INSTITUTE OF TECHNOLOGY AND MANAGEMENT
SKILLS UNIVERSITY,
KHARGHAR, NAVI MUMBAI

PYTHON PROGRAMMING LAB

Prepared by:

Name of Student: _____

Roll No: _____

Batch: 2023-27

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Exp. No	List of Experiment
1	1. Write a program to compute Simple Interest.
	2. Write a program to perform arithmetic, Relational operators.
	3. Write a program to find whether a given no is even & odd.
	4. Write a program to print first n natural number & their sum.
	5. Write a program to determine whether the character entered is a Vowel or not
	6. Write a program to find whether given number is an Armstrong Number.
	7. Write a program using for loop to calculate factorial of a No.
	1.8 Write a program to print the following pattern
	i) * * * * * * * * * * * * * * *
	ii) 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5

	iii)
	`    *`
	`  * * *`
	`* * * * *`
	`  * * * * * * *`
	`* * * * * * * * *`

2	2.1 Write a program that define the list of defines the list of define countries that are in BRICS.
	2.2 Write a program to traverse a list in reverse order. 1.By using Reverse method. 2.By using slicing
	2.3 Write a program that scans the email address and forms a tuple of username and domain.
	2.4 Write a program to create a list of tuples from given list having number and add its cube in tuple. i/p: c= [2,3,4,5,6,7,8,9]
	2.5 Write a program to compare two dictionaries in Python? (By using == operator)
	2.6 Write a program that creates dictionary of cube of odd numbers in the range.

	2.7 Write a program for various list slicing operation. a= [10,20,30,40,50,60,70,80,90,100] i. Print Complete list ii. Print 4th element of list iii. Print list from0th to 4th index. iv. Print list -7th to 3rd element v. Appending an element to list. vi. Sorting the element of list. vii. Popping an element. viii. Removing Specified element. ix. Entering an element at specified index. x. Counting the occurrence of a specified element. xi. Extending list. xii. Reversing the list.
3	3.1 Write a program to extend a list in python by using given approach. i. By using + operator. ii. By using Append () iii. By using extend ()
	3.2 Write a program to add two matrices.
	3.3 Write a Python function that takes a list and returns a new list with distinct elements from the first list.
	3.4 Write a program to Check whether a number is perfect or not.
	3.5 Write a Python function that accepts a string and counts the number of upper- and lower-case letters. string_test= 'Today is My Best Day'
4	4.1 Write a program to Create Employee Class & add methods to get employee details & print.

	4.2 Write a program to take input as name, email & age from user using combination of keywords argument and positional arguments (*args and**kwargs) using function,
	4.3 Write a program to admit the students in the different Departments(pgdm/ btech) and count the students. (Class, Object and Constructor).
	4.4 Write a program that has a class store which keeps the record of code and price of product display the menu of all product and prompt to enter the quantity of each item required and finally generate the bill and display the total amount.
	4.5 Write a program to take input from user for addition of two numbers using (single inheritance).
	4.6 Write a program to create two base classes LU and ITM and one derived class. (Multiple inheritance).
	4.7 Write a program to implement Multilevel inheritance, Grandfather?Father-?Child to show property inheritance from grandfather to child.
	4.8 Write a program Design the Library catalogue system using inheritance take base class (library item) and derived class (Book, DVD & Journal) Each derived class should have unique attribute and methods and system should support Check in and check out the system. (Using Inheritance and Method overriding)
5	5.1 Write a program to create my_module for addition of two numbers and import it in main script.
	5.2 Write a program to create the Bank Module to perform the operations such as Check the Balance, withdraw and deposit the money in bank account and import the module in main file.
	5.3 Write a program to create a package with name cars and add different modules (such as BMW, AUDI, NISSAN) having classes and functionality and import them in main file cars.

6	6.1 Write a program to implement Multithreading. Printing “Hello” with one thread & printing “Hi” with another thread.
7.	7.1 Write a program to use ‘whether API’ and print temperature of any city, also print the sunrise and sunset times for the same humidity of that area.
	7.2 Write a program to use the ‘API’ of crypto currency.

Name of Student: Bhagyashree Bhagat

Roll Number: **34**

Experiment No:(1.1)

Title:Write a program to compute Simple Interest.

Theory:

Simple Interest is a basic financial concept used to calculate the interest earned or paid on a principal amount over a period of time. It is commonly used in various financial transactions such as loans and investments.

The formula for calculating Simple Interest (SI) is:

$$SI = P \times R \times T / 100$$

Where:

- P **is the principal amount (initial amount of money).**
- R **is the rate of interest per time period.**
- T **is the time duration for which the money is borrowed or invested (in years).**

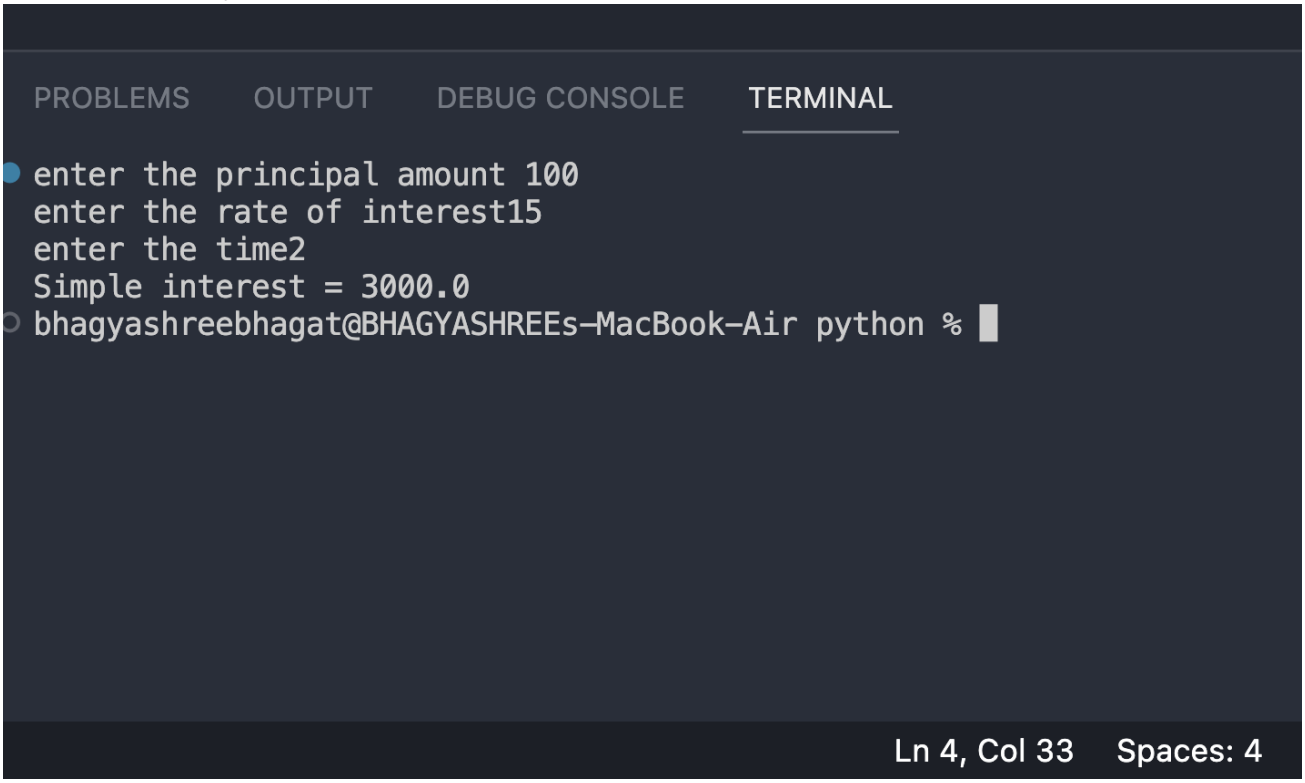
Code:

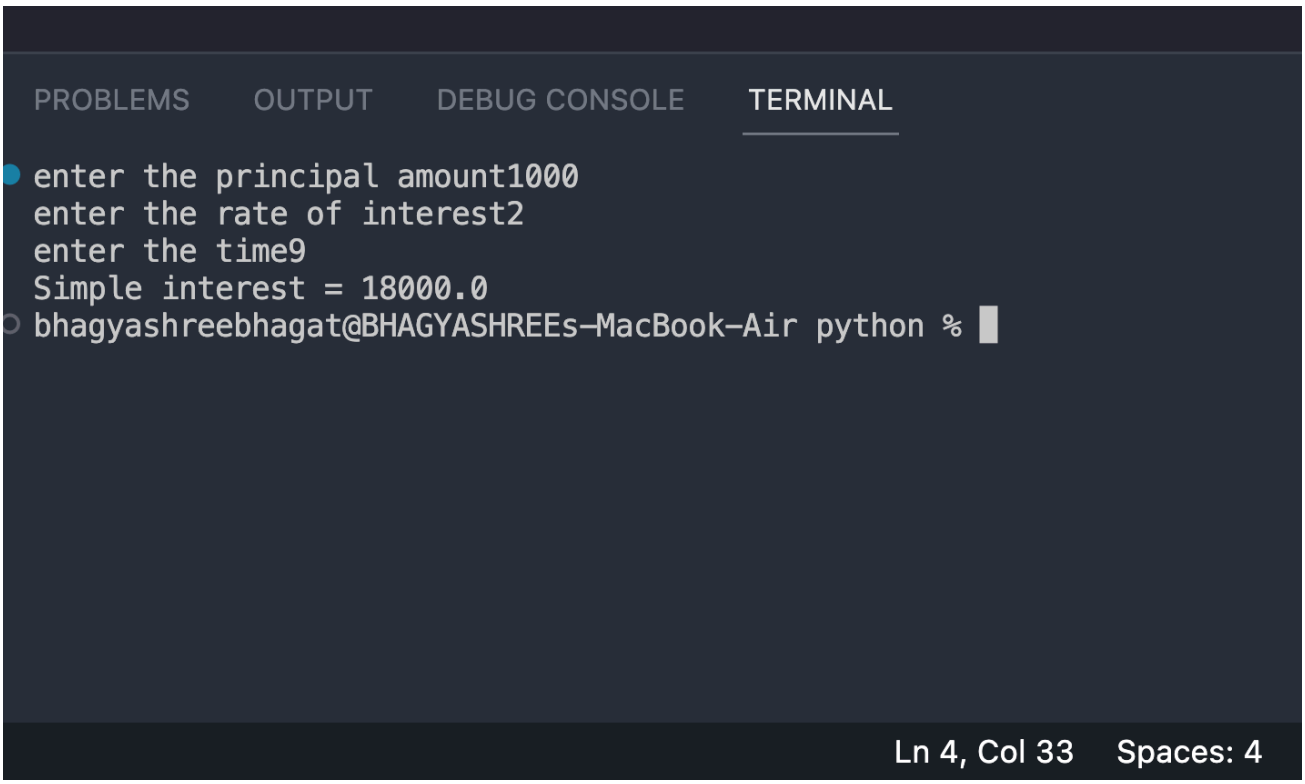
```
p=float(input("enter the principal amount"))
r=float(input("enter the rate of interest"))
t=float(input("enter the time"))
```

```
print("Simple interest =",p*r*t)
```

Output: (screenshot)

Test Case: Any two (screenshot)





Conclusion:

The program allows the user to input the principal amount, rate of interest, and time period. It then calculates the Simple Interest using the provided formula. The result is displayed to the user.

In real-world scenarios, understanding and calculating Simple Interest are essential for financial planning, loan management, and investment decisions. This simple Python program serves as a practical example of implementing a financial formula and taking user input for computation. It can be extended to handle more complex financial calculations and incorporate error handling for improved robustness.

Name of Student: Bhagyashree Bhagat

Roll Number: 34

Experiment No:1(1.2)

Title:Write a program to perform arithmetic, Relational operators.

Theory:

Arithmetic Operators:

Arithmetic operators are used for performing mathematical operations between two operands. In Python, the commonly used arithmetic operators include:

- \+ **(Addition): Adds two operands.**

- `-` (Subtraction): Subtracts the right operand from the left operand.
- `*` (Multiplication): Multiplies two operands.
- `/` (Division): Divides the left operand by the right operand.
- `%` (Modulus): Returns the remainder of the division of the left operand by the right operand.
- `//` (Floor Division): Returns the quotient of the division rounded down to the nearest integer.
- `**` (Exponent): Raises the left operand to the power of the right operand.

Relational Operators:

Relational operators are used to compare two values or expressions. They return either `True` or `False` based on the comparison. In Python, relational operators include:

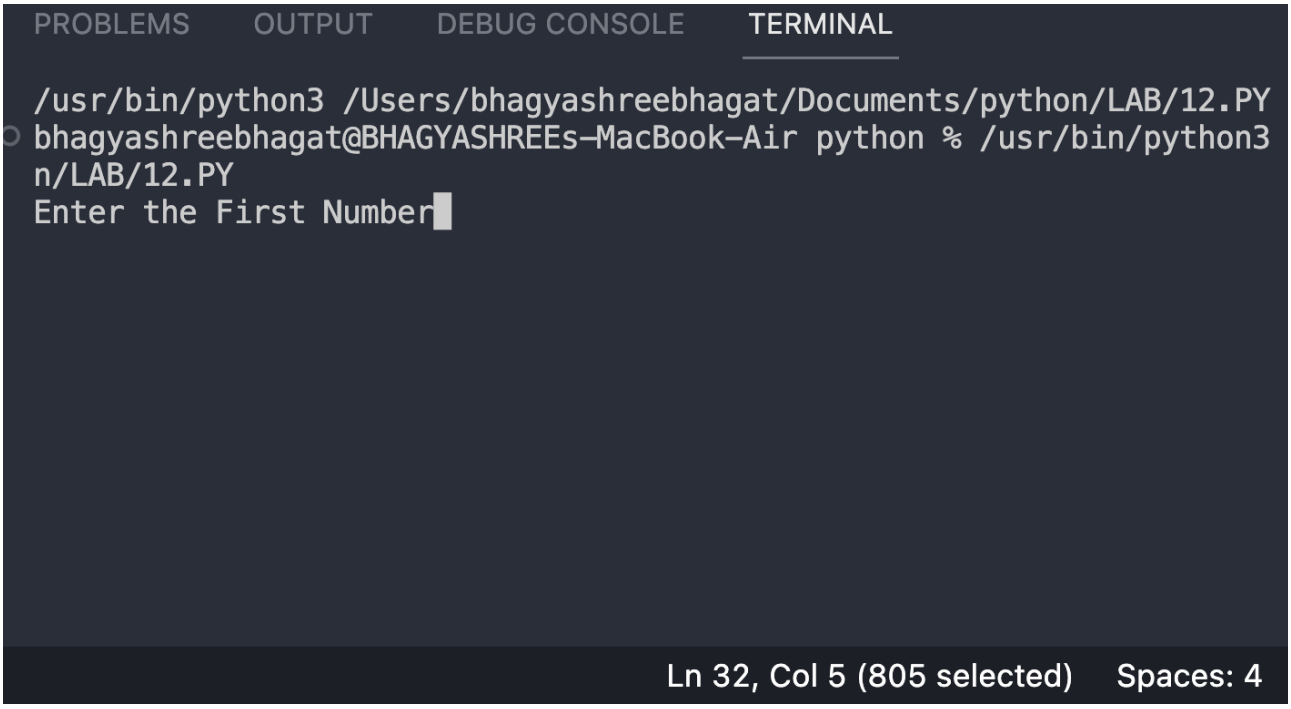
- `==` (Equal to): Returns `True` if the values of the two operands are equal.
- `!=` (Not equal to): Returns `True` if the values of the two operands are not equal.
- `<` (Less than): Returns `True` if the left operand is less than the right operand.
- `>` (Greater than): Returns `True` if the left operand is greater than the right operand.
- `<=` (Less than or equal to): Returns `True` if the left operand is less than or equal to the right operand.
- `>=` (Greater than or equal to): Returns `True` if the left operand is greater than or equal to the right operand.

Code:

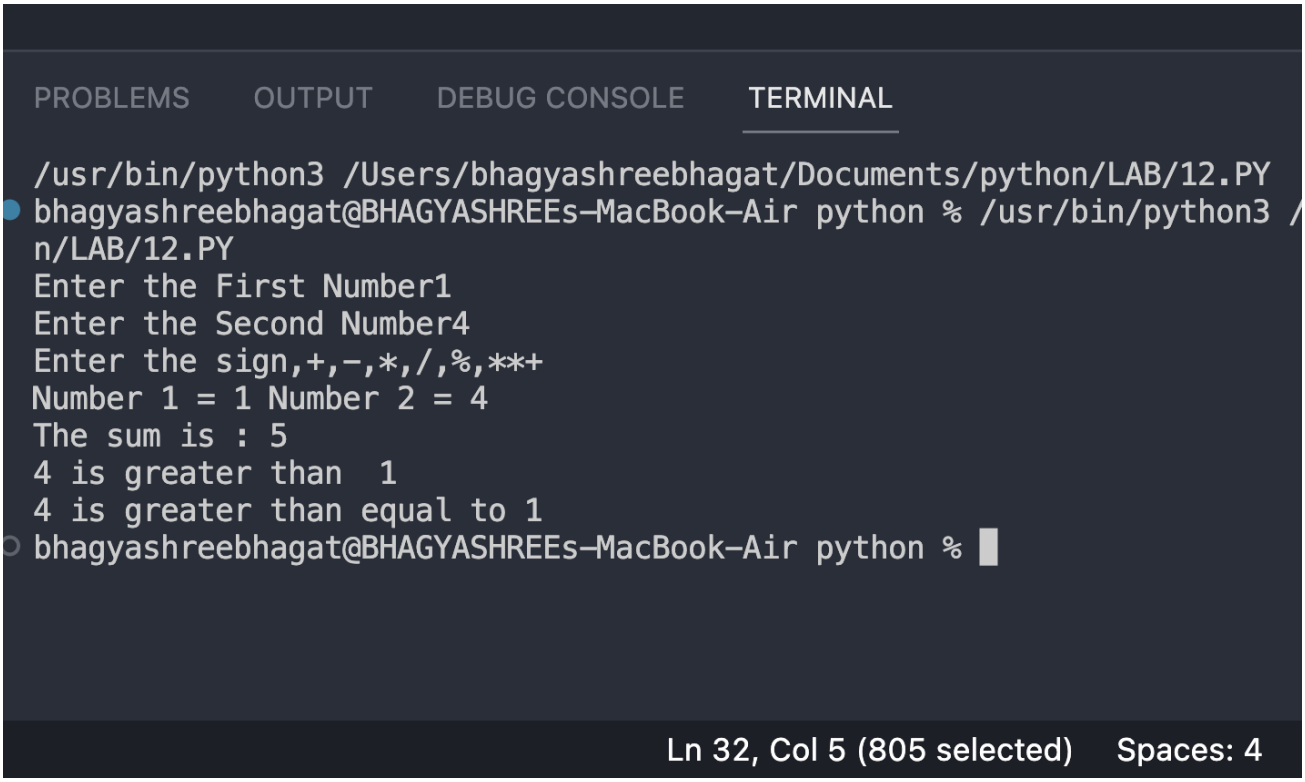
```
a=int(input("Enter the First Number"))

b=int(input("Enter the Second Number"))



userinput=input("Enter the sign,+,-,*,/,%,**")

print("Number 1 =",a,"Number 2 =",b)



if userinput == "+" :

   print("The sum is :",a+b)
```

