Fichiers

7 juillet 2016

1 Introduction

En informatique, l'information se code comme une suite de bits (en général regroupés en octets). Sur le disque dur, cette information est répartie dans des *fichiers*.

Définition 1.0.1. Un fichier est une suite (généralement finie) d'octets [1].

Un fichier peut contenir tout type d'information. Comment alors interpéter les données contenu dans ce fichier ? Il n'existe pas de règle *a priori*.

Ainsi, c'est celui qui écrit le fichier qui choisit la méthode à utiliser pour comprendre ce fichier. [2] Celui qui lit le fichier doit utiliser la même méthode pour obtenir l'information.

Heureusement, il y a des conventions qui permettent à tous de s'y retrouver.

Nous allons maintenant nous intéresser à la question suivante : comment lire/écrire des données dans des fichiers depuis python ?

2 The Good

C'est simple sur le plan conceptuel.

1. On accède au fichier (on dit qu'on l'*ouvre*), en créant 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 le contenu du fichier (un octet ou une ligne de texte ou tout ce qui reste à lire) ou pour écrire (un octet ou un texte) dans un fichier.
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, c'est-à-dire que le système d'exploitation enregistre les éventuelles modifications sur le fichier, avant de détruire l'objet fichier créé dans 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.

Pour ouvrir un fichier, on utilise la fonction python `open(nom_de_fichier, m)` où

- `nom_de_fichier` est une chaîne de caractères (type `str`) contenant le nom du fichier (plus précisément : un nom de chemin—absolu ou relatif—vers ce fichier) ;
- `m` est une chaîne de caractères (type `str`) contenant le *mode d'ouverture* du fichier :
  - `'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).

Cette fonction retourne un *objet fichier* (appelé aussi *descripteur de fichier*) à partir duquel on va pouvoir lire ou écrire dans le fichier.

Voici un exemple de lecture d'un fichier :

```
>>> 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:/usr/sbin/nologin

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

Remarque 2.0.2.

1. `readline`, `read` et `close` ne sont pas de « vraies fonctions », mais sont rattachées à l'objet `f`. Ce sont des *méthodes* applicables à cet objet.
2. La méthode `read` lit tout le fichier, depuis la position courante jusqu'à la fin. Elle renvoie une chaîne de caractères.
3. Il n'est pas nécessaire de lire l'intégralité d'un fichier avant de le fermer.
4. Lecture et écriture sur un fichier fermé renvoient une erreur.
5. La méthode `readline` retourne toute une ligne (sous forme de chaîne de caractères), retour chariot inclus.
   Par exemple, ici, `x` vaut `'root:x:0:0:root:/root:/bin/bash\n'`

Pour écrire dans un fichier, c'est le même principe.

```
>>> f = open('monfichier.txt', 'w')
>>> f.write("Bonjour,\n")
9
>>> f.write("Comment")
7
```

```
>>> f.write("ça va ?")
7
>>> f.close()
```

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

