Fichiers

Bases de Python

Introduction

Les programmes que nous écrivons manipulent des données en mémoire vive (RAM). Mais cette mémoire est **volatile** : dès que le programme s'arrête, tout est perdu. Pour conserver des informations d'une exécution à l'autre, il faut les enregistrer dans un **fichier** stocké sur le disque.

Les fichiers sont omniprésents en informatique : fichiers de configuration, journaux d'événements (*logs*), sauvegardes de scores, relevés de mesures, exports de données. Savoir lire et écrire dans un fichier est donc une compétence fondamentale.

Le gestionnaire de contexte `with`

Python offre une syntaxe sûre pour manipuler les fichiers grâce au mot-clé `with` :

```
with open('fichier.txt', 'r') as f:
    contenu = f.read()
    print(contenu)
# Ici le fichier est automatiquement fermé
```

Le bloc `with open(...) as f` ouvre le fichier et l'affecte à la variable `f`. À la sortie du bloc (lorsque l'indentation revient au niveau précédent), Python **ferme automatiquement** le fichier, même si une erreur survient pendant le traitement.

Pourquoi utiliser `with` ? Sans `with`, il faudrait appeler `f.close()` manuellement. Un oubli pourrait entraîner une perte de données ou un fichier verrouillé. Le gestionnaire de contexte élimine ce risque.

Les modes d'ouverture

La fonction `open()` prend deux arguments principaux : le **nom du fichier** et le **mode d'ouverture**.

Mode	Signification	Fichier existant	Fichier inexistant
`'r'`	Lecture (*read*)	Lit le contenu	Erreur
`'w'`	Écriture (*write*)	Écrase le contenu	Crée le fichier
`'a'`	Ajout (*append*)	Écrit à la fin	Crée le fichier

Attention : le mode `'w'` est destructif. Si le fichier existe déjà, son contenu est **intégralement effacé** avant d'écrire les nouvelles données. Pour ajouter sans effacer, il faut utiliser le mode `'a'`.

L'encodage

Pour garantir un affichage correct des caractères accentués, il est recommandé de préciser l'encodage **UTF-8** :

```
with open('fichier.txt', 'r', encoding='utf-8') as f:
    contenu = f.read()
```

Sans cette précision, Python utilise l'encodage par défaut du système, qui peut varier d'un ordinateur à l'autre et provoquer des erreurs sur les caractères accentués.

Écriture dans un fichier

La méthode `write()`

La méthode `write()` écrit une chaîne de caractères dans le fichier :

```
with open('animaux.txt', 'w', encoding='utf-8') as f:
    f.write('Chat\n')
    f.write('Chien\n')
    f.write('Lapin\n')
```

Point essentiel : la méthode `write()` n'ajoute **pas** de saut de ligne automatiquement. Il faut écrire explicitement `\n` à la fin de chaque ligne. Sans cela, tout le texte apparaîtrait sur une seule ligne.

Écrire plusieurs lignes avec `writelines()`

On peut aussi passer une **liste** de chaînes à la méthode `writelines()` :

```
animaux = ['Girafe\n', 'Tigre\n', 'Singe\n', 'Souris\n']

with open('animaux.txt', 'w', encoding='utf-8') as f:
    f.writelines(animaux)
```

Comme `write()`, cette méthode n'ajoute pas de `\n` : c'est à nous de l'inclure dans chaque élément de la liste.

Ajouter du contenu avec le mode `'a'`

Pour compléter un fichier existant sans effacer son contenu, on utilise le mode `'a'` :

```
with open('animaux.txt', 'a', encoding='utf-8') as f:
    f.write('Éléphant\n')
    f.write('Zèbre\n')
```

Le fichier contiendra maintenant six animaux : les quatre précédents suivis des deux nouveaux.

