TP 5. Manipuler des 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. C'est ce que nous allons apprendre à faire dans ce TP.

1 Travailler avec des fichiers

Un fichier se compose de données enregistrées sur un support physique (disque dur ou un autre périphérique de stockage : clé USB...). Il permet d'assurer la persistance des informations. On accède à un ficher grâce à son nom ainsi que son chemin d'accès.

Dans ce TP nous travaillons avec des *fichiers textes*.

- La fonction Python `open(nom,mode)` permet d'ouvrir un fichier.
- 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.

1.1 Chemin d'accès

Point méthode.
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 chaine de caractères donnant un chemin d'accès.

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

Exercice 1. Applications.

1. Ouvrir IDLE puis l'interroger pour trouver le chemin d'accès par défaut.
   Le chemin est .....
2. **Modifier un chemin d'accès.**
   (a) Dans votre espace personnel, dans le répertoire dédié à l'informatique, créer un sous-répertoire `tp5_fichiers`.
   (b) Trouver le **chemin d'accès absolu** à ce répertoire.
   (c) Reprendre IDLE puis modifier le chemin d'accès par défaut pour le faire pointer vers le répertoire `tp5_fichiers`.
   (d) Vérifier que cela a fonctionné avec la fonction `getcwd()`.

1.2 Ouverture d'un fichier

De nombreux modes d'ouverture sont possibles :

- `r` (pour *read*) : ouverture en lecture seule ; retourne un erreur si le fichier n'existe pas.
- `w` (pour *write*) : ouverture en écriture ; crée le fichier s'il n'existe pas.
- `rw` : pour la lecture-écriture.
- `a` (pour *append*) :pour écrire à la fin du fichier exisant ou création s'il n'existe pas.

Si l'ouverture a réussi Python retourne un objet correspondant au fichier existant. Tout comme n'importe quel autre objet Pyhton on peut définir des noms de variable qui pointent vers cet objet et travailler avec les variables.

À la fin du travail, on doit fermer l'accès au fichier à l'aide de la méthode `.close()`.

Exercice 2. Taper les instructions suivantes :

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

Vérifier que le fichier `fichierTest.txt` a été créé dans le repertoire courant.

1.3 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 chaine 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.

Exercice 3. Écire quelques lignes dans le fichier `fichierTest.txt` et le lire à l'aide de l'interpréteur Python. Dans la pratique vous devez obtenir à peu près ça :

```
>>> fich = open('fichierTest.txt','r')
>>> texte=fich.read()
>>> print(texte)
Bonne année 2014 à tout le monde.
Mes résolutions :
-> commencer à fumer ;
-> arrêter le sport.
>>> fich.close()
```

Tester également la méthode `readlines`.

Les méthodes `read` et `readlines` permettent de lire l'intégralité du fichier en une seule instruction. Ainsi le fichier est intégralement copié dans la mémoire vive de l'ordinateur. La méthode est donc mise en défaut pour de trop gros fichiers.

Exercice 4. Lecture ligne par ligne.
Dans votre répertoire `tp5_fichiers` enregistrer un nouveau fichier Python `fonctionsFichiers.py`. Dans ce même répertoire copier le fichier texte `histoirePython.txt` disponible dans le répertoire `infoPCSI\tp5` du lecteur `P:`.

1. Définir une fonction Python `compteLigne` qui prend un nom de fichier (chaine de caractères) en argument et retourne le nombre de ligne du fichier (en comptant les sauts de lignes). **Une méthode :** on lit successivement les lignes du fichier avec la méthode `readline` **tant qu**'on n'est pas passé sur une ligne vide ; un compteur s'incrémente à chaque passage ; l'ouverture et la fermeture du fichier peuvent être inclus dans le corps de la fonction.
2. Tester votre fonction sur les fichiers `fichierTest.txt` et `histoirePython.txt`.

1.4 Écrire 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.

Exercice 5. Taper les lignes de commande ci-contre et vérifier le fichier créé.

```
>>> fich = open('fichierTest2.txt','w')
>>> fich.write('ligne1\nligne2')
>>> fich.writelines(['\nla ligne3', 'et le point final.'])
>>> fich.close()
```

Exercice 6. 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 100. Chaque fichier `table_de_x.txt` est composé de 100 lignes de texte :

$$k \text{ fois } x \text{ égal } k * x$$

où k varie entre 1 et 100.