NSI - première

Python 8 - Exceptions

qkzk

2021/04/27

Une *exception* est une erreur provoquée par Python.

C'est un mécanisme universel en programmation et il existe un moyen d'*attraper* ces exceptions afin d'éviter que le programme ne plante.

Principe des exception

Lorsque Python rencontre une instruction impossible il provoque une erreur. Plus précisément on dit qu'il lève une exception.

Par exemple lorsqu'on tente d'accéder à un élément dont l'indice n'existe pas :

```
>>> l = ["a", "b"]
>>> l[0]
'a'
>>>
>>> l[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Python lève l'exception `IndexError`

Il en existe beaucoup : `ValueError`, `RuntimeError`, `ZeroDivisionError`, `KeyError`, `FileNotFoundError`

Certaines sont plus surprenantes :

```
>>> while True:
...     print(1)
1
1
1
1
^CTraceback (most recent call last):
  File "<stdin>", line 2, in <module>
KeyboardInterrupt
```

J'ai appuyé sur CTRL + C pour arrêter l'exécution et Python lève l'exception `KeyboardInterrupt`

Le programme est-il foutu dès qu'une exception est rencontrée ?

Non... il est toujours possible d'éviter qu'il ne plante en *attrapant* cette exception.

`try... except`

La syntaxe est la suivante :

```
nombre = 0

try:
    print( 1 / nombre )

except ZeroDivisionError:

    print("division impossible")

print("cette ligne sera toujours atteinte")
```

Ici on tente une opération, elle est impossible... donc on attrape l'exception `ZeroDivisionError` et on affiche le second message.

Voici la sortie obtenue :

```
division impossible
cette ligne sera toujours atteinte
```

Lorsqu'aucune exception n'est précisée dans le bloc `except`, Python attrape toutes les erreurs. Ça va plus vite à écrire mais c'est une très mauvaise pratique.

On utilise un bloc `try` lorsqu'on souhaite éviter un problème particulier, pas dans tous les cas.

Lorsqu'un bloc `try` provoque une erreur, la partie du bloc `try` précédent l'erreur est toujours exécutée :

```
>>> a = 1
>>> try:
...   a = 0
...   1 / a
... except ZeroDivisionError:
...   print("impossible")
...
impossible
>>> a
0
```

Compléments

Les exceptions ne sont pas au programme mais vous risquez d'en rencontrer... et pourrez être tenté de les utiliser.

Néanmoins, Python propose d'autres éléments de syntaxe :

```
try:
  toujours_tenté
except MonException:
  fait_autre_chose
else:
  si_pas_derreur
finally:
  sera_toujours_execute
```

Cela peut s'avérer partique pour être certain que certaines instructions sont exécutées même si le programme plante. Par exemple pour se deconnecter proprement ou fermer un fichier etc.