By Sai Kumar

EXCEPTION HANDLING

In any programming language there are 2 types of errors are possible.

1) **Syntax Errors**

2) **Runtime Errors**

Syntax Errors

The errors which occur because of invalid syntax are called syntax errors.

Eg 1

```
x = 10

if x == 10

print("Hello")
```

SyntaxError: invalid syntax

Eg 2

```
print "Hello"
```

SyntaxError: Missing parentheses in call to 'print'

Note: Programmer is responsible to correct these syntax errors. Once all syntax errors are corrected then only program execution will be started.

Runtime Errors

Also known as exceptions.

While executing the program if something goes wrong because of end user input or programming logic or memory problems etc then we will get Runtime Errors.

Eg:

1) **print(10/0)** → ZeroDivisionError: division by zero

2) **print(10/"ten")** → TypeError: unsupported operand type(s) for /:

'int' and 'str'

3)
```
x = int(input("Enter Number:"))

print(x)
```

Enter Number: **ten**

ValueError: invalid literal for int() with base 10: 'ten'

Note: Exception Handling concept applicable for Runtime Errors but not for syntax errors

What is Exception?

An unwanted and unexpected event that disturbs normal flow of program is called exception.

Eg:

ZeroDivisionError

TypeError

ValueError

FileNotFoundError

EOFError

It is highly recommended to handle exceptions. The main objective of exception handling is Graceful Termination of the program (i.e we should not block our resources and we should not miss anything)

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

Eg: For example, our programming requirement is reading data from remote file locating at London. At runtime if London file is not available then the program should not be terminated abnormally. We have to provide local file to continue rest of the program normally. This way of defining alternative is nothing but exception handling.

try:

Read Data from Remote File locating at London.

except FileNotFoundError:

use local file and continue rest of the program normally

Default Exception Handing in Python

Every exception in Python is an object. For every exception type the corresponding classes are available.

Whenever an exception occurs PVM will create the corresponding exception object and will check for handling code. If handling code is not available then Python interpreter terminates the program abnormally and prints corresponding exception information to the console.

The rest of the program won't be executed.

1) **print("Hello")**

2) **print(10/0)**

3) **print("Hi")**

Hello

Traceback (most recent call last):

File "test.py", line 2, in <module>

print(10/0)

ZeroDivisionError: division by zero

Customized Exception Handling by using try-except

It is highly recommended to handle exceptions.

The code which may raise exception is called risky code and we have to take risky code inside try block. The corresponding handling code we have to take inside except block.

try:

Risky Code

except XXX:

Handling code/Alternative Code

Without try-except:

1) **print("stmt-1")**

2) **print(10/0)**

3) **print("stmt-3")**

Output

stmt-1

ZeroDivisionError: division by zero

Abnormal termination/Non-Graceful Termination

With try-except:

1) **print("stmt-1")**

2) **try:**

3) **print(10/0)**

4) **except ZeroDivisionError:**

5) **print(10/2)**

6) **print("stmt-3")**

Output

stmt-1

5.0

stmt-3

Normal termination/Graceful Termination

Control Flow in try-except

try:

stmt-1

stmt-2

stmt-3

except XXX:

stmt-4

stmt-5

Case-1: If there is no exception

1,2,3,5 and Normal Termination

Case-2: If an exception raised at stmt-2 and corresponding except block matched 1,4,5 Normal Termination

Case-3: If an exception rose at stmt-2 and corresponding except block not matched 1, Abnormal Termination

Case-4: If an exception rose at stmt-4 or at stmt-5 then it is always abnormal termination.

Conclusions:

1) Within the try block if anywhere exception raised then rest of the try block won't be executed even though we handled that exception. Hence we have to take only risky code inside try block and length of the try block should be as less as possible.

2) In addition to try block, there may be a chance of raising exceptions inside except and finally blocks also.

3) If any statement which is not part of try block raises an exception then it is always abnormal termination.

How to Print Exception Information:

try:

1) **print(10/0)**

2) **except ZeroDivisionError as msg:**

3) **print("exception raised and its description is:",msg)**

Output exception raised and its description is: division by zero

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.

