

Informatique et Calcul Scientifique

Cours 6 : Exceptions

16.10.2024

Deux nouveaux types de structures de données :

- ▶ Les tuples, qui sont des objets indexés par position, ordonnés et non mutables
- ▶ Les dictionnaires, indexés par mot-clés, mutables, et utilisés pour hiérarchiser et structurer des données
- ▶ Les fonctions à nombre variable d'arguments.

Aujourd'hui, on verra..

- ▶ La gestion des exceptions en Python
- ▶ Une révision des notions Python vues aux cours 1 à 6.

Jusqu'à présent, nous avons été assez régulièrement confrontés à des erreurs lors de l'exécution du programme, qui entraînent son interruption. On parle alors d'**exceptions**.

- ▶ Il se peut que ces exceptions apparaissent lors de cas extrêmement particuliers et n'empêchent pas le fonctionnement général du programme. Il serait alors dommage de l'interrompre.
- ▶ Python possède un gestionnaire d'exceptions, permettant de faire fonctionner un code malgré la présence éventuelle d'exceptions et de traiter ces dernières.

Erreur VS exception

Il convient tout d'abord de faire la différence entre une erreur et une exception.

Une **erreur** correspond à un défaut de syntaxe qui empêche l'exécution du code

```
1 " 56 . )"
return 3
for i in range(3)
    print(i)
```

SyntaxError: invalid syntax

SyntaxError: 'return' outside function

SyntaxError: expected ':'

Une **exception** est une anomalie apparaissant lors de l'exécution du code, et entraînant son interruption.

```
x = 3/0
test = y
L = [1,2,3]
print(L[4])
```

ZeroDivisionError: division by zero

NameError: name 'y' is not defined

IndexError: list index out of range

Types d'exceptions

Nous avons déjà été confronté à plusieurs types d'exceptions différentes, mais il en existe bien plus¹.

Voici les principaux types d'exceptions auxquels nous serons fréquemment confrontés :

Exception	Description
IndexError	Quand l'indice demandé n'est pas atteignable.
KeyError	Quand la clé donnée n'existe pas dans le dictionnaire.
NameError	Quand un nom de variable est introuvable.
TypeError	Quand la variable n'a pas le type attendu.
ValueError	Quand la valeur d'une variable n'est pas valide.
ZeroDivisionError	Lors d'une division par zéro.

1. Une liste complète se trouve [ici](#).

Gestion des exceptions : try ... except

Pour gérer les exceptions, on utilise la syntaxe `try: ... except:`, qui est similaire à la structure conditionnelle `if ... else`.

S'agissant d'une structure de contrôle, il faut bien penser aux deux points et à l'indentation.

```
try:
    """Bloc d'instructions a tester"""
except:
    """instructions a effectuer si une exception
    est soulevee"""

print("suite du programme")
```

Les instructions à tester sont placées après le mot-clé `try:`.

- ▶ Si celles-ci soulèvent une exception, quelle qu'elle soit, le programme rentre dans le bloc `except:`.
- ▶ Dans le cas contraire, le programme est exécuté normalement.

Le mot-clé `try` doit toujours être suivi d'une clause `except`.

Exemples

```
try:
    x = [1,2,3,0]
    for i in x:
        print(3/i)
except:
    print("Division par zero")
print('Suite du programme')
```

Output :

3.0

1.5

1.0

Division par zero

Suite du programme

Exceptions spécifiques

Il est possible de gérer chaque type d'exception séparément, en la précisant à la suite du mot-clé `except`. Le bloc n'est exécuté que si le type d'exception précisé est soulevé.

Note : cette syntaxe implique bien sûr de connaître au préalable le type d'exception pouvant être soulevé.

```
try:
    x = float(input("Nombre x: "))
    y = float(input("Nombre y: "))
    print(f"{x/y = }")
except ValueError:
    print("Ce n'est pas un nombre!")
except ZeroDivisionError:
    print("Division par zero!")
```

Output :

Si x ou y pas
convertible en float :
>>> Ce n'est pas un
nombre!

Si y vaut 0 :
>>> Division par
zero!

Exceptions spécifiques

On peut même faire passer plusieurs exceptions dans le même bloc, en les mettant dans un `tuple`.

```
try:
#   y = 'texte'
#   y = 1
    x = y/0
except ZeroDivisionError:
    print('Division par zero')
except NameError:
    print('Nom non defini')
except (TypeError, ValueError):
    print('Type ou Value error')
```

Output :

Si y n'existe pas :

```
>>> Nom non defini
```

Si y est un str :

```
>>> Type ou Value error
```

Sinon :

```
>>> Division par zero
```

Forme générale

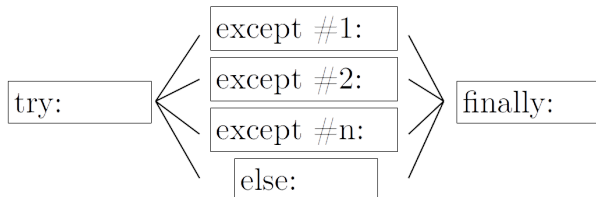
```
print("debut du programme")

try:
    """Bloc d'instructions a tester ,
    susceptible de soulever une exception"""
except (Exception1 , Exception2):
    """instructions a effectuer si une
    des exceptions 1 ou 2 est soulevee"""
except (Exception3 , Exception4):
    """instructions a effectuer si une
    des exceptions 3 ou 4 est soulevee"""
else:
    """bloc facultatif execute si
    aucun block except n'a ete
    execute"""
finally:
    """ce bloc facultatif est toujours
    execute , qu'une exception ait ete
    soulevee ou non"""

print("suite du programme")
```

