Programmation avec Python - Automne 2016
Bruno Martin & Jean-Paul Roy
L1-Sciences
http://deptinfo.unice.fr/~roy

Cours n°9

Les fichiers

Utilité des fichiers

• Le mot **fichier** provient du terme de **fiche** :

« feuille de carton sur laquelle on écrit soit les titres des ouvrages que l'on veut cataloguer, soit les renseignements sur une personne ou un fait que l'on veut garder et retrouver facilement »

• Le **fichier** désignait le recueil des fiches ou le meuble qui contient les fiches.

• Les **systèmes d'exploitation** (Linux, MacOS, Windows) permettent de stocker de grandes quantités d'informations, de les rechercher et de les classer sur disques durs (situés où ?) dans des fichiers un peu comme les fiches cartonnées.

• Système d'exploitation : ***Operating system*** (**OS**). La référence pour un informaticien est **UNIX** (Linux, MacOS).

Il y a deux systèmes de fichiers possibles

• Il y a essentiellement deux déclinaisons de systèmes de fichiers (*filesystems*) : **Unix** (Linux, MacOS) et **Windows**.

• Avec quelques légères différences entre **Linux** (Unix System V) et **MacOS** (Unix BSD).

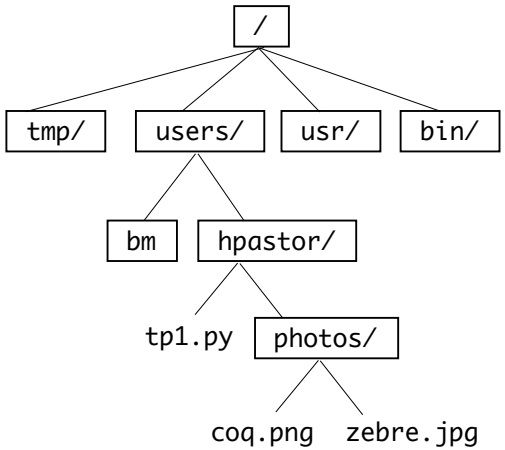
• Les fichiers (*files*) sont regroupés dans une **arborescence** (*tree*) ayant une ou plusieurs **racines** (*roots*), dont les noeuds sont les **répertoires** (*directories*, *folders*) et les feuilles sont les **fichiers** (*files*).

• Un répertoire contient des répertoires et des fichiers.

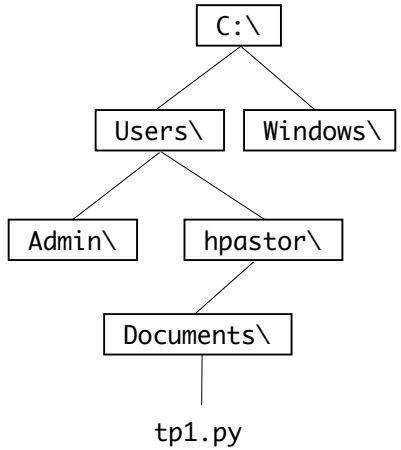
• Sur **Unix**, une seule racine nommée `/`

• Sur **Windows**, plusieurs racines nommées `A:\`, `B:\`, `C:\`, etc. En général `C:\` représente le disque principal.

UNIX (Linux)



WINDOWS



• Un répertoire ou un fichier est relié à la racine par un **chemin** unique.

```
/users/hpastor/                      c:\Users\hpastor\
/users/hpastor/photos/coq.png        c:\Users\hpastor\Documents\tp1.py
```

Chemins relatifs et absolus

• Les **chemins absolus** spécifient l'emplacement exact d'un répertoire ou fichier à l'intérieur d'une arborescence, à partir de la racine.

(Linux) `/users/hpastor/photos/coq.png`

(MacOS) `/Volumes/macleUSB/Python/tp1.py`

(Windows) `c:\Users\hpastor\Documents\tp1.py`

• Les **chemins relatifs** spécifient l'emplacement d'un répertoire ou fichier à partir d'un certain répertoire de l'arborescence (implicite, non indiqué dans le chemin lui-même).

