Document ressource

python

Les fichiers

1 Préambule

Les données informatiques (*datas*) qui doivent être mémorisées très longtemps sont enregistrées sous forme structurée dans ce qu'on appelle communément des fichiers (*files*).
Ces fichiers sont physiquement enregistrés sur des supports - dits de stockage de masse – comme le disque dur, une clé usb, un dvd, le réseau, etc...).

Quand un programme doit traiter ces données, il est hors de question de les manipuler octet par octet directement sur le support, ce serait trop compliqué et surtout très lent. Le principe est de dupliquer le fichier qui nous intéresse en RAM (mémoire dynamique rapide), de le traiter puis de le stocker à nouveau (si nécessaire) sur un support physique non volatile.

RAM (Random Access Memory) : mémoire volatile, qui s'efface si on éteint l'ordinateur. C'est la mémoire de travail qui permet des vitesses d'accès et donc exécution des milliers de fois plus rapides comparées avec les supports de masse.

2 Ouvrir et fermer un fichier

La méthode générale avec Python mais aussi de nombreux langages est :

- Ouvrir un fichier : cela permet d'accéder aux données du fichier et d'empêcher qu'un autre utilisateur le modifie en même temps que vous.
- Faire vos modification en RAM.
- Les enregistrer sur le fichier d'origine ou sur une copie.
- Fermer le fichier pour le libérer.

En python on utilise deux méthodes fondamentales : **open** et **close**.

- ***Open*** permet d'accéder aux données d'un fichier.
- ***Close*** permet de fermer et de libérer ce fichier.

2.1 Ouvrir le fichier

Le plus simple pour commencer est de mettre le fichier dans le même répertoire que votre programme (voir le § T*rouver son chemin* pour les autres cas) .

La fonction **open** contient 2 arguments au minimum : le premier est le nom du fichier avec **son extension** entre guillemets et le second est le mode d'ouverture.

Exemple : ***fichier_travail = open('fichier_sur_disque.txt', 'r')***

Les modes d'ouverture sont :

'r' pour *read :* donne la possibilité de lire dans le fichier.
'w' pour *write* :donne la possibilité d'écrire dans le fichier.
'a' pour *append* : donne la possibilité d'ajouter à un fichier.
'r+' : donne la possibilité de lire et d'écrire.

Ainsi 'a' permet de compléter un fichier alors que 'w' **écrase** ce que contenait le fichier et écrit au début.

Dans notre exemple on a ouvert le fichier nommé *fichier_sur_disque.txt* situé dans le répertoire courant, en lecture, c'est à dire qu'on ne pourra pas le modifier.
Les données sont aussi dupliquées dans la variable de type fichier nommée ici *fichier_travail*.

2.2 Enregistrer et fermer le fichier

Pour écrire dans un nouveau fichier (ou le même) il faut faire l'opération inverse avec un fichier ouvert au préalable en mode 'w', write.

Exemple(que vous pouvez tester !) :

```
f = open("fichier_sur_disque.txt","w")        # ouvre (ou crée) un fichier en mode écriture
                                              # et copie les données dans f, si le fichier existait déjà.
s='texte écrit dans le fichier\n'             # on crée une chaîne de caractère avec du texte
f.write(s)                                    # on écrit la chaîne de caractères s dans f
f.close()                                     # on ferme le fichier qui contient alors la chaîne
```

2.3 Traiter les données du fichier

On a maintenant un objet de type file qu s'appelle *fichier_travail.txt* dans notre exemple, mais comment les lire ?

On utilise la méthode read : **s=f.read()**

On récupère dans la chaîne s de type *string* tous les caractères contenus dans le fichier.
Pour lire ces caractères il faut ensuite parcourir s avec son indice.

Exemple : 1er caractère : s[0]

Pour écrire, on créée une chaîne de caractères (n'oubliez pas qu'on ne peut pas modifier une chaîne, seulement l'allonger ou partir d'une chaîne vide) et on l'écrit dans le fichier :

```
fichier_travail.write(s)  # on écrit la chaîne de caractère dans le fichier
fichier_travail.close()   # on ferme le fichier, et onle libére pour d'autres utilisations.
```

En mode 'r' ou 'r+', on peut lire avec des méthodes plus fines : **read**, **readline**, **readlines**.

- **read** : f.**read**() ou f.**read**(nombre de caractères à lire)

Si on utilise plusieurs fois cette méthode, la lecture reprend là où on l'avait laissée.

- **readline** : f.**readline**() retourne toute la ligne, y compris le caractère de fin de ligne \n
- **readlines** : f.**readlines**() retourne une liste de chaînes de caractères.

Si on veut ensuite travailler avec des nombres, on pourra utiliser les fonctions eval, int, float par exemple.

Informations avancées

Tous les fichiers ne sont pas forcément codés sous forme de caractères. Si on a besoin de lire un fichier codé en binaire, il faut le lire avec les paramètres ***'w+b'*** ou ***'r+b'***

La méthode seek : elle permet de se déplacer dans le fichier sans le lire:
f.seek(nombre de caractères dont on veut se déplacer à partir du début du fichier)

3 TROUVER SON CHEMIN (INFORMATIONS AVANCÉES)

La difficulté la plus courante est de trouver le chemin et le nom du fichier. Pour cela, deux instructions utiles, importées du module os (comme *operating system*):

from os import getcwd
getcwd () indique le répertoire de travail actuel (get current working directory)

ATTENTION : si dans votre programme vous utilisez l'instruction ***from os import **** vous importez une fonction *open* (**qui a le même nom, quelle embrouille** !) et qui ne fonctionne pas comme la méthode open décrite plus haut. Grrr !!

from os import chdir
chdir permet de changer le répertoire courant (change directory)

ex : `chdir("C:\\Users\\Truc...Machin\Desktop")` # On se place sur le bureau

Notez que le double \\ sera transformé en simple \ devant « users ». En fait les \ qui servent de séparateurs sous Windows peuvent être interprétés par Python, comme indiqué dans le tableau ci dessous :

Représentation	Description
\n	Line Feed (retour à la ligne)
\r	Carriage Return
\r\n	Carriage Return + Line Feed
\v or \x0b	Line Tabulation
\f or \x0c	Form Feed
\x1c	File Separator
\u2029	Paragraph Separator
\t	tabulation

Pour simplifier le travail dans un premier temps, il est plus prudent de copier les fichiers traités dans le répertoire courant (qu'on peut donc modifier avec la commande chdir) et d'appeler ensuite le fichier sans son chemin.

N'oubliez pas **l'extension** dans le nom !