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.

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.

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 t
    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.

Exercices

Exercice 5.1

1. Compléter le script précédent de manière à ressaisir le nombre en cas d'erreur. Par exemple :

```
>>>
    Entrer un nombre : salut !
    salut ! n'est pas un nombre !
    Entrer un nombre : 2,3
    2,3 n'est pas un nombre !
    Entrer un nombre : 2.3
    L'inverse de 2.3 est : 0.434782608696
    >>>
```

2. Compléter le script de manière à accepter la virgule comme séparateur décimal. Par exemple :

```
>>>
    Entrer un nombre : 2,3
    L'inverse de 2.3 est : 0.434782608696
    >>>
```

   On pourra utiliser la méthode `replace()` de la classe `str`

Exercice 5.2 Ecrire un script qui calcule la racine carrée d'un nombre, 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 valide !
Entrer un nombre : 16
La racine carrée de 16.0 est : 4.0
>>>
```

Exercice 5.3 Soit le script :

```
i = 0
while True:
    i += 1
    print(i)
```

Il s'agit d'une boucle sans fin. Pour arrêter ce programme, il faut appuyer sur les touches CTRL + C, ce qui lève une exception de type `KeyboardInterrupt` :

```
>>>
1
...
1216
1217
1218
1219
1220
```

```
Traceback (most recent call last):
    raise KeyboardInterrupt
KeyboardInterrupt
>>>
```

Compléter le script pour gérer proprement l'exception :

```
>>>
1
...
966
967
968
Fin du programme
>>>
```

Webographie

- Documentation sur les exceptions