PYTHON – Fichiers

Introduction

Nous allons étudier les fichiers :

comment les ouvrir, les lire, écrire dedans.

Utilité :

Jusqu'à présent, les programmes que nous avons réalisés ne traitaient qu'un très petit nombre de données. Nous pouvions donc à chaque fois inclure ces données dans le corps du programme lui-même (par exemple dans une liste).
Cette façon de procéder devient inadaptée lorsque l'on souhaite traiter une quantité d'informations plus importante.
il est temps que nous apprenions à *séparer les données et les programmes qui les traitent dans des fichiers différents*.

Pour que cela devienne possible, nous devrons doter nos programmes de divers mécanismes permettant de créer des fichiers, d'y envoyer des données et de les récupérer par la suite.

NB : Lorsque les quantités de données deviennent très importantes, il devient nécessaire d'élaborer des *bases de données relationnelles*.

Travailler avec des fichiers

L'utilisation d'un fichier ressemble beaucoup à l'utilisation d'un livre :

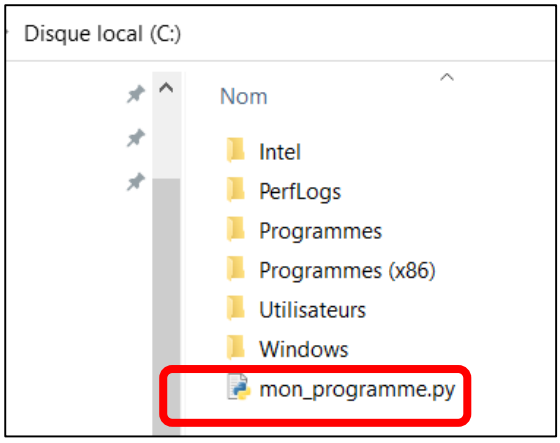
- Pour utiliser un livre, vous devez d'abord le trouver (à l'aide de son titre)
- Ensuite vous ouvrez le livre.
- Tant qu'il est ouvert, vous pouvez y lire des informations diverses, et vous pouvez aussi y écrire des annotations, mais généralement vous ne faites pas les deux à la fois.
- Lorsque vous avez fini de l'utiliser, vous le refermez.

Ce que nous venons de dire des livres s'applique également aux fichiers informatiques :

- Un fichier se compose de données enregistrées sur votre disque dur, sur une disquette, une clef USB ou un CD. Vous y accédez grâce à son nom (lequel peut inclure aussi un nom de répertoire).
- Vous pouvez considérer le contenu d'un fichier comme une suite de caractères, ce qui signifie que vous pouvez traiter ce contenu à l'aide des fonctions servant à traiter les chaînes de caractères.

le répertoire de travail courant

Quand vous lancez un programme Python directement, par exemple en faisant un double-clic dessus, le répertoire courant est celui d'où vous lancez le programme .py.
Si vous avez un fichier mon_programme.py contenu sur le disque C:, le répertoire de travail courant quand vous lancerez le programme sera C:\.



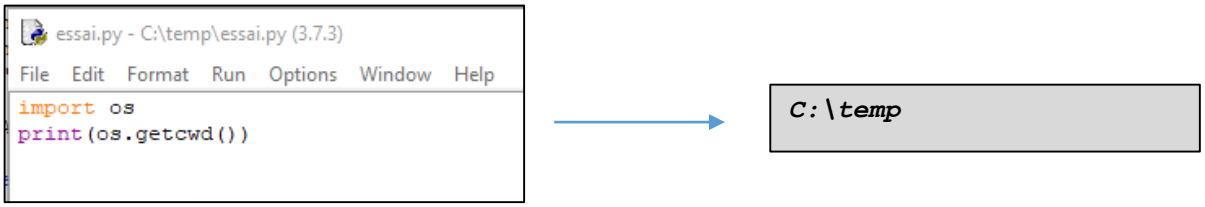
Afficher le répertoire courant

quand on lance l'interpréteur Python, on a un répertoire de travail courant.

Vous pouvez l'afficher grâce à la fonction **os.getcwd**()

(CWD = « Current Working Directory »).

Exemple, si mon programme est situé dans le répertoire C:/temp/essai.py :



Modifier le répertoire courant

Si vous souhaitez changer de répertoire de travail courant, vous devez utiliser la fonction **chdir** du module os (Change Directory).

```
import os
os.chdir("C:/tests_python")
```

Pour que cette instruction fonctionne, le répertoire doit exister.

Chemins relatifs et absolus

Pour décrire l'arborescence d'un système, on a deux possibilités :

- les chemins absolus ;
- les chemins relatifs.

1) Le chemin absolu

Quand on décrit une cible (un fichier ou un répertoire) sous la forme d'un chemin absolu, on **décrit l'intégralité du chemin (la suite des répertoires) menant au fichier**.

