Chapitre 11 : mini cours sur la gestion de fichiers en python

1 Une fonctionnalité puissante de PYTHON : les manipulations de fichiers textes

Les commandes de manipulations sur les chaînes de caractères sont encore plus intéressantes si on peut les appliquer à un fichier de textes.

1.1 Le module `os`

Ce module de PYTHON permet de dialoguer avec le système d'exploitation. Par exemple :

```
from os import getcwd # pour :  get current working directory
print(getcwd()) # indique le répertoire courant
from os import chdir
chdir("/Users/romain/Documents/MPSI/Informatique/CoursInfo2020-2021/Chapitre11-gestion-fichier"
```

Question : le chemin rentré ci-dessus est-il relatif ou absolu ? Et celui-ci ?

```
chdir("MPSI/Informatique/CoursInfo2020-2021/Chapitre11-gestion-fichier")
```

1.2 Ecrire un petit fichier texte

Vous pouvez fabriquer un fichier texte « pur » avec un éditeur de texte (attention bill gates'word-processor n'est *pas* pur).

- sous Linux avec Gedit par exemple, tapez `gedit`,
- sous MacOS avec TextEdit à condition de ne pas sauvegarder en `rtf` (rich text format) mais en `txt`,
- sous Windows avec Edit (là aussi format par défaut à vérifier ?)

J'en ai créé un que j'ai mis dans mon répertoire ci-dessus, que j'ai appelé `mon_ptit_texte.txt`

1.3 La commande `open` de PYTHON : créer, modifier des fichiers

Attention : Je vais parler ici de la commande `open` qui est une commande dans le module principal de PYTHON (pas besoin de charger de module).

Attention : ne pas faire `from os import *` sinon `open` change de nature

Examinons le code suivant :

```
MonContenu=open('essai.txt','a',encoding="utf8") # sans préciser l'encodage c'est  incertain.
MonContenu.write("J'ecris mon texte dedans")
MonContenu.write("J'ecris à la suite")
MonContenu.close()
```

La command `open` ouvre (et s'il n'existe pas crée) un fichier, qui aura pour nom `essai.txt` pour le système d'exploitation et qui figurera sous ce nom dans le *current working directory*. Elle demande un deuxième argument obligatoire, qui ici est 'a' comme ajouter (*append*) cela signifie que ce que l'on va écrire dans le fichier s'écrira à la suite de ce qui est déjà. [1]

Le troisième argument, optionnel, précise l'encodage (voir § 2 pour ce mystère).

La commande `open` renvoie un objet, que j'ai appelé `MonContenu` : c'est le nom de l'objet manipulé par le langage PYTHON.

`MonContenu` est un objet d'un type particulier..

1. Une alternative, toujours pour écrire dans un fichier, est de remplacer 'a' par 'w' : dans ce cas, s'il y avait déjà un fichier de ce nom, il est écrasé et un fichier vide est créé, dans lequel on pourra écrire. Si le fichier n'existe pas, il est créé.

```
<class '_io.TextIOWrapper'>
```

La *méthode* `write` s'applique à cet objet pour *écrire dedans, à la suite de ce qui précède.*

Le `close` est indispensable pour que le fichier redevienne disponible pour l'O.S.

1.4 La commande `open` pour la lecture seule

N'oublions pas que j'ai créé un fichier `mon_ptit_texte.txt` avec mon éditeur de texte favori et qu'il est dans mon *current working directory.*

```
MonWrap=open('mon_ptit_texte.txt','r',encoding="utf8")
MaChaine=MonWrap.read()
print(MaChaine)
MonWrap.close()
```

Cette fois, on a mis l'argument `'r'` pour `read` dans `open` : ouverture du fichier en lecture seulement. La commande `.read()` qui est une *méthode* appliquée à l'objet `MonWrap` renvoie une chaîne de caractèresqu'on stocke dans `MaChaine`. On a ainsi récupéré le contenu du fichier comme une chaîne de caractères utilisable en PYTHON.

1.5 Ce qu'on peut faire quand on lit un fichier ?

Le but est de faire des manipulations dans le texte : par exemple, remplacer certains mots dans un texte.... disons par exemple qu'on veut enlever tous les e dans un texte qui est dans un fichier, comment faire ?

D'abord il faut bien distinguer les *trois types d'objets informatiques* dont il est question ici :

- Le fichier par exemple `monPtitTexte.txt` qui est géré par le système d'exploitation.
- L'objet PYTHON, de type `io.TextIOWrapper`, qu'on appellera ici `MonWrap`, qu'on obtient quand on fait `MonWrap=open(''monPtitTexte.txt",..;)`.
- La chaîne de caractères (type `string` de PYTHON) qu'on obtient par la méthode `read` appliquée à l'objet `MonWrap`.

C'est sur la chaîne de caractères qu'on pourra travailler... ensuite, une fois modifiée, il faudra la réécrire dans le fichier.

1.6 Mise en oeuvre concrète

On veut enlever les « espaces » de `mon_ptit_texte.txt` :

```
# D'abord on lit comme on avait fait plus haut.
MonContenu=open('mon_ptit_texte.txt','r',encoding="utf8")
MaChaine=MonContenu.read()
print(MaChaine) # on a le contenu dans la chaîne de caractères :  MaChaine
MonContenu.close()
# Ensuite, on fabrique une nouvelle chaine de caractère MaChaineModifie
# Ensuite on la réécrit dans le fichier. : écrire le code correspondant !
```

Comment aurait-on fait si on avait voulu plutôt par exemple transformer majuscules en minuscules ?

