Python Programming

Narendra Allam

Chapter 12

Exception Handling

Topics Covering

- Pupose of exception handling
- try - except
- else and finally
- Types of exceptions
- Exception class
- Exceptions order
- Custom Exceptions

Exception Handling
Pupose of exception handling is system continuity. A program aborts on the occurance of an error. There are various error caused by different operations. For example when we are trying to convert a string to an int, there is a scope to get ValueError, when string contains a float instead of int. A string with non-numeric characters also causes the ValueError. IOError occures when a file doesn't exist. A KeyError occures when a key is referred which is not exisiting in a dictionary. If we have independent functionalities, error in one functionality should not stop processing of other functionalities in the program. This is where exception handling helps us. Some times it is required to give exception to some errors, and need to proceed for further processing. in python we have try-except block to handle this.

Syntax:

```
try:
    # code...
except <ExceptionClass1> as <exceptionObject>:
    # handling mechanism if any...
except <ExceptionClass2> as <exceptionObject>:
    # handling mechanism if any...
except <ExceptionClass3> as <exceptionObject>:
    # handling mechanism if any...
# .
# .
else:
    # This is executed on the success of try block
finally:
    # This is executed irrespective of success of try
```

Each error has a type. 'ValueError', 'IOError', 'KeyError' are few exception class types. When we get an error, an except block with matching *ExcpetionClass* is executed, program gets aborted if no matching ExceptionClass found.

Except blocks should be in reverse order of their inheritance hierarchie

In [1]:

```
customers = {1234: 1000, 1235: 2000, 1236: 4000, 1237: 1500, 1239: 1000}
balances = [(1234, '2700'),
            (1235, '2600'),
            (1236, '0'),
            (1237, '2900$'),
            (1234, '3200'),
            (1299, '2400'),
            (1236, '2100'),
            (1235, '2300.0'),
            (1237, '2200'),
            (1239, '2000')]


def cust_deposit_processing(balances, total):
    eod_bal = 0.0
    unprocessed_bals = []
    projected_interest = 0.0
    for custid, bal in balances:
        print ('Custmer id {} processing balance {}'.format(custid, bal))
        try:
            amount = int(bal)
            interest = amount * 0.09
            print ('Balance: {}, Interest: {}, contribution%:{}'.format(bal,
                                                                        interest,
                                                                        total / amou
            customers[custid] += amount
            eod_bal += amount
            projected_interest += interest
        except ValueError as ex:
            print('Exception : {}, unprocessed balance: {}'.format(ex, bal))
            unprocessed_bals.append(bal)
        except ZeroDivisionError as ex:
            print(ex)
            unprocessed_bals.append(bal)
        except Exception as ex:
            print ('Unhandled exception occured while processing:', ex)
            raise ex
    print ('Total eod balance: {}, expect bal: {}'.format(eod_bal, total))
    print ('Unprocessed bals:', unprocessed_bals)

def loan_processing():
    print ('Loan processing done!')


def credit_card_processing():
    print ('Credit Card processing done!')

def process():
    expected_total = 20000
    global balances
    try:
        cust_deposit_processing(balances, expected_total)
    except Exception as ex:
        print ('Balance Processing stopped, exception:', ex)
        print ('Backup has been taken successfully!')

    try:
        loan_processing()
    except Exception as ex:
```

```
        print ('Backup has been taken successfully!')

    try:
        credit_card_processing()
    except Exception as ex:
        print ('Backup has been taken successfully!')

if __name__ == '__main__':
    process()
```

```
Custmer id 1234 processing balance 2700
Balance: 2700, Interest: 243.0, contribution%:7.407407407407407
Custmer id 1235 processing balance 2600
Balance: 2600, Interest: 234.0, contribution%:7.6923076923076925
Custmer id 1236 processing balance 0
division by zero
Custmer id 1237 processing balance 2900$
Exception : invalid literal for int() with base 10: '2900$', unprocess
ed balance: 2900$
Custmer id 1234 processing balance 3200
Balance: 3200, Interest: 288.0, contribution%:6.25
Custmer id 1299 processing balance 2400
Balance: 2400, Interest: 216.0, contribution%:8.333333333333334
Unhandled exception occured while processing: 1299
Balance Processing stopped, exception: 1299
Backup has been taken successfully!
Loan processing done!
Credit Card processing done!
```

In [4]:

```
import datetime
import mysql.connector
from mysql.connector import errorcode
emp_list = []
class Employee(object):
    def __init__(self, _id, _dob, _fname, _lname, _sex, _hdate):
        self.empId = _id
        self.dob = _dob
        self.firstName = _fname
        self.lastName = _lname
        self.gender = _sex
        self.hireDate = _hdate
    def __str__(self):
        return '{}, {}, {}, {}, {}, {}'.format(self.empId, self.dob,
                                               self.firstName, self.lastName,
                                               self.gender, self.hireDate)
    def __repr__(self):
        return 'Employee({}, {}, {}, {}, {}, {})'.format(self.empId, self.dob,
                                               self.firstName, self.lastName,
                                               self.gender, self.hireDate)
def process():
    try:
        conn   = mysql.connector.connect(user   = 'naren',
                                         password = 'Python@7',
                                         host     = '127.0.0.1',
                                         database = 'employees')

        query  = "select * from employees limit 20"
        cursor = conn.cursor()
        cursor.execute(query)
        emp_list = []
        '''
        for empid, bdate, lname, fname, gender, hdate in cursor:
            emp_list.append(Employee(empid, bdate, lname, fname, gender, hdate))
        '''
        for rec in cursor:
            # print(rec)
            emp_list.append(Employee(*rec))
    except mysql.connector.Error as err:

        if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
            print("Name or password error! :( ")
        elif err.errno == errorcode.ER_BAD_DB_ERROR:
            print("Database doesn't exist!")
        else:
            print(err)

    else:
        print('Inside else')
        cursor.close()
        conn.close()
        return

    finally:
        print('Transaction backup has been taken successfully!')
        print ('Shutting down the system')
process()
```

```
Inside else
```

```
Transaction backup has been taken successfully!
Shutting down the system
```

Writing custom exceptions classs

In [5]:

```
class CustException(Exception):
    def __init__(self, *args):
        self.args = args
        self.message = 'Custom Exception'
    def __str__(self):
        return self.message

e = CustException()

raise e
```

```
---------------------------------------------------------------------------
-----
CustException                             Traceback (most recent call
 last)
<ipython-input-5-f6773b8e7660> in <module>()
      8 e = CustException()
      9
---> 10 raise e

CustException: Custom Exception
```

In [6]:

```
raise Exception('My Excpetion')
```

```
---------------------------------------------------------------------------
-----
Exception                                 Traceback (most recent call
 last)
<ipython-input-6-97a8b958448d> in <module>()
----> 1 raise Exception('My Excpetion')

Exception: My Excpetion
```