Experiment 07

Write python programs to understand

7.1) Creating User-defined modules/packages and import them in a program and understand attributes related to modules.

7.2) Creating a menu driven application which should cover a few built-in exceptions in python, also demonstrate the concept of assertion and user defined exception handling.

Roll No.	01
Name	Aamir Ansari
Class	D10-A
Subject	Python Lab
LO Mapped	LO1: Understand the structure, syntax, and semantics of the Python language LO4:Create Python applications using modules, packages, multithreading and exception handling.

Aim:

Understanding user-defined modules/packages and exception handling

Introduction:

User-defined modules:

Python files which are imported into the top-level file, or among each other, and provide separate functionalities. These files are usually not launched directly from your command prompt, and are custom-made for the purpose of the project

Modules refer to a file containing Python statements and definitions. A file containing Python code, for example: example.py, is called a module, and its module name would be an example. We use modules to break down large programs into small manageable and organized files. Furthermore, modules provide reusability of code.
We can define our most used functions in a module and import it, instead of copying their definitions into different programs

Eg.

```
# file name is operations.py
def add(a, b):
    return a + b
```

Here, we have defined a function add() inside a module named operations. The function takes in two numbers and returns their sum

To import the definitions inside a module to another module or the interactive interpreter in Python, we use the import keyword to do this. To import our previously defined module operations, we type the following in the Python prompt.

```
>>> import operations
```

This does not import the names of the functions defined in operations directly in the current symbol table. It only imports the module name operations there.
Using the module name we can access the function using the dot `.` operator. For example:

```
>>> operations.add(10,9.5)
19.5
```

User-defined Packages

To organize a large number of files in different folders and subfolders based on some criteria, so that we can find and manage them easily. In the same way, a package in Python takes the concept of the modular approach to the next logical level. As you know, a module can contain multiple objects, such as classes, functions, etc. A package can contain one or more relevant modules. Physically, a package is actually a folder containing one or more module files

To create a package named mypackage, using the following steps:

1. Create a new folder named D:\MyApp.
2. Inside MyApp, create a subfolder with the name 'mypackage'.
3. Create an empty __init__.py file in the mypackage folder.
4. Using a Python-aware editor like IDLE, create modules greet.py and functions.py with the following code:

greet.py

```
def SayHello(name):
    print("Hello ", name)
```

functions.py

```
def sum(x,y):
    return x+y

def average(x,y):
    return (x+y)/2

def power(x,y):
    return x**y
```

A package called “mypackage” is created

To test our package, navigate the command prompt to the MyApp folder and invoke the Python prompt from there

```
D:\MyApp>python
```

Import the functions module from the mypackage package and call its power() function.

```
>>> from mypackage import functions
>>> functions.power(3,2)
9
```

It is also possible to import specific functions from a module in the package.

```
>>> from mypackage.functions import sum
>>> sum(10,20)
30
>>> average(10,12)
Traceback (most recent call last):
File "<pyshell#13>", line 1, in <module>
NameError: name 'average' is not defined
```

__init__.py

The package folder contains a special file called __init__.py, which stores the package's content. It serves two purposes:

The Python interpreter recognizes a folder as the package if it contains __init__.py file.
__init__.py exposes specified resources from its modules to be imported.
An empty __init__.py file makes all functions from the above modules available when this package is imported. Note that __init__.py is essential for the folder to be recognized by Python as a package. You can optionally define functions from individual modules to be made available.

test.py

```
from mypackage import power, average, SayHello
SayHello()
x=power(3,2)
print("power(3,2) : ", x)
```

Note that functions power() and SayHello() are imported from the package and not from their respective modules, as done earlier. The output of the above script is:

```
D:\MyApp>python test.py
Hello world
power(3,3) : 27
```

Built-in Exceptions in Python

1) Exception
   Base class for all exceptions

2) StopIteration
   Raised when the next() method of an iterator does not point to any object.

3) SystemExit
   Raised by the sys.exit() function.

4) StandardError
   Base class for all built-in exceptions except StopIteration and SystemExit.

5) ArithmeticError
   Base class for all errors that occur for numeric calculation.

