Chapitre 12 : Gestion des fichiers

1. Introduction

Le défaut avec les variables, c'est qu'elles n'existent que dans la mémoire vive. Une fois votre programme arrêté, toutes vos variables sont supprimées de la mémoire et il n'est pas possible de retrouver ensuite leur valeur.

Heureusement, on peut lire et écrire dans des fichiers en langage C. Ces fichiers seront écrits sur le disque dur de votre ordinateur : l'avantage est donc qu'ils restent là, même si vous arrêtez le programme ou l'ordinateur.

2. Ouverture d'un fichier

- **Fonction .open() :**

Pour travailler avec le contenu des fichiers (lire ou écrire), il faut « ouvrir » ces fichiers avec la fonction « open() ». Cette fonction prend obligatoirement deux paramètres : le fichier (avec son chemin relatif ou absolu), et le mode d'ouverture. C'est le « mode d'ouverture » qui défini si on va simplement lire le contenu du fichier, ou si on va écrire dedans.

Syntaxe :

```
Fichier = open(chemin , mode_ouverture )
```

- **Mode d'ouverture :**

- **r :** le mode en lecture ici on ouvre le fichier juste pour la lecture et l'obtention de l'information si le fichier n'existe pas une exception se déclenche
- **w :** le mode en écriture ici on ouvre le fichier pour écrire dedans si le fichier existe déjà il écrase le contenu précédent et le remplace par le nouveau contenu. si le fichier n'existe pas il crée un

- **a :** le mode en ajout ici on ouvre un fichier existant et on ajoute le nouveau contenu après la dernière ligne. si le fichier n'existe pas on crée un comme dans le mode w

3. Lecture dans un fichier

Une grande partie de l'information en biologie est stockée sous forme de texte dans des fichiers. Pour traiter cette information, vous devez le plus souvent lire ou écrire dans un ou plusieurs fichiers. Python possède pour cela de nombreux outils qui vous simplifient la vie.

- **Méthode readlines() et .close() :**

Avant de passer à un exemple concret, créez un fichier dans un éditeur de texte que vous enregistrerez dans votre répertoire courant avec le nom zoo.txt et le contenu suivant :

```
girafe
tigre
singe
souris
```

Ensuite, testez le code suivant dans l'interpréteur Python :

```
>>> filin = open("zoo.txt", "r")
>>> filin
<_io.TextIOWrapper name='zoo.txt' mode='r' encoding='UTF-8'>
>>> filin.readlines()
['girafe\n', 'tigre\n', 'singe\n', 'souris\n']
>>> filin.close()
>>> filin.readlines()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file.
```

Il y a plusieurs commentaires à faire sur cet exemple :

- **Ligne 1 :** L'instruction open() ouvre le fichier zoo.txt. Ce fichier est ouvert en lecture seule, comme l'indique le second argument r (pour *read*) de la fonction open(). Remarquez que le fichier n'est pas encore lu, mais simplement ouvert (*un peu comme lorsqu'on ouvre un livre, mais qu'on ne l'a pas encore lu*). Le curseur de lecture est prêt à lire le premier caractère du fichier. L'instruction open("zoo.txt", "r") suppose que le fichier zoo.txt est dans le répertoire depuis lequel l'interpréteur Python a été lancé. Si ce n'est pas le cas, il faut préciser le **chemin d'accès** au fichier. Par exemple, /home/pierre/zoo.txt pour Linux ou Mac OS X ou C:\Users\pierre\zoo.txt pour Windows.

- **Ligne 2 :** Lorsqu'on affiche le contenu de la variable filin, on se rend compte que Python la considère comme un objet de type fichier ouvert **(ligne 3).**

- **Ligne 4 :** Nous utilisons à nouveau la syntaxe objet.méthode() (présentée dans le chapitre 3 *Affichage*). Ici la méthode .readlines() agit sur l'objet filin en déplaçant le curseur de lecture du début à la fin du fichier, puis elle renvoie une liste contenant toutes les lignes du fichier .

- **Ligne 5 :** Enfin, on applique la méthode .close() sur l'objet filin, ce qui, vous vous en doutez, ferme le fichier (*ceci correspondrait à fermer le livre*). Vous remarquerez que la méthode .close() ne renvoie rien mais modifie l'état de l'objet filin en fichier fermé. Ainsi, si on essaie de lire à nouveau les lignes du fichier, Python renvoie une erreur car il ne peut pas lire un fichier fermé (lignes 7 à 10).