```
Bonjour,
Commentça va ?
```

Attention, il convient de bien se souvenir que :

— `open(nom_de_fichier, 'w')` efface le fichier `nom_de_fichier` s'il existait déjà ;

— `open(nom_de_fichie, 'a')` ajoute à la fin du fichier `nom_de_fichier` s'il existait déjà.

3 The Bad

Il ne faut surtout pas oublier `f.close()` ! En effet,

- Le nombre de fichiers ouverts simultanément est limité.
- Les écritures n'ont pas lieu immédiatement :
  — Python attend d'en avoir suffisamment avant de transmettre au système d'exploitation ;
  — Le système d'exploitation écrit quand cela lui semble approprié.

Un `f.close()` ferme le fichier et force le transfert au système d'exploitation.
Il existe une autre façon de forcer l'écriture : `f.flush()` (ne dispense pas de fermer).

Surtout, une instruction `with` a été introduite en python pour s'assurer de la fermeture des fichiers (en lecture comme en écriture). Par exemple, on peut réécrire la liste d'instructions précédentes comme suit.

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

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

4 The Ugly

Une question important est de choisir la convention à utiliser pour traduire un texte en suite d'octets. Malheureusement, il n'y a pas une norme, mais plusieurs ...

4.1 ASCII

Dans les années (19)60, la norme ASCII [3] est introduit, pour représenter les caractères. Chaque caractère codé par un entier dans $[\![0, 128[\![$ et peut donc être codé sur 7 bits, donc sans problème sur un octet. Cette norme fut très vite adoptée.

Les caractères 0 à 31 ainsi que 127 ne sont pas des caractères à proprement parler, mais représentent des instructions (à destinations des imprimantes, par exemple). La liste des caractères est donnée dans la figure 1

N°	Symbole	N°	Symbole	N°	Symbole	N°	Symbole	N°	Symbole
32		52	4	72	H	92	\	112	p
33	!	53	5	73	I	93	]	113	q
34	"	54	6	74	J	94	^	114	r
35	#	55	7	75	K	95	_	115	s
36	$	56	8	76	L	96	`	116	t
37	%	57	9	77	M	97	a	117	u
38	&	58	:	78	N	98	b	118	v
39	’	59	;	79	O	99	c	119	w
40	(	60	<	80	P	100	d	120	x
41	)	61	=	81	Q	101	e	121	y
42	*	62	>	82	R	102	f	122	z
43	+	63	?	83	S	103	g	123	{
44	,	64	@	84	T	104	h	124	|
45	-	65	A	85	U	105	i	125	}
46	.	66	B	86	V	106	j	126	~
47	/	67	C	87	W	107	k		
48	0	68	D	88	X	108	l		
49	1	69	E	89	Y	109	m		
50	2	70	F	90	Z	110	n		
51	3	71	G	91	[	111	o		

FIGURE 1 – Table des caractères ASCII

4.1.1 A comme ...

ASCII : American Standard Code for Information Interchange.
Ici, «American» veut dire «États-Unien», donc :

— sans l’Amérique latine (español, português)

— ni le Québec (de langue française)

Bref, aucune lettre accentuée n’est disponible...

3. Prononcez : «à ski».

4.2 Les tentatives d'amélioration

Dans un octet, on peut avoir 256 valeurs distinctes. On peut donc coder 256 caractères sur un octet. Le code ASCII ne prend que 128 valeurs, il y a donc 128 places restantes. La norme ISO 8859-1 (Latin 1) utilise la place disponible pour ajouter les accents. Mais :

— cela ne suffit pas pour l'Europe Centrale $\rightarrow$ ISO 8859-2 (différent) ;

— Et pour le turc et le maltais et l'esperanto ? ISO 8859-3

— Et les pays baltes ? ISO 8859-4

— En russe, on n'utilise pas un alphabet bizarre ? ISO 8859-5

— Et en arabe ? ISO 8859-6

— Et en grec ? ISO 8859-7

— Et l'hébreu ? ISO 8859-8

— . . . AAAAAAARGH !

4.2.1 Et bien sûr. . .

Fidèle à sa stratégie commerciale «Embrace, extend & extinguish», Microsoft a fait ses propres versions :

CP437, CP737, CP850, CP852, CP855, CP857, CP858, CP860, CP861, CP862, CP863, CP865, CP866, CP869, CP872, Windows-1250, Windows-1251, Windows-1252, Windows-1253, Windows-1254, Windows-1255, Windows-1256, Windows-1257, Windows-1258.

. . . et a décidé de représenter les retours à la ligne différemment de tous les autres (c'est plus simple pour les imprimantes matricielles. . .).

4.3 La solution : Unicode et . . .

À la fin des années (19)80 et au début des années (19)90, l'idée de standardiser un codage universel s'est répandue.

En 2012, on trouve $1,1 \times 10^5$ «caractères» définis, couvrant 100 écritures. Chacun est repéré par un «point de code» (*code point*), entier de $[\![0, 1\,114\,112[\![$.

Mais, cela ne tient pas sur un octet, ni même sur deux. En Python, une chaîne de caractère peut être vue comme un tableau de points de code unicode.

4.4 . . . UTF-8

C'est un code de taille variable, de 1 à 4 octets. Il est compatible avec ASCII : un caractère ASCII est codé en UTF-8 sur un octet, de la même manière qu'en ASCII.

Actuellement, c'est le format le plus utilisé pour les pages web. Notamment, c'est le format canonique des fichiers textes sur toutes les plateformes modernes (sauf Microsoft). Il est disponible sur toutes les plateformes, y compris Microsoft.

4.5 La lecture de fichiers textes en Python

Ce qui suit est valable pour Python à partir de la version 3
Python tente d'être aussi raisonnable que possible :

1. Quand il lit/écrit un fichier, il décode/code les textes en utilisant le codage standard du système sur lequel il tourne (donc UTF-8 sauf pour sous MS-Windows ; Windows-1252 sur un PC français sous MS-Windows).
2. `open` dispose d'une option permettant de spécifier l'encodage à utiliser. Par exemple
   ```
   open('monfichier.txt', 'r', encoding='utf-8')
   ```

NB : Il est possible d'ouvrir un fichier en mode binaire (`open(..., 'rb')`). La lecture retourne alors des objets de type `bytes` (tableaux d'octets), les fonctions pour écrire prennent en argument des `bytes`.

5 En résumé

Pour ouvrir un fichier en mode texte :

