

Dernière mise à jour	Informatique	Denis DEFAUCHY
18/11/2021	Bases de la programmation	12 – Fichiers

Informatique

12

Fichiers

Cours

Dernière mise à jour	Informatique	Denis DEFAUCHY
18/11/2021	Bases de la programmation	12 – Fichiers

Fichiers	3
1.I. Contexte.....	3
1.II. Préliminaires sur la gestion des fichiers	4
1.III. Lecture d'un fichier	5
1.III.1 Ouverture	5
1.III.2 Lecture des lignes	6
1.III.2.a Lecture ligne par ligne	6
1.III.2.b Lecture complète	7
1.III.3 Découpage des données de chaque ligne	8
1.III.4 Suppression du retour à la ligne	8
1.III.5 Post traitement : transformer des caractères	8
1.III.6 Fermeture	9
1.IV. Création et écriture dans un fichier	10
1.IV.1 Ouverture	10
1.IV.2 Ajout de lignes.....	10
1.IV.3 Fermeture	10



Dernière mise à jour	Informatique	Denis DEFAUCHY
18/11/2021	Bases de la programmation	12 – Fichiers

Fichiers

1.1. Contexte

La lecture ou l'écriture dans un fichier texte peut servir à réaliser une multitude de tâches. Toutefois, celle que vous serez amenés à réaliser le plus souvent consistera à extraire de fichiers textes des listes de données numériques issues d'un système de mesure quelconque permettant l'exportation au format texte (exemple : accélérations d'un accéléromètre permettant de remonter à la vitesse et la position...).

Selon les logiciels utilisés, les données seront regroupées et séparées différemment. Souvent, à chaque temps de mesure ou chaque mesure sera associée une ligne.

Pour chaque ligne, les différentes données peuvent être séparées par différents « séparateurs », virgule, point-virgule, espace, tabulation... Nous allons donc apprendre à les extraire

Il est possible de faire beaucoup de choses, nous verrons ici uniquement comment :

- Lire et extraire des données sous forme de listes
- Créer un fichier et y ajouter les valeurs d'une liste

Dernière mise à jour	Informatique	Denis DEFAUCHY
18/11/2021	Bases de la programmation	12 – Fichiers

1.II. Préliminaires sur la gestion des fichiers

Dans toute la suite, j'entends par « chemin » une variable de type string qui contient soit :

- Le nom d'un fichier avec son extension : « fichier.txt »
- Le nom du fichier précédé du chemin où il est enregistré, c'est-à-dire l'ensemble des dossiers depuis C:\ jusqu'au lieu où il se trouve, chemin dans lequel on aura pris soin de doubler tous les antislash \, par exemple : « C:\\Users\\denis\\Desktop\\Exemple\\fichier.txt »

Dans le premier cas, on parle de chemin **relatif**, dans le second, on parle de chemin **absolu**.

Le chemin absolu fonctionne toujours, mais ne permet pas de faire tourner un code sur deux ordinateurs différents sans le modifier à chaque fois. Le chemin relatif est bien plus pratique, mais pour fonctionner, il faut :

- Que le fichier python qui l'appelle soit :
 - o Enregistré et pas juste ouvert dans Pyzo comme nouveau document (temporaire on ne sait où dans l'ordinateur)
 - o Présent dans le même répertoire que le fichier appelé, d'où la notion de chemin relatif
- Exécuter le fichier Python avec la commande F5 (exécuter entièrement le code) ce qui permet à Pyzo de savoir où chercher le document en chemin relatif
- Cocher l'option de Pyzo dans le menu « Run » : « Change directory when executing file » (accessible uniquement sur les dernières versions de Pyzo – Présent dans la 4.5.0 par exemple)

Autrement dit, les deux actions suivantes ne fonctionneront pas en relatif :

