Roll No. 50 Exam Seat No.

VIVEKANAND EDUCATION SOCIETY'S INSTITUTE OF TECHNOLOGY

Hashu Advani Memorial Complex, Collector's Colony, R. C. Marg, Chembur, Mumbai – 400074. Contact No. 02261532532

Since 1962

CERTIFICATE

Certified that MR. Shaikh Farhan Ahmed

of FYMCA/B has

satisfactorily completed a course of the necessary tutorials in

Python Programming Lab under my supervision

in the Institute of Technology in the academic year 20 24 - 20 25.

Principal Head of Department

Lab In-charge Subject Teacher

V.E.S. Institute of Technology, Collector Colony, Chembur, Mumbai, Maharashtra 400047

Department of M.C.A

INDEX

Final Grade	Instructor Signature

Name of Student: Shaikh Farhan Ahmed			
Roll Number: 50		**LAB Assignment Number: 1**	
Title of LAB Assignment: To write, test, and debug Basic Python programs.			
DOP: 08 – 10 – 2024		**DOS: 11 – 10 – 2024**	
CO Mapped: CO1	**PO Mapped: PO3, PO5, PSO1, PSO2**	**Faculty Signature:**	**Marks:**

Practical: 1

AIM : To write,test and debug basic python programs.

Description:

Introduction to Python :
Python is a high-level, interpreted programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991. Python's design philosophy emphasizes code readability, using significant indentation to enhance its visual structure. Python is dynamically typed, which means you don't need to declare the type of variables before using them. It is also versatile, used for web development, data science, artificial intelligence, automation, and many other fields.

Key Features of Python:
1. **Simple Syntax:** Easy to learn and use due to its minimalistic syntax.
2. Interpreted Language: Code is executed line by line, which makes it easier to debug.
3. **Cross-Platform**: Python can run on various platforms such as Windows, macOS, and Linux without modification.
4. **Extensive Libraries:** Python has a rich set of libraries for scientific computing, web development, machine learning, etc.
5. **Community Support:** With a large community, there are numerous resources available for troubleshooting and learning.

Structure of a Basic Python Program
A basic Python program has the following structure:

```
# This is a comment

import random # Importing a module

# Defining variables
x = 10
y = "Hello, World!"
```

z = True

# Performing arithmetic operation
result = x + 5

Import Statements: Libraries like math or numpy can be imported to enhance functionality.
Function Definitions: Functions are reusable blocks of code that perform specific tasks.
Main Logic: The part of the program that is executed when the script runs.

Data Types in Python:
Python supports various data types, which are broadly categorized into the following types:

Numeric Types:
- Integer (int): Whole numbers without a fractional part (e.g., 10, -5).
- Float (float): Numbers with a decimal point (e.g., 3.14, -0.5).
- Complex (complex): Numbers with a real and imaginary part (e.g., 1 + 2j).

Sequence Types:
- String (str): A sequence of characters enclosed in quotes (e.g., "Hello, World!").
- List (list): Ordered and mutable collection of items (e.g., [1, 2, 3]).
- Tuple (tuple): Ordered but immutable collection of items (e.g., (1, 2, 3)).

Mapping Types:
- Dictionary (dict): Unordered, mutable collection of key-value pairs (e.g. {'name': 'Farhan', 'age': 23}).

Set Types:

- Set (set): Unordered collection of unique items (e.g., {1, 2, 3}).
- Frozen Set (frozenset): Immutable version of a set

Boolean Type:

- bool: Represents either True or False.

None Type:

- NoneType: Represents the absence of a value, denoted by None.

Types of Operators in Python

Operators are symbols or keywords that tell the interpreter to perform specific operations.
Python supports the following types of operators:

Arithmetic Operators:
Used for basic mathematical operations.

- + : Addition
- - : Subtraction
- * : Multiplication
- / : Division (returns a float)
- // : Floor Division (returns an integer)
- % : Modulus (returns remainder)
- ** : Exponentiation

Comparison (Relational) Operators:
Used to compare two values, returning a boolean (True or False).

- == : Equal to
- != : Not equal to
- > : Greater than
- < : Less than
- >= : Greater than or equal to
- <= : Less than or equal to

Logical Operators:
Used to combine conditional statements.

- and: Returns True if both conditions are True.
- or: Returns True if at least one condition is True.
- not: Inverts the boolean value.

Assignment Operators:
Used to assign values to variables.

- =: Basic assignment.
- +=, -=, *=, /=, %=, //=, **=: Compound assignment operators.

Bitwise Operators:
Used to perform operations on binary numbers.

- &: AND
- |: OR
- ^: XOR
- ~: NOT
- <<: Left shift
- >>: Right shift

Identity Operators:
Used to compare the memory locations of two objects.

- is: Returns True if both variables point to the same object.
- is not: Returns True if both variables do not point to the same object.

Membership Operators:
Used to test if a value is found in a sequence (such as strings, lists, or tuples).

- in: Returns True if a value exists in the sequence.
- not in: Returns True if a value does not exist in the sequence.

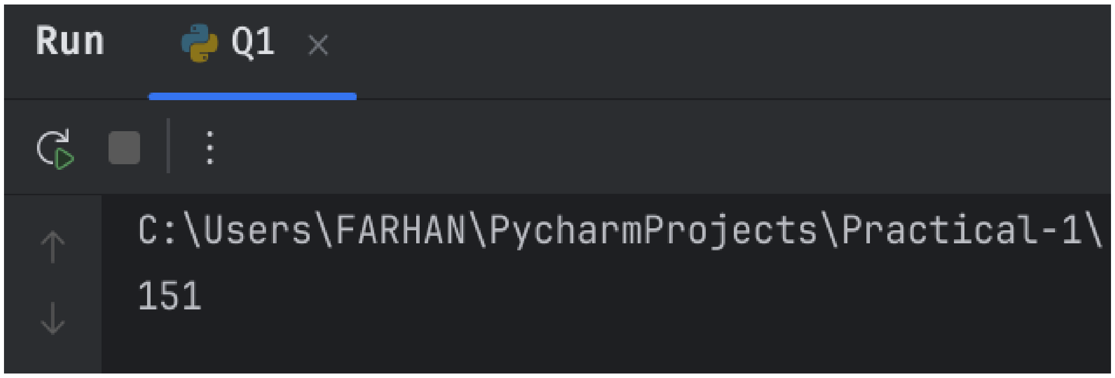
Programs:

1. **Add Three Numbers**

   Source Code:

```
number1 = 110
number2 = 37
sum_of_numbers = number1 + number2
print(sum_of_numbers)
```

Output:



```
C:\Users\FARHAN\PycharmProjects\Practical-1\
151
```

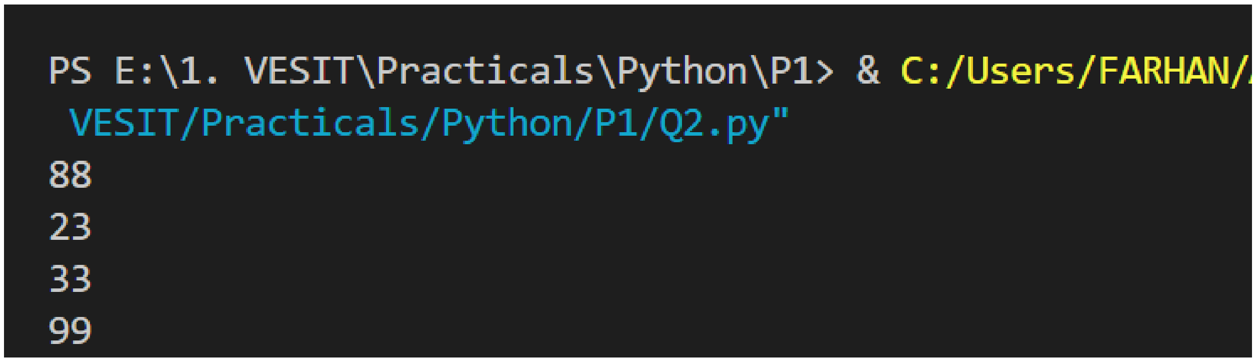
2. **To Swap two No using third variable and without using third variable**

   Source Code:

```
#declaring vairables
a=23
b=88
#swapping variables using third variable
c=a
a=b
b=c
print(a)
print(b)
```

```
#swapping variables without using third variable
x=99
y=33
x=x+y
y=x-y
x=x-y
print(x)
print(y)
```

Output:



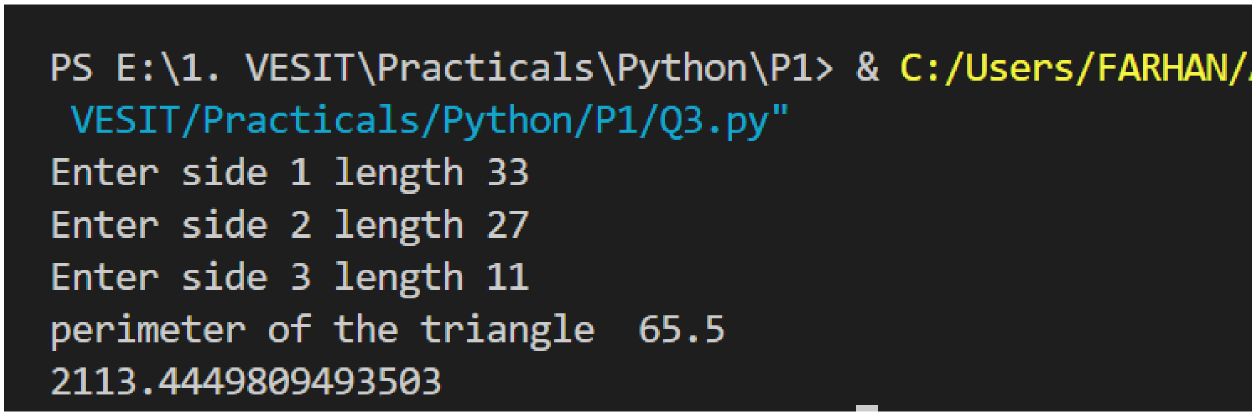
```
PS E:\1. VESIT\Practicals\Python\P1> & C:/Users/FARHAN/
 VESIT/Practicals/Python/P1/Q2.py"
88
23
33
99
```

3. **Calculate area of triangle**

Source Code:

```
from math import sqrt

a =int(input("Enter side 1 length "))
b =int(input("Enter side 2 length "))
c =int(input("Enter side 3 length "))
s= a+b+c/2

print("perimeter of triangle ", s)
area = sqrt(s*(s-a)*(s-b)*(s-c))
print(area)
```

Output:



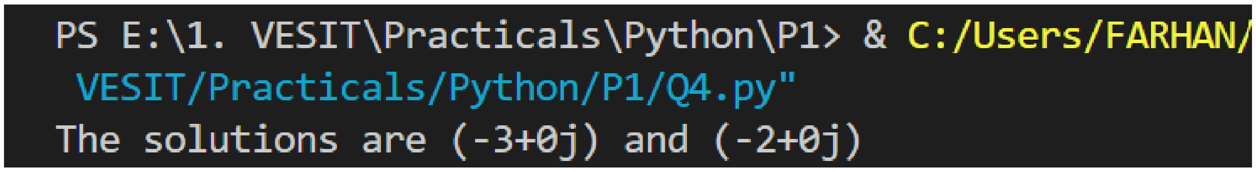
```
PS E:\1. VESIT\Practicals\Python\P1> & C:/Users/FARHAN/
 VESIT/Practicals/Python/P1/Q3.py"
Enter side 1 length 33
Enter side 2 length 27
Enter side 3 length 11
perimeter of the triangle  65.5
2113.4449809493503
```

4. **To Solve Quadratic equation**

Source Code:

```
# Solve the quadratic equation ax**2 + bx + c = 0
# import complex math module
import cmath
a = 1
b = 5
c = 6

# calculate the discriminant
d = (b**2) - (4*a*c)

# find two solutions
sol1 = (-b - cmath.sqrt(d)) / (2*a)
sol2 = (-b + cmath.sqrt(d)) / (2*a)

# Print solutions without using .format()
print("The solutions are", sol1, "and", sol2)
```

Output:



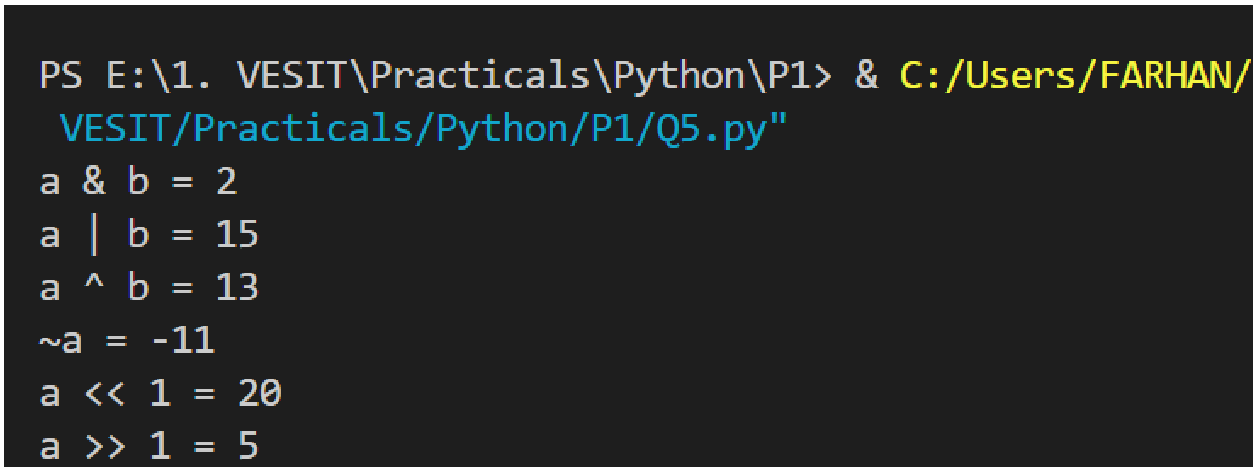
```
PS E:\1. VESIT\Practicals\Python\P1> & C:/Users/FARHAN/
 VESIT/Practicals/Python/P1/Q4.py"
The solutions are (-3+0j) and (-2+0j)
```

5. **To use Bitwise operators**

Source Code:

```
a = 10
b = 7

and_result = a & b
print("a & b =", and_result)

or_result = a | b
print("a | b =", or_result)

xor_result = a ^ b
print("a ^ b =", xor_result)

not_result = ~a
print("~a =", not_result)

left_shift = a << 1
print("a << 1 =", left_shift)

right_shift = a >> 1
print("a >> 1 =", right_shift)
```

Output:



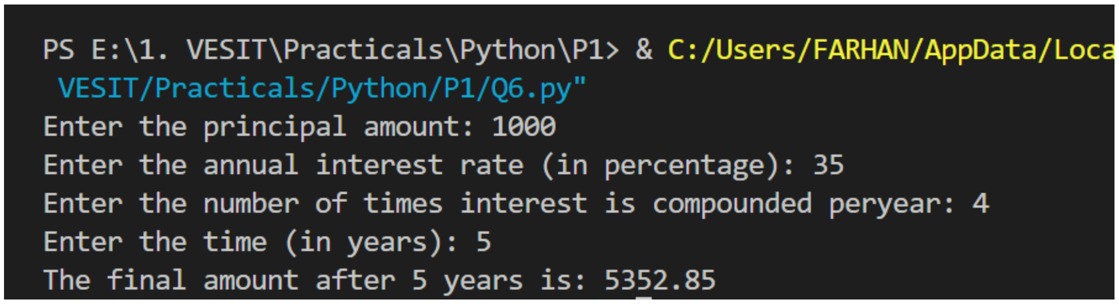
```
PS E:\1. VESIT\Practicals\Python\P1> & C:/Users/FARHAN/
 VESIT/Practicals/Python/P1/Q5.py"
a & b = 2
a | b = 15
a ^ b = 13
~a = -11
a << 1 = 20
a >> 1 = 5
```

6. **To compute compound interest given all the required values.**

Source Code:

```
P = float(input("Enter the principal amount: "))
r = float(input("Enter the annual interest rate (in percentage): ")) /
100

n = int(input("Enter the number of times interest is compounded per
year: "))

t = int(input("Enter the time (in years): "))
A = P * (1 + r/n)**(n*t)

print(f"The final amount after {t} years is: {A:.2f}")
```

Output:



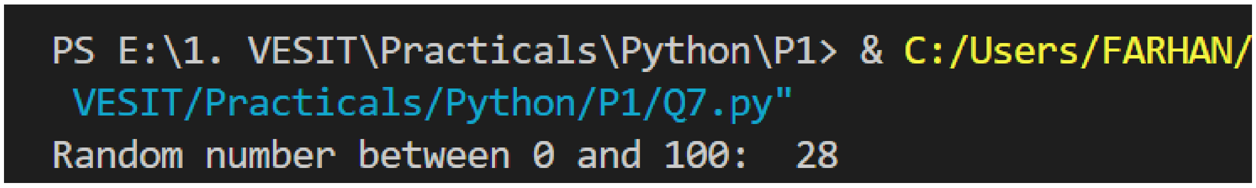
```
PS E:\1. VESIT\Practicals\Python\P1> & C:/Users/FARHAN/AppData/Loca
 VESIT/Practicals/Python/P1/Q6.py"
Enter the principal amount: 1000
Enter the annual interest rate (in percentage): 35
Enter the number of times interest is compounded peryear: 4
Enter the time (in years): 5
The final amount after 5 years is: 5352.85
```

7. **To generate a random number between 0 and 100**

Source Code:

```
import random
random_number = random.randint(0, 100)
print("Random number between 0 and 100: ",random_number)
```

Output:



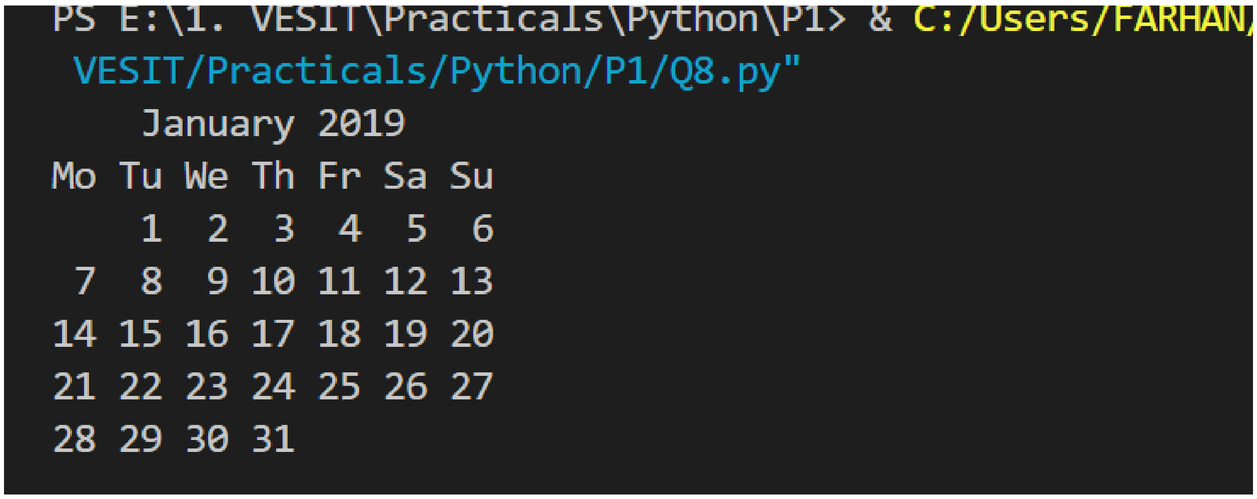
```
PS E:\1. VESIT\Practicals\Python\P1> & C:/Users/FARHAN/
 VESIT/Practicals/Python/P1/Q7.py"
Random number between 0 and 100:  28
```

8. **To display calendar for the January 2019**

Source Code:

```
import calendar
print(calendar.month(2019, 1))
```

Output:



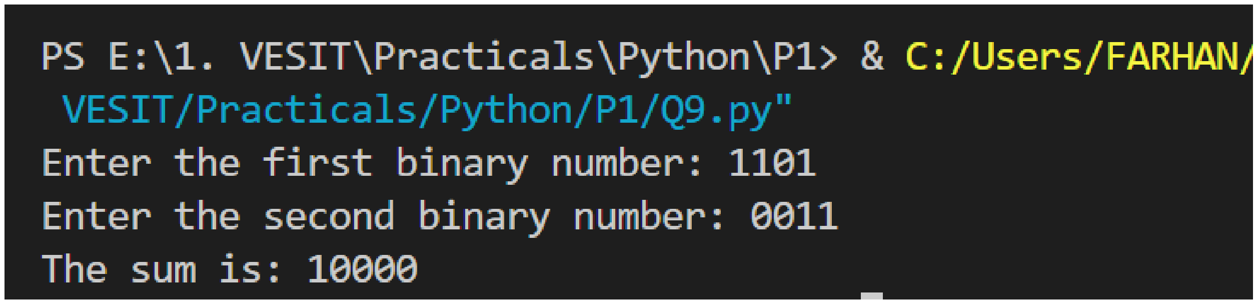
```
PS E:\1. VESIT\Practicals\Python\P1> & C:/Users/FARHAN/
 VESIT/Practicals/Python/P1/Q8.py"
    January 2019
Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

9. To add two binary numbers.

Source Code:

```
b1 = int(input("Enter the first binary number: "), 2)
b2 = int(input("Enter the second binary number: "), 2)

while b2:
        carry = b1 & b2
        b1 = b1 ^ b2
        b2 = carry << 1

print("The sum is:", bin(b1)[2:])
```

Output:



