Python pour les scientifiques
Addon 1 – Les Entrées/Sorties fichiers

Cyril Desjouy

June, 2016
Updated : August 21, 2024

Un programme peut interagir avec plusieurs éléments matériels :

- L'écran : Affichage d'information
- Le clavier : Saisie de texte par l'utilisateur
- Les périphériques de stockage : Écriture ou lecture de données dans un fichier
- Le réseau : Lecture de contenu sur internet
- Et bien d'autre : souris, trackpad, touchscreen, ...

Sous Python, ces entrées/sorties se gèrent notamment avec :

- La fonction `print` pour l'affichage d'information à l'écran
- La fonction `input` pour les saisies utilisateur
- La fonction `open` pour l'écriture ou la lecture de données dans des fichiers
- Les outils fournis par le module `urllib` pour la communication réseau (non abordée ici)

On appelle communément :

- **Entrées/Sorties standard :** les interactions avec clavier et écran
- **Entrées/Sorties fichier :** les interactions avec les fichiers

File I/O

Photography by Andrew Smith

Méthode read()

```
# La fonction open() ouvre un fichier en lecture seule par défaut
In  : ff = open("myfile.txt")   # L'objet ff hérite de la méthode...
In  : ff.read()                 # ...read() qui lit tout le fichier
Out : Python is\n\t- Free and open source\n\t- Beautiful and easy to read\n
In  : ff.read()
Out :                           # Tout le fichier a été lu...
In  : ff.close()                # ...on peut donc le fermer...
In  : ff.closed                 # ...et vérifier qu'il est bien fermé...
Out : True                      # ...avec l'attribut closed !

# Affectation du contenu du fichier à une variable
In  : ff = open("myfile.txt")   # ff est du type _io.TextIOWrapper
In  : a = ff.read()
In  : ff.close()
In  : type(a)
Out : str
In  : print(a)                  # print formate automatiquement...
Out : Python is                 # ...les séquences d'échappement \t et \n
          - Free and open source
          - Beautiful and easy to read
```

Méthodes readline() et readlines()

```
# Utilisation de la méthode readline()
In  : ff = open("myfile.txt")
In  : ff.readline()                          # La méthode readline() ...
Out : 'Python is\n'                          # ...permet de lire...
In  : ff.readline()                          # ...une ligne à la fois
Out : '- Free and open source\n'
In  : ff.readline()
Out : 'Beautiful and easy to read\n'
In  : ff.readline()
Out :                                        # Tout le fichier a été lu...
In  : ff.close()                             # ...on peut donc le fermer

# Utilisation de la méthode readlines()
In  : ff = open("myfile.txt")
In  : ff.readlines()                         # La méthode readlines()...
Out : ['Python is\n',                        # ...retourne une liste...
       '- Free and open source\n',           # ...de toutes les lignes...
       '- Beautiful and easy to read\n']     # ...du fichier
In  : ff.readlines()
Out :                                        # Tout le fichier a été lu...
In  : ff.close()                             # ...on peut donc le fermer
```

Méthode write()

```
In  : ff = open("myfile.txt", "w")
In  : ff.write("This is a test")
In  : ff.close()
```

File modes

`r`	Read-only	Read access only. Default value.
`w`	Write-only	Write access only. Existing file with same name is overwritten.
`a`	Append	Add at the end of the file. Create a new file if not existing.
`r+`	Read and write	Read and write access.
`b`	Binary mode	Used for binary files.

- Tant que le fichier n'est pas fermé à l'aide de la méthode `close()`, il n'est pas assuré que celui-ci ait été modifié par vos instructions !
- Une bonne pratique est d'utiliser un *context manager* pour manipuler des fichiers (Vu dans le chapitre *Instructions composées*)

Context manager : L'instruction composée with

```
with open("myfile.txt", "w") as ff:
    ff.write("This is a test")
```

Importance du mode d'ouverture

```
In  : ff = open("myfile.txt", "r"):
In  : ff.write("This is a test")
Out : UnsupportedOperation:  not writable
In  : ff.close()

In  : ff = open("myfile.txt", "w"):
In  : ff.read()
Out : UnsupportedOperation:  not readable
In  : ff.close()
```

- Méthode de lecture utilisée sur fichier ouvert en écriture uniquement : Exception
- Méthode d'écriture utilisée sur fichier ouvert en lecture uniquement: Exception

Note les objets de type `TextIOWrapper`

Les objets crées par la fonction `open()` sont des itérateurs. Ils héritent classiquement des méthodes spéciales `__next__()` et `__iter__()`, mais celles-ci ne peuvent être utilisées que si le fichier est ouvert en mode lecture (*read*). Quand toutes les lignes du fichier ont été lues, l'itérateur est épuisé et il retourne l'exception `StopIteration` comme expliqué dans le chapitre *Instructions composées*.

