Gestion des erreurs et des exceptions

Objectifs :

1. Comprendre l'importance de **gérer les erreurs** dans un programme.
2. Apprendre à utiliser les **blocs try/except** pour capturer et gérer les exceptions.
3. Apprendre à **personnaliser** les messages d'erreur pour rendre le programme plus robuste et convivial.
4. Utiliser **finally** pour exécuter du code quelle que soit la survenance ou non d'une exception.

Table des matières

1. Gestion des erreurs et des exceptions en Python....1
a) Pourquoi c'est important ?....1
b) Structure générale....1
c) Types d'erreurs courantes....2
d) Exemple simple....2
e) Concepts abordés :....2
f) Exercices pour cette séance :....2
2. Exercices....2
a) Exercice 1 – Conversion robuste....2
b) Exercice 2 – Accès à une liste....2
c) Exercice 3 – Ouverture de fichier....2
3. MINI-PROJET – Lecteur de notes sécurisé....3
a) Objectif....3
b) Exemple de fichier notes.txt....3
c) Tâches....3
Piste de solution....3

1. Gestion des erreurs et des exceptions en Python

a) Pourquoi c'est important ?

La gestion des erreurs permet :

- De rendre un programme plus **robuste** et fiable.
- D'éviter que des **erreurs** inattendues ne fassent planter l'application.
- D'offrir à l'utilisateur des **messages clairs** en cas de problème.

Lorsqu'une erreur survient (fichier introuvable, division par zéro, mauvaise saisie...), le programme plante. Pour éviter cela, on entoure les instructions risquées d'un bloc `try/except`.

b) Structure générale

```
try:
    # Instructions qui peuvent provoquer une erreur
except NomDeLErreur:
    # Code exécuté si une erreur se produit
else:
    # (facultatif) Code exécuté s'il n'y a pas eu d'erreur
finally:
    # (facultatif) Code toujours exécuté, erreur ou pas
```

c) Types d’erreurs courantes

Exception	Cause typique
`ZeroDivisionError`	Division par 0
`ValueError`	Mauvaise conversion de type
`TypeError`	Mauvais type de variable
`FileNotFoundError`	Fichier inexistant
`IndexError`	Accès hors limite à une liste

d) Exemple simple

```
try:
    x = int(input("Entrez un nombre : "))
    print(10 / x)
except ZeroDivisionError:
    print("Impossible de diviser par zéro.")
except ValueError:
    print("Veuillez entrer un nombre entier valide.")
finally:
    print("Fin du traitement.")
```

e) Concepts abordés :

- **try** : tente d'exécuter un bloc de code.
- **except** : capture l'erreur si le bloc `try` échoue.
- **else** : code exécuté si aucune exception n’est levée.
- **finally** : code exécuté quoiqu'il arrive, idéal pour fermer des fichiers, connexions réseau, etc.

f) Exercices pour cette séance :

- Utiliser `try/except` pour gérer une division, une conversion de types, ou une lecture de fichier.
- Gérer plusieurs types d’exceptions (par exemple, `ValueError`, `FileNotFoundError`).
- Ajouter un bloc `else` pour gérer le cas où aucune erreur ne survient.

2. Exercices

a) Exercice 1 – Conversion robuste

Demande un nombre entier à l’utilisateur et affiche son double. Gère les cas où l’utilisateur ne saisit pas un nombre.

b) Exercice 2 – Accès à une liste

Soit une liste de 5 noms. Demande à l’utilisateur un index et affiche l’élément correspondant. Gère les erreurs si l’index est hors limites ou invalide.

c) Exercice 3 – Ouverture de fichier

Demande un nom de fichier à l’utilisateur, puis affiche son contenu. Gère l’erreur si le fichier n’existe pas.

3. MINI-PROJET – Lecteur de notes sécurisé

a) Objectif

Lire un fichier `notes.txt` contenant une note par ligne. Calculer la moyenne. Le fichier peut contenir des erreurs (texte à la place d'un nombre, lignes vides...).

b) Exemple de fichier `notes.txt`

```
14.5
abc
16
17.5
```

c) Tâches

1. Lire le fichier.
2. Tenter de convertir chaque ligne en `float`.
3. Ignorer les lignes invalides (en signalant l'erreur).
4. Calculer et afficher la moyenne des notes valides.

Piste de solution

```
def lire_notes(fichier):
    notes = []
    try:
        with open(fichier, "r") as f:
            for ligne in f:
                try:
                    note = float(ligne.strip())
                    notes.append(note)
                except ValueError:
                    print(f"Erreur : '{ligne.strip()}' n'est pas une note
valide.")
    except FileNotFoundError:
        print("Fichier introuvable.")
        return []

    return notes

# Utilisation
notes = lire_notes("notes.txt")
if notes:
    print(f"Moyenne : {sum(notes)/len(notes):.2f}")
else:
    print("Aucune note valide à traiter.")
```