Error (or) Exception Handling

Errors are the problems in a program due to which the program will stop the execution. On the other hand, exceptions are raised when some internal events occur which changes the normal flow of the program.   
Two types of Errors may occur in python. 

1. Syntax errors
2. Logical errors (Exceptions)

Error handling is a concept which is used to provide alternate solution for any error occurred and continue the flow of execution smoothly till the end of the program.

**Common Exceptions**

Python provides the number of built-in exceptions, but here we are describing the common standard exceptions. A list of common exceptions that can be thrown from a standard Python program is given below.

**ZeroDivisionError**: Occurs when a number is divided by zero.

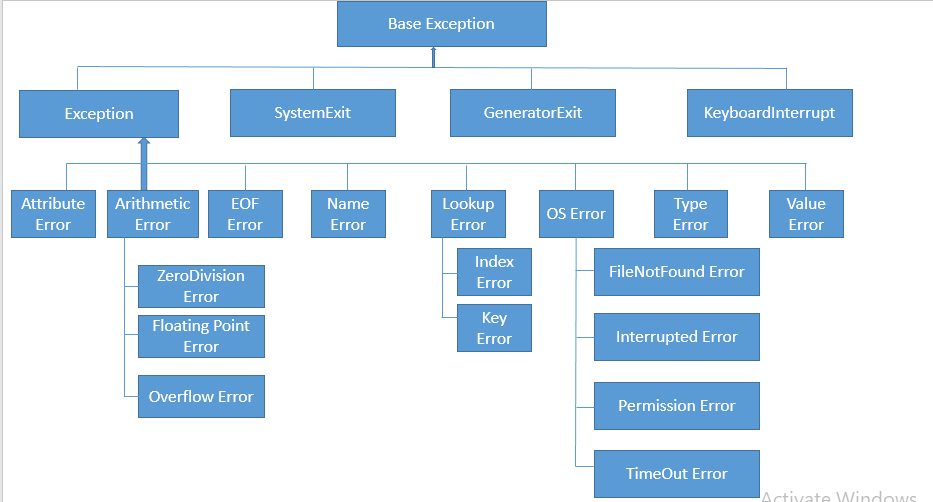
**NameError**: It occurs when a name is not found. It may be local or global.

**IndentationError**: If incorrect indentation is given.

**IOError**: It occurs when Input Output operation fails.

**EOFError**: It occurs when the end of the file is reached, and yet operations are being performed.

The following diagram shows list of errors hierarchy:



To achieve error handling in python we use the following keywords:

**try,**

**except**

**else**

**raise**

**finally**

**try**: Place the statements which are prone to error in try block.

**except**: Prove the alternate solution by catching the error in the except block.

Example 1:

# Error Handling

# try with except block

a,b = input("Enter two numbers: ").split(' ')

a = int(a)

b = int(b)

try:

    c = a/b

    print("Division result is: ", c)

except ZeroDivisionError:

    print("Pls do not enter zeros")

print("End of operation")

print("End of program...")

print("Exiting from program...")

Example 2:

# Handle multiple exceptions with a single except clause

try:

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

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

    c = a / b

    print("The answer of a divide by b:", c)

except(ValueError, ZeroDivisionError):

    print("Please enter a valid value")

Example 3:

# A single try block can have any no of except blocks

# each except block should handle an individual error

a,b = input("Enter two numbers: ").split(' ')

try:

    a = int(a)

    b = int(b)

    c = a/b

    print("Division result is: ", c)

except ZeroDivisionError:

    print("Pls do not enter zeros")

except ValueError:

    print("Enter only numbers...")

print("End of operation")

print("End of program...")

print("Exiting from program...")

Example 4:

**else**:

Else block is optional.

It executes when try is successfully executed.

There should be only one else block per try.

Else block should be written only after except block.

# try with else and except blocks

a,b = input("Enter two numbers: ").split(' ')

a = int(a)

b = int(b)

try:

    c = a/b

    print("Division result is: ", c)

except ZeroDivisionError:

    print("Pls do not enter zeros")

except ValueError:

    print("Enter only numbers...")

else:

    print("Else block will execute only when try has no error")

print("End of operation")

print("End of program...")

print("Exiting from program...")

Example 5:

**raise**:

raising an error explicitly is used to check the error handling nature of a program.

Raise can be used in any block.

# 'raise' is used to raise an error explicitly(intentionally)

a,b = input("Enter two numbers: ").split(' ')

a = int(a)

b = int(b)

try:

    raise TypeError("Just for fun")

except ZeroDivisionError:

    print("Pls do not enter zeros")

except ValueError:

    print("Enter only numbers...")

else:

    print("Else block will execute only when try has no error")

print("End of operation")

print("End of program...")

print("Exiting from program...")

Example 5.1:

def simple\_interest(amount, year, rate):

    try:

        if rate > 100:

            raise ValueError(rate)

        interest = (amount \* year \* rate) / 100

        print('The Simple Interest is', interest)

        return interest

    except ValueError:

        print('interest rate is out of range', rate)

print('Case 1')

simple\_interest(800, 6, 8)

print('Case 2')

simple\_interest(800, 6, 800)

Example 6:

**finally**:

This is block is used to execute those statements which are mandatory for execution.

Despite to error handled or not, finally block will get executed.

Generally, closure statements are written in finally block.

There should be only one finally block per try.

Finally, block should be the last block among the error handling blocks.

# try with else, except and finally blocks

a,b = input("Enter two numbers: ").split(' ')

a = int(a)

b = int(b)

try:

    raise TypeError("Just for fun")

    c=a/b

    print("Result is: ", c)

except ZeroDivisionError:

    print("Pls do not enter zeros")

except ValueError:

    print("Enter only numbers...")

else:

    print("Else block will execute only when try has no error")

finally:

    print("no matter what happens, finally gets exected at all times...")

print("End of operation")

print("End of program...")

print("Exiting from program...")