6) OverflowError
   Raised when a calculation exceeds maximum limit for a numeric type.

7) FloatingPointError
   Raised when a floating point calculation fails.

8) ZeroDivisionError
   Raised when division or modulo by zero takes place for all numeric types.

9) AssertionError
   Raised in case of failure of the Assert statement.

10) AttributeError
    Raised in case of failure of attribute reference or assignment.

11) EOFError
    Raised when there is no input from either the raw_input() or input() function and the end of file is reached.

12) ImportError
    Raised when an import statement fails.

13) KeyboardInterrupt
    Raised when the user interrupts program execution, usually by pressing Ctrl+c.

14) LookupError
    Base class for all lookup errors.

15) IndexError
    Raised when an index is not found in a sequence.

16) KeyError
Raised when the specified key is not found in the dictionary.

17) NameError
Raised when an identifier is not found in the local or global namespace.

18) UnboundLocalError
Raised when trying to access a local variable in a function or method but no value has been assigned to it.

19) EnvironmentError
Base class for all exceptions that occur outside the Python environment.

20) IOError
Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.

21) OSError
Raised for operating system-related errors.

22) SyntaxError
Raised when there is an error in Python syntax.

23) IndentationError
Raised when indentation is not specified properly.

24) SystemError
Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exist.

25) SystemExit
Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.

26) TypeError
Raised when an operation or function is attempted that is invalid for the specified data type.

27) ValueError
Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.

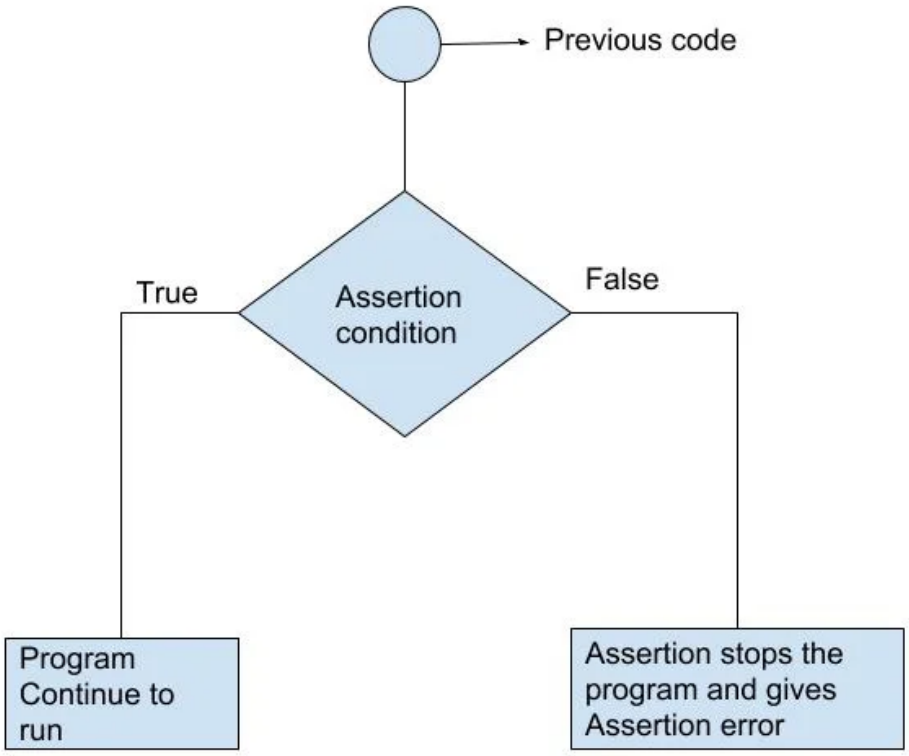
28) RuntimeError
Raised when a generated error does not fall into any category.

29) NotImplementedError
Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.

Assertion

1. Assertions are statements that assert or state a fact confidently in your program. For example, while writing a division function, you're confident the divisor shouldn't be zero, you assert the divisor is not equal to zero.
2. Assertions are simply boolean expressions that check if the conditions return true or not. If it is true, the program does nothing and moves to the next line of code. However, if it's false, the program stops and throws an error.
3. It is also a debugging tool as it halts the program as soon as an error occurs and displays it.

We can be clear by looking at the flowchart below:



Syntax for using Assert in Pyhton:

assert <condition>

assert <condition>,<error message>

In Python we can use assert statements in two ways as mentioned above.

1. `assert` statement has a condition and if the condition is not satisfied the program will stop and give AssertionError.
2. `assert` statement can also have a condition and an optional error message. If the condition is not satisfied, assert stops the program and gives AssertionError along with the error message.

Eg. calculate the average of the values passed by the user and the value should not be an empty list. We will use an assert statement to check the parameter and if the length of the passed list is zero, the program halts.

```
def avg(marks):
    assert len(marks) != 0
    return sum(marks)/len(marks)

mark1 = []
print("Average of mark1:",avg(mark1))
```

Output:

AssertionsError

Another example which satisfies the assert condition

```
def avg(marks):
    assert len(marks) != 0,"List is empty."
    return sum(marks)/len(marks)

mark2 = [55,88,78,90,79]
print("Average of mark2:",avg(mark2))

mark1 = []
print("Average of mark1:",avg(mark1))
```

Output:

Average of mark2: 78.0

AssertionError: List is empty.

1. Assertions are the condition or boolean expression which are always supposed to be true in the code.
2. `assert` statement takes an expression and optional message.
3. `assert` statement is used to check types, values of argument and the output of the function.
4. `assert` statement is used as a debugging tool as it halts the program at the point where an error occurs.

Custom Exceptions

In Python, users can define custom exceptions by creating a new class. This exception class has to be derived, either directly or indirectly, from the built-in `Exception` class. Most of the built-in exceptions are also derived from this class.

```
>>> class CustomError(Exception):
...     pass
...

>>> raise CustomError
Traceback (most recent call last):
...
__main__.CustomError

>>> raise CustomError("An error occurred")
Traceback (most recent call last):
...
__main__.CustomError: An error occurred
```

Here, we have created a user-defined exception called `CustomError` which inherits from the `Exception` class.
This new exception, like other exceptions, can be raised using the raise statement with an optional error message.

While developing a large Python program, it is a good practice to place all the user-defined exceptions that our program raises in a separate file.
Many standard modules do this. They define their exceptions separately as exceptions.py or errors.py

User-defined exception class can implement everything a normal class can do, but we generally make them simple and concise. Most implementations declare a custom base class and derive others exception classes from this base class. This concept is made clearer in the following example.

Results:

```
# module with name `christmas`
# module with 3 methods

# first helper method
def triangleShape(n):
    for i in range(n):
        for j in range(n-i):
            print(' ', end=' ')
        for k in range(2*i+1):
            print('*',end=' ')
        print()

# second helper method
def poleShape(n):
    for i in range(n):
        for j in range(n-1):
            print(' ', end=' ')
        print(' * * ')

# driver method
def tree(row):
    triangleShape(row)
    triangleShape(row)
    poleShape(row)
```

***

```
# file name `runModule.py`

# importing the module christmas
import christmas

# calling the method of `christmas module`
christmas.tree(6)
```