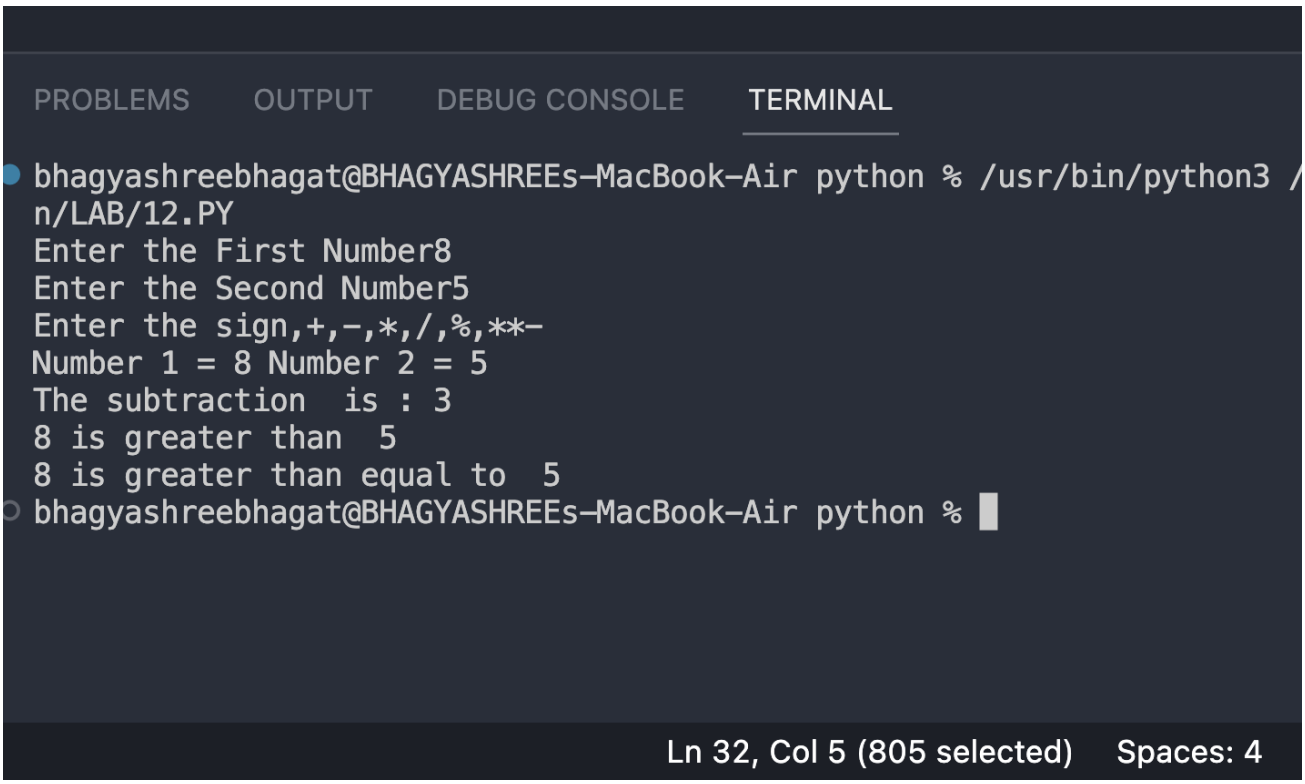
```
elif userinput=="-":
   print("The subtraction  is :",a-b)
elif userinput=="*":
   print("The multiplication is :",a*b)
elif userinput=="/":
   print("The division is :",a/b)
elif userinput=="%":
   print("The modulus is :",a%b)
elif userinput=="**":
   print("The Exponent is :",a**b)
if a<b:
   print(b,"is greater than ",a)
elif a>b:
   print(a,"is greater than ",b)
elif a==b:
       print(a,"is equal to ",b)
elif a!=b:
        print(a,"is not equal to ",b)
if a<=b:
   print(b,"is greater than equal to",a)
elif a>=b:
   print(a,"is greater than equal to ",b)
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:This program demonstrates the use of arithmetic and relational operators in Python. The arithmetic operators perform various mathematical operations on numerical values, while the relational operators compare values and return `True` or `False` based on the specified

condition. Understanding and using these operators are fundamental for performing mathematical calculations and making logical decisions in programming.

Name of Student: Bhagyashree Bhagat

Roll Number: 34

Experiment No:1(1.3)

Title:

Write a program to find whether a given no is even & odd.

Theory:

In mathematics, even and odd numbers are types of integers. An integer is considered even if it is divisible by 2 without a remainder, and it is considered odd if dividing it by 2 leaves a remainder of 1. In Python, you can determine whether a given number is even or odd using the modulo operator (%), which returns the remainder of the division.

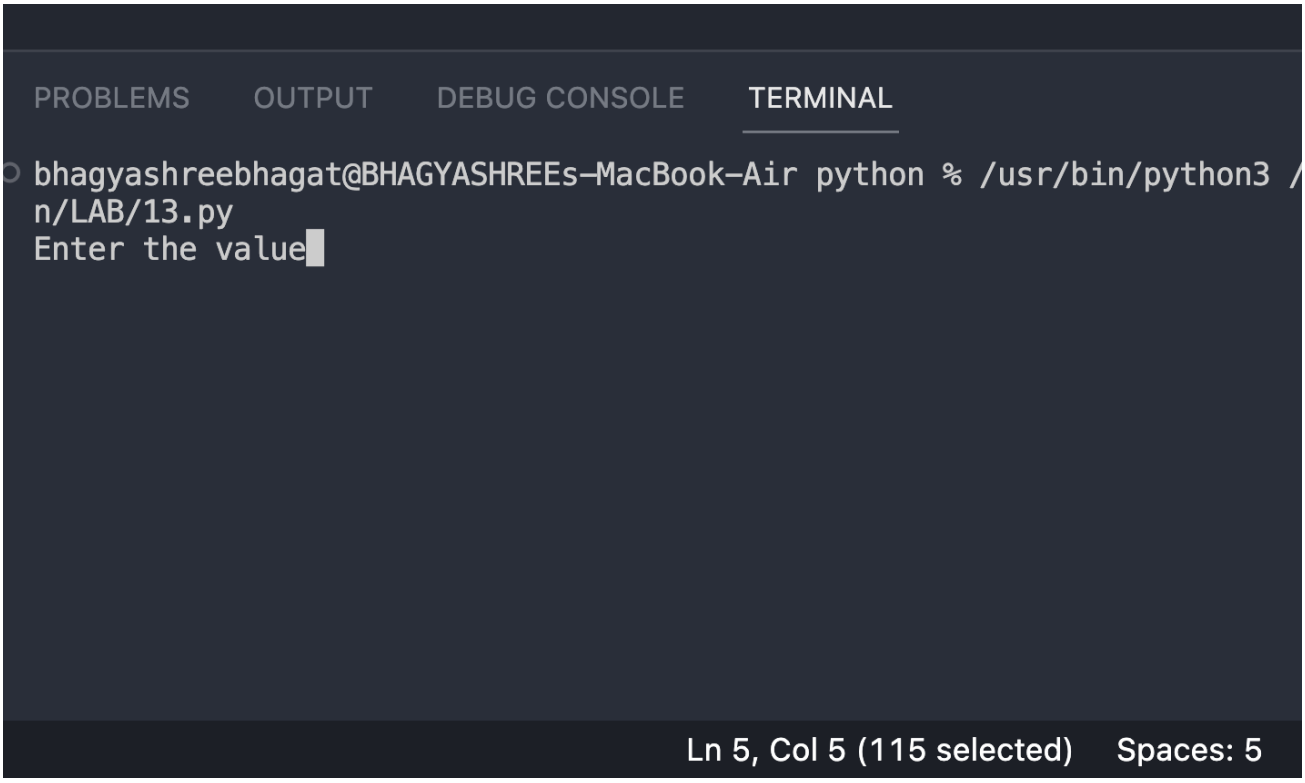
Here's how the logic works:

- **If `number % 2 == 0`, the number is even because there is no remainder when divided by 2.**
- **If `number % 2 == 1`, the number is odd because there is a remainder of 1 when divided by 2.**

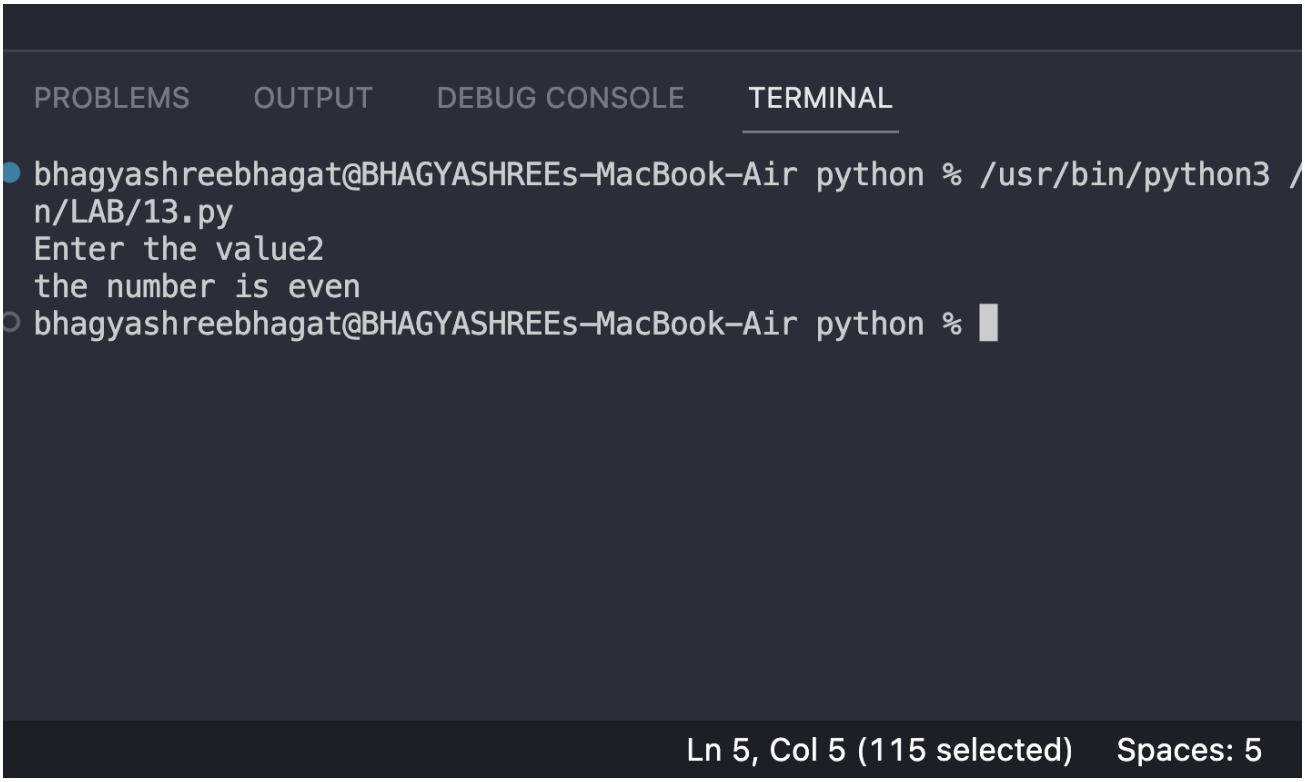
Code:

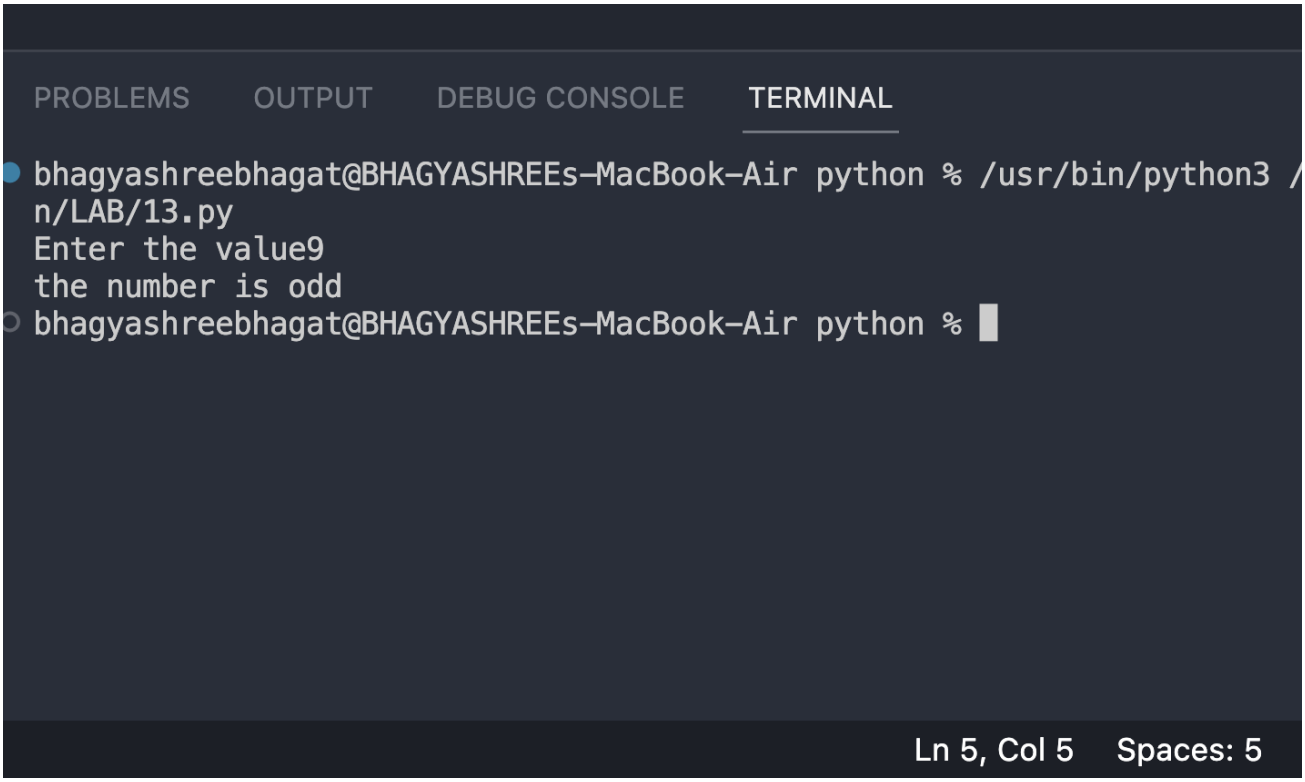
```
a=int((input("Enter the value")))

if a%2==0: print("the number is even")

else:

    print("the number is odd")
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

The program efficiently determines whether a given number is even or odd by utilizing the modulo operator. The logic is straightforward and relies on the fundamental properties of even and odd numbers. This example illustrates the use of basic arithmetic operations and conditional statements in Python.

Name of Student: Bhagyashree Bhagat

Roll Number: 34

Experiment No:1(1.4)

Title: Print First n Natural Numbers and Their Sum:

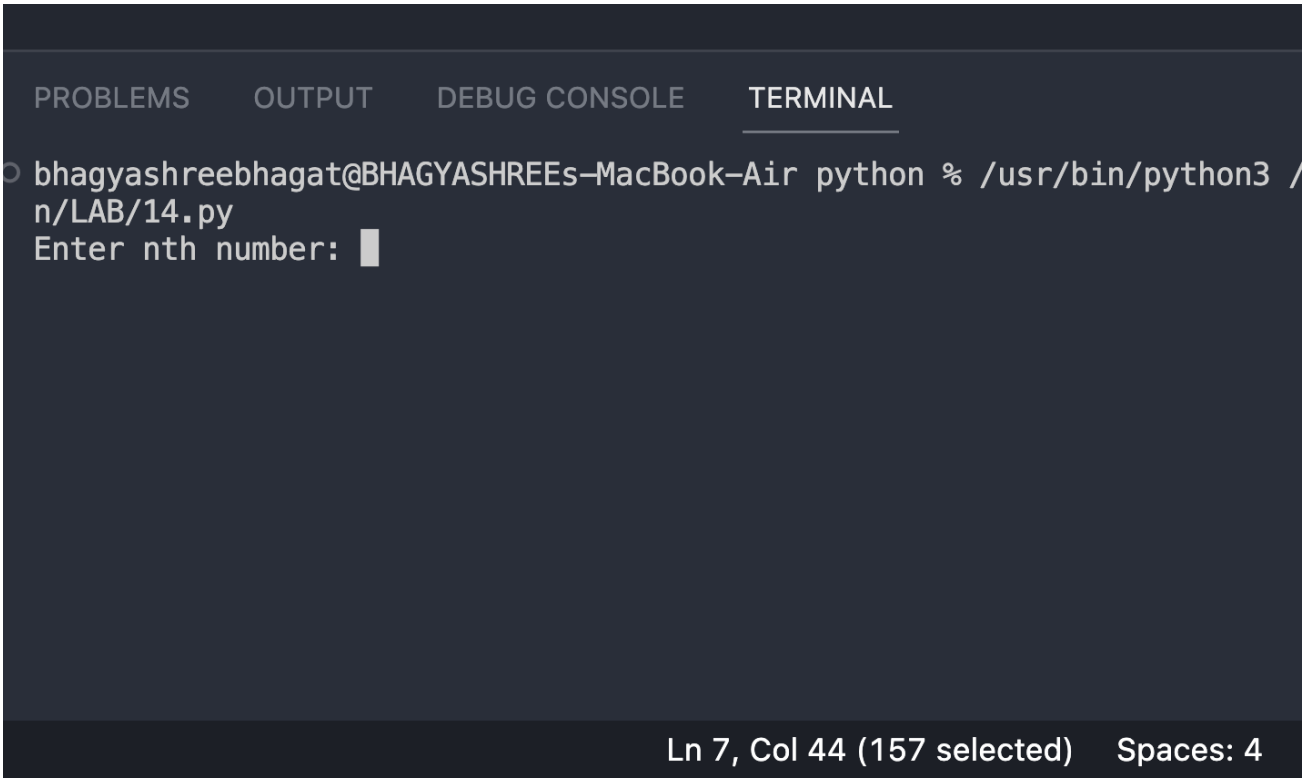
Theory:

The program prints the first n natural numbers and calculates their sum using the formula: `Sum = n * (n + 1) / 2`**.**

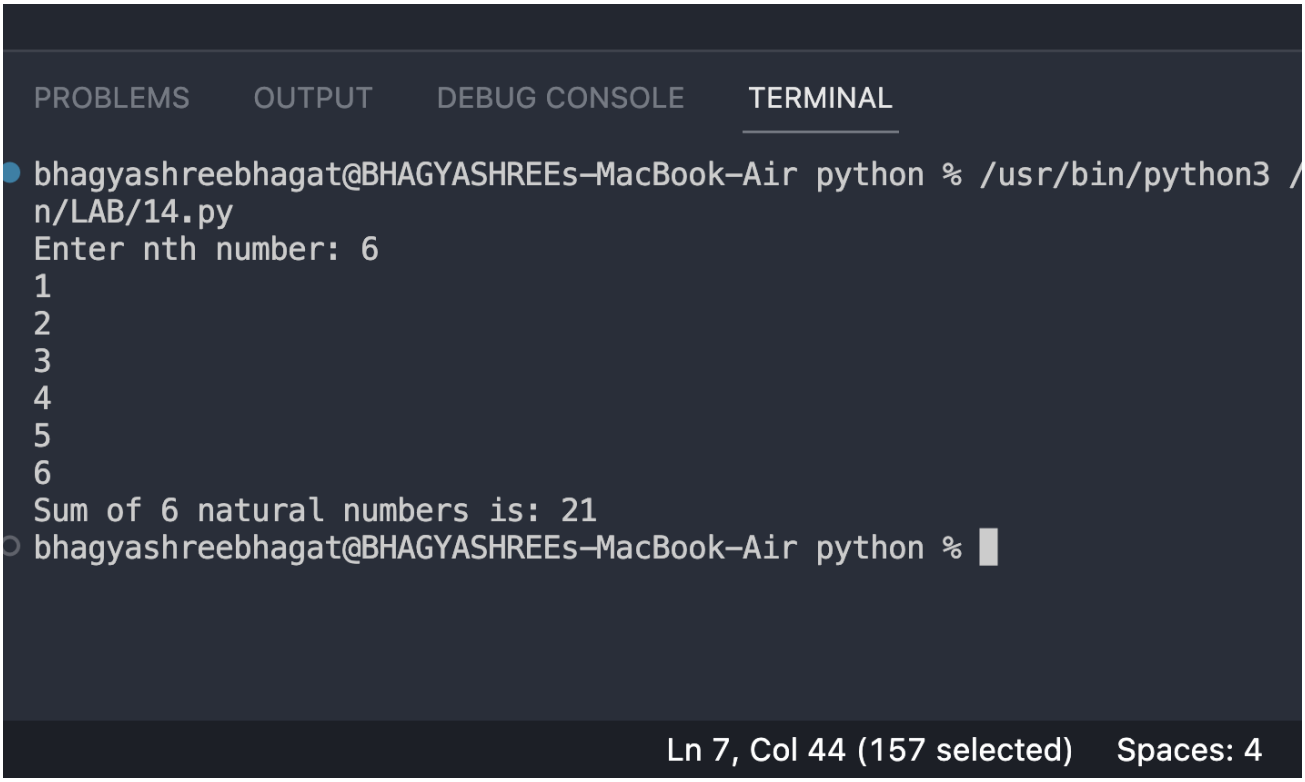
Code:

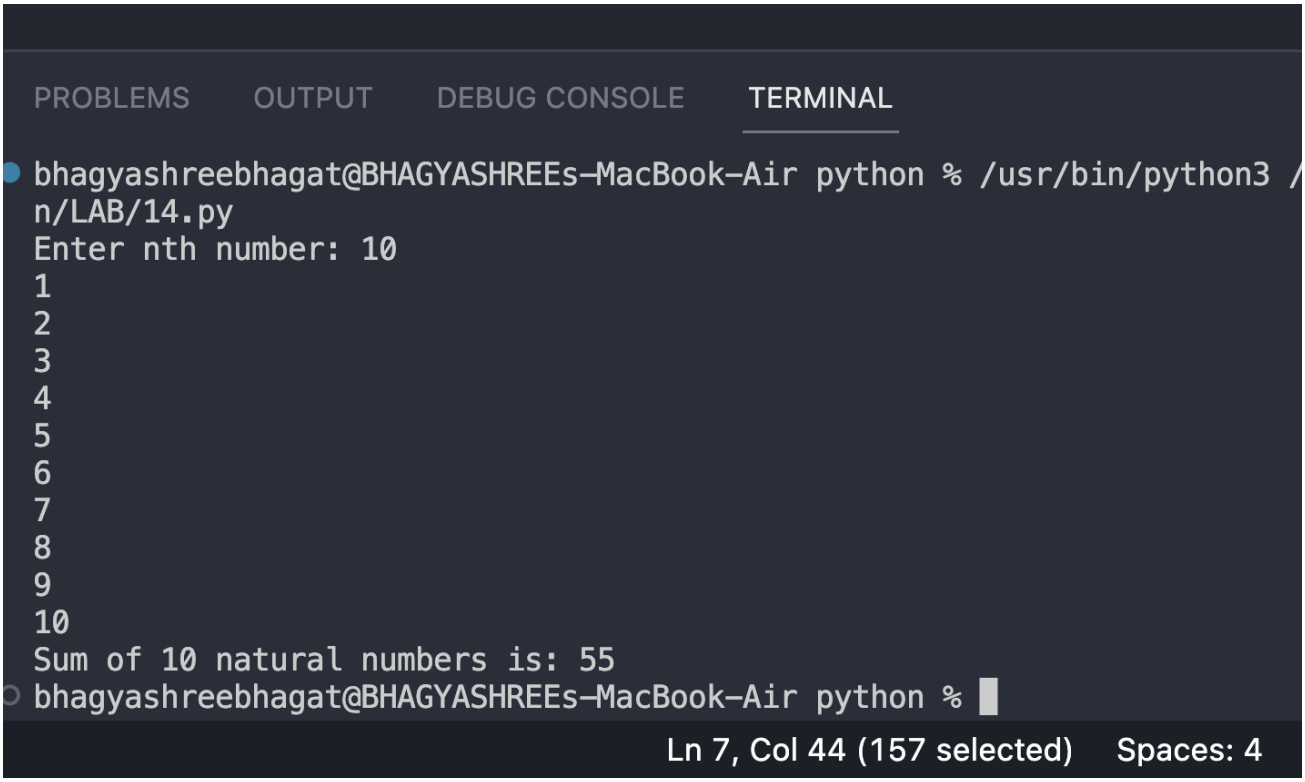
```
n = int(input("Enter nth number: "))

sum=0

for i in range(1,n+1):

   if(i>0):

       print(i)

       sum=sum+i

print("Sum of",n,"natural numbers is:",sum)
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

It provides a way to display natural numbers up to a specified limit and calculates their sum, demonstrating basic mathematical concepts.

Name of Student: __Bhagyashree Bhagat_____

Roll Number: 34_____

Experiment No:1(1.5)

Title:

Determine Whether a Character is a Vowel:

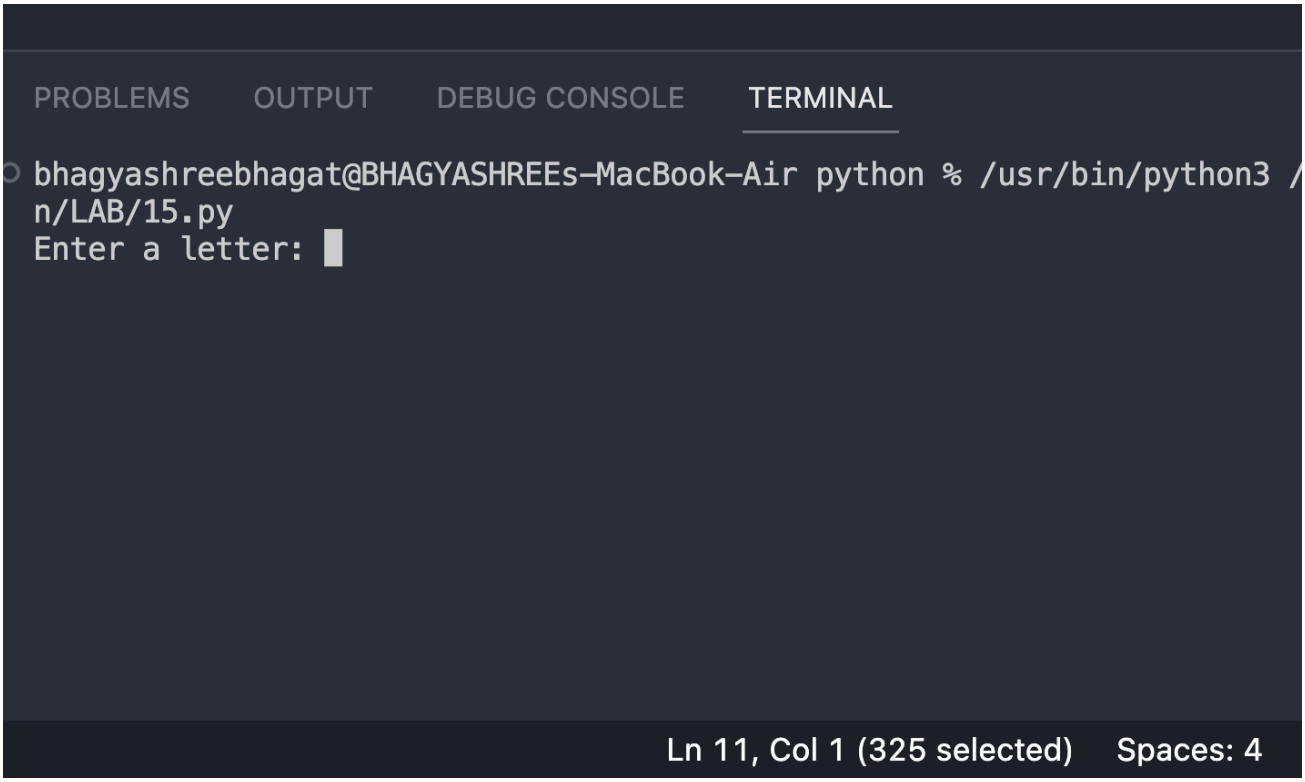
Theory:

The program checks if a given character is a vowel. It evaluates the character against a predefined set of vowels.

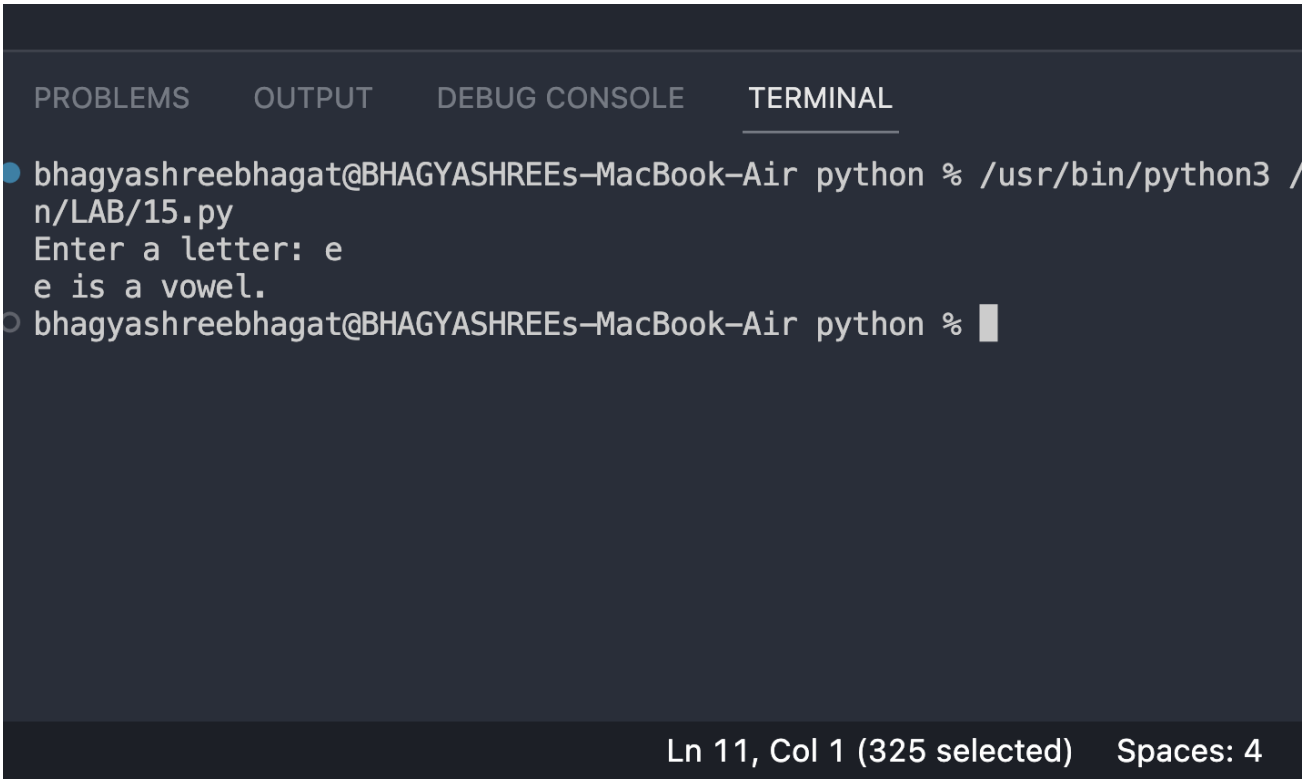
Code:

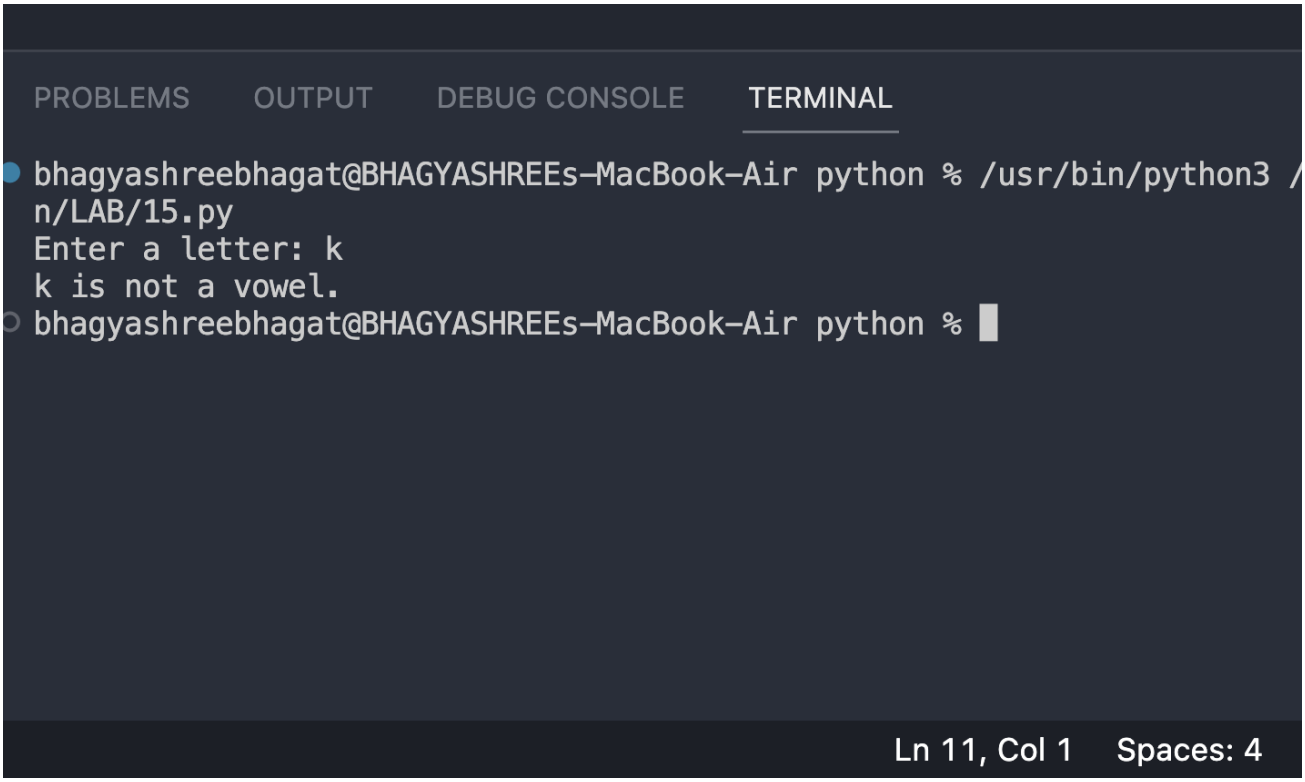
```
user_input = input("Enter a letter: ")

