UNIT 14 EXCEPTION HANDLING IN PYTHON PROGRAMMING

Structure

14.0 Introduction
14.1 Objectives
14.2 Default Exception Handler
14.3 Catching Exceptions
14.4 Raise an exception
14.5 User Defined Exceptions
14.6 Summary

14.0 INTRODUCTION

Programers always try to write an error-free program, but sometimes a program written by the programmer generates an error or not execute due to the error generated in the program. Some times the program code is correct, but still, it generates an error due to the malicious data is given as an input to the program. This malicious data may be given by the user or from a file that causes an error generation during the execution. This type of error generally occurs when we user server-side programs such as web programming and gaming servers.

Every developer is writing a code that must not generate an error during execution. We will study various types of errors that generate during the program execution or before the execution of the program code.

14.1 OBJECTIVES

After going through this unit, you will be able to :

- Understand the requirement of exception handling in programming
- Raise and catch the exceptions
- Perform exception handling in python programming

14.2 DEFAULT EXCEPTION HANDLER

Consider the following examples in which syntax is correct (as per the Python statement), but it causes an error.

Example 1.

```
>>>3/0
```

Name of Error will be ZeroDivisionError

And the reason for the error is division by zero

Example 2
>>> list1=[2, 3, 4, 5, 6]
>>>list1[5]

Name of Error is :IndexError
Reson of Error is : list index out of range

Example 3.
>>>x+3

Name of Error is : Name Error
Reason of Error is : name 'x' is not defined"

Example 4.
>>>int('22.5')

Name of Error is :ValueError
Reason of error is : invalid literal for int() with base 10: '22.5'

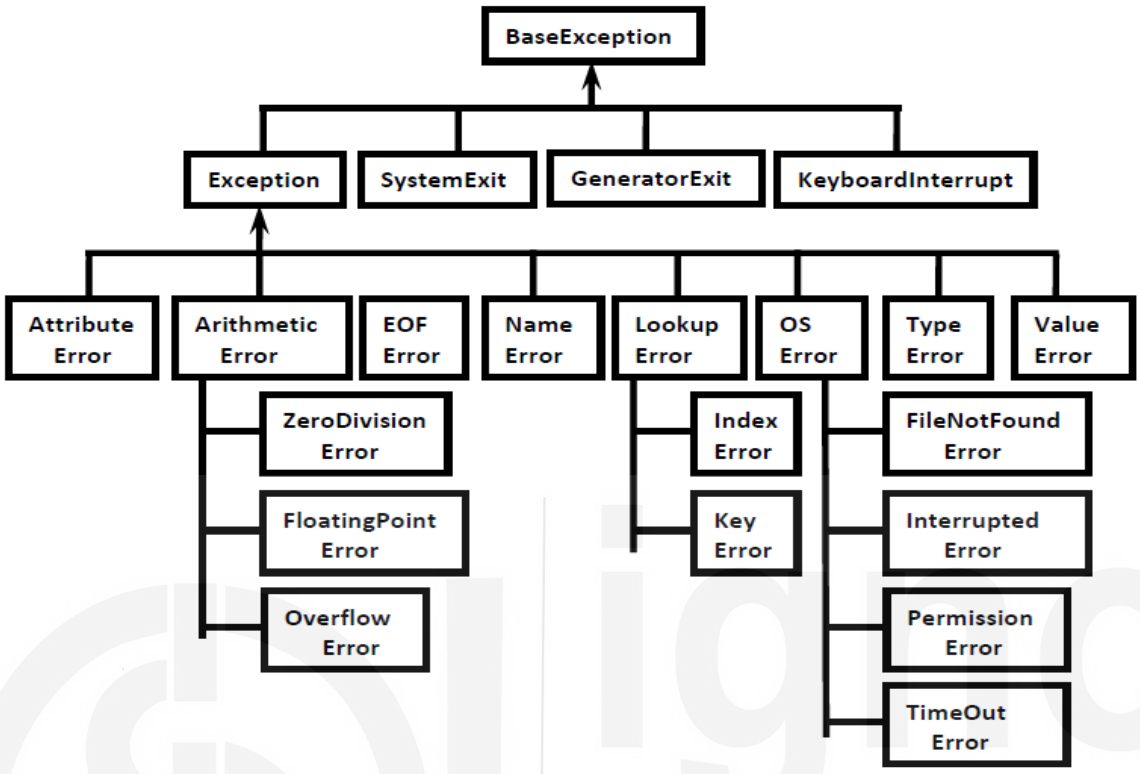
Example 5.
>>> '2'*'3'

Error name is :TypeError
Reason of Error is : can't multiply sequence by non-int of type 'str'

In each example syntax is correct, but it goes to an invalid state. These are runtime errors. In such a situation, Python interpreter generates an exception for run time errors. This exception generates an object which contains all information related to the error. In Example 5 the error message displayed: what is happening and at which line number it is happening (here line number is one because there is only one statement in the code) and the type of error is also printed.

