Chapitre 5
Manipulation des données

1. Lire et écrire dans des fichiers

Écrire dans un fichier peut être utile pour plusieurs raisons. Par exemple, pour y placer les résultats d'une tâche particulière : un score, des données issues d'un calcul... L'écriture dans un fichier est très utile lorsque l'on traite une grande quantité de données à déboguer. Une fois les données exportées dans un fichier, il est possible d'en faire l'analyse a posteriori. De cette manière, on peut affiner les conditions de réalisation d'un bogue et finalement créer un test qui permet de reproduire lesdites conditions.

Mais avant d'effectuer ces manipulations, il est nécessaire d'ouvrir un fichier.

1.1 Ouvrir et fermer un fichier

Ouvrir un fichier ou toute manipulation de fichier en langage Python est assez aisé. Il n'est pas nécessaire d'importer un quelconque package ou module pour réaliser cette tâche, il suffit d'écrire l'instruction suivante :

```
<mon_fichier> = open(<nom_fichier_a_ouvrir>,<mode>)
```

La méthode `open()` retourne un objet (`<mon_fichier>`) pour manipuler le fichier nommé `<nom_fichier_a_ouvrir>` selon le mode de fonctionnement précisé à l'aide de `<mode>`.

<nom_fichier_a_ouvrir> est une chaîne de caractères qui comporte l'adresse du fichier sur le disque.

Les modes précisés par <mode> peuvent être :

- 'w' ouverture en mode écriture
- 'r' ouverture en mode lecture
- 'a' ajout des données en fin de fichier quand le fichier est ouvert en mode écriture
- 'b' pour préciser que le fichier manipulé est un fichier binaire

Il existe d'autres modes, mais les modes précédemment cités sont les plus usités. Notez que les modes sont cumulables. Par exemple, le mode wab signifie que le fichier est ouvert en mode écriture, que les données sont placées en fin de fichier et que le fichier manipulé est un fichier binaire.

La tâche complémentaire à l'ouverture est la fermeture. La fermeture se réalise grâce à l'appel de la méthode close() de l'objet créé lors de l'ouverture :

```
<mon_fichier>close()
```

Avant de présenter un exemple, quelques recommandations :

- Il est nécessaire de s'assurer que le chemin du fichier est exact et qu'il est précisé sans adresse relative. C'est-à-dire qu'il faut préciser le chemin /home/documents/pi/.../mon_fichier.txt, et non ~/.../mon_fichier.txt. Pour que le fichier soit créé dans le même dossier que le script, il suffit de préciser un nom de fichier ; aucune adresse n'est nécessaire, car Python place le fichier dans le répertoire courant.
- Il est nécessaire de tester si l'ouverture s'est bien passée avant de faire des manipulations qui crasheraient le programme. Pour cela, il faut associer les instructions with... as... à la méthode open() de la façon suivante :

```
with open(<nom_fichier_a_ouvrir>,<mode>) as <mon_fichier> :
        <instruction 1>
        <instruction 2>
        ...
        <instruction n>
```

De cette manière, les instructions du bloc indenté ne sont effectuées que si le fichier a pu être ouvert.

Par conséquent, le code suivant :

```
with open('test_ouverture_fermeture.txt', 'w') as mon_fichier:
    print('ouverture et création fichier OK!')
    mon_fichier.close()
```

donne le résultat suivant :

```
In [3]: %run ouverture_fermeture.py
ouverture et création fichier OK!
```

Un fichier nommé test_ouverture_fermeture.txt est créé sur le disque au même emplacement que le script de test.

