900	12800		
7	François Pinault (famille)	6300	11000		
8					
9					
10					

(source : `http://www.challenges.fr/classements/fortune/`)

Si on sauve ce fichier au format CSV (Comma Separated Values), on obtient un fichier contenant ceci :

```
Nom;Patrimoine 2012 (M€);Patrimoine 2013 (M€)
Bernard Arnault;21200;24300
Gérard Mulliez (famille);18000;19000
Bertrand Puech (famille);17400;17400
Liliane Bettencourt (famille);15300;23200
Serge Dassault (famille);9900;12800
François Pinault (famille);6300;11000
```

Lecture des données :

```
with open('09-patrimoines-2012-2013.csv', 'rt') as f:
  f.readline() # on ignore la première ligne
  d = f.readlines() # les données intéressantes
```

`d` contient donc un tableau contenant la liste des lignes du fichier csv, à l'exception de la première. Chaque ligne est représentée par une chaîne de caractère.

Voyons ce qu'on peut faire sur la première ligne :

```
>>> d[0]
'Bernard Arnault;21200;24300\n'
>>> s = d[0].strip() # enlevons le retour chariot.
>>> s
'Bernard Arnault;21200;24300'
>>> valeurs = s.split(';')
>>> valeurs
['Bernard Arnault', '21200', '24300']
```

On a donc maintenant dans `valeurs` la liste des valeurs de la première ligne, sous forme de chaîne.

```
>>> nom = valeurs[0]
>>> p2012 = float(valeurs[1]) # patrimoine 2012
>>> p2013 = float(valeurs[2]) # patrimoine 2013
>>> r = (p2013 - p2012) * 0.25 # reste après taxe
>>> ligne_resultat = nom + ';' + str(r) + '\n'
```

Généralisons :

```
def traite_ligne(li):
    v = li.strip().split(';')
    r = (float(v[2]) - float(v[1])) * 0.25
    return v[0] + ';' + str(r) + '\n'
```

Il suffit alors de traiter toutes les lignes :

```
with open('09-reste-apres-taxation.csv', 'wt') as f:
    f.write('Nom;Reste après taxe (M euros)\n')
    for x in d:
        f.write(traite_ligne(x))
```

Contenu du fichier `09-fichiers.csv` produit :

```
Nom;Reste après taxe (M euros)
Bernard Arnault;775.0
Gérard Mulliez (famille);250.0
Bertrand Puech (famille);0.0
Liliane Bettencourt (famille);1975.0
Serge Dassault (famille);725.0
François Pinault (famille);1175.0
```

Exercices annexes : écrire les fonctions utiles pour calculer

- à qui il reste le plus ;
- qui a payé le plus et combien (en pourcentage du déficit budgétaire 2013). Indication : le déficit budgétaire 2013 est d'environ 62 milliards d'euros.

Remarque : On doit évidemment trouver les mêmes personnes dans les deux cas et la somme trouvée est de l'ordre de 6 milliards, soit environ 10% du déficit budgétaire de l'état (année 2013).