22_python_exceptions

December 15, 2023

1 Python Exceptions

Une exception est une erreur qui se produit lors de l'exécution d'un programme. Lorsqu'une exception se produit, le programme s'arrête et affiche un message d'erreur.

[4]:
```
7 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[4], line 1
----> 1 7 / 0

ZeroDivisionError: division by zero
```

Le code ci-dessus génère une erreur de type `ZeroDivisionError`, car il est impossible de diviser par zéro.

1.1 Python Logical Errors (Exceptions)

Les exceptions sont des erreurs qui surviennent au `runtime`(apres la compilation).

Exemples d'exceptions courantes:

- `FileNotFoundError`: fichier inexistant, quand on essaie d'ouvrir un fichier qui n'existe pas
- `ImportError`: module inexistant, quand on essaie d'importer un module qui n'existe pas
- `IndexError`: index inexistant, quand on essaie d'accéder à un élément d'une liste qui n'existe pas
- `KeyError`: clé inexistante, quand on essaie d'accéder à une clé d'un dictionnaire qui n'existe pas
- `NameError`: variable inexistante, quand on essaie d'accéder à une variable qui n'existe pas
- `TypeError`: type inattendu, quand on essaie d'utiliser un type qui n'est pas attendu
- `ValueError`: valeur inattendue, quand on essaie d'utiliser une valeur qui n'est pas attendue
- `ZeroDivisionError`: division par zéro, quand on essaie de diviser par zéro

Lorsqu'une exception se produit, `Python`crée un `Objet` de type `Exception`. Si l'exception n'est pas gérée, le programme s'arrête et affiche un message d'erreur (`traceback`).

1.2 Buikt-in Exceptions

`Python` a un ensemble d'exceptions intégrées qui sont déclenchées lorsque des erreurs spécifiques se produisent. Par exemple:

Exception	Cause de l'erreur
`AssertionError`	Lorsqu'une instruction `assert` échoue.
`AttributeError`	Lorsqu'un attribut d'un objet n'existe pas.
`EOFError`	Lorsque la fonction `input()` atteint la fin du fichier.
`FloatingPointError`	Lorsqu'une erreur de point flottant se produit.
`GeneratorExit`	Lorsqu'une instruction `generator.close()` est utilisée.
`ImportError`	Lorsqu'un module d'importation n'est pas trouvé.
`IndexError`	Lorsqu'un index de séquence n'est pas valide.
`KeyError`	Lorsqu'une clé de dictionnaire n'est pas trouvée.
`KeyboardInterrupt`	Lorsqu'une instruction d'interruption est utilisée.
`MemoryError`	Lorsqu'une opération manque de mémoire.
`NameError`	Lorsqu'un nom ou une variable n'est pas trouvé dans l'espace de noms.
`NotImplementedError`	Lorsqu'une fonctionnalité n'est pas implémentée (encore).
`OSError`	Lorsqu'une fonctionnalité liée au système d'exploitation échoue.
`OverflowError`	Lorsqu'un résultat numérique est trop grand.
`ReferenceError`	Lorsqu'une référence de variable n'est pas trouvée.
`RuntimeError`	Lorsqu'une erreur inattendue se produit.
`StopIteration`	Lorsqu'une instruction `next()` d'un générateur ne renvoie pas de valeur.
`SyntaxError`	Lorsqu'une instruction ne respecte pas la syntaxe.
`IndentationError`	Lorsqu'une instruction n'est pas indentée correctement.
`TabError`	Lorsqu'une instruction n'est pas indentée correctement.
`SystemError`	Lorsqu'une erreur interne se produit.
`SystemExit`	Lorsqu'une instruction `sys.exit()` est utilisée.
`TypeError`	Lorsqu'une fonctionnalité ou une opération est appliquée à un type incorrect.
`UnboundLocalError`	Lorsqu'une variable locale n'est pas trouvée.
`UnicodeError`	Lorsqu'une erreur Unicode se produit.
`UnicodeEncodeError`	Lorsqu'une erreur Unicode se produit lors du codage.

Exception	Cause de l'erreur
`UnicodeDecodeError`	Lorsqu'une erreur Unicode se produit lors du décodage.
`UnicodeTranslateError`	Lorsqu'une erreur Unicode se produit lors de la traduction.
`ValueError`	Lorsqu'une fonctionnalité ou une opération reçoit un argument avec le bon type mais une valeur inappropriée.
`ZeroDivisionError`	Lorsqu'une division ou un modulo par zéro est effectué pour tous les types numériques.

1.3 Catching Exceptions

Pour gérer les exceptions, on utilise l'instruction `try` et `except`.

```
try:
    # code susceptible de générer une exception
except:
    # code à exécuter si une exception se produit
```

L'instruction `try` permet de tester une instruction susceptible de générer une exception. Si aucune exception ne se produit, le bloc `except` est ignoré. Si une exception se produit, le bloc `except` est exécuté.

[9]:
```
x = 0
try:
    x = 1 / 0
except:
    print("Une erreur s'est produite")
print(x)
print("hello")
```

```
Une erreur s'est produite
0
hello
```

Ici, Nous essayon de faire une division par `0`. Comme on ne peut pas diviser par `0`, une exception de type `ZeroDivisionError` est générée. Mais cette fois ci, le programme ne s'arrête pas, car l'exception est gérée par l'instruction `try` et `except`. Le bloc `except` est exécuté et le message d'erreur est affiché. Le programme continue son exécution.

1.4 Gérer les exceptions spécifiques

On peut gérer les exceptions spécifiques en utilisant l'instruction `except` avec le type d'exception.

```
try:
    # code susceptible de générer une exception
except ZeroDivisionError:
    # code à exécuter si une exception de type ZeroDivisionError se produit
```

Ici, on gère l'exception de type `ZeroDivisionError` en utilisant l'instruction `except` avec le type d'exception. Si une exception de type `ZeroDivisionError` se produit, le bloc `except` est exécuté, le programme suit son cours. Si une exception de type différent se produit, le bloc `except` n'est pas exécuté et l'exécution du programme s'arrête.

