

XIII – Manipulations de fichiers

1/ Gestion de fichiers

On manipule les fichiers à l'aide de leurs noms, qui sont des chemins absolus ou relatifs. Les premières fonctions se trouvent dans la bibliothèque `os` et la dernière dans `shutil`.

<code>listdir('chemin')</code>	renvoie le contenu du répertoire de travail.
<code>getcwd()</code>	renvoie le chemin du répertoire de travail.
<code>chdir('chemin')</code>	change de répertoire de travail.
<code>mkdir('chemin')</code>	crée un nouveau répertoire.
<code>rename('ancien', 'nouveau')</code>	renomme un répertoire ou un fichier.
<code>rmdir('chemin')</code>	supprime un répertoire vide.
<code>remove('chemin')</code>	supprime un fichier.
<code>copy('source', 'destination')</code>	crée une copie d'un fichier

2/ Ouverture et fermeture d'un fichier

Python permet de manipuler les fichiers textes (format `.txt`) avec les instructions suivantes :

<code>f = open(nom_de_fichier, mode)</code>	ouvre le fichier <code>nom_de_fichier</code> ; on se réfère par la suite à cet objet en l'appelant <code>f</code> .
<code>f.close()</code>	ferme le fichier et enregistre les modifications.

- `nom_de_fichier` est une chaîne de caractères donnant le chemin d'accès et se terminant par `.txt`. Attention à bien utiliser des `'/'` comme dans `"C:/machin/truc.txt"`.
- `mode` est une chaîne de caractères déterminant le mode d'utilisation du fichier :

<code>mode = 'r'</code>	lecture (le fichier doit déjà exister).
<code>mode = 'w'</code>	écriture (le fichier est créé s'il n'existe pas, écrasé sinon).
<code>mode = 'a'</code>	addition (on ajoute du contenu à la fin d'un fichier existant).

Remarque : pour prévenir les oublis éventuels, on recommande d'utiliser l'instruction

```
with open(nom_de_fichier, mode) as f:
```

suivie d'une indentation, ce qui garantit la fermeture automatique du fichier.

Exemple : les deux procédés ci-dessous ont exactement le même effet

<code>f=open("mon_fichier.txt", 'w')</code> Instructions d'écriture <code>f.close()</code>	<code>with open("mon_fichier.txt", 'w') as f:</code> Instructions d'écriture
--	---

3/ Écriture dans un fichier

C'est très simple, il n'y a qu'une commande :

<code>f.write(chaine)</code>	ajoute la chaîne de caractères <code>chaine</code> au fichier <code>f</code> .
------------------------------	--

Cette instruction ajoute du contenu au fichier de manière séquentielle. De ce fait, il faudra penser à placer des `'\n'` pour sauter des lignes.

Remarque : cette commande s'utilise aussi bien en mode écriture qu'en mode addition.

Exemple : créer de la sorte un joli fichier texte à visualiser dans l'explorateur.

4/ Lecture d'un fichier

4.1) Lecture séquentielle

On dispose de plusieurs commandes pour lire un fichier `f` :

<code>f.read()</code>	renvoie le fichier sous forme de chaîne de caractères.
<code>f.read(x)</code>	renvoie <code>x</code> caractères successifs du fichier.
<code>f.readline()</code>	renvoie une ligne complète de <code>f</code> .
<code>list(f)</code>	renvoie la liste des lignes du fichier.

Remarque : Python reprend toujours la lecture à l'endroit où elle s'est arrêtée. Une fois le fichier lu dans son intégralité, il faut le rouvrir si l'on veut recommencer la lecture. Ainsi :

- ❶ en réitérant les instructions `f.read(x)` ou `f.readline()`, on lit le fichier par paquets de `x` caractères ou bien ligne par ligne ;
- ❷ après un `f.read()`, un second `f.read()` n'aura aucun effet.

Exemple : jouer avec un fichier texte existant pour tester ces différentes commandes.

4.2) Lecture par itération

Les instructions `f.read()` ou `list(f)` surchargent la machine si le fichier `f` est gros. Quant à `f.read(x)` et `f.readline()`, elles nécessitent une boucle avec un test d'arrêt, ce qui n'est pas très pratique.

Python soit loué, il y a plus simple : `f` est un objet itérable constitué d'une séquence de chaînes de caractères, ses lignes, que l'on parcourt à l'aide d'une boucle `for`.

```
with open("mon_fichier.txt", 'r') as f:
    for ligne in f:
        print(ligne)
```

Exemple : on peut afficher un texte `poeme.txt` ligne par ligne, en comptant les lignes.