Ex : `photos/coq.png` *est un chemin relatif à* `/users/hpastor/`

• Le répertoire père se note `..`

Ex : `../../bm` *si je suis dans le répertoire* `photos`

Le répertoire courant et le module os

• Il est important de savoir dans quel répertoire on est en train de travailler, afin d'accéder aux fichiers par des chemins relatifs. Ce répertoire de travail est le **répertoire courant** (*current directory*).

• Le module os de Python permet de manipuler le système de fichiers sans sortir de Python (au toplevel ou dans une fonction).

```
import os
```

• Je peux demander quel est le répertoire courant si je suis perdu :

```
>>> os.getcwd()
'/Users/jpr/Documents'
```

Get Current Working Directory

• Je peux changer de répertoire courant et me déplacer (de façon relative ou absolue) vers un autre répertoire de l'arborescence :

```
>>> os.chdir('../Desktop/Exemples/')
```

Change Directory

Chemins et chaînes de caractères

• Un chemin peut être codé en Python par une chaîne. ***Problème avec Windows qui ne suit pas les conventions UNIX*** : les \ doivent être doublés (puisque \ est un caractère d'échappement dans une chaîne).

(Linux/Mac) `'/users/hpastor/photos/coq.png'`

(Windows) `'c:\\Users\\hpastor\\Documents\\tp1.py'`

• **Un bon logiciel doit fonctionner sur tous les OS**. On peut demander en Python sur quel système on travaille :

```
>>> os.name
'posix'
```

Linux/Mac

```
>>> os.name
'nt'
```

Windows

• On peut construire un chemin de manière portable :

```
>>> os.path.join('hpastor','photos','coq.png')
'hpastor/photos/coq.png'
```

Linux/Mac

```
>>> os.path.join('hpastor','Documents','tp1.py')
'hpastor\\Documents\\tp1.py'
```

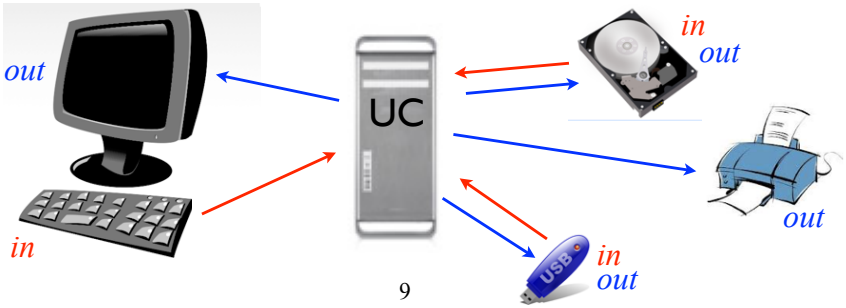
Windows

Quelques fonctions utiles du module os

`os.getcwd()`	*le répertoire courant*
`os.chdir(path)`	*changer de répertoire courant*
`os.listdir(path='.')`	*liste des fichiers et répertoires*
`os.path.join(path1,path2,...)`	*construction portable d'un chemin*
`os.remove(path)`	*suppression d'un fichier*
`os.path.isfile(path)`	*test d'existence d'un fichier*
`os.path.isdir(path)`	*test d'existence d'un répertoire*
`os.path.split(path)`	*pour extraire le fichier d'un chemin*
`os.path.getsize(path)`	*la taille d'un fichier*

Des fichiers en entrée et en sortie

- On peut vouloir **écrire** des données dans un fichier sur le disque.
- Pour ensuite **lire** ce fichier afin d'en extraire des informations, tout ou partie des données.
- Les deux activités de base sur les fichiers sont donc :
  - LECTURE (*in* : fichier en entrée)
  - ECRITURE (*out* : fichier en sortie)