if len(user_input) == 1 and user_input.isalpha():



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

   if user_input.lower() in vowels:

       print(f"{user_input} is a vowel.")

   else:

       print(f"{user_input} is not a vowel.")

else:

   print("Please enter a valid single letter.")
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

It illustrates a simple character-checking logic and is useful for tasks involving character classification.

Name of Student: Bhagyashree Bhagat

Roll Number: 34

Experiment No:1.6

Title:1.6 Check Whether a Number is an Armstrong Number:

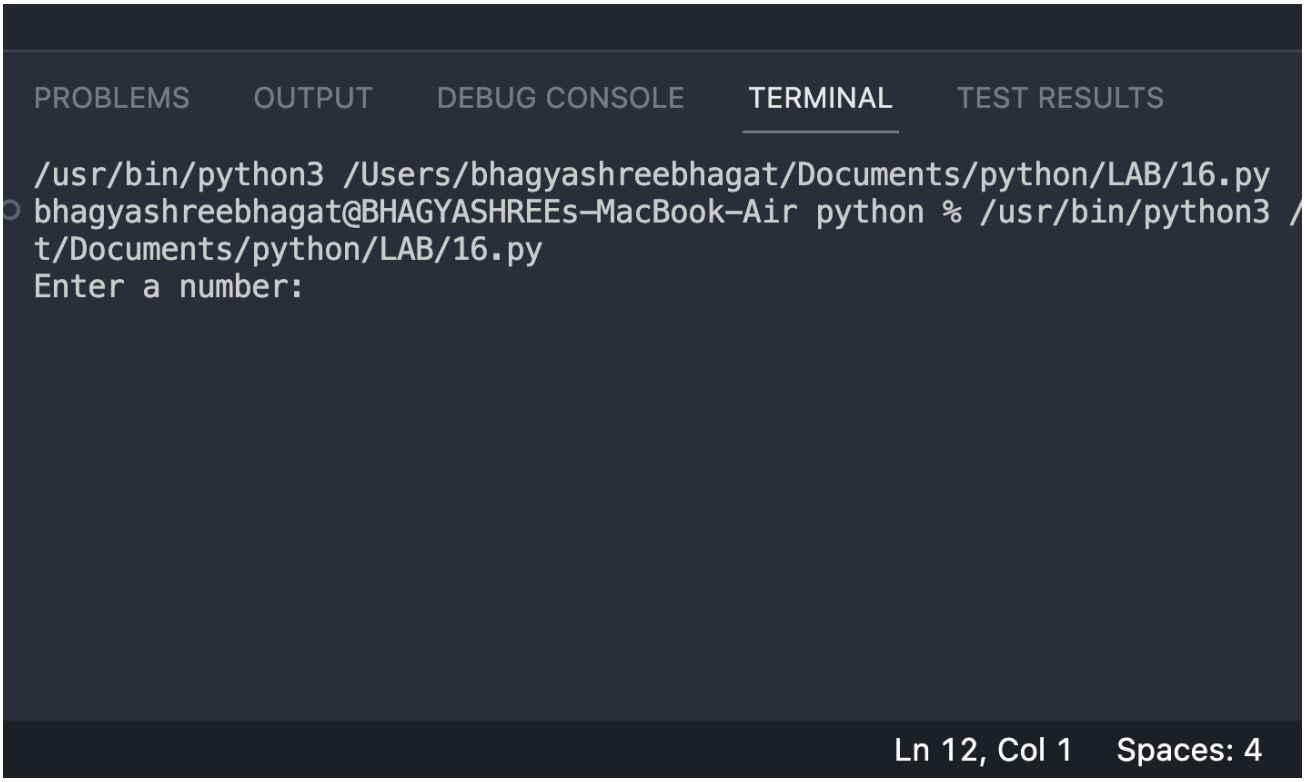
Theory:

The program determines whether a given number is an Armstrong number. Armstrong numbers have a special property where the sum of cubes of their digits equals the number itself.

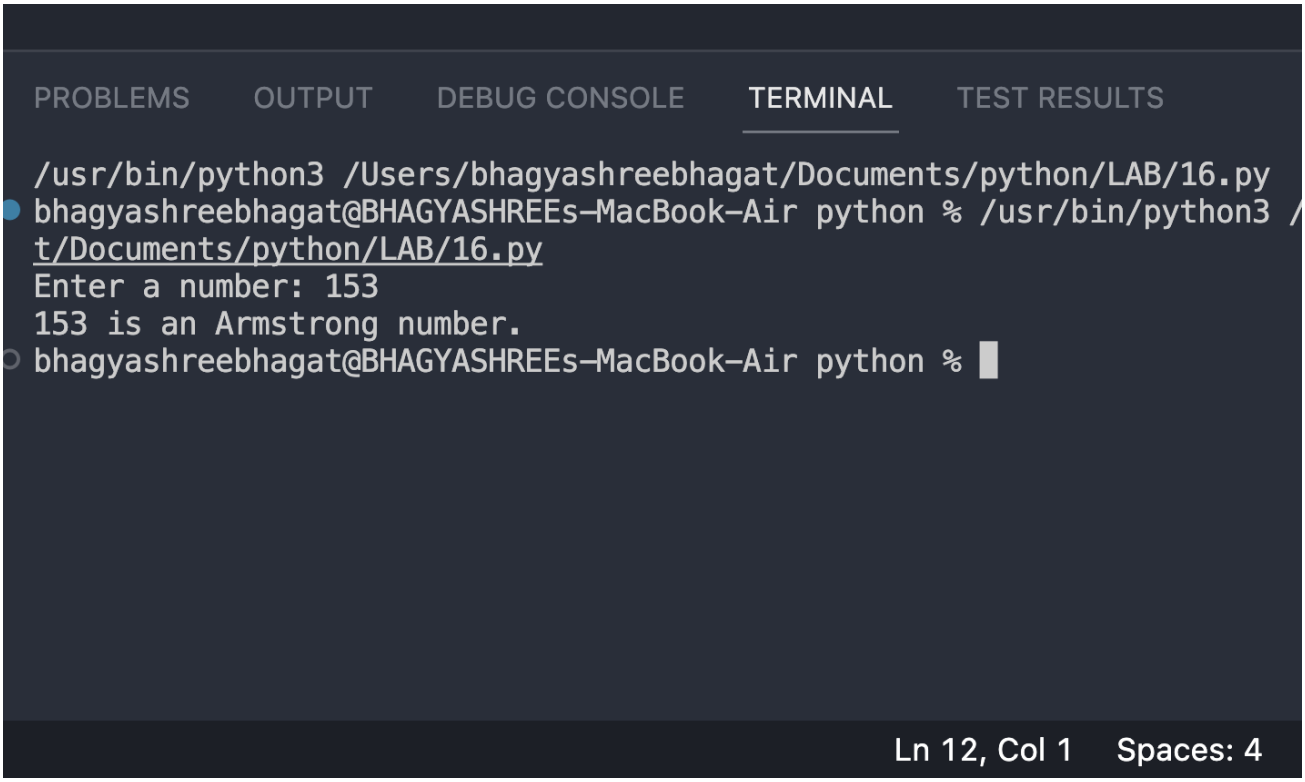
Code:

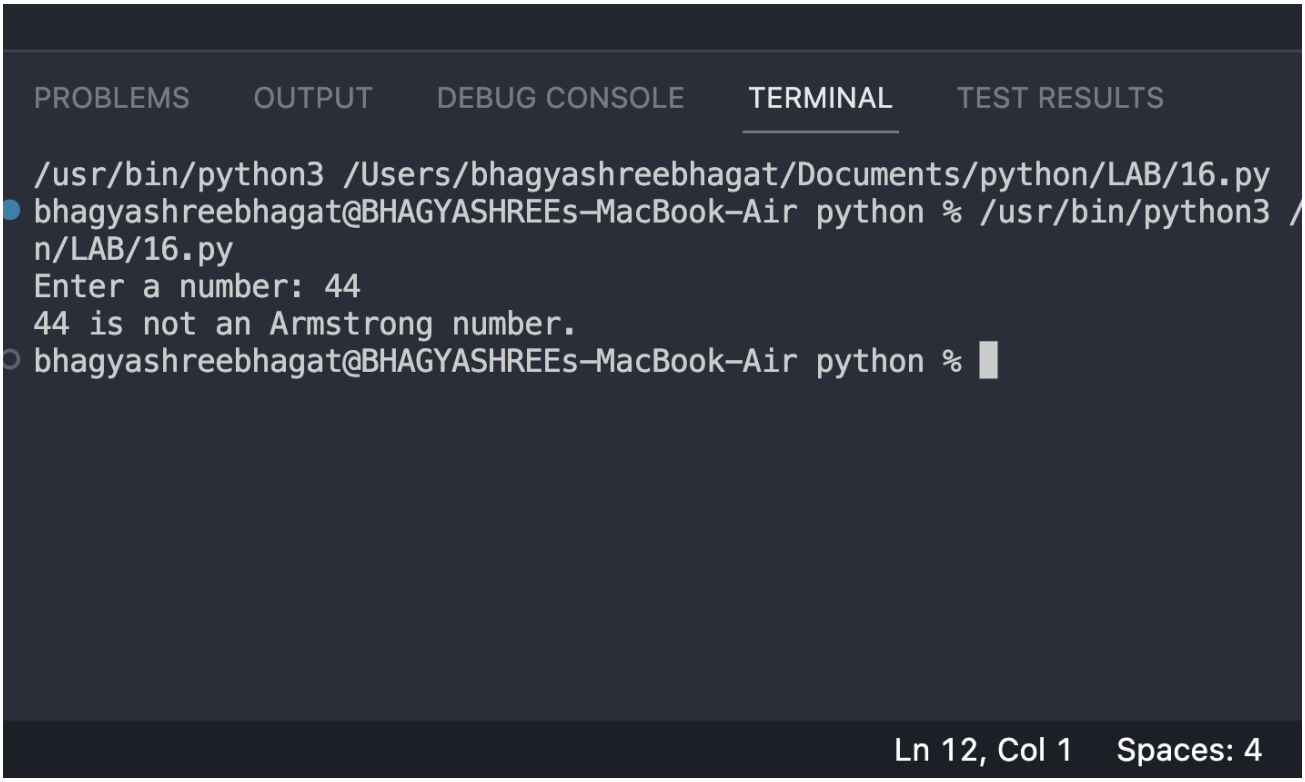
```
num = int(input("Enter a number: "))



num_str = str(num)

sum_of_powers = sum(int(digit) ** len(num_str) for digit in num_str)

if sum_of_powers == num:

   print(f"{num} is an Armstrong number.")

else:

   print(f"{num} is not an Armstrong number.")
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

It provides a method to identify Armstrong numbers, showcasing a specific type of numerical property.

Name of Student: Bhagyashree Bhagat

Roll Number: 34

Experiment No:1.7

Title:

Calculate Factorial Using a For Loop:

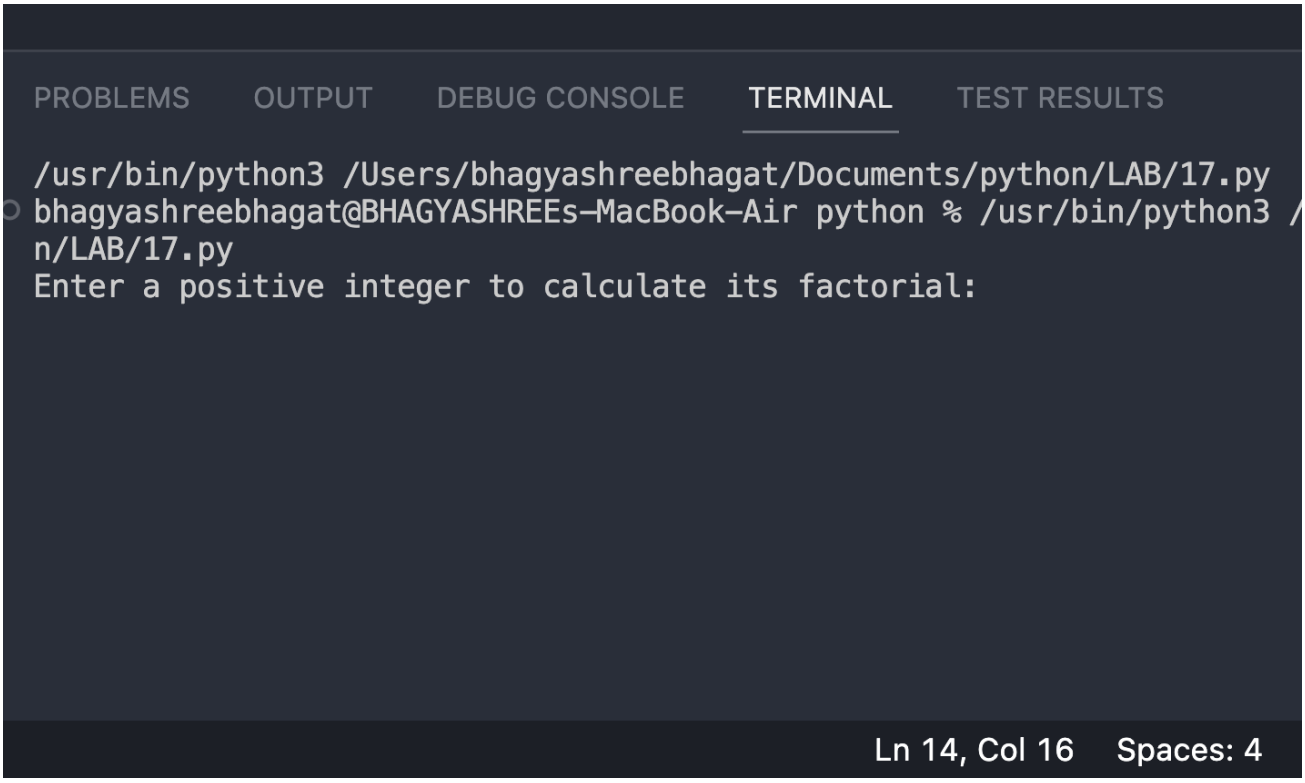
Theory:

The program calculates the factorial of a number using a for loop. Factorial is the product of all positive integers up to a given number.

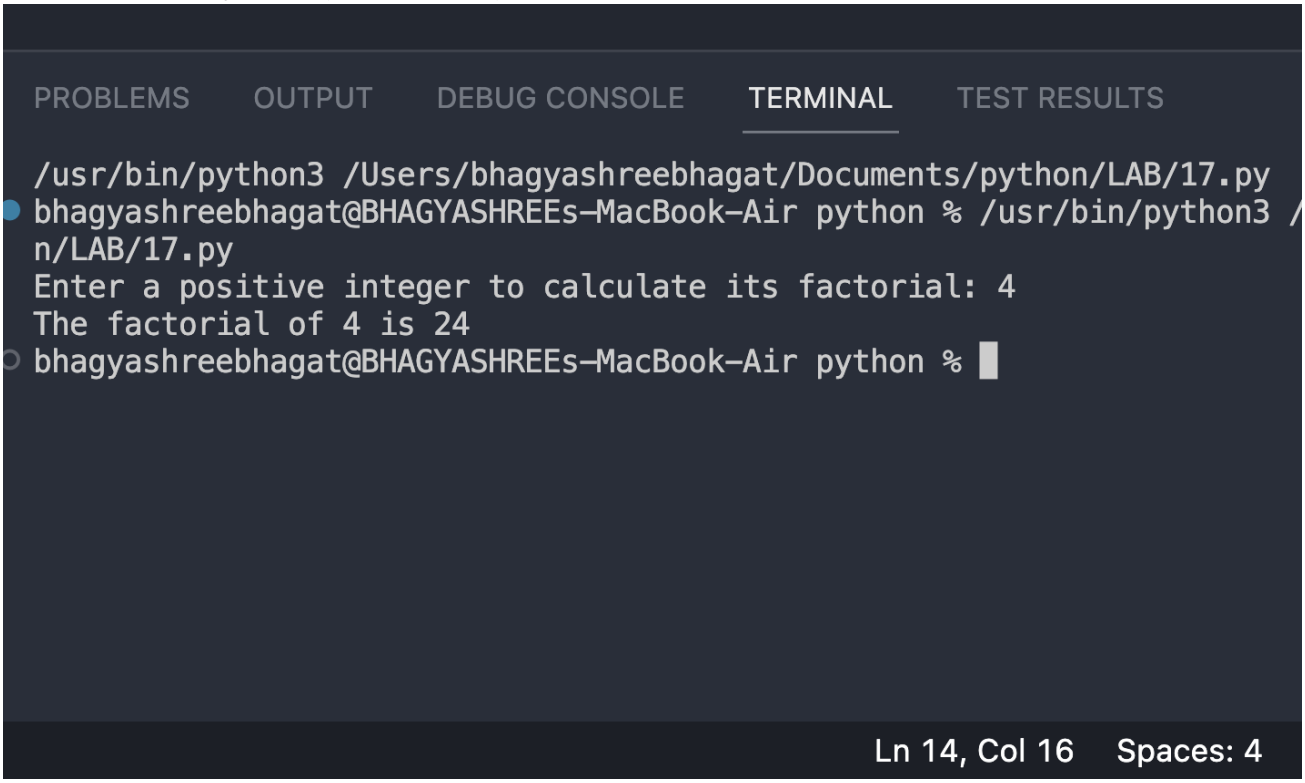
Code:

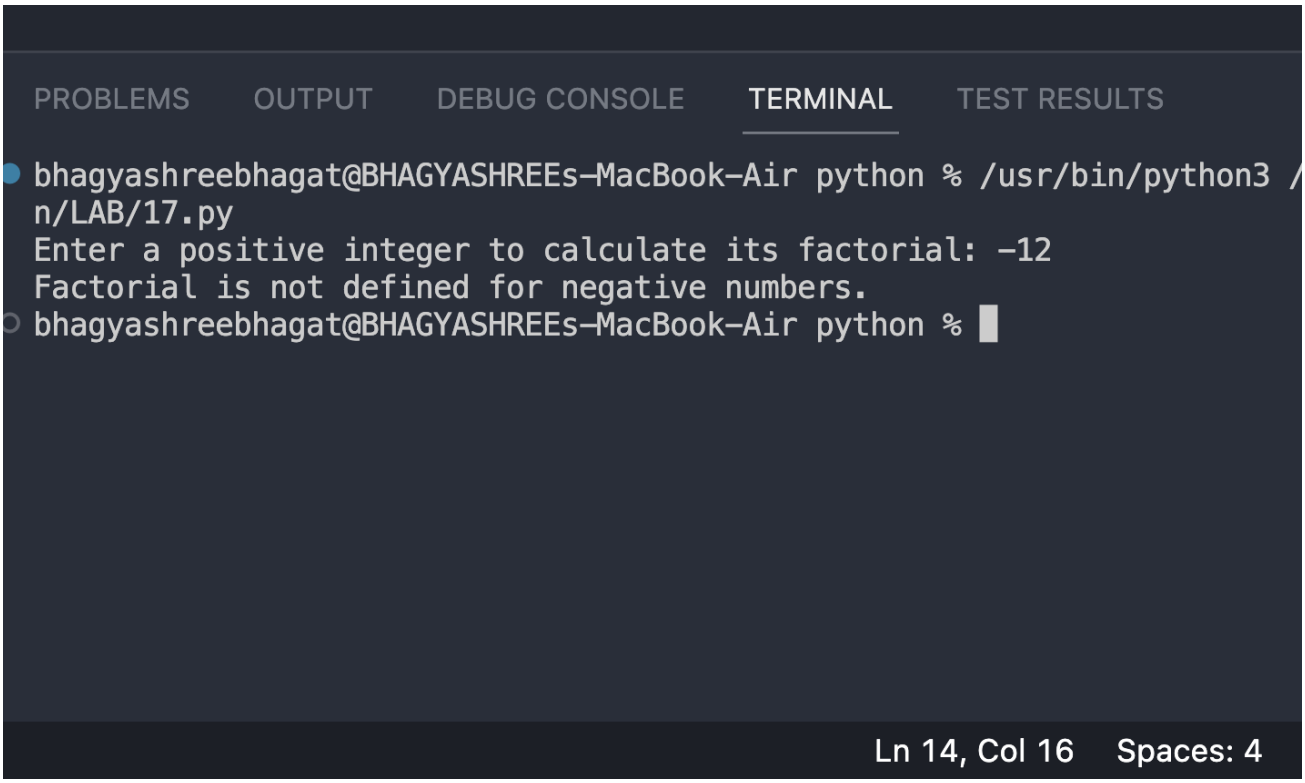
```
def factorial(number):
    fact = 1
    if number < 0:
        print("Factorial is not defined for negative numbers.")
    elif number == 0:
        print("The factorial of 0 is 1")
    else:
        for i in range(1, number + 1):
            fact*= i

        print(f"The factorial of {number} is {fact}")

user= int(input("Enter a positive integer to calculate its factorial: "))
factorial(user)
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

It highlights the use of nested loops to create visual patterns and is a common exercise in programming courses.

