NSi	**Représentation numérique de l'information** Textes	**Cours 8**

Un fichier informatique est une suite d'octets stockés en mémoire. Utiliser tel encodage plutôt que tel autre revient à interpréter la même séquence d'octets comme deux textes différents.

1. Le codage ASCII

Dans les années 1960, le besoin d'échanger du texte entre les ordinateurs a entrainé le choix d'une convention de codage standardisée. C'est ainsi qu'est né le code **ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange)
C'est une table de 128 caractères numérotés de 0 à 127 codés sur 7 bits.
La table contient les lettres majuscules, minuscules, les chiffres, de la ponctuation et des caractères invisibles comme l'espace, la tabulation ou le retour à la ligne.
Suffisante pour rédiger un texte en anglais ou écrire un programme informatique, la table ASCII ne contient aucun caractère accentué.

En Python, la fonction **ord** donnera le code ASCII d'un caractère et la fonction **chr** donnera le caractère correspondant à un code ASCII.
On peut donc voir la table en saisissant en Python le code suivant :

```
for i in range (128) :
    print (chr(i), end='')
```

```
	 - !"#$%&'()*+,-
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

2. ASCII étendu

Les tables ASCII étendues contiennent 256 caractères. Il existe une multitude de telles tables (latin 1, latin 9, windows-1252…).
Elles contiennent les 128 premiers caractères de la tables ASCII, ainsi que 128 caractères supplémentaires spécifiques selon la table. Elles seront donc codées sur 8 bits.
Par exemple en français le codage latin 1 sera :

```
for i in range (256) :
    print (chr(i), end='')
```

```
	 - !"#$%&'()*+,-
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~□□
□□□□□□□□^□□□□□□□□□‘’□□□□□˜□□□□□□□□ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓ
ÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ
```

3. Unicode

Lorsqu'un fichier est enregistré avec une telle table de caractères (**charset**) étendue, il faudra le lire avec la même table, sans quoi certains caractères ne seront pas corrects, ce phénomène est encore parfois observé sur les pages web si le navigateur n'utilise pas le bon charset.

Pour régler ce problème d'encodage, une norme est apparue dans les années 1990 : **Unicode**.
Cette table contient plus de 135 000 symboles pour pouvoir couvrir les besoins de toutes les langues.
Malheureusement, Unicode ne peut pas être utilisée directement, car la taille des fichiers serait multipliée par 3 par rapport à la tables ASCII.

Il existe plusieurs codages dont **UTF-8** : ce codage reprend le premier octet de la tables ASCII et code le reste sur 2 à 4 octets.
Pour écrire par exemple un fichier texte avec l'extension html qui va produire une page Web, il est conseillé d'utiliser l'encodage UTF-8 qui est totalement compatible avec l'ASCII.

4. Encodages en Python

Voici un code en Python qui permet d'écrire dans deux fichiers **test1.txt** et **test2.txt**.

- Dans le **premier fichier**, on écrit "bonjour" suivi d'un retour à la ligne ('\n').
- Dans le **second fichier**, on écrit 'é è ê à ù' avec les caractères é, è, ê, à, ù séparés par des tabulations ('\t') et suivis d'un retour à la ligne.

```
fic = open("test1.txt", "w")
fic.write("Bonjour\n")
fic.close()

fic = open("test2.txt", "w")
fic.write("é\tè\tê\tà\tù\n")
fic.close()
```

Si nous ouvrons ces deux fichiers avec le logiciel **Notepad++**, et si nous cliquons sur le menu **Encodage**, nous constatons que le premier est encodé en **UTF-8** et le second en **ANSI**. (**ANSI** est un nom donné à un encodage Windows appelé aussi **Windows-1252** ou **CP1252**)

On peut forcer l'écriture dans un fichier selon un encodage choisi, pour cela :

Créons une variable string ch :

```
ch="éàôù"
```

Nous pouvons l'encoder directement en ANSI :

```
ch.encode('cp1252')
```

```
b'\xe9\xe0\xf4\xf9'
```

Ou en UTF-8 :

```
ch.encode('utf-8')
```

```
b'\xc3\xa9\xc3\xa0\xc3\xb4\xc3\xb9'
```

En utilisant la fonction write et en précisant le mode wb (write binary) on peut encoder directement le fichier en utf-8

