CHAPITRE 9
FICHIERS

Programmation en Python

Fichiers

- Afin de manipuler une plus grande quantité d'information, il convient de séparer les données et les programmes qui les traitent.
- Les données seront stockées dans des fichiers et les programmes manipuleront ces fichiers.
- De plus, l'information contenue dans un programme est volatile, alors que les fichiers permettent de garder l'information de manière permanente.

Fichiers

- Un **fichier physique** est une collection d'informations numériques réunies sous un même nom et enregistrée sur une mémoire de masse (disque dur, clé USB,…).
- En Python, pour pouvoir manipuler un fichier physique, il faut lui associer un **fichier logique** correspondant, qui est objet-interface particulier. Les programmes Python se réfèrent aux fichiers logiques pour agir sur les fichiers physiques correspondants.

Fichiers

- L'instruction pour ouvrir un fichier logique à un fichier physique est :

```
fichier_logique =
open(fichier_physique, mode)
```

- L'instruction pour fermer un fichier logique est :

```
fichier_logique.close()
```

Fichiers

- `fichier_logique` est un identificateur (à penser comme une variable) qui fait référence au fichier physique que l'on désire manipuler.
- `fichier_physique` est une chaîne de caractères qui désigne le chemin d'accès (path) du fichier physique.
  - Exemple : `"/Users/jcabessa/Desktop/fichier1"`

Fichiers

- `mode` désigne le mode dans lequel on veut traiter le fichier en question. Pour les fichier à accès séquentiel, c'est une des trois chaînes suivantes :
  - `'r'` (read): le fichier est ouvert en lecture seule, il est impossible de le modifier. Si le fichier n'existe pas au préalable, cela génère une erreur.
  - `'w'` (write): le fichier est créé ou recréé (toujours) et ouvert en écriture. Si le fichier existe déjà, il sera écrasé et recréé.
  - `'a'` (append): le fichier est ouvert en écriture. Si le fichier existe, les données sont ajoutées à la fin, sinon il est créé.

Fichiers


```
# ouverture/création en mode écriture, puis fermeture
f1 = open("/Users/jcabessa/Desktop/MAIN/Courses/PYTHON/
2016_Paris_Lausanne/cours/code/fichier1.txt","w")
f1.close()

# ouverture en mode lecture puis fermeture
f1 = open("/Users/jcabessa/Desktop/MAIN/Courses/PYTHON/
2016_Paris_Lausanne/cours/code/fichier1.txt","r")
f1.close()
```

Fichiers

- On peut effectuer l'ouverture, la manipulation et la fermeture d'un fichier grâce au bloc d'instructions :

```
with open (f_physique, mode) as f_logique:
    instructions...
```

- Ne pas oublier l'indentation des instructions.
- On utilisera généralement cette syntaxe.

Fichiers


```
# ouverture/création en mode écriture, puis fermeture
with open("/Users/jcabessa/Desktop/MAIN/Courses/PYTHON/
2016_Paris_Lausanne/cours/code/fichier1.txt","w") as f1:
    pass
```

Fichiers

- Pour lire ou écrire dans un fichier, on applique des méthodes spécifiques au (à l'identificateur du fichier) logique (et non physique).
- On a la syntaxe habituelle :

```
fichier_logique.methode(args)
```

Fichiers

- `f.write(str)` : écrit la chaîne str dans f.
- `f.read(n)` : lit les n prochains caractères de f à partir de la position courante du curseur (retourne chaîne).
- `f.read()` : lit tous les caractères de f à partir de la position courante du curseur (retourne chaîne).
- `f.readline()` : lit la ligne de f à partir de la position courante du curseur (retourne chaîne).
- `f.readlines()` : lit toutes les lignes restantes de f à partir de la position courante du curseur (retourne liste de chaînes).

Fichiers

Programme

```
# création, ouverture, écriture et fermeture automatique
with open("/Users/jcabessa/Desktop/MAIN/Courses/PYTHON/
2016_Paris_Lausanne/cours/code/fichier1.txt","w") as f1:
    f1.write("1ere ligne\n") # \n code un retour de ligne
    f1.write("2eme ligne\n")
    f1.write("3eme ligne\n")
```

Fichiers


```
# ouverture, lecture et fermeture automatique
with open("/Users/jcabessa/Desktop/MAIN/Courses/PYTHON/
2016_Paris_Lausanne/cours/code/fichier1.txt","r") as f1:
    x = f1.read(5)
    print x
    y = f1.readline()
    print y
    z = f1.readlines()
    print z
```

