PYTHON QP SOLUTION

CBCS SCHEME

USN | | | | | | | | | | |

1BPLC105B

First Semester B.E./B.Tech. Degree Examination, Dec.2025/Jan.2026
Python Programming

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
Q.1	a.	Explain the salient features of Python.	6	L2	CO1
	b.	Describe the Collatz 3n + 1sequence and explain how iteration and conditional statements are used in its implementation.	8	L2	CO1
	c.	Write a Python program to calculate factorial of a number.	6	L3	CO1
		OR			
Q.2	a.	Explain Syntax Error, Runtime Error, and Semantic error with an example.	6	L2	CO1
	b.	Explain break and continue statements with example.	8	L2	CO1
	c.	Write a Python program to print only even Fibonacci numbers up to n terms.	6	L3	CO1
		Module – 2			
Q.3	a.	Explain Python string handling method with example : find ( ) , Upper ( ) , isupper ( ) , len ( ).	8	L2	CO2
	b.	Tuples are immutable. Explain with an example. And also differentiate between tuple and list.	5	L2	CO2
	c.	Write a Python program to count occurrences of characters in a string and print the count.	7	L3	CO2
		OR			
Q.4	a.	Explain the string operations in Python for slicing, concatenation, repetition and comparison with suitable example.	8	L2	CO2
	b.	What is a list? Explain append ( ), insert ( ), remove ( ), pop () with example.	5	L2	CO2
	c.	Develop a Python program to sort a given list of numbers using Bubble sort from lowest to highest order.	7	L3	CO2
		Module – 3			
Q.5	a.	Explain dictionary methods like keys( ), values( ), items( ) with an example.	6	L2	CO2
	b.	What is Broadcasting in NumPy. Develop a program to illustrate Broadcasting array elements.	6	L3	CO3

	c.	Develop a Python program to count the number of lines and words in the file.	8	L3	CO4
		OR			
Q.6	a.	Explain the concept of aliasing and copying in dictionary with an example.	6	L2	CO2
	b.	What is Masking in NumPy. Develop a program to illustrate masking to filter array elements.	6	L3	CO3
	c.	Develop a Python program to sort the contents of a text file in reverse order and write the sorted contents in to a separate file.	8	L3	CO4
		Module – 4			
Q.7	a.	Discuss the various methods of importing modules in python program.	6	L2	CO3
	b.	Define class and object. Explain with syntax and an example how to define class in Python. How to initiate a class and how the class members are accessed.	8	L2	CO5
	c.	Develop a Python program to illustrate how variable lookup follows LEGB (Local, Enclosing, Global, Built-in) rule.	6	L3	CO3
		OR			
Q.8	a.	Explain how to create user-defined modules in Python with an example. Also discuss the uses of user-defined modules.	6	L2	CO3
	b.	Explain __init__( ) and __str__( ) methods with an example.	8	L2	CO5
	c.	Develop a program that simulates a simple stopwatch that records random time intervals and calculates the average elapsed time.	6	L3	CO3
		Module – 5			
Q.9	a.	Briefly explain Assertion and raising an exception.	4	L2	CO5
	b.	What is polymorphism? Develop a program to illustrate polymorphism by defining a common interface method in two different classes.	8	L3	CO5
	c.	Create a Python class Point with attributes x and y. Demonstrate sameness using 'is' operator, deep equality using ==, and show the effect of mutability when modifying one reference.	8	L3	CO5
		OR			
Q.10	a.	Explain the term objects are mutable with an example.	4	L2	CO5
	b.	What is operator overloading? Define a Class Complex. Overload the + operator to add two complex numbers. Write a Python program to read N ($N \geq 2$) complex numbers and find their cumulative sum using operator overloading.	8	L3	CO5
	c.	Explain the need for exception handling in Python. Develop a program to illustrate: try, except, else, finally blocks and also show how to raise an exception.	8	L3	CO5

* * * * *

Module -1

Q.1 a.Explain the salient features of Python.

Ans:Salient Features of Python

1. **Simple and Easy to Learn**
   Python has a clean and readable syntax similar to English, making it beginner-friendly.
2. **Interpreted Language**
   Python executes code line by line using an interpreter, which makes debugging easier.
3. **High-Level Language**
   No need to manage memory manually; Python handles it automatically.
4. **Object-Oriented**
   Supports OOP concepts like classes, objects, inheritance, and polymorphism.

5. **Portable / Platform Independent**
   Python programs can run on Windows, Linux, macOS, etc., without modification.
6. **Large Standard Library**
   Comes with built-in modules for file handling, networking, databases, math, etc.
7. **Open Source and Free**
   Python is freely available and maintained by a large global community.
8. **Extensible and Embeddable**
   Python can integrate with languages like C/C++ and can be embedded into applications.
9. **Supports Multiple Programming Paradigms**
   Supports procedural, object-oriented, and functional programming.
10. **Rich Support for GUI and Web Development**
    Frameworks like Django, Flask, Tkinter help in web and GUI development.

b.Describe the Collatz 3n + lsequence and explain how iteration and conditional statements are used in its implementation.