```
with open(nom_de_fichier, mode) as f:
    # faire ce qu'on veut avec f
# ici f a été fermé.
```

- `nom_de_fichier` : type `str`, chemin désignant le fichier.
- `mode` : type `str`, valeurs possibles : `'r'`, `'w'`, et `'a'`.

Méthodes utiles pour la lecture en mode texte :

— `f.readline()` : lit une ligne, la retourne sous forme de chaîne.

— `f.read()` : lit tout ce qui reste, le retourne sous forme de chaîne.

— `f.readlines()` : lit toutes les lignes restantes, les retourne sous forme de liste de chaînes.

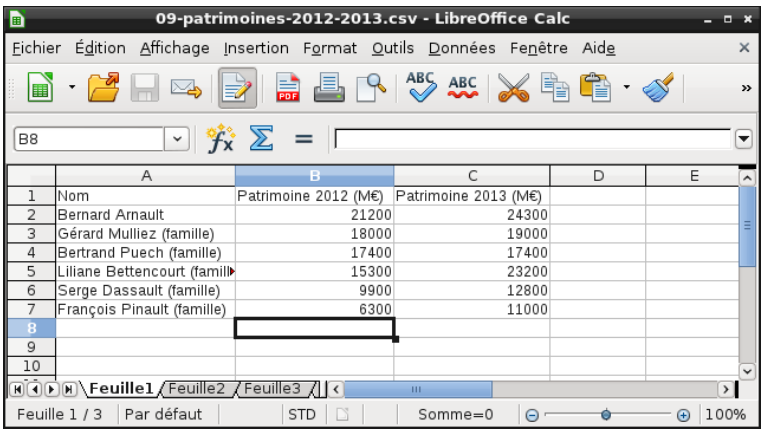
Méthode utile pour l'écriture en mode texte : `f.write(chaine)`
Ne pas oublier de fermer les fichiers si on ne les pas ouverts avec `with` ! (méthode `f.close()`)

6 Un exemple

On aimerait savoir combien une taxe de 75% sur la plus-value de leur patrimoine laisserait à certains contribuables [4].

Avertissement : les transparents suivants peuvent choquer.

4. À l'heure actuelle, une telle taxe est totalement imaginaire, la taxation à 75% des seuls **revenus au-dessus** d'un million étant au cimetière des promesses de 2012.



	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('./07-fichiers/patrimoines-2012-2013.csv', 'r') 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]

>>> s = d[0].strip() # enlevons le retour chariot.
>>> s
>>> valeurs = s.split(';')
>>> valeurs
```

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

```
>>> 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 :

```
def main():
    with open('patrimoines-2012-2013.csv', 'r') as f:
        f.readline()
        d = f.readlines()
    with open('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))

main()
```

Contenu du fichier `reste-apres-taxation.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).