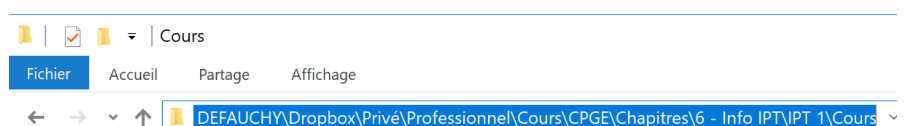
- N'exécuter qu'une portion d'un code avec Alt+Entrée
- Exécuter une commande quelconque directement dans la console


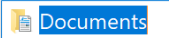
Par ailleurs, il semble que sur le réseau du Lycée, les chemins relatifs ne fonctionnent pas. De même, sur clé USB au Lycée, il faut forcément un chemin absolu.

Sous Windows, pour trouver un chemin absolu, il suffit de lancer l'explorateur Windows, d'aller dans le dossier de votre fichier, puis de cliquer dans la zone blanche après le chemin que vous voyez en haut :



Cela fera apparaître le chemin :



Remarque : lorsque vous êtes dans le dossier « Documents » : , le chemin n'est pas bon : . Il faut travailler autre part !

Dernière mise à jour	Informatique	Denis DEFAUCHY
18/11/2021	Bases de la programmation	12 – Fichiers

1.III. Lecture d'un fichier

1.III.1 Ouverture

Pour « ouvrir » un fichier texte sous python, on peut écrire :

```
fichier = open(Nom_Fichier, "r")
```

Quelques explications :

- A la variable `fichier` est associé le fichier texte. Nous pouvons parfaitement lui donner le nom que l'on veut. Dans ce cours, il me semble intéressant de l'appeler « fichier »...
- L'option `r` veut dire « read ». On ouvre donc le fichier en lecture uniquement (on ne peut pas écrire dedans).
- `Nom_Fichier` est une variable contenant le chemin du fichier à ouvrir :
 - En relatif : dans le même dossier sont présent le code python et le fichier texte de nom « Texte.txt », alors il suffit d'écrire avant le code ci-dessus :

```
Nom_Fichier = "Texte.txt"
```
 - En absolu : on veut ouvrir un fichier dans l'ordinateur à un endroit spécifique, on précise alors le chemin complet, par exemple :

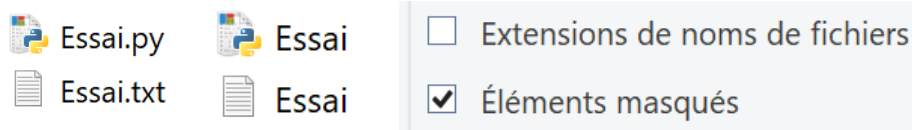
```
Nom_Fichier = "C:\\Users\\denis\\Desktop\\Texte.txt"
```

Veiller à doubler les anti-slashes

Remarques :

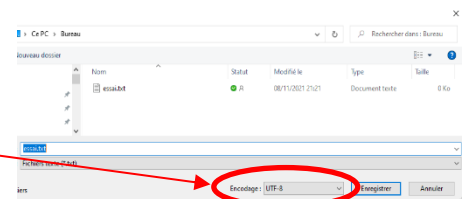
- On pourrait ne pas passer par la variable `Nom_Fichier` mais écrire directement :

```
fichier = open("Texte.txt", "r")
```
- Attention à ne pas ajouter l'extension « .txt » à un fichier texte (soit déjà un txt où l'extension serait masquée, conduisant à avoir : `essai.txt.txt...` Sur Windows, les extensions peuvent ou non apparaître selon l'option choisie, tout dépend de l'option choisie dans l'onglet « Affichage » (dépend des versions de Windows...)



- Selon les versions de logiciels, il arrive qu'il y ait une erreur à l'ouverture d'un fichier .txt à cause de son encodage, l'erreur s'appelle alors « charmap ». En fin 2021 et avec Anaconda, je ne reproduis pas cette erreur quel que soit l'encodage de mes fichiers textes... Pour solutionner ce problème s'il se présente, au choix :
 - Préciser le type d'encodage du fichier à ouvrir :

```
fichier = open('essai.txt', encoding='utf8')
```
 - Ouvrir le fichier .txt (en dehors de Python) et le réenregistrer en précisant l'encodage à utiliser
 - Sur mac par exemple, créer un fichier txt avec TextEdit et coller le contenu du fichier source dedans (merci Arsène !)