Ans: **The Collatz 3n + 1 sequence**

```
n = 1027371

while n != 1:
    print(n, end=", ")
    if n % 2 == 0:          # n is even
        n = n // 2
    else:                   # n is odd
        n = n * 3 + 1
print(n, end=".\n")
```

The condition for continuing with this loop is n != 1, so the loop will continue running until it reaches its termination condition, (i.e. n == 1).

Each time through the loop, the program outputs the value of n and then checks whether it is even or odd.

If it is even, the value of n is divided by 2 using integer division.

If it is odd, the value is replaced by n * 3 + 1.

Since n sometimes increases and sometimes decreases, there is no obvious proof that n will ever reach 1, or that the program terminates. For some particular values of n, we can prove termination. For example, if the starting value is a power of two, then the value of n will be even each time through the loop until it reaches 1. The previous example ends with such a sequence, starting with 16.

C.Write a Python program to calculate factorial ofa number.

Program:

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

num = int(input("Enter a number: "))
print("Factorial of", num, "is", factorial(num))
```

OUTPUT:

If input = 5

Output = 120

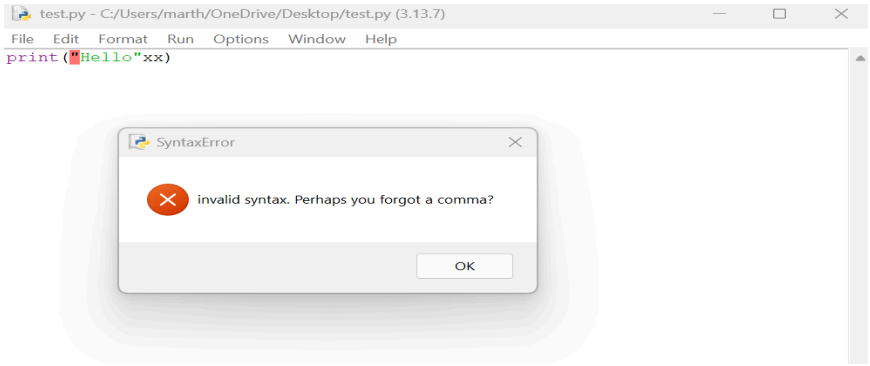
(5 × 4 × 3 × 2 × 1 = 120)

Q.2.a.Explain Syntax Error, Runtime Error, and Semantic error with an example.

Ans:SYNTAX ERROR

- Python can only execute a program if the program is syntactically correct; otherwise, the process fails and returns an error message.
- Syntax refers to the structure of a program and the rules about that structure.
- If there is any syntax error, Python interpreter will throw SyntaxError.

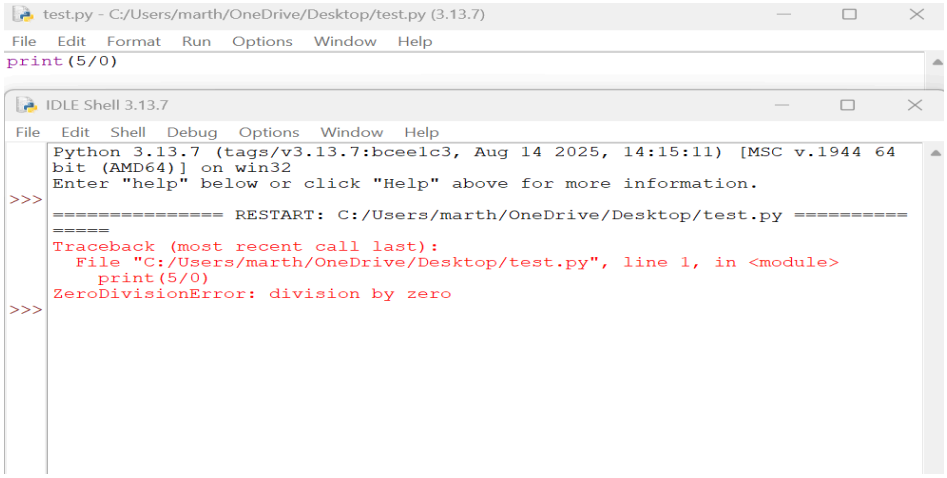
eg:-



RUNTIME ERROR

- The error does not appear until you run the program.

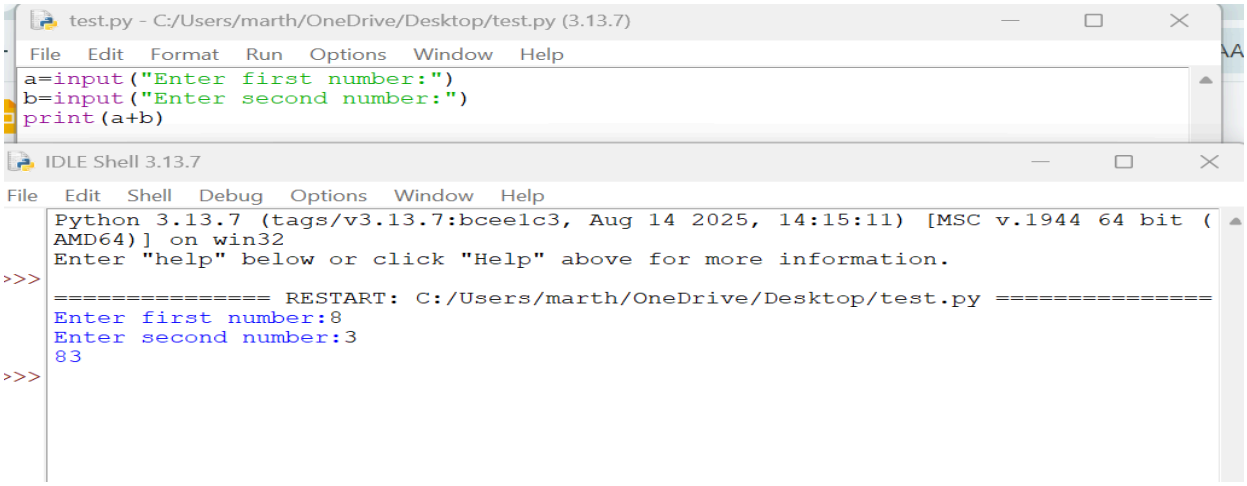
  eg:-



SEMANTIC ERROR

- If there is a semantic error in your program, it will run successfully, in the sense that the computer will not generate any error messages, but it will not do the right thing.
- The meaning of the program (its semantics) is wrong.

  eg:-



b.Explain break and continue statements with example.

Ans:Break:

The break statement is used to immediately leave the body of its loop. The next statement to be executed is the first one after the body

