PD LAB
ASSIGNMENT - 11

Name: Raunak Thanawala
Registration Number: 231070051
Branch: Computer Engineering
Batch: 3

Aim:-

Study Exception Handing in Python

Theory:-

- Exception handling in Python allows programs to deal with unexpected situations or errors gracefully without crashing.
- Python provides built-in support for handling errors through the try, except, else, and finally blocks.
- This mechanism ensures that programs can recover from errors or handle them

appropriately, improving robustness and user experience.

- Key Components:
    - try Block:
        - Contains the code that may raise an exception.
    - except Block:
        - Defines how to handle specific exceptions that occur in the try block.
    - else Block:
        - Contains code that executes if no exceptions are raised in the try block.
    - finally Block:
        - Contains code that will always execute, whether an exception occurred or not (e.g., cleaning up resources).
- Types of Errors:
    - Syntax Errors
        - Description:
            - ➢ These occur when Python's syntax rules are violated, preventing the code from being parsed correctly.
    - Logical Errors
        - Description:

            - ➢ Errors in the program's logic that cause incorrect results but do not raise exceptions.
- ○ Runtime Errors (Exceptions)
    - ■ Description:
        - ➢ Errors that occur during program execution, disrupting the program flow.
    - ■ Common Runtime Errors:
        - ➢ NameError: Raised when a variable or function is not defined.
        - ➢ TypeError: Occurs when an operation is performed on an inappropriate data type.
        - ➢ ValueError: Raised when an argument of the correct type but an invalid value is passed.
        - ➢ IndexError: Occurs when an index is out of range for a list, tuple, or string.
        - ➢ KeyError: Raised when a dictionary key is not found.
        - ➢ ZeroDivisionError: Raised when dividing by zero.

        - ➢ AttributeError: Raised when an attribute reference or assignment fails.
        - ➢ ImportError: Raised when an import statement fails to find the module or function.
        - ➢ ModuleNotFoundError: A subclass of ImportError, raised when the module being imported is not found.
        - ➢ IOError: Raised when an I/O operation fails.
- ○ System-Related Errors
    - ■ SystemError:
        - ➢ Raised when the interpreter detects an internal error but cannot explain it.
- ○ Assertion Errors
    - ■ Description:
        - ➢ Raised when an assert statement fails, indicating a condition that should never occur.
- ○ Environment Errors
    - ■ Description:

        - Errors caused by the environment, such as missing files or insufficient permissions.
    - Common Environment Errors:
        - OSError: Raised for operating system-related issues.
        - PermissionError: A subclass of OSError, raised when permission is denied.
        - TimeoutError: Raised when an operation times out.
- Arithmetic Errors
    - Description:
        - Errors related to invalid numeric operations.
    - Common Arithmetic Errors:
        - ArithmeticError: The base class for arithmetic-related exceptions.
        - ZeroDivisionError: Raised when dividing by zero.
        - FloatingPointError: Raised when a floating-point operation fails.
- Lookup Errors
    - Description:

            - ➢ Errors raised when a lookup fails.
        - Common Lookup Errors:
            - ➢ IndexError: Occurs when an index is out of range.
            - ➢ KeyError: Raised when a dictionary key is not found.
    - Custom Errors
        - In Python, you can define custom error classes to represent specific types of errors in your application.
        - This is useful for improving code clarity and providing more descriptive error messages.
        - A custom error is a subclass of the built-in Exception class.
        - It usually includes:
            - ➢ An initializer (__init__) for setting up custom messages or data.
            - ➢ Optional custom methods for handling specific error logic.

