

Institut Préparatoire aux Études d'Ingénieurs de Tunis



Leila Ben Othman

Deuxième année de classes préparatoires

SM - SP

Année universitaire: 2017 - 2018

Gestion des exceptions

Plan

- ➊ Introduction
- ➋ Exemples
- ➌ Types d'erreurs
- ➍ La clause `try... except...else`
- ➎ Les assertions

Introduction

- En *PYTHON*, comme dans la majorité des langages de programmation, certaines instructions peuvent générer des erreurs.
- Lorsqu'une erreur est générée, cela se traduit par un message d'erreur ainsi que l'arrêt du programme.
- Afin d'éviter cet arrêt "*brusque*" du programme, on veut pouvoir continuer l'exécution en adaptant légèrement le traitement
⇒ On parle de gestion d'erreurs (gestion des exceptions).
- On dispose pour cela de la structure **try...except**.

Examples (1)

```
> ma_chaine='12abc'
```

```
> int(ma_chaine)
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ValueError: invalid literal for int() with base 10: '12abc'

```
> 25/0
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ZeroDivisionError: integer division or modulo by zero

Examples (2)

```
> L=[i*2 for i in range(10)]
> L
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
> L[len(L)]
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

```
> t=('IPEIT', 'Info', 'SM5', 'SM6')
> t[1]='Info16-17'
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Types d'erreurs

Différentes types d'erreurs (les plus courantes):

- `ZeroDivisionError`: se produit quand on effectue une division par 0;
- `SyntaxError`: se produit quand la syntaxe est erronée;
- `NameError`: se produit quand on utilise une variable qui n'existe pas;
- `TypeError`: se produit quand une opération est appliquée à un objet qui n'est pas de type adéquat;
- `IOError`: se produit quand une opération d'entrée-sortie échoue;
- `IndexError`: se produit si on cherche à accéder à un élément d'une séquence, en dehors des limites de celle-ci;
- `ImportError`: se produit quand l'import d'un module échoue.

`try... except...else`

```
try:
    .....
    Bloc d'instructions 1
    .....
except:
    .....
    Bloc d'instructions 2
    .....
else:
    .....
    Bloc d'instructions 3
    .....
```

```
try :
    x=int(input("Donner un entier "))
    y=1/x
except :
    print("Erreur")
else :
    print("L 'inverse est ",y)
```

⇒ Le bloc d'instructions dans la clause **else** ne sera exécutée que si aucune exception n'a été levée.

try... except...else

```
try:
    .....
    Bloc d'instructions 1
    .....
except TypeError1:
    .....
    Bloc d'instructions 2
    .....
except TypeError2:
    .....
    Bloc d'instructions 3
    .....
else:
    .....
    Bloc d'instructions 4
    .....
```

```
try :
    x=int(input("Donner un entier"))
    y=1/x
except ZeroDivisionError:
    print("Erreur - Division par zero")
except ValueError:
    print("Erreur - Entier non valide")
else :
    print("L' inverse est ",y)
```

try... except...else

On peut appliquer un même traitement à plusieurs exceptions.

⇒ `except TypeError1 , TypeError2, . . .`

```
try:
    .....
    Bloc d'instructions 1
    .....
except TypeError1, TypeError2:
    .....
    Bloc d'instructions 2
    .....
except TypeError3:
    .....
    Bloc d'instructions 3
    .....
```

try... except...else..finally

La clause **finally** permet d'exécuter du code quel que soit le résultat de l'exécution du bloc **try**.

```
try :  
    x=int(input("Donner un entier "))  
    y=1/x  
except ZeroDivisionError :  
    print("Erreur Division par zero")  
except ValueError :  
    print("Erreur Entier non valide")  
else :  
    print("L'inverse est",y)  
finally :  
    print('Au revoir')
```

Exemple

```
#Bloc 1  
try :  
    x=3+2  
except :  
    x=2+2  
    print( 'x=' ,x)  
else :  
    x=x+1  
    print( 'x=' ,x)  
finally :  
    print( 'Au_ revoir ')
```

```
#Bloc 2  
try :  
    x=x+2  
except :  
    x=2+2  
    print( 'x=' ,x)  
else :  
    x=x+1  
    print( 'x=' ,x)  
finally :  
    print( 'Au_ revoir ')
```

Les assertions

- Les assertions sont des prédicats devant être évaluées comme `True`. Dans le cas contraire, une exception **`AssertionError`** est levée pour indiquer que l'expression est fausse.

```
> assert x>0  
> assert x%2==0  
> assert len(L)>10
```

Exemple

```
def div_2(x):  
    #je ne veux traiter que les entiers pairs  
    assert x%2==0, 'Erreur: ┐entier ┐impair'  
    return (x/2)
```