Batch: G3	**Roll No.:**	**16010421063**
Experiment / assignment / tutorial No. 9		
Grade: AA / AB / BB / BC / CC / CD /DD		
Signature of the Staff In-charge with date		

TITLE: Exception Handling

AIM: Write a python program to implement exception handling concept for registration portal.

Expected OUTCOME of Experiment:
CO2: Use different Decision Making statements and Functions in Python.
CO3: Apply Object oriented programming concepts in Python

Resource Needed: Python IDE

Theory:

Exceptions
Even if a statement or expression is syntactically correct, there might arise an error during its execution. For example, trying to open a file that does not exist, division by zero and so on. Such types of errors might disrupt the normal execution of the program and are called exceptions. An exception is a Python object that represents an error. When an error occurs during the execution of a program, an exception is said to have been raised. Such an exception needs to be handled by the programmer so that the program does not terminate abnormally. Therefore, while designing a program, a programmer may anticipate such erroneous situations that may arise during its execution and can address them by including appropriate code to handle that exception.

Built-in Exceptions
Commonly occurring exceptions are usually defined in the compiler/interpreter. These are called built-in exceptions. Python's standard library is an extensive collection of built-in exceptions that deals with the commonly occurring errors (exceptions) by providing the standardized solutions for such errors. On the occurrence of any built-in exception, the appropriate exception handler code is executed which displays the reason along with the raised exception name. The programmer then has to take appropriate action to handle it. Some of the commonly occurring built-in exceptions that can be raised in Python are explained below

Following are list of common exception class found in Python:

Class	Description
Exception	A base class for most error types
AttributeError	Raised by syntax obj.foo, if obj has no member named foo
EOFError	Raised if "end of file" reached for console or file input
IOError	Raised upon failure of I/O operation (e.g., opening file)
IndexError	Raised if index to sequence is out of bounds
KeyError	Raised if nonexistent key requested for set or dictionary
KeyboardInterrupt	Raised if user types ctrl-C while program is executing
NameError	Raised if nonexistent identifier used
StopIteration	Raised by next(iterator) if no element; see Section 1.8
TypeError	Raised when wrong type of parameter is sent to a function
ValueError	Raised when parameter has invalid value (e.g., sqrt(−5))
ZeroDivisionError	Raised when any division operator used with 0 as divisor

Exception Handling in Python:

In python, if an exception occurs which is not handled then interpreter will report the error and terminate your application. To prevent this, handle a possible exception by enclosing your statements inside a try-except statement.

This try-except statement works as follows. First, Python executes the statements between try. These statements are called the try clause.

If no exception occurs, the statements below except are skipped and the execution of the try statement is finished. The statements below except are called the except clause.

But, if an exception occurs during the execution of the try clause, the rest of the statements are skipped. Then if the type matches the exception named after the except statement, this except clause is executed. The execution continues after the try statement

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.

Except Block:

The code that follows the except statement is the program's response to any exceptions in the preceding try clause.

It is used to catch and handle the exception(s) that are encountered in the try clause.

Else Block:

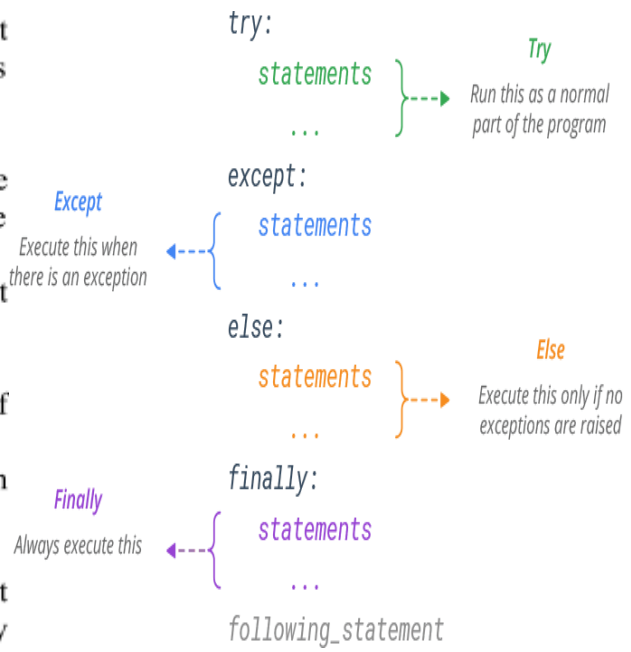
Instruct a program to execute a certain block of code only in the absence of exceptions.

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

Program:



```
a = [1, 2, 3]
try:
    print "Second element = %d" %(a[1])

    # Throws error since there are only 3 elements in array
    print "Fourth element = %d" %(a[3])

except IndexError:
    print "An error occurred"
```

Output:-

```
Second element = 2
An error occurred
```

Problem Definition:

Registration portal have a requirement where we can only register the students when their age is less than 12 and weight is less than 40, if any of the condition is not met then the user should get an appropriate exception with the warning message "Student is not eligible for registration". Implement above registration portal requirement using exception handling concept.

Books/ Journals/ Websites referred:

1. **Reema Thareja , "Python Programming: Using Problem Solving Approach", Oxford University Press, First Edition 2017, India**
2. **Sheetal Taneja and Naveen Kumar," Python Programing: A Modular Approach", Pearson India, Second Edition 2018, India**

Implementation details:

```
# The above code is asking for the name, age and weight of the
student. If the student is above 12
# years of age and weighs more than 40 kgs, then the student is not
eligible for registration.
try:
    print("Registration Portal")
    name=input("Enter your name: ")
    age=int(input("Enter your age: "))
    weight=float(input("Enter your weight: "))
    if age>12 or weight>40:
```

```
        raise Exception("Student is not eligible for registration")
except Exception as e:
    print(e)
else:
    print("Thank you for registering")
```

Output(s):

```
PS C:\Users\ARYA\Downloads\testing> python -u "c:\Users\ARYA\Downloads\testing\test.py"
Registration Portal
Enter your name: Ayra
Enter your age: 10
Enter your weight: 30
Thank you for registering
PS C:\Users\ARYA\Downloads\testing>
```

```
PS C:\Users\ARYA\Downloads\testing> python -u "c:\Users\ARYA\Downl
Registration Portal
Enter your name: Arya
Enter your age: 13
Enter your weight: 30
Student is not eligible for registration
```

```
PS C:\Users\ARYA\Downloads\testing> python -u "c:
Registration Portal
Enter your name: Arya
Enter your age: 10
Enter your weight: 50
Student is not eligible for registration
PS C:\Users\ARYA\Downloads\testing> python -u "c:
```

Conclusion:

Successfully understood the concept of error handling

Post Lab Descriptive Questions:

1) Predict the output when you run the following code:

```
try:
result = 15/0
except ArithmeticError:
print("Caught the ArithmeticError.Your divisor is zero.")
except ZeroDivisionError:
print("Caught ZeroDivisionError.Correct the divisor which is zero at present.")
```

OUTPUT- Caught the ArithmeticError.Your divisor is zero.

2) Predict the output for given code:

```
my_string="Hello"
assert my_string== "Hi", "You should use 'Hi'"
```

OUTPUT- AssertionError: You should use 'Hi'

3) Explain concept of raise exception with suitable example

We add a condition into the inbuilt set of exception conditions. There is a keyword in Python called "raise" to raise our exceptions. We can demand an exception to occur in a specific situation using this keyword.

```
try:
    if input()=="":
        raise Exception("Empty input")
except Exception as e:
    print(e)
```

Date: _____ **Signature of faculty in-charge**