CHAPITRE 10 : GESTION DES EXCEPTIONS

I. Introduction :

Intéressons-nous au script suivant :

```
# script inverse.py
chaine = input('Entrer un nombre : ')
nombre = float(chaine)
inverse = 1.0/nombre
print("L'inverse de", nombre, "est :", inverse)
```

Ce script vous demande de saisir un nombre, puis il calcule et affiche son inverse.

II. Les exceptions

Quand vous entrez un nombre, tout se déroule normalement :

```
>>> Entrer un nombre : 10
L'inverse de 10.0 est : 0.1
```

Mais que se passe-t-il autrement ?

```
>>> Entrer un nombre : bonjour
Traceback (most recent call last):
  File "inverse.py", line 3, in <module>
    nombre = float(chaine)
ValueError: could not convert string to float: bonjour
>>>
>>>Entrer un nombre : 0
Traceback (most recent call last):
  File "inverse.py", line 4, in <module>
    inverse = 1.0/nombre
ZeroDivisionError: float division by zero
>>>
```

Python a détecté une erreur : une **exception est levée**.
Ici nous avons une exception de type **ZeroDivisionError** (division par 0) et une exception de type **ValueError**.
Une exception arrête l'exécution normale d'un programme.

III. Gestion des exceptions

Heureusement, il est possible de gérer les exceptions pour éviter l'arrêt brutal du programme. Par cela, on utilise conjointement les instructions **try** et **except**.

L'instruction **else** est optionnelle :

```
try:
   chaine = input('Entrer un nombre : ')
   nombre = float(chaine)
   inverse = 1.0/nombre
except:
   #ce bloc est exécuté si une exception est levée dans le bloc try
   print("Erreur !")
else:
   #on arrive ici si aucune exception n'est levée dans le bloc try
   print("L'inverse de", nombre, "est :", inverse)

>>> Entrer un nombre : 56
L'inverse de 56.0 est : 0.0178571428571
>>> Entrer un nombre : 0
Erreur !
```

On peut distinguer les différents types d'exceptions :

```
try:
   chaine = input('Entrer un nombre : ')
   nombre = float(chaine)
   inverse = 1.0/nombre
except ValueError:
   #ce bloc est exécuté si une exception de type ValueError est levée dans le bloc try
   print(chaine, "n'est pas un nombre !")
except ZeroDivisionError:
   #ce bloc est exécuté si une exception de type ZeroDivisionError est levée dans le bloc try
   print("Division par zéro !")
else:
   #on arrive ici si aucune exception n'est levée dans le bloc try
   print("L'inverse de", nombre, "est :", inverse)
>>> Entrer un nombre : 0
Division par zéro !
>>> Entrer un nombre : bonjour
bonjour n'est pas un nombre !
```

N'oubliez pas : un programme bien écrit doit gérer proprement les exceptions.

IV. Déclencher des Exceptions

L'instruction **raise** permet au programmeur de déclencher une exception. Par exemple:

```
>>> raise NameError, 'Coucou'
Traceback (innermost last):
 File "<stdin>", line 1
NameError: Coucou
```

Le premier argument de **raise** est le nom de l'exception qui doit être déclenchée. Le second argument (optionnel) spécifie l'argument de l'exception.

V. Exceptions Définies par l'Utilisateur

Les programmes peuvent nommer leurs propres exceptions en affectant une chaîne de caractères à une variable. Par exemple.

```
>>> mon_exc = 'mon_exc'
>>> try:
...     raise mon_exc, 2*2
...     except mon_exc, val:
...     print ('Mon exception a été déclenchée, valeur:', val)
...
Mon exception a été déclenchée, valeur: 4
>>> raise mon_exc, 1
Traceback (innermost last):
File "<stdin>", line 1
mon_exc: 1
```

Plusieurs modules standards s'en servent pour rapporter les erreurs qui peuvent survenir dans les fonctions qu'ils définissent.

VI. Définir les Actions de Nettoyage

L'instruction try admet une autre clause optionnelle qui permet de définir les actions de nettoyage qui doivent être exécutées impérativement. Par exemple:

```
>>> try:
...     raise KeyboardInterrupt
... finally:
...     print ('Adieu, monde!)'
..
Adieu, monde!
Traceback (innermost last):
  File "<stdin>", line 2
KeyboardInterrupt
```

Une *clause de finalisation* (clause *finally*) est exécutée qu'une exception ait eu lieu ou non dans la clause d'essai. Si une exception a été déclenchée, elle est déclenchée à nouveau après l'exécution de la clause de finalisation. La clause de finalisation est aussi exécutée ``en sortant'' lorsque l'instruction try est interrompue par les instructions break ou return.

Une instruction try doit avoir ou bien une ou plusieurs clauses d'exception, ou bien une clause de finalisation, mais pas les deux.

VII. Tableau recapitulatif :

Clause	*Interprétation*
except:	*intercepte tous les types d'exceptions*
except <exceptionType>:	*intercepte l'exception spécifique du type précisé*
except <exceptionType>, <value>:	*intercepte l'exception spécifique du type précisé avec sa donnée supplémentaire*
except (<exceptionType1>, <exceptionType2>):	*intercepte n'importe laquelle des exceptions listées*
else:	*exécute ce bloc si aucune interception n'est levée*
finally:	*applique toujours ce bloc*
raise :	*Provoquer une exception*