- Sous Windows, on partira du nom de volume (C:/,D:/…). Par exemple : C:/test/fic.txt
- Sous les systèmes Unix, ce sera depuis **/**.

2) Le chemin relatif

Quand on décrit la position d'un fichier grâce à un chemin relatif, cela veut dire que l'**on tient compte du dossier dans lequel on se trouve** actuellement.

- Ainsi, si on se trouve dans le dossier C:/test et que l'on souhaite accéder au fichier fic.txt contenu dans ce même dossier, le chemin relatif menant à ce fichier sera fic.txt. Mais si on se trouve dans le dossier C:, le chemin relatif sera test/fic.txt.
- Quand on décrit un chemin relatif, on utilise parfois le symbole .. qui désigne le répertoire parent.
  Si le répertoire de travail courant est test et que l'on souhaite accéder à fic2.txt qui se trouve dans un répertoire test2, notre chemin relatif sera ..\test2\fic2.txt.

Nom des fichiers

Choisissez de préférence des noms de fichiers courts.
Évitez dans toute la mesure du possible les caractères accentués, les espaces et les signes typographiques spéciaux.
Dans les environnements de travail de type Unix (MacOS, Linux, ...), il est souvent recommandé aussi de n'utiliser que des caractères minuscules.

Ouverture du fichier

Pour ouvrir un fichier avec Python, on utilise la fonction **open**, disponible sans avoir besoin de rien importer.

Elle prend en paramètre :

- le **chemin** (absolu ou relatif) menant au fichier à ouvrir ;
- le **mode d'ouverture**.

  Le mode est donné sous la forme d'une chaîne de caractères :

  - 'r': ouverture en lecture (Read). C'est le mode par défaut donc le paramètre peut être omis.
  - 'w': ouverture en écriture (Write). Le contenu du fichier est écrasé. Si le fichier n'existe pas, il est créé.
  - 'a': ouverture en écriture en mode ajout (Append). On écrit à la fin du fichier sans écraser l'ancien contenu du fichier. Si le fichier n'existe pas, il est créé.

```
mon_fichier = open("fic.txt", "r")
```

Ici nous souhaitons lire le fichier. Nous avons donc utilisé le mode 'r'.

Fermeture de fichier

N'oubliez pas de fermer un fichier après l'avoir ouvert.

Si d'autres applications, ou d'autres morceaux de votre propre code, souhaitent accéder à ce fichier, ils ne pourront pas car le fichier sera déjà ouvert.

C'est surtout vrai en écriture, mais prenez de bonnes habitudes.

La méthode à utiliser est **close** :

```
mon_fichier.close()
```

Lecture de l'intégralité du fichier

Pour ce faire, on utilise la méthode read de la classe TextIoWrapper.

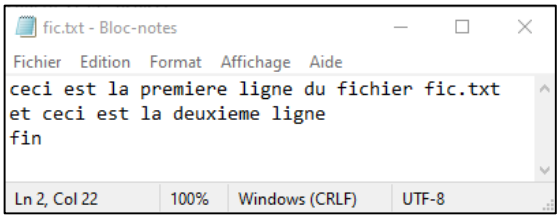
Elle renvoie tout le contenu du fichier, que l'on capture dans une chaîne de caractères :

```
mon_fichier = open("fic.txt", "r")
contenu = mon_fichier.read()
print(contenu)
mon_fichier.close()
```

Code.py

On part du principe que le fichier *fic.txt* se trouve au même endroit que le code *Code.py*, je peux donc indiquer le chemin relatif "fic.txt"

Exemple de fichier:



fic.txt

Résultat dans la console :

```
ceci est la premiere ligne du fichier fic.txt
et ceci est la deuxieme ligne
fin
```

Avec la méthode read, on peut limiter le nombre des caractères lus, par exemple:

```
suivant10= f.read(10) # lit au plus 10 caractères à partir de la position courante
```

Lecture de chaque ligne du fichier

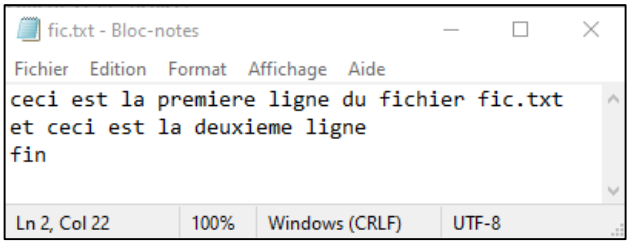
SI on veut lire chaque ligne du fichier, il suffit d'ouvrir le fichier puis d'utiliser une boucle

```
mon_fichier = open("fic.txt", "r")
for ligne in mon_fichier :
        print(ligne)
mon_fichier.close()
```

Si chaque fin de ligne contient un saut de ligne, vous aurez dans votre print(ligne)

une ligne vide traduisant un saut de ligne.