Eg:

try:

----

except ZeroDivisionError:

perform alternative arithmetic operation

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.

1) try:

2) x=int(input("Enter First Number: "))

3) y=int(input("Enter Second Number: "))

4) print(x/y)

5) except ZeroDivisionError :

6) print("Can't Divide with Zero")

7) except ValueError:

8) print("please provide int value only")

Enter First Number: 10

Enter Second Number: 2

5.0

Enter First Number: 10

Enter Second Number: 0

Can't Divide with Zero

Enter First Number: 10

Enter Second Number: ten

please provide int value only

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

1) try:

2) x=int(input("Enter First Number: "))

3) y=int(input("Enter Second Number: "))

4) print(x/y)

5) except ArithmeticError :

6) print("ArithmeticError")

7) except ZeroDivisionError:

8) print("ZeroDivisionError")

Single except Block that can handle Multiple Exceptions:

We can write a single except block that can handle multiple different types of exceptions.

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

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

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

1) try:

2) x=int(input("Enter First Number: "))

3) y=int(input("Enter Second Number: "))

4) print(x/y)

5) except (ZeroDivisionError,ValueError) as msg:

6) print("Plz Provide valid numbers only and problem is: ",msg)

Enter First Number: 10

Enter Second Number: 0

Plz Provide valid numbers only and problem is: division by zero

Enter First Number: 10

Enter Second Number: ten

Plz Provide valid numbers only and problem is: invalid literal for int() with b are 10: 'ten'

Default except Block:

We can use default except block to handle any type of exceptions.

In default except block generally we can print normal error messages.

Syntax:

except:

statements

1) try:

2) x=int(input("Enter First Number: "))

3) y=int(input("Enter Second Number: "))

4) print(x/y)

5) except ZeroDivisionError:

6) print("ZeroDivisionError:Can't divide with zero")

7) except:

8) print("Default Except:Plz provide valid input only")

Enter First Number: 10

Enter Second Number: 0

ZeroDivisionError:Can't divide with zero

D:\Python_classes>py test.py

Enter First Number: 10

Enter Second Number: ten

Default Except:Plz provide valid input only

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

1) try:

2) print(10/0)

3) except:

4) print("Default Except")

5) except ZeroDivisionError:

6) print("ZeroDivisionError")

SyntaxError: default 'except:' must be last

Note: The following are various possible combinations of except blocks

1) except ZeroDivisionError:

2) except ZeroDivisionError as msg:

3) except (ZeroDivisionError,ValueError) :

4) except (ZeroDivisionError,ValueError) as msg:

5) except :

User Defined Exceptions

Also known as Customized Exceptions or Programatic Exceptions

Some time 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 not having any idea about these. Hence we have to raise explicitly based on our requirement by using "**raise**" keyword.

Eg:

InSufficientFundsException

InvalidInputException

TooYoungException

TooOldException

How to Define and Raise Customized Exceptions

Every exception in Python is a class that extends Exception class either directly or indirectly.

Syntax:

class classname(predefined exception class name):

def __init__(self,arg):

self.msg=arg

1) class TooYoungException(Exception):

2) def __init__(self,arg):

3) self.msg=arg

TooYoungException is our class name which is the child class of Exception

We can raise exception by using raise keyword as follows

raise TooYoungException("message")

1) class TooYoungException(Exception):

2) def __init__(self,arg):

3) self.msg=arg

4)

5) class TooOldException(Exception):

6) def __init__(self,arg):

7) self.msg=arg

8)

9) age=int(input("Enter Age:"))

10) if age>60:

11) raise TooYoungException("Plz wait some more time you will get best match soon!!!")

12) elif age<18:

13) raise TooOldException("Your age already crossed marriage age...no chance of

getting marriage")

14) else:

15) print("You will get match details soon by email!!!")

Enter Age:90

__main__.TooYoungException: Plz wait some more time you will get best match soon!!!

Enter Age:12

__main__.TooOldException: Your age already crossed marriage age...no chance of getting marriage

Enter Age:27

You will get match details soon by email!!!

Note: raise keyword is best suitable for customized exceptions but not for pre-defined exceptions