Name of Student: __Bhagyashree Bhagat_____

Roll Number: 34_____

Experiment No:2.1

Title:Define a List of Countries in BRICS:

Theory:

This program creates a list of countries that are part of the BRICS alliance.

Code:

```
brics_countries = ["Brazil", "Russia", "India", "China", "South Africa"]



for country in brics_countries:

   print(country)
```

Output: (screenshot)

```
brics_countries = ["Brazil", "Russia", "India", "China", "South Africa"]


for country in brics_countries:

   print(country)
```

Test Case: Any two (screenshot)

```
brics_countries = ["Brazil", "Russia", "India", "China", "South Africa"]


for country in brics_countries:

   print(country)
```

```
brics_countries = ["Brazil", "Russia", "India", "China", "South Africa"]


for country in brics_countries:

   print(country)
```

Conclusion:

It shows how to define and initialize a list in Python, specifically containing the countries in the BRICS group.

Name of Student: __Bhagyashree Bhagat_____

Roll Number: 34_____

Experiment No:2.2

Title:Traverse a List in Reverse Order:

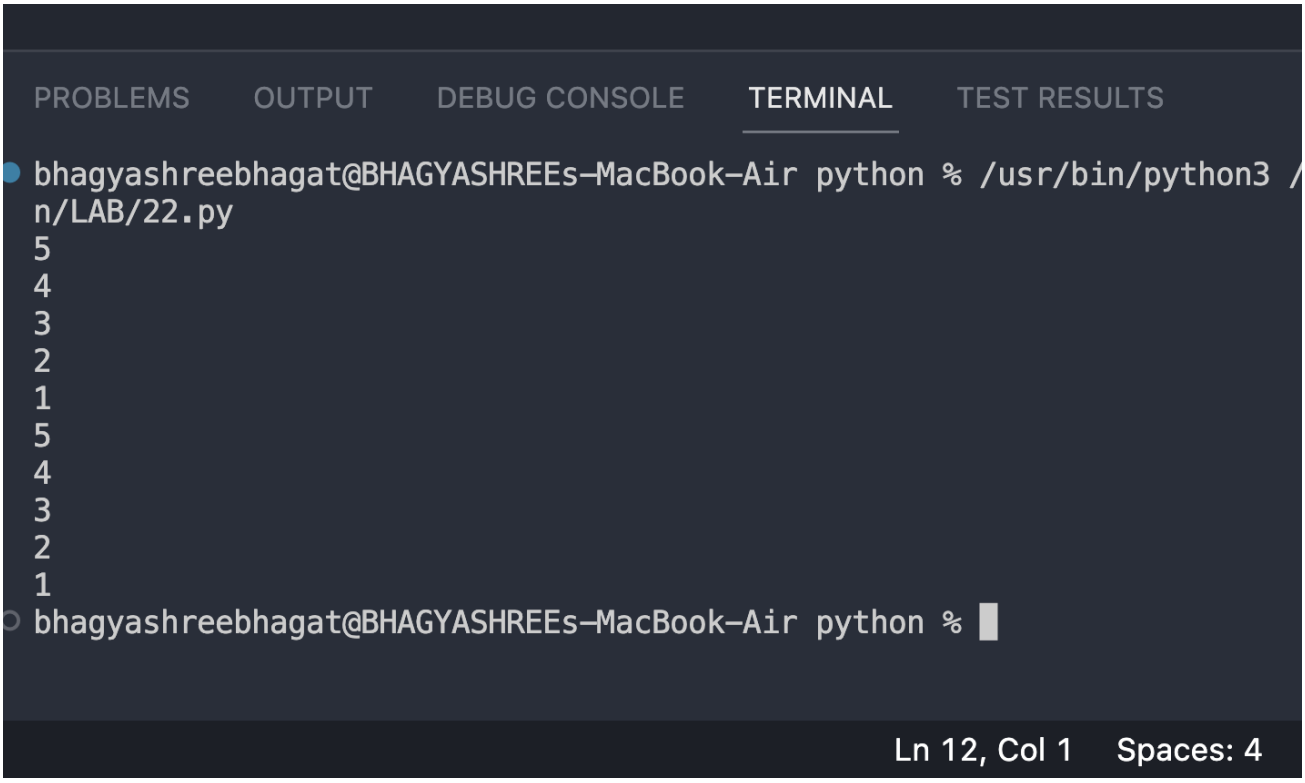
Theory:

This program extracts the username and domain from an email address and forms a tuple.

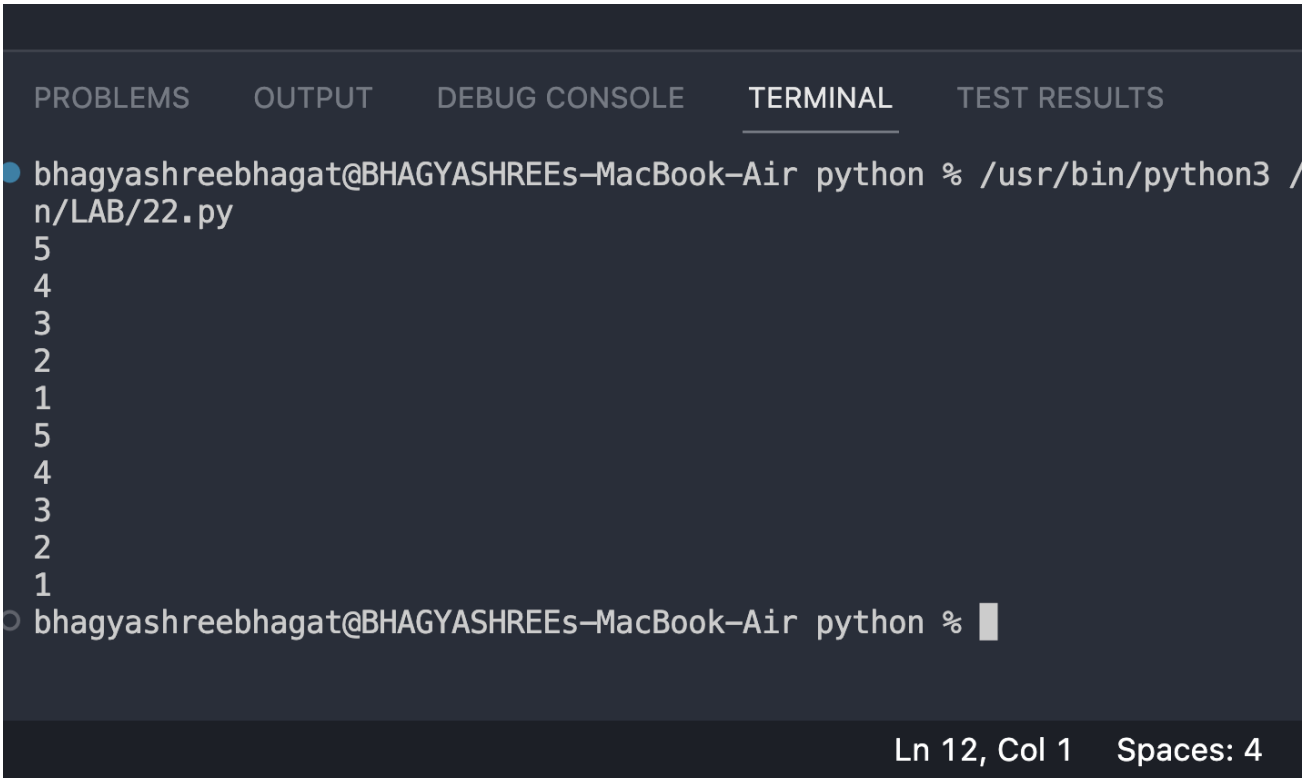
Code:

```
my_list = [1, 2, 3, 4, 5]



my_list.reverse()

for item in my_list:

   print(item)



my_list = [1, 2, 3, 4, 5]



for item in my_list[::-1]:

   print(item)
```

Output: (screenshot)



Test Case: Any two (screenshot)



Conclusion:

It provides a practical example of string manipulation, showing how to extract specific information from an email address.

Name of Student: __Bhagyashree Bhagat_____

Roll Number: 34_____

Experiment No:2.3

Title:

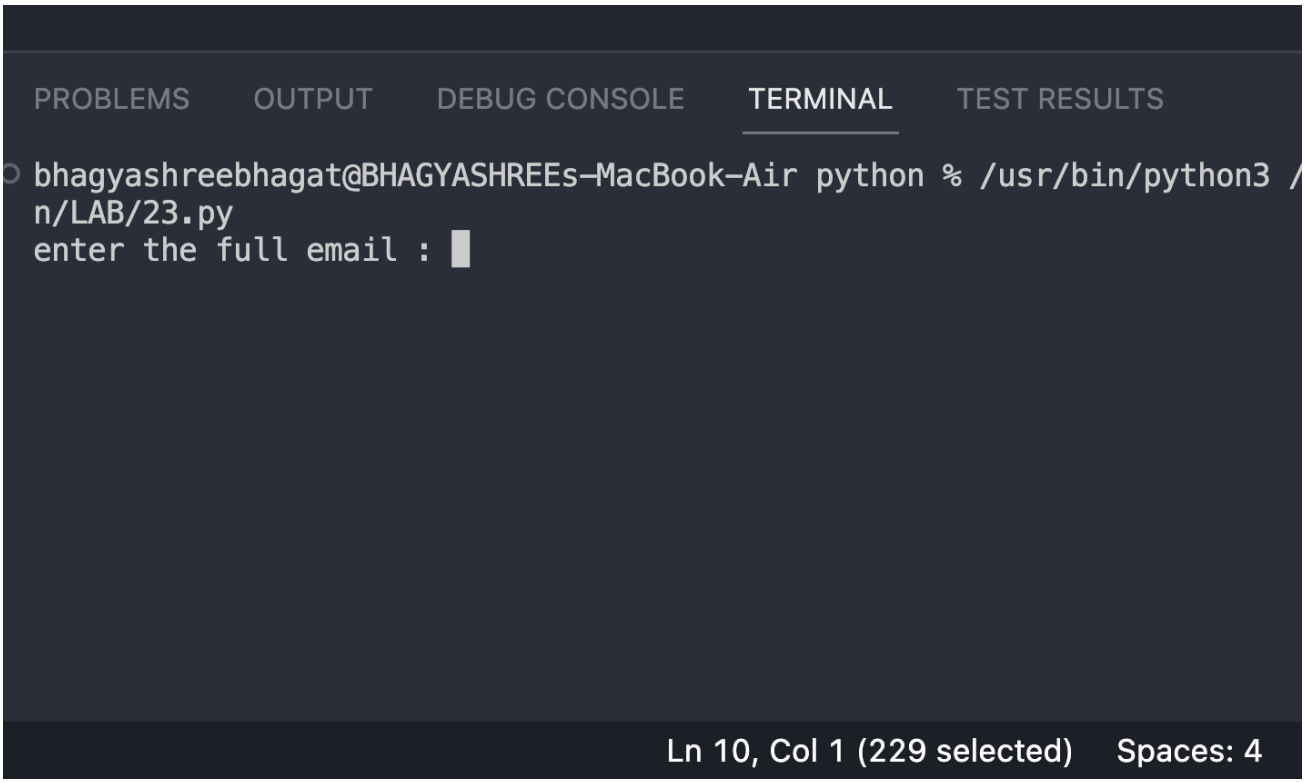
Form a Tuple of Username and Domain from Email Address:

Theory: **This program extracts the username and domain from an email address and forms a tuple.**

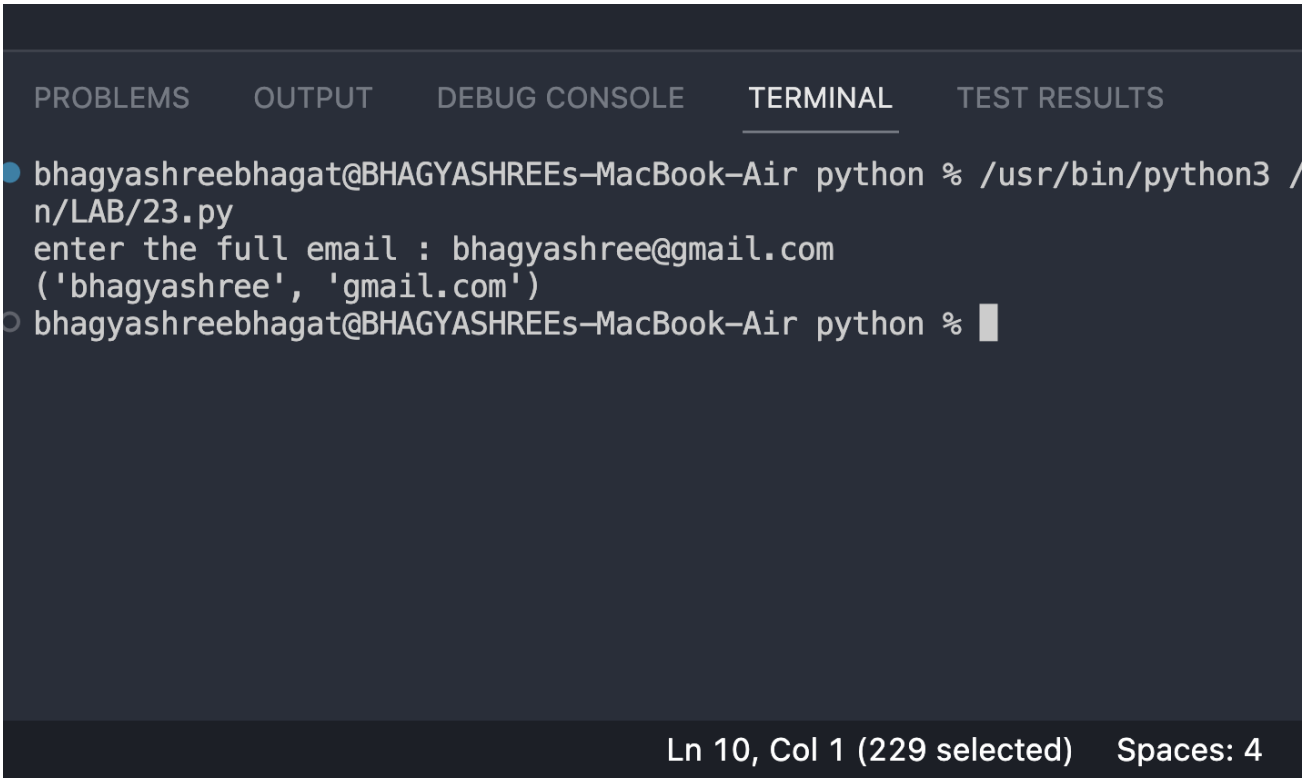
Code:

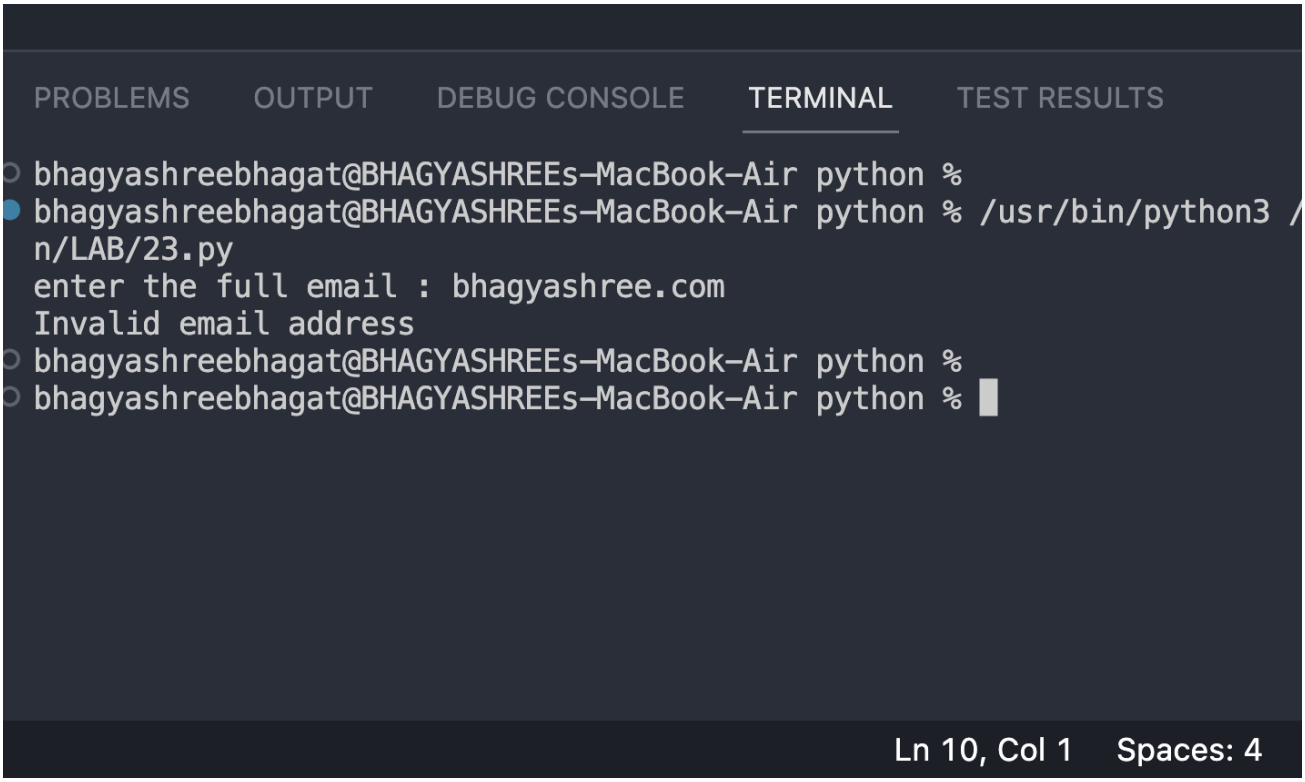
```
email = input("enter the full email : ")

parts = email.split('@')



if len(parts) == 2:

   username, domain = parts

   user_domain_tuple = (username, domain)

   print(user_domain_tuple)

else:

   print("Invalid email address")
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

It provides a practical example of string manipulation, showing how to extract specific information from an email address.

Name of Student: __Bhagyashree Bhagat_____

Roll Number: 34_____

Experiment No:2.4

Title:

Create a List of Tuples and Add Cubes:

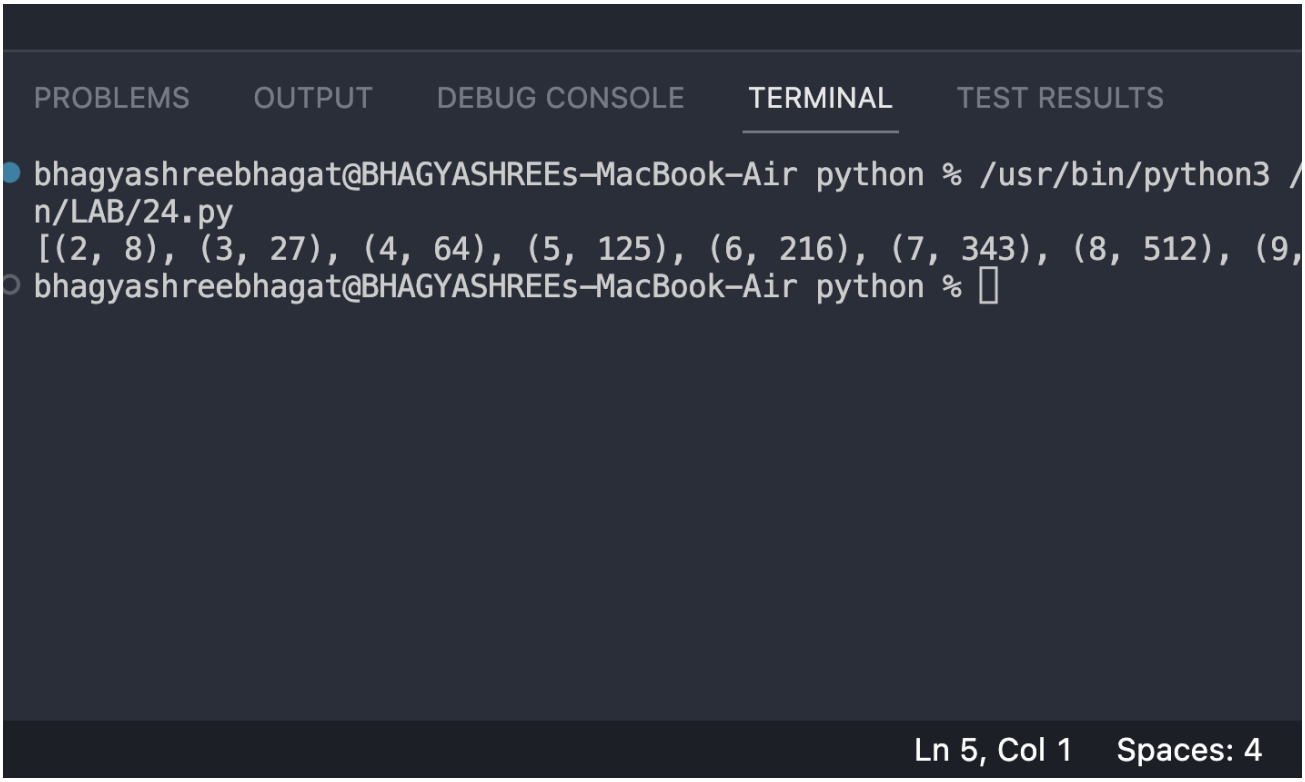
Theory:

The program generates a list of tuples from a given list and adds the cube of each number to the tuple.

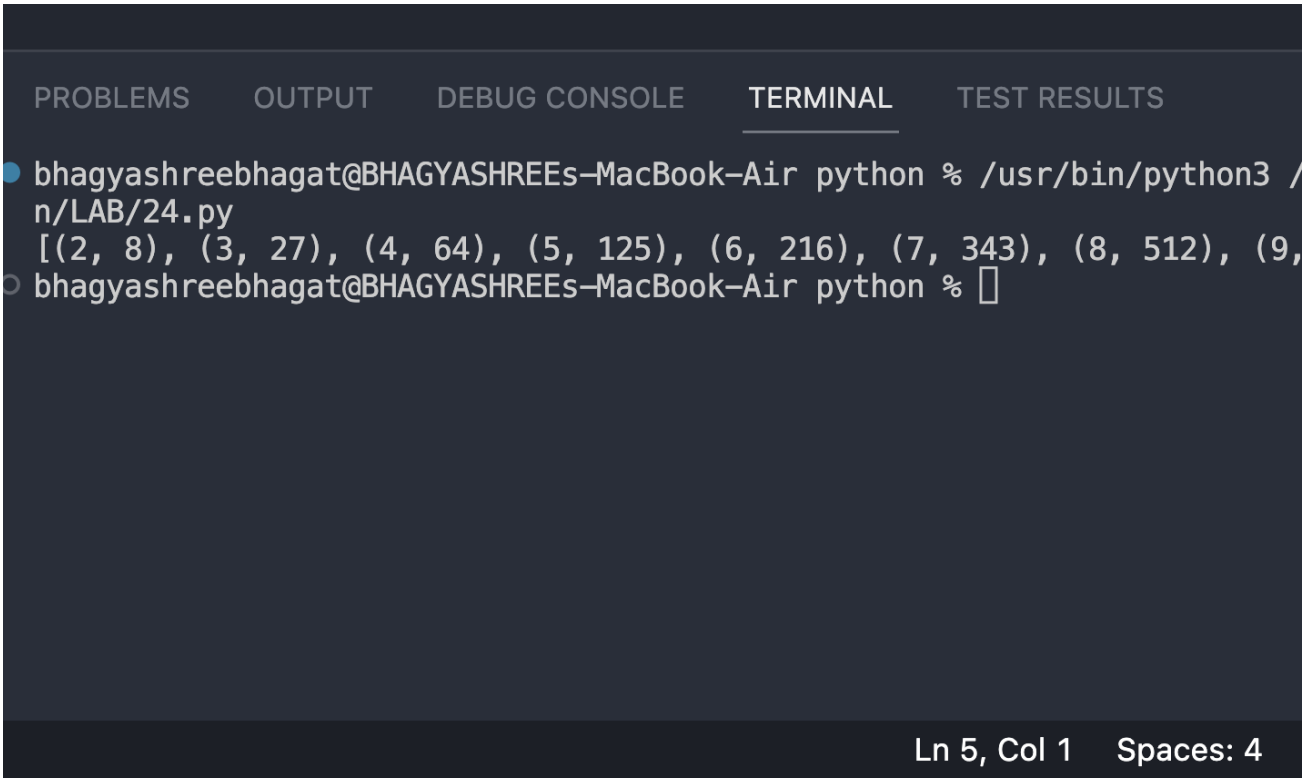
Code:

