LTCE

Experiment No. 3

Aim -

Write a program to demonstrate exception handling

1) To demonstrate exception handling by try, except & finally

Program -

```
def divide (x,y):
    try:
        result = x // y
    except ZeroDivisionError:
        print ("Sorry ! You are dividing by zero")
    else:
        print ("Yeah ! Your answer is:", result)
    Finally:
        print ("This is always executed")
divide (3,2)
divide (6,0)
```

Output ⇒ Yeah ! Your answer is : 1
This is always executed
Sorry ! You are dividing by zero
This is always executed.

2) To demonstrate exception handling for multiple blocks :

Program : For single block

```
try:
    print ("Hello World!")
```

```
except Exception as e:
    print("An exception occured:", str(e))
```

Output ⇒ Hello World!

Program: For multiple Block

```
try:
    a = 10/0
except ZeroDivisionError as e:
    print("Error", str(e))
finally:
    print("Executing finally block for Block 1\n")
try:
    my_list = [1, 2, 3]
    print(my_list[3])
except IndexError as e:
    print("Error", str(e))
finally:
    print("Executing finally block for Block 2\n")
try:
    my_dict = {"a": 1, "b": 2}
    print(my_dict["c"])
except KeyError as e:
    print("Error:", str(e))
finally:
    print("Executing finally block for Block 3\n")
```

Output ⇒ Error : division by zero
Executing finally block for Block 1

Error : list index out of range
Executing finally block for Block 2

Error : ' '
Executing finally block for Block 3

3) Write a python program for factorial of negative no. using exception handling.

Program -

```
class NegativeFactorialError(Exception):
    pass
def factorial(n):
    if n<0:
        raise NegativeFactorialError("Factorial is not defined for negative no.")
    else:
        result = 1
        for i in range(1, n+1):
            result *= i
        return result
try:
    n = int(input("enter a no:"))
    print(factorial(n))
except NegativeFactorialError as e:
    print(e)
```

Output ⇒ 1) Enter a no.: 5
120
2) Enter a no.: -5
Factorial is not defined for negative no.

4) WAP to demonstrate single inheritance in Python

It enables a derived class to inherit prop from single parent class, thus enabling code reusability & the addition of new features to existing code.

Program -

```
class Parent:
    def func1(self):
        print("This funct is in parent class.")
class child(Parent):
    def func2(self):
        print("This function is in child class.")
object = child()
object.func1()
object.func2()
```

Output:

This function is in parent class.
This function is in child class.

5) WAP to demonstrate multiple inheritance in Python

When a class can be derived from more than one base class this type is called multiple inheritance.

Program -

```
class Mother:
    mothername = ""
    def mother(self):
        print(self.mothername)

class Father:
    fathername = ""
    def father(self):
        print(self.fathername)

class Son(Mother, Father)
    def parents(self):
        print("Father:", self.fathername)
        print("Mother:", self.mothername)

S1 = Son()
S1.fathername = "RAM"
S1.Mothername = "SITA"
S1.parents()
```

Output ⇒ Father : RAM
Mother : SITA

6) WAP to demonstrate multilevel inheritance

In multilevel inheritance, features of the base class & derived class are further inherited into the new derived class.

Program –

```
class Grandfather:
    def __init__(self, grandfathername):
        self.grandfathername = grandfathername

class Father(Grandfather):
    def __init__(self, fathername, grandfathername):
        self.fathername = fathername
        Grandfather.__init__(self, grandfathername)

class Son(Father):
    def __init__(self, sonname, fathername, grandfathername):
        self.sonname = sonname
        Father.__init__(self, fathername, grandfathername)

    def print_name(self):
        print("Grandfathe name:", self.grandfathername)
        print("Father name:", self.fathername)
        print("Son name:", self.sonname)

s1 = Son('Prince', 'Rampal', 'Lal mani')
print(s1.grandfathername)
s1.print_name()
```

Output ⇒ Lal mani

Grandfather name: Lal mani

Father name: Rampal

Son name: Prince

7) WAP to demonstrate Hierarchical inheritance

When more than one derived class created from single base this type of inheritance called as hierarchial.

Program -

```
class Parent:
    def func1(self):
        print("This funct is in parent class.")

class Child1(Parent):
    def func2(self):
        print("This funct is in child 1.")

class Child2(Parent):
    def func3(self):
        print("This funct is in child 2")

Obj1 = Child1()
Obj2 = Child2()
Obj1.func1()
Obj1.func2()
Obj2.func1()
Obj2.func2()
```

Output ⇒

```
This funct is in parent class.
This funct is in child 1.
This funct is in parent class.
This funct is in child 2.
```

LTCE

8) WAP to demonstrate Hybrid inheritance

Inheritance consisting of multiple types of inheritance is called hybrid inheritance.

Program -

```
class school:
    def func1(self):
        print("This funct is in school.")

class student1(school):
    def func2(self):
        print("This funct is in student1.")

class student2(school):
    def func3(self):
        print("This funct is in studen2.")

class student3(student1, school):
    def func4(self):
        print("This funct is in student3.")

o = student3()
o.func1()
o.func2()
```

Output ⇒ This funct is in school.
This funct is in student1.

Conclusion - succesfully implemeted inheritance of various types of inheritances.