Chapitre : les exceptions

Les exceptions sont les opérations qu'effectue un interpréteur ou un compilateur lorsqu' une erreur est détectée au cours de l'exécution d'un programme. En générale, l'exécution du programme est alors interrompue, et un message d'erreur plus ou moins explicite est affiché.

Exemple :

```
>>> print(7/0)
ZeroDivisionError: int division or modulo by zero
```

Propagation des exceptions

Lorsqu'une exception est déclenchée, le déroulement normal de la fonction en cours d'exécution est interrompu, et l'exception est transmise (ou propagée) à la fonction appelante, etc ...

Cette propagation peut cependant être stoppée par un traitement approprié qu'on nomme parfois aussi récupération.

Récupération d'une exception

Récupérer une exception revient à préciser le traitement qu'il convient d'effectuer dans cette situation exceptionnelle.

En Python, la syntaxe d'une récupération d'exception est de la forme

```
try:
   # Test d'instruction(s)
except TypeDInstruction:
   # Traitement en cas d'erreur
except TypeDInstruction:
   # Traitement en cas d'erreur
……
else :
   # Traitement en cas ou aucune erreur ne survient dans le bloc
finally:
   # Instruction(s) exécutée(s) qu'il y ait eu des erreurs ou non
```

Quelques exceptions en Python

ZeroDivisionError : déclenchée lors d'une tentative de division par 0 >>> 1 / 0 Traceback (most recent call last): File "<stdin>", line 1, in <module> ZeroDivisionError: division by zero	**NameError :** déclenchée lorsqu'un identificateur inconnu est trouvé >>> print (timoleon) Traceback (most recent call last): File "<stdin>", line 1, in <module> NameError**:** name 'timoleon' is not defined	**ValueError** : déclenchée lorsque dans une donnée d'un type correct ne permet pas l'évaluation d'une expression. >>> int (' a 12') Traceback (most recent call last): File "<stdin>", line 1, in <module> ValueError: invalid literal for int() with base 10: ' A 12'

TypeError : déclenchée lorsqu'une donnée est d'un mauvais type
>>> len (1)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError**:** object of type 'int' has no len()
	IndexError : déclenchée lorsqu'un indice est hors d'un intervalle autorisé
	>>> l = [1,2,3]
	>>> l[3]
	Traceback (most recent call last):
	File "<stdin>", line 1, in <module>
	IndexError: list index out of range
		ImportError : déclenchée lors d'une tentative d'importation d'un module inexistant.
		>>> import timoleon
		Traceback (most recent call last):
		File "<stdin>", line 1, in <module>
		ImportError: No module named 'timoleon'

FileNotFoundError : déclenchée lorsqu'une opération d'ouverture de fichier échoue.
>>> entree = open ('timoleon','r')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'timoleon'
	KeyError : déclenchée lors d'une tentative d'accès à la valeur d'un dictionnaire avec une clé inexistante.
	>>> d = {}
	>>> d['timoleon']
	Traceback (most recent call last):
	File "<stdin>", line 1, in <module>
	KeyError: 'timoleon'
		RutimeError : déclenchée lorsqu'une erreur est détectée à l'exécution d'un programme, erreur qui échappe à toutes les catégories précédentes.
		>>> from factorielle import fact
		>>> fact (1000)
		...
		RuntimeError: maximum recursion depth exceeded in comparison

Exemple :

```
def division(x,y):
    try:
        r = x / y
    except NameError:
        print("La variable x ou y n'a pas été définie.")
    except TypeError:
        print("type incompatible avec la division.")
    except ZeroDivisionError:
        print("La variable y est égale à 0.")
    else:
        print("résultat = ", r)
    finally :
        print("fin du programme")
```

Execution:
Division(5,2)
Résultat=2.5
fin du programme
Division(5,0)
La variable y est égale à 0
fin du programme
Division(5,'a')
type incompatible avec la division
fin du programme

Les assertions (assert)

Les assertions sont un moyen simple de s'assurer, avant de continuer, qu'une condition est respectée.

Exemple :

```
try:
        n = float(input("Saisissez une note :"))
        assert  20>=n>=0
except ValueError:
        print("Vous n'avez pas saisi un nombre.")
except AssertionError:
        print("svp une note entre 0 et 20.")
```

raise

```
annee = input("donner une année")
try:
        annee = int(annee)
        if annee<=0:
            raise Exception("l'année saisie est négative ou nulle")
except ValueError:
        print("La valeur saisie est invalide.")
```

Vos propres exceptions

Pour des raisons pratiques, vous serez peut-être amené(e) à faire vos propres exceptions.
Nous allons comprendre comment le faire en analysant ce code :

```
class MonException(Exception):
    def __init__(self,raison):
        self.raison = raison
    def __str__(self):
        return self.raison

def multiplierpar5(n):
    if n > 20:
        raise MonException("Le nombre est trop grand !")
    else:
        return n * 5
```

Finalement, cette fonction utilisée pourrait donner ceci :

```
try:
    print multiplier_par_5(2)
    print multiplier_par_5(21)
except MonException as  e:
    print (e)
```

Exécution
10
Le nombre est trop grand !