Python

Les fichiers et le module OS

Valérie Bellynck (2009),
d'après un diaporama d'Alain Bozzi (2008),
lui-même venant de celui de Christophe Morvan (2006)

Communication avec le système

- Les contenus de fichier
  - Pourquoi faire ? Par exemple
    - assurer la persistance des données. Contrairement à la RAM , les données stockées dans un fichier sont pérennes,
    - recopier le contenu d'un fichier en le corrigeant (encodage…)
  - Les fonctions de base du noyau de Python permettent
    - de lire, et écrire dans un fichier
    - de créer un nouveau fichier
    - de modifier le contenu d'un fichier
    - de concevoir des programmes réutilisables pour remédier à de problèmes du à des contenus de fichiers
    - …

Communication avec le système

- Les répertoires
  - Pourquoi faire ?
    - automatiser des tâches sur des listes de fichiers

C'est la motivation initiale des langages de scripts et la raison de ce nom "script"

  - Ces fonctions sont contenues dans le module OS

  - Elles permettent
    - de créer ou supprimer un fichier ou un dossier
    - de changer le nom d'un élément
    - de parcourrir la liste des éléments d'un répertoire
    - de chercher un élément d'un répertoire
      ou tous les éléments plus vieux qu'une date
    - de corriger automatiquement les contenus d'une liste de fichiers
    - …

Les fichiers

Utilisation des fichiers

- Pour manipuler les fichiers, il faut :

1. créer une variable logique de type fichier (descripteur de fichier) et ouvrir le fichier soit en mode lecture ou écriture.

2. effectuer le traitement (copie d'une liste dans le fichier, copie d'un fichier dans une liste, ...)

3. fermer le fichier en fin de traitement

Les fichiers

1/ créer une variable logique de type fichier (descripteur de fichier) et ouvrir le fichier soit en mode lecture ou écriture.

```
f = open("futilisateur.txt",'r')
```

`f` : variable de type fichier, (typage automatique fait par python)

`open ()` : fonction qui permet :

- l'ouverture d'un fichier passé en paramètre : ici `futilisateur.txt`
- de spécifier par un mode les opérations qui seront permises ici :
  `r` pour `read`, on ne pourra donc que lire le fichier.

Les fichiers

2/ effectuer le traitement,

comme ici : la copie du fichier dans une liste

```
maListe = [] # déclaration d'une liste vide


# lecture de toutes les lignes du fichier (readlines avec un s)
lesLignes = f.readlines()

  for i in lesLignes :
       maListe.append(i)
```

Les fichiers

Remarques importantes :

- Vous avez du observer que le fichier `futilisateurs.txt` n'était pas utilisé dans le traitement. On l'ouvre et puis c'est tout.

- En fait `f` est une variable qui pointe sur le fichier `futilisateurs.txt`. Pour illustrer cela, pensez à la laisse d'un chien.

**ATTENTION :* *comme ces fonctions sont natives dans le noyau de python, il ne faut pas faire l'import du module os*

Les fichiers

Ecriture d'une liste dans un fichier

```
noms = [“nadège”, “ludo”, “marie”, “alain”]

# ouverture

f = open(“futilisateurs.txt”, “w”)

# traitement

for i in noms :
     f.write(i)

# fermeture du fichier du fichier
f.close()
```

Quelques modes sur les fichiers

```
r  # ouvert en lecture

w  # ouvert en écriture-écrasement

a  # ouvert en écriture-ajout

r+ # ouvert en lecture/écriture

w+ # ouvert en lecture/écriture-écrasement

a+ # ouvert en lecture/écriture-ajout
```

Quelques fonctions sur les fichiers

```
# lecture de l'ensemble du fichier pointé par f sous forme d'une
chaîne de caractères
f.read()

# retourne une chaîne d'au plus taille caractères lus dans le
fichier à partir de la position courante.
f.read(taille)

# lecture du fichier pointé par f ligne par ligne sous la forme
d'une chaîne de caractères
f.readline()

# lecture de l'ensemble du fichier pointé par f sous
  la forme d'une liste de chaîne de caractères
f.readlines()
```

Les fonctions du `module os`

Attention => `from os import *` pour les utiliser

- `getcwd()` `# retourne le répertoire courant`
- `chdir(rep)` `# se place dans le répertoire rep`
- `rename(src,dest)` `# renomme src en dest`
- `remove(chemin)` et `rmdir(rep)` `# supprime le répertoire rep`
- `mkdir(rep)` `# crée le répertoire rep`
- `listdir(rep)` `# liste les fichiers ou les répertoires dans rep`
- `system(cde)` `# exécute la commande cde`

Exemple

Que fait le script suivant ?

```
rename('prog.py','tp3.py')
mkdir('u:nmonRepnpythonntp5')
system('cat /proc/pci |grep ati > log')
```

Les fonctions du `module os.path`

- `split(chemin)` `# fournit le tuple (repertoire, fichier)`
- `join(chemin,...)` `# fournit un nom complet :`
  `#  /plus/de/un/rep/unFichier`
- `exists(chemin)` `# vrai si chemin existe`
- `isfile(chemin)` `# vrai si chemin est un fichier`
- `isdir(chemin)` `# vrai si chemin est un répertoire`
- `walk(chemin,fonc,arg)`

La fonction os.walk(path) crée un générateur de triplets (root, dirs, files) dans l'arborescence de path. Un triplet est généré par répertoire visité. root représente le chemin d'accès du répertoire visité. dirs est la liste des sous-répertoires du répertoire root et files est la liste des fichiers du répertoire root.
Voir http://docs.python.org/library/os.html#os.walk

Exemple

Lister l'arborescence d'un répertoire avec listdir()

```
#! /usr/bin/python
# -*- coding: UTF-8 -*-
from os import *

rep = getcwd() # retourne le répertoire courant
print "Le répertoire courant est : ", rep

chdir(rep) # se place dans le répertoire rep

listRep = []
# liste les fichiers ou les répertoires dans rep
listeRep = listdir(rep)

for i in listeRep:
   print i
```

Exemple

Lister l'arborescence d'un répertoire avec os.walk()

```
import os.path

def listdirectory(path):
    fichier=[]
    for root, dirs, files in os.walk(path):
        for i in files:
            fichier.append(os.path.join(root, i))
    return fichier
```

Gadget

Utiliser le module `time` pour tester la rapidité de 2 fonctions

```
import time
def compare(arg):
    a = time.clock()
      # première fonction à tester, avec arg comme paramètre
    fonction1(arg)
    b = time.clock()
      # deuxième fonction à tester, avec arg comme paramètre
    fonction2(arg)
    c = time.clock()
    return b-a, c-b
print compare('c:/python24') # pour appel avec arg='c:/python24'
```

Résultat, par exemple :

```
(1.0782314512039937, 1.0392410208560028)
      ## : les 2 fonctions se valent...
```