```
for i in [12, 16, 17, 24, 29]:
    if i % 2 == 1:  # If the number is odd
        break       #  ... immediately exit the loop
    print(i)
print("done")
```

This prints:

```
12
16
done
```

continue statement

This is a control flow statement that causes the program to immediately skip the processing of the rest of the body of the loop, for the current iteration. But the loop still carries on running for its remaining iterations:

```
for i in [12, 16, 17, 24, 29, 30]:
    if i % 2 == 1:       # If the number is odd
       continue          # Don't process it
    print(i)
print("done")
```

This prints:

```
12
16
24
30
done
```

C.Write a Python program to print only even Fibonacci numbers up to n terms.

Program:

```
n = int(input("Enter the number of terms: "))

a = 0

b = 1

print("Even Fibonacci numbers:")

for i in range(n):

    if a % 2 == 0:

        print(a, end=" ")

    a, b = b, a + b
```

OUTPUT:

If input = 10Fibonacci sequence (10 terms):
0 1 1 2 3 5 8 13 21 34

Even Fibonacci numbers:
0 2 8 34

MODULE-2

Q.3.a.Explain Python string handling method with example : find (), Upper(), isupper ( ), len ( ).

Ans:

1) ***find():***

Returns the lowest index of the substring if found.

If not found, it returns -1 (does NOT raise an error).

Syntax:

string.find(substring)

Example:

```
text = "Hello World"

print(text.find("World"))   # Output: 6

print(text.find("Python"))  # Output: -1
```

2) ***upper():***

Converts all characters of the string to uppercase.

Syntax:

string.upper()

Example:

```
text = "Hello World"

print(text.upper())   # Output: HELLO WORLD
```

3) ***isupper():***

Returns True if all characters in the string are uppercase.

Otherwise, it returns False.

Syntax:

string.isupper()

Example:

```
text1 = "HELLO"

text2 = "Hello"

print(text1.isupper())  # Output: True

print(text2.isupper())  # Output: False
```

4) ***len():***

Returns the total number of characters in a string (including spaces).

Syntax:

len(string)

Example:

```
text = "Hello World"

print(len(text))   # Output: 11
```

Q.3.b.Tuples are immutable. Explain with an between tuple and list.

Ans:

- **Tuple (Immutable)**

  A tuple is created using parentheses ().

  t = (10, 20, 30)

```
t[0] = 100   # This will give an error
```

Output:

TypeError: 'tuple' object does not support item assignment

Cannot:

Change elements

Add elements

Remove elements

➢ **List (Mutable)**

A list is created using square brackets [].