Dernière mise à jour	Informatique	Denis DEFAUCHY
18/11/2021	Bases de la programmation	12 – Fichiers

1.III.2 Lecture des lignes

1.III.2.a Lecture ligne par ligne

On peut alors parcourir chacune des lignes du fichier à l'aide d'une boucle for ainsi :

```
for Ligne in fichier:
```

A la variable `Ligne` est alors associée une ligne du fichier au format str. Cette écriture permet de parcourir toutes les lignes du fichier, les unes après les autres. Attention, `Ligne` n'est pas un nombre, c'est le contenu d'une ligne du fichier !

Un retour à la ligne est spécifié en fin de ligne par "`\n`" (visible en affichant la liste des lignes par la méthode `readlines` vue plus bas). Ainsi, on peut tester la présence d'une ligne vide (hormis la dernière) à l'aide de la commande : « `if Ligne == '\n':` ». Cela peut permettre de mettre fin à une lecture de fichier dont la fin contient plusieurs lignes vides par exemple. Pour la dernière, on peut écrire : « `if Ligne == '':` ».

Si besoin, on peut ajouter un compteur afin d'identifier les numéros de lignes pour n'en traiter qu'une partie.

ATTENTION : lorsque l'on parcourt le fichier avec `for Ligne in fichier:`, le fichier a été parcouru et ne peut pas l'être une seconde fois ! Il ne renvoie pas d'erreur pour autant, le for n'exécute juste rien... Si vous voulez parcourir deux fois le fichier (très inutile, mais vous risquez de le faire en TP), il faut ouvrir deux fois le fichier... Sinon, aucune erreur n'est affichée, mais le fichier n'est pas relu !

Dernière mise à jour	Informatique	Denis DEFAUCHY
18/11/2021	Bases de la programmation	12 – Fichiers

1.III.2.b Lecture complète

Il est possible de récupérer directement une liste des lignes d'un fichier en écrivant :

```
Liste_Lignes = fichier.readlines()
```

Il faudra toutefois faire le post traitement de chaque ligne par la suite. C'est très pratique car en 3 lignes, on obtient la liste des lignes à post traiter :

```
fichier = open("Essai.txt", 'r')
Lignes = fichier.readlines()
fichier.close()
```

Chaque élément de la liste `Liste_Lignes` est alors une chaîne de caractères contenant une des lignes du texte ouvert. Attention, la fin de chaque ligne, sauf la dernière s'il n'y a pas de retour à la ligne final, contient un `"\n"`.

Exemple : soit le fichier texte suivant :

```
Cours d'informatique
10.2
Ligne finale
```

On remarquera que le fichier se termine après le mot « finale » sans nouvelle ligne vide, c'est-à-dire sans retour à la ligne.

En exécutant le code `readlines()`, on obtient :

```
>>> Liste_Lignes
["Cours d'informatique\n", '10.2\n', 'Ligne finale']
```

Il faut donc savoir enlever le « `\n` ». Deux cas de figure :

- Il n'y a qu'une valeur numérique avec une virgule de type « . » reconnue par python : utiliser `float` qui, en plus de permettre l'utilisation du nombre, supprime le `"\n"` :

```
>>> float(Liste_Lignes[1])
10.2
```

- Soit c'est un nombre avec virgule du type « , » au lieu de « . » (on verra plus tard comment la remplacer par « . »), soit c'est un texte : le « `\n` » est vu comme un seul et unique caractère à la fin de la ligne, il suffit donc de récupérer dans la chaîne de caractères de `n` termes ses `n-1` premiers (on verra là aussi qu'il existe l'option `strip` qui réalise la même chose):

```
>>> Ligne_0 = Liste_Lignes[0]

>>> Ligne_0
"Cours d'informatique\n"

>>> Ligne_0[:len(Ligne_0)-1]
"Cours d'informatique"
```

ATTENTION : lorsque l'on parcourt le fichier avec `fichier.readlines()`, le fichier a été parcouru et ne peut pas l'être une seconde fois ! Il ne renvoie pas d'erreur pour autant, le `for` n'exécute juste rien... Si vous voulez parcourir deux fois le fichier (très inutile, mais vous risquez de le faire en TP), il faut ouvrir deux fois le fichier... Sinon, aucune erreur n'est affichée, mais le fichier n'est pas relu !

Remarque : Après avoir ouvert un fichier sous Python, écrire au singulier `fichier.readline()`, permet de lire une ligne. On peut ainsi aisément ne pas lire la première ligne en écrivant `fichier.readline()` puis `Lignes = fichier.readlines()`

Dernière mise à jour	Informatique	Denis DEFAUCHY
18/11/2021	Bases de la programmation	12 – Fichiers

1.III.3 Découpage des données de chaque ligne

Il faut alors s'adapter au cas de figure afin d'en extraire les données importantes. La fonction utile dans notre cas est la fonction « `.split` ». Voici des exemples d'utilisation :

Ligne	Commande	Résultat
<code>Ligne = "10 20 30"</code>	<code>Ligne.split()</code>	<code>['10', '20', '30']</code>
<code>Ligne = "10;20;30"</code>	<code>Ligne.split(";")</code>	

Sans arguments, il y a un découpage chaque fois qu'il y a un ou plusieurs espaces, ou des tabulations (la tabulation possède une notation : « `\t` », mais vous n'avez pas besoin de le retenir comme `split` l'enlève autant qu'un espace !).

Avec argument, il y a découpage chaque fois que l'argument mis entre guillemets est rencontré.

On obtient alors une liste de strings. On peut alors récupérer les valeurs associées en appelant chaque terme de la liste et en le transformant en entier (`int`) ou flottant (`float`).

1.III.4 Suppression du retour à la ligne

Chaque ligne d'un fichier contient à la fin un retour à la ligne `\n`. La dernière ligne peut d'ailleurs ne pas en avoir...

Ainsi, pour le retirer, deux solutions :

- `Ligne = Ligne[0, len(Ligne)-1]` ou `Ligne[:len(Ligne)-1]` qui ne fonctionnera que si la ligne contient un retour à la ligne (la dernière ligne des fichiers peut ne pas en contenir)
- `Ligne = Ligne.strip()` toujours utilisable ☺

Remarque : quand une donnée numérique est stockée sous forme de chaîne de caractères et que l'on utilise `int()` ou `float()`, le « `\n` » sera automatiquement supprimé :

```
>>> ch= "1\n"
>>> float(ch)
1.0
>>> int(ch)
1
```

1.III.5 Post traitement : transformer des caractères

La dernière chose importante à savoir faire est la transformation d'éventuelles virgules en points (la virgule sous python est le point ...). Il suffit d'utiliser la fonction « `.replace()` »

Ligne	Commande	Résultat
<code>Ligne = "1,2;2,3;3,1"</code>	<code>Ligne = Ligne.replace(",",".")</code>	<code>Ligne = "1.2;2.3;3.1"</code>

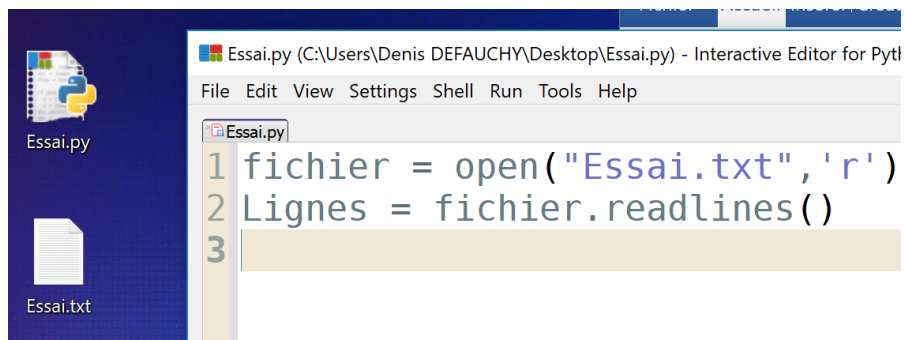
Dernière mise à jour	Informatique	Denis DEFAUCHY
18/11/2021	Bases de la programmation	12 – Fichiers