Exécution

```
1ere
ligne

['2eme ligne\n', '3eme ligne\n']
```

Fichiers

Programme

```
# À chaque réouverture, le curseur se replace au début
with open("/Users/jcabessa/Desktop/MAIN/Courses/PYTHON/
2016_Paris_Lausanne/cours/code/fichier1.txt","r") as f1:
    x = f1.read()
    print x
```

Exécution

```
1ere ligne
2eme ligne
3eme ligne
```

Fichiers

- Parcours de fichiers : on peut parcourir les fichiers caractère par caractère ou ligne par ligne à l'aide d'une boucle « for ». On utilise les syntaxes :

```
with open (f_physique, mode) as f_logique:
    for x in f_logique:
        instructions...

with open (f_physique, mode) as f_logique:
    for l in f_logique.readlines():
        instructions...
```

Fichiers

Programme

```
with open("/Users/jcabessa/Desktop/MAIN/Courses/PYTHON/
2016_Paris_Lausanne/cours/code/fichier1.txt","r") as f1:
    for x in f1:
        print x
```

Exécution

```
1ere ligne

2eme ligne

3eme ligne
```

Fichiers

Programme

```
with open("/Users/jcabessa/Desktop/MAIN/Courses/PYTHON/
2016_Paris_Lausanne/cours/code/fichier1.txt","r") as f1:
    for x in f1.readlines():
        print x
```

Exécution

```
1ere ligne

2eme ligne

3eme ligne
```

Programme

- Nous allons d'abord définir une fonction qui détermine si un fichier existe ou non sur le disque.
- La fonction retourne un tuple avec le nom du fichier physique et le booléen associé à l'existence du fichier en question.
- On utilise la gestion des exceptions (Chapitre 7).

Programme

Programme

```
def isFile(fname):
    try:
        f_log = open(fname,'r')
        f_log.close()
        return (fname,True)
    except IOError:
        return (fname,False)
```

Programme

Exécution

```
>>> isFile("carnet.txt")
('carnet.txt', True)
>>> print isFile("mon_fichier.txt")
('mon_fichier.txt', False)
```

Programme

- Nous allons écrire un programme qui affiche à l'écran un fichier dont l'utilisateur donne le nom.
- On utilise la fonction `isFile()`.

Programme


```
Programmme
while True:
    physique = raw_input("nom du fichier: ")
    if physique == "":
        break
        # vérifier si le fichier existe
    if not isFile(physique)[1]:
        print "ce fichier n'existe pas"
    else:
        # afficher les lignes sur l'écran
        with open(physique,'r') as f:
            for x in f:
                print x,
```

Programme

Exécution

```
nom du fichier: carnet.txt
Lisa 05.05.1987 0697837465
nom du fichier: bla bla
ce fichier n'existe pas
nom du fichier:
Jeremies-MacBook-Pro:code jcabessa$
```

Programme

- Nous allons écrire un programme qui stocke dans des fichiers textes les lignes tapées au clavier par l'utilisateur.
- On utilise la fonction `isFile()`.

Programme

Programme

```
while True:
    physique = raw_input("nom du fichier: ")
    if physique == "":
        break
    rep = "n"
    if isFile(physique)[1]:
        print "Ce fichier existe deja"
        rep = raw_input("Voulez-vous l'ecraser (y/n)? ")
    if not isFile(physique)[1] or rep in ["y","Y"]:
        with open(physique,'w') as fl:
            print "Entrer les lignes a stocker"
            print "derniere ligne END"
            s = raw_input("> ")
            while s != "END":
                fl.write(s + "\n")
                s = raw_input("> ")
```

Programme


```
Exécution # On crée deux fichiers avec du texte
nom du fichier: test1.txt
Entrer les lignes a stocker
derniere ligne END
> Bonjour
> Au revoir
> END
nom du fichier: test2.txt
Entrer les lignes a stocker
derniere ligne END
> Nouveau texte
> etc.
> etc
> END
nom du fichier:
Jeremies-MacBook-Pro:code jcabessa$
```

Fichiers à accès direct

- Nous avons vu le cas de fichiers dits « à accès séquentiel », i.e., où la lecture et l'écriture se fait à partir de la dernière position du curseur atteinte dans le fichier.
- Il existe des fichiers, plus lourds, dits « à accès direct », i.e., où il est possible de lire et écrire à partir de n'importe quelle position donnée.
- De tels fichiers sont définis par des modes suivis de « + » et ils possède deux nouvelles méthodes…

Fichiers à accès direct

- Pour gérer des fichier à accès direct, on utilise les `modes` suivants :
  - `'r+'` (read): le fichier est ouvert en lecture seule, il est impossible de le modifier. Si le fichier n'existe pas au préalable, cela génère une erreur.
  - `'w+'` (write): le fichier est créé ou recréé (toujours) et ouvert en écriture. Si le fichier existe déjà, il sera écrasé et recréé.
  - `'a+'` (append): le fichier est ouvert en écriture. Si le fichier existe, les données sont ajoutées à la fin, sinon il est créé.

