

# Chapitre 9 : Entrées/Sorties et Bibliothèques

Yves Guidet pour IPSSI

V1.4.7 March 24, 2017

Python est livré avec une bibliothèque standard de modules très divers.

Il existe également un grand nombre de modules à installer soi-même. On verra comment installer ces modules, sous Unix et MSWindows.

- ▶ ouverture et lecture

```
>>> x = open ('/etc/passwd', 'r')
>>> x.readline()
'root:x:0:0:root:/root:/bin/bash\n'
x.close ()
```

- ▶ la méthode *readline* retourne la chaîne vide "" si il n'y a plus rien à lire.

- ▶ lecture de tout le fichier

```
>>> x = open ('/etc/passwd', 'r')  
>>> x.readlines()[17]  
'vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nolog
```

J'adore cette compacité, je n'empêche personne de passer par une variable intermédiaire mais c'est si beau ...

Ayant lu une ligne, on peut avoir besoin de la « nettoyer » de sa fin de ligne, c'est la méthode *rstrip* qui s'y emploiera :

```
>>> 'test string\n'.rstrip()  
'test string'
```

En savoir plus ? Ce lien :

<http://docs.python.org/2/library/stdtypes.html> et cet autre  
<http://stackoverflow.com/questions/275018/how-can-i-remove-chomp-a-newline-in-python> ou encore  
<http://tinyurl.com/ykj2c8u>.

La méthode *write* permet d'écrire une *str* dans un fichier.

```
>>> x = open ('tralala', 'w')
>>> x.write('Et Python dans dans tout ca ?')
>>> x.close()
>>> from os import system
>>> system("cat tralala")
Et Python dans dans tout ca ?0
```

Le "0" n'est pas un artefact ...  
Comment faire sous MSindows ?

## Écrire dans un fichier (suite)

Cette méthode *write* n'est pas toujours pratique, on aimerait utiliser *print*. Considérons ces quelques lignes de code :

```
out = open(csv, "w")

...

print >> sys.stderr, len(items), " items"
print >> out, repr(items[0])
for item in items[1:]:
    print >> out, item.dataString()
```

On voit que » permet de le faire, ainsi que pour *stderr*.

# Et en Py3k ?

En ce cas il faudra remplacer :

```
print >>x, "coucou"
```

Par :

```
print("coucou", file=x)
```



- ▶ tubes Unix et DOS : le principe

```
>>> os.popen ("ls -A", "r").readlines() # ou dir /all
```

- Ecrire un script conjuguant un verbe dans un fichier dont le nom sera de la forme infinitif.txt

On arrive là aux outils de l'exploitant.

- ▶ `os.getcwd`, `os.chdir()`
- ▶ `os.path`
  - ▶ `basename`, `dirname`, `exists`, `getmtime`, `getsize`, `isdir`, `isfile`,  
`ismount`, `join` (portable)
- ▶ `shutil`
  - ▶ `copy`, `copy2` (`cp -p`), `copytree`
  - ▶ `move`
  - ▶ `rmtree`
- ▶ `os.walk`
  - ▶ voir l'exemple dans le `help` (attention au type du retour de la fonction)

# Passage d'argument : le module sys

- Passer des arguments à un script (noms de fichier ou autre)

```
[yves@localhost j3]$ cat scr.py
```

```
from sys import argv
```

```
print argv
```

```
[yves@localhost j3]$ python scr.py Bonjour Python
```

```
['scr.py', 'Bonjour', 'Python']
```

« System-specific parameters and functions »

Tout d'abord, `sys.argv` :

- ▶ `argv[0]` : le nom du script
- ▶ ensuite les arguments.

Ensuite `sys.exit([arg])`

```
exit(...)  
    exit([status])
```

Exit the interpreter by raising `SystemExit(status)`.

If the status is omitted or `None`, it defaults to zero (i.e., success).

If the status is numeric, it will be used as the system exit status.

If it is another kind of object, it will be printed and the system exit status will be one (i.e., failure).

En savoir plus sur `sys` ? Voir <http://docs.python.org/2/library/sys.html>.