[33]:
```
d = {}
try:
    #f = open("salut.txt")
    x2 = 1 / 1
    x = [2, 4, 6, 8]
    print(x[4])
    if "antonio" in d:
        print(d["antonio"])
except (ZeroDivisionError, IndexError) as err:
    print("Error occured:", err)
except FileNotFoundError:
    print("cree le fichier d'abord")


print("Fin du programme")
```

```
Error occured: list index out of range
Fin du programme
```

1.5 Python try with else clause

On peut utiliser l'instruction `else` avec l'instruction `try` pour exécuter un code spécifique si aucune exception n'est générée.

```
try:
    # code susceptible de générer une exception
except:
    # code à exécuter si une exception se produit
else:
    # code à exécuter si aucune exception ne se produit
```

Ici, on utilise l'instruction `else` avec l'instruction `try`. Si aucune exception ne se produit, le bloc `else` est exécuté. Si une exception se produit, le bloc `else` n'est pas exécuté.

[29]:
```
# program to print the reciprocal of even numbers

try:
    num = int(input("Enter an even number: "))
    assert num % 2 == 0
except:
    print("Not an even number!")
else:
    reciprocal = 1/num
    print(reciprocal)
```

```
print("the program continues")
```

```
Enter an even number: 13
Not an even number!
the program continues
```

> **Note**: Si une exception se produit, le bloc `else` n'est pas gérée par les instructions `except` précédentes.

1.6 Python try with finally clause

On peut utiliser l'instruction `finally` avec l'instruction `try` pour exécuter un code spécifique, que l'exception soit générée ou non.

```
try:
    # code susceptible de générer une exception
except:
    # code à exécuter si une exception se produit
else:
    # code à exécuter si aucune exception ne se produit
finally:
    # code à exécuter dans tous les cas
```

Ici, on utilise l'instruction `finally` avec l'instruction `try`. Le bloc `finally` est toujours exécuté, que l'exception soit générée ou non. Si une exception se produit, le bloc `finally` est exécuté après le bloc `except`. Si aucune exception ne se produit, le bloc `finally` est exécuté après le bloc `else`.

[35]:
```
try:
    f = open("./files/test.txt")
except FileNotFoundError:
    print("File not found")
else:
    print(f.read())
finally:
    f.close()
```

```
This is a test file.
Hello from the test file.
Hello the course
```

1.7 Python Exception with Arguments

On peut passer des arguments à une exception pour définir des informations sur l'exception. Ceci peut être utile pour déterminer la cause de l'exception et la traiter en conséquence.

```
try:
    # code susceptible de générer une exception
except ExceptionType as Argument:
    # code à exécuter si une exception se produit
```

Ici, on utilise l'instruction `except` avec le type d'exception et l'argument. L'argument est lié à l'instance de l'exception et contient des informations sur l'exception.

[38]:
```
try:
    l = [1, 2, 3]
    print(l[4])
except IndexError as e:
    print("une erreur est survennue :", e)
```

```
une erreur est survennue : list index out of range
```

1.8 Python Custom Exceptions

On peut définir des exceptions personnalisées en définissant une classe qui hérite de la classe `Exception`.

Comme les exceptions sont des objets en python, il est possible de créer ses propres exceptions et de les lancer en utilisant l'instruction `raise`.

1.8.1 Defining Custom Exceptions

On peut définir des exceptions personnalisées en définissant une classe qui hérite de la classe `Exception`.

```
class CustomError(Exception):
    ...
    pass

try:
    ...

except CustomError:
    ...
```

Note: Les exceptions personnalisées doivent hériter de la classe `Exception` ou d'une classe dérivée de la classe `Exception`.

Note: Quand on dévelope une grande application en python, il est recommandé de définir ses propres exceptions pour gérer les erreurs spécifiques à l'application.

[42]:
```
# define Python user-defined exceptions
class InvalidAgeException(Exception):
    """Raised when the input age is below 18"""
    pass

# you need to guess this number
number = 18

try:
    input_num = int(input("Enter a number: "))
    if input_num < number:
```

```
        raise InvalidAgeException
    else:
        print("Eligible to Vote")

except InvalidAgeException:
    print("Exception occurred: Invalid Age")
```

```
Enter a number: 12
Exception occurred: Invalid Age
```

1.9 Customizing Exception Classes

On peut personnaliser les exceptions en définissant des attributs pour les classes d'exception.

```
class CustomError(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return repr(self.value)

try:
  ...

except CustomError as e:
    print('My exception occurred, value:', e.value)
```

Ici, on définit l'attribut `value` pour la classe `CustomError`. Cette valeur est définie lors de la création de l'instance de l'exception. On définit également la méthode `__str__()` pour l'affichage de l'exception.

[47]:
```
class SalaryNotInRangeError(Exception):
    """Exception raised for errors in the input salary.
    Attributes:
        salary -- input salary which caused the error
        message -- explanation of the error
    """
    def __init__(self, salary, message="is not in (5000, 15000) range"):
        self.salary = salary
        self.message = f"{salary} {message}"
        super().__init__(self.message)


salary = int(input("Enter salary amount: "))
if not 5000 < salary < 15000:
    raise SalaryNotInRangeError(salary)

try:
    raise SalaryNotInRangeError(2000, message="is a static input that raise an␣
 ↪error")
```

```
except SalaryNotInRangeError as e:
    print("Received error:", e.message)
```

```
Enter salary amount: 5001
Received error: 2000 is a static input that raise an error
```

[ ]: