Chapter 3: Exception Handling in Python

**1. Explain different statements/keywords in exception handling.**

**try and except:**

If the Python program contains suspicious code that may throw the exception, we must place that code in the try block. The try block must be followed with the except statement, which contains a block of code that will be executed if there is some exception in the try block.

We can declare multiple except statements since the try block may contain the statements which throw the different type of exceptions.

The Python allows us to declare the multiple exceptions with the single except clause also.

Example:

try:

a=int(input('Enter first number='))

b=int(input('Enter second number='))

c=a/b

print('Division=',c)

except:

print('Division by zero error...')

else:

We can also use the else statement with the try-except statement in which, we can place the code which will be executed in the scenario if no exception occurs in the try block.

Example:

try:

a=int(input('Enter first number='))

b=int(input('Enter second number='))

c=a/b

except:

print('Division by zero error...')

else:

print('Division=',c)

finally:

Python provides a keyword finally, which is always executed after try and except blocks.

The finally block lets you execute code, regardless of the result of the try- and except blocks.

Example:

try:

a=10

b=0

c=a/b;

print(c)

except:

print('Exception occurs')

else:

print('Exception not occurs')

finally:

print('Exception occurs or not')

raise:

An exception can be raised forcefully by using the raise clause in Python. It is useful in in that scenario where we need to raise an exception to stop the execution of the program.

It is also used to create user defined exceptions (Custom Exceptions)

Example:

try:

age=int(input('Enter age='))

if age<18:

raise NameError

else:

print('you can vote...')

except NameError:

print('you cant vote...')

**2. Explain user defined exception with example**

Python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions.

Here is an example related to RuntimeError. Here, a class is created that is subclassed from RuntimeError. This is useful when you need to display more specific information when an exception is caught.

In the try block, the user-defined exception is raised and caught in the except block. The variable e is used to create an instance of the class Networkerror.

class Networkerror(RuntimeError):

def \_\_init\_\_(self, arg):

self.args = arg

So once you defined above class, you can raise the exception as follows −

try:

raise Networkerror("Bad hostname")

except Networkerror,e:

print e.args

Example:

class stringmismatch(Exception):

def f1(self):

print('STRINGS ARE NOT EQUAL...')

try:

s1=input('Enter first string=')

s2=input('Enter second string=')

if s1==s2:

print('Strings are equal')

else:

raise stringmismatch

except stringmismatch as s:

s.f1()