```
c = [2, 3, 4, 5, 6, 7, 8, 9]

result = [(x, x**3) for x in c]

print(result)
```

Output: (screenshot)



Test Case: Any two (screenshot)



Conclusion:

It demonstrates the creation of tuples and the addition of cubes to create more complex data structures.

Name of Student: __Bhagyashree Bhagat_____

Roll Number: 34_____

Experiment No:2.5

Title:

Compare Two Dictionaries in Python:

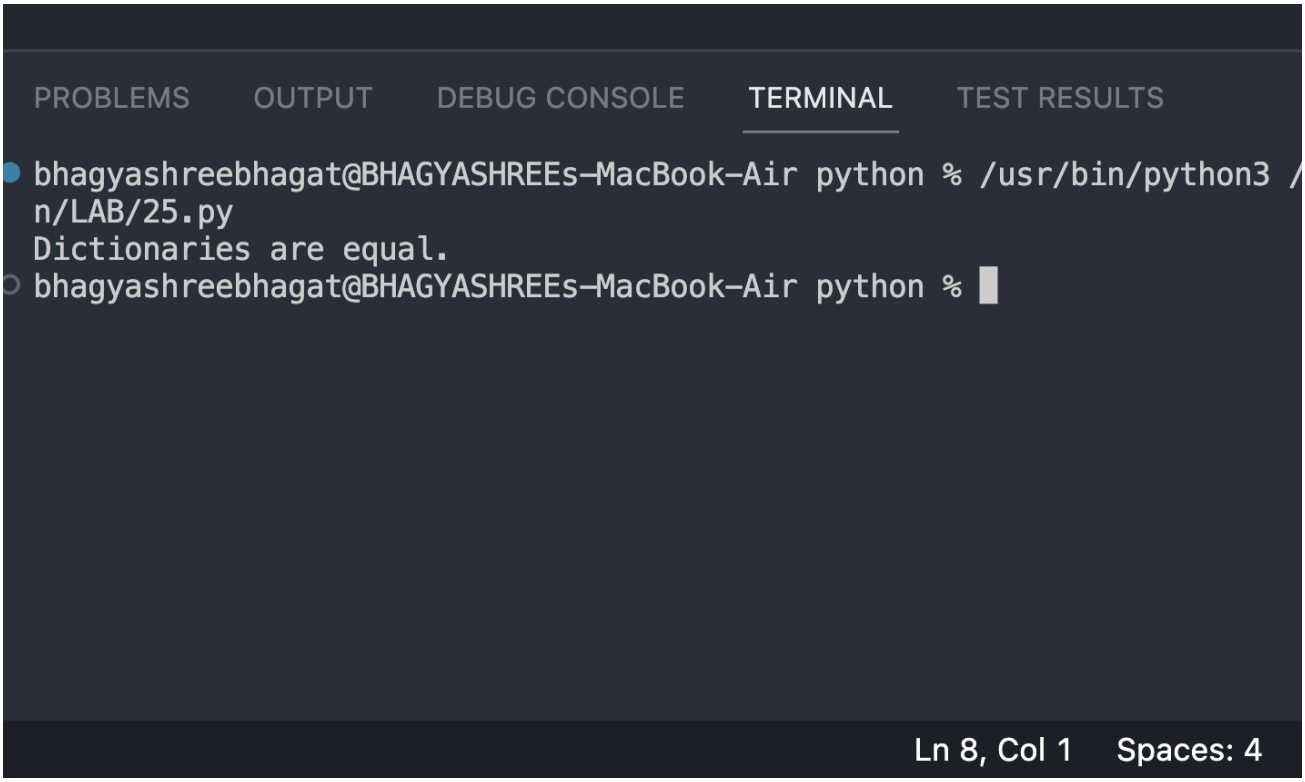
Theory:

The program compares two dictionaries using the == operator.

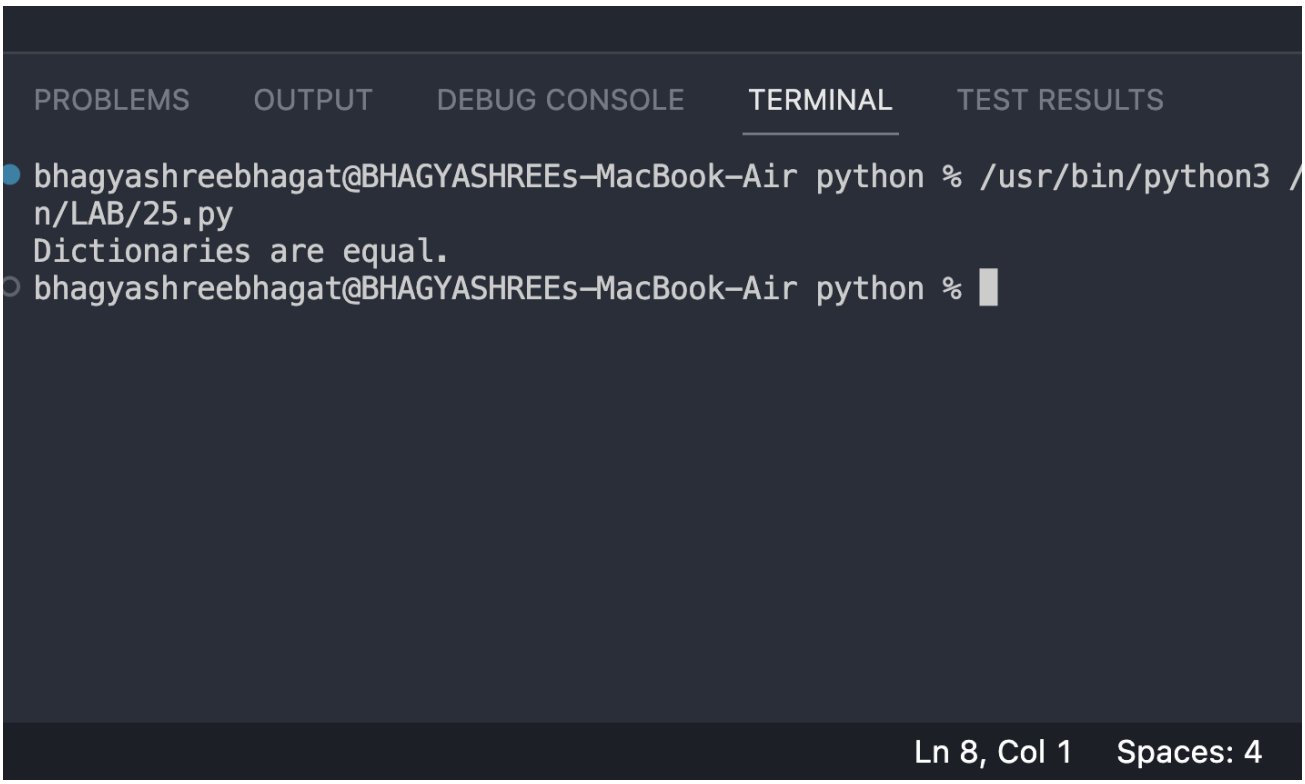
Code:

```
dict1 = {'a': 1, 'b': 2, 'c': 3}

dict2 = {'a': 1, 'b': 2, 'c': 3}



if dict1 == dict2:

   print("Dictionaries are equal.")

else:

   print("Dictionaries are not equal.")
```

Output: (screenshot)



Test Case: Any two (screenshot)



Conclusion:

It highlights a straightforward method to check for equality between dictionaries.

Experiment No: 2.6

Title:

Create a Dictionary of Cubes of Odd Numbers:

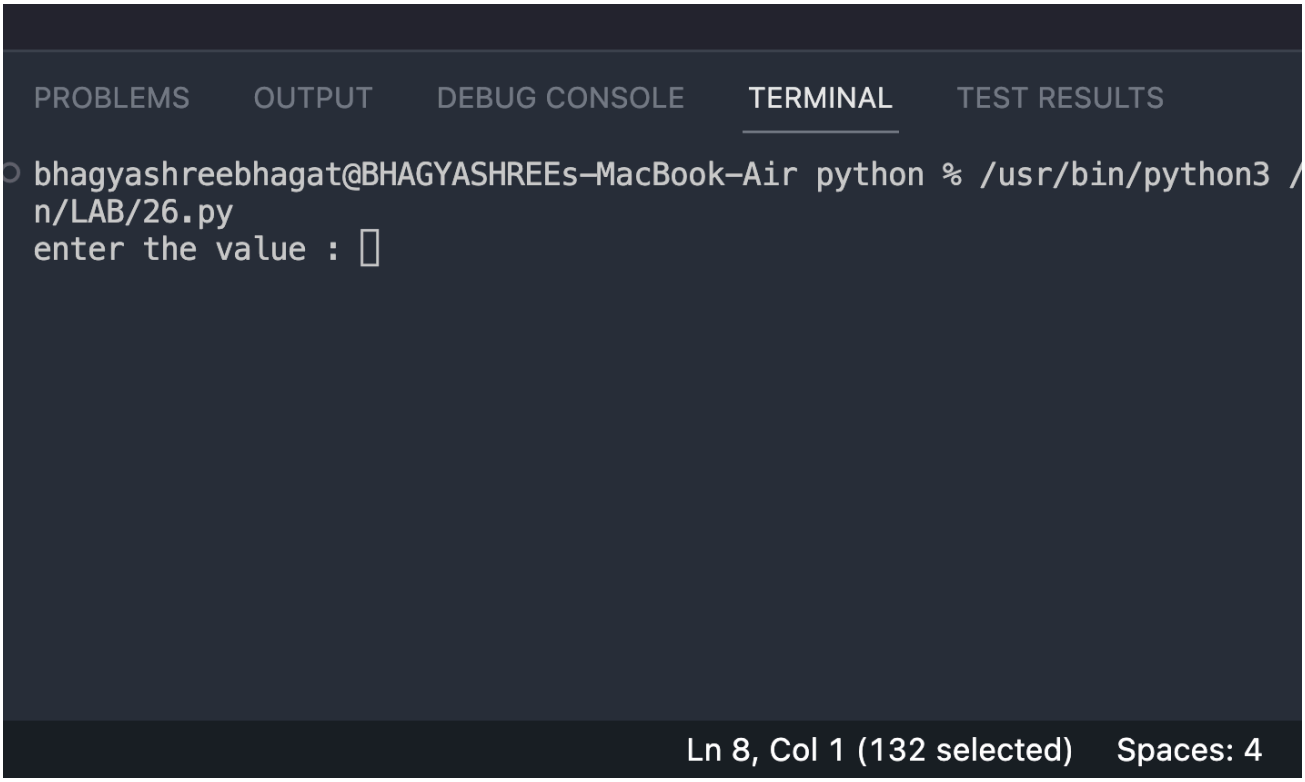
Theory:

The program creates a dictionary containing cubes of odd numbers in a specified range.

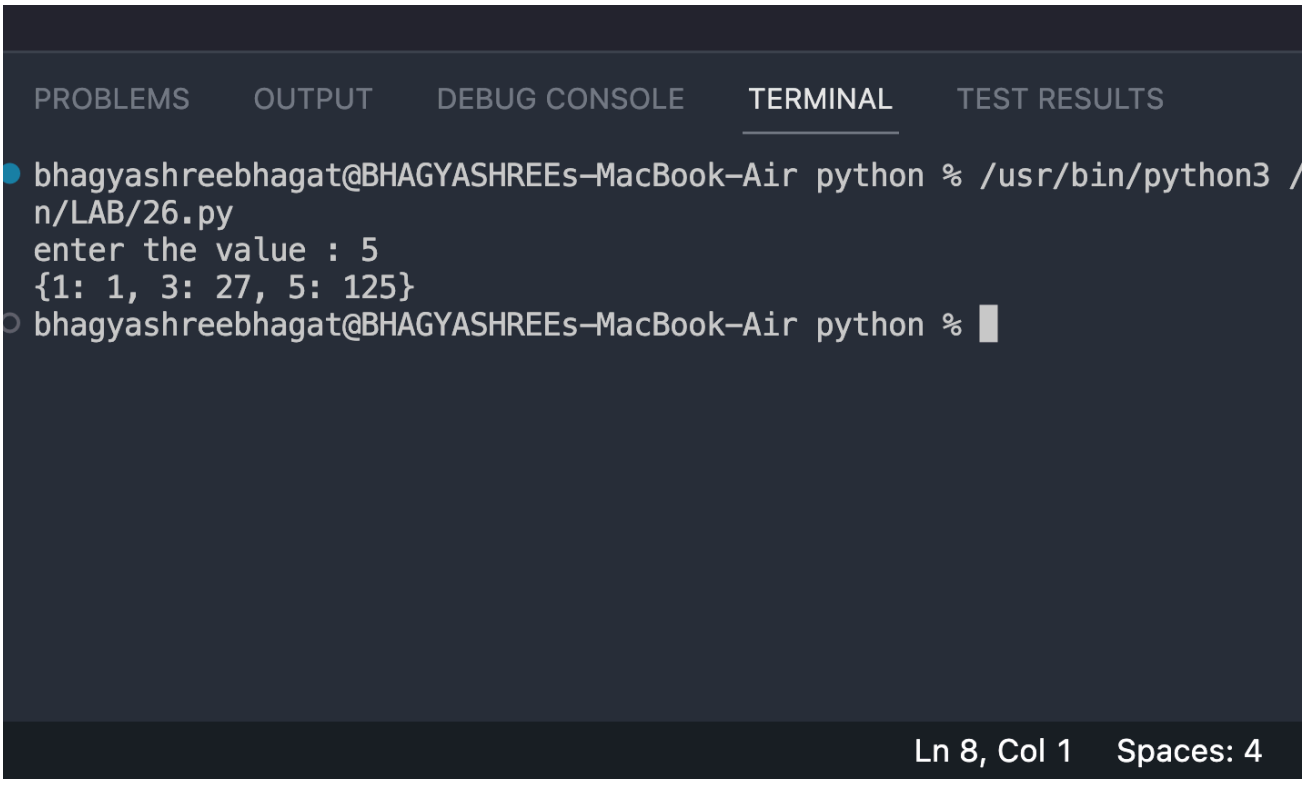
Code:

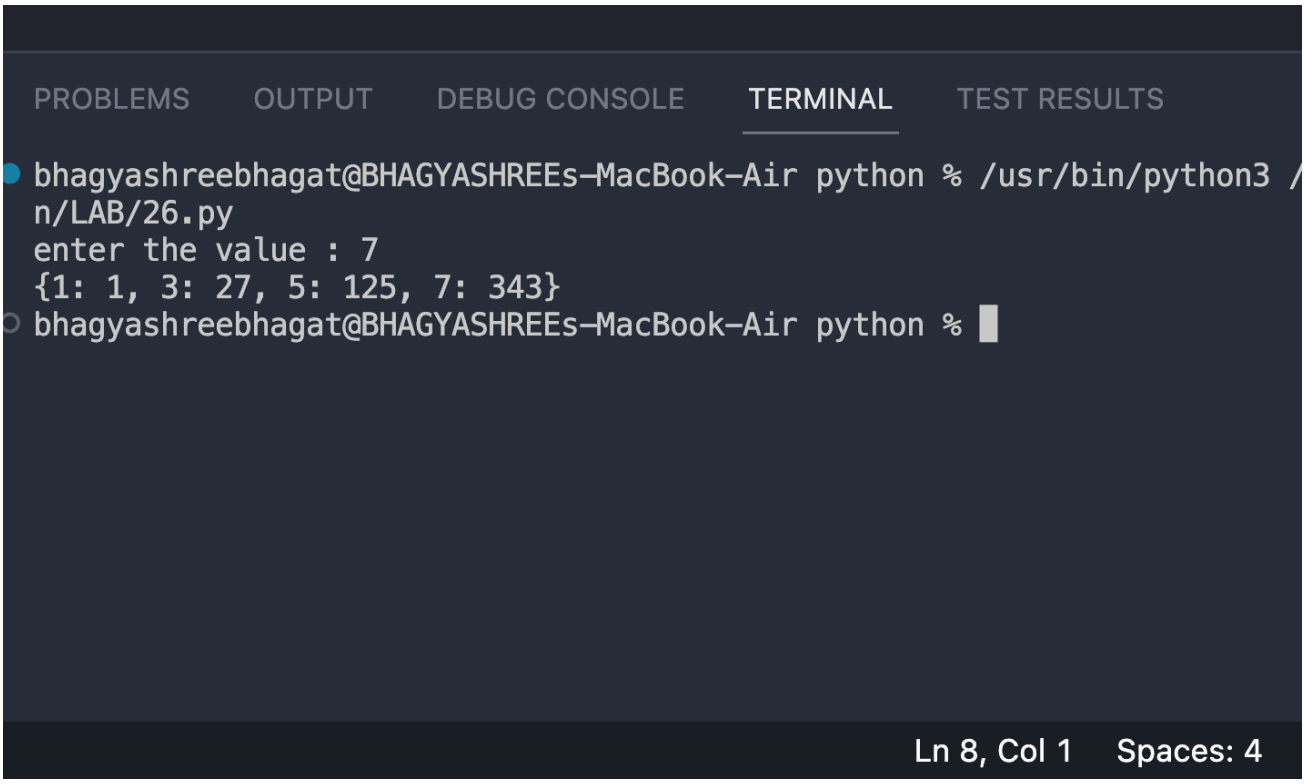
```
n = int(input("enter the value : "))
cube_dict = {}

for num in range(1, n + 1, 2):
    cube_dict[num] = num ** 3

print(cube_dict)
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

It shows how to generate dictionaries with specific properties based on a given range of values.

Experiment No:2.7

Title:

Various List Slicing Operations:

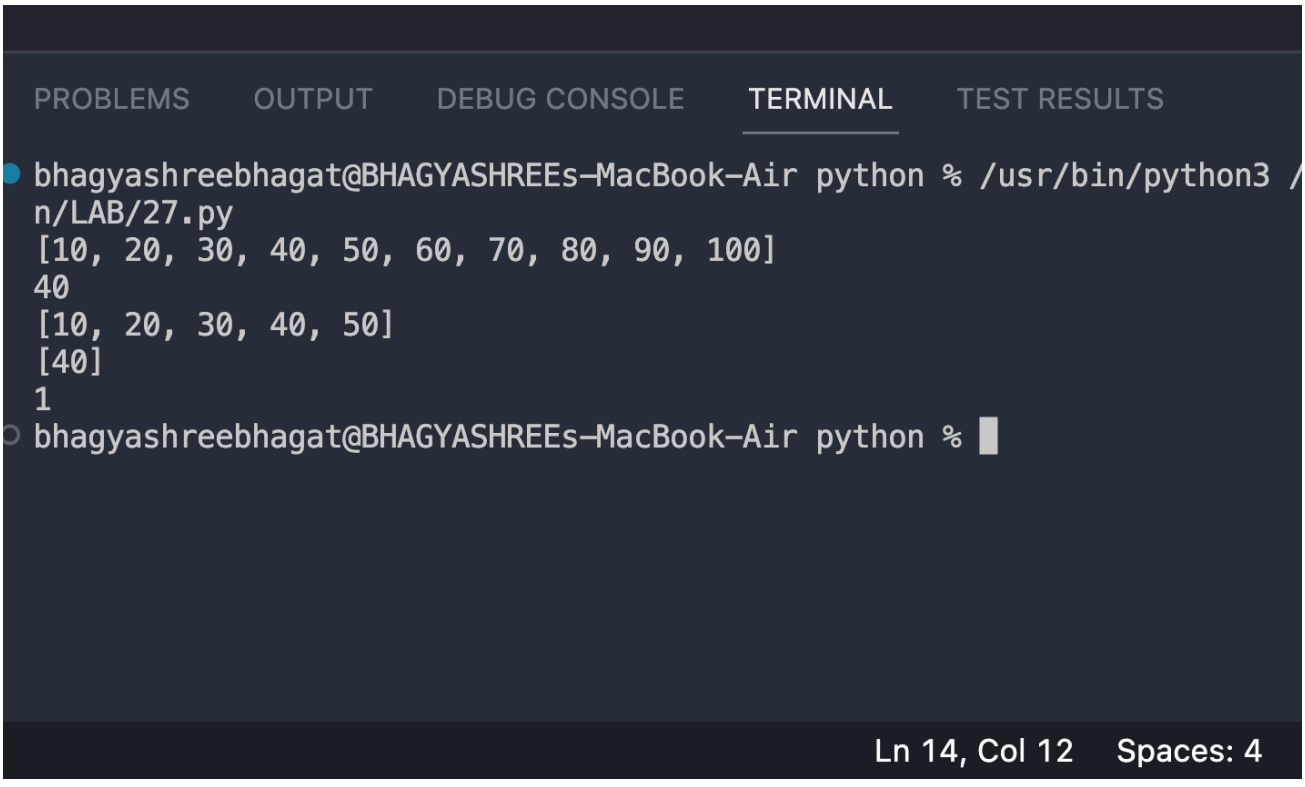
Theory:

The program demonstrates various list slicing operations such as printing, indexing, appending, sorting, popping, removing, inserting, counting, extending, and reversing.

