Les fichiers

I. Ouverture d'un fichier

En langage Python, les fonctions de gestion de fichiers sont directement accessibles.
La première chose à faire est d'utiliser la fonction **`open()`** intégrée de Python pour obtenir un **objet fichier(Pyhon file object)**.
La fonction `open()` ouvre un fichier existant ou le crée s'il n'existe pas.
Ces objets file object, contiennent des méthodes et des attributs pouvant être utilisés pour collecter des informations sur le fichier que vous avez ouvert.

```
open([emplacement/nom du fichier], [mode ouverture], encoding = [mode])
```

[emplacement] est le répertoire où se trouve le fichier.
Si l'emplacement n'est pas précisé, l'interpréteur Python cherchera uniquement à côté du module en cours d'exécution ou dans ses propres répertoires.

[nom du fichier] est le nom du fichier qu'on souhaite ouvrir ou crée.
Le mode d'ouverture comprend les paramètres suivants:

- **Le mode 'r' :** ouverture d'un fichier existant en lecture seule
- **Le mode 'w' :** ouverture en écriture seule, écrasé s'il existe déjà et crée s'il n'existe pas
- **Le mode 'a' :** ouverture et écriture en fin du fichier , pour ajouter du contenu
- **Le mode 'b' :** ouverture en mode binaire

[mode] est le type d'encodage de caractères utilisé pour l'écriture ou la lecture (UTF-8, ASCII, ...)

Un objet fichier crée par la méthode **`open()`**, est doté de certaines propriétés permettant de lire et écrire sur ce dernier.

II. Lecture d'un fichier

Plusieurs méthode sont applicables mais il faut ouvrir le fichier au préalable.

1- Méthode read()

La méthode **`read()`** permet de lire le contenu total ou partiel d'un fichier ouvert.
Elle renvoie une valeur de type **str()**.

Exemple 1 : lecture totale

```
fichier = open("exemple.txt", 'r')        # ouverture du fichier
contenu = fichier.read()                  # lecture du contenu
fichier.close()                           # fermeture du fichier
```

Exemple 2 : lecture des 20 premiers caractères

```
fichier = open("exemple.txt", 'r')        # ouverture
contenu = fichier.read(20)                # lecture
fichier.close()                           # fermeture
```

***Remarque** : Après exécution de la fonction **read(n)** (n = nombre de caractères à lire), le curseur se trouve à la **position n+1**, et donc si on exécute la fonction une **2è fois**, la lecture débutera depuis le **(n+1)è caractère**.*

Il existe une autre méthode, avec le mot-clé **`with`**, qui permet d'ouvrir et de fermer un fichier de manière efficace.
Si pour une raison ou une autre l'ouverture ou la lecture du fichier conduit à une erreur, l'utilisation de **with** garantit la bonne fermeture du fichier, ce qui n'est pas le cas dans le code précédent.

Exemple 3 : lecture des 20 premiers caractères avec with :

```
with open("exemple.txt", 'r') as fichier:
    contenu = fichier.read(20)
```

2- Lecture ligne par ligne avec les méthodes readline() et readlines()

2.1- La méthode readline()

La méthode **`readline()`** permet de lire un fichier ligne par ligne.
Elle renvoie une valeur de type **str()**.
Cette méthode pointe sur la première ligne lors de sa première exécution, ensuite sur la deuxième ligne lors de sa seconde exécution et ainsi à la n-ième exécution, elle pointe vers la n-ième ligne.

Exemple 4 : lecture du fichier ligne par ligne

```
with open("exemple.txt", 'r') as fichier :
    print(fichier.readline())             # affiche la ligne n°1
    print(fichier.readline())             # affiche la ligne n°2
```

En combinant la méthode **readline()** avec une boucle **while,** on peut lire la totalité des lignes d'un fichier.

Exemple 5 : lecture du fichier ligne par ligne avec une boucle

```
with open("exemple.txt", 'r') as fichier :
    contenu = ''
    ligne = fichier.readline()
    while ligne != "" :
        ligne = fichier.readline()
        contenu = contenu + ligne
```

2.2 – La méthode readlines()

La méthode **readlines()**, renvoie une liste dont les éléments sont les lignes du fichier.
Elle renvoie une valeur de list() contenant des valeurs de type str().

