Mécanisme de gestion des exceptions

Même si un code est syntaxiquement correct, il peut générer des erreurs. Quand Python rencontre une erreur lors de l'exécution d'un code, on dit qu'il **lève une exception**.

Cela génère l'affichage d'un message explicite et **l'arrêt** du programme si ces exceptions ne sont pas gérées.

Gérer les exceptions consiste à **anticiper les erreurs** (hors de contrôle du concepteur) de manière à ce que le programme ne s'arrête pas. Contrairement aux assertions, les exceptions permettent ainsi de ne pas planter le programme en proposant une alternative de traitement des cas problématiques avec les instructions try/except/else/finally. Elles doivent **demeurer dans le code final**.

En effet, la plupart des programmes ont vocation à être utilisés par d'autres utilisateurs que leur concepteur, et ceux-ci peuvent entrer des valeurs de variables provoquant des erreurs lors de l'exécution, même si le code du concepteur est initialement correct. Le concepteur doit anticiper ces erreurs dues à l'utilisateur.

Message d'erreur : Le message affiché contient le `traceback`, c'est à dire la pile d'appel — le chemin parcouru par l'interpréteur pour atteindre l'erreur (soit la liste des fonctions traversées pour atteindre l'erreur).
Ce message comporte le **type d'exception** levée (`ZeroDivisionError`, `IOError`, `NameError`, `SyntaxError, ValueError, TypeError`, etc.) et un **message** qui décrit le problème rencontré.

Remarque : Dans le cas d'une erreur de syntaxe, le message d'erreur indique même où est détectée l'erreur à l'aide d'une flèche (en fait, elle se trouve généralement juste avant l'endroit pointé par la flèche).

```
>>> 7 / 0
Traceback (most recent call last):
ZeroDivisionError: integer division or modulo by zero

>>> f = open('monfichier.txt','r')
Traceback (most recent call last):
IOError: [Errno 2] No such file or directory: 'monfichier.txt'

>>> 3 + message
Traceback (most recent call last):
NameError: name 'message' is not defined

>>> while True print('Coucou !')
Traceback (most recent call last):
File "tm_python", line 1

>>> while True print('Coucou !')
                 ^
SyntaxError: invalid syntax
```

Liste des exceptions courantes :

Nom de l'exception	description
AssertionError	levée par `assert condition` avec `condition == False`
ZeroDivisionError	division par 0
ImportError	Python ne parvient pas à charger un module
ModuleNotFoundError	Le module demandé est introuvable
NameError	Une variable est appelée avant d'avoir été définie
IndexError	Élément introuvable dans une liste
KeyError	Élément introuvable dans un dictionnaire
SyntaxError	La syntaxe est incorrecte. Le script s'arrête toujours
IndentationError	L'indentation est incorrecte. Le script s'arrête toujours
TypeError	On appelle une méthode / fonction pour un type qu'elle ne supporte pas

Exercice 1 : *Dans les lignes ci-dessous, un certain nombre de codes erronés sont écrits. Identifier l'exception correspondante parmi celles proposées ci-dessous et associées les au code.*

1.
```
somme = 0
L = [2, 5, '6', 9]
for element in L:
    somme = somme + element
```

2.
```
def div(x, y):
    return x/y

for i in range(5):
    print(div(5, i))
```

3.
```
x = 2
y = x + z
print(y)
```

4.
```
l = [1, 2, 3]
for i in range(5):
    print(l[i])
```

5.
```
age = "seize"
print(int(age))
```

6.
```
for i in range(5)
    print(i)
```

7.
```
open('mon_fichier')
```

a) SyntaxError: invalid syntax
b) ZeroDivisionError: division by zero
c) FileNotFoundError: [Errno 2] No such file or directory: 'mon_fichier'
d) IndexError: list index out of range
e) ValueError: invalid literal for int() with base 10: 'seize'
f) TypeError: unsupported operand type(s) for +: 'int' and 'str'
g) NameError: name 'z' is not defined

1 : … 2 : … 3 : ….

4 :… 5 : … 6 : …. 7 : ….

Pour éviter que le programme ne se termine définitivement ou qu'il se termine sans que la raison n'en soit explicite, il est conseillé de **gérer les exceptions**.

Comment gérer les exceptions?

Il suffit pour cela d'utiliser les instructions try/except et éventuellement else et finally.

- **Le bloc try** permet de tester un bloc de code et ne l'exécute que s'il ne contient aucune erreur. Dans le cas contraire, le programme ignore la totalité du code dans ce bloc et passe au bloc suivant except
- **Le bloc except** sera exécuté en cas d'erreur
- **Le bloc else** permet d'exécuter une action si aucune erreur ne survient dans le bloc try
- **Le bloc finally** vous permet d'exécuter du code, quel que soit le résultat des blocs try et except.

Exemple : structure complète possible

```
try:
  resultat = numerateur/denominateur
except NameError:
  print("La variable numerateur ou denominateur n'a pas été définie.")
except TypeError:
  print("La variable numerateur ou denomniateur possède un type incompatible avec la division")
except ZeroDivisionError:
  print("La variable denominateur est égale à 0.")
else:
  print("Le résultat est : ", resultat)
finally:
  print("Vous venez de calculer une division")
```

Exception utilisant l'instruction raise :

*On peut se demander maintenant **s'il est possible de lever une exception sans rencontrer une erreur**.*
*Exemple pour un programme qui demande à l'utilisateur de tapez son age et de lever une exception si l'**age est < 18 ans !** Bien entendu un age tapé qui est inférieur à 18 ans est une opération qui ne contient aucune erreur, et pourtant on peut quand même en lever une :*

```
try:
    age = int(input("Veuille saisir votre age"))
    if age < 18:
        raise ValueError
    else:
        print("age valide")
except :
    print("age non valide !")
```

A faire avec Spyder :

Exercice 2 : *Écrire un script qui demande un nombre et qui calcule sa racine carrée avec gestion des exceptions.*
Par exemple :
>>>
Entrer un nombre : go
go n'est pas un nombre valide !
Entrer un nombre : -5.26
-5.26 n'est pas un nombre positif !
Entrer un nombre : 16
La racine carrée de 16.0 est : 4.0
>>>

Exercice 3 : *Compléter le code suivant pour gérer les exceptions. De plus, on ne veut pas de notes négatives ou supérieur à 20.*

```
n1=int(input("note 1 ?\n"))
c1=int(input("coefficient 1 ?\n"))
n2=int(input("note 2? \n"))
c2=int(input("coefficient 2 ?\n"))
m=(n1*c1+n2*c2)/(c1+c2)
print(" la moyenne coefficientée est de", m)
```