TP n° 06 – Import/export de fichiers

Généralités sur les fichiers

Jusqu'à présent, lorsque vous avez traité des données avec un programme écrit en python, elles ont disparu à la fermeture du programme car elles étaient stockées dans la mémoire vive de l'ordinateur (Random Access Memory). Pour conserver des données pour une utilisation ultérieure, il faut les stocker sur des mémoires non volatiles (disque dur, carte mémoire, CD Rom,...). Elles sont stockées sur ces supports dans une zone physique identifiée par le système d'exploitation[1]. La suite de donnée structurée est appelée fichier (ou *file* en anglais).
Il existe deux types de fichier de données : les fichiers de type texte (encore appelé ASCII) et de type binaire.
Les fichiers de type texte sont constitués de caractères codés par un nombre entier, écrit en code binaire sur 8 bits (1 octet), compris entre 0 et 255.
Les 128 premiers caractères sont communs aux différents codes ASCII.
Les 128 caractères suivant permettre de décrire les différents caractères nationaux (exemple : é, è, à,... pour le français). Chaque code ASCII fait l'objet d'une norme ISO. Pour les langues d'Europe de l'ouest, la norme est ISO 8859-1 appelé Latin-1. Le codage est donné dans le tableau .

ISO/CEI 8859-1

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	*positions inutilisées*															
1x																
2x	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8x	*positions inutilisées*															
9x																
Ax	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯
Bx	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

L'encodage universel des caractères est réalisé soit par l'Unicode ou l'UFT-8 (Universal Transformation Format).
Les fichiers de type binaire ne contiennent pas (exclusivement) du texte. Et ils ne peuvent être convenablement traités que par des logiciels spécialisés. Un fichier PDF, une image JPEG ou un mp3 sont quelques exemples de fichiers binaires.
Dans la suite, on n'utilisera que des fichiers de type texte.

Ouvrir/fermer un fichier

On peut ouvrir un fichier grâce à la fonction `open('emplacement/nom')` qui permet d'accéder à un fichier stocké dans l'ordinateur, en spécifiant son emplacement sur le disque et son nom. Pour pouvoir manipuler les données du fichier, il faut l'affecter à une variable, comme n'importe quelle autre structure de données. Par exemple, on peut l'affecter à la variable `fichier` :

1. Aller dans le répertoire « Ressources », puis « PTSI », puis « TP6 ». Copier le fichier « TP6.txt » et coller le dans votre répertoire personnel situé dans « Dossiers personnels/PTSI ».
   Dans votre répertoire personnel, faire un clic-droit sur le fichier. Choisir « propriétés ». Sélectionner et copier le chemin d'accès qui apparaît après « Emplacement ».
2. Dans l'éditeur de Spyder, écrire l'instruction qui ouvre le fichier grâce à la fonction `open` et l'affecter à une variable (par exemple `fichier`). Dans tous les cas, on ferme toujours en fin de procédure le fichier par l'instruction :

1. En anglais « *OS* » (pour *Operating system.*)

Traitement caractère par caractère d'un fichier

Une fois affecté à une variable, un fichier est une structure de données de type `file` qui est itérable et qui peut être parcourue ligne par ligne. De plus, chaque ligne du fichier est une chaîne de caractère (type `str`) qui est elle-même itérable et qui peut être parcourue caractère par caractère. En imbriquant correctement deux boucles itératives on peut donc lire un fichier texte caractère par caractère.

3. Écrire une procédure permettant de parcourir chacun des caractères de chacune des lignes du fichier `TP6.txt` et de les afficher à l'écran.

Attention ! Si vous voulez lire plusieurs fois de suite le même fichier un problème survient : pour python, une fois qu'un fichier a été lu entièrement... il n'y a plus rien à lire ! On dit que python est positionné en fin de fichier. Pour pallier ce problème, on peut :

— fermer le fichier comme il est recommandé de le faire à la fin de chaque procédure ; puis le réouvrir et le réaffecter à une variable si on veut le relire ;
— utiliser la méthode `.seek()` qui permet de positionner python où on veut dans le fichier ; par exemple :
`fihcier.seek(0)` permet de repositionner python en début du fichier `fichier`, comme s'il venait d'être ouvert.

On souhaite maintenant récupérer la totalité des données du fichier dans une liste. La liste souhaitée est donc une liste de listes, chaque sous-liste étant une liste des mots d'une ligne. Dans le cas de notre TP, le résultat attendu est donc :

```
[ ['mot1', 'mot2', 'mot3'], ['mot4', 'mot5', 'mot6'], ['mot7', 'mot8', 'mot9'] ]
```

Dans le fichier `TP6.txt` chaque mot d'une ligne est séparé par une virgule. On dit qu'un tel fichier est un fichier en « *comma separated values* » (ou csv) [2]. D'autre part, pour signifier la fin d'une ligne d'un fichier texte, un caractère spécial (dit caractère d'échappement) est utilisé : `\n`. Cela veut dire que dans le fichier `TP6.txt`, les mots se terminent soit par `','` soit par `'\n'`.

4. À partir du code de la procédure utilisée à la question 3, écrire une procédure permettant de former la liste demandée ci-dessus pour le fichier `TP6.txt`. La logique est la suivante :
   — on lit les caractères un à un et on forme des mots avec ;
   — quand un mot est fini, on l'ajoute à la liste des mots d'une ligne ;
   — quand une ligne est finie, on l'ajoute à la liste des listes des mots de lignes.
5. (Uniquement sous Windows.) Quel est le problème rencontré en fin de fichier ? Le résoudre.