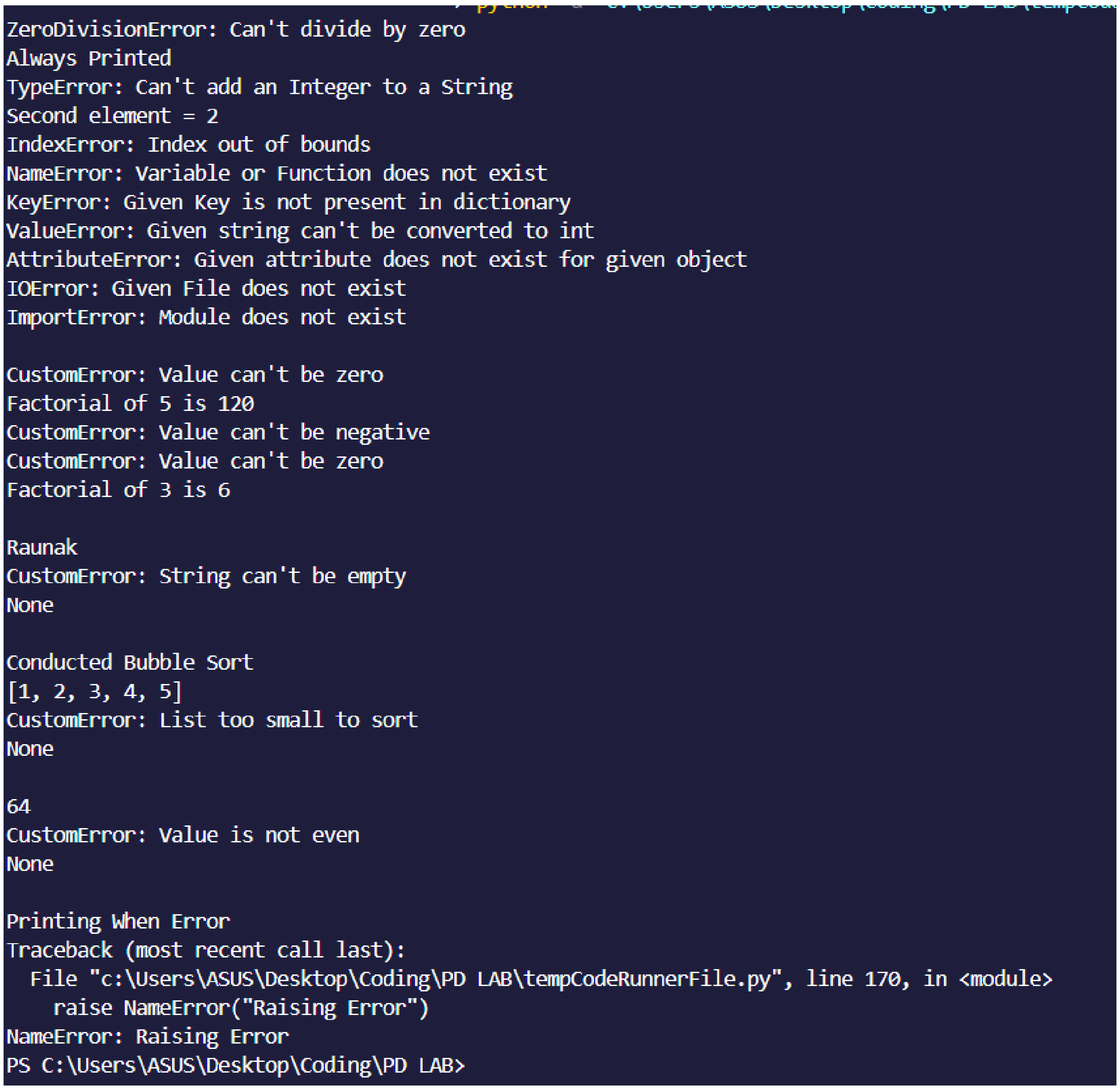
Code:

```
'''
SYNTAX ERROR:
```

```python
amount = 10000
if(amount > 2999)
    print("You are eligible to purchase Dsa Self Paced")

'''

# ZERO DIVISION ERROR
marks = 10000
try:
    a = marks / 0
    print(a)
except ZeroDivisionError:
    print("ZeroDivisionError: Can't divide by zero")
finally:
    print("Always Printed")

# TYPE ERROR
a = 1
b = "Hello"
try:
    print(a + b)
except TypeError:
    print("TypeError: Can't add an Integer to a String")

# INDEX ERROR
a = [1, 2, 3]
try:
    print("Second element = %d" % (a[1]))
    print("Fourth element = %d" % (a[3]))
except IndexError:
    print("IndexError: Index out of bounds")

# NAME ERROR
def fun(a):
    if a < 4:
        b = a / (a - 3)
    print("Value of b = ", b)

class CustomException(Exception):
    def __init__(self, message):
        super().__init__(message)

try:
    fun(5)
except ZeroDivisionError:
    print("ZeroDivisionError: Can't divide by Zero")
except NameError:
    print("NameError: Variable or Function does not exist")
else:
    print("Running Successful")

# KEY ERROR
my_dict = {'name': 'Alice', 'age': 25}
try:
    print(my_dict['address'])
except KeyError:
    print("KeyError: Given Key is not present in dictionary")

# VALUE ERROR
try:
    x = int("abc")
    print(x)
except ValueError:
    print("ValueError: Given string can't be converted to int")

# ATTRIBUTE ERROR
my_string = "hello"
try:
    my_string.append(" world")
except AttributeError:
    print("AttributeError: Given attribute does not exist for given
object")

# IO ERROR
try:
    with open('non_existent_file.txt', 'r') as file:
        content = file.read()
except IOError:
    print("IOError: Given File does not exist")
```

```
# IMPORT ERROR
try:
    import non_existent_module # type: ignore
except ImportError:
    print("ImportError: Module does not exist")

# CUSTOM EXCEPTIONS
class CustomError(Exception):
    def __init__(self, message):
        self.message = message
        super().__init__(message)

    def check_zero(value):
        if value == 0:
            raise CustomError("Value can't be zero")
    def check_neg(value):
        if value<0:
            raise CustomError("Value can't be negative")
    def check_empty(s):
        if s == '':
            raise CustomError("String can't be empty")
    def list_size(num):
        if len(num) < 2:
            raise CustomError("List too small to sort")
    def check_even(value):
        if value % 2 != 0:
            raise CustomError("Value is not even")

def factorial(n):
    try:
        CustomError.check_zero(n)
    except CustomError as e:
        print(f"CustomError: {e}")
        return 1
    try:
        CustomError.check_neg(n)
    except CustomError as e:
        print(f"CustomError: {e}")
        return factorial(-n)
    return n*factorial(n-1)

print()
print("Factorial of 5 is", factorial(5))
print("Factorial of 3 is", factorial(-3))

def ValidateName(s):
    try:
        CustomError.check_empty(s)
        return s
    except CustomError as e:
        print(f"CustomError: {e}")

print()
print(ValidateName("Raunak"))
print(ValidateName(""))

def bubble_sort(num):
    try:
        CustomError.list_size(num)
        n = len(num)
        for i in range(n - 1):
            for j in range(n - i - 1):
                if num[j] > num[j + 1]:
                    num[j], num[j + 1] = num[j + 1], num[j]
        print("Conducted Bubble Sort")
        return num
    except CustomError as e:
        print(f"CustomError: {e}")

print()
nums = [5,3,2,4,1]
print(bubble_sort(nums))
small = [1]
print(bubble_sort(small))

def square_even(n):
    try:
        CustomError.check_even(n)
        return n**2
    except CustomError as e:
        print(f"CustomError: {e}")
print()
print(square_even(8))
print(square_even(9))
# RAISING ERROR
print()
try:
    raise NameError("Raising Error")
except NameError:
    print("Printing When Error")
    raise
```

OUTPUT:



```
ZeroDivisionError: Can't divide by zero
Always Printed
TypeError: Can't add an Integer to a String
Second element = 2
IndexError: Index out of bounds
NameError: Variable or Function does not exist
KeyError: Given Key is not present in dictionary
ValueError: Given string can't be converted to int
AttributeError: Given attribute does not exist for given object
IOError: Given File does not exist
ImportError: Module does not exist

CustomError: Value can't be zero
Factorial of 5 is 120
CustomError: Value can't be negative
CustomError: Value can't be zero
Factorial of 3 is 6

Raunak
CustomError: String can't be empty
None

Conducted Bubble Sort
[1, 2, 3, 4, 5]
CustomError: List too small to sort
None

64
CustomError: Value is not even
None

Printing When Error
Traceback (most recent call last):
  File "c:\Users\ASUS\Desktop\Coding\PD LAB\tempCodeRunnerFile.py", line 170, in <module>
    raise NameError("Raising Error")
NameError: Raising Error
PS C:\Users\ASUS\Desktop\Coding\PD LAB>
```

<u>CONCLUSION:</u>

Thus we have written a program that shows exception handling in python where we see all the different types of errors encountered while coding in python.
We have also implemented Custom Errors using pythons special class which allows to do just that.