```
PS E:\1. VESIT\Practicals\Python\P1> & C:/Users/FARHAN/
 VESIT/Practicals/Python/P1/Q9.py"
Enter the first binary number: 1101
Enter the second binary number: 0011
The sum is: 10000
```

Conclusion:

From this practical i have learned to write,test and debug basic python programs.

Name of Student: Shaikh Farhan Ahmed			
Roll Number: 50		**LAB Assignment Number: 2**	
Title of LAB Assignment: To Implement Python Programs with conditional Loops.			
DOP: 08 – 10 – 2024		**DOS: 15 – 10 – 2024**	
CO Mapped: CO1	**PO Mapped: PO3, PO5, PSO1, PSO2**	**Faculty Signature:**	**Marks:**

Practical: 2

Aim: To Implement Python Programs with conditional Loops.

Description:

Conditional statements:

Conditional statements are an essential part of programming in Python. They allow you to make decisions based on the values of variables or the result of comparisons.

1. (if statement) allows you to execute a block of code if a certain condition is true.

Here is the basic syntax:

```
if condition:
    # code to execute if condition is true
```

The condition can be any expression that evaluates to a Boolean value (True or False). If the condition is True, the code block indented below the if statement will be executed. If the condition is False, the code block will be skipped.

Here is an example of how to use an if statement to check if a number is positive:

```
num = 10

if num > 0:

    print("The number is positive.")
```

The above code satisfies the condition and The number is positive will be printed.

2. (else statement) allows you to execute a different block of code if the if condition is False.

Here is the basic syntax:

```
if condition:
    # code to execute if condition is true
else:
    # code to execute if condition is false
```

If the condition is True, the code block indented below the if statement will be executed, and the code block indented below the else statement will be skipped.

If the condition is False, the code block indented below the else statement will be executed, and the code block indented below the if statement will be skipped.

Here is an example of how to use an if-else statement to check if a number is positive or negative:

```
num = -5

if num > 0:
    print("The number is positive.")
else:
    print("The number is negative."
```

In this example, we use an if-else statement to check if num is greater than 0. If it is, the message "The number is positive." is printed. If it is not (that is, num is negative or zero), the message "The number is negative." is printed.

3. (elif statement) allows you to check multiple conditions in sequence, and execute different code blocks depending on which condition is true.

Here is the basic syntax:

```
if condition1:
    # code to execute if condition1 is true
elif condition2:
    # code to execute if condition1 is false and condition2 is true
elif condition3:
    # code to execute if condition1 and condition2 are false, and
condition3 is true
else:
    # code to execute if all conditions are false
```

The elif statement is short for "else if", and can be used multiple times to check additional conditions.

Here is an example:

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

In this example a is equal to b, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

Conditional statements (if, else, and elif) are fundamental programming constructs that allow you to control the flow of your program based on conditions that you specify. They provide a way to make decisions in your program and execute different code based on those decisions.

Loops in python:

1. **for Loop:**
   The loop is used for iterating over a sequence of iterable. It allows us to perform a set of operations for each element in the sequence

   Example:
   ```
   for i in range(5):
           print(i)
   ```

   The above code will print numbers from 0 to 4

2. **while Loop:**
   The while loop repeats a block of code as long as the condition is true. It is useful when the number of iterations is known beforehand.

   Example:
   ```
   i = 0
   while i < 5:
           print(i)
           i = i + 1
   ```

List Comprehension in Python
List comprehensions are a concise way to create lists based on existing lists or other iterable objects. They use a for loop and an optional conditional statement to generate the new list.

Here is an example of how to use list comprehension to create a new list of even numbers from an existing list:
```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = [num for num in numbers if num % 2 == 0]
print(even_numbers)
```

The above code will output in: [2, 4, 6, 8, 10]

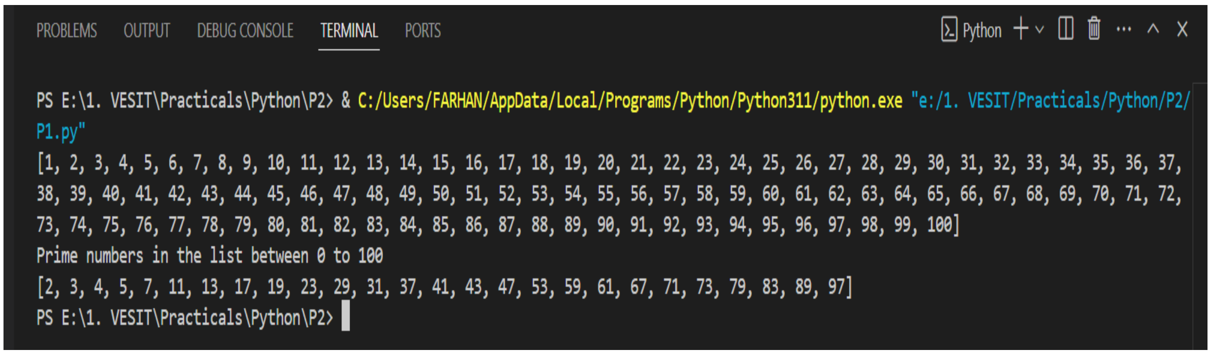
Programs:

1. To find all the prime numbers in the interval 0 to 100.

Source Code:

```
def isPrime(number):
    if number < 2:
        return False
    for i in range(2,int(number / 2)):
        if number % i == 0:
            return False
    return True

nums = [i for i in range(1, 101)]
print(nums)
prime_nums = [i for i in nums if isPrime(i)]

print("Prime numbers in the list between 0 to 100")
print(prime_nums)
```

Output:



```
PS E:\1. VESIT\Practicals\Python\P2> & C:/Users/FARHAN/AppData/Local/Programs/Python/Python311/python.exe "e:/1. VESIT/Practicals/Python/P2/
P1.py"
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72,
73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
Prime numbers in the list between 0 to 100
[2, 3, 4, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
PS E:\1. VESIT\Practicals\Python\P2>
```

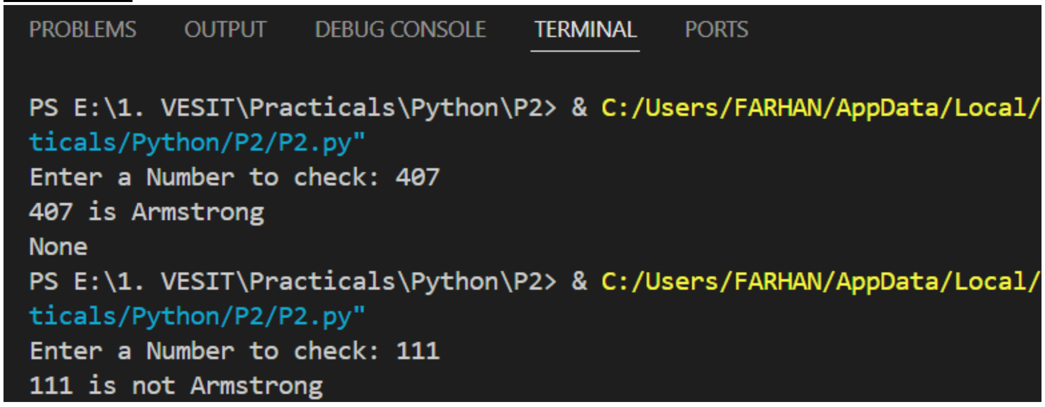
2. To check if the given number is Armstrong or not.

Source Code:

```
def isArmstrong(number):
    sum = 0
    temp = number
    while temp > 0:
        last_digit = temp % 10
        sum = sum + last_digit ** 3
        temp = temp//10
    if number == sum:
        print(number,"is Armstrong")
    else:
        print(number,"is not Armstrong")
```

```
number = int(input("Enter a Number to check: "))

print(isArmstrong(number))
```

Output:



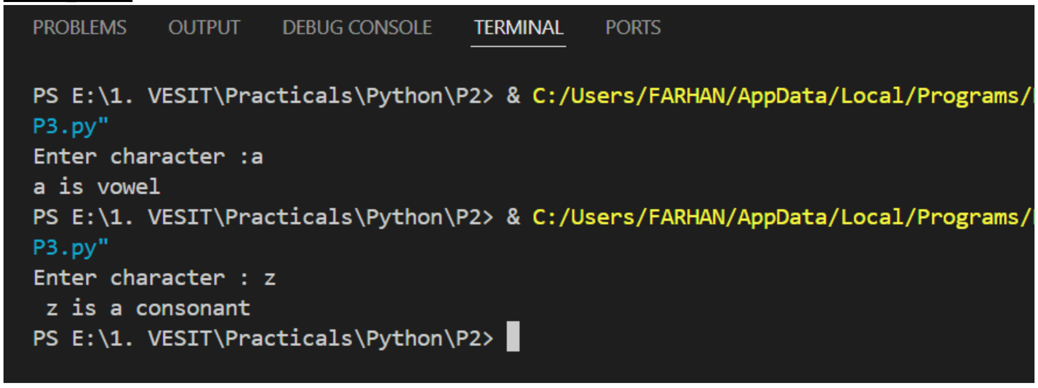
```
PS E:\1. VESIT\Practicals\Python\P2> & C:/Users/FARHAN/AppData/Local/
ticals/Python/P2/P2.py"
Enter a Number to check: 407
407 is Armstrong
None
PS E:\1. VESIT\Practicals\Python\P2> & C:/Users/FARHAN/AppData/Local/
ticals/Python/P2/P2.py"
Enter a Number to check: 111
111 is not Armstrong
```

3. To check if the given char is vowel or consonant.

Source Code:

```
input = input("Enter character :")

vowels = ['a','e','i','o','u']

if input.lower() in vowels:
    print(input, "is vowel")
else:
    print(input,"is a consonant")
```

Output:



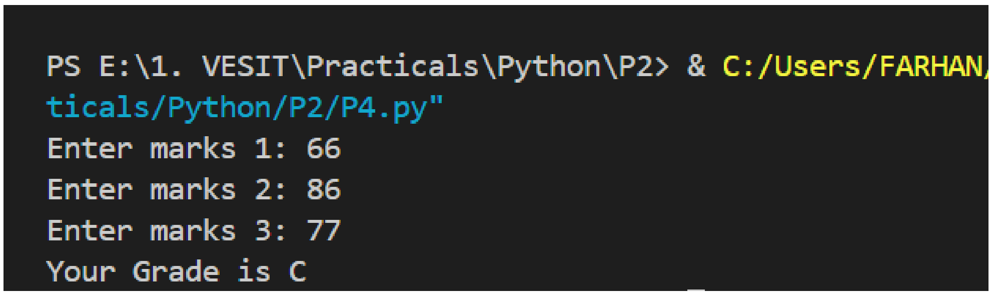
```
PS E:\1. VESIT\Practicals\Python\P2> & C:/Users/FARHAN/AppData/Local/Programs/
P3.py"
Enter character :a
a is vowel
PS E:\1. VESIT\Practicals\Python\P2> & C:/Users/FARHAN/AppData/Local/Programs/
P3.py"
Enter character : z
 z is a consonant
PS E:\1. VESIT\Practicals\Python\P2>
```

4. **Write a program to take in the marks of 3 subjects and display the grade.**

Source Code:

```
def calculateGrade(marks1,marks2,marks3):
    total = marks1 + marks2 + marks3
    average = total / 3
    if average >= 90:
        print("Your Grade is A")
    elif 80 <= average < 90:
        print("Your Grade is B")
    elif 70 <= average < 80:
        print("Your Grade is C")
    elif 60 <= average < 70:
        print("Your Grade is D")
    else:
        print("Your Grade is F")

marks1 = int(input("Enter marks 1: "))
marks2 = int(input("Enter marks 2: "))
marks3 = int(input("Enter marks
3:"))

calculateGrade(marks1,marks2,marks3)
```

Output:



```
PS E:\1. VESIT\Practicals\Python\P2> & C:/Users/FARHAN
ticals/Python/P2/P4.py"
Enter marks 1: 66
Enter marks 2: 86
Enter marks 3: 77
Your Grade is C
```

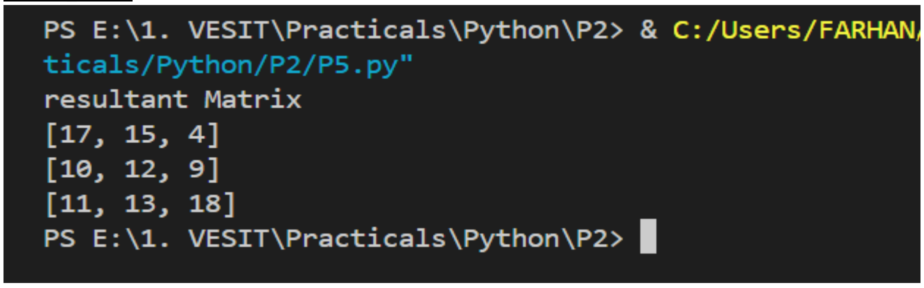
5. **Write a program to add two matrices.**

Source Code:

```
X = [[12,7,3],[4 ,5,6],[7 ,8,9]]

Y = [[5,8,1],[6,7,3],[4,5,9]]

result = [[X[i][j] + Y[i][j]  for j in range(len(X[0]))] for i
in range(len(X))]
```

```
print("resultant Matrix")
for r in result:
   print(r)
```

Output:



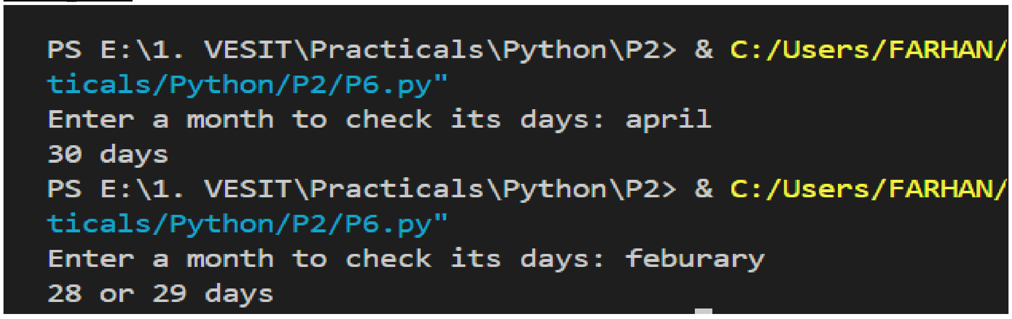
```
PS E:\1. VESIT\Practicals\Python\P2> & C:/Users/FARHAN/
ticals/Python/P2/P5.py"
resultant Matrix
[17, 15, 4]
[10, 12, 9]
[11, 13, 18]
PS E:\1. VESIT\Practicals\Python\P2>
```

6. To convert month name to number of days.

Source Code:

```
def month_to_days(month):
    input = month.lower()
    _30Days = ["april", "june", "september", "november"]
    _31Days = ["january", "march", "may", "july", "august",
"october", "december"]
    if(input == 'feburary'):
        print("28 or 29 days")
    elif(input in _30Days):
        print("30 days")
    elif(input in _31Days):
        print("31 days")
    else:
        print("Invalid Month Name")
month = input("Enter a month to check its days: ")
month_to_days(month)
```

Output:



```
PS E:\1. VESIT\Practicals\Python\P2> & C:/Users/FARHAN/
ticals/Python/P2/P6.py"
Enter a month to check its days: april
30 days
PS E:\1. VESIT\Practicals\Python\P2> & C:/Users/FARHAN/
ticals/Python/P2/P6.py"
Enter a month to check its days: feburary
28 or 29 days
```

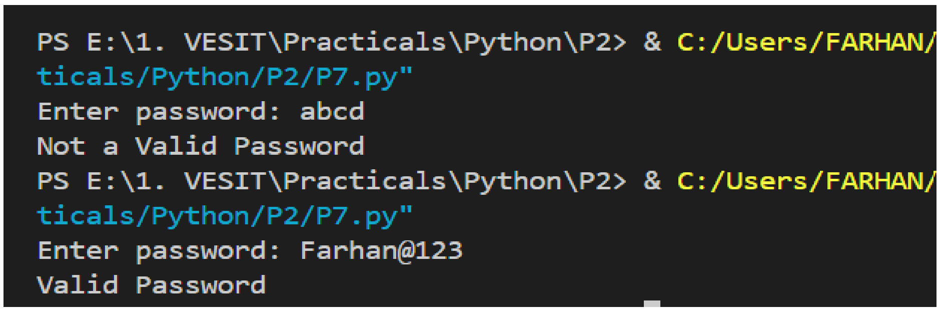
7. **To check the validity of a password input by user.**
   Rules of validation
   - At least 1 letter between [a-z] and 1 letter between [A-Z]
   - At least 1 number between [0-9]
   - At least 1 character from [$#@]
   - Minimum length 6 and Maximum length 16 characters.

Source Code:

```
import re
password = input("Enter password: ")
flag = 0
input = password
while True:
      if (len(input) < 6 or len(input) > 16):
      flag = -1
      break
      elif not re.search("[a-z]", input):
      flag = -1
      break
      elif not re.search("[A-Z]", input):
      flag = -1
      break
      elif not re.search("[0-9]", input):
      flag = -1
      break
      elif not re.search("[#@$]" , input):
      flag = -1
      break
      else:
      flag = 0
      print("Valid Password")
      break
```

```
    if flag == -1:
            print("Not a Valid Password ")
```

Output:



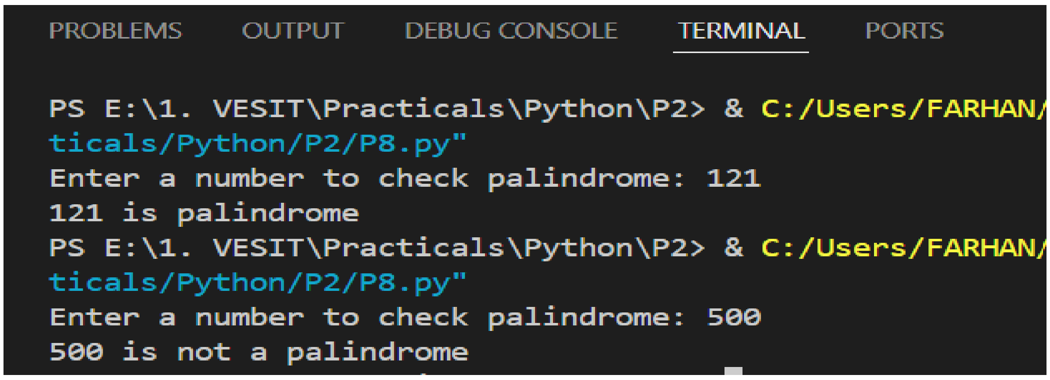
```
PS E:\1. VESIT\Practicals\Python\P2> & C:/Users/FARHAN/
ticals/Python/P2/P7.py"
Enter password: abcd
Not a Valid Password
PS E:\1. VESIT\Practicals\Python\P2> & C:/Users/FARHAN/
ticals/Python/P2/P7.py"
Enter password: Farhan@123
Valid Password
```

8. To check whether the number is palindrome or not.

Source Code:

```
def isPalindrome(number):
    rev = 0
    temp = number
    while temp != 0:
        lastdigit = temp % 10
        rev = rev * 10 + lastdigit
        temp = temp // 10
    if(number == rev):
        print(number, "is palindrome")
    else:
          print(number, "is not a palindrome")

number = int(input("Enter a number to check palindrome: "))

isPalindrome(number)
```

Output:



```
PS E:\1. VESIT\Practicals\Python\P2> & C:/Users/FARHAN/
ticals/Python/P2/P8.py"
Enter a number to check palindrome: 121
121 is palindrome
PS E:\1. VESIT\Practicals\Python\P2> & C:/Users/FARHAN/
ticals/Python/P2/P8.py"
Enter a number to check palindrome: 500
500 is not a palindrome
```

Conclusion:

The Python programs that I have executed offered a wide variety of uses, ranging from user input validation to mathematical calculations. They highlight the flexibility of Python in addressing a range of programming problems by demonstrating the efficient use of conditionals, loops, functions, and data structures. These courses offer a strong basis for comprehending and applying fundamental programming ideas.

Name of Student: Shaikh Farhan Ahmed			
Roll Number: 50		**LAB Assignment Number: 03**	
Title of LAB Assignment: To implement Python programs using List, String, Set and Dictionary			
DOP: 12 – 10 – 2024		**DOS: 16 – 10 – 2024**	
CO Mapped: CO4	**PO Mapped: PO3, PO5, PSO1, PSO2**	**Faculty Signature:**	**Marks:**

Practical No 3

Aim: To implement Python programs using List, String, Set and Dictionary

Theory:

1. List:
In Python, a list is a built-in data structure that is used to store an ordered collection of items. Lists are mutable, meaning that their contents can be changed after the list has been created. They can hold a various of data types, including integers, floats, strings, and even other lists.

Example

```
a = [1, 'apple', 3.14, [5, 6]]

print(a)
```

2. Strings:

A String represents a sequence of characters. It is an immutable data type in most of the programming languages including Java and Python, meaning that once you have created a string, you cannot change it. Python Programming does not have a character data type, a single character is simply a string with a length of 1.

Example:

```
Mystring = “Farhan”

print(Mystring)
```

3. Sets:

A Set in Python programming is an unordered collection data type that is iterable and has no duplicate elements. While sets are mutable, meaning you can add or remove elements after their creation, the individual elements within the set must be immutable and cannot be changed directly.

The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set

Example:

```
cars = {“Bmw”,”ferrari”,”McLaren”}

print(cars)
```

4.Dictionaries:

Dictionaries in Python is a data structure, used to store values in **key:value** format. This makes it different from lists, tuples, and arrays as in a dictionary each key has an associated value.

The dictionary holds pairs of values, one being the Key and the other corresponding pair element being its Key:value. Values in a dictionary can be of any data type and can be duplicated, whereas keys can’t be repeated and must be immutable.

Example:

```
Dict = {1: 'Farhan', 2: 'Zeeshan', 3: 'Haneef'}

print(Dict)
```

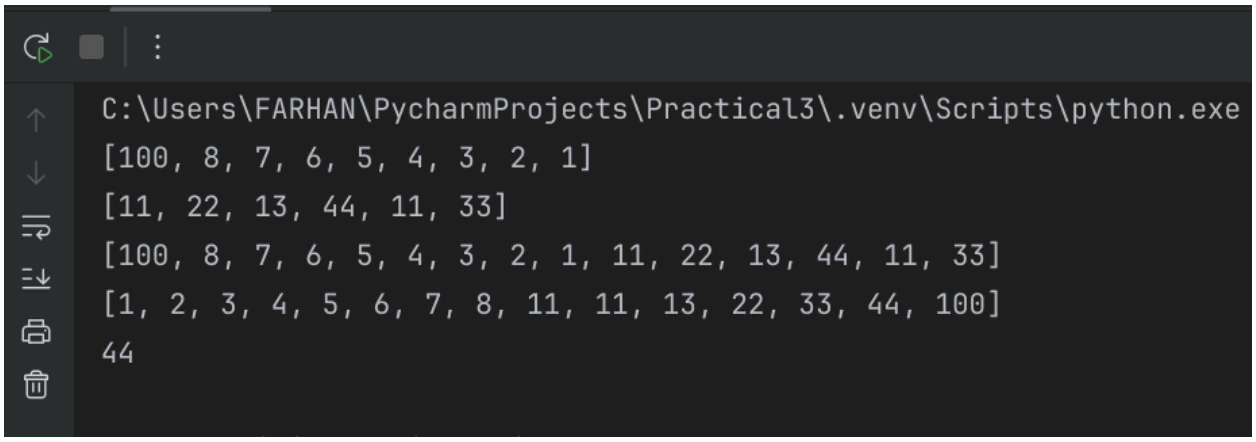
Q1. To merge two list and find second largest element in the list using bubble sort

Source Code:

```
def bubbleSort(list):
    n = len(list)
    for i in range (n):
    for j in range(i+1,n):
        if list[i] > list[j]:
            temp = list[i];
            list[i] = list[j]
            list[j] = temp
    return list


def merge_list(list1,list2):
    mergedList = list1 + list2
    return mergedList


def secondlargest(sorted_list):
    n = len(sorted_list)
    return sorted_list[n-2]


list1 = [100,8,7,6,5,4,3,2,1]
list2 = [11,22,13,44,11,33]
print(list1)
print(list2)
```

```
merged = merge_list(list1,list2)

print(merged)

sorted_list = bubbleSort(merged)

print(sorted_list)

print(secondlargest(sorted_list))
```

Output:



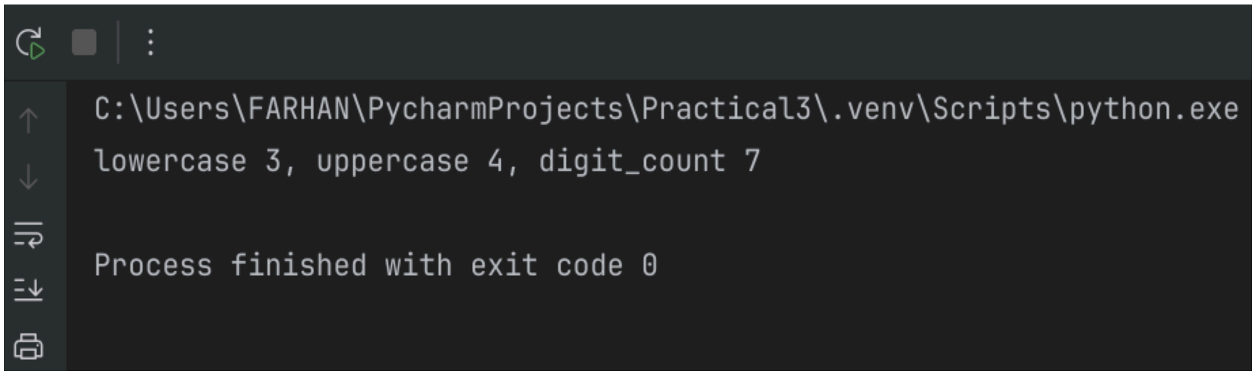
```
C:\Users\FARHAN\PycharmProjects\Practical3\.venv\Scripts\python.exe
[100, 8, 7, 6, 5, 4, 3, 2, 1]
[11, 22, 13, 44, 11, 33]
[100, 8, 7, 6, 5, 4, 3, 2, 1, 11, 22, 13, 44, 11, 33]
[1, 2, 3, 4, 5, 6, 7, 8, 11, 11, 13, 22, 33, 44, 100]
44
```

Q2. To calculate the no of uppercase, lowercase letters and digits in a string.

Source Code:

```
def check_upper_lower_digit(str):

     upper_count = 0

     lower_count = 0

     digit_count = 0


     for i in str:

     digit_count = digit_count + 1

     if i in str.lower():

          lower_count = lower_count + 1

     if i in str.upper():

          upper_count = upper_count + 1

     print("lowercase{},uppercase{},digit_count{}"

    .format(lower_count, upper_count, digit_count))


check_upper_lower_digit("aaaBBBB")
```

Output:



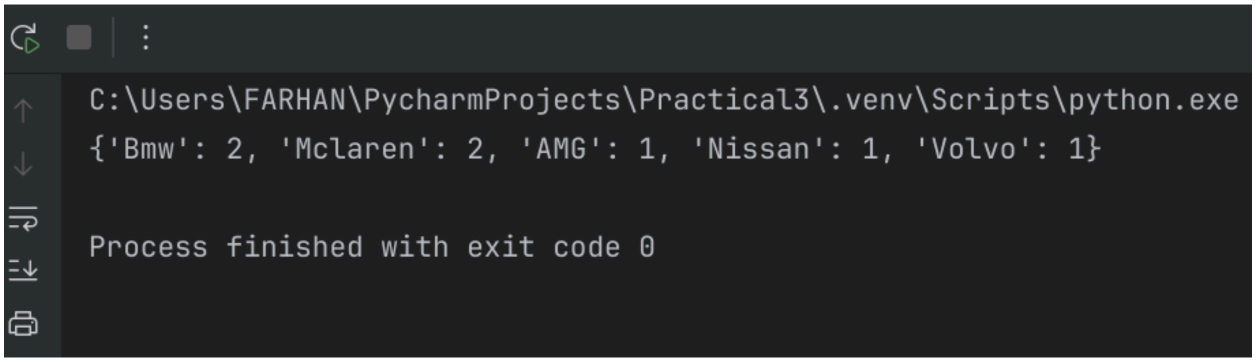
```
C:\Users\FARHAN\PycharmProjects\Practical3\.venv\Scripts\python.exe
lowercase 3, uppercase 4, digit_count 7

Process finished with exit code 0
```

Q3. To count the occurrences of each word in a given string sentence

Source Code:

```
def word_count(str):
     counts = dict()
     words = str.split()

     for word in words:
     if word in counts:
          counts[word] += 1
     else:
          counts[word] = 1
     return counts
print(word_count('Bmw Mclaren AMG Nissan Bmw Mclaren Volvo '))
```

Output:



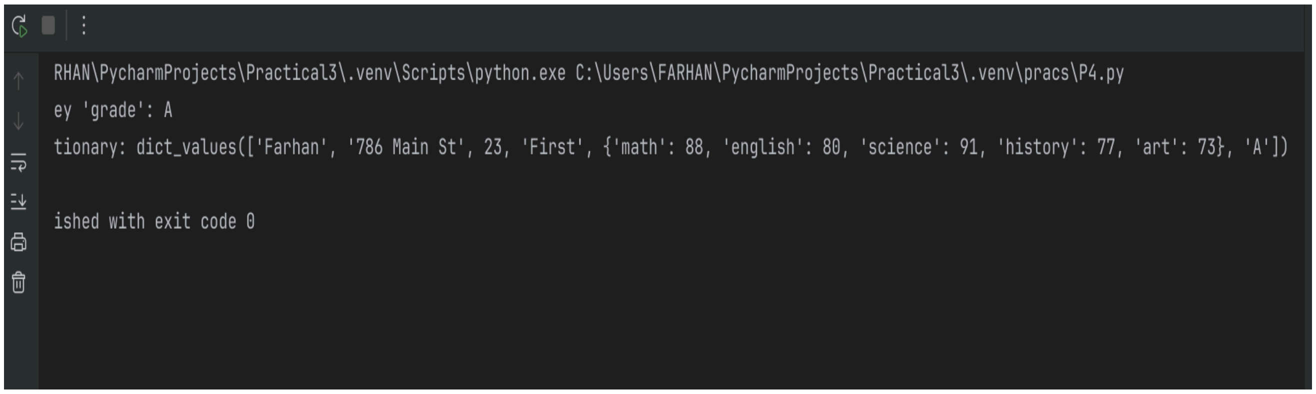
```
C:\Users\FARHAN\PycharmProjects\Practical3\.venv\Scripts\python.exe
{'Bmw': 2, 'Mclaren': 2, 'AMG': 1, 'Nissan': 1, 'Volvo': 1}

Process finished with exit code 0
```

Q4. To add key value pair to the dictionary and search and then delete the given key from the dictionary.

Source Code:

```
def manipulate_dictionary(dictionary, key, value):
    dictionary[key] = value

    if key in dictionary:
    print(f"Value for key '{key}': {dictionary[key]}")
    else:
    print(f"Key '{key}' not found in the dictionary.")

    if key in dictionary:
        del dictionary[key]
        print(f"Key '{key}' deleted from the dictionary.")
    else:
        print(f"Key '{key}' not found for deletion.")
    return dictionary
#Testing
student_info = {'name': 'Farhan', 'address': '786 Main St',
'age': 23, 'class': 'First', 'marks': {'math': 88, 'english':
80, 'science': 91, 'history': 77, 'art': 73}}
new_key = 'grade'
new_value = 'A'
updated_student_info=manipulate_dictionary(student_info,
new_key, new_value)
print("Updated Dictionary:", updated_student_info.values())
```

Output:



```
RHAN\PycharmProjects\Practical3\.venv\Scripts\python.exe C:\Users\FARHAN\PycharmProjects\Practical3\.venv\pracs\P4.py
ey 'grade': A
tionary: dict_values(['Farhan', '786 Main St', 23, 'First', {'math': 88, 'english': 80, 'science': 91, 'history': 77, 'art': 73}, 'A'])

ished with exit code 0
```

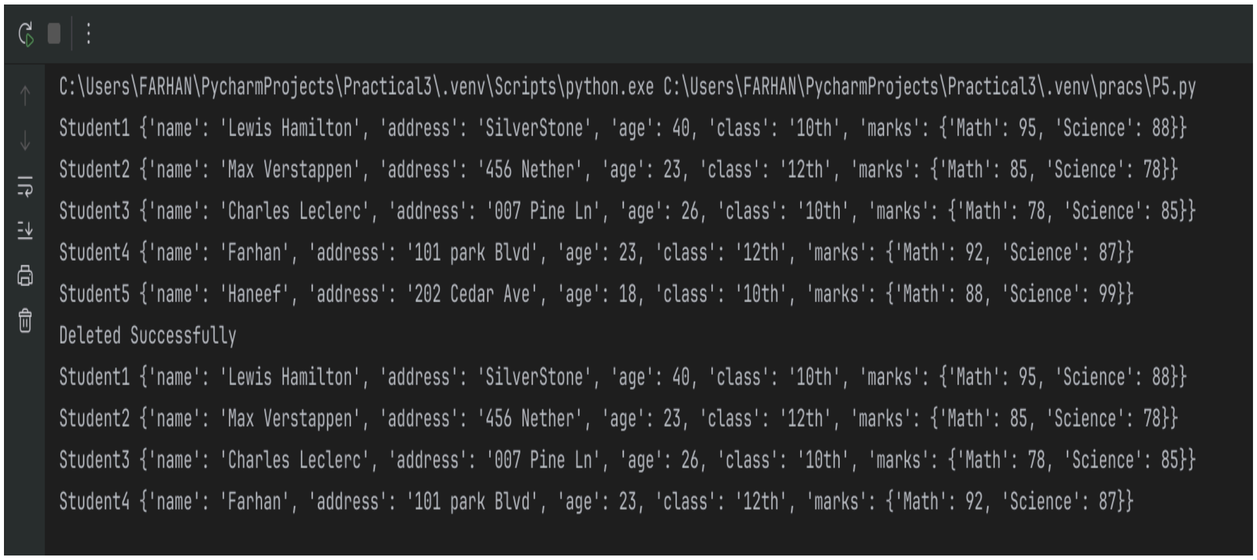
Q5. Create one dictionary of 5 students with their name, address, age, class and marks of 5 subjects. Perform all the operations on the created dictionary.

Source Code:

```
student_dict = {
     "Student1": {
     "name": "Lewis Hamilton",
     "address": "SilverStone",
     "age": 40,
     "class": "10th",
     "marks": {
          "Math": 95,
          "Science": 88,
     }
     },
     "Student2": {
     "name": "Max Verstappen",
     "address": "456 Nether",
     "age": 23,
     "class": "12th",
     "marks": {
          "Math": 85,
          "Science": 78,
     }
     },
     "Student3": {
     "name": "Charles Leclerc",
     "address": "007 Pine Ln",
     "age": 26,
     "class": "10th",
     "marks": {
          "Math": 78,
          "Science": 85,
     }
     },
     "Student4": {
     "name": "Farhan",
     "address": "101 park Blvd",
     	"age": 23,
     	"class": "12th",
     	"marks": {
     		"Math": 92,
     		"Science": 87
     	}
     	},
     	"Student5": {
     	"name": "Haneef",
     	"address": "202 Cedar Ave",
     	"age": 18,
     	"class": "10th",
     	"marks": {
     		"Math": 88,
     		"Science": 99,
     	}
     	}
     }

     def deleteStudent(dict,key):
     	value = dict.pop(key,None)
     	print("Deleted Successfully")
```

```
def iterate(dict):

    for key, value in dict.items():

    print(key,value)


iterate(student_dict)

deleteStudent(student_dict,"Student5")

iterate(student_dict)
```

Output:



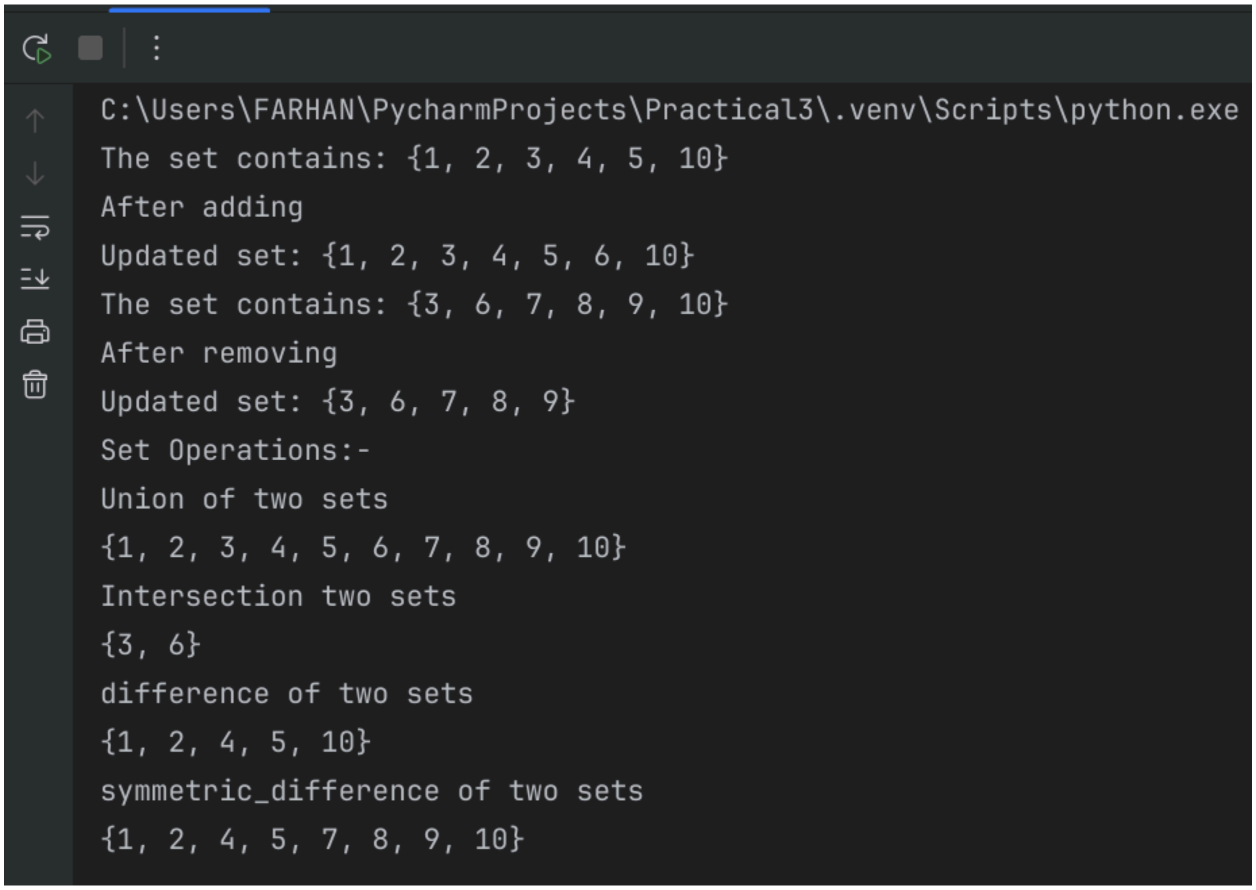
```
C:\Users\FARHAN\PycharmProjects\Practical3\.venv\Scripts\python.exe C:\Users\FARHAN\PycharmProjects\Practical3\.venv\pracs\P5.py
Student1 {'name': 'Lewis Hamilton', 'address': 'SilverStone', 'age': 40, 'class': '10th', 'marks': {'Math': 95, 'Science': 88}}
Student2 {'name': 'Max Verstappen', 'address': '456 Nether', 'age': 23, 'class': '12th', 'marks': {'Math': 85, 'Science': 78}}
Student3 {'name': 'Charles Leclerc', 'address': '007 Pine Ln', 'age': 26, 'class': '10th', 'marks': {'Math': 78, 'Science': 85}}
Student4 {'name': 'Farhan', 'address': '101 park Blvd', 'age': 23, 'class': '12th', 'marks': {'Math': 92, 'Science': 87}}
Student5 {'name': 'Haneef', 'address': '202 Cedar Ave', 'age': 18, 'class': '10th', 'marks': {'Math': 88, 'Science': 99}}
Deleted Successfully
Student1 {'name': 'Lewis Hamilton', 'address': 'SilverStone', 'age': 40, 'class': '10th', 'marks': {'Math': 95, 'Science': 88}}
Student2 {'name': 'Max Verstappen', 'address': '456 Nether', 'age': 23, 'class': '12th', 'marks': {'Math': 85, 'Science': 78}}
Student3 {'name': 'Charles Leclerc', 'address': '007 Pine Ln', 'age': 26, 'class': '10th', 'marks': {'Math': 78, 'Science': 85}}
Student4 {'name': 'Farhan', 'address': '101 park Blvd', 'age': 23, 'class': '12th', 'marks': {'Math': 92, 'Science': 87}}
```

Q6. To add and remove elements from set and perform all the set operations like Union, Intersection, Difference and Symmetric Difference

Source Code:

```
def addElement(set1):
    print(f"The set contains: {set1}")
    set1.add(6)
    print("After adding")
    print(f"Updated set: {set1}")
def removeElement(set):
    print(f"The set contains: {set}")
    set.remove(10)
    print("After removing")
    print(f"Updated set: {set}")
def setOperations(set1,set2):
    print("Union of two sets")
    print(set1.union(set2))
    print("Intersection two sets")
    print(set1.intersection(set2))
    print("difference of two sets")
    print(set1.difference(set2))
    print("symmetric_difference of two sets")
    print(set1.symmetric_difference(set2)
set1 = {1,2,3,4,5,10}

set2 = {6,3,7,8,9,10}


addElement(set1)

removeElement(set2)

print("Set Operations:-")

setOperations(set1,set2)
```

Output:



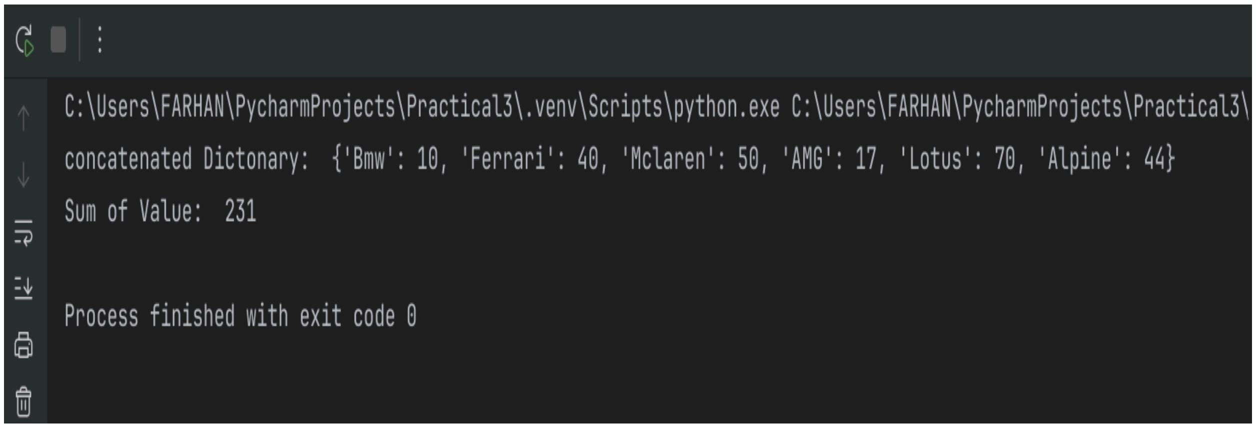
```
C:\Users\FARHAN\PycharmProjects\Practical3\.venv\Scripts\python.exe
The set contains: {1, 2, 3, 4, 5, 10}
After adding
Updated set: {1, 2, 3, 4, 5, 6, 10}
The set contains: {3, 6, 7, 8, 9, 10}
After removing
Updated set: {3, 6, 7, 8, 9}
Set Operations:-
Union of two sets
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Intersection two sets
{3, 6}
difference of two sets
{1, 2, 4, 5, 10}
symmetric_difference of two sets
{1, 2, 4, 5, 7, 8, 9, 10}
```

Q7. To concatenate two dictionaries and find sum of all values in dictionary

Source Code:

```
def concatenate_and_sum(dict1, dict2):
     concatenated_dict = {**dict1, **dict2}
     total_sum = sum(concatenated_dict.values())
     print("concatenated Dictonary: ", concatenated_dict)
     print("Sum of Value: ",total_sum)


dict1 = {'Bmw':10,'Ferrari':40,'Mclaren':50}
dict2 = {'AMG':17,'Lotus':70,'Alpine':44}
concatenate_and_sum(dict1,dict2)
```

Ouput:



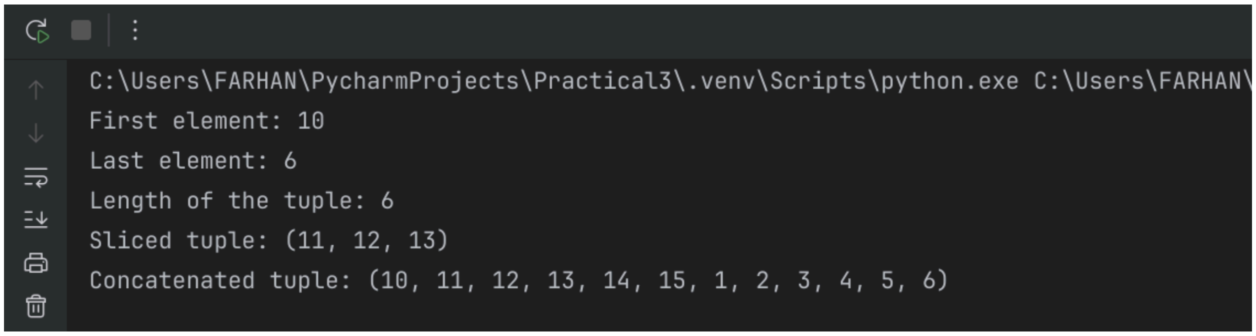
```
C:\Users\FARHAN\PycharmProjects\Practical3\.venv\Scripts\python.exe C:\Users\FARHAN\PycharmProjects\Practical3\
concatenated Dictonary:  {'Bmw': 10, 'Ferrari': 40, 'Mclaren': 50, 'AMG': 17, 'Lotus': 70, 'Alpine': 44}
Sum of Value:  231

Process finished with exit code 0
```

Q8. Perform different operations on Tuple.

Source Code:

```
def tuple_operations(tuple1,tuple2):
    first_element = tuple1[0]
    last_element = tuple2[-1]
    print("First element:", first_element)
    print("Last element:", last_element)
    length = len(tuple1)
    print("Length of the tuple:", length)
    sliced = tuple1[1:4]
    print("Sliced tuple:", sliced)
    concatenated_tuple = tuple1 + tuple2
    print("Concatenated tuple:", concatenated_tuple)
tuple1 = (10,11,12,13,14,15)
tuple2 = (1,2,3,4,5,6)
tuple_operations(tuple1,tuple2)
```

Output:



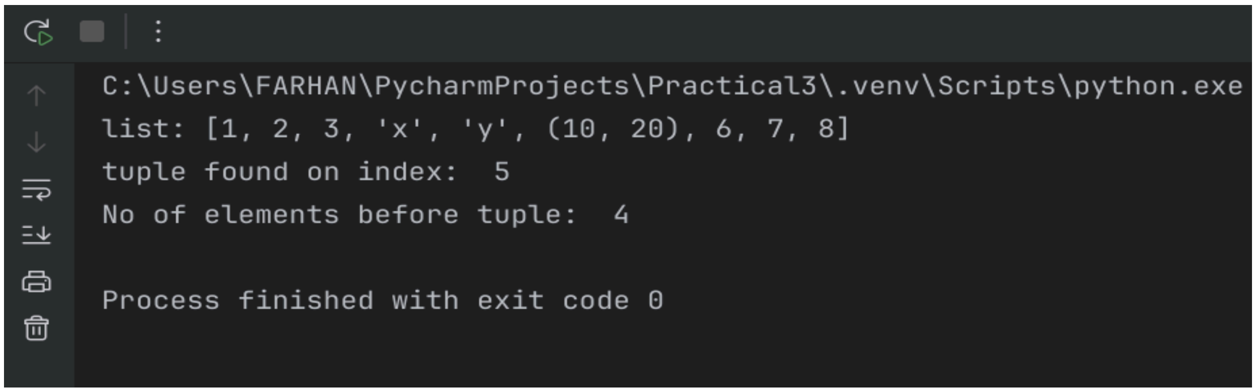
```
C:\Users\FARHAN\PycharmProjects\Practical3\.venv\Scripts\python.exe C:\Users\FARHAN\
First element: 10
Last element: 6
Length of the tuple: 6
Sliced tuple: (11, 12, 13)
Concatenated tuple: (10, 11, 12, 13, 14, 15, 1, 2, 3, 4, 5, 6)
```

Q9. Write a Python program to count the elements in a list until an element is a tuple.

Source Code:

```
def count_until_tuple(list):

     count = 0

     for i in list:

     if isinstance(i,tuple):

          break

     else:

          count = count + 1

     return count

list = [1, 2, 3, 'x', 'y', (10, 20), 6, 7, 8]

print("list:",list)

result = count_until_tuple(list)

print("tuple found on index: ", result)

print("No of elements before tuple: ",result - 1)
```

Output:



```
C:\Users\FARHAN\PycharmProjects\Practical3\.venv\Scripts\python.exe
list: [1, 2, 3, 'x', 'y', (10, 20), 6, 7, 8]
tuple found on index:  5
No of elements before tuple:  4

Process finished with exit code 0
```

Conclusion:

These Python programs demonstrate how fundamental data structures and operations are used in real-world scenarios. The practical application of lists, strings, sets, dictionaries, and tuples offers important insights into their features and applications.

Name of Student: Shaikh Farhan Ahmed			
Roll Number: 50		**LAB Assignment Number: 04**	
Title of LAB Assignment: To implement programs on Python Functions and Modules			
DOP: 26 – 11 – 2024		**DOS: 28 – 11 – 2024**	
CO Mapped: CO1	**PO Mapped: PO3, PO5, PSO1, PSO2**	**Faculty Signature:**	**Marks:**

Practical No 4

AIM: To implement programs on Python Functions and Modules

Python Function is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

Some Benefits of Using Functions

- Increase Code Readability
- Increase Code Reusability

A Python module is a file containing Python definitions and statements. A module can define functions, classes, and variables. A module can also include runnable code.
Grouping related code into a module makes the code easier to understand and use. It also makes the code logically organized.

Advantages of Python modules

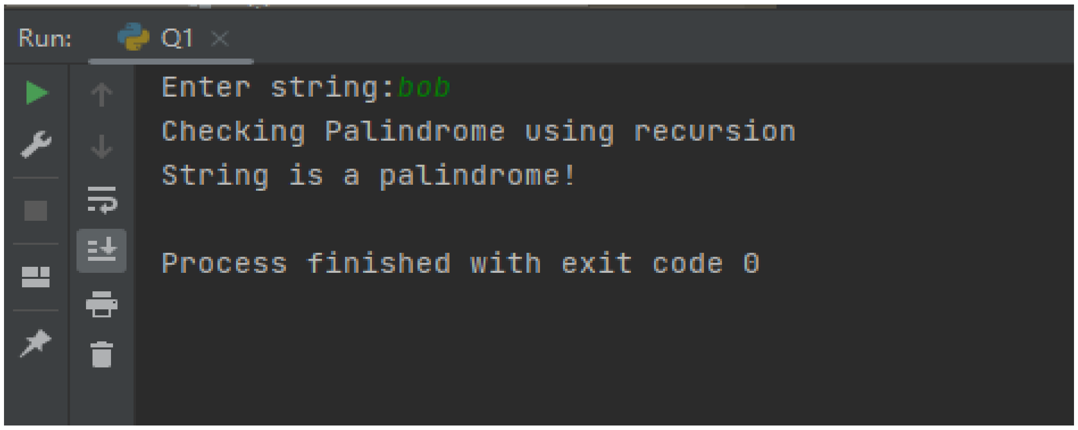
- Simplicity – Instead of focusing on the full problem, a module often concentrates on a very small section of the problem. ...
- Maintainability – Typically, modules are meant to enforce logical boundaries between distinct issue areas.

1.To check whether string is palindrome or not using function recursion

Source Code:

```
def is_palindrome(s):
    if len(s) < 1:
        return True
    else:
        if s[0] == s[-1]:
            return is_palindrome(s[1:-1])
        else:
            return False

a=str(input("Enter string:"))
print("Checking Palindrome using recursion")
if(is_palindrome(a)==True):
    print("String is a palindrome!")
else:
    print("String isn't a palindrome!")
```

Output:



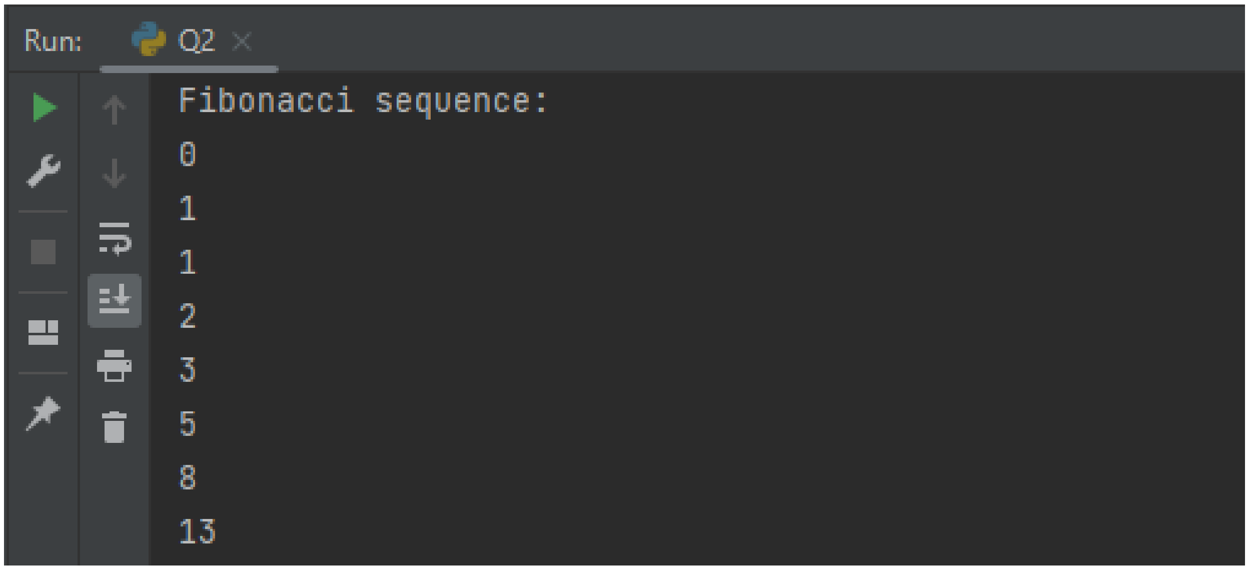
```
Enter string:bob
Checking Palindrome using recursion
String is a palindrome!

Process finished with exit code 0
```

2. To find Fibonacci series using recursion

Source Code:

```
def rec_fibo(n):
   if n <= 1:
       return n
   else:
       return(rec_fibo(n-1) + rec_fibo(n-2))

terms = 8

if terms <= 0:
   print("Please enter a positive integer")
else:
   print("Fibonacci sequence:")
   for i in range(terms):
       print(rec_fibo(i))
```

Output:



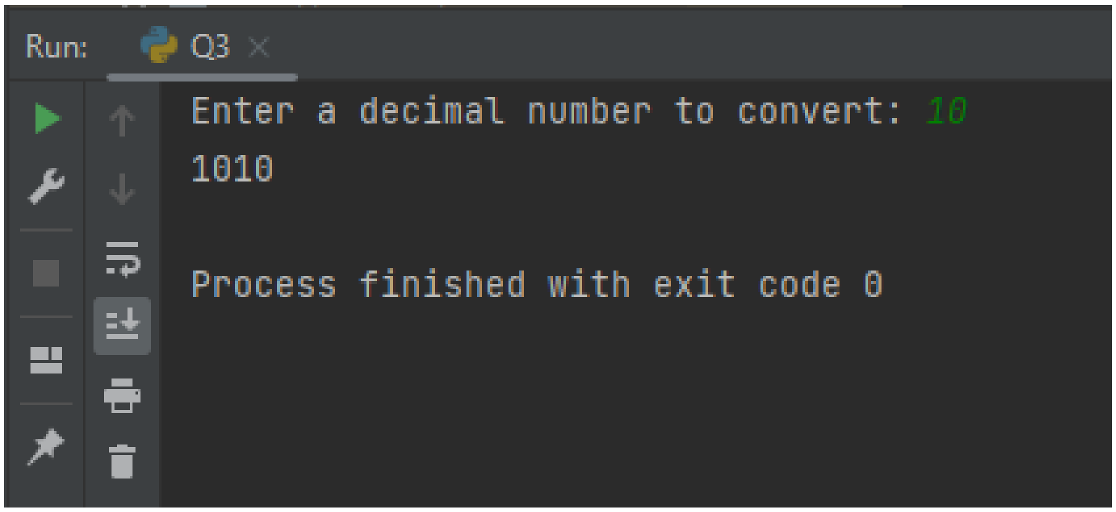
```
Fibonacci sequence:
0
1
1
2
3
5
8
13
```

3. To find binary equivalent of number using recursion

Source Code:

```
def convertToBinary(n):
   if n > 1:
       convertToBinary(n//2)
   print(n % 2,end = '')

# decimal number
input = int(input("Enter a decimal number to convert: "))

convertToBinary(input)
print()
```

Output:



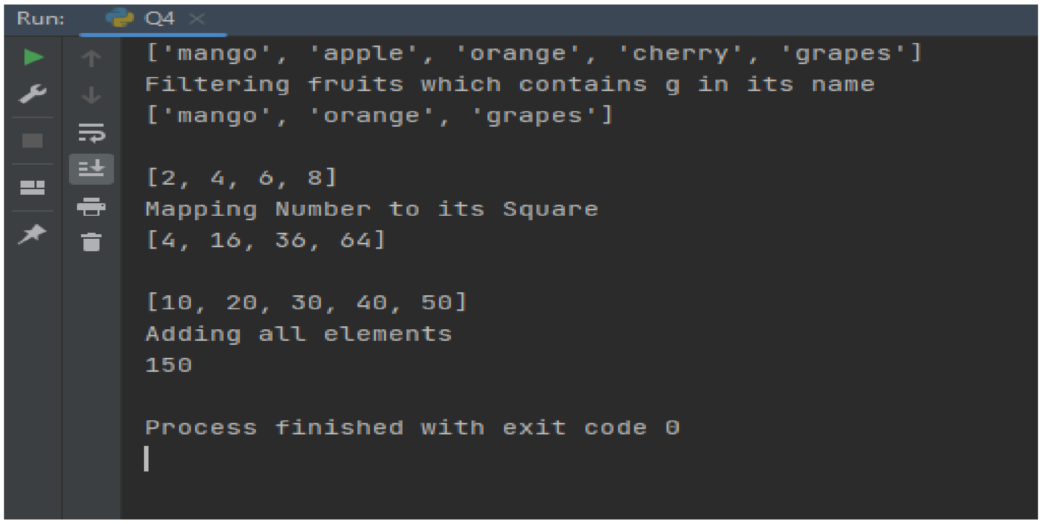
```
Enter a decimal number to convert: 10
1010

Process finished with exit code 0
```

4. To use lambda function on list to generate filtered list, mapped list and reduced list.

Source Code:

```
from functools import reduce
fruits = ['mango', 'apple', 'orange', 'cherry', 'grapes']
print(fruits)
print("Filtering fruits which contains g in its name")
print(list(filter(lambda fruit: 'g' in fruit, fruits)))
print()

arr = [2,4,6,8]
print(arr)
print("Mapping Number to its Square")
arr = list(map(lambda x: x*x, arr))
print(arr)
print()

nums = [10,20,30,40,50]
print(nums)
print("Adding all elements")
sum_of_nums = reduce(lambda x,y: x+y,nums)
print(sum_of_nums)
```

Output:



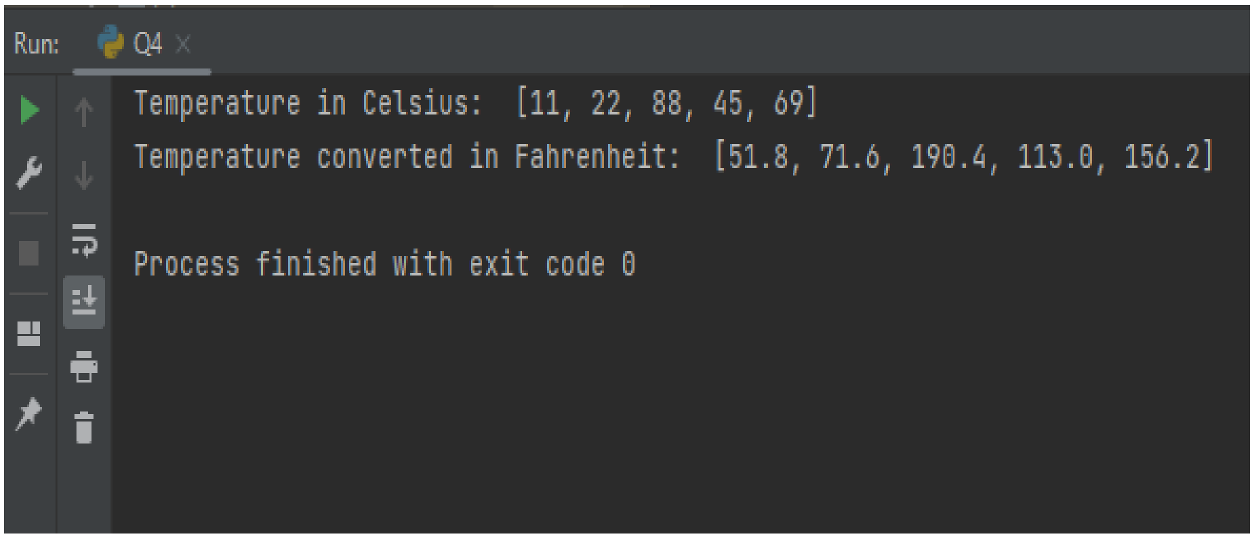
```
['mango', 'apple', 'orange', 'cherry', 'grapes']
Filtering fruits which contains g in its name
['mango', 'orange', 'grapes']

[2, 4, 6, 8]
Mapping Number to its Square
[4, 16, 36, 64]

[10, 20, 30, 40, 50]
Adding all elements
150

Process finished with exit code 0
```

5. Convert the temperature in Celsius to Fahrenheit in a list using an anonymous function.

Source Code:

```
Celsius_temps = [11,22,88,45,69]

Fahrenheit_temps = list(map(lambda celsius : (9/5) * celsius +
32, Celsius_temps))

print("Temperature in Celsius: ", Celsius_temps)
print("Temperature converted in Fahrenheit: ", Fahrenheit_temps)
```

Output:



```
Temperature in Celsius:  [11, 22, 88, 45, 69]
Temperature converted in Fahrenheit:  [51.8, 71.6, 190.4, 113.0, 156.2]

Process finished with exit code 0
```

6. To create modules in python and access functions of the module by importing it to another file/module. (Calculator program)

Source Code:

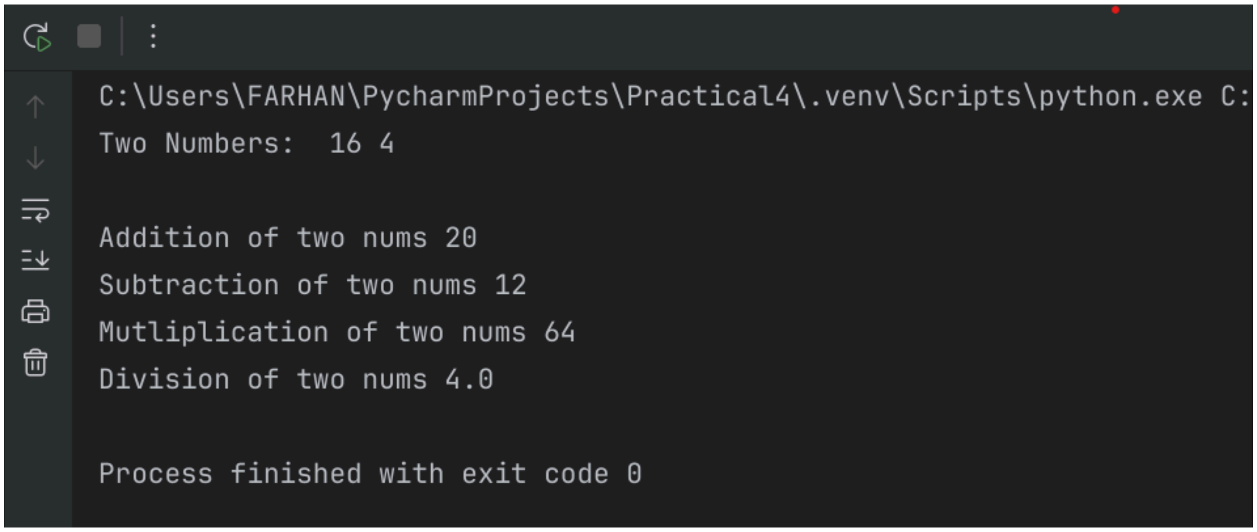
Calculator.py

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

def subtract(a,b):
     return a - b

def multiply(a,b):
     return a * b

def divide(a,b):
     if(b!=0):
     return a / b
     else:
     print("Cannot Divide by Zero")
```

Q6.py

```
import Calculator as cal
a = 16
b = 4
print("Two Numbers: ",a,b)
print()
print("Addition of two nums",cal.add(a,b))
print("Subtraction of two nums",cal.subtract(a,b))
print("Mutliplication of two nums",cal.multiply(a,b))
print("Division of two nums",cal.divide(a,b))
```

Output:



```
C:\Users\FARHAN\PycharmProjects\Practical4\.venv\Scripts\python.exe C:
Two Numbers:  16 4

Addition of two nums 20
Subtraction of two nums 12
Mutliplication of two nums 64
Division of two nums 4.0

Process finished with exit code 0
```

Conclusion:
From this practical I have learned how to implement recursive functions, Lamdas, creating and using python modules.

Name of Student: Shaikh Farhan Ahmed			
Roll Number: 50		**LAB Assignment Number: 05**	
Title of LAB Assignment: To implement programs on OOP Concepts in Python			
DOP: 30 – 11 – 2024		**DOS: 3 – 12 – 2024**	
CO Mapped: CO2	**PO Mapped: PO5,PSO1**	**Faculty Signature:**	**Marks:**

Practical No 5

AIM: To implement programs on OOP Concepts in Python

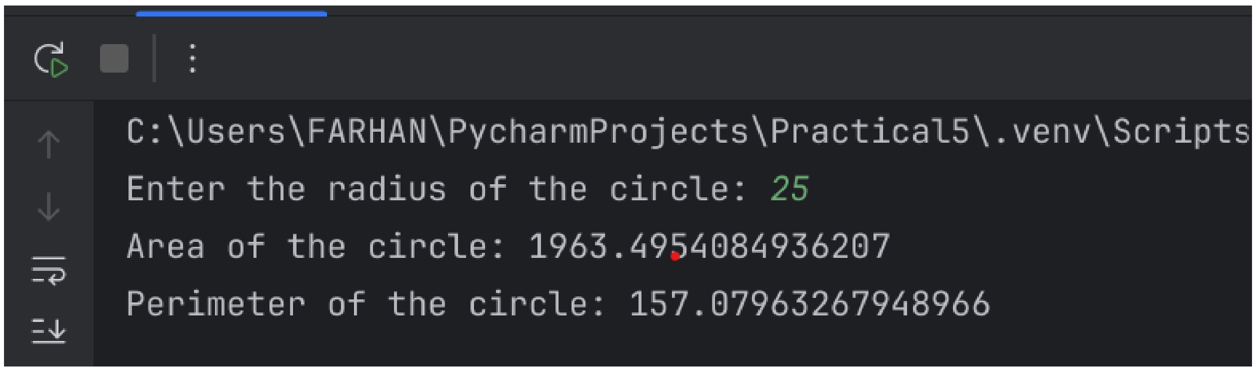
Programs:

1. Python Program to Create a Class and Compute the Area and the Perimeter of the Circle

Source Code:

```
import math
class Circle:
     def __init__(self, radius):
          self.radius = radius
     def area(self):
          return math.pi * self.radius ** 2
     def perimeter(self):
          return 2 * math.pi * self.radius

radius = float(input("Enter the radius of the circle: "))
my_circle = Circle(radius)
print(f"Area of the circle: {my_circle.area()}")
print(f"Perimeter of the circle: {my_circle.perimeter()}")
```

Output:



```
C:\Users\FARHAN\PycharmProjects\Practical5\.venv\Scripts
Enter the radius of the circle: 25
Area of the circle: 1963.4954084936207
Perimeter of the circle: 157.07963267948966
```

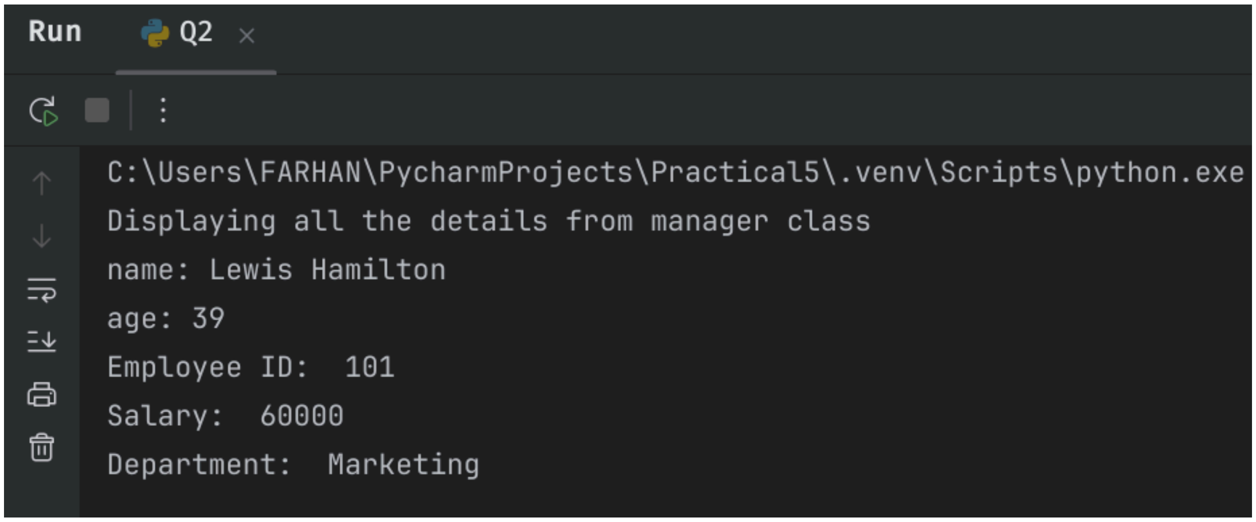
2. Write a Python program to demonstrate multiple inheritance with the following classes:

- Person: Attributes `name` and `age` with a method `display_person_details()`.
- Employee: Attributes `employee_id` and `salary` with a method `display_employee_details()`.
- Manager: Inherits from both `Person` and `Employee`, adds a `department` attribute, and a method `display_manager_details()` to display all details. Create a `Manager` object and display its details.

Source Code:

```
class Person:

     def __init__(self):

     self.name = 'Lewis Hamilton'

     self.age = 39


     def display_person_details(self):

     print("name:",self.name)

     print("age:",self.age)


class Employee:

     def __init__(self):

     self.employee_id = 101

     self.salary = 60000
     def display_employee_details(self):
     print("Employee_ID:",self.employee_id)
     print("Salary:",self.salary)

class Manager(Person,Employee):
     def __init__(self):
     Person.__init__(self)
     Employee.__init__(self)
     self.department = 'Marketing'

     def display_manager_details(self):
     print("name:",self.name)
     print("age:",self.age)
     print("Employee ID: ",self.employee_id)
     print("Salary: ",self.salary)
     print("Department: ",self.department)

manager = Manager()
print("Displaying all the details from manager class")
manager.display_manager_details()
```

Output:


Run Q2

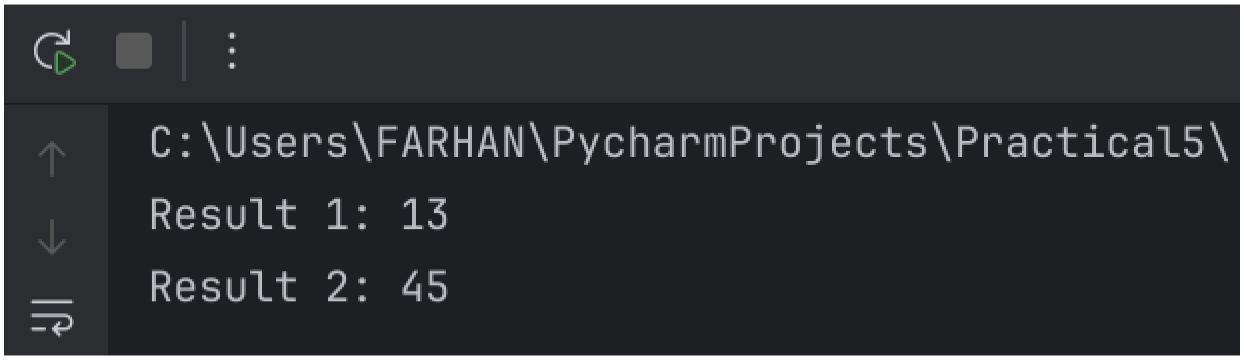
```
C:\Users\FARHAN\PycharmProjects\Practical5\.venv\Scripts\python.exe
Displaying all the details from manager class
name: Lewis Hamilton
age: 39
Employee ID:  101
Salary:  60000
Department:  Marketing
```

3. To Implement a program with the same method name and multiple arguments

Source Code:

```
class MathOperations:

     def add(self, a=0, b=0):

     return a + b

     def add(self, a=0,b=0,c=0):

     return a+b+c


math_operations = MathOperations()

result1 = math_operations.add(10, 3)

result2 = math_operations.add(2, 3, 40)

print("Result 1:", result1)

print("Result 2:", result2)
```

Output:



```
C:\Users\FARHAN\PycharmProjects\Practical5\
Result 1: 13
Result 2: 45
```

4. To Implement Operator Overloading in Python\

Create a class point and Check whether the

- 2 point objects are equal
- Add two point objects
- find the difference between 2 point objects using operator overloading

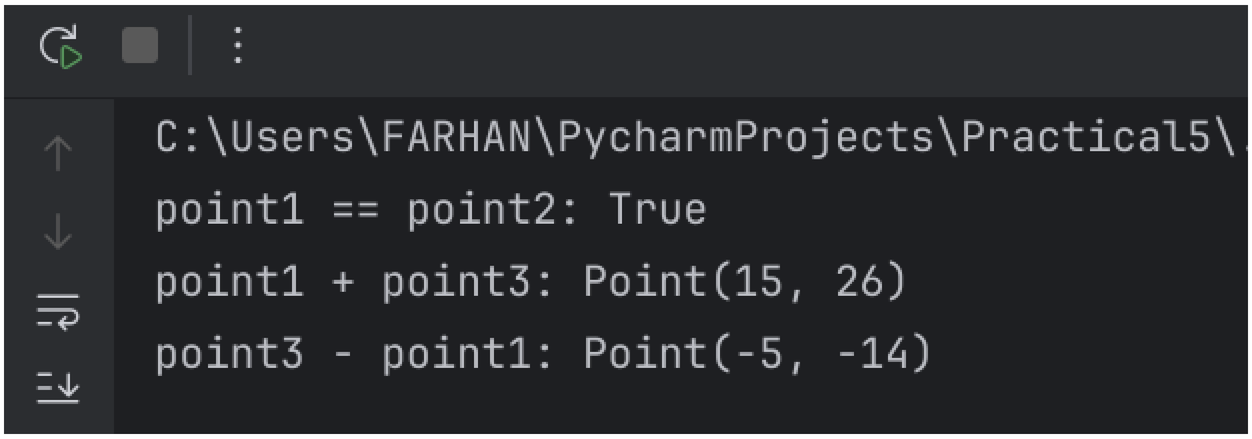
Source Code:

```
class Point:
    def __init__(self, x, y):
    self.x = x
    self.y = y
    def __eq__(self, other):
    # Overload the equality operator (==)
    if isinstance(other, Point):
        return self.x == other.x and self.y == other.y
    return False
    def __add__(self, other):
```

```
    # Overload the addition operator (+)

    if isinstance(other, Point):

        return Point(self.x + other.x, self.y + other.y)

    return NotImplemented


    def __sub__(self, other):

    # Overload the subtraction operator (-)

    if isinstance(other, Point):

        return Point(self.x - other.x, self.y - other.y)

    return NotImplemented


    def __repr__(self):

    # String representation of the Point object

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


# Create two Point objects

point1 = Point(10, 20)

point2 = Point(10, 20)

point3 = Point(5, 6)


# Check if point1 and point2 are equal

print(f"point1 == point2: {point1 == point2}")

# Add two points
```

```
result_add = point1 + point3

print(f"point1 + point3: {result_add}")


# Subtract two points

result_sub = point3 - point1

print(f"point3 - point1: {result_sub}")
```

Output:



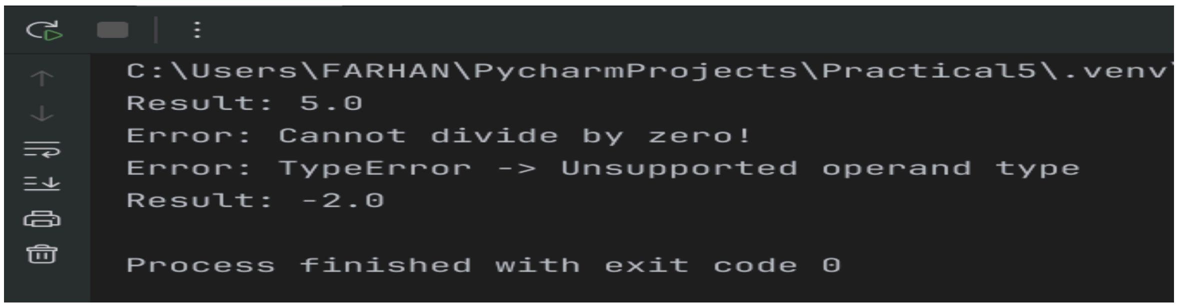
```
C:\Users\FARHAN\PycharmProjects\Practical5\
point1 == point2: True
point1 + point3: Point(15, 26)
point3 - point1: Point(-5, -14)
```

5. Write a program that handles various exceptions in Python (ZeroDivisionError, ValueError, IndexError)

Source Code:

```
def divide_numbers(a, b):

     try:

     result = a / b

     print("Result:", result)

     except ZeroDivisionError:

     print("Error: Cannot divide by zero!")

     except TypeError:

     print("Error: TypeError -> Unsupported operand type")

     except ValueError as e:

     print(f"An unexpected error occurred: {e}")

divide_numbers(10,2)

divide_numbers(10,0)

divide_numbers(10,"5")

divide_numbers(20,-10)
```

Output:



```
C:\Users\FARHAN\PycharmProjects\Practical5\.venv
Result: 5.0
Error: Cannot divide by zero!
Error: TypeError -> Unsupported operand type
Result: -2.0

Process finished with exit code 0
```

Conclusion:

These programs collectively provide a solid foundation for understanding OOP principles, inheritance, polymorphism, and effective error handling. By mastering these concepts, programmers can write more modular, reusable, and resilient code in Python.

Name of Student: Shaikh Farhan Ahmed			
Roll Number: 50		**LAB Assignment Number: 06**	
Title of LAB Assignment: To Implement programs on Data Structures using python			
DOP: 3 – 12 – 2024		**DOS: 4 – 12 – 2024**	
CO Mapped: CO3	**PO Mapped: PO5,PSO1**	**Faculty Signature:**	**Marks:**

Practical No 6

AIM: To Implement programs on Data Structures using python

Theory:

Linked List

A Linked List is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers.

Operations:

Insertion: Adding a new node to the linked list

Traversal: Visiting each node in sequence

Removal: Deleting a node from the list.

Stack

A Stack is a linear data structure that stores items in a Last-In/First-Out (LIFO) or First-In/Last-Out (FILO) manner. In stack, a new element is added at one end and an element is removed from that end only. The insert and delete operations are often called push and pop.

The functions associated with stack are:

- **empty() –** Returns whether the stack is empty – Time Complexity: O(1)
- **size() –** Returns the size of the stack – Time Complexity: O(1)
- **top() –** Returns a reference to the topmost element of the stack – Time Complexity: O(1)
- **push(a) –** Inserts the element ‘a’ at the top of the stack – Time Complexity: O(1)
- **pop() –** Deletes the topmost element of the stack – Time Complexity: O(1)

Queue

As a stack, the queue is a linear data structure that stores items in a First In First Out (FIFO) manner. With a queue, the least recently added item is removed first. A good example of the queue is any queue of consumers for a resource where the consumer that came first is served first.

Operations associated with queue are:

- **Enqueue:** Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition – Time Complexity: O(1)
- **Dequeue:** Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition – Time Complexity: O(1)
- **Front:** Get the front item from queue – Time Complexity: O(1)
- **Rear:** Get the last item from queue – Time Complexity: O(1)

Deque (Doubly Ended Queue)

In Python Deque is implemented using the module “collections“. Deque is preferred over a list in the cases where we need quicker append and pop operations from both the ends of the container, as deque provides an O(1) time complexity for append and pop operations as compared to a list that provides O(n) time complexity.

Operations on deque:

- Insertion at Rear.
- Insertion at Front.
- Deletion at Front.
- Deletion at Rear.

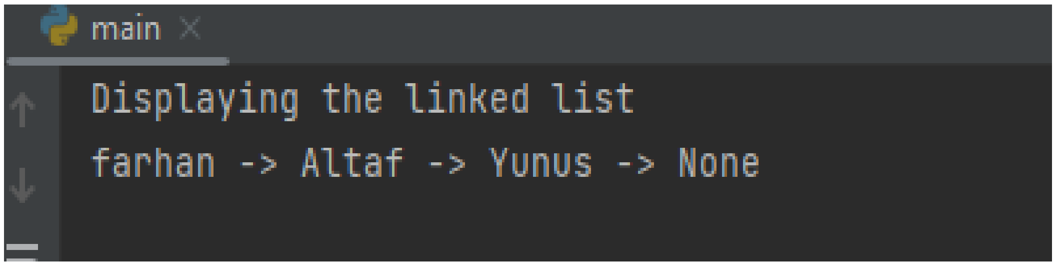
Programs:

1. To Create, Traverse, Insert and Remove Data using Linked List:

Source Code:

```
class Node:

    def __init__(self, data):

        self.data = data

        self.next = None


class HeadNode:

    def __init__(self):

        self.head = None


    def append(self, data):

        new_node = Node(data)

        if not self.head:

            self.head = new_node

        else:

            current = self.head

            while current.next:

                current = current.next

            current.next = new_node
    def display(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")


my_list = HeadNode()
my_list.append("farhan")
my_list.append("Altaf")
my_list.append("Yunus")
print("Displaying the linked list")
my_list.display()
```

Output:



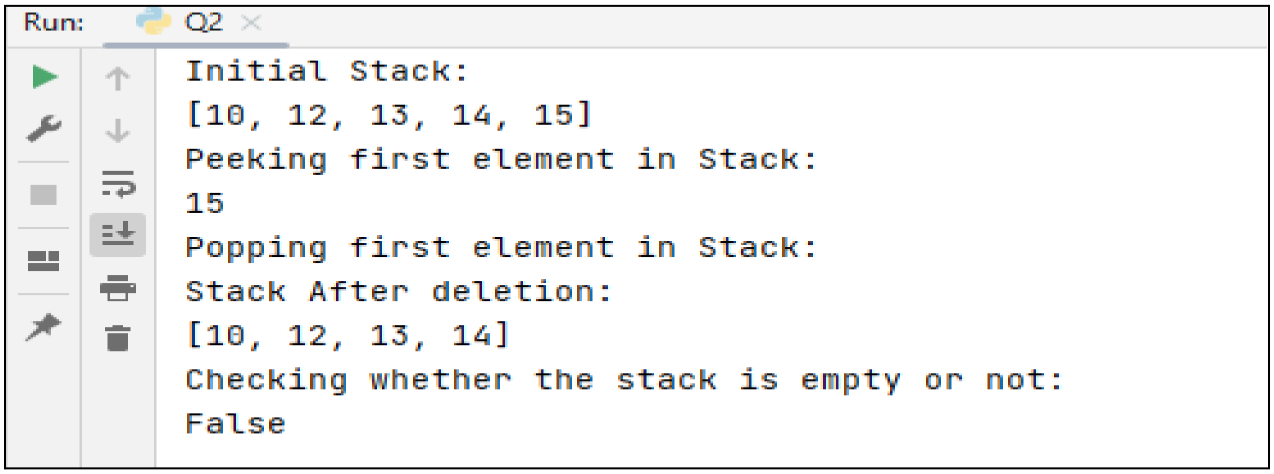
```
Displaying the linked list
farhan -> Altaf -> Yunus -> None
```

2. Implementation of Stacks.

Source Code:

```
class StackImpl:

    def __init__(self):

    self.items = []


    def pushItem(self,item):

    self.items.append(item)


    def popItem(self,item):

    if not self.isEmpty():

        return self.items.pop()

    else:

        None


    def peekItem(self):

        if not self.isEmpty():

        return self.items[-1]

        else:

        None
    def isEmpty(self):

    return not self.items

    def printStack(self):

         print(self.items)


stack = StackImpl()

stack.pushItem(10)

stack.pushItem(12)

stack.pushItem(13)

stack.pushItem(14)

stack.pushItem(15)


print("Initial Stack:")

stack.printStack()

print("Peeking first element in Stack:")

print(stack.peekItem())

print("Popping first element in Stack:")

stack.popItem(item = 15)

print("Stack After deletion:")

stack.printStack()

print("Checking whether the stack is empty or not:")

print(stack.isEmpty())
```

Output:



```
Initial Stack:
[10, 12, 13, 14, 15]
Peeking first element in Stack:
15
Popping first element in Stack:
Stack After deletion:
[10, 12, 13, 14]
Checking whether the stack is empty or not:
False
```

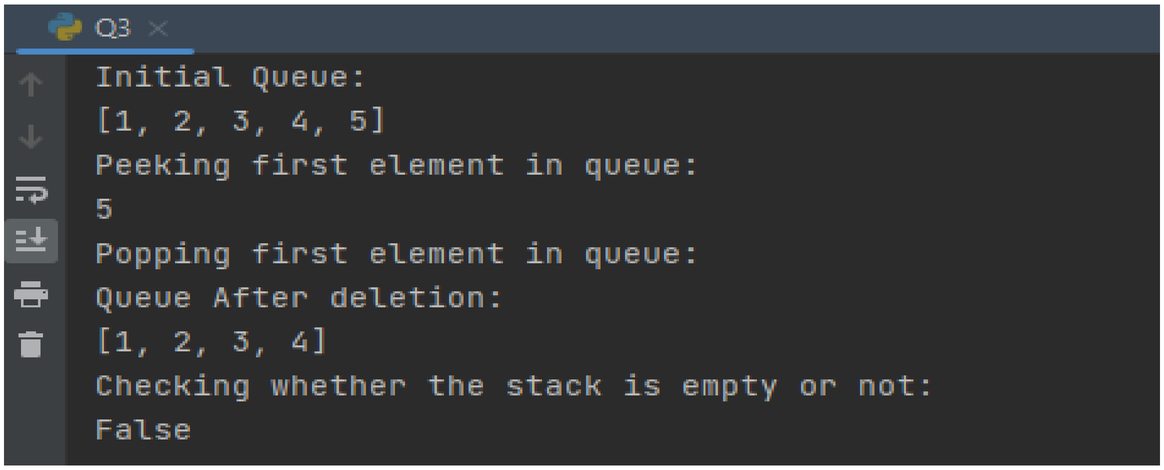
3. Implementation of queue

Source Code:

```
class QueueImpl:

    def __init__(self):

        self.items = []


    def pushItem(self,item):

         self.items.append(item)


    def popItem(self,item):

        if not self.isEmpty():

            return self.items.pop()

        else:

            None
```

```
     def peekItem(self):

              if not self.isEmpty():

              return self.items[-1]

              else:

              None


     def isEmpty(self):

         return not self.items


     def printQueue(self):

              print(self.items)


queue = QueueImpl()

queue.pushItem(1)

queue.pushItem(2)

queue.pushItem(3)

queue.pushItem(4)

queue.pushItem(5)


print("Initial Queue:")

queue.printQueue()

print("Peeking first element in queue:")
print(queue.peekItem())

print("Popping first element in queue:")

queue.popItem(item = 15)

print("Queue After deletion:")

queue.printQueue()

print("Checking whether the stack is empty or not:")

print(queue.isEmpty())
```

Output:



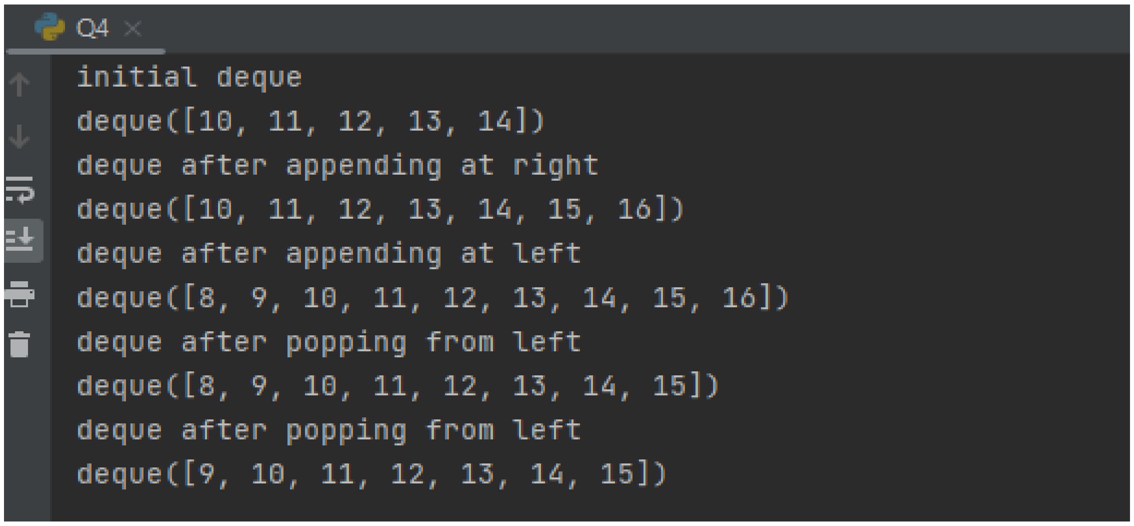
```
Initial Queue:
[1, 2, 3, 4, 5]
Peeking first element in queue:
5
Popping first element in queue:
Queue After deletion:
[1, 2, 3, 4]
Checking whether the stack is empty or not:
False
```

4. Implementation of deque

Source Code:

```
import collections

deque = collections.deque([10,11,12,13,14])

print("initial deque")

print(deque)


deque.append(15)

deque.append(16)

print("deque after appending at right")

print(deque)


deque.appendleft(9)

deque.appendleft(8)

print("deque after appending at left")

print(deque)


deque.pop()

print("deque after popping from left")

print(deque)

deque.popleft()

print("deque after popping from left")

print(deque)
```

Output:



```
initial deque
deque([10, 11, 12, 13, 14])
deque after appending at right
deque([10, 11, 12, 13, 14, 15, 16])
deque after appending at left
deque([8, 9, 10, 11, 12, 13, 14, 15, 16])
deque after popping from left
deque([8, 9, 10, 11, 12, 13, 14, 15])
deque after popping from left
deque([9, 10, 11, 12, 13, 14, 15])
```

Conclusion:

From this practical I have learned to implement data structures in python which includes Linkedlist, Stack, Queue and deque.

Name of Student: Shaikh Farhan Ahmed			
Roll Number: 50		**LAB Assignment Number: 07**	
Title of LAB Assignment: To implement programs on File Handling in Python.			
DOP: 03 – 12 – 2024		**DOS: 04 – 12 – 2024**	
CO Mapped: CO4	**PO Mapped: PO3, PO5, PSO1, PSO2**	**Faculty Signature:**	**Marks:**

Practical No 7

AIM: To implement programs on File Handling in Python

Theory:

File handling in Python refers to the process of creating, reading, writing, updating, and deleting files. It is an essential skill for working with data storage and retrieval in various applications, including logging systems, data analysis, and document generation. Files are used to store data permanently on a disk, and Python provides built-in functions andmethods to perform file operations.

These operations allow programs to interact with files in a structured manner, ensuring efficient data management.

Key Concepts in File Handling:

1. File Modes:

File modes specify the purpose of opening a file, such as reading, writing, or appending.

- The commonly used file modes are:
- 'r' (Read mode): Opens a file for reading. If the file does not exist, it raises a FileNotFoundError.
- 'w' (Write mode): Opens a file for writing. If the file exists, it overwrites the file; otherwise, it creates a new file.
- 'a' (Append mode): Opens a file for appending. New content is added at the end without affecting the existing data.
- 'r+' (Read and Write mode): Opens a file for both reading and writing.

2. File Access Methods:

- Text mode ('t'): Used for handling text files (default mode).
- Binary mode ('b'): Used for handling binary files, such as images and videos.

3. File Operations:

- Opening a File: Done using the open() function, which returns a file object.
- Reading from a File: Methods like read(), readline(), and readlines() are used to extract data.
- Writing to a File: Methods like write() and writelines() allow adding content to a file.
- Closing a File: Ensures that changes are saved and resources are freed. Python uses the close() method or the with statement to handle this automatically.

4. Error Handling:

File operations often require robust error handling to manage scenarios where files do not exist or permissions are restricted. Python uses try-except blocks to handle exceptions like FileNotFoundError and PermissionError.

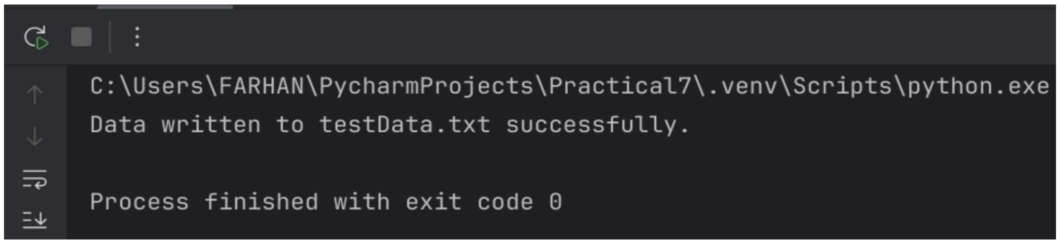
1. Write a program to open a file in write mode, write multiple lines to the file, and then close the file

Source Code:

```
file_name = "testData.txt"

# Open the file in write mode

with open(file_name, 'w') as file:

     file.write("This is line 1\n")

     file.write("This is line 2\n")

     file.write("This is line 3\n")

     print(f"Data written to {file_name} successfully.")
```

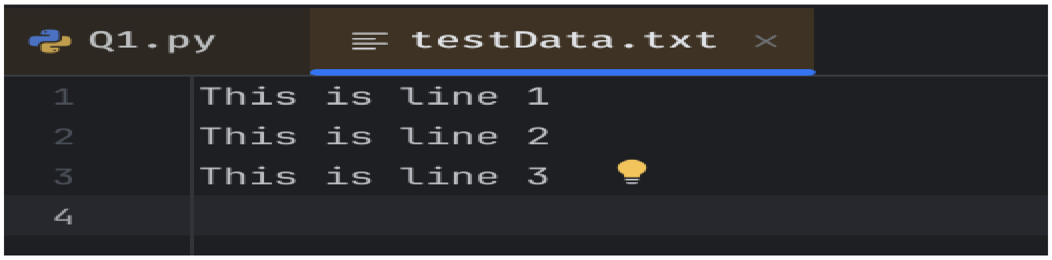
Output:

Writing to text file:



```
C:\Users\FARHAN\PycharmProjects\Practical7\.venv\Scripts\python.exe
Data written to testData.txt successfully.

Process finished with exit code 0
```

Text file after writing:



```
This is line 1
This is line 2
This is line 3
```

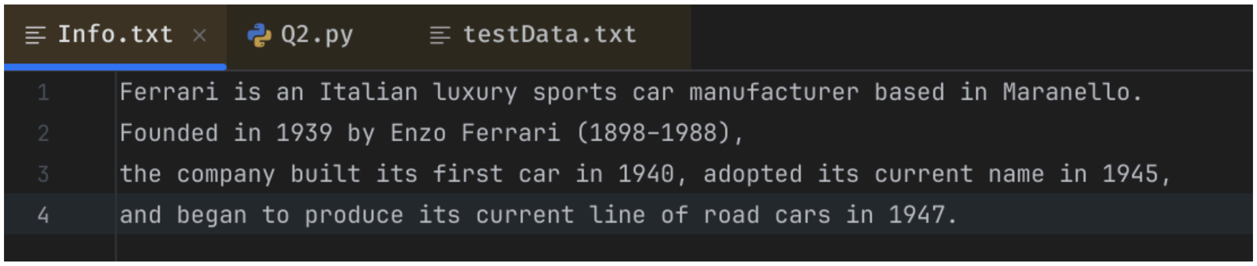
2. Write a program to read the contents of an existing file and display them line by line. If the file doesn't exist, handle the exception

Source Code:

```
file_name = "Info.txt"

try:

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

    print("File Contents:")

    for line in file:

        print(line.strip())

except FileNotFoundError:

    print(f"Error: The file {file_name} does not exist.")
```

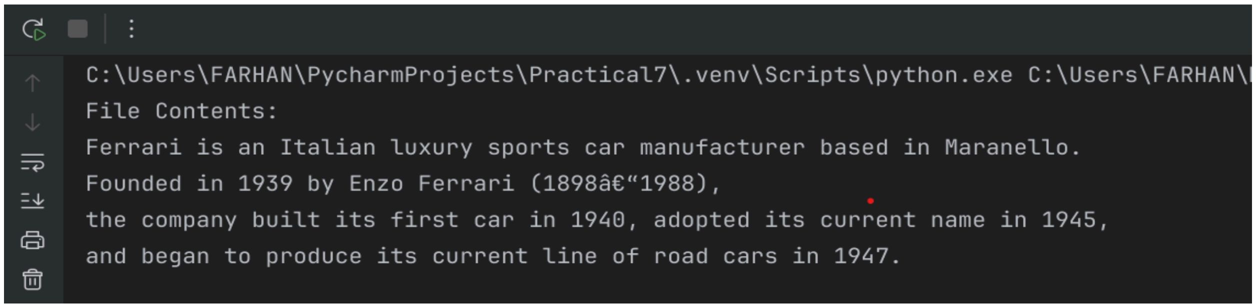
Output:

Text File Content:



```
Ferrari is an Italian luxury sports car manufacturer based in Maranello.
Founded in 1939 by Enzo Ferrari (1898–1988),
the company built its first car in 1940, adopted its current name in 1945,
and began to produce its current line of road cars in 1947.
```

Reading it from python console:



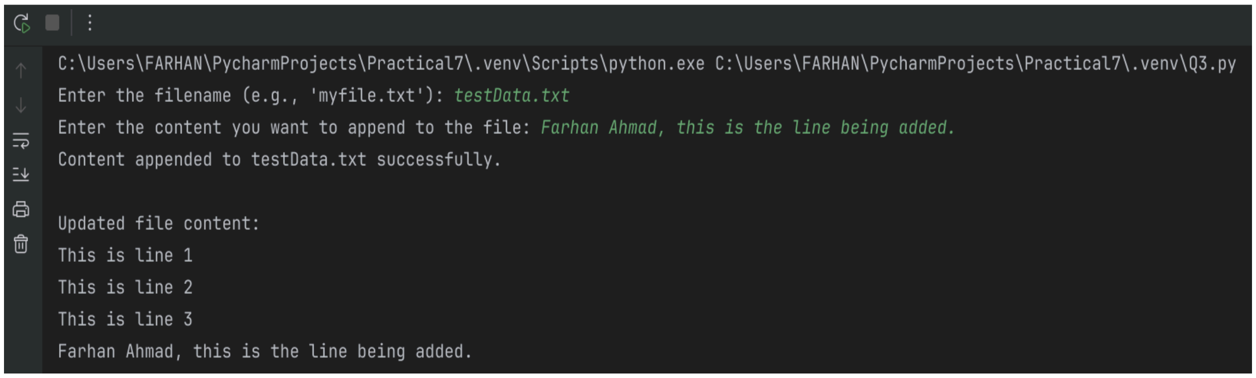
```
C:\Users\FARHAN\PycharmProjects\Practical7\.venv\Scripts\python.exe C:\Users\FARHAN\
File Contents:
Ferrari is an Italian luxury sports car manufacturer based in Maranello.
Founded in 1939 by Enzo Ferrari (1898â€“1988),
the company built its first car in 1940, adopted its current name in 1945,
and began to produce its current line of road cars in 1947.
```

3. Write a program that asks the user for a filename, appends new content to the file, and then reads the entire file content after the append.

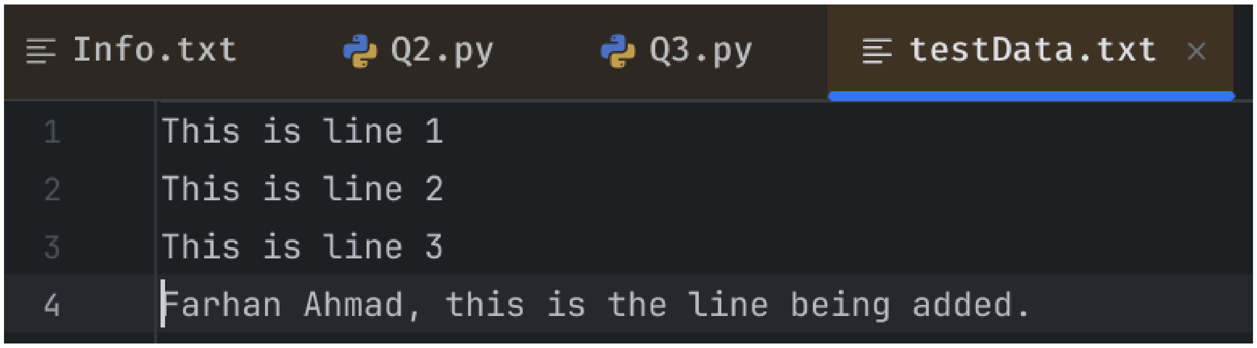
Source Code:

```
file_name = input("Enter the filename (e.g., 'myfile.txt'): ")

new_content = input("Enter the content you want to append to the
file: ")

try:

    with open(file_name, 'a') as file:

    file.write(new_content + "\n")

    print(f"Content appended to {file_name} successfully.")


# Open the file in read mode and display the updated content

    print("\nUpdated file content:")

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

    for line in file:

        print(line.strip())


except FileNotFoundError:

    print(f"Error: The file '{file_name}' does not exist.")

except PermissionError:

    print(f"Error: You do not have permission to write to
'{file_name}'.")

except Exception as e:

    print(f"An unexpected error occurred: {e}")
```

Output:



```
C:\Users\FARHAN\PycharmProjects\Practical7\.venv\Scripts\python.exe C:\Users\FARHAN\PycharmProjects\Practical7\.venv\Q3.py
Enter the filename (e.g., 'myfile.txt'): testData.txt
Enter the content you want to append to the file: Farhan Ahmad, this is the line being added.
Content appended to testData.txt successfully.

Updated file content:
This is line 1
This is line 2
This is line 3
Farhan Ahmad, this is the line being added.
```



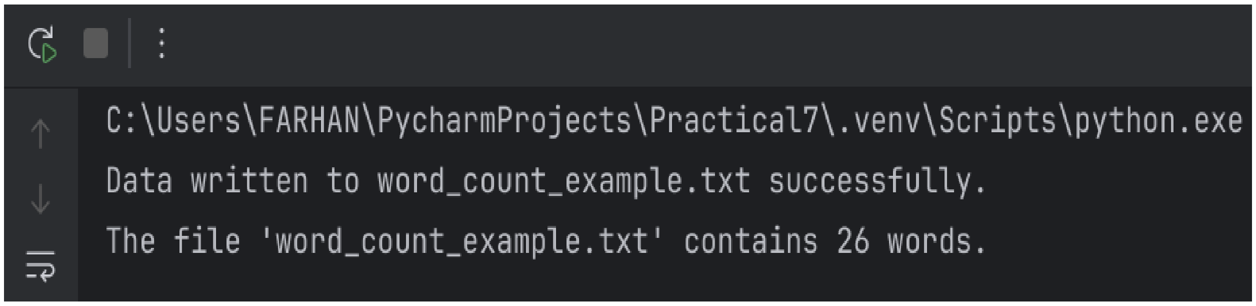
```
This is line 1
This is line 2
This is line 3
Farhan Ahmad, this is the line being added.
```

4. Write a program that creates a file, writes a few lines of text, and then counts the number of words in the file.

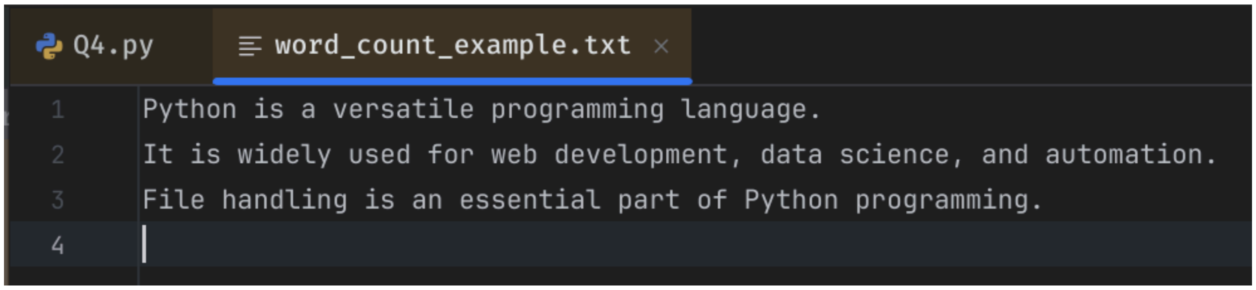
Source Code:

```
file_name = "word_count_example.txt"


with open(file_name, 'w') as file:

     file.write("Python is a versatile programming language.\n")

     file.write("It is widely used for web development, data 
science, and automation.\n")

     file.write("File handling is an essential part of Python 
programming.\n")

     print(f"Data written to {file_name} successfully.")


# Count the number of words in the file

word_count = 0

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

     for line in file:

     word_count += len(line.split())

print(f"The file '{file_name}' contains {word_count} words.")
```

Output:



```
C:\Users\FARHAN\PycharmProjects\Practical7\.venv\Scripts\python.exe
Data written to word_count_example.txt successfully.
The file 'word_count_example.txt' contains 26 words.
```



```
Python is a versatile programming language.
It is widely used for web development, data science, and automation.
File handling is an essential part of Python programming.