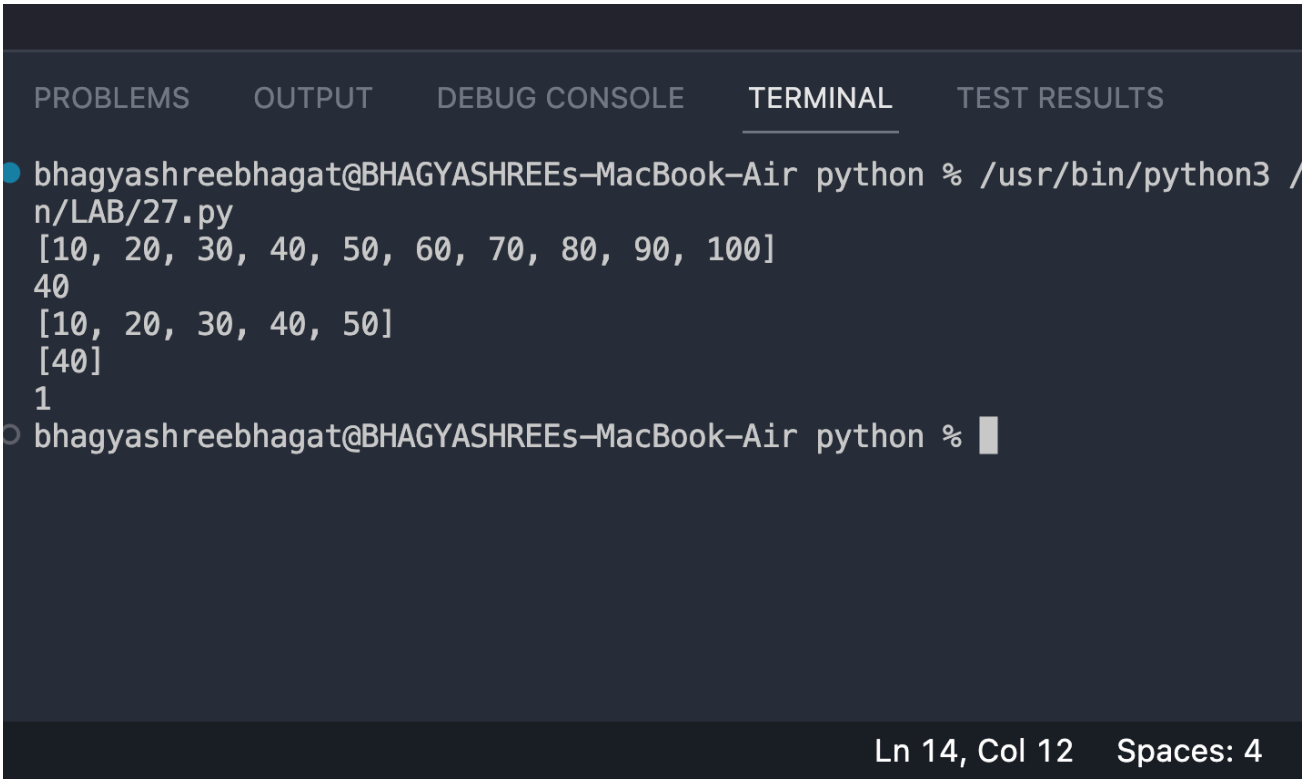
Code:

```
a = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

print(a)
print(a[3])
print(a[0:5])
print(a[-7:4])
a.append(110)
a.sort()
a.pop()
a.remove(40)
a.insert(2, 15)
print(a.count(30))
a.extend([120, 130])
a.revers
```

Output: (screenshot)



Test Case: Any two (screenshot)



Conclusion:

It covers a wide range of list manipulation techniques, showcasing the versatility of Python lists.

Experiment No:3.1

Title:

Extend a List Using + Operator, Append, and Extend:

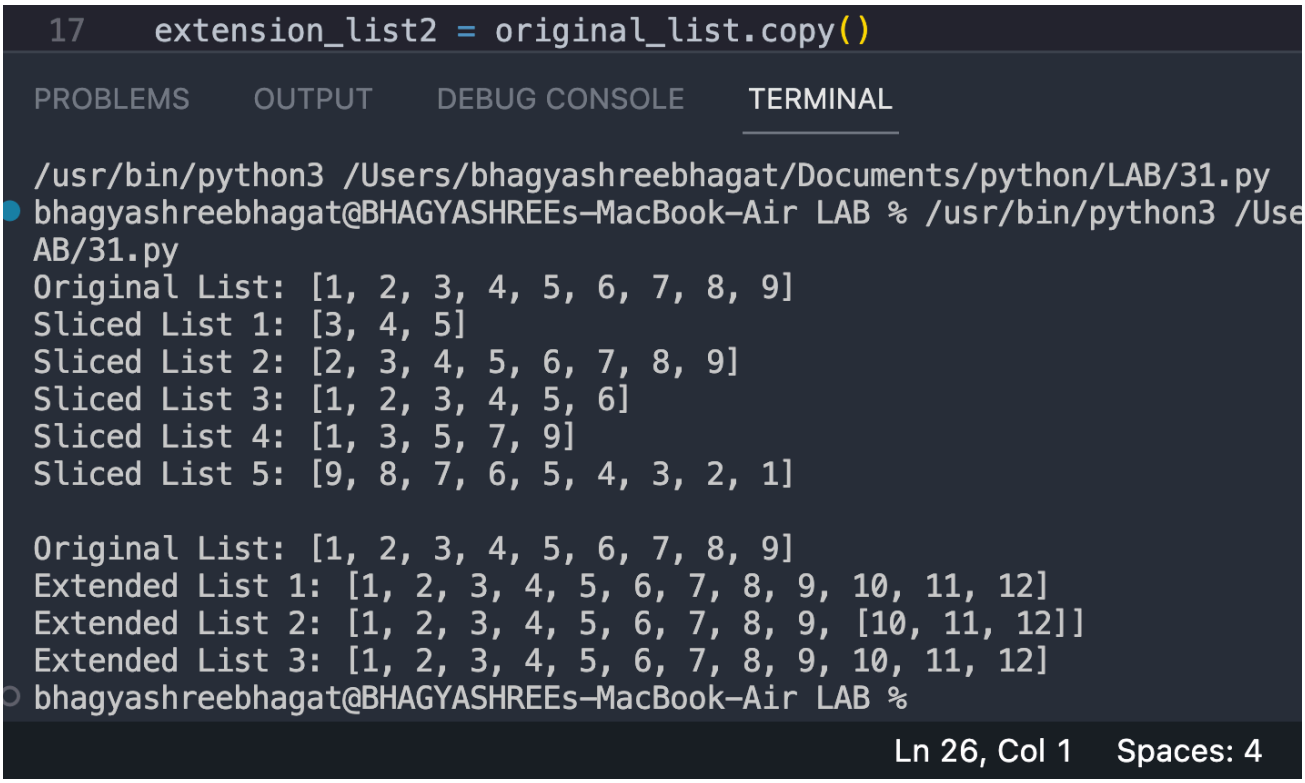
Theory:

This program shows different approaches to extend a list in Python using the + operator, append, and extend.

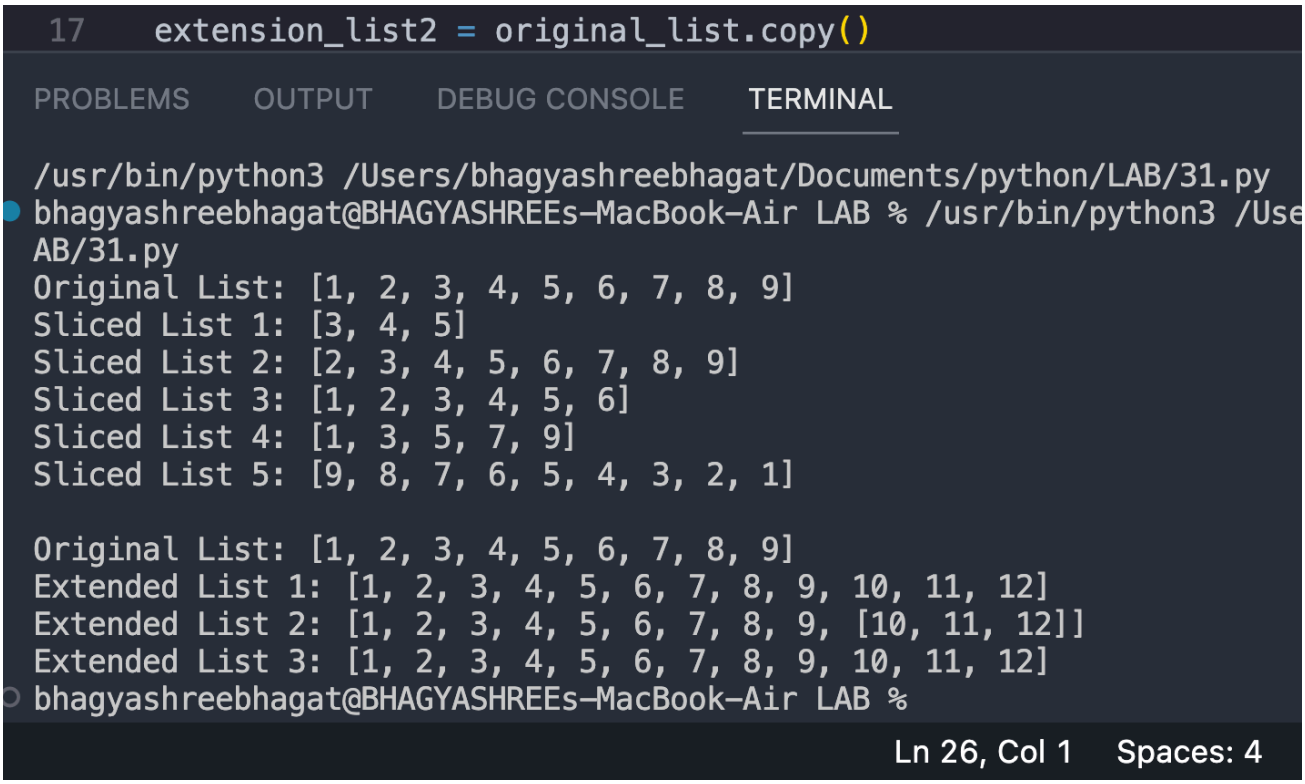
Code:

```
original_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]

sliced_list1 = original_list[2:5]
sliced_list2 = original_list[1:]
sliced_list3 = original_list[:6]
sliced_list4 = original_list[::2]
sliced_list5 = original_list[::-1]

print("Original List:", original_list)
print("Sliced List 1:", sliced_list1)
print("Sliced List 2:", sliced_list2)
print("Sliced List 3:", sliced_list3)
print("Sliced List 4:", sliced_list4)
print("Sliced List 5:", sliced_list5)

extension_list1 = original_list + [10, 11, 12]
extension_list2 = original_list.copy()
extension_list2.append([10, 11, 12])
extension_list3 = original_list.copy()
extension_list3.extend([10, 11, 12])

print("\nOriginal List:", original_list)
print("Extended List 1:", extension_list1)
print("Extended List 2:", extension_list2)
print("Extended List 3:", extension_list3)
```

Output: (screenshot)



Test Case: Any two (screenshot)



Conclusion:

It demonstrates multiple methods for adding elements to a list, providing flexibility in list manipulation.

Experiment No:3.2

Title:

Add Two Matrices:

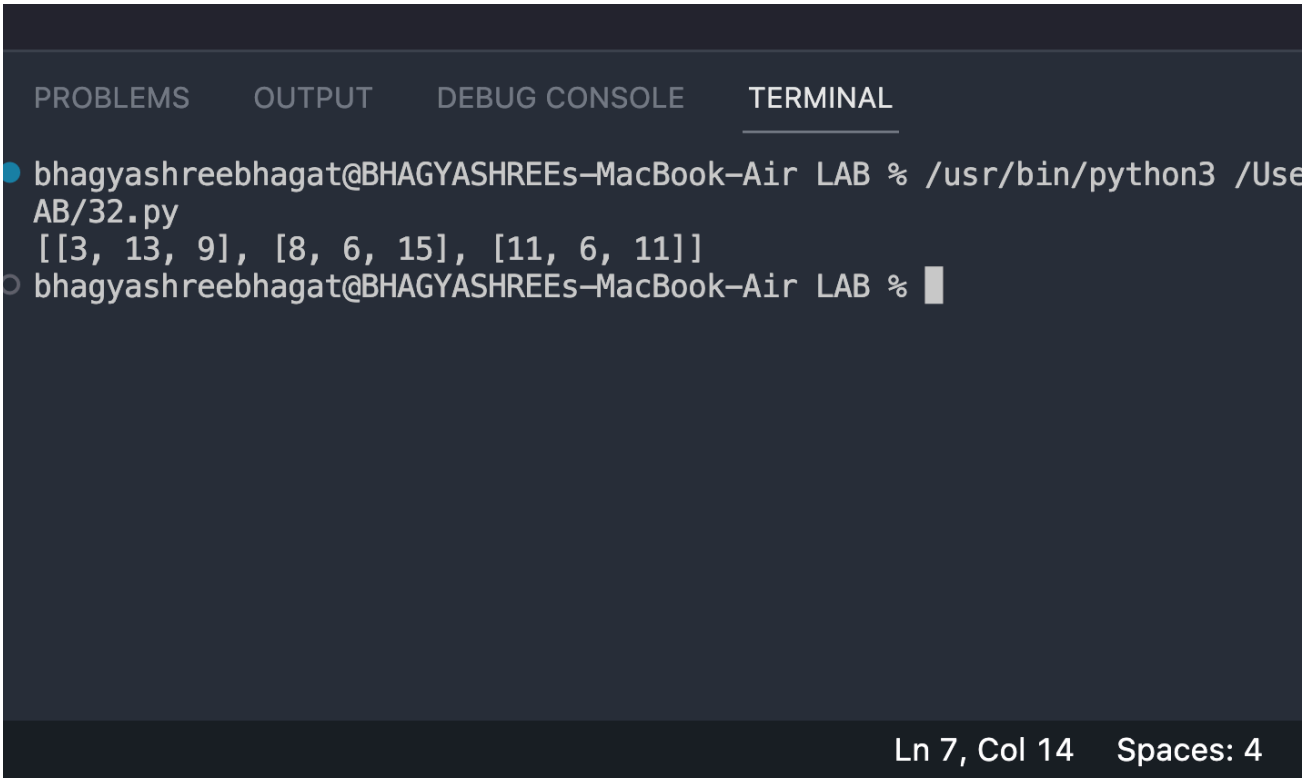
Theory:

The program adds two matrices using nested loops.

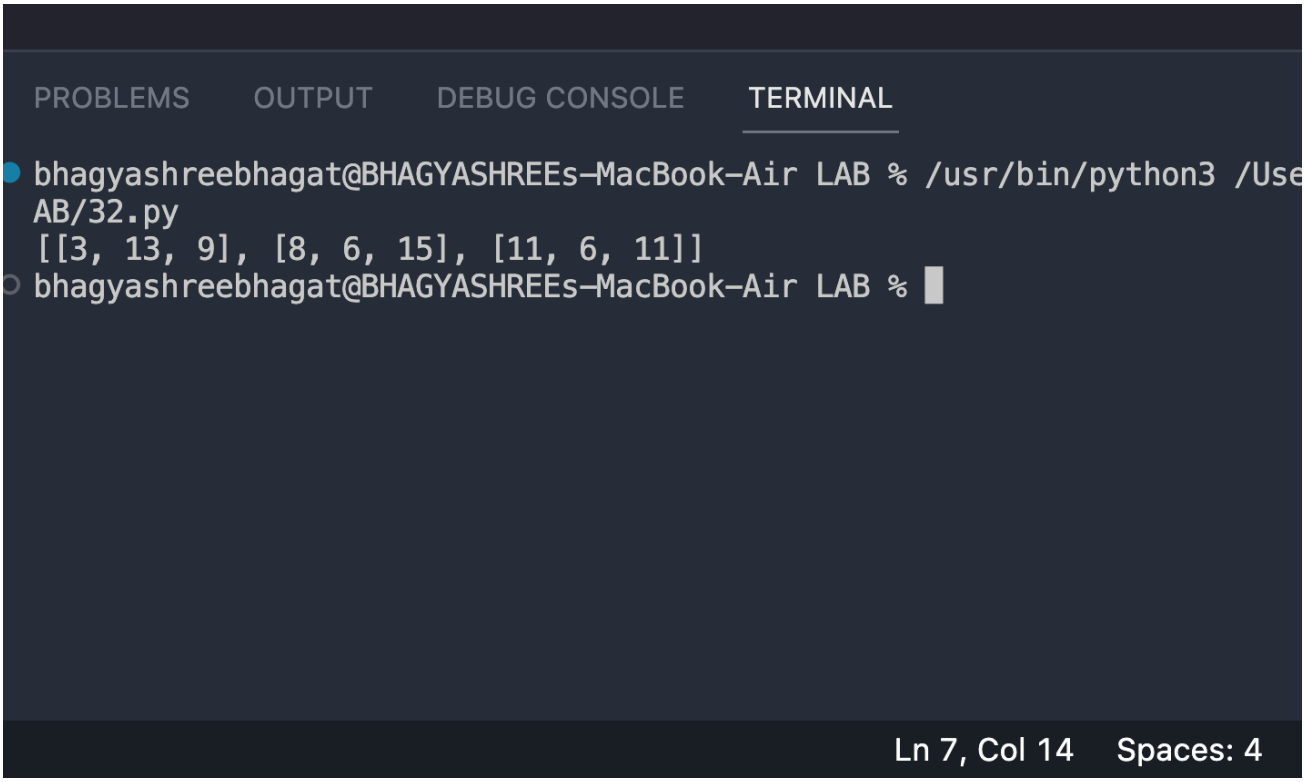
Code:

```
x=[[2,5,4],[1,3,9],[7,6,2]]
y=[[1,8,5],[7,3,6],[4,0,9]]
result=[[0,0,0],[0,0,0,],[0,0,0]]
for i in range(len(x)):
    for j in range(len(x[0])):
        result[i][j]=x[i][j] + y[i][j]
print(result)
```

Output: (screenshot)



Test Case: Any two (screenshot)



Conclusion:

It illustrates the process of matrix addition through nested loops, a fundamental operation in linear algebra.

Experiment No:3.3

Title:

Return a New List with Distinct Elements:

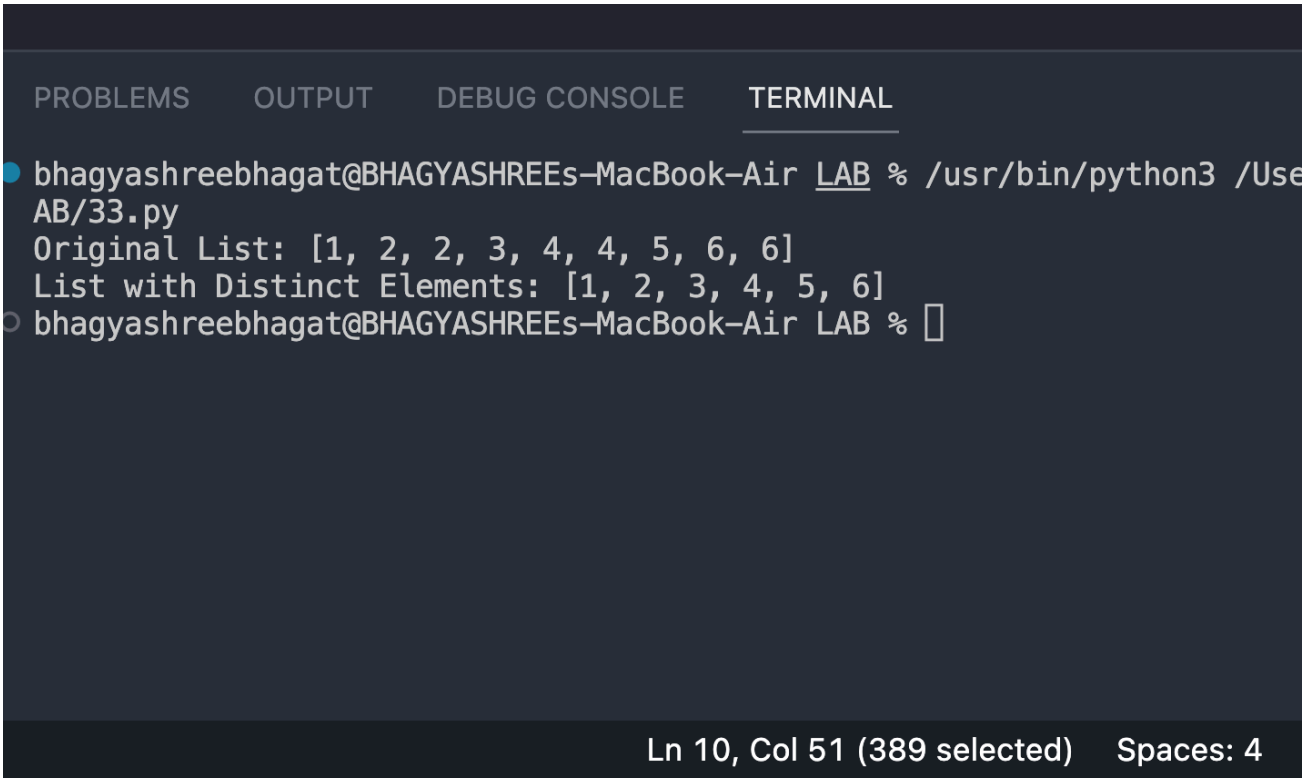
Theory:

The program creates a function that takes a list and returns a new list with distinct elements.

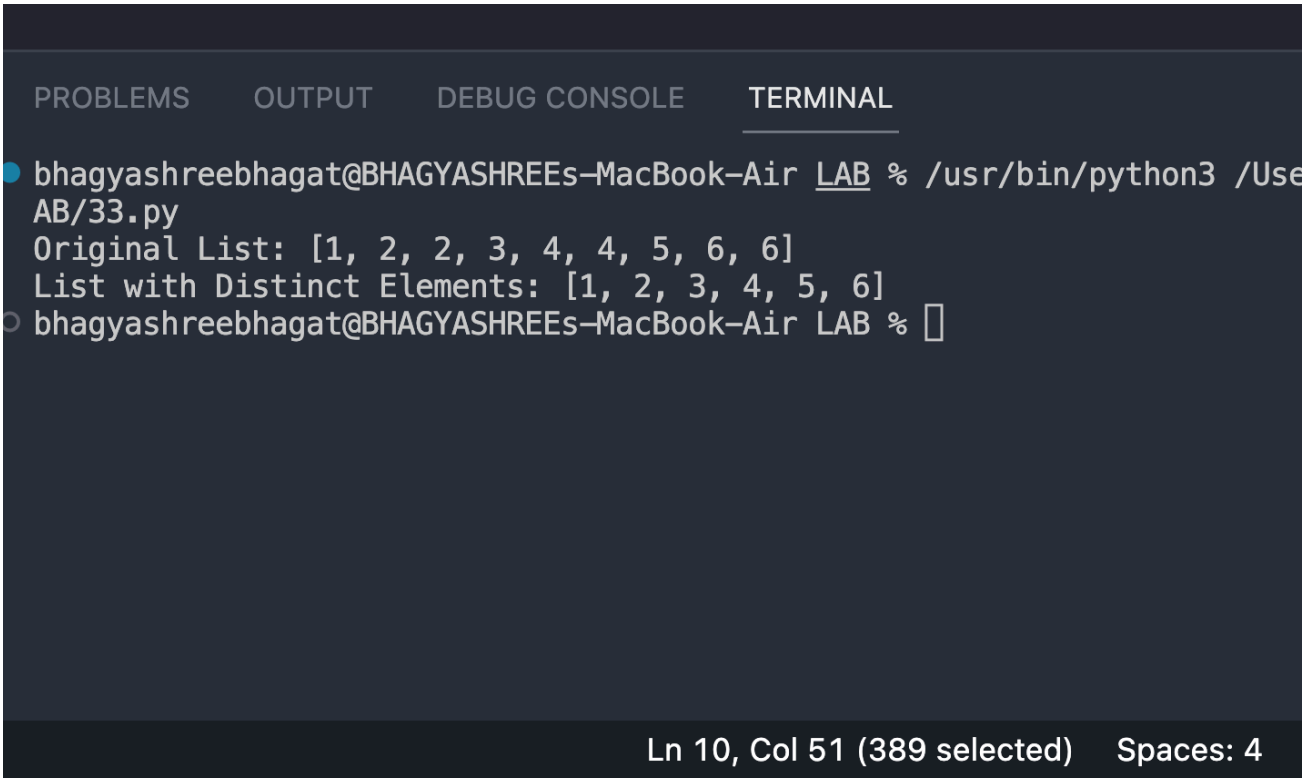
Code:

```
def get_unique_elements(input_list):
    unique_elements = []
    for element in input_list:
        if element not in unique_elements:
            unique_elements.append(element)
    return unique_elements
original_list = [1, 2, 2, 3, 4, 4, 5, 6, 6]
result_list = get_unique_elements(original_list)
print("Original List:", original_list)
print("List with Distinct Elements:", result_list)
```

Output: (screenshot)



Test Case: Any two (screenshot)



Conclusion:

It introduces the concept of functions and how they can be used to perform specific operations on data structures.