Fichiers à accès direct

- On a les deux nouvelles méthodes suivantes :
- `f.tell()` : retourne un entier qui correspond à la position actuelle du curseur dans f.
- `f.seek(n[,0,1,2])` : place le curseur de f en position n.
  - si second argument 0 ou omis, se place en position n à partir du début de f
  - si second argument 1, se place en position n à partir de la position actuelle du curseur de f
  - si second argument 2 ou omis, se place en position -n à partir de la fin de f

Fichiers à accès direct

Programme

```
# création et écriture dans un fichier à accès direct
with open("/Users/jcabessa/Desktop/MAIN/Courses/PYTHON/
2016_Paris_Lausanne/cours/code/fichier1.txt","w+") as f2:
    f1.write("1ere ligne bis\n")
    f1.write("2eme ligne bis\n")
    f1.write("3eme ligne bis\n")

# lecture d'un fichier à accès direct
with open("/Users/jcabessa/Desktop/MAIN/Courses/PYTHON/
2016_Paris_Lausanne/cours/code/fichier1.txt","r+") as f2:
        f2.seek(8)
        x = f2.read()
        print x
```

Fichiers à accès direct

Exécution

```
ne bis
2eme ligne bis
3eme ligne bis
```

Programme (faire démo)

- Au chapitre 4, on avait crée un programme qui permettait de gérer une sorte de carnet d'adresse, avec possibilité d'ajouter, rechercher des personnes, etc.
- On va améliorer ce programme pour que le carnet soit sauvé dans un fichier.

Programme (faire démo)


```
carnet = {} # création du dictionnaire

print "n : Introduire une nouvelle adresse"
print "r : Rechercher une adresse"
print "f : Fin"

while (1==1):
# cette boucle est exécutée tout le temps, i.e.,
# tant qu'elle n'est pas interrompue par break
```

Programme (faire démo)


```
action = raw_input("Que voulez-vous faire? ")
if (action in ["n", "N"]):
    # introduire une nouvelle adresse
    nom = raw_input("Entrer un nom: ")
    if nom in carnet.keys():
        print "Ce nom existe deja..."
    else:
        anniv = raw_input("anniversaire <j.m.a>: ")
        tel = raw_input("telephone: ")
        carnet[nom] = (anniv, tel)
        # on ajoute le tout dans le fichier!
        with open("/Users/jcabessa/Desktop/MAIN/Courses/
PYTHON/2016_Paris_Lausanne/cours/code/carnet.txt","a") as f:
            f.write(nom + " " + anniv + " " + tel + "\n")
```

Programme (faire démo)


```
elif (action in ["r", "R"]):
    nom = raw_input("Entrer un nom: ")
    if nom in carnet.keys():
        print "anniversaire:", carnet[nom][0]
        print "telephone:", carnet[nom][1]
    else:
        print "Ce nom n'existe pas..."
elif action in ["f", "F"]:
   # impression du dico carnet
   for k in carnet.keys():
         print k, carnet[k][0], carnet[k][1]
   break # interruption de la boucle while
```

Programme (faire démo)

- Dans ce programme, on repart chaque fois à partir d'un dictionnaire vide; cf. instruction `carnet = {}`
- On aimerait bien pouvoir garder en mémoire, dans un fichier, l'état du dictionnaire `carnet`.
- Ainsi, on pourrait repartir de ce dictionnaire `carnet` à chaque réexécution du programme.
- Pour cela, on utilise le module « pickle »…

Le module « pickle »

- Le module « pickle » permet d'implémenter un algorithme de codage et décodage d'objets Python.
- Le « pickling » désigne le processus selon lequel un objet Python est convertit en une suite de bytes, afin de pouvoir être stocké dans un fichier.
- Le « unpickling » désigne le processus inverse selon lequel une suite de bytes est reconvertie en objet Python, afin de pouvoir être réutilisée dans un programme.

Le module « pickle »

- Pour appeler le module pickle, on utilise l'instruction :

```
from pickle import *
```

- Opération de stockage d'un objet Python `x` dans un fichier logique `f_log` : on utilise l'instruction `dump`.

```
dump(x,f_log)
```

- Opération de répupération d'un objet Python qui est stocké dans un fichier logique `f_log` : on utilise l'instruction `load`.

```
load(f_log)
```

Le module « pickle »

- Exemple : on crée une liste d'objets de types différents, on la stocke dans un fichier, puis on la récupère.