Enregistrer des données structurées

On enregistre souvent plusieurs informations par ligne, séparées par un caractère convenu : par exemple le nom de chaque élève et sa note, séparés par un point-virgule. La méthode `write()` n'accepte que des chaînes de caractères : `f.write(16)` provoque une erreur `TypeError`. On construit donc chaque ligne avec une chaîne formatée (*f-string*), qui réunit le nom, le point-virgule, la note et le saut de ligne :

```
resultats = [('Alice Martin', 16), ('Bob Dupont', 12)]

with open('notes.txt', 'w', encoding='utf-8') as f:
    for nom, note in resultats:
        f.write(f'{nom};{note}\n')
```

Le fichier `notes.txt` contiendra :

```
Alice Martin;16
Bob Dupont;12
```

C'est ce fichier que l'on relit plus loin, ligne par ligne, avec la méthode `split()`.

Lecture d'un fichier

Considérons le fichier `eleves.txt` dont voici le contenu :

```
Alice Martin
Bob Dupont
Charlie Moreau
Diana Petit
```

Python offre plusieurs méthodes pour lire un fichier. Chacune a son utilité selon la situation.

Lecture ligne par ligne avec une boucle `for`

La méthode la plus courante et la plus économe en mémoire :

```
with open('eleves.txt', 'r', encoding='utf-8') as f:
    for ligne in f:
        print(ligne.strip())
```

Pourquoi `strip()` ? Suivons pas à pas ce que fait Python pour la première itération :

```
Étape 1 : Python lit la première ligne -> ligne = 'Alice Martin\n'
Étape 2 : print('Alice Martin\n') affiche 'Alice Martin' + \n du fichier
Étape 3 : print() ajoute son propre saut de ligne
Résultat : on obtient DEUX sauts de ligne -> une ligne vide apparaît
```

Pour l'éliminer, on utilise la méthode `strip()` qui supprime les caractères invisibles (espaces, tabulations, `\n`) en début et en fin de chaîne :

```
Étape 1 : ligne = 'Alice Martin\n'
Étape 2 : ligne.strip() -> 'Alice Martin'  (le \n est supprimé)
Étape 3 : print('Alice Martin') -> affichage correct, un seul saut de ligne
```

Lecture d'une seule ligne avec `readline()`

La méthode `readline()` lit **une seule ligne** et avance le curseur de lecture :

```python
with open('eleves.txt', 'r', encoding='utf-8') as f:
    premiere = f.readline()
    deuxieme = f.readline()
    print(premiere.strip())   # Alice Martin
    print(deuxieme.strip())   # Bob Dupont
```

Chaque appel à `readline()` lit la ligne suivante. C'est utile quand on veut traiter les premières lignes différemment (par exemple, sauter un en-tête).

Lecture complète avec `read()`

La méthode `read()` lit **tout le fichier** d'un coup et renvoie une unique chaîne de caractères :

```python
with open('eleves.txt', 'r', encoding='utf-8') as f:
    texte = f.read()
    print(texte)
```

C'est pratique pour les petits fichiers, mais **déconseillé** pour les fichiers volumineux car tout le contenu est chargé en mémoire.

Lecture de toutes les lignes avec `readlines()`

La méthode `readlines()` lit tout le fichier et renvoie une **liste de chaînes**, une par ligne :

```python
with open('eleves.txt', 'r', encoding='utf-8') as f:
    lignes = f.readlines()
    print(lignes)
```

Résultat :

```
['Alice Martin\n', 'Bob Dupont\n', 'Charlie Moreau\n', 'Diana Petit\n']
```

Chaque élément de la liste contient le `\n` final. Pour obtenir une liste propre, on peut utiliser une compréhension de liste :

```python
with open('eleves.txt', 'r', encoding='utf-8') as f:
    lignes = [ligne.strip() for ligne in f]
    print(lignes)
# ['Alice Martin', 'Bob Dupont', 'Charlie Moreau', 'Diana Petit']
```

La méthode `split()`

La méthode `split()` est souvent utilisée en complément de la lecture de fichiers. Elle découpe une chaîne en une **liste de sous-chaînes** selon un séparateur :

```
ligne = 'Alice Martin 16'
elements = ligne.split()
print(elements)   # ['Alice', 'Martin', '16']
```