```
l = [10, 20, 30]

l[0] = 100

print(l)
```

Output:

[100, 20, 30]

Can:

Change elements

Add elements (append())

Remove elements (remove())

Difference Between Tuple and List:

Feature	Tuple	List
Brackets Used	()	[]
Mutable?	No (Immutable)	Yes (Mutable)
Can modify elements?	No	Yes
Memory usage	Less	More
Speed	Faster	Slower

Example Showing Difference:

```
# Tuple
t = (1, 2, 3)
# t.append(4)  # Error
# List
l = [1, 2, 3]
l.append(4)
print(l)   # [1, 2, 3, 4]
```

Q.3.c.Write a Python program to count occurrences of characters in a string and print the count.

Ans:

```
# Program to count occurrences of characters in a string

string = input("Enter a string: ")

count = {}

for char in string:

    if char in count:

        count[char] += 1

    else:

        count[char] = 1

print("Character occurrences:")

for char in count:

    print(char, ":", count[char])
```

Output:

```
Enter a string: hello

Character occurrences:

h : 1

e : 1

l : 2

o : 1
```

Q.4.a.Explain the string operations in Python for slicing, concatenation, repetition and comparison with suitable example.

Ans:

➢ **Slicing**

Slicing is used to extract a portion (substring) of a string.

Syntax:

string[start : end : step]

start → starting index (inclusive)

end → ending index (exclusive)

step → increment (optional)

Example:

```
text = "Python"
print(text[0:4])   # Pyth
print(text[2:])    # thon
print(text[:3])    # Pyt
print(text[-1])    # n
print(text[::-1])  # nohtyP (reverse)
```

➢ **Concatenation**

Concatenation joins two or more strings using the + operator.

Example:

```
str1 = "Hello"
str2 = "World"
```

```
result = str1 + " " + str2

print(result)
```

Output:

```
Hello World
```

➢ **Repetition**

Repetition repeats a string multiple times using the * operator.

Example:

```
text = "Hi "

print(text * 3)
```

Output:

```
Hi Hi Hi
```

➢ **Comparison**

Strings can be compared using comparison operators like: ==, !=, >, <, >=, <=

Comparison is based on ASCII (Unicode) values.

Example:

```
a = "apple"

b = "banana"

print(a == b)   # False

print(a < b)    # True

print(a > b)    # False
```

Q.4.b.What is a list? Explain append (), insert (), remove (), pop () with example.

Ans:

A list in Python is an ordered collection of items.

Mutable (elements can be changed)

Can store different data types

Example:

```
my_list = [10, 20, 30, "Python"]

print(my_list)
```

1. ***append()***

   Adds an element at the end of the list.

   Syntax:

   ```
   list.append(element)
   ```

   Example:

   ```
   numbers = [1, 2, 3]

   numbers.append(4)

   print(numbers)
   ```

   Output:

   ```
   [1, 2, 3, 4]
   ```

2. ***insert()***

   Inserts an element at a specific position.

   Syntax:

list.insert(index, element)

Example:

numbers = [1, 2, 3]

numbers.insert(1, 10)

print(numbers)

Output:

[1, 10, 2, 3] (10 is inserted at index 1)

3. ***remove()***

   Removes the first occurrence of a specified element.

   Syntax:

   list.remove(element)

   Example:

   numbers = [1, 2, 3, 2]

   numbers.remove(2)

   print(numbers)

   Output:

   [1, 3, 2] (Only the first 2 is removed)

4. ***pop()***

   Removes and returns the element at a given index. If no index is given, it removes the last element.

   Syntax:

   list.pop(index)

Example:

```
numbers = [1, 2, 3]

numbers.pop()

print(numbers)

numbers.pop(0)

print(numbers)
```

Output:

```
[1, 2]

[2]
```

Q.4.c.Develop a Python program to sort a given list of numbers using Bubble sort from lowest to highest order.

Ans:

```
# Program to sort a list using Bubble Sort

# Taking input from user

numbers = list(map(int, input("Enter numbers separated by space: ").split()))

n = len(numbers)

# Bubble Sort Algorithm

for i in range(n):

    for j in range(0, n - i - 1):

        if numbers[j] > numbers[j + 1]:

            # Swap
```

```
            numbers[j], numbers[j + 1] = numbers[j + 1], numbers[j]
print("Sorted list (Ascending Order):")
print(numbers)
```

Output:

Enter numbers separated by space: 5 2 8 1 3

Sorted list (Ascending Order):

[1, 2, 3, 5, 8]

MODULE-3

5.a) Explain dictionary methods like keys( ), values( ), items( ) with an example.

Introduction to Dictionary (1 Mark)

A **Dictionary** in Python is a collection of **key–value pairs**.

- Keys must be **unique**
- Defined using { }
- Example:

```
student = {"roll": 101, "name": "Arun", "marks": 95}
```

5. **`keys()` Method (2 Marks)**

🔹 **Definition:**

The `keys()` method returns a **view object** that contains all the keys of the dictionary.

🔹 **Syntax:**

```
dict_name.keys()
```

🔹 **Example:**

```
student = {"roll": 101, "name": "Arun", "marks": 95}

print(student.keys())
```

🔹 **Output:**

```
dict_keys(['roll', 'name', 'marks'])
```

✔ Returns only keys
✔ Result type: `dict_keys`

3. `values()` Method (1.5 Marks)

🔹 **Definition:**

The `values()` method returns all the **values** in the dictionary.

🔹 **Syntax:**