Exemple 6 : lecture des lignes du fichier avec readlines()

```
with open("exemple.txt",'r') as fichier :
    contenu = fichier.readlines()
print(contenu[0])                         # impression de la première ligne
print(contenu[1])                         # impression de la deuxième ligne
```

III. Ecriture d'un fichier

Pour écrire dans fichier ouvert en mode écriture, on utilise la méthode **write().**

`fichier.write(contenu)` 'contenu' est une variable de type **str()**

1 – Ecriture dans un fichier existant

Exemple 7: ouvrir le fichier 'exemple.txt' et y ajouter du contenu:

```
with open ("exemple.txt", "a") as fichier :            # conservation du contenu
    fichier.write ("J'écris sans écraser le contenu !")

with open ("exemple.txt", "r") as fichier :      # lire le fichier après l'ajout
    print (fichier.read())
```

Exemple 8 : ouvrir le fichier 'exemple.txt' avec écrasement du contenu existant:

```
with open ("exemple.txt", "w") as fichier :              # écrasement du contenu
    fichier.write ("J'écris en écrasant le contenu !")

with open ("exemple.txt", "r") as fichier :      # lire le fichier après l'ajout
    print (fichier.read())
```

2 – Ajouter des lignes à un fichier en Python avec la méthode writelines()

La méthode **`writelines()`**, permet d'ajouter une liste de chaînes de caractères (une liste de lignes) à un fichier ouvert en mode écriture

Exemple 9 : ajouter une liste des lignes à un fichier

```
with open ("exemple.txt", "w") as fichier
    l = ["ligne1\n","ligne2\n,"ligne3\n"]
    fichier.writelines(l)

with open ("exemple.txt", "r") as fichier :
    print (fichier.read())
```

Le caractère \n est le 'retour chariot'

IV. Exemples de fichiers de données

1 – Fichiers de configuration

Exemple : configuration.ini

```
# paramètres de configuration du serveur
[settings]
host :  127.0.0.1
username :  root
password :  root
```

2 – Fichier au format CSV

Le sigle **CSV** signifie '**comma separated values**' qui veut dire **'valeurs séparées par des virgules'**, qui est défini comme un format de fichier simple qui utilise une structuration spécifique pour organiser les données tabulaires. Il stocke des données tabulaires telles qu'une feuille de calcul ou une base de données en texte brut et a un format commun pour l'échange de données.

```
nom,prenom,date_naissance
Durand,Jean-Pierre,23/05/1985
Dupont,Christophe,15/12/1967
Terta,Henry,12/06/1978
```

Attention : les valeurs ne sont pas toujours séparées par des virgules

tabulation \t
point-virgule ;

3 – Format JSON

JSON (**JavaScript Object Notation**) est un format de données populaire et standard utilisé pour représenter et stocker des données structurées constitués de paires attribut-valeur similaire à un dictionnaire Python. Il est courant de transmettre et de recevoir des données entre un serveur et une application Web au format JSON.
Celui-ci depuis longtemps devenu indépendant du langage et existe comme son propre standard (inutile de passe vons donc heureusement éviter JavaScript pour traiter les fichiers json.

Exemple: fichier json décrivant une personne

```
{"name": "Albert",
"email": "albert99@gmail.com",
"loisirs": ["Sport", "Cinema", "Lecture" , "Voyage"]
}
```

4 – Le format XML :

Le format eXtensible Markup Language (XML) est un format qui utilise des balises pour structurer les données dans le fichier texte. Les balises sont utilisées pour encadrer un contenu : il y a une balise ouvrante et une balise fermante.

Exemple :

```
<data>
        <fruit>
                <name>Banane</name>
                <price>5.99</price>
                <code>77</code>
        </fruit>
        <fruit>
                <name>Pomme</name>
                <price>2.99</price>
                <code>99</code>
        </fruit>
</data>
```