============================================

Exception Handling

============================================

Index

---------

=>Purpose of Exception Handling

=>Types of Errors

i) Compile Time Errors

ii) Logical Errors

iii) Runtime Errors

=>Definition of Exception

=>Definition of Exception handling

=>Types of Exceptions.

a) Pre-defined Exceptions

b) Programmer-Defined Exceptions

=>Handling Exceptions

=>Keywords for Handling Exceptions

1. try

2. except

3. else

4. finally

5. raise

=>Syntax for Handling Exceptions

=>Programming Examples

---------------------------------------------------------------------

=>Development of Programmer-Defined Exceptions

=>Programming Examples

--------------------------------------------------------------------

=>ATM Case Study

================================================

Building Blocks of Exception Handling

================================================

1. When the application user or end user enters Invalid Input then we get Run time Errors. (Invalid Input----->Run time Error)

2. By default Runtime Erros always generates Technical Error Messages.

-------------------------------------

3. Definition of Exception: Every Runtime Error is called Exception

------------------------------------- (Invalid Input---->Runtime Error------>Exception )

Hence Every Valid Input gives Result.

Every InValid Input gives Exception.

4. By default All exceptions gives Technical Error Messages.

-------------------------------------------------------

5. Definition of Exception Handling:

-------------------------------------------------------

=>The Process of converting Technical Error Messages into User-Friendly Error Messages is called Exception Handling

6. When Exception Occurs, Internally PVM perform 3 steps

a) PVM Terminates the program execution Abnormally.

b) PVM comes out of Program flow

c) PVM generates Technical Error Message by default.

7. To Perform Step-(a), Step-(b) and Step-(c) by PVM, PVM creates an object of appropriate exception class.

8. When an exception occurs, PVM creates an object of appropriate

exception class. Hence Every Invalid Input gives Exception and Every Exception considered as object of appropriate exception class

9. Hence Every Exception is python is Object of appropriate exception class.

==============================x=====================================

Types of Errors in Exception Handling

================================================

=>The Purpose of Exception Handling is that " To Build Robust (Strong) Applications ".

=>In Real Time, to develop any project, we must choose one programming language and by using that language, we develop, compile and execute the programs. During this process we get 3 types of Errors. They are

1. Compile Time Errors

2. Logical Errors

3. Runtime Errors

--------------------------------------------------------------------------------------------------------------------------------

1. Compile Time Errors

--------------------------------------------------------------------------------------------------------------------------------

=>Compile Time Errors are those which are occurring during Compilation Process.

=>Compile Time Errors occurs due to Syntaxes are not followed.

=>Compile Time Errors must be solved by Programmers at development time.

--------------------------------------------------------------------------------------------------------------------------------

2. Logical Errors

--------------------------------------------------------------------------------------------------------------------------------

=>Logical Errors are those which are occurring during Run time or Execution Process.

=>Logical Errors occurs due to Wrong Representation of Logic.

=>Logical Errors gives Wrong Results or Inconsistent Results.

=>Logical Errors must be solved by Programmers at development time.

-----------------------------------------------------------------------

3. Runtime Errors (Implementation Errors)

-----------------------------------------------------------------------

=>Runtime Errors are those which are occurring during Run time or Execution Process.

=>Runtime Errors occurs due to Wrong or Invalid Input Entered by End User or Application User.

=>By default, Runtime Error gives Technical Error Messages and these messages Understandable by Programmers but not by End Users. So, Industry is recommended to generate User-Friendly Error Messages by using Exception Handling.

--------------------------------------------------------------------------------------------------------------------------------

=================================

Handling the exception in Python

=================================

=>Handling the exception in Python is nothing but Coverting Technical Error Messages into User-Friendly Error Messages.

=>To do this convertion, In Python Programming we have 5 keywords. They are

1. try

2. except

3. else

4. finally

5. raise

-------------------------------------------------------------------------------------------------------------------

Syntax for handling the exceptions

-------------------------------------------------------------------------------------------------------------------

try:

Block of statements

Generating Exceptions

except <exception-class-name-1>:

Block of statetements generates

User-Friendly Error Messages

except <exception-class-name-2>:

Block of statetements generates

User-Friendly Error Messages

---------------------------------------------------

---------------------------------------------------

except <exception-class-name-n>:

Block of statetements generates

User-Friendly Error Messages

else:

Block of statetements generates

Results

finally:

Block of statetements executes

compulsorily

-------------------------------------------------------------------------------------------------------------------

Detailed Explanation of all Block---tomorrow

try block:

---------------

except

else

finally

=======================================

Types of Exceptions in Python

=======================================

=>In Python Programming, we have 2 types of exceptions. They are

1. Pre-Defined or Built-in Exceptions

2. Programmer / User / Custom-Defined Exceptions

---------------------------------------------------------

1. Pre-Defined or Built-in Exceptions

---------------------------------------------------------

=>These exceptions are defined and developed by Python Developers and they are available as part of Python Software and used by python Programmers for dealing Universal Problems.

=>Some of the Universal Problems are

a) Division by Zero (ZeroDivisionError)

b) Invalid number format ( ValueError )

c) Invalid Indices ( IndexError)

d) Wrong module name (ModuleNotFoundError ).........etc

------------------------------------------------------------------------------------------------------------------------------

2. Programmer / User / Custom-Defined Exceptions

------------------------------------------------------------------------------------------------------------------------------

=>These exceptions are defined and developed by Python Programmers and they are available as part of Python Project and used by other python Programmers who are in the project for dealing with Common Problems.

=>Some of the Common Problems are

a) Attempting to enter Invalid PIN

b) Attempting to enter Wrong User Name and Password

c) Attempting to withdraw more amount than existing

...............................etc

------------------------------------------------------------------------------------------------------------------------------

#Program for accepting two integer values and find their div

#Div1.py

print ("Program Execution Strated:")

s1=input("Enter First Value:")

s2=input("Enter Second Value:")

a=int(s1) #---------------------ValueError

b=int(s2) #---------------------ValueError

print ("First Value:{}".format(a))

print("Second Value:{}".format(b))

c=a/b #---------------------ZeroDivisionError

print("Div:{}".format(c))

print("Program Execution Ended:")

#Program for accepting two integer values and find their div

#Div2.py

try:

print("Program Execution Strated:")

s1=input("Enter First Value:")

s2=input("Enter Second Value:")

a=int(s1) #---------------------ValueError

b=int(s2) #---------------------ValueError

c=a/b #---------------------ZeroDivisionError

except ZeroDivisionError:

print("\nDON'T ENTER ZERO FOR DEN...")

except ValueError:

print("\nDON'T ENTER STRS, ALNUMS AND SYMBOLS")

else:

print("First Value:{}".format(a))

print("Second Value:{}".format(b))

print("Div:{}".format(c))

print("Program Execution Ended:")

finally:

print("\ni am from finally block:")

=================================================

Explanation for the keywords in handling exceptions

=================================================

-----------

1) try:

-----------

=>It is the block in which we write block statements generates exceptions. In other words, whatever the statements are generating exceptions, those statements must write within try block and try block is called exception monitoring block.

=>When the exception occurs in try block then PVM Comes out of try Block and executes appropriate except block

=>After executing appropriate except block, PVM never goes to try block to execute rest of the statements.

=>Every try block must be immediately followed by except block (otherwise we get Syntax Error)

=>Every try block must contain at least one except block and recommended to define / write multiple except blocks for generating multiple user-friendly error messages.

-----------------

2)except

-----------------

1) It is the block in which we write block of statements gives User-Friendly Error Messages. In Other words, except block supresses the technical error Messages and Generates User-Freindly Erros Mesasges and this block is called Exception Processing Block.

NOTE: - Handling Exception= try block + except block

2) except block will execute when there is an exception in try block.

3) Even we write multiple except block,PVM execute appropriate except block depends on type of exception occurs in try block.

4) The place of writing except block is after try block and before else block (if present)

-----------------------

3. else

------------------------

1) It is the block, in which we define Block of statements generates results and this block is known as Result generated Block

2) else block will execute where is no exception in try block.

3) Writing else block is Optional

4) The place of writing else is after except block and before finally (if present)

--------------------------------

4. finally

---------------------------------

1. It is the Block, in which we block of statements used for Relinquishing (Close / release / clean-up/give-up) the Resources (Files, Databases) which are obtained in try block.

2. Writing / Defining finally Block is Optional

3. finally block will execute compulsorily (if we write)

4. The place of writing finally is after else block (if it present)

==================================x====================================

Various Forms of except blocks

============================================

=>except block contains Variouis Forms. They are

---------------------------------------------------------------------------------------------------------------------------------------------------

Form-1: This Syntax handle one exception at a time and generates user-friendly Error message one at a time.

----------------------------------------------------------------------------------------------------------------------------------------------------

Syntax: try:

----------------------

----------------------

except <exception-class-name>:

------------------------------------

------------------------------------

Examples: Refer Div2.py Program

-----------------------------------------------------------------------------------------------------

Form-2:The feature is called "Multi Exception Handling Block"

Here with Single except block, we can handle multiple specific exceptions and generates multiple user-friendly error messges.

-----------------------------------------------------------------------------------------------------

try:

------------------------------------

------------------------------------

except (exceptio-class-1,exceptio-class-2,...exceptio-class-n):

-------------------------------------------

Multiple User-friendly Error Messages

------------------

Examples: Refer Div3.py Program

-----------------

-----------------------------------------------------------------------------------------------------

Form-3: Handling the Single Specific Exception with alias name

here alias name of exception can capture the Tech error of corresponding exception.

-----------------------------------------------------------------------------------------------------

Syntax: try:

----------------------

----------------------

except <exception-class-name-1> as alias name:

------------------------------------

------------------------------------

except <exception-class-name-2> as alias name:

------------------------------------

------------------------------------

------------------

Examples: Refer Div4.py Program

-----------------

-----------------------------------------------------------------------------------------------------

Form-4 : default except block

=>This default block will execute when no exception matches with exception exceptions.

Syntax: try:

----------------------

----------------------

except :

------------------------------------

------------------------------------

------------------

Examples: Refer Div5.py Program

-----------------

-----------------------------------------------------------------------------------------------------

Standard Syntax-1:

try:

Block of statements

Generating Exceptions

except <exception-class-name-1>:

Block of statetements generates

User-Friendly Error Messages

except <exception-class-name-2>:

Block of statetements generates

User-Friendly Error Messages

---------------------------------------------------

---------------------------------------------------

except <exception-class-name-n>:

Block of statetements generates

User-Friendly Error Messages

except : #default Exception block

default error messages

else:

Block of statetements generates

Results

finally:

Block of statetements executes

compulsorily

(OR)

ry:

------------------------------------

------------------------------------

except (exceptio-class-1,exceptio-class-2,...exceptio-class-n):

-------------------------------------------

Multiple User-friendly Error Messages

except : #default Exception block

default error messages

else:

Block of statetements generates

Results

finally:

Block of statetements executes

compulsorily

------------------

Examples: Refer Div6.py Program

-----------------

#Program for accepting two integer values and find their div

#Div2.py

try:

print("Program Execution Strated:")

s1=input("Enter First Value:")

s2=input("Enter Second Value:")

a=int(s1) #---------------------ValueError

b=int(s2) #---------------------ValueError

c=a/b #---------------------ZeroDivisionError

except ZeroDivisionError:

print ("\nDON'T ENTER ZERO FOR DEN...")

except ValueError:

print ("\nDON'T ENTER STRS, ALNUMS AND SYMBOLS")

else:

print ("First Value: {}”. format(a))

print ("Second Value: {}".format(b))

print("Div:{}".format(c))

print ("Program Execution Ended:")

finally:

print("\ni am from finally block:")

#Program for accepting two integer values and find their div

#Div3.py

try:

print ("Program Execution Strated:")

s1=input ("Enter First Value:")

s2=input ("Enter Second Value:")

a=int(s1) #---------------------ValueError

b=int(s2) #---------------------ValueError

c=a/b #---------------------ZeroDivisionError

except (ZeroDivisionError,ValueError): # Multi Exception Handling Block

print ("\nDON'T ENTER ZERO FOR DEN...")

print("DON'T ENTER STRS, ALNUMS AND SYMBOLS")

else:

print("First Value:{}".format(a))

print("Second Value:{}".format(b))

print("Div:{}".format(c))

print("Program Execution Ended:")

finally:

print("\ni am from finally block:")

#Program for accepting two integer values and find their div

#Div4.py

try:

print ("Program Execution Strated:")

s1=input("Enter First Value:")

s2=input("Enter Second Value:")

a=int(s1) #---------------------ValueError

b=int(s2) #---------------------ValueError

c=a/b #---------------------ZeroDivisionError

except ZeroDivisionError as z:

print(z) # division by zero

except ValueError as k:

print(k) # invalid literal for int() with base 10: '10abc'

else:

print("First Value:{}".format(a))

print("Second Value:{}".format(b))

print("Div:{}".format(c))

print("Program Execution Ended:")

finally:

print("\ni am from finally block:")

