Gestion des exceptions en Python avec `try` et `except`

En Python, les exceptions sont des erreurs qui surviennent pendant l'exécution d'un programme. Pour éviter que ces erreurs ne provoquent l'arrêt brutal du programme, Python propose des structures comme `try` et `except` pour gérer ces situations de manière élégante.

Structure de base : `try` et `except`

La structure de base pour gérer les exceptions est la suivante :

```
try:
    # Code susceptible de provoquer une exception
    resultat = 10 / 0
except ZeroDivisionError:
    # Code exécuté si une exception de type ZeroDivisionError est levée
    print("Erreur : Division par zéro !")
```

Explication :

1. `try` : Contient le code qui pourrait générer une exception.
2. `except` : Capture et gère l'exception spécifique.

Gérer plusieurs types d'exceptions

Vous pouvez gérer différents types d'exceptions en utilisant plusieurs blocs `except` :

```
try:
    valeur = int(input("Entrez un nombre : "))
    resultat = 10 / valeur
except ValueError:
    print("Erreur : Vous devez entrer un nombre valide.")
except ZeroDivisionError:
    print("Erreur : Division par zéro interdite.")
```

Capturer toutes les exceptions

Pour capturer toutes les exceptions, utilisez un `except` sans spécifier de type :

```
try:
    resultat = 10 / 0
except:
    print("Une erreur est survenue.")
```

Cependant, il est recommandé de capturer des exceptions spécifiques pour éviter de masquer des erreurs inattendues.

Utilisation de `else`

Le bloc `else` s'exécute uniquement si aucune exception n'est levée dans le bloc `try` :

```
try:
    resultat = 10 / 2
except ZeroDivisionError:
    print("Erreur : Division par zéro.")
else:
    print(f"Résultat : {resultat}")
```

Utilisation de `finally`

Le bloc `finally` s'exécute toujours, qu'une exception soit levée ou non. Il est souvent utilisé pour libérer des ressources (fichiers, connexions, etc.) :

```
try:
    fichier = open("exemple.txt", "r")
    contenu = fichier.read()
except FileNotFoundError:
    print("Erreur : Fichier introuvable.")
finally:
    print("Fermeture du fichier.")
    fichier.close()
```

Lever une exception manuellement

Vous pouvez lever une exception avec l'instruction `raise` :

```python
def verifier_age(age):
    if age < 18:
        raise ValueError("L'âge doit être supérieur ou égal à 18.")
    return "Accès autorisé."

try:
    print(verifier_age(16))
except ValueError as e:
    print(f"Erreur : {e}")
```

Bonnes pratiques

1. **Précisez les exceptions** : Évitez d'utiliser un `except` générique.
2. **Soyez clair** : Fournissez des messages d'erreur explicites.
3. **Nettoyez les ressources** : Utilisez `finally` ou des gestionnaires de contexte (`with`) pour libérer les ressources.

Avec ces concepts, vous pouvez écrire un code Python robuste et capable de gérer les erreurs efficacement !