TP N°14 : Les fichiers

Objectif : Apprendre à lire et écrire des fichiers texte via Python.
Avant de commencer : Créez un dossier *TP_14* dans « *D :\votre_classe\votre_nom_prenom* » dans lequel vous placerez tous vos scripts et les fichiers que vous créerez durant ce TP. Créer dans ce dossier le fichier texte nommé « test.txt » contenant quelques lignes de texte .

1 Définition

Un *ficher* est généralement le moyen utilisé pour sauvegarder les données de vos programmes pour les réutiliser plus tard. Python permet d'interagir avec des fichiers (écriture, lecture). On s'intéresse aujourd'hui aux fichiers contenant du texte.

2 Lecture de fichier

En programmation, la lecture d'un fichier se fait en trois temps :

L'ouverture du fichier : se réalise grâce à la fonction *open* qui a deux paramètres : le nom de fichier et le mode de traitement du fichier ("r" pour la lecture d'un fichier).

La lecture du contenu du fichier : L'intégralité d'un fichier peut être récupérée dans une unique chaîne de caractères grâce à la méthode *read*.

La fermeture du fichier : s'effectue grâce à la méthode *close*. Elle peut s'effectuer même si le fichier n'a pas été lu.

Exemple :

```
f = open("test.txt","r") # ouverture
texte = f.read() # récupération du texte dans une variable
print(texte) # affichage du texte récupéré
f.close() # fermeture
```

Remarques :

- La fonction `open` retourne un *descripteur* de fichiers qui est un objet particulier. Ce descripteur doit être récupéré lors de l'appel (par exemple dans une variable `f`) pour pouvoir être manipulé par la suite.
- Une seconde possibilité de lecture consiste à indiquer en paramètre de la fonction `read()` le nombre de caractères qu'on désire lire. Exemple : `ch = f.read(5)`
- Deux autres méthodes permettant de lire un fichier texte ligne par ligne : `readline()` et `readlines()`.

Exercice 1

1. Que renvoie la méthode `read()` lorsque la fin du fichier est atteinte ?
2. Décrire le fonctionnement des commandes `f.readline()`, `readline(n)` et `readlines()` ?
3. Écrivez une fonction `lecture` qui lit le fichier « test.txt » ligne par ligne et affiche son contenu.
4. Chaque ligne de texte se termine par la séquence de caractères `\n` qui s'appelle *séquence d'échappement* et désigne un passage à la ligne. Adapter la fonction `lecture` pour ne pas afficher ces séquences `\n`.
5. Modifiez la fonction `lecture` pour afficher le contenu d'un fichier dont le nom est donné en paramètre. Validez cette nouvelle fonction avec le fichier « test.txt ».
6. Essayez cette fonction avec un autre fichier texte.
7. Essayez cette fonction avec un nom de fichier qui n'existe pas. Modifiez la fonction `lecture` de manière à afficher "Le fichier <nom du fichier> n'existe pas".

3 Écriture de fichier

Comme pour la lecture, l'écriture d'un fichier se fait en trois temps :

L'ouverture du fichier : la valeur du paramètre "mode d'ouverture" est "w" ou "a" au lieu de "r".

L'écriture du contenu du fichier : se fait via la méthode `write()` en mettant en paramètre la chaîne de caractères à écrire dans le fichier.

La fermeture du fichier : est obligatoire pour enregistrer les modifications.

```
f␣=␣open("test_2.txt","w")␣#␣ouverture
f.write("Bonjour␣")
f.write("Ahmed\n")␣#␣première␣ligne
f.write("tu␣apprends␣Python␣!")␣#␣deuxième␣ligne
f.close()␣#␣fermeture␣et␣enregistrement
```

Remarques :

- Le mode d'ouverture "w" pour "write" permet d'écrire dans un fichier. Avec ce mode d'ouverture, si le fichier n'existe pas il sera créé, et s'il existe son ancien contenu sera écrasé avant de commencer l'écriture.
- Le mode d'ouverture "a" pour "append" permet d'ajouter des informations dans un fichier. Si le fichier n'existe pas, il sera créé, et s'il existe l'écriture commence juste à la fin du fichier.
- L'écriture du fichier n'est pas faite au moment de l'usage de la fonction `write()` mais au moment de la fermeture avec `close()`.

Exercice 2

1. Écrivez une fonction `ecriture` qui écrit dans un fichier ce qui est saisi au clavier jusqu'à ce qu'une ligne vide soit saisie.
2. Testez votre fonction et observez en particulier que toutes les lignes saisies sont mises bout à bout dans le fichier.
3. Améliorer votre fonction `ecriture` de manière à ce que chaque ligne saisie soit sur une ligne dans le fichier écrit.
4. Écrivez une fonction `ajout` qui Ajoute quelques lignes dans un fichier (vérifiez que cela est bien réalisé).

