Fichiers externes : fichiers textes

Les programmes informatiques ont souvent pour objet le traitement de données. Pour cela, il faut dans un premier temps récupérer ces données. Elles ne resteront alors externes au programme qui les traite mais localisées dans un dossier de travail commun au programme de traitement.
Dans cette partie, nous ne nous intéresserons qu'à des données contenues dans des fichiers de type « texte » comportant l'extension `.txt` ou `.dat`.

1. Principe

Un fichier « texte » contient simplement des caractères (pas de mise en forme de type gras, italique etc… : ceci correspond à des fichiers de traitement de texte). Le fichier « texte » à traiter doit être placé dans un répertoire (dossier) de travail qui sera commun au programme qui analysera les données.

2. Variable de flux

Une fois votre fichier à traiter copié dans le répertoire de travail dédié à votre étude, il faut récupérer les données qu'il comporte pour les traiter dans votre programme (lui-même présent dans le répertoire de travail donc). Ainsi, pour interagir avec un tel fichier à l'aide d'un code Python, on passe par la création d'une variable particulière nommée « flux d'entrée-sortie » qui contiendra les données du fichier externe. Une fois cette variable de flux, contenant dans notre cas du texte, créée**, *c'est sur celle-ci*** qu'il faudra travailler. ***C'est sur celle-ci (variable flux)*** que les différentes instructions s'appliqueront.

La variable de flux d'entrée-sortie est créée par l'instruction **`open()` :**

```
variable_flux = open("dijkstra.txt",'w', encoding='utf8')
```

Comme on le voit ci-dessus, l'instruction **`open()`** est accompagnée de précision indiquant le mode d'ouverture du texte suivant que l'on veut faire: le lire, l'écraser, y ajouter du texte etc…

- Premier paramètre : Il s'agit du nom du fichier à ouvrir, ***extension comprise***.

ATTENTION ! Le fichier à ouvrir doit absolument se trouver dans le répertoire de travail !

- Deuxième paramètre : Il s'agit du mode d'ouverture.

`'r'`	(**r**ead) Ouverture pour lecture seule (option par défaut). Le ficher doit exister dans le répertoire courant
`'w'`	(**w**rite) Ouverture pour écriture seule. Lorsque le fichier n'existe pas il est créé dans le répertoire courant (ou le répertoire donné); si le fichier existe déjà il est écrasé.
`'a'`	(**a**ppend) Ouverture pour écriture seule. Lorsque le fichier n'existe pas il est créé dans le répertoire courant ; si le fichier existe alors les données sont écrites à la suite.
`'r+'`	Ouverture pour lecture et écriture. Le fichier doit exister, il est alors réactualisé.

- Troisième paramètre : encodage, nécessaire seulement si le texte n'est pas correctement encodé à la lecture.

ATTENTION ! Fermeture nécessaire !

Une fois votre travail sur les données terminé, il faut prendre l'habitude de fermer la variable de travail.

```
variable_flux.close()
```

3. Traitement d'un fichier

Le fichier va être traité en utilisant des instructions appliquées ***sur la variable de flux*** créée à l'ouverture suivant le modèle :

```
variable_flux.instruction(paramètres propres à l'instruction)
```

Voici différentes instructions utilisables sur les fichiers de type « texte » dont vous devez vous rappeler des grands principes et des différences possibles plutôt que de leur spécification exacte.

3.1. Ecriture :

Une fois un fichier ouvert en mode écriture (‘`w'` ou `'a'` en général), on peut le « remplir » avec l’instruction write :

```
flux = open("fichier1.txt",'w')
flux.write("Il est possible d'écrire sur ce fichier nommé fichier1.txt")
flux.close()
```

Le fichier `fichier1.txt` n’existant pas, il est créé dans le répertoire courant.

Attention ! Cette instruction ne rajoute pas de retour à la ligne. Si on veut demander un retour à la ligne, il faut le demander explicitement en écrivant le caractère ' **`\n`** '. Ce caractère apparaît tel quel si on demande à l’interpréteur le contenu de la chaîne de caractères, mais sera interprété correctement par les fonctions **`print`** et **`write.`** Pour information, il existe un caractère similaire pour les tabulations : '**`\t`**'.

3.2. Lecture :

- **Les instructions `read()`, `readline()` et `readlines()`**

Si le fichier est ouvert en mode lecture (‘r’ en général), on peut récupérer son contenu.

	`read()`	**`readline()`**	**`readlines()`**

In[1]	`variable_flux = open("dijkstra.txt",'r')`
	`texte1 = variable_flux.read()`
	`variable_flux.close()`
		`variable_flux = open("dijkstra.txt",'r')`
		`texte2 = variable_flux.readline()`
		`variable_flux.close()`
			`variable_flux = open("dijkstra.txt",'r', encoding = 'utf8')`
			`texte3 = variable_flux.readlines()`
			`variable_flux.close()`

In[2]	`Texte1`	`Texte2`	`Texte3`
Out[2]	`Edsger Wybe Dijkstra: nÃ© Ã Rotterdam le 11 mai 1930 et mort2 Ã Nuenen le 6 aoÃ»t 2002. C'est un mathÃ©maticien et informaticien nÃ©erlandais du XXe siÃ¨cle. Il est Ã l'origine de l'algorithme Ã©ponyme, l'algorithme de Dijkstra, permettant de calculer des plus courts chemins dans un graphe orientÃ©.`	`Edsger Wybe Dijkstra:`	`['Edsger Wybe Dijkstra:\n', 'né à Rotterdam le 11 mai 1930 et mort2 à Nuenen le 6 août 2002.\n', "C'est un mathématicien et informaticien néerlandais du XXe siècle.\n", "Il est à l'origine de l'algorithme éponyme, l'algorithme de Dijkstra, permettant de calculer des plus courts chemins dans un graphe orienté."]`
Action	Cette fonction renvoie tout le contenu du fichier sous forme d’une unique chaîne de caractères	Cette fonction lit la ligne courante, renvoie une chaîne de caractère correspondante, et passe à la ligne suivante au prochain appel.	Cette fonction permet de récupérer immédiatement toutes les lignes dans une liste d’éléments de type `str`, chaque ligne étant un élément finissant par `\n.`