Experiment No:3.4

Title: Check Whether a Number is Perfect:

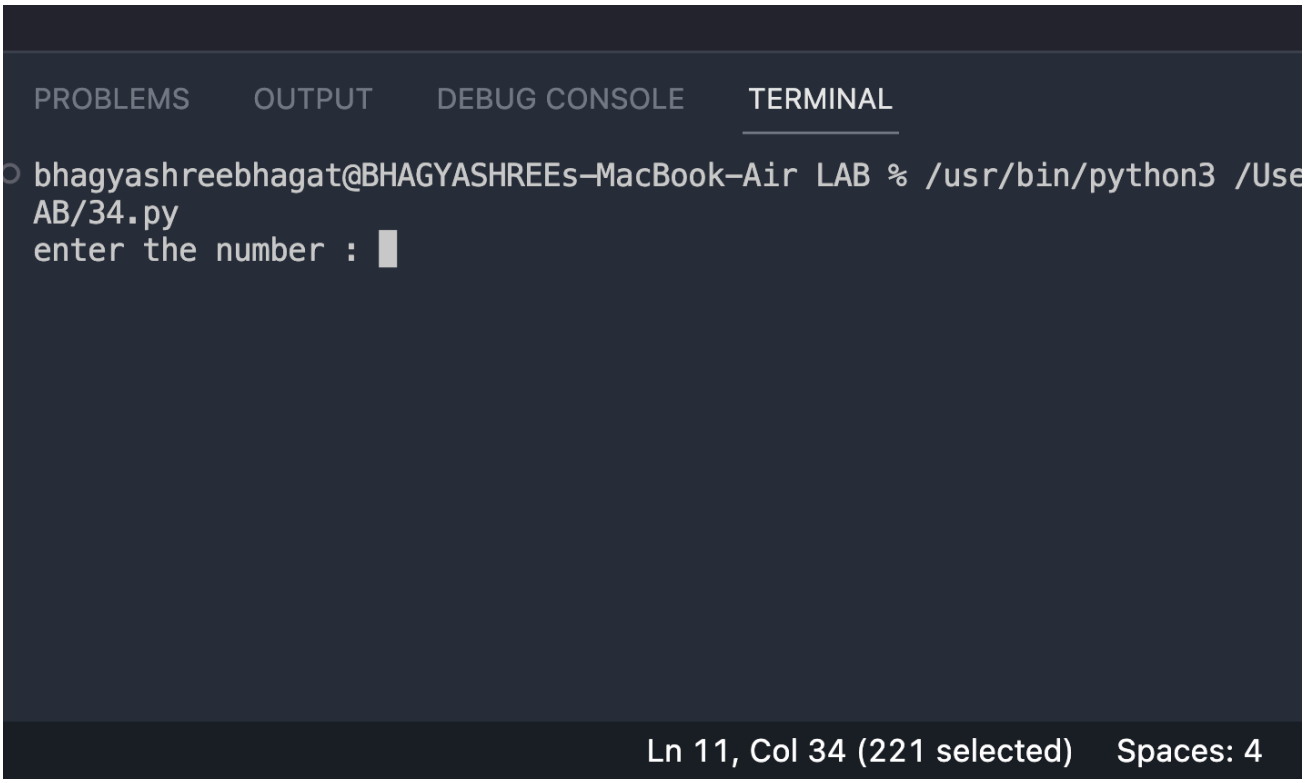
Theory:

The program determines whether a number is a perfect number, which is a number equal to the sum of its factors excluding itself.

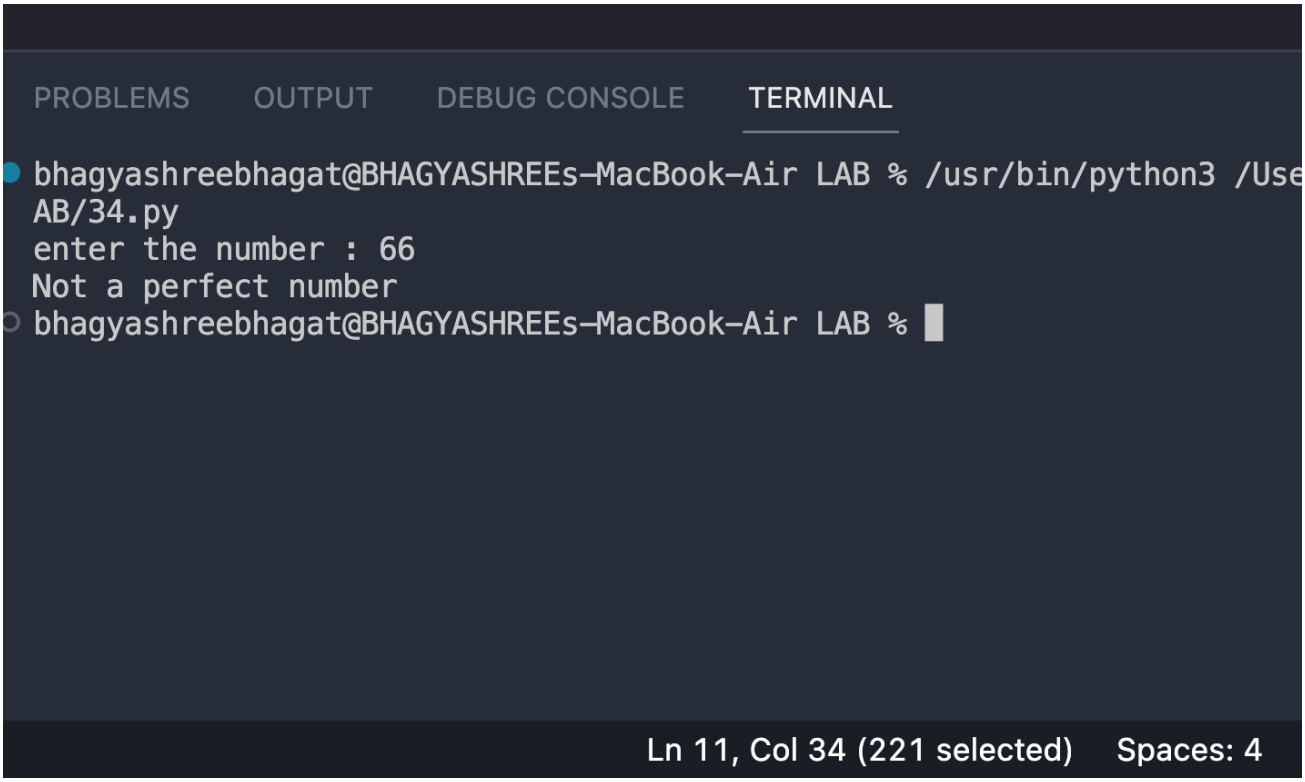
Code:

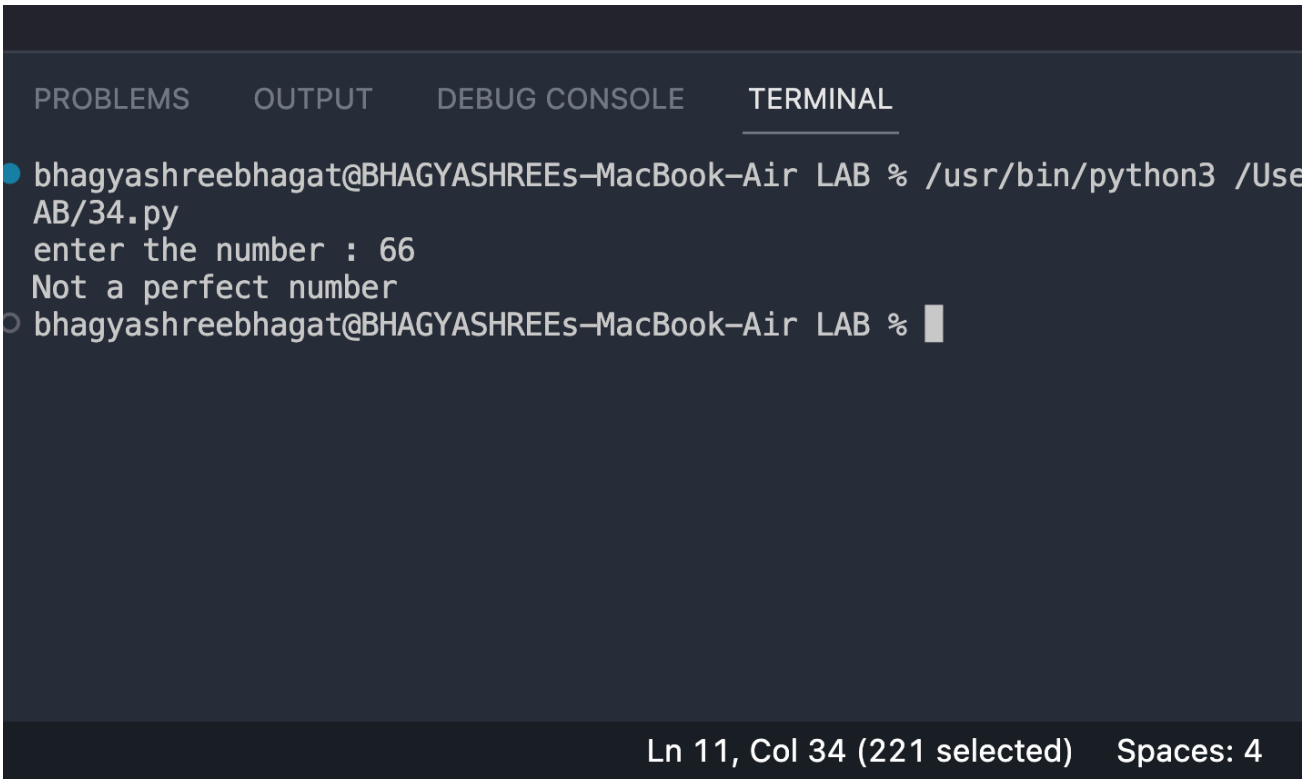
```
num = int(input("enter the number : "))
sum_divisors = 0

for i in range(1, num):
    if num % i == 0:
        sum_divisors += i

if sum_divisors == num:
    print("Perfect number")
else:
    print("Not a perfect number")
```

Output: (screenshot)



Test Case: Any two (screenshot)


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
bhagyashreebhagat@BHAGYASHREEs-MacBook-Air LAB % /usr/bin/python3 /Use
AB/34.py
enter the number : 66
Not a perfect number
bhagyashreebhagat@BHAGYASHREEs-MacBook-Air LAB %
Ln 11, Col 34 (221 selected)
Spaces: 4


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
bhagyashreebhagat@BHAGYASHREEs-MacBook-Air LAB % /usr/bin/python3 /Use
AB/34.py
enter the number : 66
Not a perfect number
bhagyashreebhagat@BHAGYASHREEs-MacBook-Air LAB %
Ln 11, Col 34 (221 selected)
Spaces: 4

Conclusion:

It introduces the concept of perfect numbers and showcases how to check for this property in a program.

Experiment No:3.5

Title:

Count Upper and Lower-case Letters:

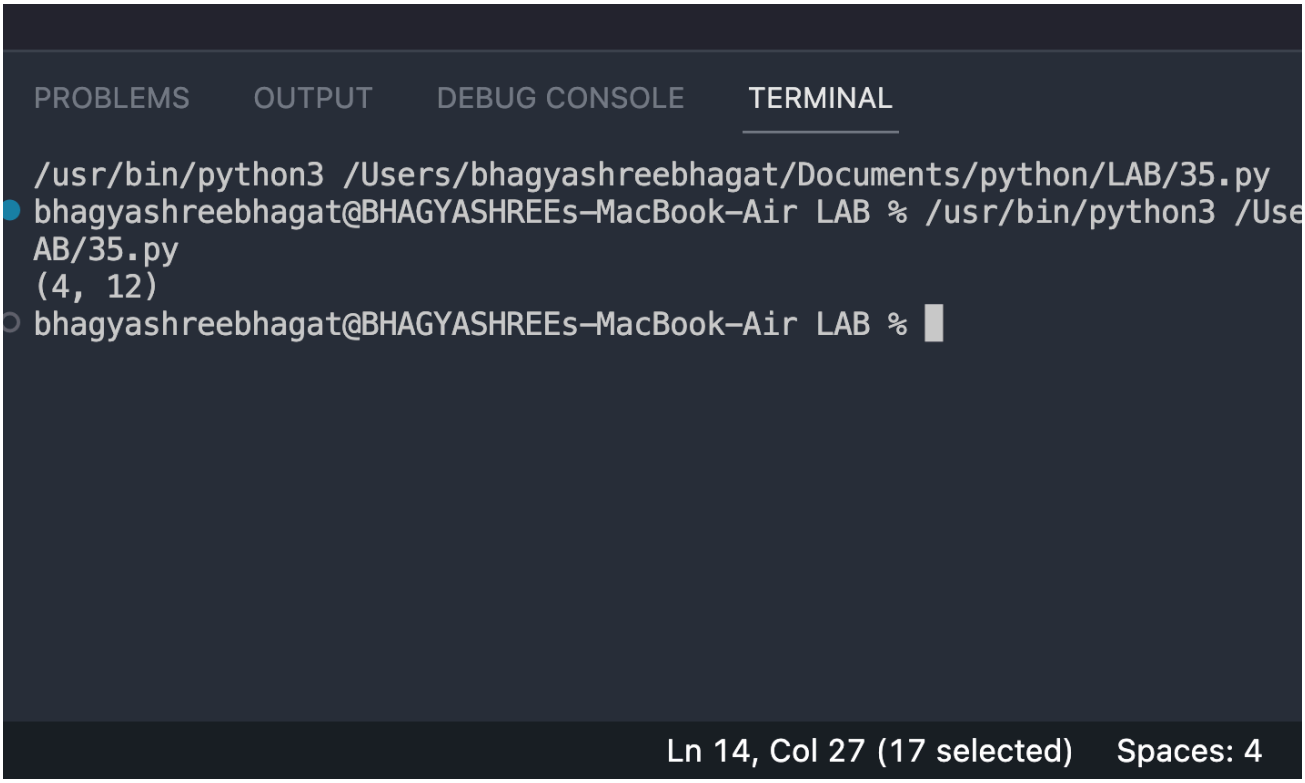
Theory:

The program counts the number of upper- and lower-case letters in a given string.

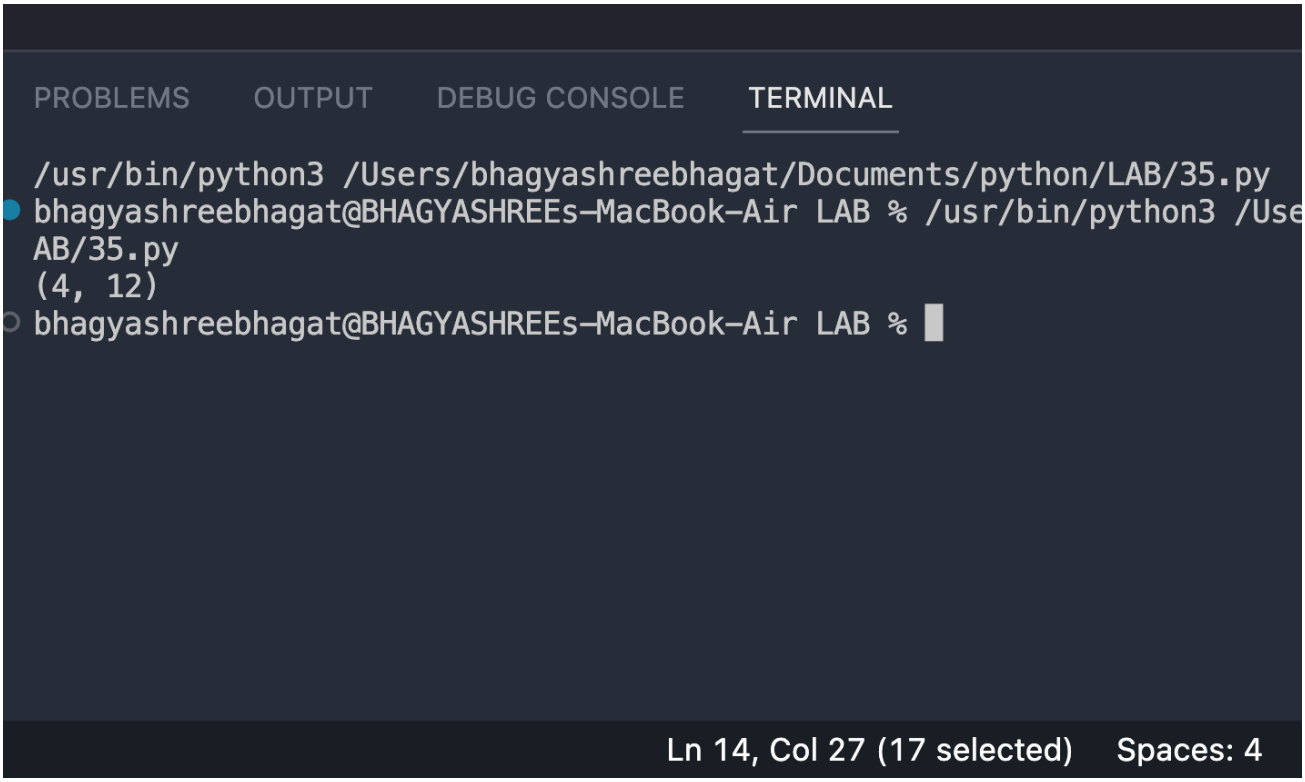
Code:

```
def count_upper_lower(string_test):
    upper_count = 0
    lower_count = 0

    for char in string_test:
        if char.isupper():
            upper_count += 1
        elif char.islower():
            lower_count += 1

    return upper_count, lower_count

string_test = 'Today is My Best Day'
result = count_upper_lower(string_test)
print(result)
```

Output: (screenshot)



Test Case: Any two (screenshot)



Conclusion:It demonstrates string manipulation and character counting in Python.

Experiment No:4.1

Title:

Create Employee Class and Methods:

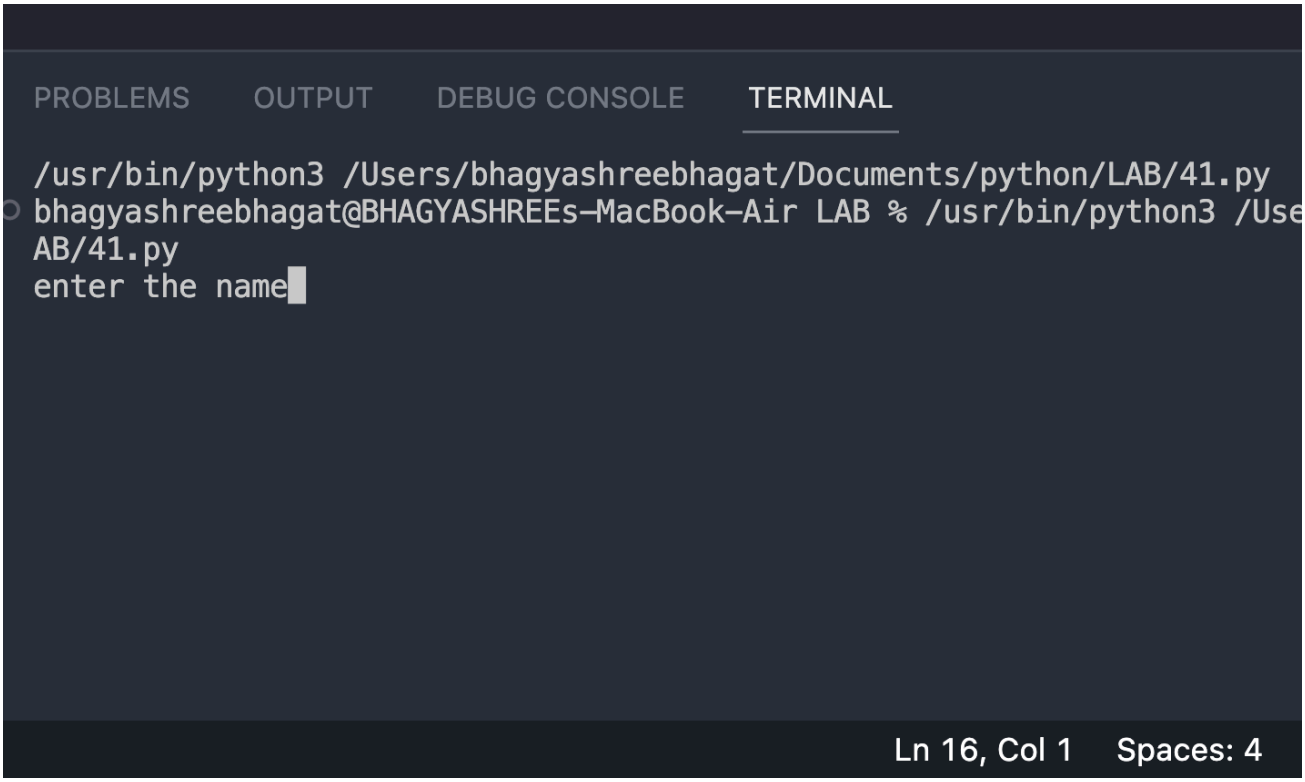
Theory:

This program defines an Employee class and adds methods to get and print employee details.