Lorsque l'on manipule ainsi le fichier, on peut couper les lignes en mots pour les compter, et même se débarrasser des '\n' intempestifs pour compter les caractères. Soit `s` une chaîne :

<code>s.strip()</code>	supprime les espaces et tabulations en début et fin de chaîne.
<code>s.split(mot)</code>	la liste de chaînes obtenues en coupant <code>s</code> au niveau du séparateur <code>mot</code> , qui disparaît alors.
<code>s.split()</code>	idem en coupant la chaîne au niveau des espaces.
<code>mot.join(l)</code>	convertit une liste <code>l</code> de chaînes en une chaîne, en les joignant à l'aide du séparateur <code>mot</code> .

Prenons la chaîne `s="Des instants distants"`. En tapant `'inc'.join(s.split('an'))`, on obtient `"Des instincts distincts"`.

Exemple : on peut compter les mots d'un fichier à l'aide de l'instruction `f.read().split()`.

Si l'on ne se soucie pas de la ponctuation, ça ne fonctionne pas. On remédie à cela en remplaçant d'abord tous les symboles de ponctuation de la chaîne `f.read()` par des espaces grâce à la fonction suivante :

```
def misenforme(texte):
    t=texte
    for c in ",.'-:;":
        t=" ".join(t.split(c))
    return t
```

4.3) Extraction de données

Les données sont en général stockées sous une forme déterminée dans les fichiers textes. En connaissant la forme utilisée, on peut alors récupérer et exploiter ces données.

Exemple : on dispose d'un fichier texte `notes.txt` contenant les listes des notes obtenues par un certain nombre de personnes. On suppose que chaque ligne est de la forme

`"nom_personne*note_1*note_2*note_3*..."`

On veut afficher, pour chaque personne, son nom, la liste de ses notes ainsi que la moyenne de celles-ci. Pour cela, on va :

- ❶ Créer une fonction `Moyenne` qui renvoie la moyenne d'une liste de réels.
- ❷ Créer une fonction `Extraction` qui prend en argument une ligne et renvoie sous forme de couple le nom et la liste des notes. Attention, celles-ci sont des chaînes de caractères et il ne faudra pas oublier de les convertir en nombres.
- ❸ Utiliser ces deux fonctions pour obtenir le résultat voulu.

Exercices

Exercice 1.

Créer une fonction `Ecriture` qui prend en entrée le nom d'un fichier et écrit dans ce fichier ce qui est entré au clavier, jusqu'à ce qu'une ligne vide soit saisie.

- ❶ Dans un premier temps, on mettra « bêtement » les entrées à la suite les unes des autres.
- ❷ Dans un second temps, on changera automatiquement de ligne à chaque entrée.

Exercice 2.

On dispose en mémoire d'une chaîne de caractères `s`. Supprimer les espaces puis inverser l'ordre des caractères de `s` afin de voir si c'est un palindrome.

Exercice 3.

Créer une fonction `Analyse(phrase)` qui prend en argument une phrase (sous forme de chaîne de caractères) et qui compte le nombre de mots, puis le nombre de voyelles.

Exercice 4.

Écrire une fonction `Stockage_Donnees` qui prend en entrée le nom d'un fichier et écrit dans ce fichier ce qui est entré au clavier, en respectant les contraintes suivantes :

- ❶ les entrées successives sont séparées par `*` dans le fichier ;
- ❷ si l'on saisit un espace, on change de ligne ;
- ❸ si l'on saisit une ligne vide, on ferme le fichier.

Par exemple, le fichier `notes.txt` a été créé avec cette fonction.

Exercice 5.

Écrire, dans les lignes d'un fichier, les entiers de 1 à 20 suivis d'un espace et de leurs carrés.

Exercice 6.

Reprendre le fichier précédent, en remplaçant les espaces par des virgules. Pour cela :

- 1) on crée un nouveau fichier contenant les données sous le bon format ;
- 2) on efface l'ancien fichier ;
- 3) on renomme le nouveau fichier comme l'ancien.

Exercice 7.

Procéder de même avec le fichier `poeme.txt` de manière à ce que :

- 1) chaque ligne soit de la forme `numero_ligne:nombre_mots_ligne:texte` ;
- 2) la dernière ligne soit `nombre_lignes:nombre_mots`.

Exercice 8.

Créer une fonction `Lecture` qui prend en entrée le nom d'un fichier et affiche son contenu ligne par ligne, puis donne le nombre de lignes et le nombre de caractères.