#Program for accepting two integer values and find their div

#Div5.py----(Not recommended in real time)

try:

print ("Program Execution Strated:")

s1=input ("Enter First Value:")

s2=input ("Enter Second Value:")

a=int(s1) #---------------------ValueError

b=int(s2) #---------------------ValueError

c=a/b #---------------------ZeroDivisionError

except:

print("Oooops., something went wrong!!!")

else:

print("First Value:{}".format(a))

print("Second Value:{}".format(b))

print("Div:{}".format(c))

print("Program Execution Ended:")

finally:

print("\ni am from finally block:")

#Program for accepting two integer values and find their div

#Div6.py----Kvr Programmer--defined code in 2022 Oct

try:

print("Program Execution Strated:")

s1=input ("Enter First Value:")

s2=input ("Enter Second Value:")

a=int(s1) #---------------------ValueError

b=int(s2) #---------------------ValueError

c=a/b #---------------------ZeroDivisionError

#Code in 2023 adding Sandeep

s="PYTHON"

print(s[10])

except ZeroDivisionError :

print("\nDON'T ENTER ZERO FOR DEN...")

except ValueError :

print("\nDON'T ENTER STRS, ALNUMS AND SYMBOLS")

except IndexError:

print("Index wrong plz check:")

except: # Default except block ---must be written at last

print("Oooops , some thing went wrong!!!")

else:

print("First Value:{}".format(a))

print("Second Value:{}".format(b))

print("Div:{}".format(c))

print("Program Execution Ended:")

finally:

print("\ni am from finally block:")

#MulTableEx1.py

n=int(input("Enter a number for generating Mul Table:"))

if(n<=0):

print("{} is invalid input".format(n))

else:

print("="\*50)

print("Mul Table for :{}".format(n))

print("=" \* 50)

i=1 # Init part

while(i<=10):

print("\t{} x {} = {}".format(n,i,n\*i))

i=i+1

else:

print("=" \* 50)

#Program for finding common words among multiple lines.

#CommonWords.py

line1=input("Enter First Line of text:")

line2=input("Enter Second Line of text:")

print("-"\*50)

print("Line1:{}".format(line1))

print("Line2:{}".format(line2))

print("-"\*50)

lst1=line1.split()

lst2=line2.split()

st1=set(lst1)

st2=set(lst2)

cw=st1&st2

print("Number Common words={}".format(len(cw)))

print("List of Common words={}".format(cw))

#Program for finding Frequency of Number of Occurences of Every Char of given line of text.

#HINT: aaakkkLL---Input

# a3 k 3 L 2--Output

#CountChars.py

line=input("Enter Line of Text:")

d={} # Empty dict

for ch in line:

if ch in d:

d[ch]=d[ch]+1

else:

d[ch]=1

else:

print("Given Line:{}".format(line))

for c,nc in d.items():

print("\t{}->{}".format(c,nc))

#Program for finding Sum of digits of a given number

#DigitsSumEx1.py

n=int(input("Enter a number:"))

if(n<=0):

print("{} invalid input".format(n))

else:

print("Given Number:{}".format(n))

s=0

while(n>0): # 4523

d=n%10

s=s+d

n=n//10

else:

print("Sum of digits of Given Number:{}".format(s))

#Program for finding Sum of digits of a given number

#DigitsSumEx2.py

n=int(input("Enter a number:"))

if(n<=0):

print("{} invalid input".format(n))

else:

n=str(n)

s=0

print("Given Number:{}".format(n)) # 4567

for v in n:

d=int(v)

s=s+d

else:

print("sum of digits of given number={}".format(s))

#Program for finding the factors of a given Number

#FactorsEx1.py

n=int(input("Enter a number for finding factors:"))

if(n<=0):

print("{} is invalid input:".format(n))

else:

for i in range(1,n+1):

if(n%i==0):

print("\t{}".format(i))

#Program for finding the factors of a given Number

#FactorsEx2.py

n=int(input("Enter a number for finding factors:"))

if(n<=0):

print("{} is invalid input:".format(n))

else:

for i in range(1,n//2+1):

if(n%i==0):

print("\t{}".format(i))

#Program generating Multable by using for loop

#MulTableForloop.py

n=int(input("Enter a Number for generating Mul table:"))

if(n<=0):

print("{} is invalid input:".format(n))

else:

print("-"\*50)

print("Mul table for {}".format(n))

print("-"\*50)

for i in range(1,11):

print("\t{} x {} = {}".format(n,i,n\*i))

else:

print("-"\*50)

#Program reading any value and find reverse of that value

#ReverseValEx1.py

val=input("Enter any val:")

revval=val[::-1]

print("Original value:{}".format(val))

print("Reversed value:{}".format(revval))

#Program reading any numerical value and find reverse of that value

#ReverseValEx2.py

n=int(input("Enter any val:"))

if(n<=0):

print("{} is invalid input:".format(n))

else:

print("Original value:{}".format(n)) # 3456-------------------------

rv=0

while(n>0):

d=n%10

rv=rv\*10+d

n=n//10

else:

print("Reversed Number:{}".format(rv))

#Program for deciding whether string is palindrome or not

#StringPalinEx1.py

s=input("Enter any String:")

rs=s[::-1]

if(s==rs):

print("Given String is Palindrome")

else:

print("Given String is Not a Palindrome")

#Program for deciding whether string is palindrome or not

#StringPalinEx2.py

s=input("Enter any Value:")

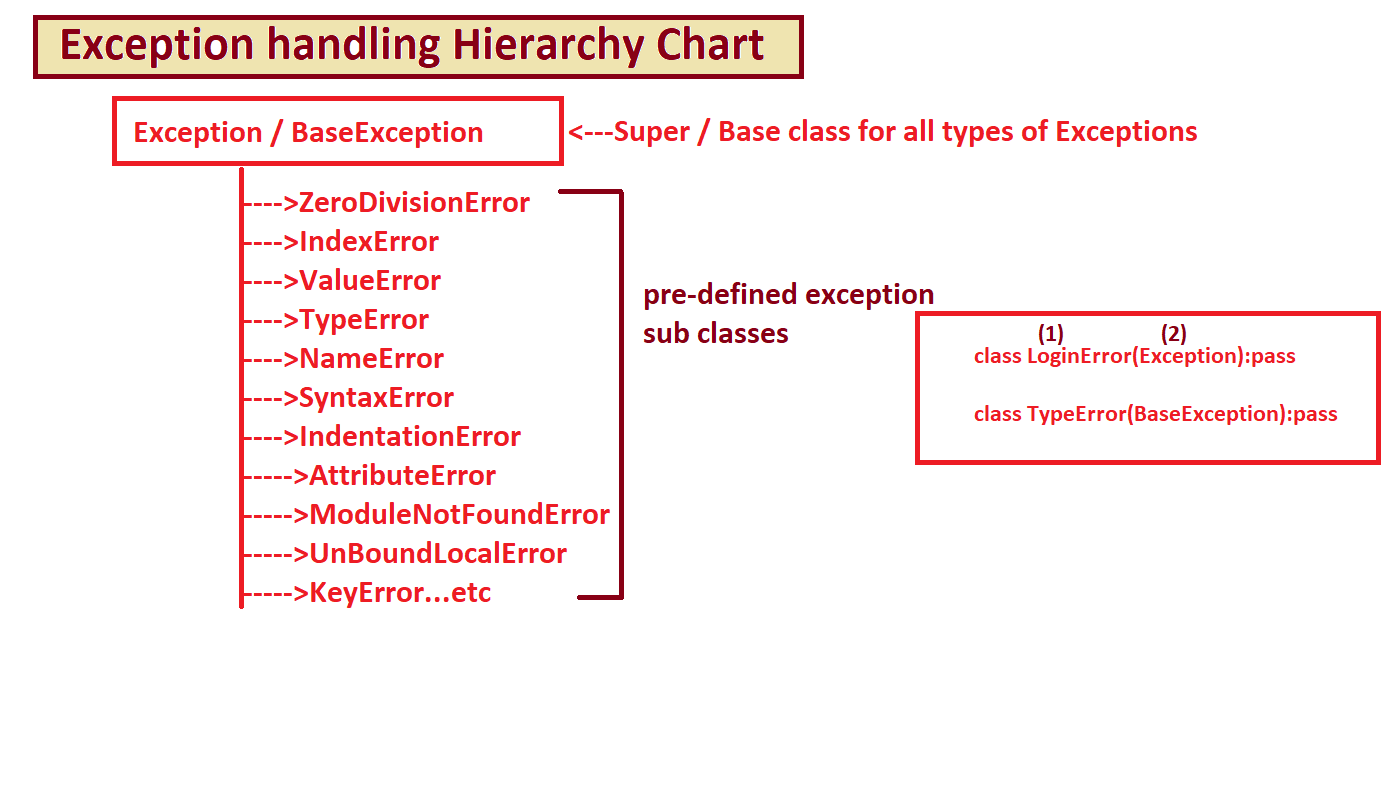
rs=s[::-1]