1.2 Écriture

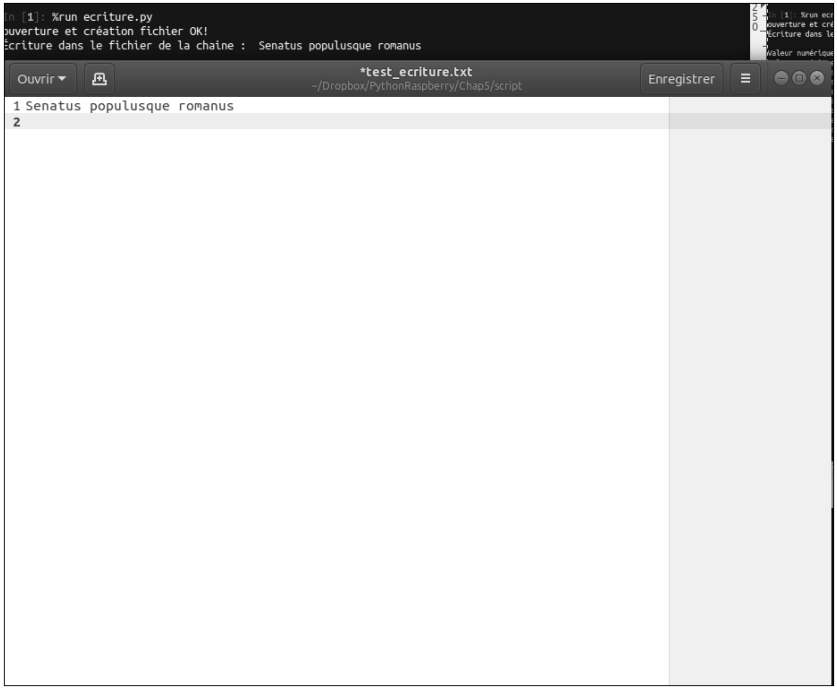
L'écriture est extrêmement simple et passe par l'appel de la méthode `write()` par l'objet créé à l'ouverture du fichier. L'exemple qui suit permet d'écrire la chaîne « Senatus populusque romanus » :

```
with open('test_ecriture.txt', 'w') as mon_fichier:
    print('ouverture et création fichier OK!')
    chaine_ecrire = "Senatus populusque romanus\n"
    mon_fichier.write(chaine_ecrire)
    print("Écriture dans le fichier de la chaîne : ", chaine_ecrire)
    mon_fichier.close()
```

L'exécution de ce script donne le résultat suivant :

```
In [5]: %run ecriture.py
ouverture et création fichier OK!
Écriture dans le fichier de la chaîne :  Senatus populusque romanus
```

La capture d'écran suivante montre le contenu dans le fichier créé test_ecriture.txt.



Le fichier étant ouvert en mode texte, par opposition au mode binaire qui sera développé ci-après, il n'est possible d'écrire que des chaînes de caractères. En effet, si des valeurs numériques sont passées en paramètre de la manière suivante :

```
mon_fichier.write(2)
```

alors, une erreur est générée par l'interpréteur :

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~/Documents/Redaction/PythonRaspberry/Chap5/script/ecriture.py in
<module>()
      4     mon_fichier.write(chaine_ecrire)
      5     print("Écriture dans le fichier de la chaîne : ", chaine_ecrire)
----> 6     mon_fichier.write(2)
      7     mon_fichier.close()

TypeError: write() argument must be str, not int
```

Pour réaliser ce genre d'écriture, il est nécessaire de convertir la valeur à l'aide de la fonction `str()` :

```
mon_fichier.write(str(2))
```

À présent, voyons comment ajouter au fichier le code nécessaire à l'écriture d'un tableau de 10 valeurs numériques générées aléatoirement. Ces valeurs seront suivies d'un renvoi à la ligne, il ne doit y avoir qu'une valeur par ligne.

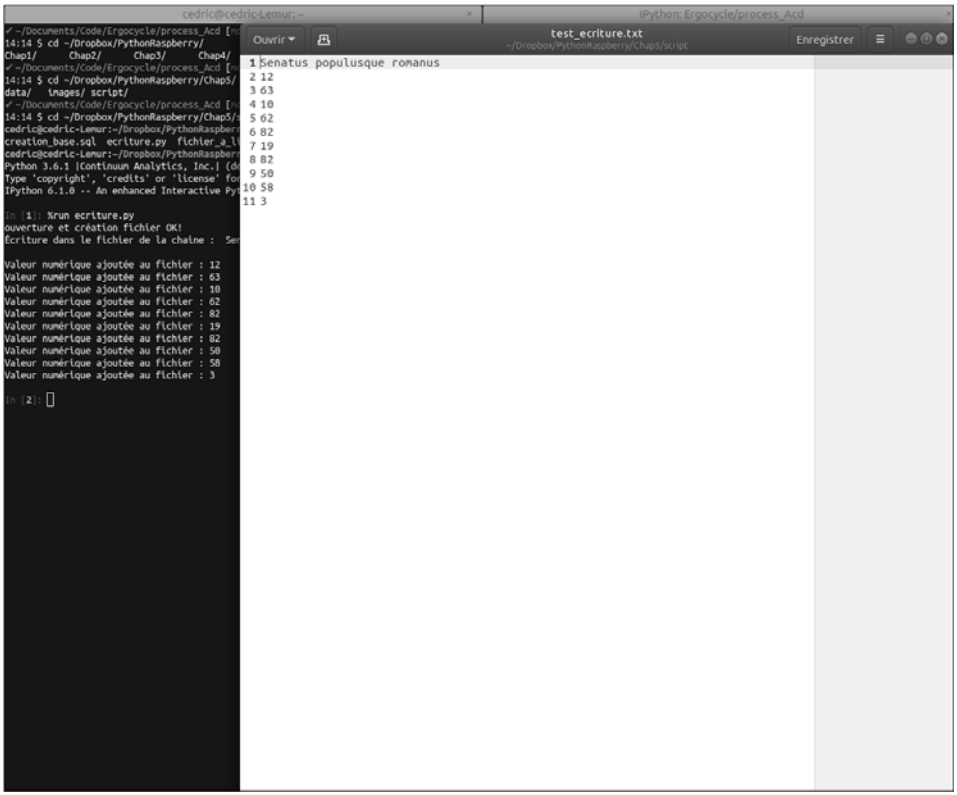
Les valeurs aléatoires sont générées à l'aide de la fonction `randint()`, contenue dans la librairie random qui doit être importée.

