CHAPITRE 9 : LECTURE ET ÉCRITURE DANS UN FICHIER

Un fichier stocke des informations sur un support physique (disque dur, clé USB, CD, DVD, carte mémoire SD...).
Ouvrir un fichier consiste à le charger dans la mémoire vive (RAM) de l'ordinateur (c'est une mémoire volatile : elle s'efface quand on éteint l'ordinateur). Enregistrer un fichier consiste à l'écrire sur un support physique de stockage (l'information est alors conservée de manière permanente).
Il existe deux types de fichiers :

- Les **fichiers textes** : l'information est stockée sous forme de caractères lisibles par un éditeur de texte (principalement des lettres et des chiffres).
- Les **fichiers binaires** : l'information est stockée en binaire (une suite d'octets dont la valeur est comprise entre 0x00 et 0xFF).

I. Le module OS.

Le module OS (Operating System) permet la manipulation de fichiers et de répertoires : parcourir, renommer, créer ou détruire mais pas la copie de fichiers.

- La fonction **getcwd()** renvoie le répertoire courant (GET Current Working Directory).
- La fonction **chdir(*nomrep*)** se place dans le répertoire ***nomrep*** (Change DIRectory).
- La fonction **rename(*src*, *dest*)** renome ***src*** en ***dest***.
- La fonction **mkdir(*chemin*)** crée le répertoire ***chemin*** (MaKe DIRectory).
- La fonction **remove(chemin)** ou **rmdir(*chemin*)** supprime ***chemin*** (ReMove DIRectory).
- La fonction **listdir(*chemin*)** donne la liste des fichiers de ***chemin***.

II. Écriture dans un fichier

2.1 Le mode write.

L'écriture dans un fichier se fait avec la fonction open() en mode écriture.
Ancienne méthode :

```
# création et ouverture du fichier test.txt en mode write 'w' (écriture)
# si le fichier test.txt existe déjà, il est écrasé
fichier = open('test.txt', 'w')      # instanciation de l'objet fichier de la classe file
fichier.write('Bonjour à tous !') # écriture dans le fichier avec la méthode write()
fichier.close() # fermeture du fichier avec la méthode close()
```

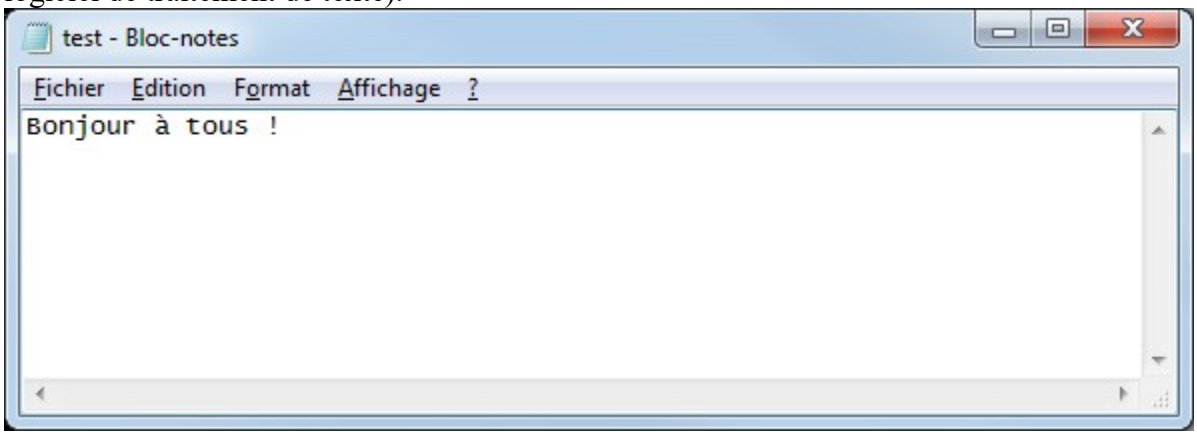
Le fichier test.txt est créé dans le répertoire courant.

Depuis la version 2.5, Python introduit le mot-clé **with** qui permet d'ouvrir et fermer un fichier de manière commode. Si pour une raison ou une autre l'ouverture conduit à une erreur (problème de droits, etc), l'utilisation de **with** garantit la bonne fermeture du fichier (ce qui n'est pas le cas avec l'utilisation de la méthode open().
La variable fichier est créé temporairement, et est uniquement accessible dans le bloc indenté avec le with.
Le fichier est automatiquement fermé à la fin du bloc, même en cas d'erreurs.