Output:

```
E:\Practice\python\Try_Packages>python runModule.py
            *
          * * *
        * * * * *
      * * * * * * *
    * * * * * * * * *
  * * * * * * * * * * *
            *
          * * *
        * * * * *
      * * * * * * *
    * * * * * * * * *
  * * * * * * * * * * *
           * *
           * *
           * *
           * *
           * *
           * *
```

```
# contents under `mypackage` folder

# file name, __init__.py
from .greet import sayHello

# file name, greet.py
# method, part of `greet` module of `mypackage` package
# method to greet
def sayHello():
    print("My purpose is just to greet you :)")

# file name, operations.py
# methods of `operations` module of `mypackage` package
# method to add two numbers
def sum(n1, n2):
    print("Sum is",n1+n2)

# method to multiply two numbers
def product(n1, n2):
    print("Product is",n1*n2)

# method to divide two numbers
```

```
def division(n1, n2):
    print("Division is",n1/n2)

# driver file outside outside `mypackage` but in same directory
# file name, runPackage

# import from packages
from mypackage import sayHello
from mypackage import operations

# from greet module
print()
sayHello()

# form operations module
print()
operations.sum(12.3, 42)
operations.product(2, 5)
operations.division(12, 3)
```

```
E:\Practice\python\Try_Packages>python runPackage.py

My purpose is just to greet you :)

Sum is 54.3
Product is 10
Division is 4.0
```

```
# AssertionError
def assertionError(n1, n2):
    try:
        assert n2 != 0, "Invalid Operation"
        print(n1 / n2)

    # the errror_message provided by the user gets printed
    except AssertionError as msg:
        print(msg)

# ImportError
def importError():
```

```
    try:
        from mypackage import greet
    except Exception as e:
        print(e)

# IndexError
def indexError(index):
    try:
        arr = [5, 10, 15, 20, 25]
        print("Size of array is 5")
        print(arr[index])
    except IndexError as e:
        print(e)

# KeyError
def keyError(key):
    try:
        dict = {"one": "Lorem", "two": "Ipsum", "three": "Dolor", "four": "Sit"}
        print("Dictionary have 4 keys (from one to four)")
        print(dict[key])
    except KeyError as e:
        print("Key not found",e)

# NameError
def nameError():
    try:
        knownVariable = 20
        print(knownVariable)
        print(unknownVariable)
    except NameError as e:
        print(e)

# OverflowError
def overflowError():
    i=1
    try:
        f = 3.0**i
        for i in range(100):
            print(i, f)
            f = f ** 2
```

```
    except OverflowError as err:
        print('Overflowed after ', f, err)

# SyntaxError
def syntaxError():
    try:
        print('Valid syntax')
        print(eval('**Not Valid Syntax**'))

    except SyntaxError as err:
        print (err)

# ValueError
def valueError():
    try:
        print (float('Aamir'))
    except ValueError as e:
        print (e)

# ZeroDivisionError
def zeroDivisionError(n):
    try:
        print(20/n)

    except ZeroDivisionError as msg:
        print(msg)

choice = 0
while (True):
    print("""
    1. AssertionError       6. OverflowError
    2. ImportError          7. SyntaxError
    3. IndexError           8. ValueError
    4. KeyError             9. ZeroDivisionError
    5. NameError            10. EXIT
    """)
    choice = int(input("Enter your choice :  "))

    if (choice == 1):
        print("Enter two numbers to divide")
```

```
        n1 = int(input("Enter number 1 :  "))
        n2 = int(input("Enter number 2 :  "))
        assertionError(n1, n2)

    elif (choice == 2):
        importError()

    elif (choice == 3):
        index = int(input("Enter an index (between 0 to 4) :  "))
        indexError(index)

    elif (choice == 4):
        key = input("Enter a key (between `one` to `four`) :  ")
        indexError(key)

    elif (choice == 5):
        nameError()

    elif (choice == 6):
        overflowError()

    elif (choice == 7):
        syntaxError()

    elif (choice == 8):
        valueError()

    elif (choice == 9):
        n = int(input("Enter a number to divide from :  "))
        zeroDivisionError(n)

    elif (choice == 10):
        print("*** E X I T I N G ***")
        break;
    else:
        print("Invalid choice")
```

Output:

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_07\Code>python Built_in-Exceptions.py

    1. AssertionError       6. OverflowError
    2. ImportError          7. SyntaxError
    3. IndexError           8. ValueError
    4. KeyError             9. ZeroDivisionError
    5. NameError            10. EXIT

Enter your choice :  1
Enter two numbers to divide
Enter number 1 :  5
Enter number 2 :  10
0.5

    1. AssertionError       6. OverflowError
    2. ImportError          7. SyntaxError
    3. IndexError           8. ValueError
    4. KeyError             9. ZeroDivisionError
    5. NameError            10. EXIT

Enter your choice :  1
Enter two numbers to divide
Enter number 1 :  5
Enter number 2 :  0
Invalid Operation

    1. AssertionError       6. OverflowError
    2. ImportError          7. SyntaxError
    3. IndexError           8. ValueError
    4. KeyError             9. ZeroDivisionError
    5. NameError            10. EXIT

Enter your choice :  2
No module named 'mypackage'

