TP : manipulations de fichiers

Les résultats obtenus dans un interpréteur Python peuvent être parfois trop nombreux pour être copiés et traités à la main ; si on ferme l'interpréteur, on perd ces informations : c'est le problème de la persistance des données. Une méthode efficace est de travailler avec des données stockées dans un **fichier**. On se contentera dans ce TP de gérer des fichiers textes (extension .txt ou pas d'extension du tout ; sous sous Windows, cela correspond en gros aux fichiers direct
ement lisibles à l'aide du Bloc-Notes)

1. Chemin d'accès

Un fichier se compose de données enregistrées sur un support physique (disque dur ou un autre périphérique de stockage : clé USB...). On accède à un ficher grâce à son nom ainsi que son **chemin d'accès**. Le nom du fichier peut être fourni en **absolu** (avec le **chemin complet**) ou en **relatif** : son emplacement dépend alors de l'endroit dans l'arborescence où est exécuté le programme.

Le module **os** contient des fonctions permettant de dialoguer avec le système d'exploitation (quel qu'il soit). Les fonctions que nous utiliserons sont les suivantes :

- **getcwd**( ) : permet de connaître le répertoire courant ;
- **chdir**( ) : permet de modifier le répertoire courant ; elle prend en entrée la chaîne de caractères donnant un chemin d'accès.

Par défaut, Python travaille dans le répertoire **courant**. On peut lui imposer de travailler dans un autre répertoire ainsi :

```
>>> import os as os
>>> os.getcwd( )
'C :\Python32'
>>> os.chdir('C :\ cpge \ informatique')
>>> os.getcwd( )
'C :\ cpge \ informatique'
```

Application 1 :

1. Ouvrir l'interpréteur de Python puis l'interroger pour trouver le chemin d'accès par défaut.
2. Créer un **répertoire Python** dans le bureau ;
3. Dans le répertoire **Python** créer un sous-répertoire **tp5_fichiers**.
4. Trouver le **chemin d'accès absolu** à ce répertoire.

5. Reprendre l'interpréteur Python, puis modifier le chemin d'accès par défaut pour le faire pointer vers le répertoire tp5_fichiers.
6. Vérifier que cela a fonctionné avec la fonction **getcwd**().

2. Ouverture d'un fichier

De nombreux modes d'ouverture d'un fichier sont possibles (**r** (pour read) , **w** (pour write), **a** (pour append)). Si l'ouverture a réussi Python retourne un objet correspondant au fichier existant. À la fin du travail, on doit fermer l'accès au fichier à l'aide de la méthode .**close**( ).

Application 2 :

```
>>> fich = open('test.txt','w')
>>> fich.close()
```

Vérifier que le fichier **test.txt** a été créé dans le répertoire courant.

3. Lecture d'un fichier

Pour un fichier ouvert en **lecture**, on dispose des méthodes suivantes :

- **.read()** qui récupère les données présentes dans le fichier sous forme d'une **chaîne de caractères.**
- **.readline()** qui lit une seule ligne à la fois. Chaque appel de la méthode retourne la ligne suivante jusqu'à la fin du fichier où la méthode retourne un caractère vide.
- **.readlines()** qui retourne une **liste** dont chaque élément est **une ligne du fichier**.

Application 3 :

Écrire quelques lignes dans le fichier **test.txt** et le lire à l'aide de l'interpréteur Python.

```
>>> fich = open('test.txt','r')
>>> texte=fich.read( )
>>> print(texte)
CPGE Agadir.
Cours Informatique
Classe MPSI2.
>>> fich.close()
```

Tester également la méthode **readlines**.

Application 4 :

Ecrire une fonction qui prend en argument un nom de fichier texte (chaîne de caractères) et qui renvoie le nombre de lignes et le nombre de caractères du fichier.

4. Ecriture dans un fichier

Pour un fichier ouvert en écriture, on dispose des méthodes suivantes :

- **.write()** qui écrit le paramètre passé en argument.
- **.writelines()** qui écrit séquentiellement dans le fichier les éléments de la liste passée en paramètre.

```
>>>fichier=open('test.txt','w')          # mode "w" (écriture)
>>>fichier.write("bonjour a tous")
>>>fichier.close()

>>>fichier=open('test.txt','a')           # mode "a" (ajout)
>>>fichier.write(" et a toutes \n Bienvenue dans Python")
>>>fichier.close()

>>>fichier=open('test.txt','r')           # mode "r" (lecture)
>>>fichier.readlines()
['bonjour a tous et a toutes \n', ' Bienvenue dans Python']
>>>fichier.close()
```