- **L’instruction `split()`**

Cette fonction propre aux objets de type **`str`** fait quelque chose de similaire à **`readlines()`** : au lieu de découper les éléments d’un fichier ligne par ligne, elle découpe les éléments d’une chaîne selon un délimiteur donné (par défaut les espaces) :

In[1]	`texte_a_decouper= "Il y a beaucoup trop de mots ici. Il faudrait les séparer. Ou au moins séparer les phrases!"`	
	`texte_split_sans_parametre = texte_a_decouper.split()`	`texte_split_point = texte_a_decouper.split('.')`
In[2]	`texte_split_sans_parametre`	`texte_split_point`
Out[2]	`['Il', 'y', 'a', 'beaucoup', 'trop', 'de', 'mots', 'ici.', 'Il', 'faudrait', 'les', 'séparer.', 'Ou', 'au', 'moins', 'séparer', 'les', 'phrases!']`	`['Il y a beaucoup trop de mots ici', ' Il faudrait les séparer', ' Ou au moins séparer les phrases!']`

Exercice 1: Ecriture de fichiers textes

1- **Créer** un dossier ITC/DM1/exo1**. Ouvrir** un nouveau script 'exo1', **l'enregistrer** dans le dossier de travail. Sur Spyder **placer** le répertoire de travail courant au niveau de ce dossier.

2- **Création de fichier texte :**

2-a- Importance de la fermeture d'un flux : Sur votre script, **créer** un fichier texte dont le nom sera : « `testa.txt` » par le biais d'une variable de type flux nommée par exemple **`flux_a`** :

```
v_fluxa = open("testa.txt",'w')
v_fluxa.write("j'écris quand je veux")
```

Lancer les instructions puis **observer** le dossier ITC/DM1/exo1. **Tenter** de supprimer le fichier de votre dossier.

Ajouter les instructions de fermeture du flux (`v_fluxa.close()`), **lancer** les instructions puis **supprimer** le fichier.

2-b- Expérience sur le mode `'w'`: Ecrire dans un fichier texte « `testb.txt` » (en passant par le biais d'un flux) la phrase ci-dessous :`"Si j'écris quelque chose, je modifie le fichier dans le répertoire courant"`. **Lancer** les instructions puis **observer** l'explorateur de variables, et les fichiers crées présents dans ITC/DM1/exo1. **Ecraser** le texte du dernier fichier en le remplaçant par celui-ci : « `Et là, j'écrase tout… ».`

2-c- Création de fichier d'après une liste de nombre : `futur_texte = [21, 32, 53]`

- **Créer** un fichier texte dont le nom sera : « `testc.txt` » par une variable de type flux à l'aide de l'instruction **open**, toujours en mode **`'w'`**.

- **Ecrire** dans ce fichier (en passant par le biais du flux) chaque élément de la liste en allant à la ligne à chaque fois. **Fermer le flux. Lancer** les instructions

2-d- L'instruction 'a' : reprendre le fichier testc afin d'y ajouter le nombre 0 sur autant de ligne nécessaire afin que le texte comprenne en tout 10 lignes.

Exercice 2: Lecture de fichiers textes

1- **Créer** un dossier ITC/DM1/exo2. **Copier** le fichier `Bio_Dijkstra.txt` fourni sur Cahier de prépa dans ce même dossier de travail. **Se placer** dans le répertoire courant.

2- **Ouvrir** le fichier à traiter en mode lecture tour à tour à l'aide des instructions suivantes **: `read()`** puis **`readlines() (cf cours).`**

Rq : Si le texte apparait avec des caractère spéciaux, ajouter, après le mode d'ouverture, le code suivant : **`, encoding='utf8'`**.

3- **Choisir** le mode adéquat permettant de créer ensuite une liste `phrases` comprenant chaque phrase du texte initial sans le titre. Pour cela, **utiliser** l'instruction **`split`**.

Exercice 3: Traitement des données d'un fichier texte dans un fichier texte

1- **Créer** un dossier ITC/DM1/exo3. **Y copier** le fichier `moliere.txt` (sur Cahier de prépa). **Se placer** dans le répertoire courant. **Affecter** à une variable `liste_moliere` l'ensemble du texte sous forme de liste des lignes du texte initial et à une variable `texte_moliere` une chaine de caractère contenant tout le texte initial.

2- **Ecrire** une fonction `compter_repliques(fichier_t, personnage)` qui prend en paramètres le nom du fichier à lire et le nom du personnage étudié et compte le nombre de répliques qu'il prononce dans le texte proposé.

3- **Ecrire** une procédure `fichier_repliques(fichier_a_modif, l_noms, l_comptes)` qui prend en paramètres le nom du fichier à modifier, une liste de noms et une liste d'entiers, et écrit pour chaque couple nom/score une ligne dans le fichier sous la forme : « `Manu : 25634` ». **Tester** avec

```
LesFemmesSavantes_pers = ['HENRIETTE','ARMANDE','TRISSOTIN','CHRYSALE']
ComptesActe1Scene1 = [11,10,0, 0]
```

4- **Utiliser** les deux fonctions précédentes pour écrire une fonction `nb_repliques(fichier_texte,l_pers,f_compte)` qui compte l'ensemble des répliques de chacun personnages d'une liste pour un fichier « texte » donné et l'écrit dans un fichier `f_compte`.