It is the responsibility of python virtual machine to create the object corresponding to the exception generated to handle the program code. If the code responsible for handling the exception is not there in the program code, then python interpreter will suspend the execution of the program code since an exception is generated the python print the information about the generated exception.

14.3 CATCHING EXCEPTIONS



In Python, every exception is class, and these classes are the derived class of the BaseException class. ThusBaseException class is the topmost class in the hierarchy of exception classes.

Exception handling using try-except:

try:

risky code(generating exception)

except:

corresponding to risky code(handler)

The code which generates an exception must be written in a block with the name specify by 'try'and the code which will handle the exception will be given in 'except' block.

If the three lines code are written in Python

```
print("IGNOU")
print(87/0)
print("Welcome to the University")
>>>
…
```

Here the first line will be executed, and IGNOU will be printed when Python interpreter will execute 2nd lines of the program then it generates an exception, and the default exception handler will print the error code "division by Zero"then after the program is terminated. Thus the third line of the program will not execute. And this is also called as abnormal termination.

If we rewrite the code with the try and except then the code will become

```
print("IGNOU")
```

Example 1: Implementation of try and except block.

```
try:
            print(10/0)
except ZeroDivisionError:
            pass
print ("Welcome to the University")
print("IGNOU")
```

```
Ln: 6  Col: 14
================= RESTART: D:/python/IGNOU/Unit 14/Example 1.py ================
Welcome to the University
IGNOU
>>>
```

Here program terminated successfully. We can also print the error message by modifying the code as:

Example 2:

Example 2.py - D:/python/IGNOU/Unit 14/Example 2.py (3.8.5)

File Edit Format Run Options Window Help

```
try:
            print(23/0)
except ZeroDivisionError as msg1:
            print("Eception raised : ",msg1)
print("Welcome to the University")
```

```
Ln: 6  Col: 0
================= RESTART: D:/python/IGNOU/Unit 14/Example 2.py ================
Eception raised :  division by zero
Welcome to the University
>>>
Ln: 14  Col: 4
```

In the above code, the first line executed successfully and "IGNOU" will be printed. When the python interpreter executes the print statement "(23/0)" an exception is raised, and a message given in the print statement will be printed with the name of the exception "division by zero". After that,the interpreter executes the last statement, and it will print "Welcome to the University."

If there are more than one except block written in the try block,then the try block will execute the except block which is responsible for the code.

Example 3: Try block with more than one except block.

```
try:
        num1=int(input("Enter First Number: "))
        num2=int(input("Enter Second Number: "))
        print(num1/num2)
except ZeroDivisionError as msg1:
        print("Can't Divide with Zero : ",msg1)
except ValueError as msg1:
        print("Please Enter integer value only : ",msg1)
```

Ln: 8 Col: 0

```
================ RESTART: D:/python/IGNOU/Unit 14/Example 3.py ================
Enter First Number: 3
Enter Second Number: 4
0.75
>>>
================ RESTART: D:/python/IGNOU/Unit 14/Example 3.py ================
Enter First Number: 4
Enter Second Number: 0
Can't Divide with Zero :  division by zero
>>>
================ RESTART: D:/python/IGNOU/Unit 14/Example 3.py ================
Enter First Number: 0
Enter Second Number: 4
0.0
>>>
================ RESTART: D:/python/IGNOU/Unit 14/Example 3.py ================
Enter First Number: 4
Enter Second Number: a
Please Enter integer value only :  invalid literal for int() with base 10: 'a'
```

With the help of single except block, we can handle multiple exceptions:
Example 4: Multiple exception handling with a single try block and multiple except block.

```
try:
        num1=int(input("Enter First Number: "))
        num2=int(input("Enter Second Number: "))
        print(num1/num2)
except ZeroDivisionError as msg1:
        print("Can't Divide with Zero : ",msg1)
except ValueError as msg1:
        print("Please Enter integer value only : ",msg1)
```

Ln: 9 Col: 0

```
================ RESTART: D:/python/IGNOU/Unit 14/Example 4.py ================
Enter First Number: 4
Enter Second Number: 0
Can't Divide with Zero :  division by zero
>>>
================ RESTART: D:/python/IGNOU/Unit 14/Example 4.py ================
Enter First Number: 0
Enter Second Number: 3
0.0
>>>
================ RESTART: D:/python/IGNOU/Unit 14/Example 4.py ================
Enter First Number: 5
Enter Second Number: a
Please Enter integer value only :  invalid literal for int() with base 10: 'a'
>>>
```

Corresponding to each error there will be an except block if corresponding except block is not there then a except block written at the end of all except block is used for the error. And it must be the last except block in all.