if(s==rs):

print("Given String is Palindrome")

else:

print("Given String is Not a Palindrome")



=============================================

Development of Programmer-Defined Exceptions

=============================================

=>These exceptions are defined and developed by Python Programmers and they are available as part of Python Project and used by other python Programmers who are in the project for dealing with Common Problems.

=>Some of the Common Problems are

a) Attempting to enter Invalid PIN

b) Attempting to enter Wrong User Name and Password

c) Attempting to withdraw more amount than existing

...............................etc

--------------------------------------------------------------------------------------------------------------------

=>Steps for Developing Programming Exceptions.

--------------------------------------------------------------------------------------------------------------------

Step-1: Choose the Programmer-defined class name

Step-2: The Programmer-defined class name must Inherit from Base class exception called "Exception or Base Exception"

Step-3: The above development must be saved on some file name with an extension.py (Module Name)

-----------------------------------------------------------------------

Example: class Login Error (Exception):pass

class InSuffBal(BaseException):pass

=>Here LoginError and InSuffBal are comes under Programmer-defined Exception sub classes.

-----------------------------------------------------------------------

=================================================

raise key word

=================================================

=>raise keyword is used for hitting / raising / generating the exception provided some condition must be satisfied.

=>raise keyword always used inside of Function Definition only.

=>PVM uses raise keyword implicitly for hitting pre-defined Exception whereas Programmer makes the PVM to use raise keyword explicitly for Hitting or Generating Programmer-defined Exceptions.

=>Syntax:- if (Test Cond):

raise <exception-class-name>

=>Syntax:- def functionname(list of formal parms if any):

-----------------------------------------------------

if (Test Cond):

raise <exception-class-name>

--------------------------------------------------------

Examples:

-------------------

from kvr import KvrDivisionError

def division(a,b):

if(b==0):

raise KvrDivisionError

else:

return (a/b)

-----------------------------------------------------------------------

#division.py--File Name and Module name

from kvr import KvrDivisionError

def division(a,b): # Common Function ---Phase-II

if(b==0):

raise KvrDivisionError # Hitting or generating the exception

else:

return (a/b)

#Main Program

#DivisionDemo.py

from division import division

from kvr import KvrDivisionError

try:

a=int(input("Enter Value of a:"))

b=int(input("Enter Value of b:"))

res=division(a,b)#Function Call

except KvrDivisionError: # Phase-3--Handling the exception

print ("\nDon't Enter Zero for Den...")

except ValueError:

print ("\nDon't enter strs, alnums and symbols")

else:

print("Division=“, res)

#MulTabExcept.py--Phase=I

class NegativeNumberError(Exception):pass

class ZeroError(BaseException):pass

#MulTable.py--File Name and Module Name--Phase-II

from MulTabExcept import NegativeNumberError,ZeroError

def table(n):

if(n<0): # Hit the exception--NegativeNumberError

raise NegativeNumberError

elif(n==0): # Hit the exception --ZeroError

raise ZeroError

elif(n>0):

print("="\*50)

print("Mul Table for :{}".format(n))

print("=" \* 50)

for i in range(1,11):

print("\t{}x{}={}".format(n,i,n\*i))

else:

print("=" \* 50)

#MulTableDemo.py---main program--Phase-III

from MulTable import table

from MulTabExcept import NegativeNumberError,ZeroError

n=int(input("Enter a Number:"))

try:

table(n) #

except NegativeNumberError:

print ("Don't Enter -Ve Number")

except ZeroError:

print("Don't enter Zero for Mul table")

except:

print ("Something went wrong!!!")

finally:

print ("I am from finally Block")