```
from random import randint
with open('test_ecriture.txt', 'w') as mon_fichier:
    print('ouverture et création fichier OK!')
    chaine_ecrire = "Senatus populusque romanus\n"
    mon_fichier.write(chaine_ecrire)
    print("Écriture dans le fichier de la chaîne : ", chaine_ecrire)

    for i in range(10):
        val = randint(0, 100)
        mon_fichier.write(str(val)+'\n')
        print("Valeur numérique ajoutée au fichier : " + str(val))
    mon_fichier.close()
```

Ce qui produit la sortie console suivante :

```
In [12]: %run ecriture.py
ouverture et création fichier OK!
Écriture dans le fichier de la chaîne :  Senatus populusque romanus

Valeur numérique ajoutée au fichier : 60
Valeur numérique ajoutée au fichier : 84
Valeur numérique ajoutée au fichier : 75
Valeur numérique ajoutée au fichier : 39
Valeur numérique ajoutée au fichier : 99
Valeur numérique ajoutée au fichier : 54
Valeur numérique ajoutée au fichier : 19
Valeur numérique ajoutée au fichier : 89
Valeur numérique ajoutée au fichier : 27
Valeur numérique ajoutée au fichier : 45
```

La capture d'écran suivante montre le contenu du fichier après écriture :



1.3 Lecture

La lecture fonctionne exactement sur le même principe. Il y a une condition nécessaire à l'exécution correcte d'un script de lecture : le fichier que l'on veut lire doit exister. Cela peut paraître bête, mais c'est une faute de débutant qui peut vous faire perdre pas mal de temps !

À titre d'exemple, nous allons réaliser la lecture du fichier précédemment conçu et qui sera renommé fichier_a_lire.text.

Afin d'effectuer la lecture, nous utilisons l'instruction `readline()` qui retourne le contenu d'une ligne. Pour lire la première ligne du fichier fichier_a_lire.txt, il est nécessaire d'adapter le code précédent de la manière suivante :

```
with open('fichier_a_lire.txt', 'r') as mon_fichier:
    print('ouverture fichier OK!')
    chaine_lue = mon_fichier.readline()
    print("Chaîne lue : ", chaine_lue)
    mon_fichier.close()
```

La sortie console permet de vérifier que nous avons bien la première ligne du fichier qui contient : « Senatus populusque romanus ».

```
In [15]: %run lecture.py
ouverture fichier OK!
Chaîne lue :  Senatus populusque romanus
```

Remarque

Nous utilisons, ici, la méthode `readline()` qui permet de lire ligne par ligne, et non la fonction `read()` qui permet de lire tout le fichier en une seule fois. L'utilisation de `read()` est possible, mais dans ce cas, sous-optimale, car elle nécessite un très gros traitement de la chaîne de caractères qui contient toutes les valeurs du fichier.

Afin de relire les valeurs numériques, il est nécessaire de faire une boucle pour parcourir chaque valeur puis la convertir en entier avec `int()`.

Pour réaliser cette boucle, il y a deux solutions. Si nous connaissons la quantité de données à lire, nous utilisons une boucle `for` :

```
with open('fichier_a_lire.txt', 'r') as mon_fichier:
    print('ouverture fichier OK!')
    chaine_lue = mon_fichier.readline()
    print("Chaîne lue : ", chaine_lue)

    for i in range(10):
        val_str = mon_fichier.readline()
        val_int = int(val_str)

        print("Entier lu : ", str(val_int))
    mon_fichier.close()
```