# argv : un exemple

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

from sys import argv, exit

if len(argv) == 1:
    exit(argv[0] + " : pas d'argument")

print "je suis", argv[0], "et je suis appelé avec", len(argv) - 1, "arguments"
for a in argv[1:]:
    print a,
```

## argv : un exemple (suite)

Ce qui donne :

```
yves@r5:argv$ ./constantin.py
./constantin.py : pas d'argument
yves@r5:argv$ ./constantin.py eins zwei drei
je suis ./constantin.py et je suis appelé avec 3 arguments eins zwei drei
```

Souvent ces arguments seront des noms de fichiers, on considérera le module *fileinput*.

Lorsque les arguments passés au script sont des fichiers, on aura tout intérêt à utiliser ce module.

On utilisera généralement le schéma suivant :

```
import fileinput
for line in fileinput.input():
    process(line)
```

Si la liste est vide, on parcourra *sys.stdin* c'est à dire l'entrée standard.



Le « process » du transparent précédent consiste souvent en une *modification* du fichier. L'« inplace editing » permettra de le faire, comme dans l'exemple suivant :

```
import fileinput
for line in fileinput.FileInput(inplace = 1, backup = '.BAK'):
    line = line.replace("python", "Py3k")
    print line
```

Si "tyty" et "toto" sont passés en argument au script, ils seront sauvegardés en "tyty.BAK" et "toto.BAK", alors qu'en fin de traitement "tyty" et "toto" contiendront les substitutions.

Pour en savoir plus on ira voir ceci

<http://docs.python.org/2/library/fileinput.html>, et pour Py3k ce lien <http://docs.python.org/3/library/fileinput.html>.

## glob : pour utiliser des métacaractères

La fonction *glob* retourne une liste de fichiers.

```
>>> import glob
>>> glob.glob('*.*')
['glob.t2t', 'glob.xhtml']
```

Moins gourmande en mémoire *iglob* retourne un *itérateur* (voir <http://tinyurl.com/ot5cvsv>).

En savoir plus : voir <http://docs.python.org/3/library/glob.html>.

# Substitution de commandes à la Unix

Ce mécanisme est bien connu des Unixiens : elle consiste à récupérer la sortie d'une commande.

En Python on utilisera le module *commands*.

```
>>> import commands  
>>> help(commands.getoutput)
```

```
getoutput(cmd)
```

Return output (stdout or stderr) of executing cmd in a s

Notons qu'on peut également (et simultanément) recevoir le *status*.

# Passage d'argument : exercices

- ▶ passer en argument des infinitifs, des températures, des rayons  
...
- ▶ inplace edit : écrire un script insérant un « shebang » aux fichiers passés en paramètres

- ▶ epoch *cf* <http://docs.python.org/library/time.html>

```
>>> import time
>>> time.time() # float (secondes)
1201068455.7889349
>>> time.localtime()
(2008, 1, 23, 7, 7, 44, 2, 23, 0)
>>> time.asctime()
'Wed Jan 23 07:07:54 2008'
```

L'epoch n'est autre que la date de naissance « officielle » d'Unix, à savoir le 1er janvier 1970.

# Dates : en savoir plus

Chercher *calendaire* dans ENI.

Pour ce qui est du parallélisme on ira voir ENI, ainsi que (pour les *threads*),

<http://yvesguidet.no-ip.biz/boa/2014/déc/parallelism.xhtml>

Pour les *sockets*, voir

<http://yvesguidet.no-ip.biz/boa/2014/juin/sockets4jules.xhtml>.

Bien sûr Python est très fort en réseau, que ce soit au niveau des sockets ou des protocoles, il sait tout faire, entre autres du CGI.

Et même au niveau du paquet, il y a l'outil Scapy (<http://fr.wikipedia.org/wiki/Scapy>).

Rappelons que Saint.py utilise urllib2, décrite dans :

<http://docs.python.org/library/internet>

Enfin ceux qui s'intéressent à SMTP regarderont

<http://tinyurl.com/5rjkg3w> et pour *ssh*, *ftp* (et *sftp*) le bouquin ENI.

On va parler d'openstack, mais d'abord quelques mots sur les modules non standard.



# Modules Python : installation

Certains modules ne sont pas dans la distribution standard :

```
yves@bella:installModule$ python
Python 2.7.4 (default, Apr 19 2013, 18:28:01)
[GCC 4.7.3] on linux2
Type "help", "copyright", "credits" or "license" for more info
>>> import matplotlib
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named matplotlib
```

Source : <https://docs.python.org/2/install/>

# trivial installation

- ▶ rpm, deb
- ▶ pas de *setup*

Sinon on peut utiliser pip <http://bit.ly/1pKiN0y>.

## un exemple : matplotlib

```
yves@bella:installModule$ apt-cache search matplotlib
python-matplotlib - Python based plotting system in a style
python-matplotlib-data - Python based plotting system (data
...
```