Example 5: Implementation of default except for block

```
try:
            num1=int(input("Enter 1st Number: "))
            num2=int(input("Enter 2nd Number: "))
            print(num1/num2)
except ZeroDivisionError as msg:
            print("Plz provide value greater than ZERO value in second number and the error is :
except:
            print("Default Except:Plz provide valid input")
```

Ln: 9 Col: 0

```
>>>
================= RESTART: D:/python/IGNOU/Unit 14/Example 5.py ================
Enter 1st Number: 0
Enter 2nd Number: 4
0.0
>>>
================= RESTART: D:/python/IGNOU/Unit 14/Example 5.py ================
Enter 1st Number: 4
Enter 2nd Number: 0
Plz provide value greater than ZERO value in second number and the error is : division by zero
>>>
======================== RESTART: D:/python/IGNOU/Unit 14/Example 5.py ========================
Enter 1st Number: a
Default Except:Plz provide valid input
>>>
```

Some times exceptions may or may not be raised, and sometimes exceptions may or may not be handled. Irrespective of both of these we still want to execute some code. And such codes are written in finally block.

try:

Code with Error

except:

Code to handle error

finally:

Necessary code

Example 6: Implementation of finally block.

```
try :
            print("Block inside Try")
            print(35/0)
except ZeroDivisionError:
            print("Except Block")
finally:
            print("Necessary Code")
```

Ln: 8 Col: 0

```
======================== RESTART: D:/python/IGNOU/Unit 14/Example 6.py ========================
Block inside Try
Except Block
Necessary Code
>>>
```

Nested try-except block:

If a try block is written inside another try block, then it is called as nested.

The code having higher risk must be defined in the innermost try block. If there is any exception raised by the innermost try block, then it will be handled by the innermost except block. If the innermost except block is not able to handle the errorthenexceptblock defined outside the inner try block will handle the exception.

Example 7: Implementation of the nested try block

```
try:                                        # try block
        print("Outer try block")
        try:                                # nested try block
                print("Inner try block ")
                print(22/0)
        except ZeroDivisionError:
                print("Except block of Inner try block")
        finally:
                print("Finally block of  Inner try block")
except:
        print("Except block of  Outer try block")
finally:
        print("Finally Block of Outer try block")
```

Ln: 7 Col: 63

```
========== RESTART: D:/python/IGNOU/Unit 14/Example 7.py ==========
Outer try block
Inner try block
Except block of Inner try block
Finally block of  Inner try block
Finally Block of Outer try block
>>>
```

A single except block can be used to handle multiple exceptions. Consider the given an example:

Example 8:Single except block handling multiple exceptions.

```
try:
    num1=int(input("Enter first Number: "))
    num2=int(input("Enter Second Number: "))
    print(num1/num2)
except (ZeroDivisionError,ValueError) as printmsg:
    print("Invalid Number & Error is: ",printmsg)
```

Ln: 5 Col: 50

```
========== RESTART: D:/python/IGNOU/Unit 14/Example 8.py ==========
Enter first Number: 4
Enter Second Number: 0
Invalid Number & Error is:  division by zero
>>>
========== RESTART: D:/python/IGNOU/Unit 14/Example 8.py ==========
Enter first Number: 4
Enter Second Number: two
Invalid Number & Error is:  invalid literal for int() with base 10: 'two'
>>>
```

In case of different types of error, we can use a default except for block. This block is generally used to display normal error messages. The default except block must be the last block of all except blocks.

Example 9: Implementation of default except for block.

```
try:
    n1=int(input("Enter First Number: "))
    n2=int(input("Enter Second Number: "))
    print(n1/n2)
except ZeroDivisionError:
    print("ZeroDivisionError:Can't divide with zero")
except:
    print("Default Except:Plz provide valid input only")
```

Ln: 9 Col: 0

```
========== RESTART: D:/python/IGNOU/Unit 14/Example 9.py ==========
Enter First Number: 5
Enter Second Number: 6
0.8333333333333334
>>>
========== RESTART: D:/python/IGNOU/Unit 14/Example 9.py ==========
Enter First Number: 22
Enter Second Number: 0
ZeroDivisionError:Can't divide with zero
>>>
========== RESTART: D:/python/IGNOU/Unit 14/Example 9.py ==========
Enter First Number: 22
Enter Second Number: ten
Default Except:Plz provide valid input only
>>>
```

14.4 RAISE AN EXCEPTION

Some times user want to generate an exception explicitly to inform that this is the exception in this code. Such type of exceptions is called as user-defined exceptions or customized exceptions.