    1. AssertionError       6. OverflowError
    2. ImportError          7. SyntaxError
    3. IndexError           8. ValueError
    4. KeyError             9. ZeroDivisionError
    5. NameError            10. EXIT

Enter your choice :  3
Enter an index (between 0 to 4) :  1
Size of array is 5
10

    1. AssertionError       6. OverflowError
    2. ImportError          7. SyntaxError
    3. IndexError           8. ValueError
    4. KeyError             9. ZeroDivisionError
    5. NameError            10. EXIT

Enter your choice :  3
Enter an index (between 0 to 4) :  12
Size of array is 5
list index out of range
```

```
    1. AssertionError       6. OverflowError
    2. ImportError          7. SyntaxError
    3. IndexError           8. ValueError
    4. KeyError             9. ZeroDivisionError
    5. NameError            10. EXIT

Enter your choice :  4
Enter a key (between `one` to `four`) :  two
Dictionary have 4 keys (from one to four)
Ipsum

    1. AssertionError       6. OverflowError
    2. ImportError          7. SyntaxError
    3. IndexError           8. ValueError
    4. KeyError             9. ZeroDivisionError
    5. NameError            10. EXIT

Enter your choice :  4
Enter a key (between `one` to `four`) :  ten
Dictionary have 4 keys (from one to four)
Key not found 'ten'

    1. AssertionError       6. OverflowError
    2. ImportError          7. SyntaxError
    3. IndexError           8. ValueError
    4. KeyError             9. ZeroDivisionError
    5. NameError            10. EXIT

Enter your choice :  5
20
name 'unknownVariable' is not defined

    1. AssertionError       6. OverflowError
    2. ImportError          7. SyntaxError
    3. IndexError           8. ValueError
    4. KeyError             9. ZeroDivisionError
    5. NameError            10. EXIT

Enter your choice :  6
0 3.0
1 9.0
2 81.0
3 6561.0
4 43046721.0
5 1853020188851841.0
6 3.4336838202925124e+30
7 1.1790184577738583e+61
8 1.3900845237714473e+122
9 1.9323349832288915e+244
Overflowed after  1.9323349832288915e+244 (34, 'Result too large')
```

```
    1. AssertionError       6. OverflowError
    2. ImportError          7. SyntaxError
    3. IndexError           8. ValueError
    4. KeyError             9. ZeroDivisionError
    5. NameError            10. EXIT

Enter your choice :  7
Valid syntax
invalid syntax (<string>, line 1)

    1. AssertionError       6. OverflowError
    2. ImportError          7. SyntaxError
    3. IndexError           8. ValueError
    4. KeyError             9. ZeroDivisionError
    5. NameError            10. EXIT

Enter your choice :  8
could not convert string to float: 'Aamir'

    1. AssertionError       6. OverflowError
    2. ImportError          7. SyntaxError
    3. IndexError           8. ValueError
    4. KeyError             9. ZeroDivisionError
    5. NameError            10. EXIT

Enter your choice :  9
Enter a number to divide from :  3
6.666666666666667

    1. AssertionError       6. OverflowError
    2. ImportError          7. SyntaxError
    3. IndexError           8. ValueError
    4. KeyError             9. ZeroDivisionError
    5. NameError            10. EXIT

Enter your choice :  9
Enter a number to divide from :  0
division by zero

    1. AssertionError       6. OverflowError
    2. ImportError          7. SyntaxError
    3. IndexError           8. ValueError
    4. KeyError             9. ZeroDivisionError
    5. NameError            10. EXIT

Enter your choice :  10
*** E X I T I N G ***
```

```
# class User_Error is extended from super class Exception
class User_Error(Exception):
    # Constructor method
    def __init__(self, value):
        self.value = value
    # __str__ display function
    def __str__(self):
        return(repr(self.value))

try:
    raise(User_Error("User defined error"))
    # Value of Exception is stored in error

except User_Error as error:
    print('A New Exception occured:',error.value)
```

Output:

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_07\Code>python User_defined-Exception.py
A New Exception occured: User defined error
```

Conclusion:

Hence we have successfully understood and used the concept of Modules, Packages and error handling in python