Exception Handling in Python

Anoop S Babu
Faculty Associate
Dept. of Computer Science & Engineering
bsanoop@am.amrita.edu

Python Errors and Built-in Exceptions

- A python program terminates as soon as it encounters an unhandled error.
- These errors can be broadly classified into two classes:
  - Syntax Errors
  - Logical Errors (Exceptions)

Python Syntax Errors

- Error caused by **not following the proper structure** (syntax) of the language is called **syntax error** or **parsing error.**

```
>>> if a < 3
  File "<interactive input>", line 1
    if a < 3
           ^
SyntaxError: invalid syntax
```

- Here that a colon **:** is missing in the if statement.

Python Logical Errors (Exceptions)

- Errors that occur at runtime (after passing the syntax test) are called **exceptions** or **logical errors.**
  - Eg: ZeroDivisionError, IndexError, FileNotFoundError

```
>>> 1 / 0
Traceback (most recent call last):
 File "<string>", line 301, in runcode
 File "<interactive input>", line 1, in <module>
ZeroDivisionError: division by zero

>>> open("imaginary.txt")
Traceback (most recent call last):
 File "<string>", line 301, in runcode
 File "<interactive input>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'imaginary.txt'
```

Catching Exceptions in Python

- Exceptions can be handled using a **try** statement.
  - The **critical operation** which can raise an exception is placed inside the **try** clause.
  - The code that handles the exceptions is written in the **except** clause.

Command Prompt - python

```
>>> a=10
>>> b=5
>>> try:
...     print(a/b)
... except:
...     print('divide by 0')
... ^Z

2.0
>>> a=10
>>> b=0
>>> try:
...     print(a/b)
... except:
...     print('Division by zero')
... ^Z

Division by zero
>>> _
```

Catching Specific Exceptions in Python

- A try clause can have any number of except clauses to handle different exceptions
- Allows each exception be specific to errors.

Command Prompt - python

```
c:\Python>python
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:
34:40) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" fo
r more information.
>>> try:
...     print(x)
... except NameError:
...     print('x undefined')
... except:
...     print('other error')
... ^Z

x undefined
>>>
```

Raising Exceptions in Python

- can also manually raise exceptions using the **raise** keyword.

```
x = -1
if x < 0:
  raise Exception("Sorry, no numbers below zero")
```

```
Traceback (most recent call last):
  File "<string>", line 12, in <module>
Exception: Sorry, no numbers below zero
```

Example 2

Command Prompt - python

```
c:\Python>python
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:
34:40) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" fo
r more information.
>>> import math
>>> try:
...     a=input('enter a number')
...     a=int(a)
...     if a<0:
...             raise ValueError('-ve number')
... except ValueError as b:
...     print(math.factorial(a))
...
enter a number-5
Traceback (most recent call last):
  File "<stdin>", line 5, in <module>
ValueError: -ve number
```

try with else

- We can use the **else** keyword to define a block of code to be executed if no errors were raised.

```
try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")
```

```
Hello
Nothing went wrong
```

Example 2

```
try:
    num = int(input("Enter a number: "))
    assert num < 0
except:
    print("Positive number!")
else:
    num = num * -1
    print(num)
```

```
Enter a number: -5
5
```

Note: The assert keyword lets you test if a condition in your code returns True, if not, the program will raise an AssertionError.

try with finally

- The try statement in Python can have an **optional** finally clause.
- The finally block, will be **executed regardless if the try block raises an error or not.**

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

```
Something went wrong
The 'try except' is finished
```

User-Defined Exception in Python

- In Python, users can define **custom exceptions by creating a new class.**
- This exception class has to be derived, either directly or indirectly, from the built-in **Exception class**.

```python
# Define Python user-defined exceptions
class Error(Exception):
    """Base class for other exceptions"""
    pass
class ValueTooSmallError(Exception):
    """Raised when the input value is too small"""
    pass
class ValueTooLargeError(Error):
    """Raised when the input value is too large"""
    pass

number = 10
while True:
    try:
        num = int(input("Enter a number: "))
        if num < number:
            raise ValueTooSmallError
        elif num > number:
            raise ValueTooLargeError
        break
    except ValueTooSmallError:
        print("This value is too small, try again!")
    except ValueTooLargeError:
        print("This value is too large, try again!")
print("Congratulations! You guessed it correctly.")
```

```
Enter a number: 5
This value is too small, try again!
Enter a number: 20
This value is too large, try again!
Enter a number: 10
Congratulations! You guessed it correctly.
```