```

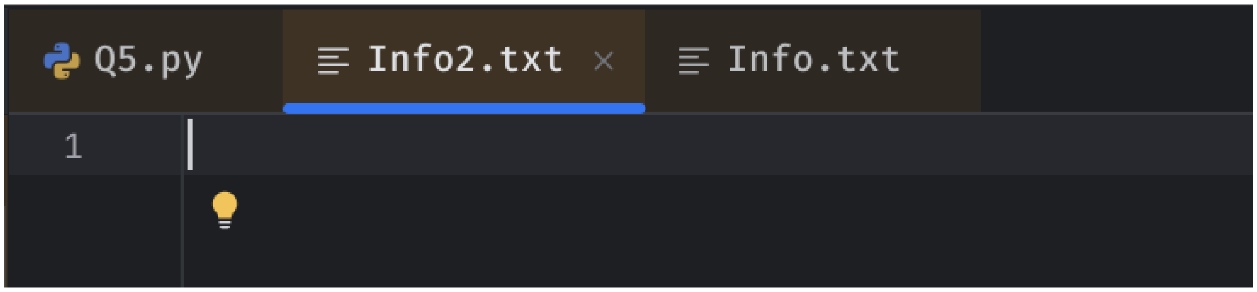
5. Write a program to open a file and copy its content into another file.

Source Code:

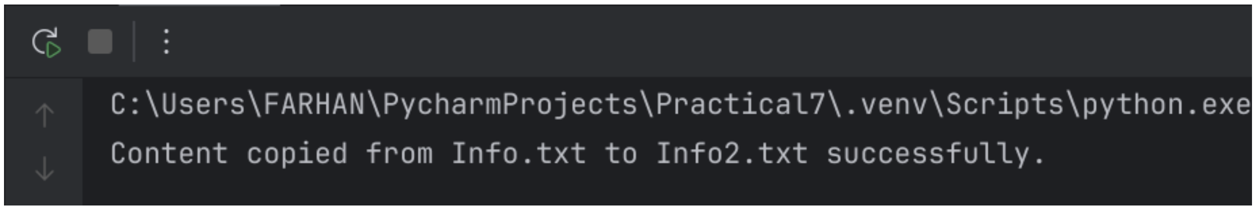
```
# Program to copy content from one file to another

source_file = "Info.txt"

destination_file = "Info2.txt"

try:

# Open the source file and read its content

     with open(source_file, 'r') as src:

     content = src.read()

# Open the destination file and write the content

     with open(destination_file, 'w') as dest:

     dest.write(content)

     print(f"Content copied from {source_file} to
{destination_file} successfully.")

except FileNotFoundError:

     print(f"Error: The source file {source_file} does not
exist.")
```

Output:

Text file before copying:

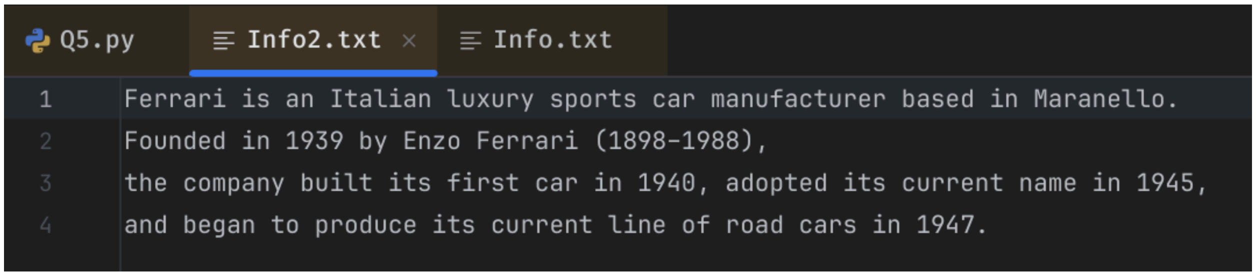


Copying content from one file to another:



```
C:\Users\FARHAN\PycharmProjects\Practical7\.venv\Scripts\python.exe
Content copied from Info.txt to Info2.txt successfully.
```

Text file after copying:



```
Ferrari is an Italian luxury sports car manufacturer based in Maranello.
Founded in 1939 by Enzo Ferrari (1898-1988),
the company built its first car in 1940, adopted its current name in 1945,
and began to produce its current line of road cars in 1947.
```

Conclusion:

From this practical I have learned:

1. Creating files and writing data to them.

2. Reading and displaying file contents line by line.

3. Appending data to existing files and retrieving updated content.

4. Counting words in a file, demonstrating how to manipulate text programmatically.

5. Copying file content from one file to another, showcasing efficient file transfer methods.

Name of Student: Shaikh Farhan Ahmed			
Roll Number: 50		**LAB Assignment Number: 08**	
Title of LAB Assignment: To Implement GUI programming in Python.			
DOP: 3 – 12 – 2024		**DOS: 4 – 12 – 2024**	
CO Mapped: CO5	**PO Mapped: PO3, PO5, PSO1, PSO2**	**Faculty Signature:**	**Marks:**

Practical No 8

AIM: To implement GUI programming in python.

Theory:

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python **Tkinter** is the fastest and easiest way to create GUI applications. Creating a GUI using Tkinter is an easy task.

There are two main methods used which the user needs to remember while creating the Python application with GUI.

Tk()

To create a main window, tkinter offers a method 'Tk(screenName=None, baseName=None, className='Tk', useTk=1)'. To change the name of the window, you can change the className to the desired one. The basic code used to create the main window of the application is:

mainloop()

There is a method known by the name mainloop() is used when your application is ready to run. mainloop() is an infinite loop used to run the application, wait for an event to occur, and process the event as long as the window is not closed.

1. To Design Login Page.

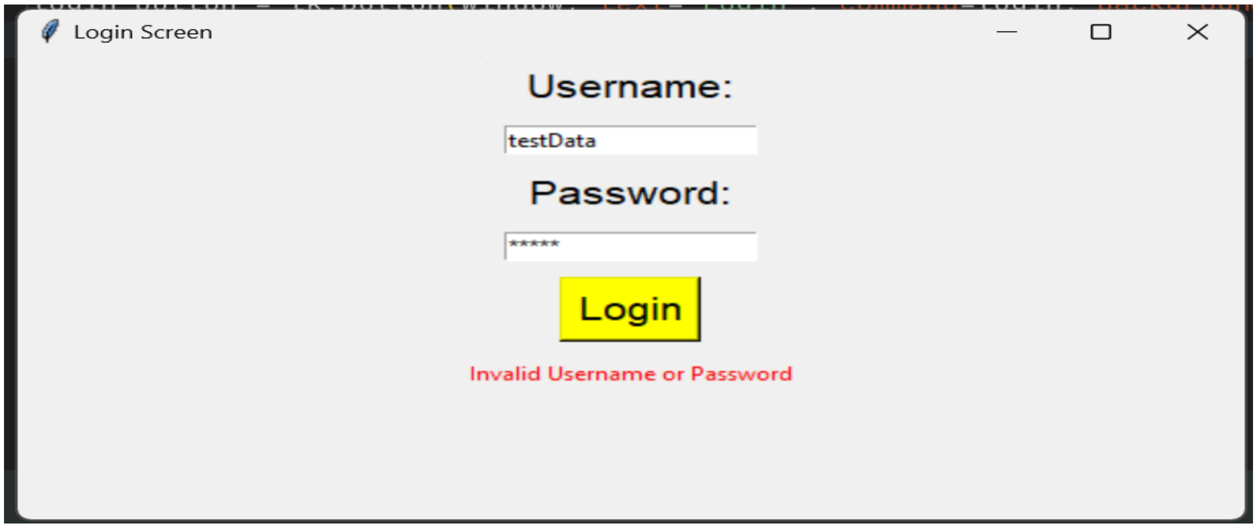
Source Code:

```
import tkinter as tk

from tkinter import font

# Function to handle the login action

def login():

     username = username_entry.get()

     password = password_entry.get()

     if username == "farhan" and password == "blaze123":

     label_status.config(text="Login Successful!", fg="green")

     else:

     label_status.config(text="Invalid Username or Password",

     fg="red")


# Creating the main window

window = tk.Tk()

window.title("Login Screen")

window.geometry("600x300")

custom_font = font.Font(size=16)

# Creating and placing widgets

username_label = tk.Label(window, text="Username:",
font=custom_font)

username_label.pack(pady=5)

username_entry = tk.Entry(window)
```

```
username_entry.pack(pady=5)

password_label = tk.Label(window, text="Password:",
font=custom_font)

password_label.pack(pady=5)

password_entry = tk.Entry(window, show="*") # Hide the password
text

password_entry.pack(pady=5)

login_button = tk.Button(window, text="Login", command=login,
background="yellow",

fg="black", font=custom_font)

login_button.pack(pady=5)

label_status = tk.Label(window, text="", fg="white" )

label_status.pack(pady=5)

# Starting the main loop

window.mainloop()
```

Output:

Entering invalid username and password:



Login Screen

Username:

testData

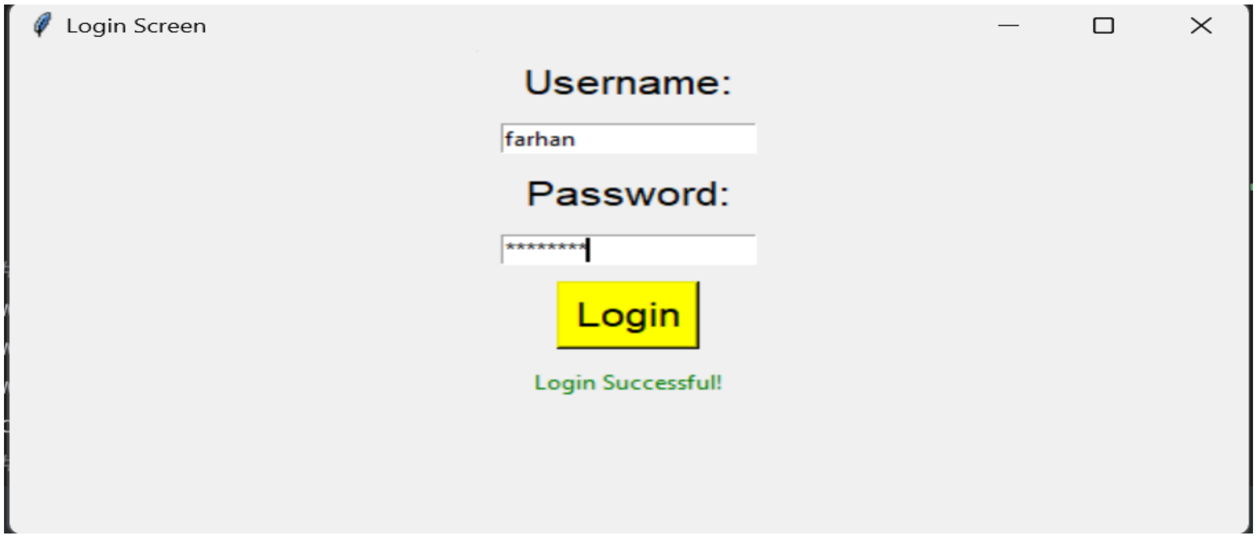
Password:

*****

Login

Invalid Username or Password

Entering Valid Username and Password:



Login Screen

Username:

farhan

Password:

****

Login

Login Successful!

2. To Design Student Information Form/Library management Form/Hospital Management Form

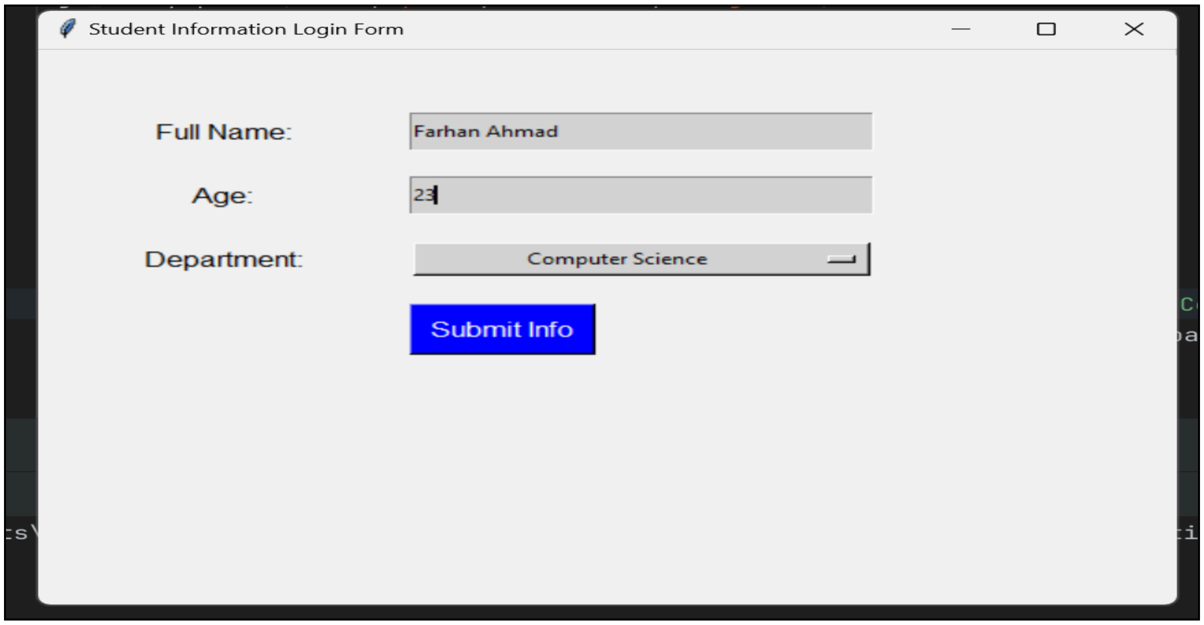
Source Code:

```
import tkinter as tk

from tkinter import font


# Function to handle form submission

def submit():

    name = name_entry.get()

    age = age_entry.get()

    department = department_var.get()

    if name and age and department:

    label_status.config(text="Details submitted successfully!",

    fg="green")

    else:

    label_status.config(text="Fill in correct details!",

    fg="red")


# Creating the main window

window = tk.Tk()

window.title("Student Information Login Form")

window.geometry("600x400")

custom_font = font.Font(size=12)
```

```
# Creating and placing widgets

name_label = tk.Label(window, text="Full Name:", width=10,
font=custom_font)

name_label.place(x=50, y=50, width=100, height=30)

name_entry = tk.Entry(window)

name_entry.place(x=200, y=50, width=250, height=30)

name_entry.config(bg="#D3D3D3")

age_label = tk.Label(window, text="Age:", width=10,
font=custom_font)

age_label.place(x=50, y=100, width=100, height=30)

age_entry = tk.Entry(window)

age_entry.place(x=200, y=100, width=250, height=30)

age_entry.config(bg="#D3D3D3")

department_label = tk.Label(window, text="Department:",
font=custom_font)

department_label.place(x=50, y=150, width=100, height=30)


# Creating a variable for the department drop-down

department_var = tk.StringVar(window)

department_var.set("")

# Creating a drop-down menu for department selection

department_choices = ["Mathematics", "Physics", "Chemistry",
"Biology", "Computer Science"]

department_dropdown = tk.OptionMenu(window, department_var,
*department_choices)
```

```
department_dropdown.place(x=200, y=150, width=250, height=30)

department_dropdown.config(bg="#D3D3D3")

# Creating the submit button

submit_button = tk.Button(window, text="Submit Info",
command=submit, bg="blue", fg="white",

font=custom_font)

submit_button.place(x=200, y=200, width=100, height=40)

label_status = tk.Label(window, text="", fg="black",
font=custom_font)

label_status.place(x=180, y=250)


# Starting the main loop

window.mainloop()
```

Output:

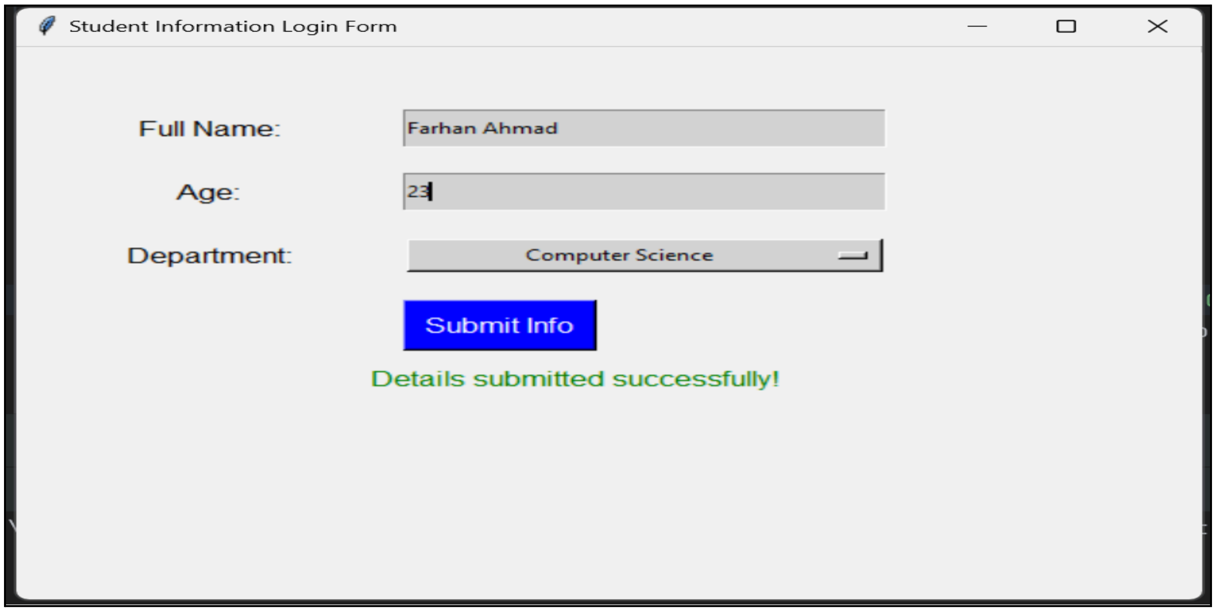


Student Information Login Form

Full Name: Farhan Ahmad

Age: 23

Department: Computer Science



Student Information Login Form

Full Name: Farhan Ahmad

Age: 23

Department: Computer Science

Details submitted successfully!

Conclusion:

From this practical I have learned GUI programming using tkinter which is essential for creating user-friendly interfaces like a Login Page or forms for Student Information as It allows users to interact with software visually, making it more intuitive and accessible.

Name of Student: Shaikh Farhan Ahmed			
Roll Number: 50		**LAB Assignment Number: 09**	
Title of LAB Assignment: To implement Database Connectivity in python			
DOP: 3 – 12 – 2024		**DOS: 4 – 12 – 2024**	
CO Mapped: CO5	**PO Mapped: PO3, PO5, PSO1, PSO2**	**Faculty Signature:**	**Marks:**

Practical No 9

AIM: To implement Database Connectivity in python

Theory:

MySQL is a Relational Database Management System (RDBMS) whereas the structured Query Language (SQL) is the language used for handling the RDBMS using commands i.e Creating, Inserting, Updating and Deleting the data from the databases.

MySQL Connector/Python enables Python programs to access MySQL databases

MySQL Connector/Python includes support for:

- Almost all features provided by MySQL Server version 8.0 and higher.
- Converting parameter values back and forth between Python and MySQL data types
- All MySQL extensions to standard SQL syntax.
- Protocol compression, which enables compressing the data stream between the client and server.

1. Write a Python program to connect to a MySQL database, create a table called employees, and insert records (name, department, salary). Then, update the salary of a specific employee based on their name and display the updated record.

