Informatique en CPGE (2018-2019)
Les fichiers

1 Gestion des fichiers

Un programme s'exécute dans la mémoire volatile (RAM) de l'ordinateur. Pour conserver une trace, il faut utiliser une mémoire permanente comme un disque ou une clé où les données sont organisées en fichiers par le système d'exploitation.

Le répertoire par défaut est le répertoire où est installé Python. Si on travaille avec un fichier Python, le répertoire courant est celui où est enregistré le fichier Python. On peut changer ce répertoire de travail avec la fonction `chdir` (change directory) du module `os`, par exemple :

```
from os import chdir
chdir('C:/Users/Toto/Info/tp7')
```

Le répertoire "tp7" est maintenant le répertoire où seront créés les fichiers. Attention, ce répertoire doit préalablement exister. ".py", le répertoire le fichier ".py".

1.1 Ouverture d'un fichier

Il y a trois manières d'ouvrir un fichier en mode texte : on utilise la fonction `open` qui prend comme premier paramètre le nom du fichier et en second paramètre 'w' pour le mode "écriture", ou 'r' pour le mode "lecture" ou 'a' pour le mode "ajout".

- En mode écriture, le fichier est créé dans le répertoire courant du programme ou écrasé s'il existe déjà puis ouvert en écriture.
- En mode ajout le fichier doit déjà exister, il est ouvert en écriture et toutes les données écrites sont automatiquement ajoutées à la fin du fichier.
- En mode lecture le fichier doit déjà exister et est ouvert en lecture.

La syntaxe est :

```
fic1 = open('fichier1', 'w')
fic2 = open('fichier2', 'r')
fic3 = open('fichier3', 'a')
```

1.2 Fermeture d'un fichier

Pour fermer un fichier, il y a une seule instruction :

```
fic1.close()
```

A la fin de l'utilisation, il est **impératif** de fermer le fichier, sinon son contenu ne peut pas être garanti.

2 Ecriture et lecture

2.1 Ecriture

Par défaut, Python utilise les fichiers en mode texte et on y écrit des chaînes de caractères (type **str**) en utilisant la méthode `write`. Voici un premier exemple :

```
fic = open('fichier1.txt', 'w')
fic.write('Bonjour, comment allez-vous ?')
fic.close()
```

Ce code crée un fichier "fichier1.txt" dans le répertoire courant. Ce fichier contient sur une ligne la phrase : Bonjour, comment allez-vous ?

Pour écrire sur plusieurs lignes, on utilise le caractère '\n' qui provoque un retour à la ligne.
Le code est :

```
fic = open('fichier1.txt', 'w')
fic.write('Bonjour,' + '\n' + 'comment allez-vous ?')
fic.close()
```

Le fichier contient alors les deux lignes :
Bonjour,
comment allez-vous ?

Attention le code qui suit ne fonctionne pas pour écrire la phrase sur deux lignes :

```
fic = open('fichier1.txt', 'w')
fic.write('Bonjour,')
fic.write('comment allez-vous ?')
fic.close()
```

On obtient la phrase sur une seule ligne sans espace après "Bonjour,".

Si on relance le même programme, le fichier qui existe déjà va être écrasé et réécrit.
Si on ne souhaite pas l'écraser et écrire à la suite dans ce fichier, on le rouvre avec l'instruction `fic=open('fichier1.txt','a')`.
Le code qui suit ajoute une nouvelle ligne à la fin du fichier texte :

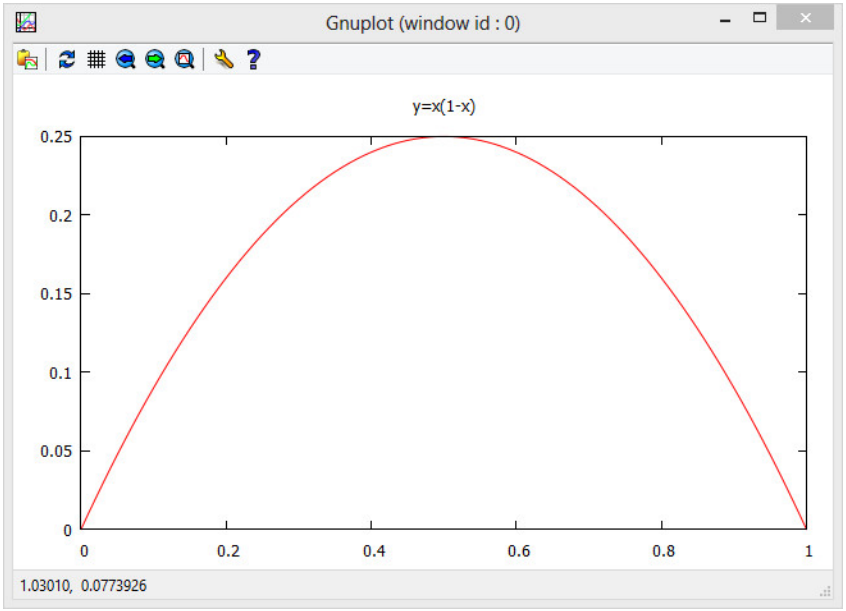
```
fic = open('fichier1.txt', 'a')
fic.write('\n' + 'Au revoir')
fic.close()
```

