Chapitre 6 : Manipulation des Fichiers en Python

Par Robert DIASSÉ

1. Introduction

Un fichier est un espace de stockage permettant d'enregistrer des données de manière persistante. Contrairement aux variables qui disparaissent après l'exécution d'un programme, les fichiers permettent de conserver des informations.

Python permet de manipuler facilement des fichiers avec la fonction `open()`, qui permet d'ouvrir, lire, écrire et fermer des fichiers.

2. Ouvrir et fermer un fichier en Python

2.1. Ouvrir un fichier

L'ouverture d'un fichier se fait avec la fonction `open()`, qui prend en paramètres :

- **Le nom du fichier**
- **Le mode d'ouverture**

Voici les modes d'ouverture les plus utilisés :

Mode	Signification
`"r"`	Lecture (mode par défaut)
`"w"`	Écriture (écrase le fichier s'il existe)
`"a"`	Ajout de données à la fin du fichier
`"x"`	Création d'un nouveau fichier (échoue si le fichier existe déjà)
`"r+"`	Lecture et écriture
`"w+"`	Lecture et écriture (écrase le fichier)
`"a+"`	Lecture et ajout

Exemple d'ouverture d'un fichier en mode lecture :

```
fichier = open("exemple.txt", "r")  # Ouvre le fichier en lecture
```

2.2. Fermer un fichier

Une fois que l'on a fini d'utiliser un fichier, il est important de le fermer pour éviter des fuites de mémoire ou des erreurs.

```
fichier.close()
```

Une méthode plus propre consiste à utiliser le mot-clé `with`, qui assure que le fichier est automatiquement fermé à la fin du bloc :

```
with open("exemple.txt", "r") as fichier:
    contenu = fichier.read()  # Lit le contenu du fichier
# Le fichier est automatiquement fermé après ce bloc
```

---

3. Lire un fichier en Python

Python offre plusieurs façons de lire un fichier.

3.1. Lire tout le contenu avec `read()`

```
with open("exemple.txt", "r") as fichier:
    contenu = fichier.read()
    print(contenu)
```

1. Le fichier `exemple.txt` est ouvert en mode lecture (`"r"`).
2. La méthode `.read()` lit tout le fichier et stocke son contenu dans la variable `contenu`.
3. `print(contenu)` affiche le contenu du fichier.

3.2. Lire une seule ligne avec `readline()`

```
with open("exemple.txt", "r") as fichier:
    ligne = fichier.readline()
    print(ligne)
```

1. Le fichier est ouvert en mode lecture.
2. La méthode `.readline()` lit **seulement** la première ligne.
3. `print(ligne)` affiche cette ligne.

3.3. Lire toutes les lignes avec `readlines()`

```
with open("exemple.txt", "r") as fichier:
    lignes = fichier.readlines()
    print(lignes)
```

1. Le fichier est ouvert en mode lecture.
2. `.readlines()` stocke toutes les lignes du fichier sous forme de **liste**.
3. `print(lignes)` affiche cette liste où chaque élément est une ligne du fichier.

---

4. Écrire dans un fichier en Python

4.1. Écriture avec `write()`

```
with open("exemple.txt", "w") as fichier:
    fichier.write("Première ligne du fichier.\n")
    fichier.write("Deuxième ligne ajoutée.")
```

1. Le fichier est ouvert en mode écriture (`"w"`). **Attention : ce mode efface le contenu existant !**
2. `.write()` permet d'écrire du texte dans le fichier.
3. `\n` est utilisé pour ajouter un retour à la ligne.

4.2. Ajouter du contenu avec `a` (append mode)

```
with open("exemple.txt", "a") as fichier:
    fichier.write("\nNouvelle ligne ajoutée à la fin.")
```

1. Le fichier est ouvert en mode ajout (`"a"`).
2. `.write()` ajoute du texte à la fin du fichier sans effacer le contenu existant.

---

5. Lire un fichier ligne par ligne

```
with open("exemple.txt", "r") as fichier:
    for ligne in fichier:
        print(ligne.strip())  # Enlève les espaces et sauts de ligne
```

1. On ouvre le fichier en mode lecture.
2. On parcourt chaque ligne avec `for ligne in fichier`.
3. `.strip()` permet de supprimer les espaces et sauts de ligne superflus.

---

6. Gestion des erreurs lors de la manipulation des fichiers

6.1. Gérer les erreurs avec `try-except`

```
try:
    with open("fichier_inexistant.txt", "r") as fichier:
        contenu = fichier.read()
except FileNotFoundError:
    print("Erreur : Le fichier n'existe pas.")
except IOError:
    print("Erreur : Problème lors de la lecture du fichier.")
```

1. Le programme tente d'ouvrir un fichier qui n'existe pas.
2. Si le fichier est introuvable, `FileNotFoundError` est levé et un message est affiché.
3. `IOError` capture d'autres erreurs liées à l'accès au fichier.

---

7. Exercice corrigé : Analyse d'un fichier texte

Énoncé :

Créer un programme qui lit un fichier `data.txt`, compte le nombre de lignes, le nombre de mots et le nombre de caractères, puis affiche ces informations.

Solution :

Étape 1 : Ouvrir et lire le fichier

```
with open("data.txt", "r") as fichier:
    contenu = fichier.readlines()
```

Étape 2 : Compter les lignes, mots et caractères

```
nombre_lignes = len(contenu)
nombre_mots = sum(len(ligne.split()) for ligne in contenu)
nombre_caracteres = sum(len(ligne) for ligne in contenu)
```

Étape 3 : Afficher les résultats

```
print("Nombre de lignes :", nombre_lignes)
print("Nombre de mots :", nombre_mots)
print("Nombre de caractères :", nombre_caracteres)
```

---

8. Exercices à faire

Exercice 1 : Lecture d'un fichier texte

Écrire un programme qui ouvre un fichier nommé `texte.txt` et affiche son contenu ligne par ligne.

Exercice 2 : Comptage des lignes d'un fichier

Créer un programme qui lit un fichier et affiche le nombre total de lignes qu'il contient.

Exercice 3 : Ajout de texte dans un fichier

Créer un programme qui demande à l'utilisateur d'entrer un texte et qui l'ajoute dans un fichier `notes.txt`.

Exercice 4 : Gestion des erreurs

Modifier un programme pour gérer les erreurs lorsque le fichier n'existe pas ou ne peut pas être ouvert.