Exemple :



Résultat dans la console :

```
ceci est la premiere ligne du fichier fic.txt

et ceci est la deuxieme ligne

Fin
```

Lecture : autre possibilités

Autres possibilités de lire les données d'un fichier:

```
ligne = f.readline() # lit une ligne et la stocke comme une chaîne de caractères
lignes = f.readlines() # lit toutes les lignes et les stocke comme une liste de chaînes
lignes = list(f) # lit toutes les lignes et les stocke comme une liste de chaînes
```

Les deux dernières méthodes risquent d'utiliser beaucoup de mémoire si le fichier est grand. Il vaut mieux les éviter s'il n'est pas nécessaire de mémoriser toutes les données en même temps.

Écriture dans un fichier

il faut ouvrir le fichier avant tout avec open :

- mode w : écrase le contenu éventuel du fichier

Puis, pour écrire dans un fichier, on utilise la méthode **write** en lui passant **en paramètre la chaîne à écrire** dans le fichier.

Elle renvoie le nombre de caractères qui ont été écrits.

```
mon_fichier = open("fic.txt", "w")
mon_fichier.write("Premier test d'écriture\n")
mon_fichier.close()
```

Attention :

- La méthode write n'accepte en paramètre que des chaînes de caractères. Si vous voulez écrire dans votre fichier des nombres, il faudra les convertir en chaîne avant de les écrire *str()* et les convertir en entier après les avoir lus *int()*.
- La méthode write n'ajoute pas d'elle-même le retour à la ligne. Si on le veut, il faut insérer le caractère saut de ligne « \n » après chaque écriture.

Ajout dans un fichier

il faut ouvrir le fichier avant tout avec open :

- mode a : ajoute ce que l'on écrit à la fin du fichier, sans l'écraser comme c'est le cas dans le mode 'w'.
  Si le fichier demandé n'existe pas, il sera créé.

Puis, pour écrire dans un fichier, on utilise également la méthode **write**

```
mon_fichier = open("fic.txt", "a")
mon_fichier.write("ajout de texte\n")
mon_fichier.close()
```

Ouverture du fichier

Deuxième méthode d'ouverture de fichier :

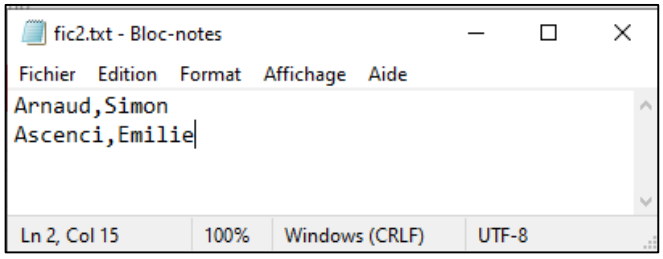
```
with open("fic.txt", 'r') as f:
   for ligne in f:
      # traitement prévu
```

à la fin du bloc with open(), le fichier est automatiquement fermé

Exemple de code

Lire un fichier contenant les noms d'étudiants formaté comme ci-dessous:

(ligne par ligne : Nom,Prénom avec une virgule comme séparateur)




```
division= []
with open("fic2.txt") as f:
        for ligne in f:
                division.append(ligne.strip().split(','))
print(division)
```

- Chaque ligne du fichier se termine avec un retour à la ligne ('\n') : strip() nous en débarrasse
- puis split() sépare le résultat au niveau de la virgule en deux chaînes (sous forme d'une liste contenant nom et prénom).
- Ensuite la liste est ajouté à la fin de la liste d'étudiants division avec append(
- A la sortie du bloc with, le fichier est fermé et la variable division contient les noms et prénoms des étudiants: [('Arnaud', 'Simon'), ('Ascensi', 'Emilie'), ...]

Résultat dans la console :

```
[['Arnaud', 'Simon'], ['Ascenci', 'Emilie']]
```

Ensuite, on veut saisir (au clavier) les notes du groupe d'étudiants, et l'enregistrer dans un nouveau fichier

```
division= []
with open("fic2.txt") as f:
        for ligne in f:
                division.append(ligne.strip().split(','))
print(division)
fichier_de_notes = input("Quel nom de fichier pour les notes? ")
with open(fichier_de_notes, 'w') as f:
   for nom, prenom in division:   # parcours d'une liste
      note = float(input('La note de ' + prenom + ' ' + nom + ' :'))
      f.write(nom + ',' + prenom + ',' + str(note) + '\n')
```

- Un nouveau fichier est créé.
- Dans chaque ligne on écrit les données d'un étudiant (nom, prénom et la note séparés avec des virgules).
- A la sortie du bloc with, le fichier est fermé et devient utilisable pour d'autres applications.

```
[['Arnaud', 'Simon'], ['Ascenci', 'Emilie']]
Quel nom de fichier pour les notes? fic3.txt
La note de Simon Arnaud :3
La note de Emilie Ascenci :12
```

Création du nouveau fichier fic3.txt :