```
dict_name.values()
```

🔹 **Example:**

```
print(student.values())
```

🔹 **Output:**

```
dict_values([101, 'Arun', 95])
```

✔ Returns only values

4. `items()` Method (1.5 Marks)

🔹 **Definition:**

The `items()` method returns both **keys and values** as tuples.

🔹 **Syntax:**

```
dict_name.items()
```

🔹 **Example:**

```
print(student.items())
```

🔹 **Output:**

```
dict_items([('roll', 101), ('name', 'Arun'), ('marks', 95)])
```

✔ Returns list of tuples
✔ Useful in loops

🔹 **Example Using Loop:**

```
for key, value in student.items():
    print(key, value)
```

5. b) What is Broadcasting in NumPy. Develop a program to illustrate broadcasting array elements.

Definition of Broadcasting (2 Marks)

Broadcasting in **NumPy** is a mechanism that allows arithmetic operations to be performed on arrays of **different shapes**, without explicitly reshaping them.

It automatically expands the smaller array to match the shape of the larger array.

Rules of Broadcasting (1 Mark)

Two arrays are compatible for broadcasting if:

1. Their dimensions are equal, OR
2. One of the dimensions is **1**

Broadcasting starts comparing dimensions from the **rightmost side**.

Example Program to Illustrate Broadcasting (3 Marks)

Example 1: Scalar Broadcasting

```
import numpy as np

# Create a 1D array

a = np.array([10, 20, 30, 40])

# Add scalar value

result = a + 5

print("Original Array:", a)

print("After Broadcasting (a + 5):", result)
```

🔹 **Output:**

```
Original Array: [10 20 30 40]

After Broadcasting (a + 5): [15 25 35 45]
```

✔ Here, scalar 5 is automatically added to each element.

🔹 **Example 2: Array Broadcasting (1D + 2D)**

```
import numpy as np

A = np.array([[1, 2, 3],

              [4, 5, 6]])

B = np.array([10, 20, 30])

result = A + B

print("Matrix A:\n", A)
```

```
print("Array B:", B)

print("After Broadcasting (A + B):\n", result)
```

🔹 **Output:**

```
Matrix A:

 [[1 2 3]

  [4 5 6]]

After Broadcasting:

 [[11 22 33]

  [14 25 36]]
```

✔ 1D array `B` is automatically expanded to match rows of matrix `A`.

Advantages of Broadcasting

- Avoids writing loops
- Improves performance
- Memory efficient

5. c) Develop a Python program to count the number of lines and words in the file.

```
# Program to count number of lines and words in a file

filename = input("Enter the file name: ")

try:

    with open(filename, 'r') as file:

        line_count = 0
```

```
        word_count = 0
        for line in file:
            line_count += 1
            words = line.split()
            word_count += len(words)
        print("Number of lines:", line_count)
        print("Number of words:", word_count)
except FileNotFoundError:
    print("File not found. Please check the file name.")
```

Sample Input File

Assume file `sample.txt` contains:

```
Python is easy.
It is powerful.
```

Output:

```
Number of lines: 2
Number of words: 5
```

6. a) Explain the concept of aliasing and copying in dictionary with an example.

Introduction (1 Mark)

In Python, dictionaries are **mutable objects**.
When assigning one dictionary to another, it may create:

- **Aliasing** (same memory reference)
- **Copying** (separate memory reference)

Understanding the difference is important to avoid unexpected modifications.

Aliasing in Dictionary (2 Marks)

🔹 **Definition:**

Aliasing occurs when two variables refer to the **same dictionary object** in memory.

If one variable changes the dictionary, the other also reflects the change.

🔹 **Example:**

```
student1 = {"name": "Arun", "marks": 90}

student2 = student1   # Aliasing

student2["marks"] = 95

print("student1:", student1)

print("student2:", student2)
```

🔹 **Output:**

```
student1: {'name': 'Arun', 'marks': 95}

student2: {'name': 'Arun', 'marks': 95}
```

✔ Both variables refer to the same object
✔ Change in one affects the other

Copying in Dictionary (3 Marks)

🔹 **Definition:**

Copying creates a **new dictionary object** with the same elements.

Changes in one dictionary will **not affect** the other.

There are two types:

- Shallow Copy
- Deep Copy (advanced concept)

🔹 **Method 1: Using `copy()` (Shallow Copy)**

```
student1 = {"name": "Arun", "marks": 90}

student2 = student1.copy()   # Copying

student2["marks"] = 95

print("student1:", student1)

print("student2:", student2)
```

🔹 **Output:**

```
student1: {'name': 'Arun', 'marks': 90}

student2: {'name': 'Arun', 'marks': 95}
```

✔ Changes do not affect original dictionary

🔹 **Method 2: Using `dict()` Constructor**

