Marwadi
University

01CE1705-Programming with Python

Unit-4 Exception Handling

Prof. Chetan Chudasama
Computer Engineering Department

Outline

- Types of Errors
- Exception
- Exception Handling
- Exception Hierarchy
- try,except block
- try with multiple except
- Single except block
- Default except block
- finally block
- else Block
- Types of exception

Types of errors

- In Any Programming Language there are 2 types of possible errors.
  1. Syntax Errors
  2. Runtime Errors

Syntax Errors:-

- The error which occurs because of invalid syntax, such type of error are considered as Syntax Errors.
- Programmer is responsible to correct these syntax errors. Once all syntax errors are corrected then only program execution will be started.

Example:

```
a=10          print "Marwadi University"
if a==10
   print('a is 10’)
```

Runtime Errors:-

- While executing the program, at runtime if something goes wrong because of user input or programming logic then we will get runtime errors. It is also known as exceptions.
- Exception handling concept is applicable for runtime errors not for syntax errors.

Example:

```
x=int(input("Enter First Number"))#x=10,x=ten
y=int(input("Enter Second Number"))#y=0
print(x/y)
```

Example:

```
f=open("sairam.txt")
print(f.read())
```

Exception

- An unexcepted event that disturbs normal flow of program is called Exception.
  Example: InternetError, SleepingError ,TyrePuntureError, FileNotFoundError, ZeroDivisionError, ValueError

Exception Handling:-

- Exception handling does not mean repairing an exception. We have to define alternative way to continue rest of the program normally.

Case 1:This week I am planning to go to my native place by bus.

Case 2:This week I am planning to go to my native place by bus. If bus is not available, then I will try for train. If train is not available, then I will try for flight. Still, it is not available then I will go for cab. This way of defining alternative way is called exception handling.

Example:

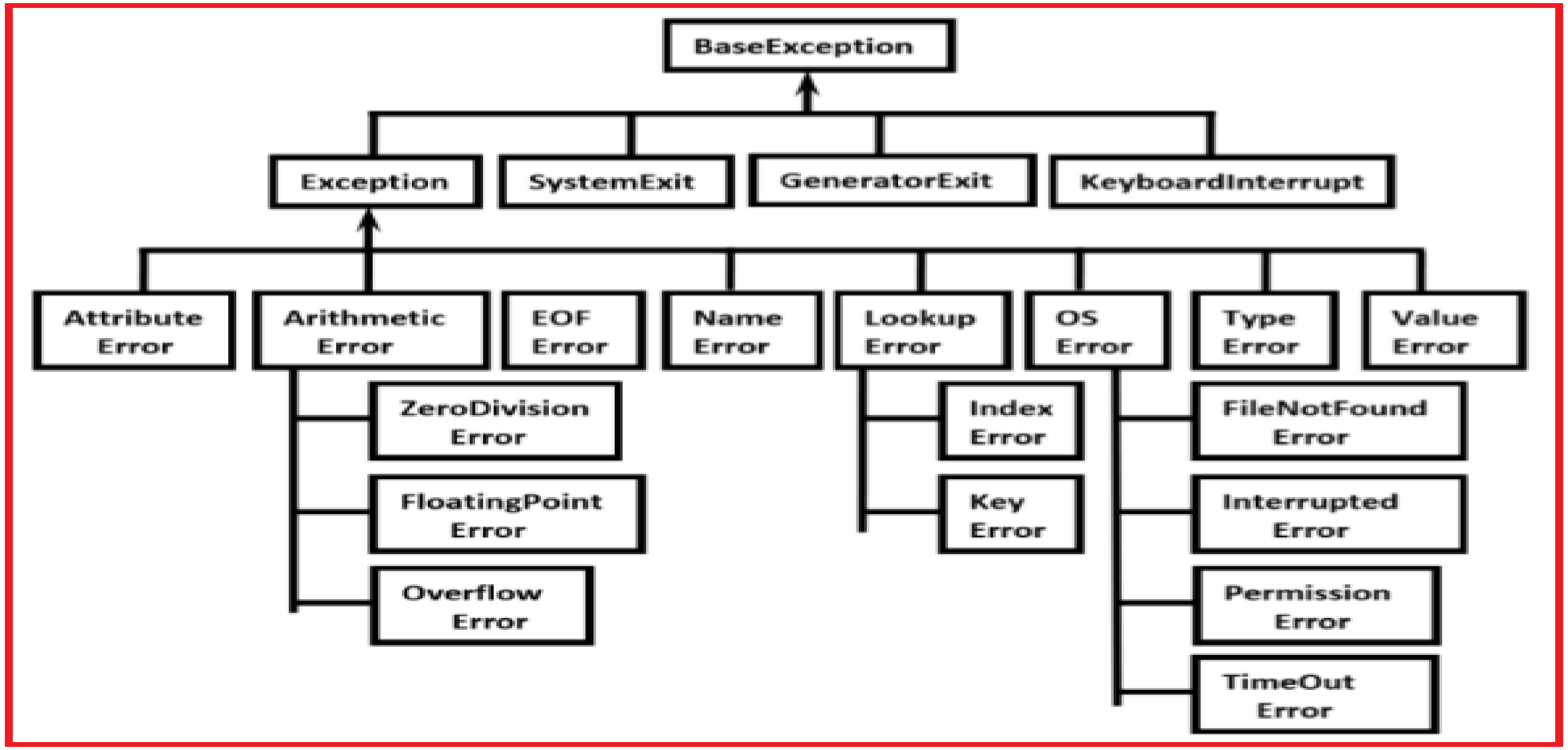
```
try:
        read data from file marwadiUni.txt
except FileNotFoundError:
        use the local file and continue rest of the program normally
```