#AtmExcept.py--File Name and Module name

class DepositError(Exception):pass

class WithdrawError(BaseException):pass

class InSuffFundError(Exception):pass

#AtmMenu.py---File Name and module name

def menu():

print("-"\*50)

print("\tATM OPerations")

print("-"\*50)

print("\t1.Deposit")

print("\t2.Withdraw")

print("\t3.Bal Enq")

print("\t4.Exit")

print("-"\*50)

#AtmOperations.py--File and module

from AtmExcept import DepositError,WithdrawError,InSuffFundError

bal=500.00 # Global variables

def deposit ():

damt=float(input("Enter how much amount u want to deposit:"))#implcitly raises ValueError

if(damt<=0): # Exception Involved

raise DepositError # Hitting or raising exception explcitly

if(damt>0):

global bal

bal=bal+damt

print("Ur Account xxxxxxx123 credited with INR:{}".format(damt))

print ("Now Account xxxxxxx123 Balanace after depositINR:{}".format(bal))

def withdraw():

global bal

wamt=float(input("Enter how much amount u want to Withdraw:"))#implcitly raises ValueError

if(wamt<=0):

raise WithdrawError # Hitting or raising exception explcitly

elif((wamt+500)>bal):

raise InSuffFundError # Hitting or raising exception explcitly

else:

bal=bal-wamt

print("Ur Account xxxxxxx123 debited with INR:{}".format(wamt))

print("Now Account xxxxxxx123 Balanace after withdraw INR:{}".format(bal))

def balenq():

print("Ur Account xxxxxxx123 BalanceINR:{}".format(bal))

#AtmDemo.py-----File Name and Main Program

import sys

from AtmMenu import menu

from AtmOperations import deposit,withdraw,balenq

from AtmExcept import DepositError,WithdrawError,InSuffFundError

while(True):

menu()

try:

ch=int(input("Enter Ur Choice:"))

match(ch):

case 1:

try:

deposit()

except ValueError:

print("Don't enter strs, alnums and symbols as deposit amt:")

except DepositError:

print("Don't Deposit -Ve and Zero Value:")

case 2:

try:

withdraw()

except ValueError:

print("Don't enter strs, alnums and symbols as withdraw amt:")

except WithdrawError:

print("Don't withdraw -Ve and Zero Value:")

except InSuffFundError:

print("Ur Account does not have suff Funds--Read Python Notes")

case 3:

balenq()

case 4:

print("Thx for using this program")

sys.exit()

case \_:

print("Ur Selection of Operation is Wrong-try again")

except ValueError:

print("Don't enter strs, alpnums and symbols for choice of Operations:")

#Maxpalindromeex1.py

lst=[ 1, 232, 5545455, 999999, 1212 , 8558 ]

ml=0

cl=0

pos=-1

for i in range(0,len(lst)):

cv=str(lst[i])

if(cv==cv[::-1]):

cl=len(cv)

if(cl>ml):

ml=cl

pos=i

else:

print("Max palindrome Value={}".format(lst[pos]))

#Maxpalindromeex2.py

lst=[ "mom","madam","liril","malayalam","racecar"]

ml=0

cl=0

pos=-1

for i in range(0,len(lst)):

cv=str(lst[i])

if(cv==cv[::-1]):

cl=len(cv)

if(cl>ml):

ml=cl

pos=i

else:

print("Max palindrome Value={}".format(lst[pos]))

#Maxpalindromeex3.py

lst=[ 1, 232, 5545455, 999999, 1212 , 8558 ]

lst=[ "mom","car","madam","liril","mala","racecar"]

ltp=[(val,len(val)) for val in lst if val==val[::-1] ]

print(ltp) # [('mom', 3), ('madam', 5), ('liril', 5), ('malayalam', 9), ('racecar', 7)]

d=dict(ltp)

val=max(d.values()) # val=9

maxpalin=[ k for k,v in d.items() if v==val]

print("\nMax Lengh palindrome=",maxpalin)

#Maxpalindromeex4.py

lst=[ 1, 232, 5545455, 999999, 1212 , 8558 ]

ltp=[(str(val), len(str(val))) for val in lst if str(val)==str(val)[::-1] ]

print(ltp) #

d=dict(ltp)

val=max(d.values())

maxpalin=[ k for k,v in d.items() if v==val]

print("\nMax Lengh palindrome=",maxpalin[0])