Python Programming

RGM College of Engineering & Technology (Autonomous)

Department of Computer Science & Engineering

EXCEPTION HANDLING - 3

Guido Van Rossum

Agenda:

1. try with multiple except blocks
2. Single except block that can handle multiple different exceptions
3. Default except block
4. finally block
5. finally block vs os._exit(0)

7. try with multiple except blocks

- The way of handling exception is varied from exception to exception.
- Hence for every exception type a separate except block we have to provide. (i.e., try with multiple except blocks is possible and recommended to use).

Syntax:

```
try:
    -------
    -------
    -------

except ZeroDivisionError:
    perform alternative arithmetic operations

except FileNotFoundError:
    use local file instead of remote file
```

Key Point:

- If try with multiple except blocks available then based on raised exception the corresponding except block will be executed.

Demo Program:

I.

```
try:
    x=int(input("Enter First Number: "))
    y=int(input("Enter Second Number: "))
    print('The Result is : ',x/y)

except ZeroDivisionError :
        print("Can't Divide with Zero")

except ValueError:
    print("please provide int value only")
```

```
Enter First Number: 10
Enter Second Number: 2
The Result is :  5.0
```

II.

```
try:
    x=int(input("Enter First Number: "))
    y=int(input("Enter Second Number: "))
    print('The Result is : ',x/y)

except ZeroDivisionError :
        print("Can't Divide with Zero")

except ValueError:
    print("please provide int value only")
```

```
Enter First Number: 10
Enter Second Number: 0
Can't Divide with Zero
```

III.

```
try:
    x=int(input("Enter First Number: "))
    y=int(input("Enter Second Number: "))
    print('The Result is : ',x/y)

except ZeroDivisionError :
        print("Can't Divide with Zero")

except ValueError:
    print("please provide int value only")
```

```
Enter First Number: 10
Enter Second Number: 2.5
please provide int value only
```

Note:

- If try with multiple except blocks available then the order of these except blocks is important.
- Python Virtual Machine will always consider from top to bottom until mentioned except block is identified.

Another Demo Program:

I.

```
try:
    print(10/0)

except ZeroDivisionError:
    print('ZeroDivisionError')

except ArithmeticError:
    print('ArithmeticError')
```

```
ZeroDivisionError
```

II.

```
try:
    print(10/0)

except ArithmeticError:
    print('ArithmeticError')

except ZeroDivisionError:
    print('ZeroDivisionError')
```

```
ArithmeticError
```

Note:

- ZeroDivisionError is the child class of ArithmeticError.
- ArithmeticError is the child class of Exception.
- Exception is the child class of BaseException.

8. Single except block that can handle multiple different exceptions

- If handling code is same for multiple exceptions, then instead of taking multiple exception blocks, We can write a single except block that can handle all those exceptions.

Syntax:

except (Exception1,Exception2,exception3,..): (or)

except (Exception1,Exception2,exception3,..) as msg:

Note:

- Parenthesis are mandatory and this group of exceptions internally considered as tuple.

Demo Program:

I.

```
try:
    x=int(input("Enter First Number: "))
    y=int(input("Enter Second Number: "))
    print('The Result is :',x/y)

except (ZeroDivisionError,ValueError) as msg:
    print("The Raised Exception is: ",msg.__class__.__name__)
    print("Description of Exception: ",msg)
    print('Please Provide Valid Input Only...')
```

```
Enter First Number: 10
Enter Second Number: 0
The Raised Exception is:  ZeroDivisionError
Description of Exception:  division by zero
Please Provide Valid Input Only...
```

II.

```
try:
    x=int(input("Enter First Number: "))
    y=int(input("Enter Second Number: "))
    print('The Result is :',x/y)

except (ZeroDivisionError,ValueError) as msg:
    print("The Raised Exception is: ",msg.__class__.__name__)
    print("Description of Exception: ",msg)
    print('Please Provide Valid Input Only...')
```

```
Enter First Number: 10
Enter Second Number: ten
The Raised Exception is:  ValueError
Description of Exception:  invalid literal for int() with base 10: 'ten'
Please Provide Valid Input Only...
```

9. Default except block

- We can use default except block to handle any type of exceptions.
- In default except block generally we can print exception information to the console.

Syntax:

```
except:
    statements
```

Eg:

```
try:
    x = int(input('Enter First Number :'))
    y = int(input('Enter Second Number :'))
    print('The Result is :',x/y)

except ZeroDivisionError:
    print('ZeroDivisionError : cannot divide with zero')

except:
    print('Default except : Please provide valid input only')
```

```
Enter First Number :10
Enter Second Number :0
ZeroDivisionError : cannot divide with zero
```

Eg:

```
try:
    x = int(input('Enter First Number :'))
    y = int(input('Enter Second Number :'))
    print('The Result is :',x/y)

except ZeroDivisionError:
    print('ZeroDivisionError : cannot divide with zero')

except:
    print('Default except : Please provide valid input only')
```

```
Enter First Number :10
Enter Second Number :two
Default except : Please provide valid input only
```

Eg:

```
try:
    x = int(input('Enter First Number :'))
    y = int(input('Enter Second Number :'))
    print('The Result is :',x/y)

#except ZeroDivisionError:
#    print('ZeroDivisionError : cannot divide with zero')

except:
    print('Default except : Please provide valid input only')
```

```
Enter First Number :10
Enter Second Number :0
Default except : Please provide valid input only
```

Note:

- If try with multiple except blocks available then default except block should be last, otherwise we will get SyntaxError.

