Programmation en Python - Séance 6 : Gestion de fichiers — erreurs — exceptions

Hervé TALE KALACHI

ENSPY – Université de Yaoundé 1

Gestion de fichiers

Pourquoi lire et écrire des fichiers ?

- Sauvegarder les résultats d'un programme (ex : notes, logs, rapports...).
- Lire des données externes pour traitement (ex : CSV, texte, JSON...).
- Créer des applications interactives ou persistantes.

Types de fichiers les plus courants :

- **Texte** : .txt, .csv, .json, .py
- **Binaire** : .jpg, .mp3, .exe, etc.

En Python, la gestion de fichiers repose sur la fonction intégrée `open()` et le concept de **flux** (stream) : ouvrir $\rightarrow$ lire/écrire $\rightarrow$ fermer.

Ouvrir un fichier

Syntaxe générale :

```
f = open("nom_fichier.txt", mode)
# ... opérations ...
f.close()
```

Modes d'ouverture les plus utilisés :

- `'r'` → lecture (read)
- `'w'` → écriture (write, efface le contenu existant)
- `'a'` → ajout à la fin (append)
- `'r+'` → lecture et écriture

Exemple :

```
f = open("exemple.txt", "w")
f.write("Bonjour Python !\n")
f.close()
```

Astuce : toujours fermer le fichier après usage (`close()`).

Explorer et gérer les répertoires avec os

Objectif : savoir **où Python lit et écrit** les fichiers, et comment changer de dossier si nécessaire.

1) Trouver le répertoire courant :

```
import os

print(os.getcwd()) # get current working directory
```

2) Changer de répertoire courant :

```
os.chdir("/home/etudiant/Documents") # ou "C:\\Users\\Nom\\
    Documents"
print("Nouveau dossier :", os.getcwd())
```

3) Vérifier et créer un dossier :

```
if not os.path.exists("donnees"):
    os.mkdir("donnees") # cree le dossier s'il n'existe pas
```

Lire un fichier texte

Méthodes principales :

```
f = open("exemple.txt", "r")

contenu = f.read() # tout lire
print(contenu)

f.seek(0) # revenir au début
ligne = f.readline() # lire une seule ligne
lignes = f.readlines() # lire toutes les lignes (liste)

f.close()
```

Autre méthode (plus sûre) :

```
with open("exemple.txt", "r") as f:
    contenu = f.read()
    print(contenu)
# Fermeture automatique ici
```

Mot-clé `with` : garantit la fermeture automatique du fichier.

Écrire dans un fichier

Exemple simple :

```
with open("notes.txt", "w") as f:
    f.write("Alice : 15\n")
    f.write("Bob : 12\n")
    f.write("Claire : 17\n")
```

Ajouter sans écraser :

```
with open("notes.txt", "a") as f:
    f.write("David : 14\n")
```

Écrire à partir d'une liste :

```
etudiants = ["Alice\n", "Bob\n", "Claire\n"]
with open("noms.txt", "w") as f:
    f.writelines(etudiants)
```

Lecture ligne par ligne

Objectif : lire un fichier volumineux sans tout charger en mémoire.

```
with open("notes.txt", "r") as f:
    for ligne in f:
        print(ligne.strip()) # strip() enlève \n
```

Avantage :

- Lecture progressive (moins de mémoire utilisée).
- Parfait pour les fichiers log, CSV, ou grands textes.

Astuce : Vous pouvez combiner cette méthode avec des `split()` pour traiter chaque ligne.

Exemple pratique — gestion de notes

Programme complet :

```
# 1) Écriture
with open("notes.txt", "w") as f:
    f.write("Alice 15\nBob 12\nClaire 17\n")
# 2) Lecture et calcul
total = 0
n = 0
with open("notes.txt", "r") as f:
    for ligne in f:
        nom, note = ligne.split()
        total += float(note)
        n += 1
print("Moyenne :", total / n)
```

Résultat :

```
Moyenne : 14.67
```

Observation : on peut combiner lecture, boucle et conversion de types.

Erreurs et exceptions — introduction

Problème : Lorsqu'une erreur se produit, le programme s'arrête brutalement. Python lève alors une **exception**.

Exemples fréquents :

- `ZeroDivisionError` — division par zéro.
- `ValueError` — valeur invalide.
- `TypeError` — type non compatible.
- `FileNotFoundError` — fichier introuvable.
- `IndexError` — indice hors limite.