Ouverture d'un fichier en écriture

- Je souhaite créer un fichier `test.txt` contenant des résultats.
- Nous ne travaillerons dans ce cours qu'avec des **fichiers de texte**. Dans un tel fichier, nous ne pourrons donc déposer **que des chaînes de caractères** ! Pour déposer `-234`, nous déposerons `'-234'`.
- Commençons par créer un **nouveau fichier** `test.txt` **en écriture** (*write*) avec la fonction `open(`*filename*`,'w',`*encoding*`)` où *filename* est une chaîne contenant le chemin (absolu ou relatif) menant au fichier. Si un ancien fichier de ce nom existe déjà, il est remplacé.

```
f_out = open('test.txt', 'w', encoding = 'utf-8')
```

```
>>> f_out
<_io.TextIOWrapper name='test.txt' mode='w' encoding='utf-8'>
```

- La valeur de `f_out`, résultat de `open`, est un objet *descripteur de fichier*. L'encodage par défaut est US-ASCII.

- Une fois le fichier ouvert, il est prêt à recevoir des données.

```
for i in range(1,5) :
    f_out.write('5 * {} = {}\n'.format(i,5*i))
```

- Vous avez noté la **chaîne formatée** `'xx{}xx{}x'.format(e1,e2)`
- A la fin du traitement, n'oubliez pas de fermer le fichier (*close*) !

```
f_out.close()
```

- Le tout se fait dans une fonction :

```
def creer_fichier(f) :
    f_out = open(f,'w',encoding='utf-8')
    for i in range(1,5) :
        f_out.write('5 * {} = {}\n'.format(i,5*i))
    f_out.close()
```

Ouverture

Traitement

Fermeture

- Si le traitement est long, on peut le déléguer à une fonction spécialisée à laquelle on passera le descripteur de fichier `f_out`.

- Il ne reste qu'à invoquer la fonction pour créer un fichier dans le répertoire courant :

```
>>> creer_fichier('test.txt')      # aucun résultat
>>> os.listdir()
['.DS_Store', 'code9.py', 'Cours9.key', 'test.txt']
```

- Maintenant le fichier `test.txt` existe sur le disque. Il faudra l'ouvrir avec un **éditeur de texte compatible Unicode** (ex: IDLE).

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
```

test.txt

- La méthode `write` permet d'écrire une ligne (terminée par un \n).
- La méthode `writelines` permet d'écrire une liste de lignes :

```
liste = ['5 * {} = {}\n'.format(i,5*i) for i in range(1,5)]
f_out.writelines(liste)
```

Ecrire à la fin d'un fichier

• Il peut être intéressant d'**ajouter des lignes** à un fichier. Il faut alors l'ouvrir en écriture en mode `'a'` et non `'w'`.

```
f_out = open('test.txt', 'a', encoding = 'utf-8')
```

• Je vais rajouter deux lignes à la fin de mon fichier :

```
def augmenter_fichier(f) :
    f_out = open(f,'a',encoding='utf-8')
    for i in range(5,7) :
        f_out.write('5 * {} = {}\n'.format(i,5*i))
    f_out.close()
```

NB : Il n'est pas possible de supprimer des lignes dans un fichier. Il faut créer un nouveau fichier et détruire l'ancien !

Ouverture d'un fichier en **lecture**

• Problème inverse : comment lire le fichier texte `test.txt` ?

• Je dois connaître son encodage ! Or je sais qu'il est en UTF-8.

```
f_in = open('test.txt','r',encoding = 'utf-8')
```

• Je peux lire d'un seul coup la **totalité du fichier** dans une seule chaîne de caractères, avec la méthode `read()`.

```
>>> texte = f_in.read()            # une seule lecture !
>>> f_in.close()
>>> texte
'5 * 1 = 5\n5 * 2 = 10\n5 * 3 = 15\n5 * 4 = 20\n'
>>> print(texte)
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
```

*N.B. La **fin de ligne** est codée différemment suivant les systèmes. Sur MacOS-X, c'est '\r'. Sur Linux, c'est '\n'. Sur Windows, c'est '\r\n'. Python s'adapte au système utilisé.*

• Ou bien je lis le fichier **ligne à ligne** avec la méthode `readline()`, jusqu'à obtention d'une ligne vide.