```
yves@zette14:tmp> ll matplotlib-1.3.1.tar.gz
-rw-r--r-- 1 yves users 42163182 avril 28 12:41 matplotlib-1.3.1.tar.gz
```

Util. 7Zip sous windows, ... ou Cygwin ;)

*Pip* fonctionne sous MSWindows.

Pour en savoir plus, chercher *eggs* dans le support.

Attention à l'utilisation derrière un *proxy*.

Consulter aussi :

- ▶ <http://continuum.io/downloads>
- ▶ <http://ipython.org/install.html> (pour IPython)
- ▶ <https://www.enthought.com/products/canopy/> (pour SciPy)
- ▶ <http://docs.continuum.io/anaconda/>

Voir wikipédia (<http://fr.wikipedia.org/wiki/OpenStack>).

OpenStack est un projet informatique de service d'infrastructure (Infrastructure as a Service (IaaS)) du domaine du cloud computing, mené par la Fondation OpenStack.

Voir aussi *pypi*

(<https://pypi.python.org/pypi/python-openstackclient>). On voit que le package est en Python 2.x.

Pas encore en Py3k

(<https://www.python.org/download/releases/3.0/>), donc.

Voir ENI. Pour s'entraîner on pourra requêter *cff.sql*.

Python ne possède pas (à ma connaissance) de module abstrait (au sens de Java) permettant de se connecter à un SGBD quelconque. En revanche, étant à la pointe du progrès dans le monde du Web (entre autres) on trouvera plusieurs *ORM*.

Il s'agit de sauver des objets dans un SGBDR.

**Rédaction réservée.**

En attendant, voir ENI.



Citons :

- ▶ PIL
- ▶ le module ndimage de scipy, (orienté mathématique)
- ▶ opencv qui a des binding python.
- ▶ Finalement il y a ITK, avec wrapITK.

Tout cela est donné dans l'ordre croissant de difficulté d'utilisation.  
Voir aussi pygame

# Compléments : interfaces graphiques

TkInter est le module standard pour les interfaces graphiques.

PyGTK permet d'utiliser la bibliothèque GTK+. On pourra consulter :

<http://daniel.coquette.free.fr/dotclear/index.php/post/2006/12/11/Creer-des-interfaces-graphique-avec-PyGTK-et-Glade>

TkInter n'est autre qu'un wrapper du Tk de Tcl/Tk.

Pour PyQt (d'origine Nokia, bibliothèque rivale de GTK) voir :

<http://fr.wikibooks.org/wiki/PyQt>

On va vous en dire un peu plus.

On va considérer :

- ▶ TkInter
- ▶ PyGTK
- ▶ PyQt
- ▶ wxPython

*TkInter* est le module **standard** pour les interfaces graphiques. Un des plus anciens outils libres se nomme *Tcl/Tk* (prononcer Tickle-Tikay) ; il se compose :

- ▶ d'un shell *Tcl* (Tool Command Language)
- ▶ d'une bibliothèque *Tk* (ToolKit).

*TkInter* utilise *Tk*, en fait *TkInter* n'est autre qu'un wrapper du *Tk* de *Tcl/Tk*.

Gtk+ (Gimp ToolKit), est une bibliothèque écrite en C pour Gimp, puis Gnome.

PyGTK permet d'utiliser la bibliothèque GTK+. On pourra consulter :

<http://daniel.coquette.free.fr/dotclear/index.php/post/2006/12/11/Creer-des-interfaces-graphique-avec-PyGTK-et-Glade>

Noter que PygObject (voir <http://bit.ly/1k4Ga1Z>) remplace PyGTK, comme expliqué ici :

<http://www.developpez.net/forums/d1063879/general-developpement/programmation-systeme/linux/gtk/pygtk/pygtk-passage-pygobject-gtk-3-0-a/>. (ou <http://bit.ly/1IE8WCM>).

*Glade* est un générateur d'interfaces (*GUI Builder*) et *pyGlade* sa traduction en Python. Il génère des ".ui" au format XML. Voir la page d'accueil sur sourceforge <http://pyglade.sourceforge.net/>

De même que *Gnome* était une « riposte » 100% libre à *KDE*, de même *GTK+* a été conçue comme l'anti-Qt.

*Qt* (prononcer *cute*, i.e. « mignon » et non *cutee*) est en effet une bibliothèque pas entièrement libre, développée par la société norvégienne *Troll tech*, rachetée par *Nokia* qui s'en sert pour les applis pour mobile. Pour PyQt voir :

<http://fr.wikibooks.org/wiki/PyQt>

*Qt* est toujours distribué avec une licence double.

Datant de 1992 mais toujours actif, le projet *wxWidgets* (à l'origine *wxWindows*) a pour objet le développement d'applis graphiques s'exécutant tant sous X11 que sous Windows. Voir Wikipédia pour plus d'info sur son « wrapper » *wxPython*.

Il s'agit de sauver des objets dans une base de données relationnelle, comme *Hibernate* en *Java*. Cela implique une sérialisation de ces objets.

Le module *pickle* est décrit dans le bouquin ENI.



L'acronyme anglais CSV (pour « Comma-Separated Values ») désigne des fichiers texte contenant des données en colonne, généralement séparées par des virgules, et utilisés par des tableurs, parfois aussi des SGBD.

# Création d'un fichier csv

On va utiliser un objet *writer*, la méthode *writerow* écrivant une ligne dans le fichier.

