Lecture et écriture de fichiers en Python

Ivan Noyer

Lycée Leconte de Lisle - Saint-Denis.

Crédits

- http://fr.openclassrooms.com/informatique/cours/apprenez-a-programmer-en-python/les-fichiers-2
- http://www.lamsade.dauphine.fr/~manouvri/PYTHON/TPPython.pdf

Connaître la position du script pour l'interpréteur

- Importer le module os.
- Savoir où je suis :

```
1 import os
2 os.getcwd()#ou se trouve le script utilise
```

```
1 '/home/ivan/CPGE/Info/CoursInfo/ReadWrite'
```

- Changer le répertoire courant :

```
1 os.chdir('../Dictionnaires')#le repertoire par def
2 os.getcwd()#'/home/ivan/CPGE/Info/CoursInfo/Dictio
```

Pythonisme

```
1 def f(l):
2     if l :
3         print( 1)
4     else :
5         print (2)
6
7 f([1]) # affiche 1
8 f([])  #affiche 2
```

Le test `if l` signifie donc « si `l` est non vide ».

Création d'un objet fichier en lecture

- Dans un fichier `essai` *texte* écrivons :

```
coucou
papa
toto
```

- On utilise la fonction `open` disponible sans rien devoir importer. Le chemin d'accès *relatif* au fichier est entré entre quotes. On choisit l'option « r » (read) pour « lecture en mode texte ».

```
1 f = open('essai','r')
2 s = f.read()
3 print( s)
4 f.close()#fermeture du fichier
```

Contenu exhaustif, fermeture

- Retour :

```
coucou
papa
toto
```

- Ne pas oublier de fermer le fichier après utilisation.
- Si on a un doute, on peut vérifier que le fichier est bien fermé :

```
1 print (f.closed) #True : ferme, False : ouvert
```

```
True
```

Lecture ligne par ligne

- On peut lire le fichier ligne par ligne en suivant l'algo suivant :

```
ligne = ligne un du fichier
afficher(ligne)
tant que ligne existe faire :
    ligne=ligne suivante
    afficher(ligne)
```

- En Python :

```
f = open('essai','r')
l = f.readline()
print (l),
while l:
    l=f.readline()
    print (l),
print ('ceci n\'est pas dans le fichier')
f.close()
```

Lecture ligne par ligne (suite)

- ```
  coucou
  papa
  toto
  ceci n'est pas dans le fichier
  ```
- Dans `essai`, pour `Python` : 4 lignes + une ligne vide.
  ```

Lecture en mode binaire

- En mode texte, un fichier est lu caractère par caractère.
- En mode binaire, un fichier est lu octet par octet : c'est un peu différent car les caractères sont parfois codés sur plus d'un octet !
- Utilité : par exemple, si on veut la taille d'un fichier, il faut utiliser la lecture en mode binaire 'rb' (read binary) :

Lecture en mode binaire

```
1 #ouverture en mode read binary 'rb'
2 Fichier = open('essai','rb')
3 data = Fichier.read()
4 # affichage de la taille du fichier
5 print ('Taille du fichier :',len(data),'octets')
6 # affichage du contenu (en hexadecimal)
7 import binascii
8 print ('contenu en hexadecimal : ' +\
9 binascii.hexlify(data))
10 #hexlify:Hexadecimal representation of binary data.
11 # fermeture du fichier avec la methode close()
12 Fichier.close()
```

On obtient :

```
Taille du fichier : 16 octets
contenu en hexadecimal : 436f75636f75....
```

Deux modes d'écriture

- L'option « `'w'` » : ouvrir un fichier en mode écriture/écrasement. Le texte initial est perdu.
- L'option « `'a'` » : ouvrir un fichier en mode écriture/ajout. Le nouveau texte vient à la suite du texte initial.
- Dans tous les cas, si le fichier n'existe pas, il est créé.

Exemple

```
1 fich=open('essai','a')
2 fich.write('\ntiti')#écrire titi à la fin
3 fich.close()
4 fich=open('essai','r')
5 print (fich.read())
6 fich.close()
```

On obtient

```
coucou
papa
toto
titi
```

Remarquons que la méthode `write` ne prend en paramètres que des chaînes de caractères. Pour écrire un nombre, le transformer en `str`.

Présentation du problème

Quand on ouvre un fichier avec :

```
1 f = open("notes.txt", "r")
2 # ... on lit ou écrit ...
3 f.close()
```

- on doit penser soi-même à appeler `f.close()` pour libérer le fichier.

Présentation du problème

Quand on ouvre un fichier avec :

```
1 f = open("notes.txt", "r")
2 # ... on lit ou écrit ...
3 f.close()
```

- on doit penser soi-même à appeler `f.close()` pour libérer le fichier.
- Si on oublie, ou si une erreur se produit avant le `close()`, le fichier peut rester ouvert, ce qui pose parfois problème (fichier verrouillé, données non écrites, etc.).

L'instruction `with`

En Python, on peut écrire :

```
1 with open("notes.txt", "r") as f:
2     # on utilise f ici
3     contenu = f.read()
```

Dès que l'exécution sort du bloc, Python ferme automatiquement le fichier, même si une erreur s'est produite à l'intérieur. sûr, plus propre

Pourquoi l'utiliser

- plus de sécurité : le fichier est toujours fermé correctement ;
- moins d'oublis : pas de close() à écrire ;
- code plus lisible : on voit immédiatement quelles lignes utilisent le fichier ;
- bonnes pratiques : c'est la manière ≪ professionnelle ≫ standard.