What is the main objective of Exception Handling:-

- It is highly recommended to handle exception.
- The main objective of exception handling is graceful termination/normal termination of the application(we should not block our resources and we should not miss anything)

Exception Hierarchy

- For Every exception in Python there is one class.
- All exception classes are child classes of BaseException either directly or indirectly.
- Hence BaseException acts as root for Python Exception Hierarchy.



Exception handling using try,except

- Every exception in python is an object. For every exception type the corresponding class is available.
- When an exception occurs PVM will create the corresponding exception object and will check for handling code. If handling code is available, then it will be executed and rest of the program will be executed normally.

Example:

```
try:
        print(10/0)
except ValueError:
        print('Exception occurs')
```

```
try:
        print(10/0)
except ZeroDivisionError:
        print('Exception occurs')
```

- If handling code is not available, then PVM will terminates program abnormally and print corresponding exception information.
- To prevent this abnormal termination, we should handle exception by using try-except blocks.

- The code which may generate an exception is called risky code and we have to write that risky code inside try block.
- Exception handling code must be written inside except block. Note:-The length of the try block should be less as possible.

Print exception information to console:-

```
try:
    print(10/0)
except ZeroDivisionError as msg:
    print("The Type of exception",type(msg))
    print("The Type of exception",msg.__class__)
    print("The exception class name",msg.__class__.__name__)
    print("The description of exception",msg)
```

Note:-If you want to print exception information, use **as** keyword.

try with multiple except block

- The way of handling an exception is varied from exception to exception.
- Hence for every possible exception type, we have to take seperate except block. try with multiple except block is possible and recommended to use.

Syntax:

```
try:
        .....
        .....
except ZeroDivisionError:
        perform alternate arithmetic operations
except FileNotFoundError:
        use local file instead of remote file
```

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

Example:

```
try:
	x=int(input("Enter First Number"))
	y=int(input("Enter Second Number"))
	print("The result",x/y)
except ZeroDivisionError:
	print('Can not divide with zero')
except ValueError:
	print("Please provide integer value only")
```

- If try with multiple except blocks available, then the order of these except block is important. Python virtual machine will always consider from top to bottom until matched except block identified.

Single except block

- If handling code is same for multiple exceptions, then instead of taking different except block, we can take single except block that can handle all those exception.

Syntax:-

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

- parenthesis is mandatory and this group of exception internally considered as tuple.

