

Stéphane Guinard

Les exceptions

Comment gérer les erreurs en Python ?

Division des enseignements en informatique

Prenons un fichier contenant le code python suivant :

```
def une_fonction(a):  
    return 1 / a  
  
def une_autre_fonction():  
    une_fonction(0)  
  
une_autre_fonction()
```

Rappel : les erreurs

L'exécution de ce scripte provoque l'erreur suivante :

```
Traceback (most recent call last):
  File "script.py", line 7, in <module>
    une_autre_fonction()
  File "script.py", line 5, in une_autre_fonction
    une_fonction(0)
  File "script.py", line 2, in une_fonction
    return 1 / a
ZeroDivisionError: division by zero
```

Comment lire les erreurs

Un message d'erreur se lit de bas en haut ! Ainsi on peut remonter la pile des appels.

Sous l'erreur ci dessus, se cache en réalité une **exception** de type `ZeroDivisionError`.

Qu'est ce qu'une exception ?

Mécanisme pour gérer des erreurs survenues lors de l'exécution d'un programme.

Pourquoi faut-il gérer certaines exceptions ?

- Apporter une solution à un problème bloquant
- Éviter d'interrompre le programme

Il existe d'autres solutions, **mais c'est la manière de faire en Python** :

```
def une_fonction(a):  
    # Gestion de l'erreur en amont  
    if a == 0:  
        print("Division par 0 impossible !")  
        return None  
    else:  
        return 1 / a
```

Soulever une exception

Des exceptions sont automatiquement levées par les fonctions *built-in* de python, cependant, il est possible de **lever** ses propres exception à l'aide du mot-clef `raise`.

```
def ma_fonction(age):  
    if age < 0:  
        raise ValueError("'age' doit etre positif  
            !")  
    # suite de la fonction
```

```
>>> ma_fonction(-2)  
Traceback (most recent call last):  
  File "<interactive input>", line 1, in <module>  
  File "<interactive input>", line 3, in  
      ma_fonction  
ValueError: 'age' doit etre positif !
```

On utilise la combinaison de mots-clé `try/except` pour créer des blocs d'instructions, où les instructions pouvant mener à une erreur, pourront être gérées.

```
try:  
    # ce qui peut produire une exception  
except NomException:  
    # ce qu'il faut faire si l'exception se dé  
    clanche
```

Traiter une exception

On utilise la combinaison de mots-clef try/except pour créer des blocs d'instructions où les instructions pourront être gérées.

Exemple :

```
liste = ['toto', 'titi', 'tata'...]  
i = int(input())  
try:  
    choix = liste[10 // i]  
except ZeroDivisionError:  
    print("Division par zero impossible")  
    choix = liste[0]  
except IndexError:  
    print("Probleme d'index")  
    choix = liste[len(liste)]
```

Mots-clef `finally` et `else` :

```
try:
    # ce qui peut produire une exception
except NomException:
    # ce qu'il faut faire si l'exception se déclenche
else:
    # ce qu'il faut faire si aucune exception n'a été levée
finally:
    # ce qui sera exécuté dans tous les cas
```

Traiter une exception

Mots-clef finally et else :

Exemple :

```
try:
    f = open('fichier.txt', 'w')
    # écriture dans le fichier
except IOError:
    print("Probleme lors de l'écriture du fichier
          ")
else:
    print("Ecriture OK")
finally:
    f.close()
```

Astuce contre les oublis de fermeture, le mot-clef with :

```
try:
    with open('fichier.txt', 'w') as f:
        # écriture dans le fichier
except (IOError, FileNotFoundError):
    # gérer l'erreur
```

Exceptions fréquentes

- `NameError` : variable ou fonction manipulée non déclarée
- `TypeError` : type de la variable incohérent avec l'opération demandée
- `ValueError` : le type est correct, mais pas la valeur
- `ZeroDivisionError` : division par zéro
- `IndexError` / `KeyError` : tentative d'accès à une séquence/dictionnaire avec un indice/clé inexistant
- `FileNotFoundError` : le fichier n'existe pas
- `IOError` : erreur lors de la manipulation d'un fichier
- `SyntaxError` : erreur de syntaxe (indentation, parenthèse...)