Cas pratique : Redirection de flux de sortie (`stdout`, `stderr`)

Un **context manager** peut être utilisé pour rediriger la sortie standard (`sys.stdout`) ou la sortie d'erreur (`sys.stderr`) vers un fichier ou une autre destination.

---

1. Rediriger `stdout` vers un fichier

```
import sys

class RedirectStdout:
    def __init__(self, filename):
        self.filename = filename
        self.file = None
        self.original_stdout = sys.stdout  # Sauvegarde du stdout origin

    def __enter__(self):
        """Redirige la sortie standard vers un fichier"""
        self.file = open(self.filename, "w")
        sys.stdout = self.file  # Redirection du stdout
        return self  # Permet d'accéder à l'instance dans le `with`

    def __exit__(self, exc_type, exc_value, traceback):
        """Restaure la sortie standard et ferme le fichier"""
        sys.stdout = self.original_stdout
        self.file.close()

# Utilisation :
with RedirectStdout("output.txt"):
    print("Ce message sera écrit dans output.txt")
    print("Même ce deuxième message !")

print("Ce message s'affiche normalement dans la console.")
```

✅ **Avantages** : - Toute sortie imprimée via `print()` sera enregistrée dans `output.txt`. - À la fin du bloc `with`, `sys.stdout` est restauré, donc `print()` redevient normal.

---

2. Redirection vers une variable (capture de stdout)

Si l'on veut capturer le texte imprimé **dans une variable**, voici une version modifiée :

```
import sys
import io

class CaptureStdout:
    def __enter__(self):
        """Capture le stdout dans une variable"""
        self.original_stdout = sys.stdout
        self.captured_output = io.StringIO()
        sys.stdout = self.captured_output
        return self  # Retourne l'objet pour accéder au texte

    def __exit__(self, exc_type, exc_value, traceback):
        """Restaure la sortie standard"""
        sys.stdout = self.original_stdout

    def get_output(self):
        """Retourne le texte capturé"""
        return self.captured_output.getvalue()

# Utilisation :
with CaptureStdout() as capture:
    print("Ceci est un test.")
    print("Capture réussie !")

result = capture.get_output()
print(f"Contenu capturé :\n{result}")
```

✅ **Pourquoi c'est utile ?** - Permet d'analyser ou stocker le contenu généré par `print()` . - Utile pour **tester** du code qui affiche du texte.

---

3. Rediriger `stderr` (erreurs) vers un fichier

De la même manière, on peut rediriger `sys.stderr` :

```
import sys

class RedirectStderr:
    def __init__(self, filename):
        self.filename = filename
        self.file = None
        self.original_stderr = sys.stderr

    def __enter__(self):
        self.file = open(self.filename, "w")
        sys.stderr = self.file
        return self

    def __exit__(self, exc_type, exc_value, traceback):
        sys.stderr = self.original_stderr
        self.file.close()

# Utilisation :
with RedirectStderr("errors.log"):
    print("Message normal")  # Ce message ira dans stdout (console)
    raise ValueError("Erreur simulée")  # Cette erreur sera écrite dans
```

✅ **Pourquoi c'est utile ?** - Permet de **logguer** toutes les erreurs dans un fichier. - Évite d'afficher des erreurs à l'écran dans une application.

---

4. Version avec `contextlib` (plus concis)

On peut simplifier cela avec `contextlib.redirect_stdout()` et `contextlib.redirect_stderr()` :

```
from contextlib import redirect_stdout, redirect_stderr

# Redirection stdout vers un fichier
with open("output.txt", "w") as f:
    with redirect_stdout(f):
        print("Ce message va dans output.txt")

# Redirection stderr vers un fichier
with open("errors.log", "w") as f:
    with redirect_stderr(f):
        raise RuntimeError("Ceci est une erreur capturée.")
```

✅ **Pourquoi l'utiliser ?** - Évite d'écrire une classe pour les cas simples. - Solution **plus rapide et lisible**.

📌 Conclusion

Méthode	Cas d'utilisation	Avantages	Inconvénients
Classe `RedirectStdout`	Rediriger `stdout` vers un fichier	Personnalisable, extensible	Plus de code qu'avec `contextlib`
Classe `CaptureStdout`	Capturer `stdout` dans une variable	Idéal pour les tests	Utilisation spécifique
Classe `RedirectStderr`	Rediriger `stderr` (erreurs)	Utile pour logguer les erreurs	Moins utile pour des erreurs occasionnelles
`contextlib.redirect_stdout()`	Redirection simple	Très concis et efficace	Moins flexible