4 Compléments : gestion de chemins d'accès

Un fichier n'est pas nécessairement dans le dossier de travail. Il peut s'avérer nécessaire de préciser sa localisation. Les chemins *absolus* et *relatifs* sont deux moyens de décrire le chemin menant à des fichiers ou répertoires.

Une première solution consiste à spécifier le *chemin absolu* (chemin précis) du fichier qu'on désire manipuler. Pour ce la, on décrit la suite des répertoires menant au fichier à partir du nom de volume (C :, D :).

```
f = open("D:/votre_classe/votre_nom_prenom/tp_14/test.txt","r")#1 slash «/» ou 2 antislashs «\\»
```

Une deuxième solution consiste savoir dans quel dossier travaille l'interpréteur. Au lancement de l'interpréteur, le *répertoire de travail courant* est celui dans lequel se trouve l'exécutable de l'interpréteur ou celui d'où vous lancez votre programme.

```
import os                # os : operating system
print(os.getcwd())       # Afficher le répertoire courant (cwd : Current Working Directory)
```

Cela peut permettre d'utiliser des *chemins relatifs* (relatifs à l'endroit où travaille Python). Donc il est possible de changer de répertoire de travail de l'interpréteur.

```
os.chdir("D:/votre_classe/votre_nom_prenom/tp_14") # chdir : change directory
f = open("test.txt","r") # accéder directement à votre fichier
```

Exercice 3

1. Écrivez une fonction `copier(fichier)` qui recopie un fichier texte. Le programme demandera à l'utilisateur le nom (avec chemin d'accès) de fichier et le recopier dans le même emplacement.
2. Écrire une fonction `comparer` qui compare les contenus de deux fichiers et signale la première différence rencontrée.
3. A partir de deux fichiers préexistants A et B, construire un fichier C qui contienne alternativement un élément de A, un élément de B, un élément de A, et ainsi de suite, jusqu'à atteindre la fin de l'un des deux fichiers originaux. Compléter ensuite C avec les éléments restants sur l'autre.

Exercice 4

Écrire un script python qui permet de définir et d'appeler les fonctions suivantes :

1. Créer une fonction `saisie_listes(n,m)` qui saisie et retourne n listes chacune de m entiers positifs. Tester cette fonction pour n=10 et m=10.
2. Créer une fonction `remplir(fichier, listes)` qui enregistre les listes dans un fichier texte dont le chemin est passé en paramètres. Tester cette fonction pour enregistrer les listes crées dans la question 1 dans un fichier nommé «fichier1.txt».
   Exemple :
   La liste `[[12, 5], [23, 4], [25, 1]]` sera enregistrée dans le fichier text sous le format suivant :
   12,5
   23,4
   25,1
3. Créer une fonction `trier_listes_fichier(fichier_source, fichier_distination)` qui permet de lire les listes d'entiers à partir de `fichier_source`, trier les entiers de chaque liste dans l'ordre croissant et enregistrer les résultats dans le `fichier_distination`. Tester cette fonction avec
   `fichier_source` : «fichier1.txt» et `fichier_distination` : «fichier2.txt».
4. Créer une fonction `trier_lignes_fichier(fichier_source, fichier_distination)` qui permet de lire les listes d'entiers à partir de `fichier_source`, trier les listes d'entiers dans l'ordre croissant selon le premier entier de chaque liste et enregistrer les résultats dans le `fichier_distination`. Tester cette fonction avec `fichier_source` : «fichier2.txt» et `fichier_distination` : «fichier3.txt».
5. Créer une fonction `recherche(n,fichier)` qui retourne les numéros de lignes et numéros de colonnes de chaque occurrence de l'entier n dans le fichier passé en paramètres ou bien elle retourne `None` si l'entier n'existe pas. Tester cette fonction sur le fichier «fichier3.txt» pour un entier n saisi au clavier.

Exercice 5

1. Écrivez un script qui cherche un mot dans un fichier texte et affiche, pour chaque occurrence trouvée, le numéro de la ligne contenant ce mot et la position du mot dans cette ligne. Le programme demandera à l'utilisateur le nom (avec chemin d'accès) de fichier et le mot à chercher.
2. Modifier le script précédent pour qu'il puisse chercher un mot dans une liste de fichiers d'un répertoire donné. Le programme demandera le nom du répertoire (avec chemin d'accès), le mot à chercher et affichera dans ce cas : le nom du fichier, le numéro de ligne et la position de chaque occurrence du mot cherché.
3. Modifier le script précédent pour qu'il puisse chercher un mot dans une liste de fichiers d'un répertoire donné et récursivement dans les fichiers de ses sous-répertoires.