Le module « pickle »


```
date  = "17-JAN-1990"
piece = ("roi", "dame", "fou", "tour", "cavalier", "pion")
b = "blanc"
n = "noir"
joueurs = (("Karpov",b), ("Kasparov",n))
jeu = {("e",1) : (piece[0],b), ("e",3) : (piece[0],n)}
# liste de chaîne, tuple, dictionnaire,...
liste = [date, piece, b, n, joueurs, jeu]
```

Le module « pickle »


```
# On se place dans le répertoire voulu
import os
os.chdir("/Users/jcabessa/Desktop/MAIN/Courses/PYTHON/
2016_Paris_Lausanne/cours/code")

# On crée un fichier dans lequel on stocke l'objet liste.
# On utilise l'instruction dump.
from pickle import *
with open("my_file.txt","w") as f_log:
    dump(liste,f_log) # on dump l'objet liste
```

Le module « pickle »


```
# On récupère l'objet liste.
# On utilise l'instruction load.
with open("my_file.txt","r") as f_log:
    liste = load(f_log) # on load le contenu de f_log

# On a bien récupéré l'objet liste de départ
print liste
for x in liste:
    print x
    print type(x)
```

Le module « pickle »


```
# On a bien récupéré l'objet liste de départ
print liste
['17-JAN-1990',
('roi', 'dame', 'fou', 'tour', 'cavalier', 'pion'),
'blanc', 'noir',
(('Karpov', 'blanc'), ('Kasparov', 'noir')),
{('e', 1): ('roi', 'blanc'), ('e', 3): ('roi', 'noir')}]
```

Le module « pickle »


```
# Les différents types des objets sont tous compris
for x in liste:
    print x
    print type(x)
17-JAN-1990
<type 'str'>
('roi', 'dame', 'fou', 'tour', 'cavalier', 'pion')
<type 'tuple'>
blanc
<type 'str'>
noir
<type 'str'>
(('Karpov', 'blanc'), ('Kasparov', 'noir'))
<type 'tuple'>
{('e', 1): ('roi', 'blanc'), ('e', 3): ('roi', 'noir')}
<type 'dict'>
```

Programme (faire démo)

- On modifie le programme de notre carnet d'adresse de sorte que le dictionnaire carnet soit « dumpé » dans un fichier `carnet.txt` après chaque utilisation.
- À cahque nouvelle utilisation du programme, si le fichier `carnet.txt` existe, on load le dictionnaire stocké dans ce fichier.
- On utilise la fonction `isFile()` déjà programmée.

Programme (faire démo)


```
# On rappelle la fonction isFile.
def isFile(fname):
    try:
        f_log = open(fname,'r')
        f_log.close()
        return (fname,True)
    except IOError:
        return (fname,False)
```

Programme (faire démo)


```
# Récupération ou création du dictionnaire carnet
# On se place dans le répertoire voulu
import os
os.chdir("/Users/jcabessa/Desktop/MAIN/Courses/PYTHON/
2016_Paris_Lausanne/cours/code")

from pickle import *
    if isFile("carnet.txt")[1] == True:
        with open("carnet.txt","r") as f_log:
            carnet = load(f_log) # On load le dico
    else:
        carnet = {} # On crée le dico
```

Programme (faire démo)


```
print "n : Introduire une nouvelle adresse"
print "r : Rechercher une adresse"
print "f : Fin"

while (True):
    action = raw_input("Que voulez-vous faire? ")
    if (action in ["n", "N"]):
        # introduire une nouvelle adresse
        nom = raw_input("Entrer un nom: ")
        if nom in carnet.keys():
            print "Ce nom existe deja..."
        else:
            anniv = raw_input("anni <jour.mois.annee>: ")
            tel = raw_input("telephone: ")
            carnet[nom] = (anniv, tel)
```

Programme (faire démo)


```
    elif (action in ["r", "R"]):
        nom = raw_input("Entrer un nom: ")
        if nom in carnet.keys():
            print "anniversaire:", carnet[nom][0]
            print "telephone:", carnet[nom][1]
        else:
            print "Ce nom n'existe pas..."
    elif action in ["f", "F"]:
        # ouverture en mode "w": on recrée le fichier
        with open("carnet.txt","w") as f_log:
            # On dump le dico dans carnet.txt
            dump(carnet,f_log)
            print carnet
            break
```