```
student2 = dict(student1)
```

This also creates a copy.

6. b) What is Masking in NumPy. Develop a program to illustrate masking to filter array elements.

Definition of Masking (2 Marks)

Masking in **NumPy** refers to the technique of using **Boolean arrays** to filter or select specific elements from a NumPy array based on a condition.

It allows us to extract elements that satisfy certain criteria without using loops.

Concept of Boolean Mask (1 Mark)

When a condition is applied to a NumPy array:

```
array > 10
```

It produces a **Boolean array** (True/False values).

This Boolean array is called a **mask**.

Program to Illustrate Masking (3 Marks)

- **Example 1: Filter elements greater than 10**

```
import numpy as np

# Create NumPy array

arr = np.array([5, 12, 8, 20, 3, 15])

# Create mask

mask = arr > 10

# Apply mask

filtered = arr[mask]

print("Original Array:", arr)

print("Mask:", mask)

print("Filtered Elements (>10):", filtered)
```

- **Output:**

```
Original Array: [ 5 12  8 20  3 15]
```

```
Mask: [False  True False  True False  True]

Filtered Elements (>10): [12 20 15]
```

🔹 **Example 2: Filter Even Numbers**

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6])

even_numbers = arr[arr % 2 == 0]

print("Even Numbers:", even_numbers)
```

🔹 **Output:**

```
Even Numbers: [2 4 6]
```

Advantages of Masking

- Eliminates loops
- Faster computation
- Cleaner and readable code

6. c) Develop a Python program to sort the contents of a text file in reverse order and write the sorted contents into a separate file.

Program to sort contents of a file in reverse order

input_file = input("Enter input file name: ")

output_file = input("Enter output file name: ")

```
try:
    # Read contents from input file
    with open(input_file, 'r') as file:
        lines = file.readlines()
    # Remove newline characters and strip spaces
    lines = [line.strip() for line in lines]
    # Sort in reverse order
    lines.sort(reverse=True)
    # Write sorted contents to output file
    with open(output_file, 'w') as file:
        for line in lines:
            file.write(line + "\n")
    print("File sorted in reverse order successfully.")
except FileNotFoundError:
    print("Input file not found.")
```

Output:

Sample Input File

Input File (data.txt)

apple

orange

banana

mango

Output File (sorted.txt)

orange

mango

banana

apple

MODULE-4

7. a) Discuss the various methods of importing modules in python program.

In Python, modules are imported using different methods depending on the requirement.

1. Import the entire module

```
import math
print(math.sqrt(25))
```

- All functions are accessed using module_name.function_name.

2. Import specific functions from a module

```
from math import sqrt
print(sqrt(36))
```

- No need to use module name.

3. Import multiple specific functions

```
from math import sqrt, pi
print(sqrt(16), pi)
```

4. Import all names using * (not recommended)

```
from math import *
print(sqrt(49))
```

- May cause namespace conflicts.

5. Import with alias

```
import math as m
print(m.sqrt(64))
```

6. Import specific function with alias

```
from math import sqrt as s
print(s(81))
```

7. b) Define class and object. Explain with syntax and an example how to define class in Python. How to initiate a class and how the class members are accessed.

Class Definition – A class is a blueprint or template for creating objects. It defines attributes (data) and methods (functions).

Object Definition – An object is an instance of a class. It represents a real-world entity.

Syntax of Class:-

```
class ClassName:
    def __init__(self, parameters):
```

```
        self.variable = parameters

    def method_name(self):
        # body
```

Example Program

```
class Student:
   def __init__(self, name, marks):
       self.name = name
       self.marks = marks

   def display(self):
       print("Name:", self.name)
       print("Marks:", self.marks)

# Creating (initiating) object
s1 = Student("Rahul", 85)

# Accessing class members
print(s1.name)      # Access attribute
s1.display()        # Access method
```

7. c) Develop a Python program to illustrate how variable lookup follows LEGB (Local, Enclosing, Global, Built-in) rule.

LEGB represents the order in which Python searches for a variable name:

1. L – Local → Inside current function
2. E – Enclosing → Inside outer (enclosing) function
3. G – Global → Defined at top level of file

4. B – Built-in → Predefined names in Python (like print(), len())

Python searches in this order:
Local → Enclosing → Global → Built-in

```
x = "Global Variable"

def outer():
    x = "Enclosing Variable"

    def inner():
        x = "Local Variable"
        print("Inside inner():", x)    # Local scope

    inner()
    print("Inside outer():", x)        # Enclosing
scope

outer()
print("Outside functions:", x)         # Global scope
```

Output:-

Inside inner(): Local Variable
Inside outer(): Enclosing Variable
Outside functions: Global Variable

8. a) Explain how to create user-defined modules in Python with an example. Also discuss the uses of user-defined modules.

Definition:

A user-defined module is a Python file created by the programmer that contains functions, classes, or variables which can be reused in other programs.