Nouvelle méthode :

```
with open('test.txt', 'w') as fichier:
    fichier.write('Bonjour à tous !') # écriture dans le fichier avec la méthode write()
```

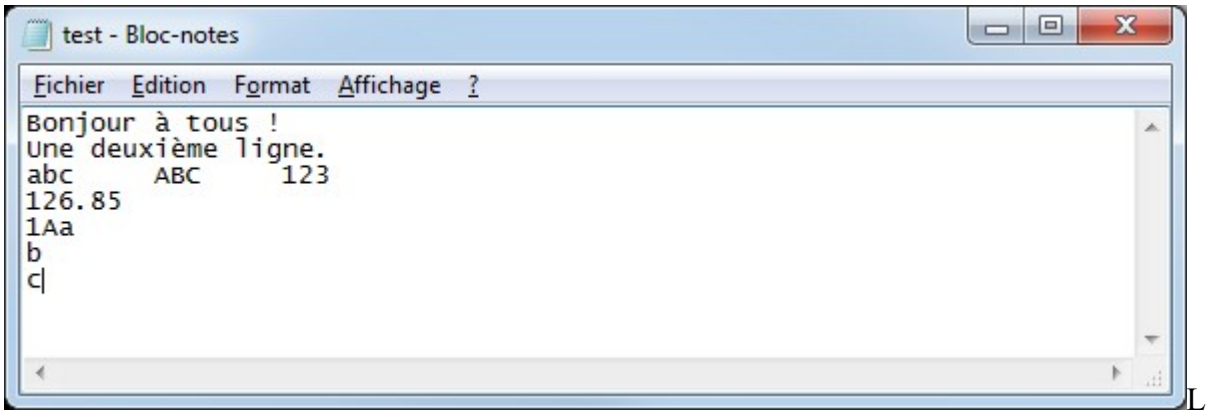
Vous pouvez vérifier son contenu en l'ouvrant avec un éditeur de texte (ou même avec un logiciel de traitement de texte).



2.2 Le mode append.

Pour écrire à la fin d'un fichier, on utilise la fonction open() en mode ajout :

```
# ouverture du fichier test.txt en mode append 'a' (ajout)
with open('test.txt', 'a') as fichier:
    fichier.write('Bonjour à tous !') # écriture dans le fichier avec la méthode write()
    fichier.write('\nUne deuxième ligne.\n')# '\n' saut de ligne
    fichier.write('abc\tABC\t123\n')      # '\t' tabulation
    fichier.write(str(126.85) + '\n')     # str() transforme un nombre en chaîne de caractères
    fichier.write('\x31\x41\x61\n')       # écriture de '1Aa' en code ASCII
    fichier.write(chr(0x62) + '\n')       # écriture de 'b' en code ASCII
    fichier.write(chr(99))                # écriture de 'c' en code ASCII
```



L

III. Lecture dans un fichier.

3.1 Lecture en mode binaire.

En mode binaire, un fichier est lu octet par octet.
Par exemple, si on veut la taille d'un fichier, il faut utiliser la lecture en mode binaire 'rb' (read binary) :

```
import binascii

# ouverture du fichier test.txt en mode read binary 'rb' (lecture en mode binaire)
with open('test.txt', 'rb') as fichier:
    data = fichier.read()
```

```
    # affichage de la taille du fichier
    print('Taille du fichier :', len(data), 'octets')
    # affichage du contenu (en octet)
    print(data)
    # affichage du contenu (en hexadécimal)
    print(binascii.hexlify(data))
```

```
>>>
Taille du fichier : 85 octets
b'Bonjour \xe0 tous !Bonjour \xe0 tous !\r\nUne deuxi\xe8me ligne.\r\nabc\tABC\t123\r\n126.85\r\n1Aa\r\nb\r\nc'
b'426f6e6a6f757220e020746f75732021426f6e6a6f757220e020746f757320210d0a556e6520646575786e86d65206c69676e652e0d0a6162630941424309313233 0d0a3132362e38350d0a3141610d0a620d0a63'
```

3.2 Lecture en mode texte.

En mode texte, un fichier est lu caractère par caractère.
La lecture dans un fichier texte se fait avec la fonction open() en mode lecture :

Pour lire dans un fichier test, Python offre plusieurs possibilités :

➢ La méthode **readlines()** qui lit toutes les lignes du fichier et les range dans une séquence. Utile si on souhaite lire tout le fichier d'un coup :

```
touteslesligne = fichier.readlines()
```

Exemple avec le fichier du 2.2 :

```
# ouverture du fichier test.txt en mode read 'r' (lecture en mode texte)
with open('test.txt', 'r') as fichier:
    li = fichier.readlines()      # lecture dans le fichier avec la méthode readlines()
    print('Contenu du fichier :\n')
    print(li)            # affichage du contenu intégral du fichier
```

```
>>>
Contenu du fichier :
['Bonjour à tous !Bonjour à tous !\n', 'Une deuxième ligne.\n', 'abc\tABC\t123\n', '126.85\n', '1Aa\n', 'b\n', 'c']
```

La méthode **readlines()** permet donc de lire l'intégralité d'un fichier en une instruction seulement. Cela n'est possible toutefois que si le fichier à lire n'est pas trop gros : puisqu'il est copié intégralement dans une variable, c'est-à-dire dans la mémoire vive de l'ordinateur, il faut que la taille de celle-ci soit suffisante. Si vous devez traiter de gros fichiers, utilisez plutôt la méthode **readline()** dans une boucle.