2 Quelques précisions sur l'encodage des caractères

2.1 L'ASCII historique

ASCII est l'acronyme de American Standard Code for Information Interchange. Il s'agit d'une norme de codage de caractères très ancienne à l'échelle de l'informatique : les américains n'utilisant pas d'accents pensaient alors (années 1960) qu'il suffirait de coder des caractères sur 7 bits, i.e. avec 128 nombres numérotés de 0 à 127. Si l'on ajoute que les 20 premiers caractères codent des objets liées aux séparations et aux transmissions et pas des « vrais » caractères cela fait peu !

Il peut ainsi savoir que le `A` majuscule se trouve codé par le chiffre `65`, voici un extrait de Wikipédia :

48	060	30	0110000	0	Le chiffre zéro
49	061	31	0110001	1	Le chiffre un
50	062	32	0110010	2	Le chiffre deux
51	063	33	0110011	3	Le chiffre trois
52	064	34	0110100	4	Le chiffre quatre
53	065	35	0110101	5	Le chiffre cinq
54	066	36	0110110	6	Le chiffre six
55	067	37	0110111	7	Le chiffre sept
56	070	38	0111000	8	Le chiffre huit
57	071	39	0111001	9	Le chiffre neuf
58	072	3A	0111010	:	*Deux-points*
59	073	3B	0111011	;	*Point-virgule*
60	074	3C	0111100	<	*Inférieur*
61	075	3D	0111101	=	*Égal*
62	076	3E	0111110	>	*Supérieur*
63	077	3F	0111111	?	*Point d'interrogation*
64	0100	40	1000000	@	*Arobace* (aussi dénommé *Arobase* ou *A commercial*[8])
65	0101	41	1000001	A	
66	0102	42	1000010	B	
67	0103	43	1000011	C	
68	0104	44	1000100	D	
69	0105	45	1000101	E	
70	0106	46	1000110	F	
71	0107	47	1000111	G	
72	0110	48	1001000	H	
73	0111	49	1001001	I	

et que le `a` minuscule se trouve codé par le chiffre `97`, ce qui a l'air très quelconque mais en base deux : $65 = 1000001_{[2]}$ et $97 = 1100001_{[2]}$.

La conversion majuscule minuscule en ASCII revient à changer le deuxième bit !

2.2 L'ASCII étendu

Les problèmes ont donc commencé quand les différents pays, avec leurs langues différentes, ont choisis d'utiliser le 8ème bit pour coder les caractères accentués [2]... la plus connue est la norme ISO 8859-1, aussi appelée *Latin-1*, qui étend l'ASCII avec les caractères accentués utiles aux langues comme le français et l'allemand. Mais laisse de côté les lettres grecques ou cyrilliques... pour lesquelles il y a un autre codage ASCII étendu sur 8 bits, bien sûr incompatible... de même pour l'arabe etc.. tout cela avec des normes aux doux noms de ISO 8859-7,8859-8 etc...

L'intérêt de ces normes est leur simplicité : chaque caractère est codé par un octet (8bits) et donc une chaîne de caractères est une séquence d'octets. C'est ainsi que fonctionnait l'ancien type `string` de PYTHON dans les versions antérieures à la version 3.0.

2. au départ le 8ème bit existait déjà, mais comme bit de contrôle en cas d'erreur..

2.3 Les normes Unicode

Avec la globalisation et Internet, on a compris (au début des années 1990) la nécessité de pouvoir encoder [3], dans un même texte, tous les caractères de n'importe quel alphabet ! Une organisation internationale a été créée : le *consortium Unicode*.

Le principe (double) :

> • chaque caractère de n'importe laquelle des langues et des signes utilisés sur terre est codé par un nombre *unique*, indépendant des machines et des systèmes d'exploitation. A titre d'information ce nombre de caractères est de l'ordre de 250000.
> • En revanche, la façon dont ce nombre est ensuite codé en machines a le droit de dépendre du support matériel ou logiciel (O.S.)

L'intérêt du premier point est évident. Celui de second est de ne pas fixer une façon d'encoder qui serait trop rigide, qui prendrait trop de place quand on n'en a pas besoin, et qui ne serait pas compatible avec des codages plus anciens.

2.4 Conversion en PYTHON d'une chaîne de caractères en chaîne bytes

Considérons le code suivant :

```
reve="un été sur une île"
code_u=reve.encode("utf8")#
code_l=reve.encode("Latin-1")#
print(code_u)
print(code_l)
```

Avec le retour :

```
b'un \xc3\xa9t\xc3\xa9 sur une \xc3\xaele'
b'un \xe9t\xe9 sur une \xeele'
<class 'bytes'>
```

La *méthode* `encode` s'applique sur la chaîne reve en faisant ce que son nom indique ! Elle renvoie un objet qui est de classe *bytes*. Cette classe dit que les objets sont des octets mis à la suite.

L'affichage de `code_u` montre qu'en utf-8 les caractères accentués sont codés sur 2 octets (16bits) par exemple le é est remplacé par `\xc3\xa9` i.e. le nombre hexadécimal C3A9. Rappelons qu'avec deux chiffres en hexa. on code exactement un octet. En revanche en Latin-1 le é est remplacé par `\xe9` i.e. seulement le code E9 autrement dit 233.

Remarque : Il nous manque maintenant peu de chose pour comprendre comment faire nos méthodes maisons pour remplacer les méthodes sur les chaînes de caractères comme celle qui transforme majuscule en minuscule par exemple... mais les enjeux de ces conversions de chaînes de caractères en *bytes* sont (heureusement) plus nombreux que cela... par exemple la compression sans perte d'un texte, la cryptographie... et d'autres choses encore qui pourront servir pour vos TIPE.

3. Ce mot est un anglicisme, car il veut dire la même chose que le français *coder*, mais il s'est imposé !