Step 1: Create a Python File

Create a file with `.py` extension.
Example: `mymodule.py`
(The file name becomes the module name.)

Step 2: Define Functions / Classes in the Module

mymodule.py

```
def add(a, b):
  return a + b

def greet(name):
  return "Hello " + name
```

Step 3: Import the Module in Another Program

main.py

```
import mymodule

print(mymodule.add(5, 3))
print(mymodule.greet("Anita"))
```

Output

```
8
Hello Anita
```

Uses of User-Defined Modules

1. Code reusability
2. Better program organization
3. Avoid code repetition
4. Easy maintenance and debugging

8. b) Explain __init__ () and __str__ () methods with an example.

1. `__init__()` Method

- Constructor method.
- Automatically called when object is created.
- Initializes object attributes.

2. `__str__()` Method

- Returns string representation of object.
- Called when print(object) is used.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f"Name: {self.name}, Age: {self.age}"

p1 = Person("Ravi", 25)
print(p1)
```

Output:-

Name: Ravi, Age: 25

8. c) Develop a program that simulates a simple stopwatch that records random time intervals and calculates the average elapsed time.

```
import time
import random

def stopwatch_simulation():
```

```
    intervals = []

    for i in range(5):
        start = time.time()

        # Simulate random delay (1 to 3 seconds)
        delay = random.randint(1, 3)
        time.sleep(delay)

        end = time.time()
        elapsed = end - start
        intervals.append(elapsed)

        print(f"Interval {i+1}: {elapsed:.2f}
seconds")

    average = sum(intervals) / len(intervals)
    print(f"\nAverage Time: {average:.2f} seconds")

stopwatch_simulation()
```

- `time.time()` records current time.
- `time.sleep()` pauses execution.
- Random delays simulate stopwatch intervals.
- Average time is calculated using sum/length.

Output:-

Interval 1: 1.00 seconds
Interval 2: 2.00 seconds
Interval 3: 3.00 seconds
Interval 4: 1.00 seconds
Interval 5: 2.00 seconds

Average Time: 1.80 seconds

MODULE-5

Q.9.a.Briefly explain Assertion and raising an exception.

Ans:Assertion

- An assertion is a statement used to test whether a condition is true.
- If the condition is True, the program continues.
- If the condition is False, Python raises an AssertionError.
- It is mainly used for debugging purposes.

Syntax:

assert condition, "Error message"

Example:

```
x = 10

assert x > 0, "x must be positive"
```

If `x` is negative, it raises an error.

Raising an Exception

- Raising an exception means forcing an error intentionally using the `raise` keyword.
- It is used to handle invalid conditions in a program.

Syntax:

raise ExceptionType("Error message")

Example:

```
age = -5

if age < 0:

    raise ValueError("Age cannot be negative")
```

Q.9.b. What is polymorphism? Develop a program to illustrate polymorphism by defining a common interface method in two different classes.

Ans: Polymorphism means *"many forms"*.
In Python, polymorphism allows different classes to have methods with the same name but different implementations.It enables us to use a common interface for different types of objects.

Example Program: Common Interface in Two Different Classes

Here, both classes have a common method area(), but they calculate it differently.

```
class Circle:

    def __init__(self, radius):

        self.radius = radius

        def area(self):

        return 3.14 * self.radius * self.radius

class Rectangle:

    def __init__(self, length, breadth):

        self.length = length

        self.breadth = breadth

        def area(self):
```

```
        return self.length * self.breadth

# Polymorphism in action

c = Circle(5)

r = Rectangle(4, 6)

print("Area of Circle:", c.area())

print("Area of Rectangle:", r.area())
```

Output

Area of Circle: 78.5

Area of Rectangle: 24

Q.9.c.Create a Python class Point with attributes x and y. Demonstrate sameness using 'is' operator, deep quality using , mutability when modifying one reference.

Ans:

is checks object identity (same memory), while == checks value equality, and mutability affects all references pointing to the same object.

Program:

```
class Point:

    def __init__(self, x, y):

        self.x = x

        self.y = y

    # Define equality (deep equality)
```

```
    def __eq__(self, other):
        if isinstance(other, Point):
            return self.x == other.x and self.y == other.y
        return False
# Creating objects
p1 = Point(2, 3)
p2 = Point(2, 3)
p3 = p1   # Reference copy (same object)
#Sameness using 'is'
print("p1 is p2:", p1 is p2)   # False (different objects)
print("p1 is p3:", p1 is p3)   # True (same object)
#Equality using '=='
print("p1 == p2:", p1 == p2)   # True (values are same)
# Mutability demonstration
p3.x = 10   # Modify through p3
print("After modifying p3.x = 10")
print("p1.x:", p1.x)   # Changes because p1 and p3 refer to same object
print("p3.x:", p3.x)
```

Output:

```
p1 is p2: False
p1 is p3: True
```

p1 == p2: True

After modifying p3.x = 10

p1.x: 10

p3.x: 10

Q.10.a.Explain the term objects are mutable with an example.

Ans:

An object is mutable if its value (internal state) can be changed after it is created. In Python, some objects can be modified after creation. If you can change their contents without creating a new object, they are called mutable objects.

Examples of mutable objects:

List

Dictionary

Set

Most user-defined class objects

Example with List (Mutable Object)

```
numbers = [1, 2, 3]

print("Before change:", numbers)

numbers[0] = 10   # Modifying the list

print("After change:", numbers)
```

Output:

Before change: [1, 2, 3]

After change: [10, 2, 3]

The list changed without creating a new list. This shows that lists are mutable.

Example Showing Same Reference Effect

a = [1, 2, 3]

b = a # Both refer to same object

b[0] = 100

print("a:", a)

print("b:", b)

Output:

a: [100, 2, 3]

b: [100, 2, 3]

Changing b also changes a because both refer to the same mutable object

Q.10.a. Briefly explain Assertion and raising an exception.

Assertion (2 Marks)

- **Definition:** An **Assertion** is a debugging tool used to test whether a condition is true during program execution.

If the condition is **False**, Python raises an `AssertionError`.

- **Syntax:**

```
assert condition, "Error message"
```

- **Example:**

```
x = 10
assert x > 0, "x must be positive"
print("Valid value")
```

✔ If `x > 0` → Program runs normally
✖ If condition fails → `AssertionError` is raised

🔹 **Purpose:**

- Used during debugging
- Helps detect logical errors early

Raising an Exception (2 Marks)

🔹 **Definition:**

Raising an exception means forcing an error to occur using the `raise` keyword.

It is used when we want to handle invalid conditions explicitly.

🔹 **Syntax:**

```
raise ExceptionType("Error message")
```

🔹 **Example:**

```
age = -5
if age < 0:
    raise ValueError("Age cannot be negative")
```

✔ Stops program execution
✔ Displays custom error message

Q.10.b. What is polymorphism? Develop a program to illustrate polymorphism by defining a common interface method in two different classes.

Definition of Polymorphism (2 Marks)

Polymorphism is an important concept in **Object-Oriented Programming (OOP)** that allows the same method name to perform different tasks depending on the object.

The word polymorphism means:

“Many Forms”

In Python, polymorphism allows different classes to define methods with the same name, and those methods can be called in a uniform way.

Types of Polymorphism (1 Mark)

1. **Compile-time Polymorphism** (Method Overloading – limited in Python)
2. **Run-time Polymorphism** (Method Overriding / Duck Typing)

Program to Illustrate Polymorphism (5 Marks)

🔹 **Problem:**

Define two classes with a common method `area()` to calculate area of different shapes.

🔹 **Python Program**

```
# Class 1
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width


# Class 2
class Circle:
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius
```

```
# Polymorphism in action
def display_area(shape):
    print("Area:", shape.area())


# Create objects
r = Rectangle(10, 5)
c = Circle(7)

display_area(r)
display_area(c)
```

🔹 **Output:**

```
Area: 50
Area: 153.86
```

Explanation

- Both classes define a common method `area()`
- The function `display_area()` calls `shape.area()`
- Depending on the object passed (`Rectangle` or `Circle`), the correct method is executed
- This demonstrates **runtime polymorphism**

Q.10.c. Create a Python class Point with attributes x and y. Demonstrate sameness using 'is' operator, deep equality using == , and show the effect of mutability when modifying one reference.

Concept Explanation (2 Marks)

- **`is` operator** → Checks whether two variables refer to the **same object (memory location)**.
- **`==` operator** → Checks whether two objects have **equal values (deep equality)**.
- Since objects are **mutable**, modifying one reference affects all aliases pointing to the same object.

Python Program (4 Marks)

```
# Define Point class
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    # Override equality operator
    def __eq__(self, other):
        if isinstance(other, Point):
            return self.x == other.x and self.y == other.y
        return False

    def __repr__(self):
        return f"Point({self.x}, {self.y})"


# Create objects
p1 = Point(2, 3)
p2 = Point(2, 3)
p3 = p1   # Aliasing (same reference)

# Check sameness using 'is'
print("p1 is p2:", p1 is p2)
print("p1 is p3:", p1 is p3)
```

```
# Check deep equality using '=='
print("p1 == p2:", p1 == p2)

# Demonstrate mutability
p3.x = 10

print("After modifying p3:")
print("p1:", p1)
print("p3:", p3)
```

Output (1 Mark)

```
p1 is p2: False
p1 is p3: True
p1 == p2: True
After modifying p3:
p1: Point(10, 3)
p3: Point(10, 3)
```

Explanation (1 Mark)

- `p1 is p2` → False (different memory locations)
- `p1 is p3` → True (same object reference)
- `p1 == p2` → True (values are equal)
- Modifying `p3.x` changes `p1.x` because both refer to the same object.