**Experiment No. 7**

**Title: Error Handling/Exception Handling.**

**Batch: B1 Roll No: 1914078 Experiment No.:7**

**Aim:** To introduce Error Handling/Exception Handling in Python .

**Resources needed:** Python IDE

### Theory:

### A Python program terminates as soon as it encounters an error. In Python, an error can be a syntax error or an exception.

### An exception is an error or abnormal condition that happens during execution of a program which can cause termination of program. When that error occurs, Python generate an exception that can be handled, which avoids your program to crash. Tracebacks are generated due to runtime errors. Example:  IndexError, ImportError, IOError, ZeroDivisionError, TypeError.

### print( 0 / 0)

### Traceback (most recent call last):

### File "<stdin>", line 1, in <module>

### ZeroDivisionError: integer division or modulo by zero

### Syntax errors occur when the parser detects an incorrect statement due to  improper structure (syntax) of the language .

### Example:

### print( 0 / 0 ))

### File "<stdin>", line 1 print( 0 / 0 )) ^

### SyntaxError: invalid syntax

Python has many built-in exceptions which forces your program to output an error when something in it goes wrong. When these exceptions occur, it causes the current process to stop and passes it to the calling process until it is handled. If not handled, our program will crash. **Exception** is the base class for all the exceptions in python. Python provides us some basic exception classes which are already defined under Exception class and can be used in generic cases

|  |  |
| --- | --- |
| **Exception Class** | **Event** |
| ArithmeticError | Raised when numeric calculations fails |
| FloatingPointError | Raised when a floating point calculation fails |
| ZeroDivisionError | Raised when division or modulo by zero takes place for all numeric types |
| AssertionError | Raised when Assert statement fails |
| OverflowError | Raised when result of an arithmetic operation is too large to be represented |
| ImportError | Raised when the imported module is not found |
| IndexError | Raised when index of a sequence is out of range |
| KeyboardInterrupt | Raised when the user interrupts program execution, generally by pressing Ctrl+c |
| IndentationError | Raised when there is incorrect indentation |
| SyntaxError | Raised by parser when syntax error is encountered |
| KeyError | Raised when the specified key is not found in the dictionary |
| NameError | Raised when an identifier is not found in the local or global namespace |
| TypeError | Raised when a function or operation is applied to an object of incorrect type |

### Exception Handling Techniques in Python

### Try block:  Python executes code following the try statement as a “normal” part of the program. All statements are executed until an exception is encountered.

### try:

### a=5

### b='0'

### print(a/b)

### except:

### print('Some error occurred.')

### print("Out of try except blocks.")

### Except block: The code that follows the except statement is the program’s response to any exceptions in the preceding tryclause. It is used to catch and handle the exception(s) that are encountered in the try clause.

### #try with multiple except blocks

### try:

### a=5

### b=0

### print (a/b)

### except TypeError:

### print('Unsupported operation')

### except ZeroDivisionError:

### print ('Division by zero not allowed')

### print ('Out of try except blocks')

### Else Block:  instruct a program to execute a certain block of code only in the absence of exceptions. It lets you code sections that should run only when no exceptions are encountered in the try clause.

### Finally: finally enables you to execute sections of code that should always run, with or without any previously encountered exceptions.

### try:

### print("try block")

### x=int(input('Enter a number: '))

### y=int(input('Enter another number: '))

### z=x/y

### except ZeroDivisionError:

### print("except ZeroDivisionError block")

### print("Division by 0 not accepted")

### else:

### print("else block")

### print("Division = ", z)

### finally:

### print("finally block")

### x=0

### y=0

### print ("Out of try, except, else and finally blocks." )

### How to raise an exception in Python?

### raise allows you to throw an exception at any time ie raise statement allows the programmer to force a specified exception to occur. It is used to throw an exception if a condition occurs. The statement can be complemented with a custom exception as well.

### x = 10

### if x > 5:

### raise Exception('x should not exceed 5. The value of x was: {}'.format(x))

### assert enables to verify if a certain condition is met and throw an exception if it isn’t. It prevents program from crashing.  If the condition is True, then program can continue. If the is False, then program throw an AssertionError exception.

### num=int(input('Enter a number: '))

### assert num>=0

### print('You entered: ', num)

### User Defined Exceptions

### Programmers may name their own exceptions by creating a new exception class. Exceptions need to be derived from the Exception class, either directly or indirectly.