Application 5 :

Créer un sous répertoire **tablesMultiplication** dans lequel vous créer les fichiers textes **table_de_x.txt** où **x** est un entier compris entre **1** et **20**. Chaque fichier **table_de_x.txt** est composé de **100** lignes de texte : **k fois x égal k * x** (où k varie entre 1 et 100)

5. Exercices divers sur les fichiers

Exercice 1 : **Fichier de nombres**

(1) **Lecture d'un fichier de nombres**

Cet exercice traite un problème extrêmement fréquent dans les applications. Nous supposons que les lignes du fichier n'ont pas une structure fixe mais qu'elles sont « propres » : il n'y a que des nombres et des caractères blancs (espaces, tabulations et fins de ligne) qui les séparent, comme ceci :

2.5 3 5.25 8
-0.5

9 8.2 7.95 4.3 4.25 4.1

etc.

Écrire une fonction qui prend en argument un fichier **fichierNombres.txt** et qui renvoie la liste de ces nombres.

(2) **Production d'un fichier de nombres**

Écrire une fonction qui reçoit un nom de fichier et trois séquences X, Y , Z (supposées de même longueur) de nombres flottants et produit un fichier de texte où chaque ligne contient un entier **i** et un **triplet (X[i] , Y [i] , Z[i])** présentés de la manière suivante :

0001 (5.148, 12.000, -8.100)
0002 (21.739, 4.640, 0.000)

Exercice 2 : **les fichiers .csv**

Les fichiers **.csv** (comma separated values) sont des fichiers **textes**. Leur particularité est la présence d'un caractère séparateur (la virgule ou le point virgule). On représente souvent ce type de fichier dans un tableau de la façon suivante : chaque ligne correspond à une ligne du tableau et le séparateur correspond aux séparations entre les colonnes. Ainsi, un fichier **.csv** peut être visualisé au choix avec un éditeur de texte ou un tableur.

Par exemple le fichier **eleves.txt** qui contient:

Prénom ; **Taille** ; **Poids** ;
Yassine ; 169 ; 70 ;
Mehdi ; 166 ; 62 ;
Ilyass ; 171 ; 69 ;

se représente dans un tableur ainsi :

Prénom	Taille	Poids
Yassine	169	70
Mehdi	166	62
Ilyass	171	69

Remarque :

- On utilise sur les fichiers **.csv** (qui sont donc des fichiers **textes**) les mêmes commandes que pour les fichiers texte.

Ecrire une fonction qui lit le fichier **eleves.txt** (de l'exemple précèdent) et qui renvoie le **dictionnaire eleves** : {**Prenom : (Taille, Poids)**}.

Par exemple : { "Yassine" : (169,70), "Mehdi" : (166,62), "Ilyass" : (171,69)}

Pour cela :

(1) Lire le fichier ligne par ligne, et pour chaque ligne,

(2) la nettoyer de ses caractères espaces et ' **;**' en début et fin de ligne (méthode **str.strip( )**)

(3) la découper en une liste de trois mots [Prenom, Taille, Poids] (méthode **str.split( )**)

(**4**) créer l'entrée correspondante dans le dictionnaire.

Exercice 3 : Opérations sur les fichiers

(**1**) Écrire une fonction qui prend en argument deux fichiers **fichier1** et **fichier2** et qui crée une copie du **fichier1** dans **fichier2** où tous les caractères ont été mis en majuscules.
(**2**) Écrire une fonction qui prend en argument deux fichiers **fichier1** et **fichier2** et qui écrit à la fin de **fichier1** le contenu de **fichier2**
(**3**) Écrire une fonction qui prend en argument un fichier **fichier1** et qui renvoie un nouveau fichier texte ne contenant que les lignes du fichier **fichier1** qui commencent par un 'e'.
(**4**) Ecrire une fonction qui prend en argument un fichier texte **fichier1** et un caractère **c** et qui crée un fichier contenant le contenu de **fichier1** dans lequel tous les **espaces** sont remplacés par le caractère **c**.
On pourra utiliser **join**.
(**5**) Écrire une fonction qui affiche la phrase la plus longue d'un fichier donné en argument.
(**6**) Écrire une fonction qui recopie le texte d'un fichier **fichier1** dans un nouveau fichier **fichier2**, mais formaté de façon à contenir exactement 50 caractères par ligne.