Par défaut, `split()` utilise les espaces comme séparateur. On peut spécifier un autre caractère :

```
ligne = 'Alice;Martin;16'
elements = ligne.split(';')
print(elements)   # ['Alice', 'Martin', '16']
```

Application typique : lire un fichier structuré où chaque ligne contient plusieurs informations séparées par un caractère convenu.

```
# Fichier notes.txt :
# Alice Martin;16
# Bob Dupont;12

with open('notes.txt', 'r', encoding='utf-8') as f:
    for ligne in f:
        parties = ligne.strip().split(';')
        nom = parties[0]
        note = int(parties[1])
        print(f'{nom} a obtenu {note}/20')
```

Suivons le traitement de la première ligne :

```
Étape 1 : ligne = 'Alice Martin;16\n'
Étape 2 : ligne.strip() -> 'Alice Martin;16'
Étape 3 : 'Alice Martin;16'.split(';') -> ['Alice Martin', '16']
Étape 4 : parties[0] -> 'Alice Martin'   parties[1] -> '16'
Étape 5 : int('16') -> 16
Étape 6 : affiche 'Alice Martin a obtenu 16/20'
```

Résumé

Méthode	Renvoie	Usage typique
`for ligne in f`	Chaîne (ligne par ligne)	Traitement ligne par ligne
`f.readline()`	Chaîne (une ligne)	Lire les premières lignes
`f.read()`	Chaîne (tout le fichier)	Petit fichier
`f.readlines()`	Liste de chaînes	Stocker toutes les lignes
`f.write(s)`	—	Écrire une chaîne
`f.writelines(L)`	—	Écrire une liste de chaînes

Exercices

Exercice 1 Écrire un programme qui crée un fichier `fruits.txt` contenant les noms de cinq fruits, un par ligne. Puis modifier le programme pour **ajouter** trois fruits supplémentaires au fichier existant (sans effacer les cinq premiers).

Exercice 2 On dispose du fichier `fruits.txt` créé à l'exercice précédent.

1. Écrire un programme qui lit le fichier et affiche chaque fruit précédé de son numéro (exemple : `1. Pomme`).
2. Écrire un programme qui lit le fichier et affiche le nombre total de fruits.

Exercice 3 On crée le fichier suivant :

```
with open('test.txt', 'w') as f:
    f.write('Bonjour')
    f.write('le')
    f.write('monde')
```

1. Sans exécuter le code, prédire le contenu du fichier `test.txt`.
2. Vérifier en exécutant le programme.
3. Modifier le programme pour que chaque mot apparaisse sur une ligne distincte.

Exercice 4 On dispose d'un fichier `mesures.txt` contenant une température par ligne :

```
18.5
21.3
19.7
22.1
17.8
20.4
```

Écrire un programme qui lit toutes les températures, puis affiche la température minimale, la température maximale et la moyenne (arrondie à un chiffre après la virgule).

Exercice 5 On dispose d'un fichier `eleves.txt` dont chaque ligne contient un nom et une note séparés par un point-virgule :

```
Alice Martin;16
Bob Dupont;12
Charlie Moreau;8
Diana Petit;14
Eve Bernard;18
```

Écrire un programme qui :

1. affiche le nom et la note de chaque élève ;
2. calcule et affiche la moyenne de la classe ;
3. affiche les noms des élèves ayant une note supérieure ou égale à la moyenne.

Exercice 6 Écrire un programme qui lit le fichier `eleves.txt` de l'exercice précédent et crée un fichier `reussite.txt` contenant uniquement les lignes des élèves ayant obtenu 10 ou plus.

Exercice 7 Écrire une fonction `statistiques(nom_fichier)` qui reçoit le nom d'un fichier texte et renvoie un dictionnaire contenant le nombre de lignes, le nombre de mots (en utilisant `split()`) et le nombre de caractères (espaces et sauts de ligne compris). Pour l'exemple ci-dessous, le fichier `eleves.txt` se termine par un saut de ligne.

```
stats = statistiques('eleves.txt')
print(stats)
# {'lignes': 5, 'mots': 10, 'caracteres': 77}
```