Eg:

```
try:
    print(10/0)

except:
    print("Default Except")

except ZeroDivisionError:
    print("ZeroDivisionError")
```

```
  File "<ipython-input-7-df4e78ca6bbc>", line 2
    print(10/0)
    ^
SyntaxError: default 'except:' must be last
```

Eg:

```
try:
    print(10/0)
except ZeroDivisionError:
    print("ZeroDivisionError")
except:
    print("Default Except")
```

```
ZeroDivisionError
```

Another case:

```
try:
    print(10/0)
except ZeroDivisionError:
    print("ZeroDivisionError")
except ValueError:
    print("ValueError")
except:
    print("Default Except")
```

```
ZeroDivisionError
```

Eg:

```
try:
    print(10/0)
except ValueError:
    print("ValueError")
except ZeroDivisionError:
    print("ZeroDivisionError")

except:
    print("Default Except")
```

```
ZeroDivisionError
```

Key Conclusions:

1. If default except block is defined then compulsory it must be last except block otherwise we will get SyntaxError.

2. This restriction is applicable only for default except block but not for normal except blocks (i.e., Normal except blocks can be in any order).

Various Possible combinations of except blocks:

- The following are various possible combinations of except blocks:

1. except ZeroDivisionError:
2. except (ZeroDivisionError):
3. except ZeroDivisionError as msg:
4. except (ZeroDivisionError) as msg:
5. except (ZeroDivisionError,ValueError) :
6. except (ZeroDivisionError,ValueError) as msg:
7. except:

Important Conclusions:

1. If except block is defined for only one exception, then parenthesis are optional. If multiple exceptions are there, then parenthesis are mandatory.
2. If we use parenthesis, then 'as' must be outside of the parenthesis only.

10. finally block

Scenario 1:

- It is not recommended to place clean up code(Resource Deallocating Code or Resource Releasing code like closing database connection) inside try block because there is no guarantee for the execution of every statement inside try block always.

try:

1. Open DB connection
2. Read data ==> If Exception Raises Here , then Clean up code Never executed.
3. Close DB connection ==> Resource Deallocating Code or Clean Up code

except:

Handling Code

Scenario 2:

- It is not recommended to maintain clean up code inside except block, because if there is no exception, then except block won't be executed.

```
try:
    1. open db connection
    2. read data
    ==> if exception raises here, then clean up code never executed.

except:
    close db connection        ==> Resource Deallocating code or Clean up code
```

- Hence we required some place to maintain clean up code which should be executed always irrespective of whether exception raised or not raised and whether the exception handled or not handled.
- Such type of best place is nothing but finally block.

- The main purpose of finally block is to maintain clean up code.

Syntax:

```
try:
        Risky Code
except:
        Handling Code
finally:
        Cleanup code
```

Note:

- The speciality of finally block is it will be executed always whether exception raised or not raised and whether exception handled or not handled.

The following three cases will describes the speciality of 'finally' block:

Case 1: If there is no exception

```
try:
    print("try")
except:
    print("except")
finally:
    print("finally")
```

```
try
finally
```

Case 2: If there is an exception raised but handled

```
try:
    print("try")
    print(10/0)
except ZeroDivisionError:
    print("except")
finally:
    print("finally")
```

```
try
except
finally
```

Case 3: If there is an exception raised but not handled

```
try:
    print("try")
    print(10/0)
except NameError:
    print("except")
finally:
    print("finally")
```

```
try
finally

---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-3-6bb408b89075> in <module>
      1 try:
      2     print("try")
----> 3     print(10/0)
      4 except NameError:
      5     print("except")

ZeroDivisionError: division by zero
```

11. finally block vs os._exit(0)

- There is only one situation where finally block won't be executed. i.e., whenever we are using os._exit(0) function.
- Whenever we are using os._exit(0) function then Python Virtual Machine itself will be shutdown.
- In this particular case only, finally block won't be executed.

Demo Program:

I.

```
import os
try:
    print("try")
    #os._exit(0)
except NameError:
    print("except")
finally:
    print("finally")
```

```
try
finally
```

II.

```
import os
try:
    print("try")
    os._exit(0)
except NameError:
    print("except")
finally:
    print("finally")        # Executed in Editplus
```

```
C:\Pythonpractice>py ex1.py
try
```

Key Points to know about os._exit(0):

- Here 0 represents status code
- Zero means normal termination
- Non-Zero means Abnormal Termination
- The status code internally used by PVM
- Whether it is zero or non-zero there is no difference in the result of the program.

Any question?

If you try to practice programs yourself, then you will learn many things automatically

Spend few minutes and then enjoy the study

Thank You