```
def afficher_fichier(f) :              # ligne à ligne
    f_in = open(f,'r',encoding='utf-8')
    i = 1
    while True :
        ligne = f_in.readline()
        if ligne == '' : break     # fin du fichier !
        print(i,':\t',ligne,sep='',end='')
        i = i + 1
    f_in.close()
```

```
>>> afficher_fichier('test.txt')
1:      5 * 1 = 5
2:      5 * 2 = 10
3:      5 * 3 = 15
4:      5 * 4 = 20
```


• Il est parfois possible d'éviter `readline()`, car un descripteur de fichier est un objet itérable :

```
def nb_lignes(f) :
    f_in = open(f,'r',encoding='utf-8')
    cpt = 0
    for ligne in f_in :
        cpt = cpt + 1
    f_in.close()
    return cpt
```

```
>>> nb_lignes('test.txt')
6
```

```
def nb_lignes(f) :
    f_in = open(f,'r',encoding='utf-8')
    cpt = sum(1 for ligne in f_in)
    f_in.close()
    return cpt
```

• Enfin, la méthode `readlines()` permet d'obtenir en une seule instruction la **liste de toutes les lignes** d'un fichier texte.

```
cpt = len(f_in.readlines())
```

Travail en lecture et écriture

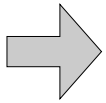
• Souvent, on prend un fichier en entrée et on produit un autre fichier en sortie (transformation).

• Exemple. J'ai un fichier dont chaque ligne contient des nombres. Je veux remplacer chaque ligne par la somme de ces nombres.

fichier en entrée

```
83 21 10 34 98
11 34 18 69 76
92 88 28 54 62
17 33 45 13 82
18 11 88 67 85
88 51 89 66 20
98 49 72 29 59
33 78 86 42 62
```

alea.txt



en sortie

```
246
208
324
190
269
314
307
301
```

somme.txt


```
def creer_alea(f) :
    # entre n = 4 et 10 lignes, et 5 entiers par ligne
    n = randint(4,10)
    f_out = open(f,'w',encoding='utf-8')
    for i in range(n) :                  # je produis n lignes
        for i in range(5) :
            f_out.write('{} '.format(randint(10,100)))
        f_out.write('\n')
    f_out.close()
```

```
def transformer(f1,f2) :
    f_in = open(f1,'r',encoding='utf-8')
    f_out = open(f2,'w',encoding='utf-8')
    for ligne in f_in :
        somme = sum(map(int,ligne.split()))
        f_out.write('{}\n'.format(somme))
    f_out.close()
    f_in.close()
```

Woaw !

```
>>> creer_alea('alea.txt')
>>> transformer('alea.txt','somme.txt')
```


Comment découper une ligne de texte ?

• Très souvent, l'information stockée dans un fichier contient divers éléments sur chaque ligne, séparés par une virgule (ou un espace, etc). C'est ce que fait Excel lorsqu'il sauve une feuille de calcul au format texte, pour que le programmeur puisse l'exploiter en programmant.

• Pour découper une ligne et obtenir une liste de ses constituants sous forme de chaînes, on utilisera la méthode `str.split(sep=' ')` :

```
>>> 'anglais 12 10 15 8 17'.split()
['anglais', '12', '10', '15', '8', '17']
>>> 'anglais,12,10,15,8,17'.split(',')
['anglais', '12', '10', '15', '8', '17']
```

• Inversement, la méthode `sep.join(L)` permet de recoller les éléments d'une liste de chaînes, avec un séparateur :

```
>>> '-'.join(['anglais','12','8','15'])
'anglais-12-8-15'
```


Résumé sur les fichiers-texte

• Ouverture en lecture :

```
f_in = open('foo.txt','r',encoding='utf-8')
```

• Ouverture en écriture :

```
f_out = open('foo.txt','w',encoding='utf-8')
f_out = open('foo.txt','a',encoding='utf-8')
```

• Fermeture :

```
f.close()
```

• Lecture :

```
f_in.read()        f_in.readline()        f_in.readlines()
```

• Ecriture :

```
f_out.write(x)
f_out.writelines(liste)
```