Example 7:

# try with default Exception class

a,b = input("Enter two numbers: ").split(' ')

a = int(a)

b = int(b)

try:

    raise AttributeError("Just for fun")

except ZeroDivisionError:

    print("Pls do not enter zeros")

except ValueError:

    print("Enter only numbers...")

except Exception as e:

    print(e, 'has been handled here')

else:

    print("Else block will execute only when try has no error")

print("End of operation")

print("End of program...")

print("Exiting from program...")

Example 8:

# try with finally block

try:

    print("test case...")

    raise ZeroDivisionError()

finally:

    print("This is finally....")

Example 9:

# try with mulitiple raised errors.

# only the first error can be handled.

try:

    raise ZeroDivisionError()

    raise AttributeError()

except ZeroDivisionError:

    print("ZDE handled...")

except AttributeError:

    print("AE handled...")

Example 10:

# nested try block

try:

    print("first line")

    print("second line")

    try:

        print("third line")

    except:

        print("This is except for innner try block")

except:

    print("This is except for outer try...")

Example 11:

# try with function

def calc():

    x,y = input("Enter two values: ").split()

    x,y = int(x),int(y)

    try:

        z = x/y

        print("Result is:", z)

    except ZeroDivisionError:

        print("ZDE handled...")

    pass

calc()

Example 12:

# error propagation

def calc():

    x,y = input("Enter two values: ").split()

    x,y = int(x),int(y)

    z = x/y

    print("Result is:", z)

    pass

try:

    calc()

except ZeroDivisionError:

    print("ZDE handled...")

#error propagation

def a():

    print(10/0)

def b():

    a()

def c():

    try:

        b()

    except ZeroDivisionError:

        print("ZDE handled in fun c")

c()

Example 13:

# rethrowing an error

def one():

    try:

        print(10/0)

    except ZeroDivisionError as zde:

        raise zde

def two():

    try:

        one()

    except ZeroDivisionError:

        print("ZDE handled...")

two()

## Custom and User-defined Exceptions

Sometimes we have to define and raise exceptions explicitly to indicate that something goes wrong. Such a type of exception is called a **user-defined exception**or**customized exception.**

The user can define custom exceptions by creating a new class. This new exception class has to derive either directly or indirectly from the built-in class Exception. In Python, most of the built-in exceptions also derived from the Exception class.

Example 1:

class Error(Exception):

    """Base class for other exceptions"""

    pass

class ValueTooSmallError(Error):

    """Raised when the input value is small"""

    pass

class ValueTooLargeError(Error):

    """Raised when the input value is large"""

    pass

while(True):

    try:

        num = int(input("Enter any value in 10 to 50 range: "))

        if num < 10:

            raise ValueTooSmallError

        elif num > 50:

            raise ValueTooLargeError

        break

    except ValueTooSmallError:

            print("Value is below range..try again")

    except ValueTooLargeError:

            print("value out of range...try again")

print("Great! value in correct range.")

Explanation: In the above example, we create two custom classes or user-defined classes with names, ValueTooSmallError and ValueTooLargeError.When the entered value is below the range, it will raise ValueTooSmallError and if the value is out of then, it will raise ValueTooLargeError.

### Customizing Exception Classes

We can customize the classes by accepting arguments as per our requirements. Any custom exception class must be Extending from BaseException class or subclass of BaseException.

In the above example, we create a custom class that is inherited from the base class Exception. This class takes one argument age. When entered age is negative, it will raise NegativeAgeError.

Example 2:

class NegativeAgeError(Exception):

    def \_\_init\_\_(self, age, ):

        message = "Age should not be negative"

        self.age = age

        self.message = message

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

if age < 0:

    raise NegativeAgeError(age)

# Output:

# raise NegativeAgeError(age)

# \_\_main\_\_.NegativeAgeError: -9

Example 3:

def sum\_of\_list(numbers):

    return sum(numbers)

def average(sum, n):

    # ZeroDivisionError if list is empty

    return sum / n

def final\_data(data):

    for item in data:

        print("Average:", average(sum\_of\_list(item), len(item)))

list1 = [10, 20, 30, 40, 50]

list2 = [100, 200, 300, 400, 500]

# empty list

list3 = []

lists = [list1, list2, list3]

final\_data(lists)

Example 4:

# A python program to create user-defined exception

# class MyError is derived from super class Exception

class MyError(Exception):

    # Constructor or Initializer

    def \_\_init\_\_(self, value):

        self.value = value

    # \_\_str\_\_ is to print() the value

    def \_\_str\_\_(self):

        return(repr(self.value))

try:

    raise(MyError(3\*2))

# Value of Exception is stored in error

except MyError as error:

    print('A New Exception occured: ',error.value)

Example 5:

# class Error is derived from super class Exception

class Error(Exception):

    # Error is derived class for Exception, but

    # Base class for exceptions in this module

    pass

class TransitionError(Error):

    # Raised when an operation attempts a state

    # transition that's not allowed.

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

        self.prev = prev

        self.next = nex

        # Error message thrown is saved in msg

        self.msg = msg

try:

    raise(TransitionError(2,3\*2,"Not Allowed"))

# Value of Exception is stored in error

except TransitionError as error:

    print('Exception occured: ',error.msg)

Example 6:

# NetworkError has base RuntimeError

# and not Exception

class Networkerror(RuntimeError):

    def \_\_init\_\_(self, arg):

        self.args = arg

try:

    raise Networkerror("Error")

except Networkerror as e:

    print (e.args)