```
import csv
f = csv.writer(open("unfichier.csv", "wb"))
f.writerow(["Nom", "Adresse", "Courriel"])
f.writerow(["Yves", "30 av. de la porte de Choisy", "yves.guidet@gmail.com"])
```

Noter l'ouverture en mode binaire, ignorée sous Unix et nécessaire sous MSWindows. Après l'exécution du script, on constate l'arrivée dans le répertoire d'un beau fichier csv.

```
yves@r5:slides$ cat unfichier.csv
Nom,Adresse,Courriel
Yves,30 av. de la porte de Choisy,yves.guidet@gmail.com
```

Cette fois on va créer un objet *reader*, et utiliser la boucle *for* pour parcourir les lignes du fichier.

```
import csv

cr = csv.reader(open("unfichier.csv","rb"))

for row in cr:
    print row,
    print '\t', len(row), " colonnes"
```

Souvent, en France du moins, on utilise on préfère le point-virgule à la virgule, en raison de la représentation des flottants.

D'autres fois, on prend la tabulation comme délimiteur.

Bref, la notion de fichier `csv` étant imprécise, Python a introduit la notion de *dialecte*.

```
>>> import csv
>>> csv.list_dialects()
['excel-tab', 'excel']
```

L'utilisateur pourra définir ses propres dialectes, qui seront des classes dérivées de `csv.Dialect`, comme décrit dans ce lien :

<http://tinyurl.com/mq6x2ua>

# Utilisation de dictionnaires

Il est possible, du moins si la première ligne contient les noms des colonnes, de récupérer les données d'un fichier sous forme de dictionnaire. Ainsi, en utilisant un objet *csv.DictReader* à la place de notre *csv.reader* :

```
import csv
```

```
l = csv.DictReader(open("unfichier.csv", "rb"))
```

```
for row in l:  
    print row
```

donnera à l'exécution :

```
yves@r5:slides$ ./lireDict.py
```

```
{'Nom': 'Yves', 'Adresse': '30 av. de la porte de Choisy', 'Courriel': 'yves.gu
```

# Exceptions

Comme toujours on prendra garde aux catastrophes qui peuvent se produire au cours des Entrées/Sorties , ou dans le traitement :

```
import csv
import sys

f = open(sys.argv[1], 'rt')
try:
    reader = csv.reader(f)
    for row in reader:
        print row
finally:
    f.close()
```

On pourra aussi utiliser *with* :