La fonction `open()` retourne un objet de type `TexIOWrapper` permettant de

- Lire le contenu de fichiers **uniquement** sous forme de chaînes de caractères
- Écrire des données **uniquement** sous forme de chaînes de caractères

Pour l'écriture d'autres types d'objets, il faut procéder à la conversion :

- Par exemple, pour l'écriture d'une liste `lst` :

```
In  : f = open("myfile.txt", "w")
In  : f.write(str(lst))
In  : f.close()
```

- et pour sa lecture :

```
In  : f = open("myfile.txt", "r")
In  : lst = eval(f.read())   # eval() permet d'évaluer une expression
In  : f.close()
```

Pour la lecture et l'écriture d'objets `ndarray`, `numpy` propose des outils simples

Numpy functions for File I/O

Method	Description
`loadtxt()/savetxt()`	Read/Save an array to a text file
`load()/save()`	Read/Save an array to a `.npy` binary file
`load()/savez()`	Save several arrays into an `.npz` uncompressed archive
`load()/savez_compressed()`	Save several arrays into an `.npz` compressed archive

Syntaxe : savetxt/loadtxt et save/load

```
In  : import numpy as np
In  : arr1 = np.array([1, 2, 3])
In  : arr2 = np.array([4, 5, 6])

# Sauvegarde...
In  : np.savetxt("myfile.txt", arr1)   # np.save('myfile.npy', arr1)

# Chargement...
In  : np.loadtxt("myfile.txt")         # np.load('myfile.npy')
Out : array([1, 2, 3])
```

Syntaxe : les archives savez/savez_compressed

```
# Sauvegarde...
In  : np.savez("myfile", a1=arr1, a2=arr2)

# Chargement...
In  : data = np.load("myfile.npz")
In  : data.keys()          # Une archive contient plusieurs objets
Out : [a1, a2]
In  : print(data[a2])      # dont l'accès se fait par clé
Out : array([4, 5, 6])
```

Note sur l'usage du disque et sur les temps de chargement/sauvegarde :

Saving/loading time for an array of 10^6 random elements		
Function	**Execution time**	**Disk usage**
`loadtxt()/savetxt()`	900ms/700ms	24 Mo
`load()/save()`	2ms/25ms	7.7 Mo
`load()/savez()`	45μs/45ms	7.7 Mo
`load()/savez_compressed()`	45μs/325ms	7.2 Mo (7.4 Ko for np.zeros)

Note sur les outils fournis par la bibliothèque `scipy`:

- Gestion des fichiers Matlab™(`scipy.io`):
  - `loadmat()` : Lecture de fichiers MATLAB™
  - `savemat()` : Écriture de fichiers MATLAB™
  - `whosmat()` : Liste les variables dans un fichier MATLAB™
- Gestion des fichiers wave (`scipy.io.wavfile`):
  - `read(filename)` : Ouvre un fichier WAV (retourne `rate` et `data`)
  - `write(filename, rate, data)` : Écriture d'une `ndarray` dans un fichier WAV.

En cours de rédaction

Lorsqu'il s'agit de gérer des objets `ndarray` :

Fonctions fournies par `numpy`

- `load()/loadtxt()`
- `save()/savetxt()/savez()/savez_compressed()`

Sinon :

Fonction built-in `open()`

- Conversion de l'objet à sauvegarder en chaîne de caractères
- Méthodes et attributs hérités par les objets `TextIOWrapper`
  - Pour la lecture : `read()/readline()/readlines()`[1]
  - Pour l'écriture : `write()`
  - Pour la fermeture : `close()`

[1] Ces méthodes retournent des `str` ou des listes de `str` uniquement