Table des matières

1 Commandes pour la manipulation d'un fichier texte **1**
1.1 Objet fichier 1
1.2 Méthodes des objets fichiers 1

2 Exemples **1**

3 Stockage de données numériques dans un fichier texte **2**
3.1 $1^{ère}$ méthode : une valeur par ligne 2
3.2 $2^{ème}$ méthode : utiliser un séparateur et `split()` 2
3.3 $3^{ème}$ méthode : avec une liste et `eval()` 3

1 Commandes pour la manipulation d'un fichier texte

1.1 Objet fichier

En `python`, création et manipulation d'un fichier se font par l'intermédiaire d'un objet particulier, appelé objet-fichier, généré par la fonction :

`objet_fichier = open(nom du fichier, mode d'accès)` .

Les paramètres sont des chaines de caractère :

– `nom du fichier` est le nom du fichier, avec son extension.

– `mode d'accès` peut être :

1. `'w'` : (write) ouverture pour écriture seule. Lorsque le fichier n'existe pas il est créé dans le répertoire courant ; lorsque le fichier existe il est écrasé.
2. `'a'` : (append) ouverture pour écriture seule. Lorsque le fichier n'existe pas il est créé dans le répertoire courant ; lorsque le fichier existe les données écrites le seront à la suite.
3. `'r'` : (read) ouverture pour lecture seule. Le ficher doit exister dans le répertoire courant.
4. `'+'` : ouverture pour lecture et écriture. Le fichier doit exister.

Une fois la lecture et l'écriture dans le fichier terminés il faut refermer le fichier avec l'instruction :

```
objet_fichier.close()
```

1.2 Méthodes des objets fichiers

Un objet-fichier admet les méthodes :

Lorsque **ouvert en écriture** :

`write()` écrit une chaine de caractère en fin de fichier ouvert en écriture.

`writelines()` écrit une liste de chaˆ nes de caractères en les concaténant.

lorsque **ouvert en lecture** :

`read()` lit l'intégralité du fichier.

`readline()` lit la ligne suivante.

`readlines()` retourne une liste contenant toutes les lignes du fichier.

On peut aussi parcourir le fichier ligne après ligne avec une boucle `for` :

```
for ligne in objet_fichier
```

2 Exemples

Voici un fichier texte, saisi à l'aide d'un éditeur de texte :

monfichier.txt

```
Premiere ligne
Deuxieme ligne...Suite deuxieme ligne
Troisieme ligne
```

On peut le lire par le code :

```
>>> objet = open('monfichier.txt','r')
>>> lect = objet.read()
>>> print(lect)
Premiere ligne
Deuxieme ligne...Suite deuxieme ligne
Troisieme ligne
>>> objet.close()
```

```
>>> objet = open('monfichier.txt','r')
>>> liste = objet.readlines()
>>> print liste
['Premiere ligne\n', 'Deuxieme ligne...Suite deuxieme ligne\n', 'Troisieme ligne\n']
>>> objet.close()
```

Compter le nombre de ligne du fichier texte par :

```
def nbreLignes(fichier):
    objet = open(fichier,'r')
    compteur = 0
    for ligne in objet:    # Parcours des lignes du fichier
        compteur += 1
    return compteur
```

```
>>> nbreLignes('monfichier.txt')
3
```

Ajouter une ligne en fin du fichier texte :

```
def ajouteLigne(fichier,ligne):
    objet = open(fichier,'a')
    objet.write(ligne + '\n')
    objet.close()
```

```
>>> ajouteLigne('monfichier.txt','Quatrieme ligne rajoutee')
```

Notre fichier texte est devenu :

monfichier.txt

```
Premiere ligne
Deuxieme ligne...Suite deuxieme ligne
Troisieme ligne
Quatrieme ligne rajoutee
```

Exemple : supprimer une ligne d'un fichier texte :

```
def effaceLigne(fichier,i):
    objet = open(fichier,'r')
    liste = objet.readlines()
    liste.pop(i-1)
    objet.close()
    objet = open(fichier,'w')
    objet.writelines(liste)
    objet.close()
```

monfichier.txt

```
Premiere ligne
Troisieme ligne
Quatrieme ligne rajoutee
```

3 Stockage de données numériques dans un fichier texte

3.1 $1^{ère}$ méthode : une valeur par ligne

Première méthode : On peut le faire en les stockant sous forme de chaines de caractères séparées par un séparateur ; le retour à la ligne `'\n'` par exemple :

Exemple : une fonction écrivant sous forme texte des données numériques issues d'une liste :

```
def wdata(fichier,liste):
    f = open(fichier,'w')    # Ouverture en écriture
    for x in liste:    # Parcours de la liste
        # Ecriture de la donnée convertie en str + '\n'
        f.write(str(x)+'\n')
    objet.close()
```

Lecture des données à partir du fichier et renvoi dans une liste :

```
def rdata(fichier):
    f = open(fichier,'r')
    liste = f.readlines()
    f.close()
    return [float(x) for x in liste]
    # Convertir en float
```

Création d'un fichier `datafile` contenant les nombres entiers de 1 à 10.

```
>>> wdata('datafile',[1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

datafile

```
1
2
3
4
5
6
7
8
9
10
```

Lecture de ses données :

```
>>> print(rdata('datafile'))
[1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]
```

Ajout d'une donnée en fin de fichier :

```
def adata(fichier,x):    # Ajout d'un nombre en fin de fichier
    f = open(fichier,'a'); f.write(str(x)+'\n'); f.close()
```

```
>>> adata('datafile', 11)
```

3.2 $2^{ème}$ méthode : utiliser un séparateur et `split()`

Deuxieme méthode : On peut le faire en les stockant sur une ligne sous forme de chaines de caractères séparées par un séparateur : un point-virgule `';'` par exemple :

Fonction écrivant des données numériques issues d'une liste :

```
def wdata(fichier,liste):
    f = open(fichier,'w')    # Ouverture en écriture
    for x in liste[:-1]:    # Parcours de la liste
        # Ecriture de la donnée convertie en str + ';'
        f.write(str(x) + ';')
    f.write(str(liste[-1]))
    objet.close()
```

Lecture des données à partir du fichier et renvoi dans une liste :

```
def rdata(fichier):
    f = open(fichier,'r')
    ligne = f.readline()
    liste = ligne.split(';')     # crée la liste des mots séparés par ';'
    f.close()
    result = []
    for x in liste:
        result.append(float(x))
```

3.3 $3^{ème}$ méthode : avec une liste et `eval()`

Troisième méthode : On stocke la liste des données numériques (convertie en chaine de caractère).

Fonction écrivant des données numériques issues d'une liste :

```
def wdata(fichier,liste):
    f = open(fichier,'w')    # Ouverture en écriture
    f.write(str(liste))
    objet.close()
```

• Pour la lecture nous allons utiliser la fonction `eval()` : elle prend en paramètre une chaine de caractère représentant une donnée, ou une instruction, et la convertit en ce qu'elle représente :

```
>>> eval('[1, 2]')
[1,2]
>>> eval('[1,2]\n')    # en ignorant les caractères spéciaux !
[1, 2]
>>> eval("print('bonjour')")    # Pour une instruction elle provoque l'exécution
bonjour
```

```
def rdata(fichier):
    f = open(fichier,'r')
    ligne = f.readline()
    liste = eval(ligne)     # eval() reconvertit en ce que ça représente
    return liste
```