Gestion des fichiers en Python

I. Introduction

II. Ouverture / Fermeture des fichiers

III. Lecture des fichiers

IV. Écriture dans les fichiers

V. Utilisation du bloc ‘with’

VI. Travailler avec différents formats de fichiers

VII. Gestion des erreurs

VIII. Conclusion

Radouan DAHBI

I.

Introduction

Introduction

La gestion des fichiers est une compétence essentielle en Python. Elle permet de lire, écrire et manipuler des données stockées dans des fichiers sur le disque dur. Que ce soit pour traiter des fichiers texte, des données structurées ou pour enregistrer les résultats d'un programme, savoir comment gérer les fichiers est indispensable pour tout développeur en Python.

Introduction

Pourquoi gérer les fichiers en Python ?

Les fichiers sont un moyen courant de stocker et d'échanger des données. En Python, la gestion des fichiers permet de :

- **Lire des données** depuis des fichiers pour les traiter ou les analyser.
- **Écrire des données** dans des fichiers pour les sauvegarder ou les partager.
- **Manipuler différents types de fichiers**, tels que les fichiers texte, CSV, JSON, etc.
- **Automatiser des tâches** liées au traitement de données, comme la génération de rapports ou la transformation de données.

Comprendre comment interagir avec les fichiers permet de développer des applications plus complètes et utiles.

II.

Ouverture / Fermeture des fichiers

Ouverture et Fermeture des fichiers

Pour travailler avec un fichier en **Python**, il faut d'abord l'**ouvrir** en utilisant la fonction intégrée open(). Cette fonction retourne un objet fichier qui peut être utilisé pour lire ou écrire dans le fichier. Une fois les opérations terminées, il est important de **fermer** le fichier en appelant la méthode close() pour libérer les ressources.

Syntaxe de base :

```
fichier = open('chemin/vers/fichier.txt', mode)
## Opérations sur le fichier
fichier.close()
```

Ouverture et Fermeture des fichiers

- Modes courants :
  - `'r'` : Lecture uniquement (erreur si le fichier n'existe pas).
  - `'w'` : Écriture (crée un nouveau fichier ou écrase l'existant).
  - `'a'` : Ajout de contenu à la fin du fichier.
  - `'b'` : Mode binaire pour les fichiers non textuels.

III.

Lecture des fichiers

Lecture des fichiers

Une fois le fichier ouvert en mode lecture (‘ **r** ’), plusieurs méthodes permettent de lire son contenu.

Lire tout le contenu

```
fichier = open('donnees.txt', 'r')
contenu = fichier.read()
print(contenu)
fichier.close()
```

Lire ligne par ligne

```
fichier = open('donnees.txt', 'r')
ligne = fichier.readline()
while ligne:
    print(ligne.strip())
    ligne = fichier.readline()
fichier.close()
```

Lecture des fichiers

Lire toutes les lignes

```
fichier = open('donnees.txt', 'r')
lignes = fichier.readlines()
for ligne in lignes:
    print(ligne.strip())
fichier.close()
```

Itérer sur le fichier

```
fichier = open('donnees.txt', 'r')
for ligne in fichier:
    print(ligne.strip())
fichier.close()
```

IV.

Écriture dans les fichiers

Écriture dans les fichiers

Pour écrire dans un fichier, il faut l'ouvrir en mode écriture (' **w** '), ajout (' **a** ') ou lecture-écriture (' **r+** ')

Écrire du texte

```
fichier = open('resultats.txt', 'w')
fichier.write('Ceci est un test.\n')
fichier.write('Écriture dans un fichier en Python.\n')
fichier.close()
```

Écrire plusieurs lignes

```
lignes = ['Ligne 1\n', 'Ligne 2\n', 'Ligne 3\n']
fichier = open('resultats.txt', 'w')
fichier.writelines(lignes)
fichier.close()
```

Écriture dans les fichiers

En ouvrant le fichier en mode ajout (‘ **a** ’), le nouveau contenu est ajouté à la fin du fichier sans effacer le contenu existant.

Ajouter du contenu à un fichier existant

```
fichier = open('resultats.txt', 'a')
fichier.write('Nouvelle ligne ajoutée.\n')
fichier.close()
```

V.

Utilisation du bloc ‘with’

Utilisation du bloc 'with'

L'utilisation du mot-clé with permet de gérer **automatiquement** l'ouverture et la fermeture du fichier, même en cas d'exception. Cela rend le code plus sûr et plus lisible.

Syntaxe :

```
with open('chemin/vers/fichier.txt', mode) as fichier:
    # Opérations sur le fichier
```

Le fichier est automatiquement fermé à la fin du bloc with.

VI.

Fichiers CSV

Fichiers CSV

Les fichiers CSV (Comma-Separated Values) sont couramment utilisés pour stocker des données tabulaires.

Lecture dans un fichier CSV :

```
import csv

with open('donnees.csv', 'r') as fichier_csv:
    lecteur = csv.reader(fichier_csv)
    for ligne in lecteur:
        print(ligne)
```

Écriture dans un fichier CSV :

```
import csv

donnees = [['Nom', 'Age'], ['Alice', '30'], ['Bob', '25']]

with open('resultats.csv', 'w', newline='') as fichier_csv:
    ecrivain = csv.writer(fichier_csv)
    ecrivain.writerows(donnees)
```

VII.

Gestion des erreurs

Gestion des erreurs

Lors de la **gestion des fichiers**, plusieurs **exceptions** peuvent survenir, comme FileNotFoundError, PermissionError, IOError. Il est important de gérer ces exceptions pour rendre le programme plus robuste.

Exemple de gestion d'exception :

```
try:
    with open('inexistant.txt', 'r') as fichier:
        contenu = fichier.read()
except FileNotFoundError:
    print("Erreur : le fichier n'a pas été trouvé.")
except PermissionError:
    print("Erreur : permissions insuffisantes pour ouvrir le fichier.")
except Exception as e:
    print(f"Erreur inattendue : {e}")
```

VIII.

Conclusion

Conclusion

- La gestion des fichiers est une compétence incontournable pour tout développeur Python.
- Utilisez le bloc with pour simplifier et sécuriser vos manipulations.
- Adoptez des pratiques robustes pour gérer les erreurs et éviter les problèmes liés aux fichiers.