Objectif : Apprendre à **intercepter** ces erreurs pour éviter l'arrêt du programme et afficher un message explicite.

Structure de base : try / except

Syntaxe générale :

```
try:
    # bloc de code àrisque
except NomException:
    # traitement de l'erreur
```

Exemple :

```
try:
    x = int(input("Entrez un entier : "))
    print(10 / x)
except ZeroDivisionError:
    print("Erreur : division par zéro interdite.")
except ValueError:
    print("Erreur : entrée non valide.")
```

Résultat : le programme ne s'arrête pas, il gère l'erreur proprement.

try / except / else

Structure complète :

```
try:
    # bloc risqué
except TypeErreur:
    # gestion spécifique
except Exception:
    # gestion générale
else:
    # exécuté si aucune exception n'est levée
```

Exemple :

```
try:
    n = int(input("Entrez un nombre positif : "))
    assert n >= 0
except AssertionError:
    print("Erreur : le nombre doit être positif.")
else:
    print("Merci, nombre accepté :", n)
```

Le bloc `finally`

But : exécuter du code dans tous les cas, qu'il y ait erreur ou non.

```
try:
    f = open("exemple.txt", "r")
    contenu = f.read()
except FileNotFoundError:
    print("Le fichier est introuvable.")
finally:
    print("Fermeture du programme.")
```

Résultat : le bloc `finally` s'exécute toujours — utile pour libérer des ressources (fichiers, connexions, etc.)

Lever une exception manuellement

Mot-clé : `raise` permet de déclencher une exception de façon contrôlée.

```
def racine(x):
    if x < 0:
        raise ValueError("x doit être positif")
    return x ** 0.5

try:
    print(racine(-9))
except ValueError as e:
    print("Erreur détectée :", e)
```

Utilité :

- Pour imposer des contraintes dans une fonction.
- Pour signaler des cas non autorisés dans un code métier.

Créer une exception personnalisée

Définition :

```
class ErreurNote(Exception):
    """Exception levée si la note est hors limites."""
    pass

def saisir_note(note):
    if note < 0 or note > 20:
        raise ErreurNote("Note invalide (doit être entre 0 et 20)
            .")
    print("Note enregistrée :", note)

try:
    saisir_note(25)
except ErreurNote as e:
    print("Erreur :", e)
```

Avantage : permet de distinguer clairement vos propres erreurs des exceptions standard de Python.

Gestion d'erreurs — fichier absent

Exemple avec `try/except` :

```
try:
    with open("inexistant.txt", "r") as f:
        contenu = f.read()
except FileNotFoundError:
    print("Le fichier n'existe pas.")
```

Autres exceptions possibles :

- `PermissionError` — droits insuffisants.
- `UnicodeDecodeError` — problème d'encodage.
- `IOError` — erreur générale d'entrée/sortie.

Astuce : toujours entourer vos lectures/écritures d'un bloc `try/except`.

À vous de jouer — exercices sur la gestion de fichiers

```
# A) Créez un fichier "data.txt" contenant 5 prénoms.
# Puis lisez et affichez chaque prénom.

# B) Écrivez un programme qui demande àl'utilisateur
# d'entrer des phrases et les sauvegarde dans "journal.txt".

# C) Àpartir d'un fichier "notes.txt", calculez la moyenne des
    notes.

# D) Copiez le contenu d'un fichier source vers un fichier
    destination.

# E) Écrivez un script qui compte le nombre de lignes, de mots
# et de caractères dans un fichier texte.
```

Astuce : utilisez `split()`, `len()` et `with open(...)`.

À vous de jouer — exercices sur les exceptions

```
# A) Écrivez un programme qui demande un entier
# et affiche sa racine carrée, avec gestion de ValueError.

# B) Écrivez une fonction division(a, b)
# qui renvoie a/b mais gère la division par zéro.

# C) Créez une fonction lire_fichier(nom)
# qui gère FileNotFoundError et renvoie None si absent.

# D) Modifiez la fonction factorielle(n)
# pour lever une exception si n est négatif.

# E) Créez une exception personnalisée AgeInvalide
# pour refuser un âge inférieur à0 ou supérieur à130.
```

Astuce : combinez `try` / `except` / `else` / `finally` pour des scripts robustes.