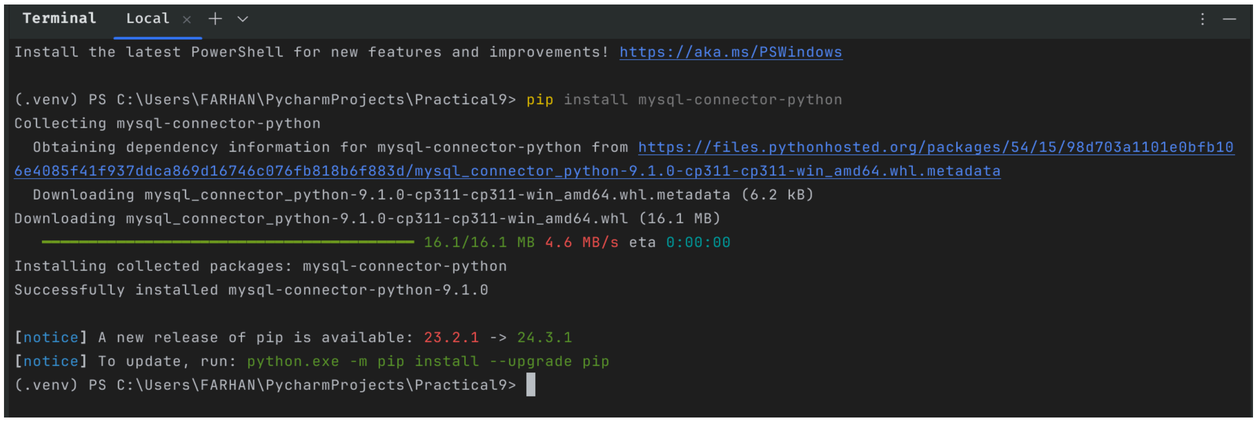
Source Code:

```
import mysql.connector


Create connection object

db_connection = mysql.connector.connect(

     host="localhost",

     user = "root",

     password = "root",

     database = "employee_db"

)


Create a cursor object

cursor = db_connection.cursor()


#Creating db

cursor.execute("CREATE DATABASE IF NOT EXISTS employee_db")

#Creating table and populating it.

cursor.execute("CREATE TABLE Employees(name
VARCHAR(50),department VARCHAR(50),salary int)")

cursor.execute("CREATE TABLE Students(id int, name
VARCHAR(50),age int)")
```

```
cursor.execute("INSERT INTO Employees(name,department,salary)
VALUES('Farhan','IT',50000)")

cursor.execute("INSERT INTO Employees(name,department,salary)
VALUES('Zeeshan','IT',50000)")

cursor.execute("INSERT INTO Employees(name,department,salary)
VALUES('Haneef','HR',40000)")

cursor.execute("INSERT INTO Employees(name,department,salary)
VALUES('Mugi','Marketing',60000)")

cursor.execute("INSERT INTO Employees(name,department,salary)
VALUES('Lewis','Finance',50000)")

db_connection.commit()


cursor.execute("UPDATE Employees SET salary = 80000 WHERE name =
'Farhan'")

db_connection.commit()


#Fetching Records

cursor.execute("SELECT * FROM Employees")

for x in cursor:

     print(x)
```

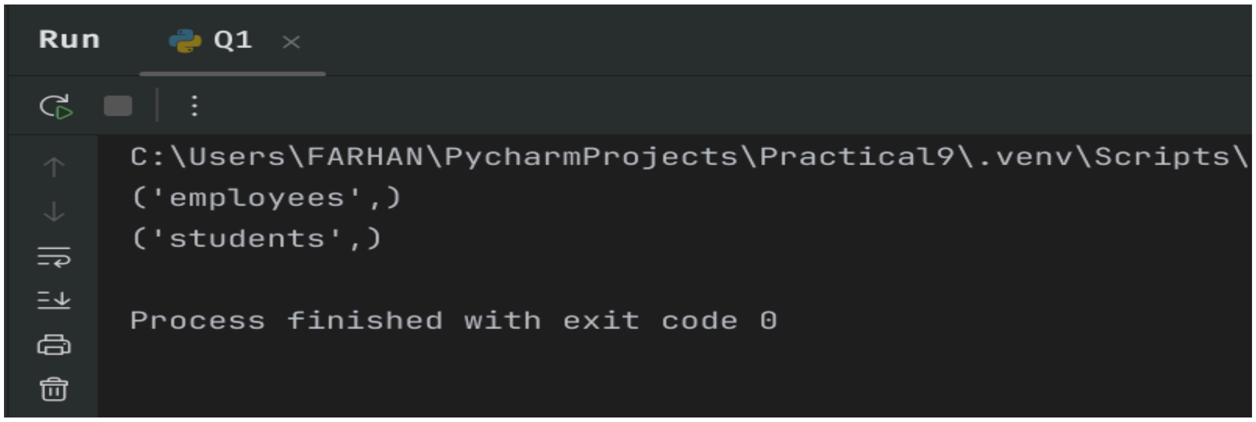
Output:

Installing mysql-connector-python module:



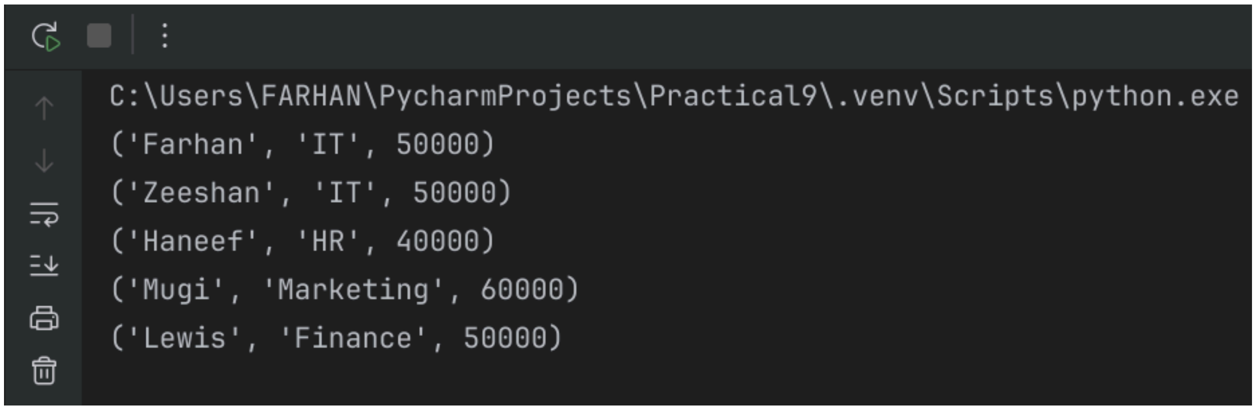
```
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

(.venv) PS C:\Users\FARHAN\PycharmProjects\Practical9> pip install mysql-connector-python
Collecting mysql-connector-python
  Obtaining dependency information for mysql-connector-python from https://files.pythonhosted.org/packages/54/15/98d703a1101e0bfb10
6e4085f41f937ddca869d16746c076fb818b6f883d/mysql_connector_python-9.1.0-cp311-cp311-win_amd64.whl.metadata
  Downloading mysql_connector_python-9.1.0-cp311-cp311-win_amd64.whl.metadata (6.2 kB)
Downloading mysql_connector_python-9.1.0-cp311-cp311-win_amd64.whl (16.1 MB)
   ---------------------------------------- 16.1/16.1 MB 4.6 MB/s eta 0:00:00
Installing collected packages: mysql-connector-python
Successfully installed mysql-connector-python-9.1.0

[notice] A new release of pip is available: 23.2.1 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
(.venv) PS C:\Users\FARHAN\PycharmProjects\Practical9>
```

Displaying Database Tables:



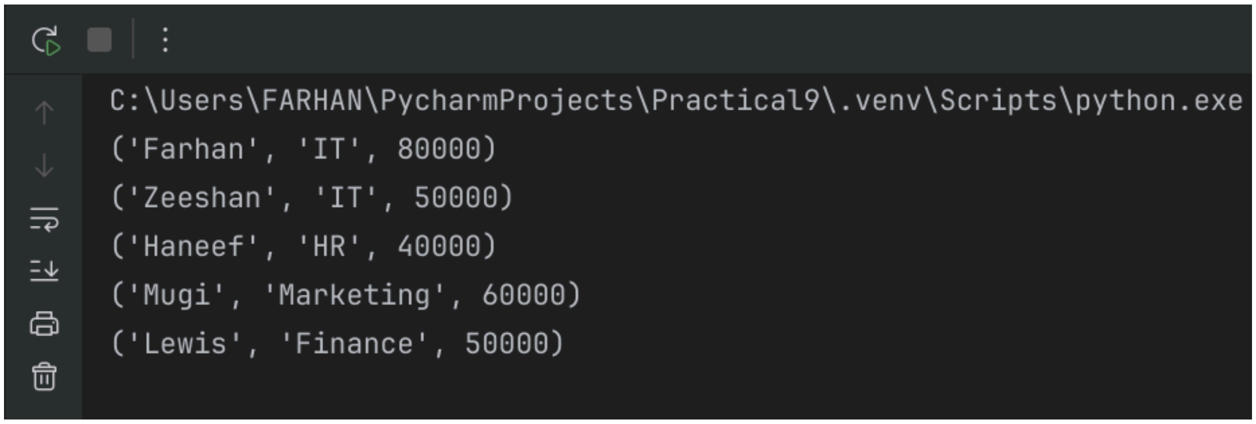
```
C:\Users\FARHAN\PycharmProjects\Practical9\.venv\Scripts\
('employees',)
('students',)

Process finished with exit code 0
```

Displaying all records of Employee Table (Initial Table):



```
C:\Users\FARHAN\PycharmProjects\Practical9\.venv\Scripts\python.exe
('Farhan', 'IT', 50000)
('Zeeshan', 'IT', 50000)
('Haneef', 'HR', 40000)
('Mugi', 'Marketing', 60000)
('Lewis', 'Finance', 50000)
```

Database after updating the salary value of Farhan:



```
C:\Users\FARHAN\PycharmProjects\Practical9\.venv\Scripts\python.exe
('Farhan', 'IT', 80000)
('Zeeshan', 'IT', 50000)
('Haneef', 'HR', 40000)
('Mugi', 'Marketing', 60000)
('Lewis', 'Finance', 50000)
```

2. Write a program to connect to a MySQL database, update a specific record in a table based on a condition, and then display the updated table contents(eg. update student age based on name).

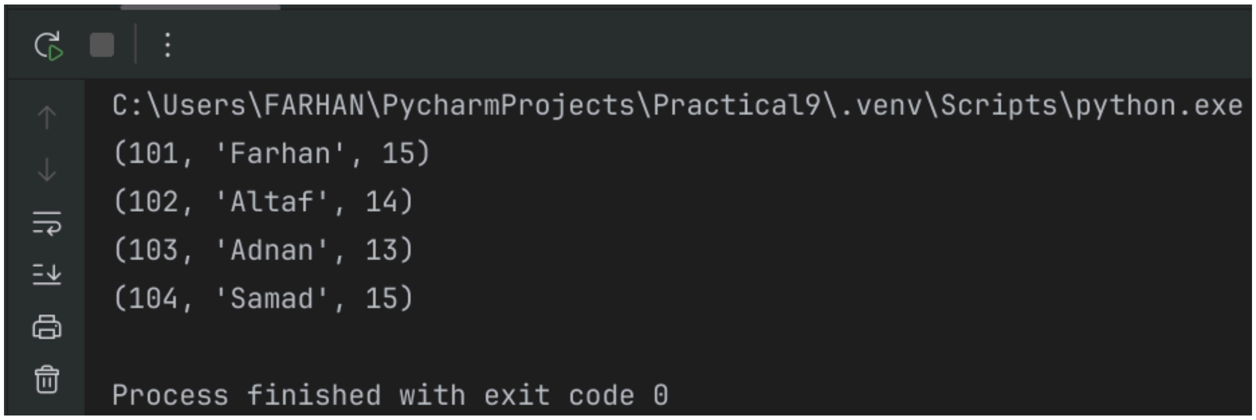
Source Code:

```
from dataclasses import dataclass

import mysql.connector

from Q1 import cursor


#Create connection object

db_connection = mysql.connector.connect(

     host="localhost",

     user = "root",

     password = "root",

     database = "employee_db"

)
```

```
cursor = db_connection.cursor()

#inserting values in table.

cursor.execute("INSERT INTO Students(id,name,age)
VALUES(101,'Farhan',15)")

cursor.execute("INSERT INTO Students(id,name,age)
VALUES(102,'Altaf',14)")

cursor.execute("INSERT INTO Students(id,name,age)
VALUES(103,'Adnan',13)")

cursor.execute("INSERT INTO Students(id,name,age)
VALUES(104,'Samad',15)")

db_connection.commit()


#Fetching all the records

cursor.execute("SELECT * FROM Students")

for x in cursor:

     print(x)


#Updating age of a student

cursor.execute("UPDATE Students SET age = 20 WHERE name =
'Farhan'")

db_connection.commit()

#Fetching all the records

cursor.execute("SELECT * FROM Students")

for x in cursor:

     print(x)
```

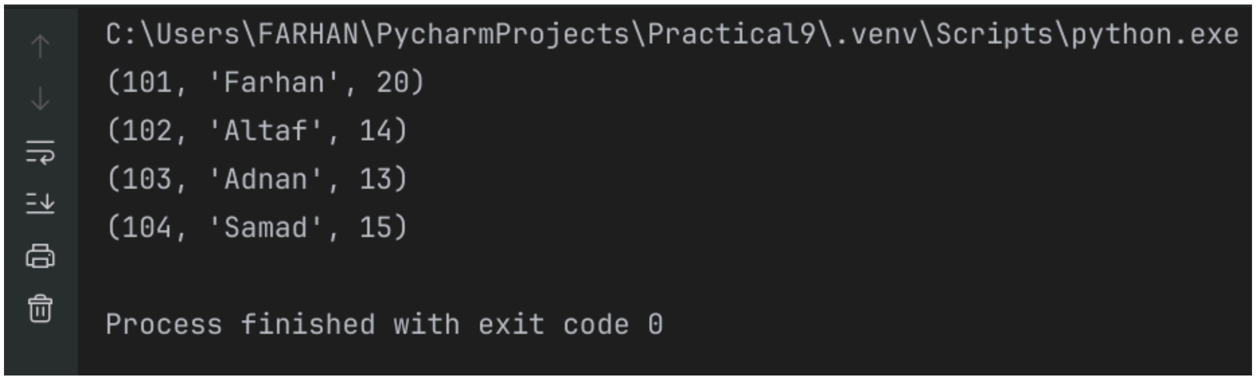
Output:

Initial Students Database:



```
C:\Users\FARHAN\PycharmProjects\Practical9\.venv\Scripts\python.exe
(101, 'Farhan', 15)
(102, 'Altaf', 14)
(103, 'Adnan', 13)
(104, 'Samad', 15)

Process finished with exit code 0
```

Database after updating age of Farhan:



```
C:\Users\FARHAN\PycharmProjects\Practical9\.venv\Scripts\python.exe
(101, 'Farhan', 20)
(102, 'Altaf', 14)
(103, 'Adnan', 13)
(104, 'Samad', 15)

Process finished with exit code 0
```

Conclusion:

From this Practical I have learned how to connect MySql Database with python with the help of mysql-connector module and perform operations like create,insert,update on the database.

Name of Student: Shaikh Farhan Ahmed			
Roll Number: 50		**LAB Assignment Number: 10**	
Title of LAB Assignment: To Implement Threads in Python			
DOP: 3 – 12 – 2024		**DOS: 4 – 12 – 2024**	
CO Mapped: CO6	**PO Mapped: PO3, PO5, PSO1, PSO2**	**Faculty Signature:**	**Marks:**

Practical No 10

AIM: To implement Threads in Python

Theory:

Thread:

A thread is an entity within a process that can be scheduled for execution. Also, it is the smallest unit of processing that can be performed in an OS (Operating System). In simple words, a thread is a sequence of such instructions within a program that can be executed independently of other code. For simplicity, you can assume that a thread is simply a subset of a process! A thread contains all this information in a Thread Control Block (TCB).

To create a thread in Python, you use the **threading.Thread** class. This class takes two arguments: the target function, which is the function that the thread will execute, and the args tuple, which is a tuple of arguments to be passed to the target function.

Once you have created a thread, you can start it using the start() method. This will cause the thread to begin executing the target function. To wait for a thread to finish executing, you can use the join() method. This will block the current thread until the target thread has finished executing.

Example-

```
import threading

def my_thread():

        print('Printing form thread!')

t = threading.Thread(target=my_thread)

t.start()

t.join()
```

Python Multithreading:

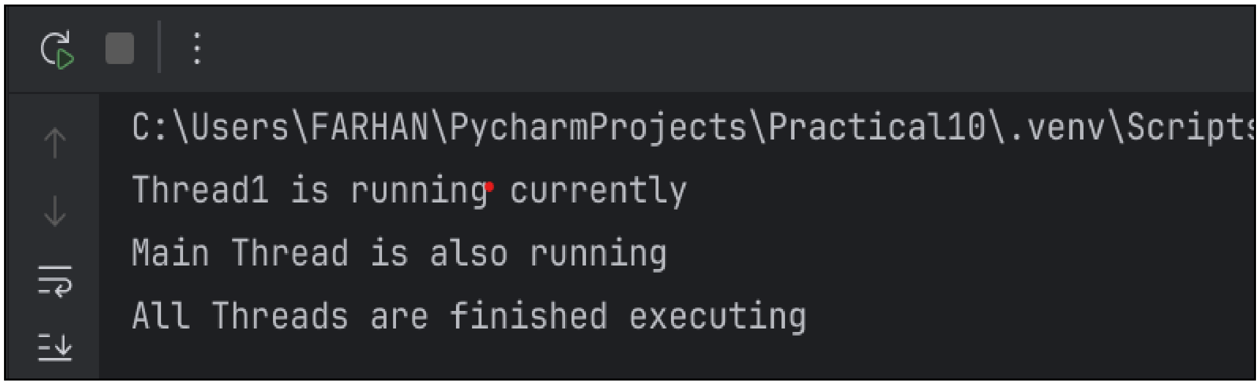
Multithreading is a stringing procedure in Python programming to run various strings simultaneously by quickly exchanging between strings with a central processor help (called setting exchanging). In addition, it makes it possible to share its data space with the primary threads of a process, making it easier than with individual processes to share information and communicate with other threads. The goal of multithreading is to complete multiple tasks at the same time, which improves application rendering and performance.

1. To do design the program for starting the thread in python

Source Code:

```
import threading

def Thread1():

    print("Thread1 is running currently")


#Initialize and start thread

init_thread = threading.Thread(target=Thread1())

init_thread.start()


#Main Thread

print("Main Thread is also running")

init_thread.join()

print("All Threads are finished executing")
```

Output:



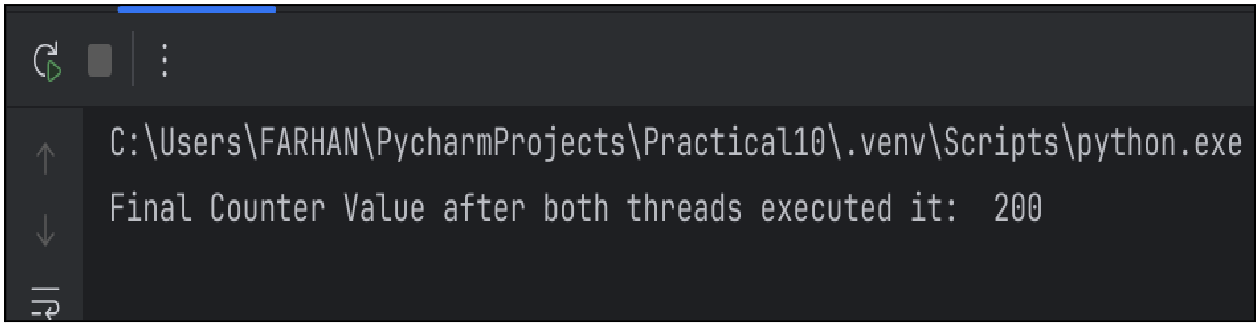
```
C:\Users\FARHAN\PycharmProjects\Practical10\.venv\Scripts
Thread1 is running currently
Main Thread is also running
All Threads are finished executing
```

2. Write a program to illustrate the concept of Synchronization

Source Code:

```
import threading

import threading as thread

counter = 0


counter_lock = thread.Lock()

#Increase counter safely

def increase_counter():

    global counter

    with counter_lock:

        for _ in range(100):

            counter = counter + 1


thread1 = threading.Thread(target=increase_counter())

thread2 = threading.Thread(target=increase_counter())

thread1.start()

thread2.start()


#waiting for both threads to execute

thread1.join()

thread2.join()

print("Final Counter Value after both threads executed it:
",counter)
```

Output:



```
C:\Users\FARHAN\PycharmProjects\Practical10\.venv\Scripts\python.exe
Final Counter Value after both threads executed it:  200
```

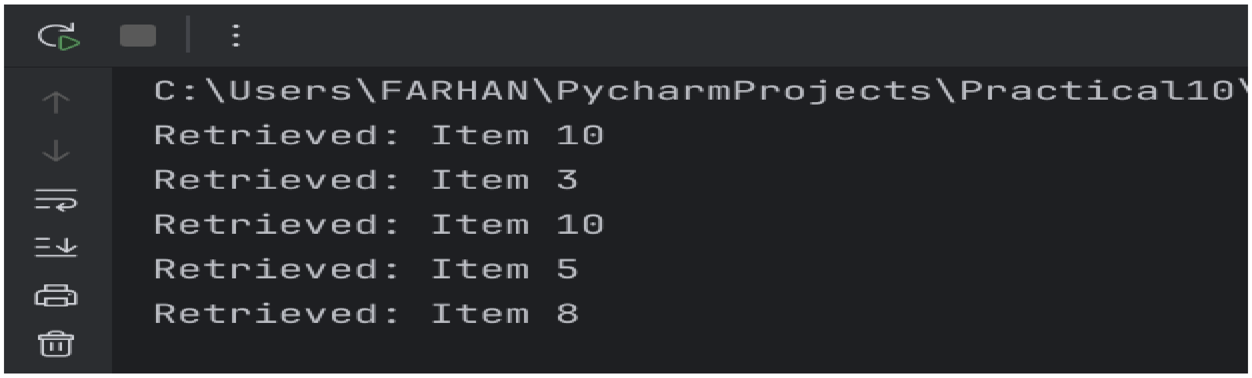
3. Write a program for creating multithreaded priority queue

Source Code:

```
import queue

import threading

import time

import random as rd


# Creating a priority queue

p_queue = queue.PriorityQueue()

stop = threading.Event()


def insert_into_queue():

     for i in range(5):

     item = rd.randint(1, 10)

     p_queue.put((item, f"Item {item}"))

     time.sleep(1)
```

```
def retrieve_from_queue():
     while not stop.is_set():
     try:
          item = p_queue.get(timeout=1)
          print(f"Retrieved: {item[1]}")
          p_queue.task_done()
     except queue.Empty:
          continue


insert_thread = threading.Thread(target=insert_into_queue)
retrieve_thread = threading.Thread(target=retrieve_from_queue)
insert_thread.start()
retrieve_thread.start()
insert_thread.join()
stop.set()
retrieve_thread.join()
```

Output:



```
C:\Users\FARHAN\PycharmProjects\Practical10
Retrieved: Item 10
Retrieved: Item 3
Retrieved: Item 10
Retrieved: Item 5
Retrieved: Item 8
```

Conclusion:

In this practical, we aimed to implement multithreading in Python using the threading` module. We created and managed multiple threads within a single process to achieve concurrent execution of code. Threads allow us to perform tasks concurrently, which is particularly useful for both CPU-bound and I/O-bound operations. We demonstrated how to create and start threads, run tasks concurrently, and coordinate their execution, all while ensuring that our program remains efficient and responsive through the use of Python's `threading` module.