



Fichiers texte

1 Lire et écrire des fichiers texte

La **lecture** des **fichiers texte** est une des premières étapes du **traitement** et de l'**exploitation** des **données expérimentales**. Aussi, on peut conserver des données après l'exécution d'un programme en les écrivant dans un fichier, pour pouvoir par exemple s'en servir lors d'une exécution ultérieure. Nous nous limiterons ici aux **fichiers texte**, lisibles avec un éditeur de texte (bloc note, notepad++,...). Leur extension est souvent `.txt`, mais cette extension n'est pas obligatoire, on retrouve souvent `.csv`, `.json`,...

Définition

Fichier texte

Un fichier texte contient une suite de caractères. Python lit son contenu sous forme de **chaînes de caractères** (`str`), même lorsque ces caractères représentent des nombres.

1.1 Ouvrir un fichier

La fonction intégrée `open()` permet d'ouvrir un fichier via python. Cette fonction reçoit en argument le **chemin du fichier** et un **mode d'ouverture**. Elle renvoie un objet qui permet de lire ou d'écrire dans ce fichier.

```
1 fichier = open("donnees.txt", "r") # ouvre le fichier en lecture
2 contenu = fichier.read()          # lit son contenu (une str)
3 # toutes autres instructions
4 fichier.close()                   # ferme le fichier
```

On distingue le chemin relatif et le chemin absolu.

Chemin relatif : Python cherche le fichier dans le **dossier de travail courant** – qui n'est pas nécessairement le dossier où se trouve le script `.py`. En définissant correctement le dossier de travail courant, cela revient à dire que Python cherche le fichier dans le dossier où se situe le programme lui même.

Chemin absolu : désigne le fichier à partir de la racine du système de fichiers. Le chemin absolu permet d'accéder et d'opérer sur n'importe quel fichier présent dans la mémoire.

Exemple

Le dossier `C:/TP` contient `analyse.py`. L'instruction `open("mesures.txt", "r")` permet d'ouvrir le fichier `C:/TP/mesures.txt`, si `C:/TP` est le dossier de travail courant. L'instruction `open("C:/TP/mesures.txt", "r")` désigne le même fichier par un chemin absolu et ne nécessite pas que `analyse.py` soit dans le même dossier.

Remarque : Il est plus simple d'utiliser le chemin relatif pour éviter les problèmes courants d'OS, de session...

Le tableau 1 présente les principaux modes d'ouverture.

Mode	Utilisation	Si le fichier existe déjà
"r"	Lecture seule ; le fichier doit exister.	Son contenu est conservé.
"w"	Écriture ; crée le fichier si nécessaire.	Son contenu est effacé dès l'ouverture.
"a"	Ajout à la fin ; crée le fichier si nécessaire.	Son contenu est conservé.
"r+"	Lecture et écriture ; le fichier doit exister.	Son contenu n'est pas effacé à l'ouverture.

TABLEAU 1 – Principaux modes d'ouverture d'un fichier texte

Remarque : Il existe un argument optionnel `encoding`, par défaut Python choisit l'encodage de l'environnement. Pour un fichier contenant des accents, il peut être préférable de préciser `encoding="utf-8"` à la lecture *et* à l'écriture.

Question 1 : Imaginons un fichier de donnée issue de manipulations expérimentales ; quel mode choisir pour lire ces données en étant sûr de ne pas les modifier ? Quel mode choisir dans le cas où on souhaite ajouter une nouvelle ligne sans effacer les précédentes ?

Réponse 1 : On utilise respectivement "r" et "a". Le mode "w" effacerait le contenu précédent.

1.2 Fermer automatiquement avec `with`

La méthode `close()` ferme le fichier : elle **libère les ressources** associées et, si le fichier a été ouvert en écriture, termine l'enregistrement des données. Pour ne pas oublier cette étape, en Python on privilégie l'instruction `with` : le fichier est **fermé automatiquement** à la sortie du bloc, même si une erreur survient pendant son utilisation.

```
1 with open("donnees.txt", "r") as fichier: # ouvre en lecture
2     # instructions utilisant le fichier
3
4 # suite des instructions non indentées : le fichier se ferme
```

Dans le code précédent, la variable `fichier` désigne le fichier ouvert dans le bloc indenté.

1.3 Lire le contenu

Un fichier texte est souvent organisé en lignes. Le caractère `\n` marque un saut de ligne, ce qui est souvent synonyme de fin de ligne (*il y a d'autres façon, cf séquences d'échappement*).

L'objet `fichier`

- conserve une **position de lecture**, comparable à un **curseur**. Une lecture commence à cette position, renvoie des caractères, puis fait avancer le curseur ;
- est **itérable** : dans `for ligne in fichier`, ses éléments successifs sont les **lignes du fichier**, chacune reçue dans une variable `ligne` de type `str`. `fichier` n'est pas pour autant une séquence Python : on ne peut pas écrire `fichier[0]` ni `len(fichier)`

Le tableau 2 présente les méthodes les plus couramment utilisées pour lire un fichier texte.

