Student Name: _____ **Roll No:** _____ **Section:** _____

Lab No. 13

Lab 13- Exception Handling

Objectives:

- Introduction Exception Handling
- Need of Exception Handling
- What is Exception Handling
- Exception Handling in Python

1. Introduction to Exception Handling

- Exception is an indication that something went wrong in the program execution and it can be recovered
- Error is a condition when something serious went wrong and it can't be recovered Regular expressions use two types of characters

- Many times though, a program results in an error after it is run even if it doesn't have any syntax error. Such an error is a runtime error, called an exception. A number of built-in exceptions are defined in the Python library.

- It may cause damage to system resources, such as files. Hence, the exceptions should be properly handled so that an abrupt termination of the program is prevented.

The most common Exception are:

- ZeroDivisionError
- ValueError
- TypeError

Python uses try and except keywords to handle exceptions. Both keywords are followed by indented blocks.

Syntax

```
try :
    #statements in try block
except :
    #executed when error in try block
```

Student Name: _____ **Roll No:** _____ **Section:** _____

2. Try and catch block

The try: block contains one or more statements which are likely to encounter an exception. If the statements in this block are executed without an exception, the subsequent except: block is skipped.

If the exception does occur, the program flow is transferred to the except: block. The statements in the except: block are meant to handle the cause of the exception appropriately. For example, returning an appropriate error message

Exercise 1:

```
try:
    a=10
    b=5
    print(a/b)
except:
    print('Some error occurred.')
print("Out of try except blocks.")
```

Output:

```
try:
    a=10
    b='0'
    print(a/b)
except:
    print('Some error occurred.')
print("Out of try except blocks.")
```

Output:

Student Name: _____ **Roll No:** _____ **Section:** _____

Exercise 2:

```
try:
    a=5
    b='0'
    print (a+b)
except TypeError:
    print('Unsupported operation')
print ("Out of try except blocks")
```

Output:

A single try block may have multiple except blocks. The following example uses two except blocks to process two different exception types:

Exercise 3:

```
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')
```

Output:

Student Name: _____ **Roll No:** _____ **Section:** _____

3. Else and finally

In Python, keywords else and finally can also be used along with the try and except clauses. While the except block is executed if the exception occurs inside the try block, the else block gets processed if the try block is found to be exception free.

Syntax:

```
try:
    #statements in try block
except:
    #executed when error in try block
else:
    #executed if try block is error-free
finally:
    #executed irrespective of exception occurred or not
```

The finally block consists of statements which should be processed regardless of an exception occurring in the try block or not. As a consequence, the error-free try block skips the except clause and enters the finally block before going on to execute the rest of the code. If, however, there's an exception in the try block, the appropriate except block will be processed, and the statements in the finally block will be processed before proceeding to the rest of the code.

Exercise 4:

```
try:

    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)
```

Student Name: _____ Roll No: _____ Section: _____

```
finally:
    print("finally block")

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

Output:

Raise an Exception

Python also provides the raise keyword to be used in the context of exception handling. It causes an exception to be generated explicitly. Built-in errors are raised implicitly. However, a built-in or custom exception can be forced during execution.

Exercise 5:

```
try:
    x=int(input('Enter a number upto 100: '))
    if x > 100:
        raise ValueError(x)
except ValueError:
    print(x, "is out of allowed range")
else:
    print(x, "is within the allowed range")
```

Output:

Student Name: _____ **Roll No:** _____ **Section:** _____

:

Programming Exercise (Python)

Task1:

Write a program that take a user input to catch an exception of **ZeroDivisionError** and **Value Error**

Task2:

Write a program that take a user input to catch an exception of **TypeError**