Exception definition:

Need of exception handling:

Keywords used in exception handling:

Try, except, raise, finally

class ExceptionDemo :

def devide(self , a, b):

res = a/b

print("Result : ",res)#realted stat to exception

print("End of function")

return

a = int(input("Enter a value : "))

b = int(input("Enter b value : "))

obj = ExceptionDemo()

obj.devide(a,b)

class ExceptionDemo :

def devide(self , a, b):

try:

res = a/b

print("Result : ",res)#realted stat to exception

except ZeroDivisionError:

print("Denominator should not be zero..")

print("End of function")

return

a = int(input("Enter a value : "))

b = int(input("Enter b value : "))

obj = ExceptionDemo()

obj.devide(a,b)

class ExceptionDemo :

def check():

str = input("Enter integer string : ")

i = int(str)

print("int value : ", i)

return

ExceptionDemo.check()

class ExceptionDemo :

def check():

str = input("Enter integer string : ")

try:

i = int(str)

print("int value : ", i)

except ValueError:

print("Invalid String conversion")

return

ExceptionDemo.check()

MultiCatch:

class MultiCatch :

def check():

a = 0

b = 0

try:

s1 = input("Enter a value : ")

s2 = input("Enter b value : ")

a = int(s1)

b = int(s2)

c = a/b

print("Result : ",c)

except ValueError:

print("Error : Invalid String conversions")

except ZeroDivisionError:

print("Error : Denominator should not be zero")

print("End...")

return

MultiCatch.check()

class MultiCatch :

def check():

a = 0

b = 0

try:

s1 = input("Enter a value : ")

s2 = input("Enter b value : ")

a = int(s1)

b = int(s2)

c = a/b

print("Result : ",c)

except(ValueError, ZeroDivisionError):

print("Exception")

print("End...")

return

MultiCatch.check()

class MultiCatch :

def check():

a = 0

b = 0

try:

s1 = input("Enter a value : ")

s2 = input("Enter b value : ")

a = int(s1)

b = int(s2)

except ValueError:

print("Error-1")

except ValueError:

print("Error-2")

return

MultiCatch.check()

class MultiCatch :

def check():

try:

str = input("Enter a value : ")

a = int(str)

print("a value : ",a)

except ZeroDivisionError:

print("Exception")

return

MultiCatch.check()

Opening File:

try:

f = open('g:/input.txt', 'r')

print("File opened in read mode")

except IOError:

print("Exception : No such file")

class ExceptionDemo:

def openFile(self):

try:

file = open("g:/input.txt", "r")

print("File opened in read mode....")

except FileNotFoundError:

print("No such file to open...")

return

obj = ExceptionDemo()

obj.openFile()

class ExceptionDemo:

def test(self):

try:

# a = 10/5

a = 10/0

print("Result in try block : ",a)

except ZeroDivisionError:

print("Exception Handling...")

finally:

print("Finally block.....")

return

obj = ExceptionDemo()

obj.test()

class ExceptionCheck:

def fileOpen(self):

f = ''

try:

f = open('g:/input1.txt', 'r')

print("File opened in read mode")

except IOError:

print("Exception : No such file")

finally:

if(f==''):

print("File not opened, no need to close...")

else:

f.close()

print("File closed....")

obj = ExceptionCheck()

obj.fileOpen()

Creating User-defined Exception

Exception can be either pre-defined or user-defined.

We can create exceptions depends on application requirement.

A user defined exception need to extend Pre-defined Exception class.

class MyException(Exception):

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

self.msg = msg

def getMessage(self):

return self.msg

class Sample:

def test(self):

raise MyException("Error\_msg")

obj = Sample()

obj.test()

Handling user exception:

class MyException(Exception):

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

self.msg = msg

def getMessage(self):

return self.msg

class Sample:

def test(self):

raise MyException("Error\_msg")

obj = Sample()

try:

obj.test()

except MyException as e:

print("Exception caught : ",e)

print("Exception caught : ",e.getMessage())

Banking application:

class Account:

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

self.balance = balance

def getBalance(self):

return self.balance

def withdraw(self, amount):

if(amount <= self.balance):

print("Collect cash : ",amount)

self.balance = self.balance - amount ;

else:

print("Insufficient funds...")

return

class Banking:

def main():

amt = int(input("Enter initial amount in the account : "))

acc = Account(amt)

print("Balance in account : ", acc.getBalance())

wi\_amt = int(input("Enter amount to withdraw : "))

acc.withdraw(wi\_amt)

print("Final balance : ", acc.getBalance())

return

if \_\_name\_\_ =="\_\_main\_\_":

Banking.main()

class B(Exception):

pass

class C(B):

pass

class D(C):

pass

for cls in [B, C, D]:

try:

raise cls()

except D:

print("D")

except C:

print("C")

except B:

print("B")

class LowBalanceError(Exception):

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

self.name = name

def getErrorMessage(self):

return self.name

class Account:

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

self.balance = balance

def getBalance(self):

return self.balance

def withdraw(self,amount):

print("Trying to withdraw : ",amount)

print("Available balance : ",self.balance)

if(amount <= self.balance):

self.balance = self.balance - amount

print("Collect cash : ",amount)

else:

raise LowBalanceError("Low Balance")

class Banking:

def main():

amount = int(input("Enter initial amount : "))

acc = Account(amount)

print("Balance in account : ", acc.getBalance())

amount = int(input("Enter amount to withdraw : "))

try:

acc.withdraw(amount)

except LowBalanceError as lbe:

print("Exception : ",lbe.getErrorMessage())

print("Final balance : ", acc.getBalance())

if \_\_name\_\_ == "\_\_main\_\_" :

Banking.main()