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)

### Activity:

### Write a Python program to define a class Calculator that defines following operations: add, subtraction, multiply, division and power. Raise following exceptions ArithmeticError, FloatingPointError, ZeroDivisionError, OverflowError, AttributeError and also have multiple except blocks

### Write a Python program to check whether entered name by the user consists of only alphabets. If so display name entered successfully; else raise an exception and display the message “Name not entered in prescribed format”

### Code/Program:

### 1. Program

class Calculator():

    def \_\_init\_\_(self, a=0, b=0):

        self.a = a

        self.b = b

    def add(self):

        try:

            if self.a > 5000000 or self.b > 5000000:

                raise OverflowError('Very large values')

            else:

                return self.a+self.b

        except OverflowError:

            print('Values too large to handle')

    def sub(self):

        try:

            if self.a > 5000000 or self.b > 5000000:

                raise OverflowError('Very large values')

            elif self.a-self.b != (self.a-(int(self.a))-(self.b-int(self.b))):

                raise FloatingPointError('Inaccurate answer')

            else:

                return self.a-self.b

        except FloatingPointError:

            print('Accurate answer could not be calculated')

        except OverflowError:

            print('Values too large to handle')

    def mul(self):

        try:

            if self.a > 5000000 or self.b > 5000000:

                raise OverflowError('Very large values')

            else:

                return self.a\*self.b

        except OverflowError:

            print('Values too large to handle')

    def div(self):

        try:

            if self.b == 0:

                raise ZeroDivisionError("Cannot divide a number by 0")

            else:

                return self.a/self.b

        except ZeroDivisionError:

            print('Cannot divide a number by 0')

        except OverflowError:

            print('Values too large to handle')

    def power(self):

        try:

            if self.a < 0 or self.b < 0:

                raise ValueError('Values entered should be non-negative')

            if self.a > 5000000 or self.b > 5000000:

                raise OverflowError('Very large values')

            else:

                return self.a \*\* self.b

        except OverflowError:

            print('Values Too large to handle')

        except ValueError:

            print('Values entered should be non-negative')

### 

### 

### 

### Output:

### Graphical user interface, text, application, email Description automatically generated

### 2. Program:

s = input('Enter a Name(only alphabets):\t')

try:

    if s.isalpha():

        print(s)

        print("Name entered Sucessfully")

    elif s.isalpha() == False:

        raise TypeError("Only alphabets are allowed")

except TypeError:

    print('Name not entered in prescribed format')

### Output:

### 

### 

### Post Tutorial Questions:

### 1.What do you mean by chaining exceptions? Give Example.

**Ans:** The raise statement allows an optional from which enables chaining exceptions. This can

be useful when we are transforming exceptions. Chaining exception happens

automatically when an exception is raised inside an except or finally section.

Example:

**>>> def** func():

**...**  **raise** IOError

**...**

**>>> try**:

**...**  func()

**... except** IOError **as** exc:

**...**  **raise** RuntimeError('Failed to open database') **from** **exc**

**...**

Traceback (most recent call last):

File "<stdin>", line 2, in <module>

File "<stdin>", line 2, in func

OSError

The above exception was the direct cause of the following exception:

Traceback (most recent call last):

File "<stdin>", line 4, in <module>

RuntimeError: Failed to open database

Exception chaining can be disabled by using from None idiom:

**try**:

**...**  open('database.sqlite')

**... except** IOError:

**...**  **raise** RuntimeError **from** None

**...**

Traceback (most recent call last):

File "<stdin>", line 4, in <module>

RuntimeError

**Outcomes:**

**CO3:** Apply Object Oriented Programming Concepts in Python

**Conclusion: (Conclusion to be based on the objectives and outcomes achieved)**

Through this experiment we learnt the concepts of error handling/exception handling and

implemented them using python programs.

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of faculty in-charge with date**

**References:**

**Books:**

1. Reema Thareja , *Python Programing: Using Problem Solving Approach,* Oxford University Press, First Edition 2017, India
2. Dr. R. Nageswara Rao, *Core Python Programming,* Wiley Publication, Second Edition 2018,India
3. Sheetal Taneja and Naveen Kumar, *Python Programing: A Modular Approach,* Pearson India
4. Swarroop C.H, *Byte of python,* e-book, Kindle edition
5. Martin C Brown, *The Complete Reference Python,* Brandon A Nordin, First Edition 2001