```
import csv
with open('eggs.csv', 'wb') as csvfile:
    spamwriter = csv.writer(csvfile, delimiter=' ')
# on continue
```

La commande *with* est décrite dans ENI.

Grâce à un "*dir*" obtenir la liste des fichiers d'un répertoire, les ranger dans un fichier *csv*.

- ▶ Package, collection, structures complexes (double Q ?)
- ▶ Tests unitaires
- ▶ E/S fichier, module pickle
- ▶ Vue sur l'ensemble des librairies python, eco-system python
- ▶ Sites internet python à connaître
- ▶ Py Lint pour analyse du code
- ▶ Utilisation de bibliothèque en C dans un script Python, package ctypes



# Package, collection, structures complexes (double queues)

Package, collection : OK

Voir <http://en.wikipedia.org/wiki/Deque>

On ira d'abord lire les **bonnes pratiques** dans ENI (chercher *carre*) et **en exercice** on écrira une fonction de test pour au moins un des exercices déjà vus.

Voir ENI (chercher pyLint) et <https://pypi.python.org/pypi/pylint>

# Vue sur l'ensemble des bibliothèques python, écosystème Python

Voir le bouquin ENI.

# Utilisation de bibliothèque en C dans un script Python , package ctypes

Pour interfacer du C via swig on lira : <http://matthieu-brucher.developpez.com/tutoriels/python/swig-numpy/> (l'ancienne version du bouquin ENI).

Pour ctypes on ira voir

<http://docs.python.org/2/library/ctypes.html>

NumPy :

<http://yvesguidet.no-ip.biz/boa/2014/juin/NumPy.shtml>

SciPy : <http://yvesguidet.no-ip.biz/boa/2014/juin/SciPy.shtml>

Voir aussi <http://stackoverflow.com/questions/874461/read-mat-files-in-python> pour l'import de fichiers

Matlab.

Matplotlib : *cf infra*.

Voir aussi FFT et SymPy (plus bas).

# Qu'est-ce que Matplotlib?

On jette un œil à wikipédia

<http://en.wikipedia.org/wiki/Matplotlib>. On y trouve :

- ▶ matplotlib est une bibliothèque de *plotting* pour Python et NumPy
- ▶ API OO for englober les figures dans des applications à base de wxPython, Qt, or GTK+.
- ▶ API procédurale "pylab" en faisant un clone de MATLAB.
- ▶ utilisé aussi par SciPy

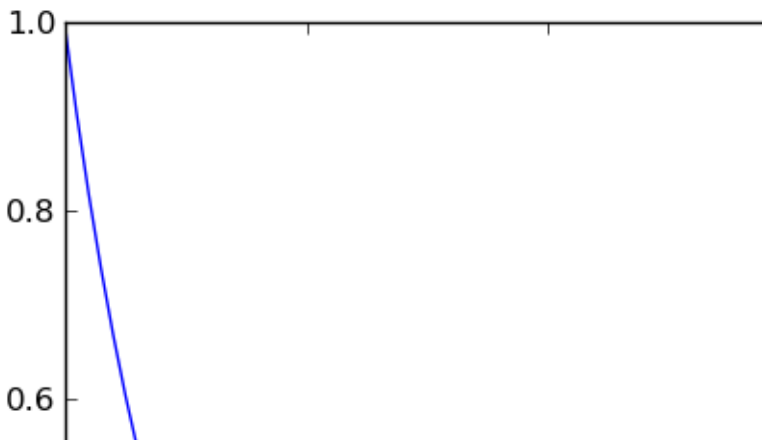
On y trouve aussi un petit exemple "basique" :

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> a = np.linspace(0,10,100)
>>> b = np.exp(-a)
>>> plt.plot(a,b)
>>> plt.show()
```

Si on lance l'exemple ci-dessus, on voit que *matplotlib* est inconnu.  
Le paquet *python-matplotlib* existe en *.deb* comme en *.rpm*.

# L'exemple ci-dessus à nouveau

Une fois *matplotlib* installé, on recommence et on obtient une jolie figure :





# Des tutos pour aller plus loin ?

Et si on se trouvait un *tuto* ? Il y a pléthore.

Voyons déjà celui du LORIA

<http://www.loria.fr/~rougier/teaching/matplotlib/>, notons ce qu'il dit sur *IPython*, et essayons l'exemple.

# Une autre courbe

On remet ça avec  $\text{sqrt}(4 - x^2)$ .

```
1 #! /usr/bin/env python
2
3 import matplotlib.pyplot as plt
4 import numpy as np
5 a = np.linspace(-2,2)
6 b = np.sqrt(4 - a*2) # mardi 20 mai 2014, 17:47:06 (U
7 plt.plot(a,b)
8 plt.show()
```

On jette un œil à ce *linspace* :

```
linspace(start, stop, num=50, endpoint=True, retstep=False)
    Return evenly spaced numbers over a specified interval.
```

```
Returns 'num' evenly spaced samples, calculated over the
interval ['start', 'stop' ].
```

Dans ce tuto

<http://shreevatsa.wordpress.com/2010/03/07/matplotlib-tutorial/>,  
on trouve un bel exemple d'utilisation des *comprehensions* :

```
1 #! /usr/bin/env python
2
3 import matplotlib.pyplot as plot
4 import math
5
6 xs = [0.01*x for x in range(1000)] #That's 0 to 10 in steps of 0.01
7 ys = [math.sin(x) for x in xs]
8 plot.plot(xs, ys)
9 plot.savefig("sin.png")
10
11
```

Et cette fois on sauve la figure sans l'afficher.

Il s'agit de logiciel de **calcul formel** sur des symboles), par opposition au calcul numérique où on ne manipule que des nombres. Pour l'install voir via git <http://bit.ly/1pKgaMd> ou plus généralement : ici <http://bit.ly/1mhLbRv>

On pourra lire ceci

<http://docs.sympy.org/latest/tutorial/index.html>, et aussi (tuto + exemples) cela

<http://dakarlug.org/pat/scientifique/barcamp/symbolic.html>.

Décorateurs (*design pattern*) voir <http://bit.ly/1vlkgSW>