Forme générale : explications

Bloc <code>try:</code>	Ce bloc est quitté dès qu'une ligne de code soulève une exception.
Blocs <code>except:</code>	Plusieurs blocs <code>except</code> peuvent exister, mais seul le bloc correspondant à la première exception soulevée sera exécuté.
Bloc <code>else:</code>	Bloc facultatif. Celui-ci n'est exécuté que si aucun bloc <code>except</code> a été exécuté.
Bloc <code>finally:</code>	Bloc facultatif, toujours exécuté après le traitement des exceptions.



Le bloc finally est **toujours** exécuté

Même si on est dans une fonction et qu'on arrive à une instruction `return` dans le bloc `try` ou `else` ou un bloc `except`, le bloc `finally` est exécuté avant de quitter la fonction.

```
def f():  
    try:  
        y = float(input("Nombre y: "))  
        print(f"1/y = {1/y}")  
    except ValueError:  
        print("Ce n'est pas un nombre!")  
        return -1  
    except ZeroDivisionError:  
        print("Division par zero!")  
        return -2  
    else:  
        return 0  
    finally:  
        print("FINALLY")  
  
print("la fonction retourne", f())
```

Output :

input : onze

Ce n'est pas un
nombre!

FINALLY

la fonction
retourne -1

Exemple

```
try:
    x = float(input("Quelle temperature
                    fait-il?"))
except ValueError:
    print("Vous n'avez pas rentre un
          nombre")
    print('Temperature par default : 20 C')
    x = 20
except TypeError:
    print("Erreur de type")
    print('Temperature par default : 20 C')
    x = 20
except:
    print("Un autre type d'exception")
    print('Temperature par default : 20 C')
    x = 20
else:
    print("Aucune exception")
finally:
    print(f"Il fait {x} degrees")
```

Output :

input : 25

Aucune
exception

Il fait 25.0
degrees

input : onze

Vous n'avez pas
rentre un
nombre

Temperature par
default : 20 C

Il fait 20
degrees

Exception personnalisée

Dans certains cas, il est pratique de définir soi-même une exception. Celle-ci peut correspondre à un cas particulier, ou à une situation non-physique par exemple.

Pour générer une exception personnalisée, on utilise le mot-clé `raise` dans le bloc `try`.

```
try:
    """instructions a tester. S'il
    y a un probleme, on peut soulever
    une exception personnalisée"""
    raise Exception("Description de l'exception")
except Exception:
    """Traitement de l'exception"""
```

L'exception peut-être quelconque (`Exception`), mais on peut également préciser son type à la suite du mot-clé `raise`.

Par exemple :

```
raise TypeError("Les types ne sont pas compatibles")
```

Exception personnalisée

On peut ainsi modifier l'exemple précédent pour ignorer les températures non-physiques.

```
try:
    x = float(input("Quelle temperature fait-il?"))
    if x < -273.15 or x > 100:
        raise Exception("Temperature non physique")
except ValueError:
    print("Vous n'avez pas rentre un nombre")
    print("Temperature par default : 20 C")
    x = 20
except:
    if x < -273.15:
        print("Il fait plus froid que le zero absolu !!")
    else:
        print("Il ne peut pas faire aussi chaud !")
    print("Temperature par default : 20 C")
    x = 20
else:
    print("Aucune exception")
finally:
    print(f"Il fait {x} degrees")
```

input : 30

Aucune exception

Il fait 30.0 degrees

input : -400

Il fait plus froid que le zero absolu !!

Temperature par default : 20 C

Il fait 20 degrees

input : 600

Il ne peut pas faire aussi chaud !

Temperature par default : 20 C

Il fait 20 degrees

Exception personnalisée - encore un exemple

On génère une exception si l'utilisateur entre un nombre négatif.

```
x = int(input("donnez moi un nombre positif "))  
if x < 0:  
    raise Exception("j'ai dit positif!")
```

donnez moi un nombre positif
-2
Exception: j'ai dit positif!

Et dans un try-except :

```
try:  
    x = int(input("donnez moi un nombre positif "))  
    if x < 0:  
        raise Exception("j'ai dit positif!")  
except ValueError:  
    print("ce n'est pas un nombre!")  
except:  
    print("votre nombre est negatif je crois")  
print("je continue")
```

donnez moi un nombre positif
aaa
ce n'est pas un nombre!
je continue

donnez moi un nombre positif
-2
votre nombre est négatif je
crois
je continue

donnez moi un nombre positif
4
je continue