➢ La méthode **readline()** (au singulier) qui lit une ligne du fichier (jusqu'au retour à la ligne). Indispensable pour lire des lignes au coup par coup.

```
uneligne = fichier.readline()
```

Notez bien que **readline()** est une méthode qui renvoie une chaîne de caractères, alors que la méthode **readlines()** renvoie une liste de chaîne de caractères. À la fin du fichier, **readline()** renvoie une **chaîne vide**, tandis que **readlines()** renvoie une **liste vide**.

➢ La méthode **read(size)** qui lit une quantité de données égale à la taille size.
Si la taille est omise, la méthode **read()** lit le fichier intégralement comme la méthode **readlines()**.

```
ncaractere = fichier.read(n) #lit les n caractères courants à partir de la position courante
```

➢ Enfin, on peut aussi utiliser l'objet fichier comme un itérateur.
Pratique si on souhaite traiter l'une après l'autre toutes les lignes d'un fichier.

```
for ligne in fichier:
  print(ligne, end='')
```

pour se positionner dans le fichier :

```
fichier.seek(n) #se positionne sur le nième caractère
```

LECTURE ET ÉCRITURE DANS UN FICHIER - TESTS

I. 1°) Écrire un programme qui écrit dans un fichier le résultat de n lancers de 3 dés ainsi que la somme, du style :
Numéro de lancer | dé1 | dé2 | dé3 | => somme

```
>>> (executing file "lancers_de.py")
Entrer le nombre de lancers : 100
1       | 5 | 1 | 3 |       => 9
2       | 3 | 6 | 6 |       => 15
3       | 1 | 2 | 6 |       => 9
4       | 6 | 4 | 4 |       => 14
```

2°) Afficher le contenu du fichier.

II. Suite de Fibonacci.
1°) Créer un programme qui écrit dans le fichier *fibonacci.txt*, les *p* premiers termes de la suite de Fibonacci, *p* étant un nombre entier demandé.
Si $n = 5$, le contenu du fichier se présentera sous la forme :
1 1 2 3 5 …
2°) Compléter le programme afin d'afficher les termes de la suite sauvegardés dans le fichier.
3°) Compléter le programme pour afficher le $p^{ième}$ terme de la suite avec *p* entier positif ou nul.

III. Triangle de Pascal.
Voici le début du triangle de Pascal :

Exposant	Coefficients					
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
…	…					

1°) Créer un programme qui écrit dans le fichier *pascal.txt*, toutes les lignes du triangle de Pascal de l'exposant 1 jusqu'à un exposant *n* demandé.
Si $n=2$, le contenu du fichier se présentera sous la forme :
0 | 1
1 | 1 1
2 | 1 2 1
2°) Compléter le programme qui permet d'afficher le triangle de Pascal sauvegardé.

IV. On considère le fichier au format FASTA (voir wikipédia pour plus d'information sur ce format de fichier) test.fa qui est un extrait de séquences nucléotiques (ARN messagers) provenant de l'organisme entérobactérie Escherichia coli.

1°) Créer la fonction affiche(f) qui prend en paramètres :

- En entrée : un nom de fichier au format fasta. (on utilisera test.fa pour tester la fonction).

La fonction affiche le contenu du fichier ligne par ligne et le numéro de la ligne courante.

- En sortie : rien.

Exemple avec le fichier test.fa :

```
1 : >32115099|locus|VBIEscCol129921_0001| Aspartokinase (EC 2.7.2.4) / Homoserine dehydrogenase (EC 1.1.1.3) [Escherichia coli str. K-12 substr. MG1655]
2 : gtgttgaagttcggcggtacatcagtggcaaatgcagaacgttttctgcgtgttgccgat
```

2°) Créer la fonction nbcarac(f) qui prend en paramètres :

- En entrée : un nom de fichier au format fasta.
- En sortie : le nombre de caractères dans le fichier.

3°) Créer la fonction nbseq(f) qui prend en paramètres :

- En entrée : un nom de fichier au format fasta.
- En sortie : le nombre de séquences nucléotiques dans le fichier.

4°) Créer la fonction afficheseq(f,n) qui prend en paramètres :

- En entrée : un nom de fichier au format fasta et le numéro de séquence nucléotique à afficher dans le fichier.
- En sortie : la séquence nucléotique à afficher dans le fichier.