Voici maintenant un exemple complet de lecture d'un fichier avec Python.

```
>>> filin = open("zoo.txt", "r")
>>> lignes = filin.readlines()
>>> lignes
['girafe\n', 'tigre\n', 'singe\n', 'souris\n']
>>> for ligne in lignes:
...     print(ligne)
...
girafe

tigre

singe

souris

>>> filin.close()
```

Vous voyez qu'en cinq lignes de code, vous avez lu, parcouru le fichier et affiché son contenu.

Remarque :

- Chaque élément de la liste lignes est une chaîne de caractères. C'est en effet sous forme de chaînes de caractères que Python lit le contenu d'un fichier.
- Chaque élément de la liste lignes se termine par le caractère \n. Ce caractère un peu particulier correspond au « saut de ligne » qui permet de passer d'une ligne à la suivante. Ceci est codé par un caractère spécial que l'on représente par \n. Vous pourrez parfois rencontrer également la notation octale \012. Dans la suite de cet ouvrage, nous emploierons aussi l'expression « retour à la ligne » que nous trouvons plus intuitive.
- Par défaut, l'instruction print() affiche quelque chose puis revient à la ligne. Ce retour à la ligne dû à print() se cumule alors avec celui de la fin de ligne (\n) de chaque ligne du fichier et donne l'impression qu'une ligne est sautée à chaque fois.
- **Le mot clé « with » :**

Il existe en Python le mot-clé with qui permet d'ouvrir et de fermer un fichier de manière efficace. Si pour une raison ou une autre l'ouverture ou la lecture du fichier conduit à une erreur, l'utilisation de with garantit la bonne fermeture du fichier, ce qui n'est pas le cas dans le code précédent. Voici donc le même exemple avec with :

```
>>> with open("zoo.txt", 'r') as filin:
...     lignes = filin.readlines()
...     for ligne in lignes:
...         print(ligne)
...
girafe

tigre

singe

souris

```

Remarque :

- L'instruction with introduit un bloc d'indentation. C'est à l'intérieur de ce bloc que nous effectuons toutes les opérations sur le fichier.
- Une fois sorti du bloc d'indentation, Python fermera **automatiquement** le fichier. Vous n'avez donc plus besoin d'utiliser la méthode .close().

- **Méthode .read() :**

Il existe d'autres méthodes que .readlines() pour lire (et manipuler) un fichier. Par exemple, la méthode .read() lit tout le contenu d'un fichier et renvoie une chaîne de caractères unique.

```
>>> with open("zoo.txt", "r") as filin:
...     filin.read()
...
'girafe\ntigre\nsinge\nsouris\n'
>>>
```

- **Méthode .readline() :**

La méthode .readline() (sans s à la fin) lit une ligne d'un fichier et la renvoie sous forme de chaîne de caractères. À chaque nouvel appel de .readline(), la ligne suivante est renvoyée. Associée à la boucle while, cette méthode permet de lire un fichier ligne par ligne.

```
>>> with open("zoo.txt", "r") as filin:
...     ligne = filin.readline()
...     while ligne != "":
...         print(ligne)
...         ligne = filin.readline()
...
girafe

tigre

singe

souris
```

- **Itérations directe sur le fichier :**

Python essaie de vous faciliter la vie au maximum. Voici un moyen à la fois simple et élégant de parcourir un fichier.

```
>>> with open("zoo.txt", "r") as filin:
...     for ligne in filin:
...         print(ligne)
...
girafe

tigre

singe

souris
```

L'objet filin est « itérable », ainsi la boucle for va demander à Python d'aller lire le fichier ligne par ligne.

Remarque :

Les méthodes abordées précédemment permettent d'accéder au contenu d'un fichier, soit ligne par ligne (méthode .readline()), soit globalement en une seule chaîne de caractères (méthode .read()), soit globalement avec les lignes différenciées sous forme d'une liste de chaînes de caractères (méthode **.readlines()**). Il est également possible en Python de se rendre à un endroit particulier d'un fichier avec la méthode **.seek()** mais qui sort du cadre de cet ouvrage.

4. Écriture dans un fichier :

- **Méthode .write()** :

Écrire dans un fichier est aussi simple que de le lire. Voyez l'exemple suivant :

```
1  >>> animaux2 = ["poisson", "abeille", "chat"]
2  >>> with open("zoo2.txt", "w") as filout:
3  ...     for animal in animaux2:
4  ...         filout.write(animal)
5  ...
```

Quelques commentaires sur cet exemple :

- **Ligne 1 :** Création d'une liste de chaînes de caractères animaux2.
- **Ligne 2 :** Ouverture du fichier zoo2.txt en mode écriture, avec le caractère w pour *write*. L'instruction with crée un bloc d'instructions qui doit être indenté.
- **Ligne 3 :** Parcours de la liste animaux2 avec une boucle for.
- **Ligne 4 :** À chaque itération de la boucle, nous avons écrit chaque élément de la liste dans le fichier. La méthode .write() s'applique sur l'objet filout. Notez qu'à chaque utilisation de la méthode .write(), celle-ci nous affiche le nombre d'octets (équivalent au nombre de caractères) écrits dans le fichier (lignes 6 à 8). Ceci est valable uniquement dans l'interpréteur, si vous créez un programme avec les mêmes lignes de code, ces valeurs ne s'afficheront pas à l'écran.

