Course Name: Fundamentals of Programming & Data Science	**Course Code:** CMPE-112L
Assignment Type: Lab	**Dated:** 18th March 2024
Semester: 2nd	**Session:** 2023
Lab/Project/Assignment #: 7	**CLOs to be covered: CLO 4**
Lab Title: Exception handling	**Teacher Name:** Engr. Afeef Obaid

Lab Evaluation:

CLO 4	Display well-commented code					
Levels (Marks)	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**	**Level 6**
(10)						
					Total	**/10**

Rubrics for Current Lab Evaluation

Scale	**Marks**	**Level**	**Rubric**
Excellent	**10**	L1	Submitted all lab tasks, BONUS task, have good understanding.
Very Good	**8**	L2	Submitted the lab tasks but have good understanding
Good	**6**	L3	Submitted the lab tasks but have weak understanding.
Basic	**4**	L4	Submitted the lab tasks but have no understanding.
Barely Acceptable	**2**	L5	Submitted only one lab task.
Not Acceptable	**0**	L6	Did not attempt

LAB No. 7

Lab Goals/Objectives:

By reading this manual, students will be able:

- To understand the concept of Exception handling in Python
- To Become familiar with the concept of Try-Except in Python
- To learn about Finally Keyword in Python
- To learn about the built in error modules in Python

Equipment Required: Computer system with Pycharm IDE and python package installed on it

Exception Handling

- Exception handling is the process of responding to unwanted or unexpected events when a computer program runs. Exception handling deals with these events to avoid the program or system crashing, and without this process, exceptions would disrupt the normal operation of a program.

Exception Handling in Python

- Python has many built-in exceptions that are raised when your program encounters an error (something in the program goes wrong).

- When these exceptions occur, the Python interpreter stops the current process and passes it to the calling process until it is handled. If not handled, the program will crash.

Example

```
x=input("Enter Any Number ")
if int(x)%2==0:
   print(f"{x} is Even")
else:
   print(f"{x} is Odd")
```

In the above example if you entered any integer value then you will get no error but if you entered any value other than integer like float or string you will get a **ValueError** and your Python interpreter stops the current process and the program will crash.

Python Try Except

try except blocks are used in python to handle errors and exceptions. The code in try block runs when there is no error. If the try block catches the error, then the except block is executed.

Syntax:

```
try:
    #statements which could generate exception
except:
    #Solution of generated exception
```

Example

```
x=input("Enter Any Number ")
try:
   if int(x)%2==0:
      print(f"{x} is Even")
   else:
      print(f"{x} is Odd")
except:
   print("Some error occured")
```

Python Try Except Else

You can use the else keyword to define a block of code to be executed if no errors were raised.

Syntax:

```
try:
    #statements which could generate exception
except:
    #Solution of generated exception
else:
    # statements when no error occurs
```

Example

```
x=input("Enter Any Number ")
try:
   if int(x)%2==0:
      print(f"{x} is Even")
   else:
      print(f"{x} is Odd")
except:
   print("Some error occured")
else:
   print("Your code has no error")
```

Python Built in Exception Handling Modules

In Python, there are several built-in error handling modules that you can use to handle exceptions gracefully. Here are some of the commonly used modules:

- **Exception**
  This is the base class for all built-in exceptions. You can use it to catch any type of exception that might occur in your program.
- **AssertionError**
  This exception is raised when an assert statement fails.
- **AttributeError**
  This exception is raised when an object does not have a requested attribute.
- **EOFError**
  This exception is raised when there is no more input from the input() function.

- **FloatingPointError**
  This exception is raised when a floating point operation fails.

- **ImportError**
  This exception is raised when a module cannot be imported.

- **IndexError**
  This exception is raised when an index is out of range.

- **KeyError**
  This exception is raised when a key is not found in a dictionary.

- **KeyboardInterrupt**
  This exception is raised when the user interrupts the program with a keyboard interrupt by pressing Ctrl + F2

- **MemoryError**
  This exception is raised when the program runs out of memory.

- **NameError**
  This exception is raised when a name is not found.

- **OSError**
  This exception is raised when a system call fails.

- **OverflowError**
  This exception is raised when the result of an arithmetic operation is too large to be represented.

- **SyntaxError**
  This exception is raised when there is a syntax error in the code.

- **TypeError**
  This exception is raised when an operation or function is applied to an object of inappropriate type.

- **ValueError**
  This exception is raised when a function receives an argument of correct type but inappropriate value.

- **ZeroDivisionError**
  This exception is raised when division or modulo by zero is encountered.

By handling these exceptions gracefully in your program, you can make your code more robust and prevent it from crashing.

You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error:

Example

```
input("Enter Any Number ")
try:
   if int(x)%2==0:
      print(f"{x} is Even")
   else:
      print(f"{x} is Odd")
   10/0
except NameError as e:
   print(e)
except ValueError as i:
   print(i)
except Exception as l:
   print(l)
else:
   print("Your code has no error")
```

Finally Clause

The finally code block is also a part of exception handling. When we handle exception using the try and except block, we can include a finally block at the end. The finally block is always executed, so it is generally used for doing the concluding tasks like closing file resources or closing database connection or may be ending the program execution with a delightful message.

Syntax:

```
try:
  #statements which could generate exception
except:
  #solution of generated exception
finally:
   #block of code which is going to execute in any situation
```

Example

```
try:
   x=int(input("Enter any number "))
   for i in range (1,11):
      print(f"{x} x {i} = {x*i}")
except Exception as e:
   print(e)
finally:
   print("I will execute always whether there is an error or not.")
```

Point to Ponder: Statements in the finally clause will still execute whether finally clause used or not, you can see it in below example, so why we are using this finally clause?

Example

```
try:
   x=int(input("Enter any number "))
   for i in range (1,11):
      print(f"{x} x {i} = {x*i}")
except Exception as e:
   print(e)
print("I will execute whether their is error or not.")
```

One of the important use cases of finally clause is in a function which returns a value.

Example

```
def table():
   try:
      x=int(input("Enter any number "))
      for i in range (1,11):
         print(f"{x} x {i} = {x*i}")
      return 1
   except Exception as e:
      print(e)
      return 0
   finally:
      print("I will execute whether their is error or not.")
table()
```

Raise an exception

As a Python developer you can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, use the raise keyword.

Example

```
import math
x=int(input("Enter any number "))
if type(x) != int:
   raise TypeError("Enter Integer")
elif int(x)<0:
   raise Exception("Enter positive number")
else:
   print(f"Sqaure root of {x} = {math.sqrt(int(x))}")
```

Example

```
a=input("Enter password ")
if len(a)<4:
   raise Exception(" Length of password must be greater than 4, Try new password")
else:
   print("Password created")
```

Lab Tasks

- Write the python program to calculate the factorial of a number and do exception handling if user enters any negative number or a string using assert.
- Write the python program that take the 2 x 2 matrix from the user and calculate its inverse and handle the possible errors occurs.
- Write a UDF that takes a decimal number and return its hexadecimal equivalent and handle the TypeError if user pass any string in UDF and then display a message that “UDF call successfully”.
- Write a python program that takes user bio data including (Name, Address, Contact no., age and gender) and raise the exceptions as following.
    - Raise exception if name has some digit in it
    - Raise exception if length of address is than 3 characters
    - Raise exception if contact no. contains any alphabet
    - Raise exception if age is less than 0 and greater than 150 years
    - Raise exception if gender is other than male or female