1.III.6 Fermeture

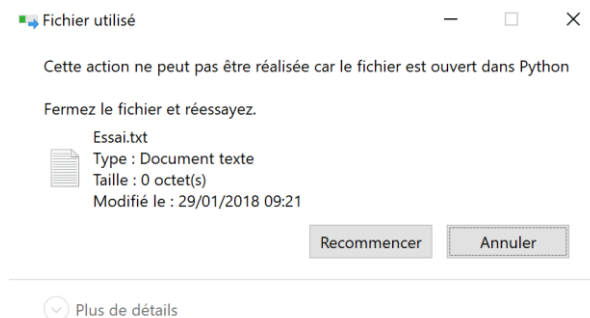
Après avoir lu un fichier texte, il faut le refermer sous Python avec la fonction « `.close` »:

```
fichier.close ()
```

Attention : Lorsqu'un fichier a été ouvert sous Python sans être refermé, en exécutant par exemple le code suivant :



Lorsque l'on essaye de supprimer le fichier créé sur le bureau, une erreur apparaît :



Pour le supprimer, il suffit d'abord, de le fermer en exécutant la commande `close()`, puis de le supprimer.

ATTENTION : j'ai par le passé arrêté Python et redémarré mon ordinateur suite à un problème, sans avoir exécuté la commande `close`... Je n'ai plus jamais réussi à le supprimer, même en essayant de le rouvrir sous python pour le refermer, même en passant par des forums et la commande MSDOS... Donc méfiez-vous !

Dernière mise à jour	Informatique	Denis DEFAUCHY
18/11/2021	Bases de la programmation	12 – Fichiers

1.IV. Création et écriture dans un fichier

1.IV.1 Ouverture

Commençons par ouvrir le fichier, voyons ici deux modes d'ouverture :

Mode ajout : ouvre le fichier et ajoute du texte après ce qui est présent (on ne peut pas lire ce qui est avant !)	Mode écrasement : ouvre le fichier en écrasant son contenu
<code>fichier = open(Nom_Fichier, "a")</code>	<code>fichier = open(Nom_Fichier, "w")</code>

Remarques :

- Si le fichier n'existe pas, il est créé
- Avec ces deux modes d'ouverture, les fichiers ne peuvent être lus. En mode w, c'est de toute manière évident. En mode a, python se place à la fin du fichier pour y ajouter des données...

1.IV.2 Ajout de lignes

Pour ajouter du texte à un fichier texte, il suffit d'utiliser la commande « **.write** »

```
fichier.write(str(Liste[i]))
```

Ceci est un exemple où l'on ajoute le terme d'une liste de nombres. Il faut que l'argument soit une chaîne de caractères.

Si ce que l'on ajoute doit être une ligne, on ajoute "\n") à la fin pour indiquer un renvoi à la ligne :

```
fichier.write(str(Liste[i]) + "\n")
```

Pour ajouter uniquement un renvoi à la ligne, il suffit donc d'écrire :

```
fichier.write("\n")
```

Remarque : pour afficher « \n » dans le texte et donc ne pas créer de retour à la ligne, il faut doubler l'antislash : `fichier.write("\\n")`

1.IV.3 Fermeture

Pour finaliser un fichier texte, il faut le fermer avec la commande « **.close** » :

```
fichier.close()
```

Remarque : un fichier non fermé ne se supprime plus via Windows...