```
ch="eaou"
utf=ch.encode('utf-8')

fic = open('utf.txt','wb')
fic.write(utf)
fic.close()
```

5. Gestion des fichiers textes en Python

- **Ouverture et fermeture d'un fichier**

La fonction **open** prend deux paramètres, le nom du fichier et le mode d'ouverture :

- '**w**' pour le mode "**écriture**" (write),
- '**r**' pour le mode "**lecture**" (read),
- '**a**' pour le mode "**ajout**" (append)

La syntaxe est la suivante : `fic = open('fichier.txt','w')`
Il est essentiel de fermer un fichier qui a été ouvert avec la fonction **close**. La syntaxe est : `fic.close()`
Lorsqu'on utilise la fonction open, si le fichier n'existe pas il sera créé, s'il existe le fichier existant sera ouvert.
Par défaut, le fichier est créé dans le dossier où est enregistré le fichier Python.

- **Écriture**

La méthode **write** prend en paramètre une chaîne de caractères (type str),
Pour écrire sur plusieurs lignes, on utilise le caractère `\n` qui force un retour à la ligne :

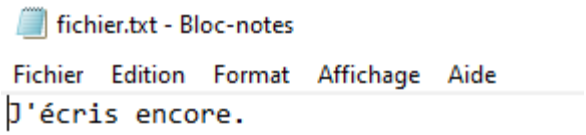
```
fic.write("J'écris dans un fichier\nA la fin, je le ferme.")
```

ou plus lisible

```
fic.write("J'écris dans un fichier" + "\n" + "A la fin, je le ferme.")
```

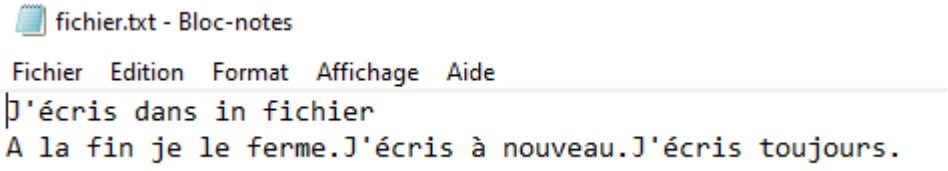
Si le fichier est fermé puis à nouveau ouvert en écriture, c'est un nouveau fichier qui sera écrit.
Ce qui était écrit auparavant **est perdu**.

```
fic = open('fichier.txt','w')
fic.write("J'écris dans in fichier\nA la fin je le ferme.")
fic.write("J'écris à nouveau.")
fic.close()

fic = open('fichier.txt','w')
fic.write("J'écris encore.")
fic.close()
```



Pour écrire à nouveau dans un fichier déjà fermé, on l'ouvre en mode "**ajout**" (append en anglais).
Dans ce cas, le texte est écrit à la suite du précédent.

```
fic = open('fichier.txt','w')
fic.write("J'écris dans in fichier\nA la fin je le ferme.")
fic.write("J'écris à nouveau.")
fic.close()

fic = open('fichier.txt','a')
fic.write("J'écris toujours.")
fic.close()
```



Si les données à écrire sont de **type numérique**, il faudra les convertir au préalable en **type str**.

- **Lecture**

L'ouverture d'un fichier en mode lecture s'effectue avec la méthode **read**.
Plusieurs instructions sont disponibles :

- `ch=fic.read(n)` lit **n caractères**,
- `ch=fic.read()` lit **tout le fichier**,
- `ch=fic.readline()` lit **la ligne courante** et passe à la suivante.
- `ch=fic.readlines()` lit **toutes les lignes**

Dans les trois premiers cas, la variable **ch** est une **chaîne de caractères**.
Dans le dernier cas, la variable **ch** est une **liste de chaînes de caractères**. Chaque élément est une ligne du fichier. L'instruction `ch=[x for x in fic]` produira le même résultat.

Deux méthodes sur les chaînes de caractères sont importantes dans le traitement des données lues.
Si **ch** est une chaîne de caractères, alors :

- `ch.rstrip()` supprime le caractère de fin de ligne (par exemple "\n") ;
- `ch.split(sep)` coupe la chaîne ch suivant le délimiteur sep, et renvoie une liste de sous-chaînes de ch.
  Avec `ch="un,deux,trois"` alors `ch.split(',')` renvoie `['un','deux','trois']`.

Le séparateur par défaut est l'espace.
Si les données à lire sont des nombres qui doivent être utilisés dans des calculs, il faut les convertir en type int ou en type float.
Voici un exemple :

```
fic = open("fichier.dat", "w")
fic.write(str(5) + "\t" + str(8.3) + "\t" + str(1e-4) + "\n")
fic.write(str(8) + "\t" + str(32.7) + "\t" + str(1e2))
fic.close()
# "\t" ajoutera une tabulation

fic = open("fichier.dat", "r")
for ligne in fic:
    print("---")
    print(ligne)
    liste = ligne.rstrip().split("\t")
    print(liste)
    a,b,c = [float(x) for x in liste]
    print(a,b,c)

fic.close
```