Instruction	Résultat
<code>fichier.read()</code>	Une chaîne contenant tout le fichier.
<code>fichier.read(n)</code>	Une chaîne contenant au plus les <i>n</i> prochains caractères; <code>read(1)</code> lit un seul caractère.
<code>fichier.readline()</code>	Une chaîne contenant la prochaine ligne; une chaîne vide à la fin du fichier.
<code>fichier.readlines()</code>	Une liste contenant les lignes du fichier.
<code>for ligne in fichier:</code>	Parcours des lignes une à une, sans créer d'abord leur liste complète.
<code>fichier.seek(0)</code>	Remplace le curseur au début du fichier.

TABLEAU 2 – Façons courantes de lire un fichier texte

Exemple

Déplacement du curseur

```

1 with open("donnees.txt", "r") as fichier: # curseur au debut
2     debut = fichier.read(3)                # lit 3 caracteres
3     suite = fichier.read(2)                # lit les 2 suivants
4     fichier.seek(0)                       # revient au debut
5     premiere = fichier.readline()         # lit la premiere ligne

```

Après avoir parcouru toutes les lignes, le curseur est à la fin du fichier. La chaîne vide "" renvoyée par `read(1)` ou `readline()` indique que la fin est atteinte.

Question 2 : Si un fichier commence par abcde, déterminer les valeurs des variables `debut` et `suite` dans l'exemple précédent. À quelle position se trouve le curseur juste après ces deux lectures?

Réponse 2 : `debut` contient "abc" et `suite` contient "de". Le curseur se trouve alors après le cinquième caractère.

Pour **exploiter des nombres** lus dans un fichier, on **convertit les chaînes de caractère** avec `int()` ou `float()`.

Il faut parfois traiter la ligne avant de pouvoir faire la conversion. Le tableau 3 présentent quelques méthodes utiles pour cela.

Méthode sur une chaîne	Résultat
<code>ligne.rstrip()</code>	Par défaut, renvoie la ligne sans les espaces ni le saut de ligne situés à sa fin .
<code>ligne.replace(",", ".")</code>	Revoie la ligne dans laquelle chaque virgule est remplacée par un point.
<code>ligne.split(";")</code>	Découpe la ligne au niveau des points-virgules et renvoie une liste contenant les sous-chaînes de caractères.

TABLEAU 3 – Méthodes utiles pour traiter une ligne dans un fichier

Exemple

Lire un nombre par ligne

Si le fichier `mesures.txt` contient une mesure décimale sur chaque ligne, par exemple 12,5 puis 14,0 :

```
1 mesures = [] # liste des resultats
2 with open("mesures.txt", "r") as fichier: # ouvre en lecture
3     for ligne in fichier: # lit une ligne (str)
4         ligne = ligne.rstrip() # retire le saut de ligne
5         ligne = ligne.replace(",", ".") # adapte le separateur decimal
6         mesures.append(float(ligne)) # convertit et ajoute
```

La liste `mesures` contient alors `[12.5, 14.0]` ; le fichier n'a pas été modifié.

1.4 Écrire dans un fichier

La méthode `write()` reçoit une **chaîne de caractères**. Elle n'ajoute pas automatiquement de saut de ligne : on écrit `\n` lorsque l'on veut passer à la ligne suivante.

```
1 with open("mesures.txt", "w") as fichier: # cree ou efface
2     fichier.write("12.5\n") # ecrit la 1re ligne
3     fichier.write(str(14.0) + "\n") # convertit, puis ecrit
```

Pour ajouter une nouvelle mesure au même fichier lors d'une autre exécution, on l'ouvre en mode "a".

Question 3 : Après le programme précédent, quel sera le contenu de `mesures.txt` ? Que se passerait-il si on relançait ce programme sans changer le mode d'ouverture ?

Réponse 3 : Le fichier contient deux lignes : 12.5 puis 14.0. En mode "w", une nouvelle exécution efface d'abord le fichier puis réécrit ces deux lignes : elles ne sont pas ajoutées une seconde fois.

1.5 Enregistrer des données séparées (.csv)

Pour stocker plusieurs valeurs sur une ligne, on choisit un **séparateur**, par exemple une tabulation, de séquence d'échappement `\t`, ou un point-virgule.

Exemple

Deux valeurs sur une ligne

Supposons que la première ligne de `mesures.csv` soit 0;20.5 le temps et la température sont séparés par un point-virgule.

```
1 with open("mesures.csv", "r") as fichier: # ouvre en lecture
2     ligne = fichier.readline() # lit "0;20.5\n"
3     valeurs = ligne.rstrip().split(";") # obtient ["0", "20.5"]
4     temps = float(valeurs[0]) # obtient 0.0
5     temperature = float(valeurs[1]) # obtient 20.5
```

On peut aussi utiliser une affectation élément à élément en faisant attention à la conversion en `float`.