Si nous ouvrons le fichier zoo2.txt avec un éditeur de texte, voici ce que nous obtenons :

poissonabeillechat

Ce n'est pas exactement le résultat attendu car implicitement nous voulions le nom de chaque animal sur une ligne. Nous avons oublié d'ajouter le caractère fin de ligne après chaque nom d'animal.

Pour ce faire, nous pouvons utiliser l'écriture formatée :

```
1  >>> animaux2 = ["poisson", "abeille", "chat"]
2  >>> with open("zoo2.txt", "w") as filout:
3  ...     for animal in animaux2:
4  ...         filout.write("{}\n".format(animal))
5  ...
```

- **Ligne 4 :** L'écriture formatée vue au chapitre 3 *Affichage* permet d'ajouter un retour à la ligne (\n) après le nom de chaque animal.
- **Lignes 6 à 8 :** Le nombre d'octets écrits dans le fichier est augmenté de 1 par rapport à l'exemple précédent car le caractère retour à la ligne compte pour un seul octet.

Le contenu du fichier zoo2.txt est alors :

```
1  poisson
2  abeille
3  chat
```

Vous voyez qu'il est extrêmement simple en Python de lire ou d'écrire dans un fichier.

5. Ouvrir deux fichiers avec l'instruction with :

On peut avec l'instruction with ouvrir deux fichiers (ou plus) en même temps. Voyez l'exemple suivant :

```
1  with open("zoo.txt", "r") as fichier1, open("zoo2.txt", "w") as fichier2:
2      for ligne in fichier1:
3          fichier2.write("* " + ligne)
```

Si le fichier zoo.txt contient le texte suivant :

```
souris
girafe
lion
singe
```

alors le contenu de zoo2.txt sera :

```
* souris
* girafe
* lion
* singe
```

Dans cet exemple, with permet une notation très compacte en s'affranchissant de deux méthodes .close().

6. Importance des conversions de types avec les fichiers

Vous avez sans doute remarqué que les méthodes qui lisent un fichier (par exemple .readlines()) vous renvoient systématiquement des chaînes de caractères. De même, pour écrire dans un fichier il faut fournir une chaîne de caractères à la méthode .write().

Pour tenir compte de ces contraintes, il faudra utiliser les fonctions de conversions de types vues au chapitre 2 *Variables* : int(), float() et str(). Ces fonctions de conversion sont essentielles lorsqu'on lit ou écrit des nombres dans un fichier.

En effet, les nombres dans un fichier sont considérés comme du texte, donc comme des chaînes de caractères, par la méthode .readlines(). Par conséquent, il faut les convertir (en entier ou en *float*) si on veut effectuer des opérations numériques avec.

7. Méthodes utiles :

- **Méthode .seek(position) :**

C’est une méthode qui prend en paramètre une position et le curseur se positionne dans cette position et retourne le reste de la ligne ou du fichier.

- **Méthode .tell() :**

Cette méthode nous informe sur la position du curseur dans cette instant. Elle ne prend en paramètre rien et renvoie la position actuelle

- **Méthode .splitines() :**

Cette méthode comme la méthode .split() vu dans les chaines de caractère et qui retourne une liste des ligne découpe une chaine des caractères selon les saut de ligne .

8. Exercices :

Exercice 1 : Gestion des notes :

Le fichier notes.txt contient les notes obtenues par des étudiants pour le cours de Python. Chaque ligne du fichier ne contient qu'une note.

1- créez le fichier notes.txt contenant la note de chaque stagiaire et enregistrez-le dans votre répertoire de travail.

2- Créez un programme Python qui lit chaque ligne de ce fichier, extrait les notes sous forme de *float* et les stocke dans une liste.

3- Calculez et afficher la moyenne des notes avec deux décimales.

4- Le programme réécrira ensuite les notes dans le fichier notes2.txt avec une note par ligne suivie de « redoublant » si la note est inférieure à 10 et « admis » si la note est supérieure ou égale à 10. Toutes les notes seront écrites avec une décimale.