🐍 Cheatsheet : Gestion des Exceptions Python 🐍

Ceci est un guide rapide pour maîtriser les mots-clés try, except, finally, et raise pour écrire du code robuste.

Mot-Clé	Rôle (Concept Usine à Cookies)	Syntaxe de Base	Principe d'Expert
try	**L'Étape Risquée.** Contient le code qui pourrait potentiellement générer une erreur.	try:	La logique critique et risquée doit toujours être placée ici.
except	**Le Plan B / Réparation.** Se déclenche uniquement si une exception se produit dans le try correspondant.	except TypeDException:	**Soyez spécifique.** Ne "rattrapez" (catch) que les exceptions que vous savez gérer (ZeroDivisionError, FileNotFoundError, etc.).
except...as e	Récupérer l'information sur l'erreur (le message d'erreur).	except ValueError as e:	Permet de lire et de journaliser (log) le message exact de l'erreur (e).
finally	**Le Nettoyage Obligatoire.** Le code s'exécute **toujours**, qu'il y ait eu une erreur ou non.	finally:	**Utilisation clé :** Fermer des ressources (fichiers, connexions réseau) pour éviter les fuites, peu importe ce qui se passe.
raise	**Tirer l'Alarme.** Déclenche manuellement une	raise TypeDException("Message descriptif")	Créez vos propres exceptions personnalisées (ex:

	exception pour signaler une condition impossible ou invalide.		class CookieError(Exception): pass) pour une meilleure clarté.
Re-raise	**Passer la Patate Chaude.** Relancer l'exception courante à un niveau supérieur du code.	except ValueError: suivi de raise (sans argument)	Permet de faire un traitement local (logging, nettoyage) *avant* d'envoyer l'erreur au niveau supérieur pour une gestion globale.

🛠️ Modèles de Code Essentiels

1. Structure Complète

C'est le modèle le plus utilisé dans les librairies.

```
def lire_et_traiter(chemin_fichier):
    fichier = None # Initialiser la variable
    try:
        fichier = open(chemin_fichier, 'r')
        donnees = fichier.read()
        # Traitement des données (étape risquée)
        return donnees
    except FileNotFoundError:
        # Gère si le fichier n'existe pas
        print("Erreur : Fichier introuvable.")
        return None
    except Exception as e:
        # Gère toutes les autres erreurs imprévues
        print(f"Une erreur inattendue est survenue : {e}")
        raise # Relance l'erreur après l'avoir notée
    finally:
        # Assure la fermeture, même en cas d'erreur
        if fichier:
            fichier.close()
            print("Le fichier est bien fermé (finally).")
```

2. Relance d'Exception (Re-raise)

Pour tracer un problème tout en le signalant.

```
def fonction_enfant():
    try:
        # Code qui échoue ici
        resultat = 1 / 0
    except ZeroDivisionError as e:
        # 1. On effectue une action de journalisation locale
        print("LOG: Le calcul a échoué dans fonction_enfant.")
        # 2. On relance l'exception originale sans la modifier
        raise # TIRE LA SONNETTE D'ALARME !
```

3. Utilisation de with (Le meilleur finally)

En Python, pour les ressources qui doivent être fermées (fichiers, connexions), with est la meilleure pratique car il garantit l'appel implicite de finally.

```
# 'with' gère l'ouverture et la fermeture automatiquement
try:
    with open("donnees.txt", 'r') as f:
        contenu = f.read()
    # Le fichier est garanti d'être fermé ici,
    # même si une exception se produit après le 'with'

except IOError:
    print("Problème lors de la lecture/écriture.")
```