This user-defined exception is a class defined by the user, and this class is derived from the Exception class. Pythoninterpreter does not have any information related to the user-defined exception class. Thus, it must be raised explicitly by the user.

The keyword*raise* is used to raise the class when it is required.

Example 10: Implementation of user-defined exceptions.

```
class EligibleForEntrance (Exception ) :
        def __init__(self,argl) :
                self.msgl=argl

class NotEligible (Exception) :
        def __init__ (self,argl) :
                self.msgl=argl

percentage=int (input ("Enter Your PCM Marks percentage"))
if percentage>50:
        raise EligibleForEntrance("You can apply for the entrance examination")
else:
        raise NotEligible("You cant apply for the entrance examination")
```

Ln: 5 Col: 31

```
Enter Your PCM Marks percentage 88
__main__.EligibleForEntrance: You can apply for the entrance examination
>>>
================ RESTART: D:/python/IGNOU/Unit 14/Example 10.py ================
Enter Your PCM Marks percentage 44
__main__.NotEligible: You can't not apply for the entrance examination
```

14.5 USER-DEFINED EXCEPTIONS

A programmer can create his exception, called a user-defined exception or custom exception. A user-defined exception is also a class derived from exception class.

Thus, class, is the user define a class which is a subclass of the Exception class.

There are two steps in using user-defined exception:

Step 1: By inheriting Exception class create a user-defined exception class.

Step 2: Raise this exception in a program where we want to handle the exception.

Syntax :

```
classMyException(Exception):
        pass
classMyException(Exception)
        def __init__(self,argumnet):
                self.msg=argumnet
```

A *raise* statement is used to raise the statement.

Example 11: Program to raise an exception.

```
class TooYoungException(Exception):
    def __init__(self,arg):
        self.msg=arg

class TooOldException(Exception):
    def __init__(self,arg):
        self.msg=arg

age=int(input("Enter Age:"))
if age>60:
    raise TooYoungException("Plz wait some more time you will get best match soon!!!")
elif age<18:
    raise TooOldException("Your age already crossed marriage age...no chance of getting marriage")
else:
    print("You will get match details soon by email!!!")
```

Ln: 8 Col: 0

```
========== RESTART: D:/python/IGNOU/Unit 14/Example 11.py ==========
Enter Age:55
Plz wait some more time you will get best match soon!!!
>>>
========== RESTART: D:/python/IGNOU/Unit 14/Example 11.py ==========
Enter Age:33
You will get match details soon by email!!!
>>>
```

Check Your Progress

1. Give the name of the error generated by the following python codes.
   a) 4 / 0
   b) (3+4]
   c) lst = [4;5;6]
   d) lst = [14, 15, 16]
      lst[3]
   e) x + 5
   f) '2' * '3'
   g) int('4.5')
   h) 2.0**10000
   i) for i in range(2**100):
      pass
2. Define the minimum number of except statements that can be possible in the try-except block.

   ---

3. Describe the conditions in which finally block executed.

   ---

4. Select the keyword from the following, which is not an exception handling in Python.
   a) try
   b) except
   c) accept
   d) finally

   ---

5. Give the output generated from the following Python code?
   lst = [1, 2, 3, 4]
   lst[4]

   ---

6. What will be the output generated with the following code?

```
def function1():
        try:
        function2(var1, 22)
        finally:
        print('after function2')
        print('after function2?')
function1()
```

a)Output will not be generated
b) after f?
c) error
d) after f

14.6 SUMMARY

In this unit, it is defined that a program may produce an error even though the programmer is writing error-free code. These are the mistakes that change the behaviour of our program code.

In this unit, it is defined that the default exception handlers are there in Python. How these exception handler works are described in this unit.

By default, there are exception handlers in Python. But a user can also raise the exception which is a customized exception, not a language defined exception.

SOLUTION TO CHECK YOUR PROGRESS

Check Your Progress

1.
   a) ZeroDivisionError
   b) SyntaxError
   c) SyntaxError
   d) IndexError
   e) NameError
   f) TypeError
   g) ValueError
   h) OverflowError
   i) KeyboardInterrupt Error
2. At least one except statement.
3. Finally block is always executed.
4. C
5. IndexErrorbecause the maximum index of the list given is 3.
6. C, since function finction1 is not defined.