Fichier texte

On a déjà parlé de ASCII et Unicode.

Fichiers et encodages

fichier = suite d'octets. Quelle est leur signification ?
Le fichier peut avoir une extension. Le format `.txt` sous **Windows** indique qu'il s'agit d'un fichier texte. On peut préciser l'encodage dans le fichier. Par exemple en html, dans l'entête :

```
<meta charset="utf-8">
```

ASCII

Déjà mentionné, 128 caractères sans accent, donc 7 bits par caractère.
Dans les fait on utilise plutôt un des codages suivants qui sont compatibles avec ASCII si bien qu'on n'utilise jamais moins d'un octet par caractère.

ISO 8859-1 *Latin-1 = Europe occidentale [1986]*

Ajoute des caractères accentués à ASCII. Suffit pour le français.
Tend à disparaître au profit d'**UTF-8** (pas plus de 10 % d'utilisation)

ANSI *Windows-1252 ou CP1252*

C'est un codage *Windows*, très proche de **ISO 8859-1**.
ANSI se contente de remplacer certains codes de contrôles (*octets ne correspondant pas à un caractère affichable*) par des caractères.

UTF-8 *Universal character set Transformation Set - 8 bits*

C'est un codage fait pour utiliser **Unicode** mais en allégeant un peu.
Avec UTF-8, selon les cas, un caractère peut-être codé sur 1, 2 ou 4 octets.

↪ Quand c'est un caractère ASCII, il ne prend qu'un octet. Ainsi, un fichier en ASCII est parfaitement compatible avec UTF-8. C'est un avantage de cet encodage.

↪ Un caractère comme "é" va être encodé par deux octets : `0xC3 A9`.

↪ L'encodage est fait pour que l'on sache à chaque fois combien on devra lire d'octets pour le prochain caractère.

↪ Quand l'octet commence par 0, ce sera juste cet octet là, donc il reste 7 bits qui feront de l'ASCII.

↪ Quand l'octet commence par 110, il faut prendre cet octet et le suivant qui commence forcément par 10.

On aura 110xxxxx 10xxxxxx, les x forment la partie codante de l'octet.

Par exemple "é" donne 0xC3 A9 = 110[0 0011] 10[10 1001] -> c'est le code 00011101001 = 0xE9, c'est à dire "é" en Unicode.

↪ cela continue avec d'autres configurations pour les caractères moines fréquents.

Manipulation de fichiers

Les fichiers considérés contiennent :

- des phrases
  ex : on veut analyser les statistiques d'un roman. Nombres de mots, etc.
- des données
  ex : une suite de cotations boursières séparées par ,
  On parle de fichier *.csv* = **C**omma **S**eparated **V**alues

Ouverture

```
f = open('nom_du_fichier', mode, encodage='utf8')
```

- mode `'w'` pour écriture *écrase le fichier si déjà existant*
- mode `'r'` pour lecture
- mode `'a'` pour ajout
- `'b'` pour binaire
  on indique `'wb'` pour écriture binaire, `'rb'` pour lecture binaire.
  Correspond à une édition façon éditeur hexadécimal. On donne les octets tels qu'ils sont écrits dans le fichier sans chercher à les interpréter.
- Si le mode n'est pas binaire, on est en mode texte.
  Dans ce cas, il faut connaître l'encodage. Le mieux est de le préciser explicitement en écrivant `encodage = 'utf8'` par exemple.
- mode `'x'` pour création *échoue si le fichier existe*

Manipulation des lignes

```
f.write("chaine de caractère") # ajoute la chaîne de caractère
                               # au fichier
f.read()       # renvoie l'ensemble du texte
f.read(n)      # renvoie les n prochains caractères
f.readline()   # renvoie la prochaine ligne
f.readlines()  # renvoie l'ensemble des lignes
               # sous forme d'un tableau de chaînes.
```

Bon à savoir

Ces caractères sont importants :
`'\n'` : saut de ligne
`'\t'` : tabulation

Quand on récupère une ligne de fichier texte, elle contient souvent le `'\n'` final et on peut vouloir s'en débarrasser. Cette commande est là pour cela :

```
chaine.rstrip() # Supprime les caractères en fin de ligne
                # comme espace et \n.
```

Quand on récupère une ligne de fichier `.csv`, par exemple :
`chaine = "Michel,Dupont,12/11/1984,3 rue de la fontaine"`
On va vouloir la fractionner sous forme d'un tableau :

```
['Michel', 'Dupont', '12/11/1984', '3 rue de la fontaine']
```

Cette commande est là pour cela :

```
chaine.split(',')
```

Libérer la ressource

Quand Python ouvre un fichier, il demande au **système d'exploitation** (*OS*) de lui donner l'accès à ce fichier. Quand c'est une demande d'écriture, l'OS **verrouille** le fichier pour qu'il n'y ait pas deux programmes écrivant en même temps dans un même fichier. Il est donc important de **libérer** le fichier quand on a fini de l'utiliser.

```
f.close() # libère le fichier
```

Pour être certain de ne pas oublier la fermeture, on peut empaqueter l'utilisation de fichier dans une structure de programmation :

```
with open('nom_du_fichier', mode, encodage='utf8') as f:
    # on peut utiliser f tant qu'on est à l'intérieur de ce with
# dès qu'on sort du with, f est fermé.
```