Pour écrire des données numériques de type **int** ou **float**, il suffit de les convertir préalablement en type **str**.
Par exemple si on souhaite écrire un tableau de valeurs pour une fonction, on convertit les nombres de type **float** en chaîne de caractères de type **str** et on utilise la méthode **write** :

```
def f(x):
    return x * (1 - x)
fic = open('fichier2.dat', 'w')
for i in range(101):
    x = i / 100
    y = f(x)
    fic.write(str(x) + '\t' + str(y) + '\n')
# '\t' pour une tabulation et '\n' pour un retour à la ligne
fic.close()
```

Ensuite on ouvre un logiciel comme Gnuplot par exemple où on écrit les instructions :
set title "y=x(1-x)"
plot 'C :\Users\Toto\Info\tp7\fichier1.dat' w l

Gnuplot lit alors le fichier texte de données et trace la courbe correspondante.



2.2 Lecture

Pour lire dans un fichier texte, on ouvre le fichier en lecture et on utilise la méthode `read`. Il ne faut pas oublier de fermer le fichier après la lecture. (fic.read(n) lit n caractères, fic.read() lit tout le fichier).

```
fic = open('fichier1.txt', 'r')
monfichier = fic.read() # monfichier est de type str
print(monfichier) # ou print(fic.read()) lit et affiche tout le fichier
fic.close()
```

L'objet `fic` est un ensemble de lignes, chacune étant une chaîne de caractères. On peut récupérer une ligne dans une variable de type **str** avec le code suivant :

```
fic = open('fichier1.txt', 'r')
ligne = fic.readline() # lit la ligne courante et passe à la suivante
fic.close()
```

On peut aussi récupérer toutes les lignes dans une liste. Chaque élément de la liste est alors une chaîne de caractères. Chaque chaîne est une ligne du fichier.

```
fic = open('fichier1.txt', 'r')
lignes = fic.readlines() #liste de lignes
print(len(lignes)) # affiche le nombre de lignes
print(lignes[0]) # affiche la première ligne
fic.close()
```

On peut décomposer chaque ligne qui est une chaîne de caractères en mots. La méthode `rstrip()` supprime le caractère de retour à la ligne et la méthode `split()` découpe la ligne et la transforme en une liste de mots lorsque ces mots sont séparés par des espaces ou n'importe quel caractère.

Pour ce code, on a créé un fichier 'lecture.txt' qui contient la chaîne "bonjour tout le monde 25 53.4".

```
fic = open('lecture.txt', 'r')
chaine = fic.read()
print(chaine.split())
fic.close()
```

Ce code affiche : `['bonjour', 'tout', 'le', 'monde', '25', '53.4']`. On a créé une liste de mots à partir d'une chaîne.

Si les données à récupérer sont de type numérique, il suffit alors de convertir chaque mot en **int** ou en **float**. Avec le fichier de donnés "fichier2.dat', créé plus haut, qui contient des lignes de deux flottants séparés par une tabulation ('\t'), le code est :

```
fic = open('fichier2.dat', 'r')
for ligne in fic:
    coord = ligne.rstrip().split('\t') # coord est une liste de 2 mots
    x, y = float(coord[0]), float(coord[1])
```

3 Fichiers binaires

Un fichier est une suite de 0 et de 1 que l'on regroupe par octet. Par exemple l'octet qui vaut 41 en hexadécimal peut représenter le nombre entier 65 qui indiquera la quantité de rouge pour un pixel donné dans une image ou bien le code ASCII du caractère 'A' s'il s'agit d'un fichier texte.

C'est le format du fichier qui indique comment il va être interprété.

Les méthodes `write(n)` et `read(n)` permettent d'écrire ou de lire n octets en mode binaire. Les objets lus ou écrits sont du type **bytes** (tableau d'octets).

```
f = open("nom1", "rb") # "b" utilisé pour ouvrir le fichier en mode binaire
g = open("nom2", "wb") # "wb" utilisé pour écrire en mode binaire
octets = f.read(3) # lit 3 octets  (octets est du type bytes)
g.write(octets) # écrit les 3 octets
# ou g.write(b'\x00\x41\x80\xff') écrit 4 octets
# ou g.write(b'\x00A\x80\xff') écrit les 4 mêmes octets
# ou g.write(bytes([0,65,128,255])) écrit les 4 mêmes octets
f.close()
g.close()
```