2. Un autre type de format très utilisé est le format « *tab separated values* » (ou tsv) dans lequel chaque mot d'une ligne est séparé par une tabulation.

Lire, écrire

Il est possible d'ouvrir un fichier dans plusieurs *modes* : lire (`'r'`), ajouter à la fin (`'a'`), écrire (`'w'`). Les différents modes déterminent le type d'accès aux données (lecture (R) ou écriture (W)) et si les données éventuellement initialement présentes dans le fichier sont effacées à l'ouverture.

mode	accès	réinitialisation	remarque
r	R	non	le fichier doit exister
r+	R/W	non	le fichier doit exister
a	W	non	le fichier doit exister
a+	W/R	non	le fichier est créé s'il n'existe pas
w	W	oui	le fichier est créé s'il n'existe pas
w+	W/R	oui	le fichier est créé s'il n'existe pas

Par défaut, si on ne précise aucun mode (comme à la section I), c'est le mode `'r'` qui est utilisé.
Par exemple, pour ouvrir un fichier en lecture/écriture sans réinitialiser le fichier on utilise :
Par exemple, pour ouvrir un fichier en écriture uniquement en réinitialisant le fichier on utilise :
On évite absolument d'ouvrir en écriture un fichier qui contient des données que l'on souhaite conserver[3]**.**
Pour écrire une chaîne de caractère dans le fichier on peut utiliser la méthode `.write` :
`fichier.write(s)` écrit la chaîne de caractères `s` dans le fichier `fichier`.

6. À partir du code de la procédure utilisée à la question précédente, écrire une procédure permettant de réécrire un nouveau fichier nommé « TP6tab.txt » dans lequel toutes les virgules sont remplacées par des tabulations. (Les tabulations sont codées par le caractère spécial `\t`.)

Utilisation des méthodes Python

La lecture itérative caractère par caractère de la section II peut avantageusement être remplacée par les méthodes python associées aux fichiers. Les principales méthodes de la classe `file` sont :

- `.read()` retourne tout le fichier comme une chaîne de caractères,
- `.read(n)` retourne une chaîne de caractère d'au moins n octets,
- `.readline()` retourne une ligne (la ligne courante),
- `.readlines()` retourne la liste de toutes les lignes du fichier,
- `.write(s)` écrit la chaîne de caractères `s` dans le fichier (à la position courante),
- `.writelines(lines)` écrit la ligne (liste de chaînes de caractères) `lines` dans le fichier (à la position courante),
- `.close()` ferme le fichier,
- `.next()` retourne la ligne suivante,
- `.seek(0)` repositionne la lecture/écriture au début du fichier.

D'autre part, certaines méthodes de la classe `str` sont très utiles quand on manipule les données d'un fichiers :

- `.strip()` élimine tous les espaces et caractères d'échappement qui commencent ou terminent une ligne (le marqueur de fin de ligne `'\n'` notamment),
- `.split(separateur)` retourne une liste des séquences de caractères qui dans la chaîne de caractères sont séparées par `separateur`,
- `.replace(ancienne, nouvelle)` renvoie une copie de la chaîne de caractère dans laquelle `chaine` la suite de caractères `ancienne` par `nouvelle`.

7. Refaire les questions 4 et 6 à l'aide de méthodes python.
8. Écrire à l'aide de méthodes python une procédure permettant de copier le contenu du fichier `TP6tab.txt` dans un nouveau fichier nommé « TP6.tsv ».

Exercice

Le fichier `notes.csv` est situé dans le même répertoire que `TP6.txt` et contient le bulletin scolaire de quinze élèves avec, sur chaque ligne, séparés par des virgules : leur nom, leur prénom et trois notes (écrites avec un séparateur décimal anglo-saxon, c'est-à-dire un point). Les coefficients de chacune des notes sont

3. Comme les fichiers données initiales de ce TP par exemple...

(dans l'ordre) 1, 2 et 3. On souhaite modifier le fichier pour rajouter une colonne supplémentaire dans laquelle figure la moyenne de chaque élève et rajouter une ligne d'en-têtes pour que le fichier soit plus explicite.

9. Copier/coller le fichier `notes.csv` dans votre répertoire personnel.
10. Écrire une procédure qui :
    - récupère les données présentes dans le fichier,
    - calcule la moyenne de chaque élève,
    - calcule la moyenne des moyennes de la classe et l'écart-type des moyennes de la classe,
    - crée un nouveau fichier `notes_et_moyennes.csv`, dont le contenu est celui du fichier `notes.csv` avec :
        - une ligne supplémentaire de commentaire en en-tête de fichier décrivant les différentes colonnes (nom, prénom,notes1,notes2,notes3,moyenne),
        - ajoute la moyenne de chaque élève en fin de ligne,
        - calcule la moyenne globale et l'écart-type des moyennes de la classe et les affiche à l'écran.

Pour le calcul de l'écart-type, on doit importer la fonction racine carrée qui n'est pas présente dans les bibliothèques du cœur de python mais est présente dans la bibliothèque `math`. En début de script, juste avant les premières instructions, on insère :
Ce qui permet d'utiliser ensuite la fonction racine carrée dont la syntaxe est :
`math.sqrt(nombre)`