Example:-

```
str1=input("Enter name ")
try:
	no=int(input("Enter number "))
	print(str1+no)
except (TypeError,ValueError) as msg:
	print(msg.__class__.__name__)
```

- In the above example, string and an integer cannot be added,PVM generates TypeError.
- The second input should have been an integer, if we passed a string 'rajkot' therefore ValueError is raised.

Default Except block:-

- It will be executed if there is no other except block is matched.

Syntax:

```
except:
        statements
```

- If default except block defined then compulsory it must be last except block otherwise we will get SyntaxError.
- This restriction is applicable only for default except block, normal except block can be in any order.

Example:

```
try:
        x=int(input("Enter First Number\n"))
        y=int(input("Enter Second Number\n"))
        print(x/y)
except ZeroDivisionError:
  print('Cannot divide with zero')
except:
  print('default except:please provide valid input only')
```

finally block

- It is not recommended to place cleanup code(resource deallocation code like closing database connection etc) inside try block, because there is no guarantee for execution of every statement inside try block.
- It is not recommended to place cleanup code inside except block because if there is no exception then except block wont be executed.
- Hence we required some place to maintain cleanup code which should be executed always irrespective of whether exception occurs or not and whether exception is handled or not. Such type of best place is nothing but finally block.
- So main purpose of finally block is to maintain cleanup code.
- The speciality of finally block is, it will be executed always irrespective of whether exception is occurs or not and whether exception is handled or not.

Syntax:

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

Example:-

```
try:
  f=open("test.txt")
except:
  print('Could not open file')
finally:
  f.close()
print('Program continue')
```

else block

- We can use else block with try-except-finally blocks.
- If there is no exception in try block then only else will be executed.

Syntax:

```
try:
        Risky code
except:
        Handling code
        It will be executed if an exception inside try
else:
        It will be executed if there is no exception inside try
finally:
        cleanup code
        will be executed whether exception raised or not raised and handled or not handled
```

Note:-This is python specific concept. In java we can't use else with try, catch and finally. But in python we can use else everywhere(with loop and with try, except and finally).

- There is no chance of executing both except and else simultaneously.
- If we take else block, compulsory except block should be there. Else without except block is always invalid.

Example:

```
f=None
try:
        f=open("sairam.txt","r")
except:
        print("Some problem while opening file")
```

```
else:
        print("file opened successfully")
        print("Content of the file")
        print(f.read())
finally:
        if f is not None:
                f.close()
```

Types of Exception

- There are two types of exception
  1.Predefined Exceptions
  2.User defined Exception

Predefined Exceptions:-

- It is also knowing as inbuilt exceptions or PVM exceptions.
- These will be raised automatically by Python Virtual Machine Whenever a particular event occurs.

Example1:-Whenever we are trying to perform division by zero,automatically python will raise ZeroDivisionError.

```
print(10/0)#ZeroDivisionError
```

Example2:-Whenever we are trying to convert str value to int type and string does not contain integer value then we will get ValueError automatically.

```
x=int('ten')#ValueError
```

User Defined Exception:-

- It is also known as Customized Exceptions or Programmatic Exceptions.
- Sometimes we have to define and raise exceptions explicitly to indicate that something goes wrong such type of exceptions are called user defined exceptions or customized exceptions.
- Programmer is responsible to define these exceptions and Python Virtual Machine not having any idea about these.

•Hence we have to raise explicitly based on our requirement by using 'raise' keyword.

Example: NumberDivideByOne, InSufficientBalance, TooYoungException, TooOldException

How to define and raise customized exception

- Every exception in python is a class and it should be child class of BaseException.
- We can raise exception by using raise keyword.

Example:-

```
class TooYoungException(Exception):
        def __init__(self,msg):
                self.msg=msg
class TooOldException(Exception):
        def __init__(self,msg):
                self.msg=msg
age=int(input("Enter  age  "))
if age>60:
        raise TooOldException('Your age already crossed marriage age, no chance of
        getting marriage')
```

```
elif age<18:
        raise TooYoungException("Please wait some more time, definitely you will get best
        match")
else:
        print("You will get match details soon by email")
```