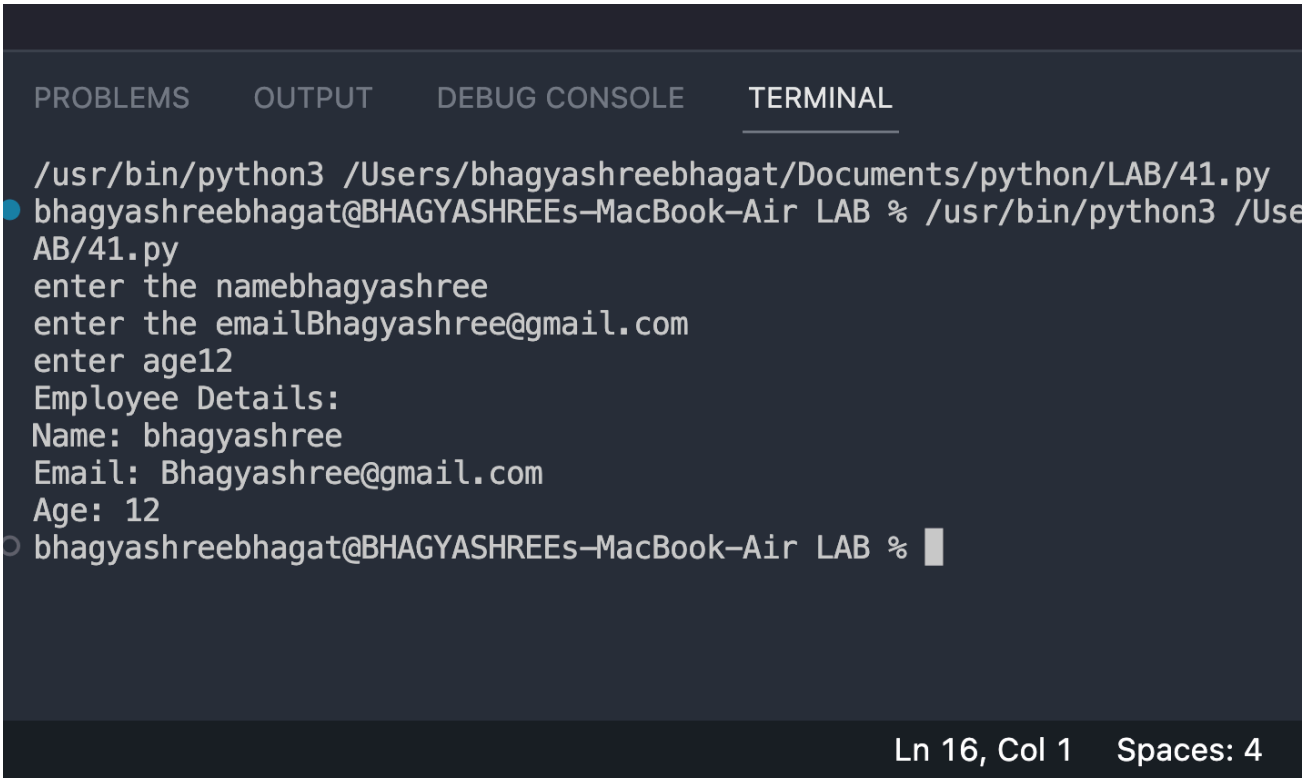
Code:

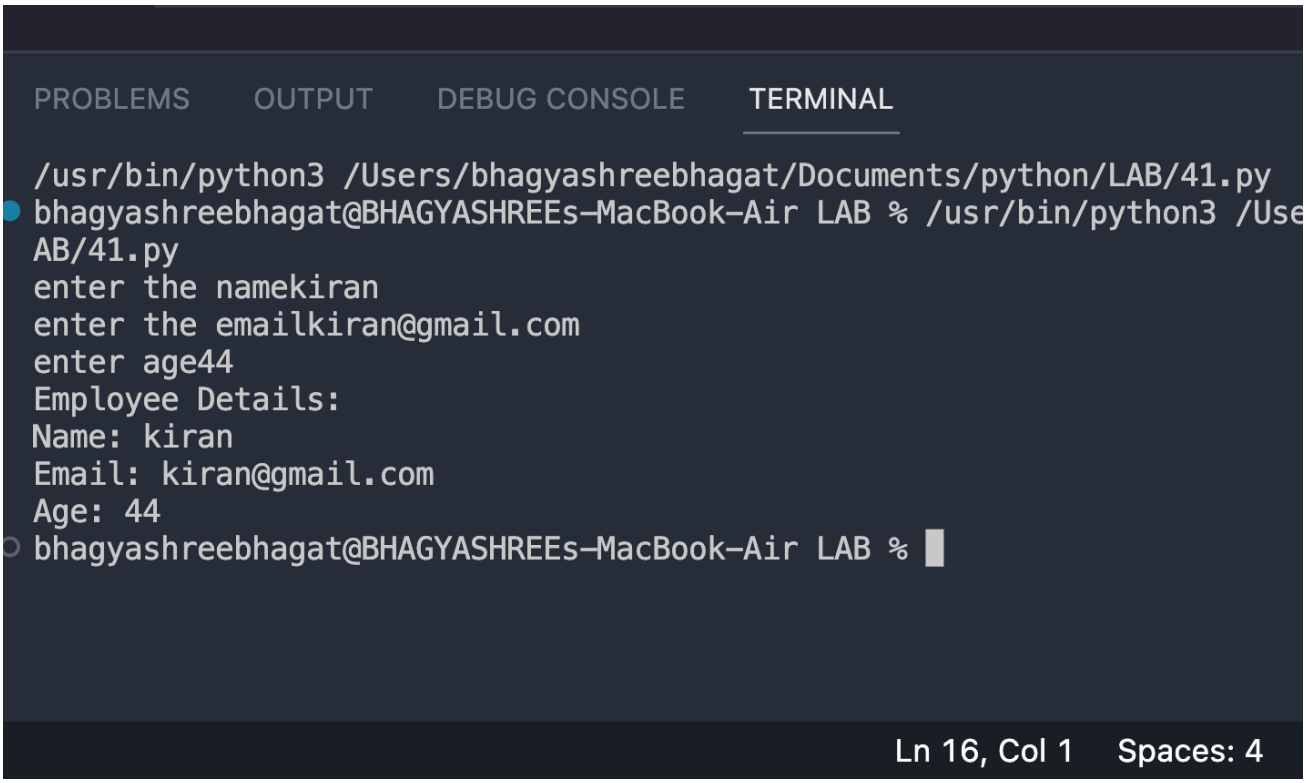
```
class Employee:
    def get_details(self):
        self.name = input("enter the name")
        self.email = input("enter the email")
        self.age = int(input("enter age"))

    def print_details(self):
        print("Employee Details:")
        print(f"Name: {self.name}")
        print(f"Email: {self.email}")
        print(f"Age: {self.age}")

emp = Employee()
emp.get_details()
emp.print_details()
```

Output: (screenshot)



Test Case: Any two (screenshot)


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
/usr/bin/python3 /Users/bhagyashreebhagat/Documents/python/LAB/41.py
bhagyashreebhagat@BHAGYASHREEs-MacBook-Air LAB % /usr/bin/python3 /Use
AB/41.py
enter the namebhagyashree
enter the emailBhagyashree@gmail.com
enter age12
Employee Details:
Name: bhagyashree
Email: Bhagyashree@gmail.com
Age: 12
bhagyashreebhagat@BHAGYASHREEs-MacBook-Air LAB %
Ln 16, Col 1
Spaces: 4


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
/usr/bin/python3 /Users/bhagyashreebhagat/Documents/python/LAB/41.py
bhagyashreebhagat@BHAGYASHREEs-MacBook-Air LAB % /usr/bin/python3 /Use
AB/41.py
enter the namekiran
enter the emailkiran@gmail.com
enter age44
Employee Details:
Name: kiran
Email: kiran@gmail.com
Age: 44
bhagyashreebhagat@BHAGYASHREEs-MacBook-Air LAB %
Ln 16, Col 1
Spaces: 4

Conclusion:It showcases the creation of classes, methods, and the use of objects to model real-world entities.

Experiment No:4.2

Title: Input Name, Email, and Age Using *args and **kwargs:

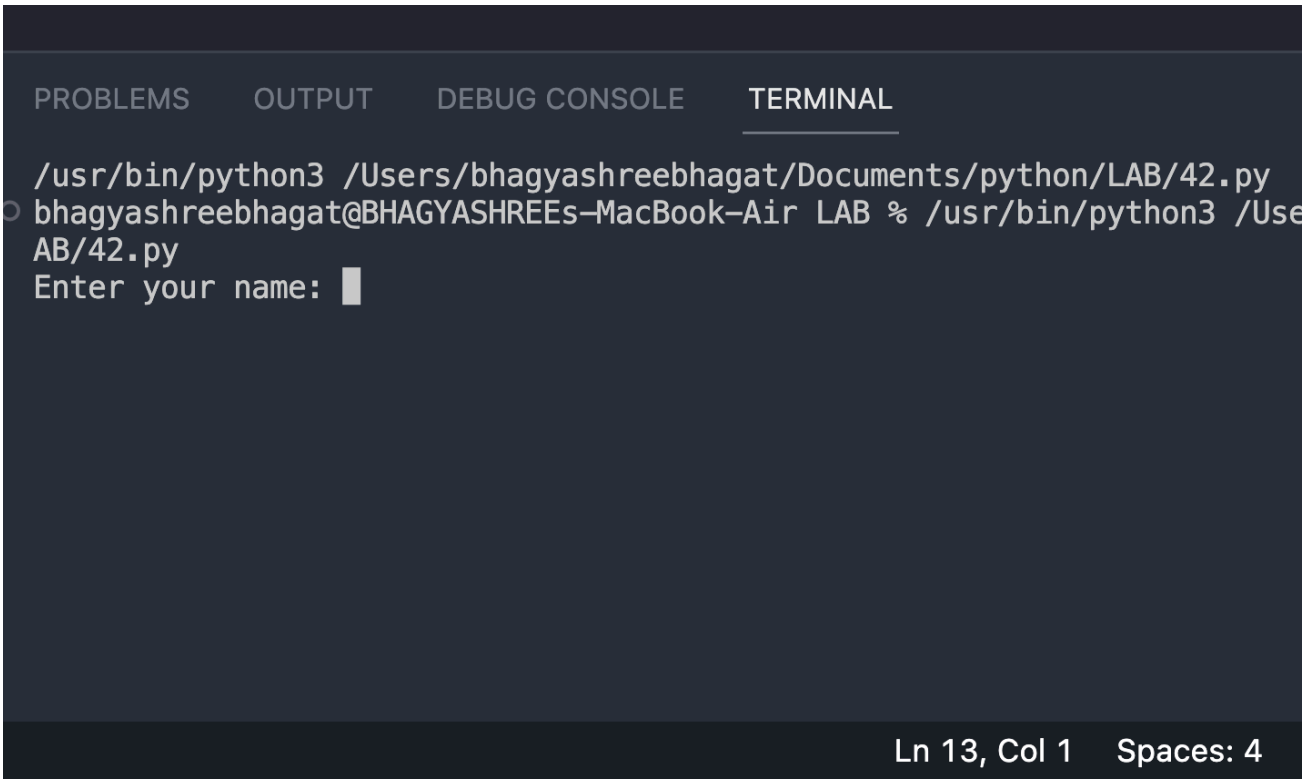
Theory:

The program takes input for name, email, and age from the user using a combination of keyword arguments and positional arguments.

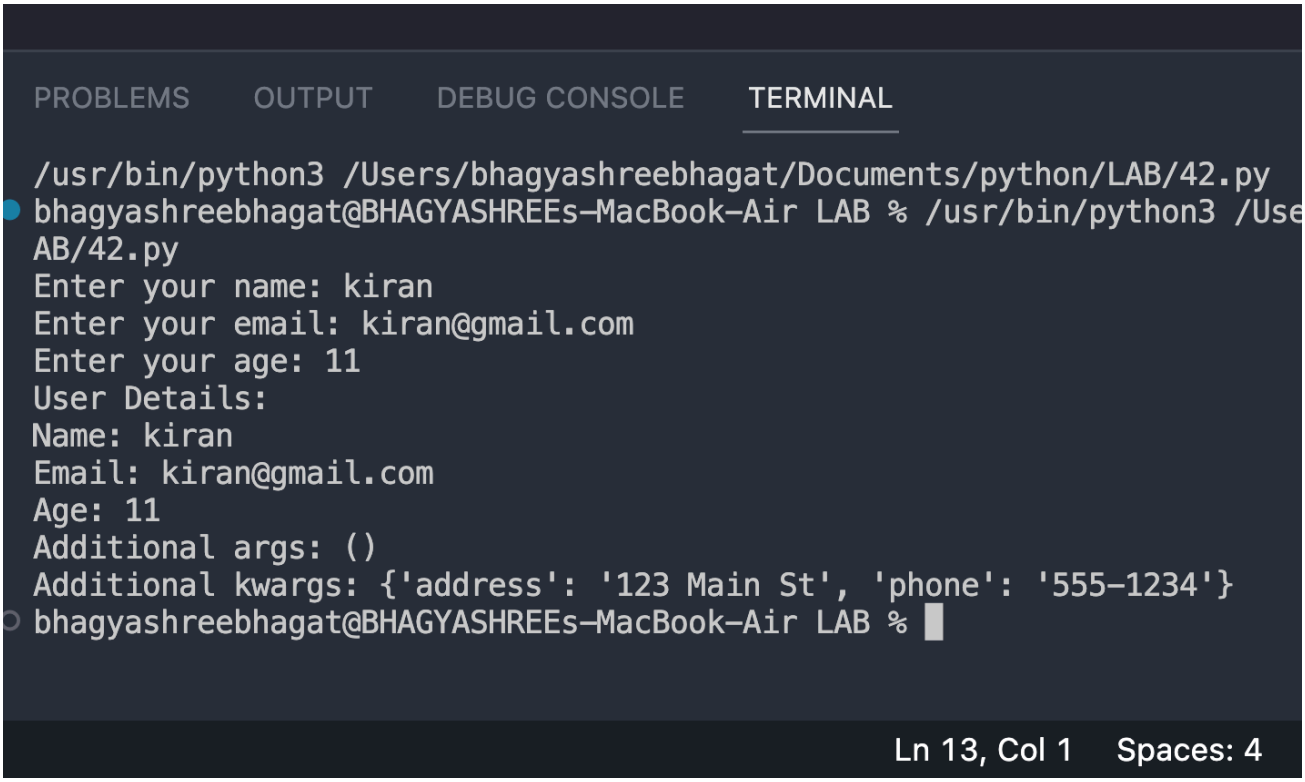
Code:

```
def get_user_details(name, email, age, *args, **kwargs):
    print("User Details:")
    print(f"Name: {name}")
    print(f"Email: {email}")
    print(f"Age: {age}")
    print("Additional args:", args)
    print("Additional kwargs:", kwargs)
user_name = input("Enter your name: ")
user_email = input("Enter your email: ")
user_age = int(input("Enter your age: "))
get_user_details(user_name, user_email, user_age, address="123 Main St", phone="555-1234")
```

Output: (screenshot)



Test Case: Any two (screenshot)



Conclusion:

It demonstrates the use of flexible argument handling in functions, enhancing code readability and usability.

Experiment No:4.3

Title:

Admit Students in Different Departments:

Theory:

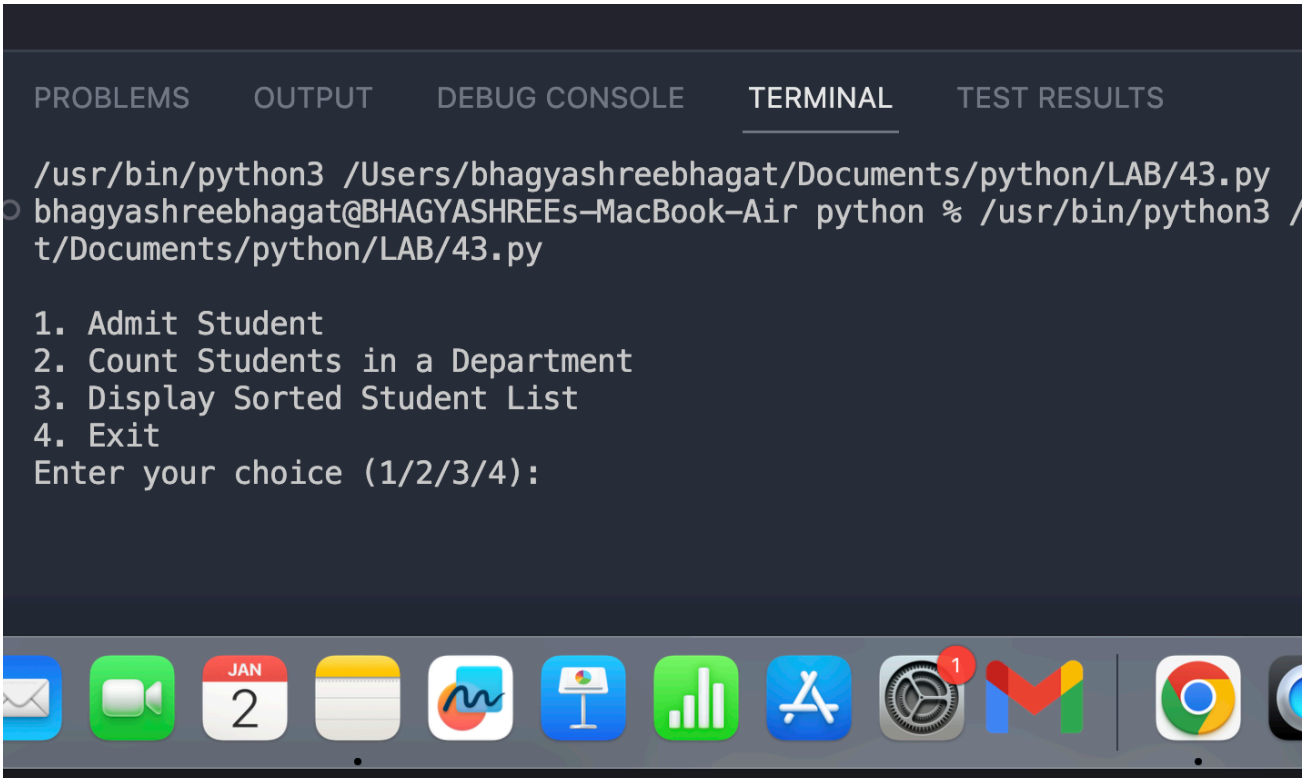
The program uses classes, objects, and constructors to admit students to different departments and counts the number of students.

Code:`class` **Student**:

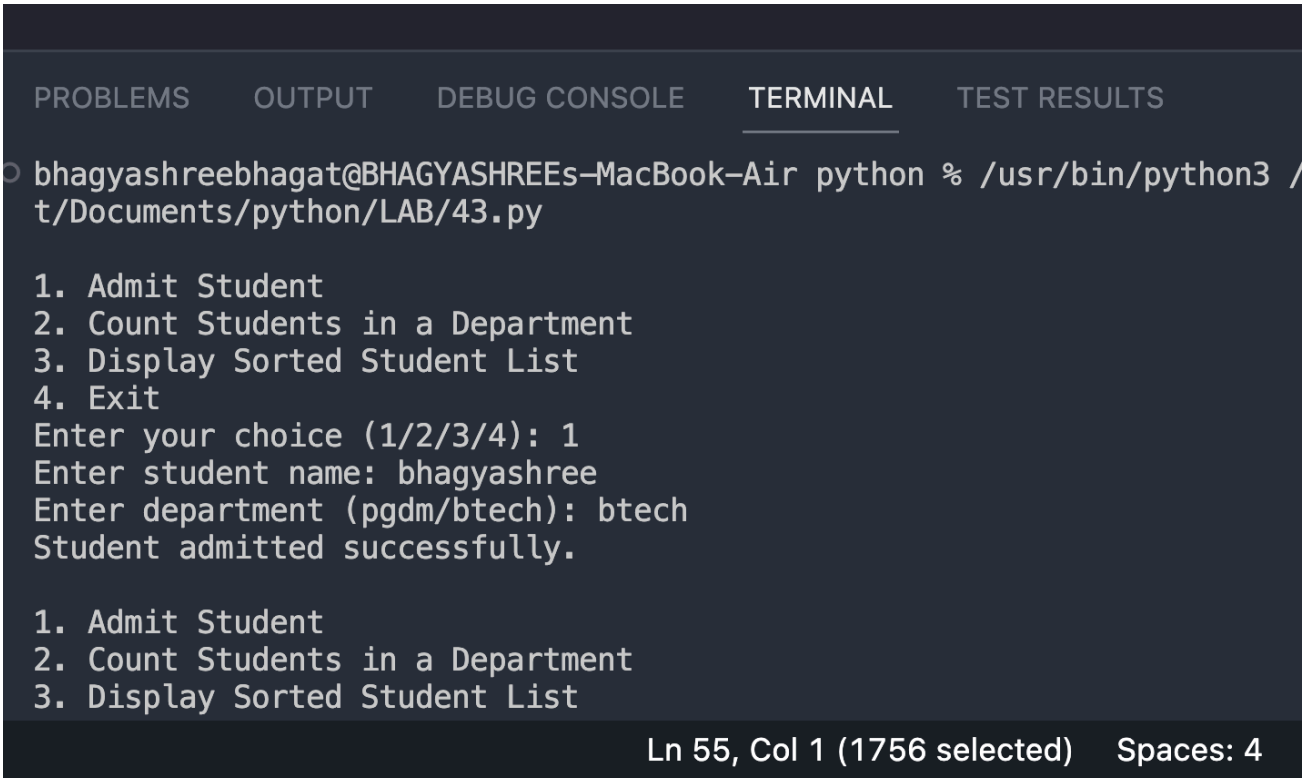
```
    def __init__(self, name, department):
        self.name = name
        self.department = department

class AdmissionSystem:
    def __init__(self):
        self.students = []

    def admit_student(self, name, department):
        student = Student(name, department)
        self.students.append(student)

    def count_students(self, department):
        count = sum(1 for student in self.students if
student.department == department)
        return count

    def sort_students(self):
        self.students.sort(key=lambda x: x.name)

if __name__ == "__main__":
    admission_system = AdmissionSystem()

    while True:
        print("\n1. Admit Student")
```

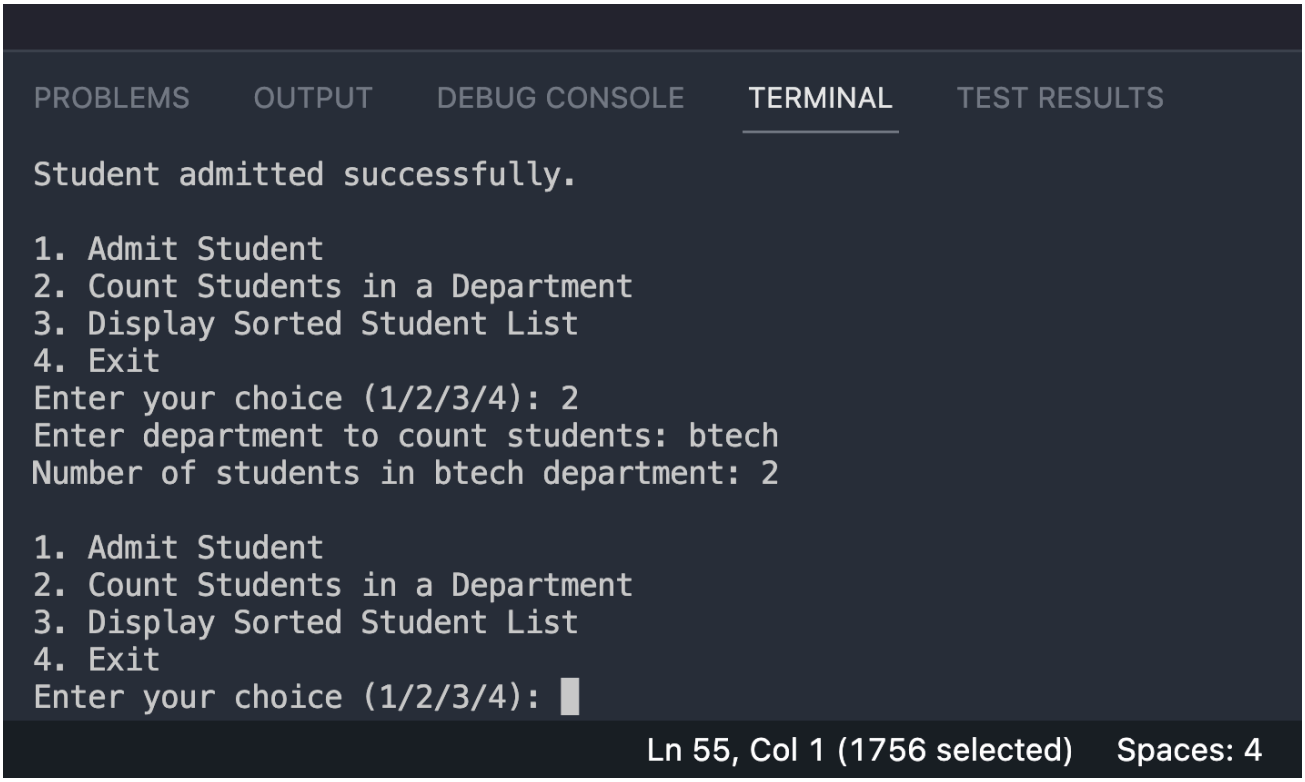
```
        print("2. Count Students in a Department")
        print("3. Display Sorted Student List")
        print("4. Exit")

        choice = input("Enter your choice (1/2/3/4): ")

        if choice == "1":
            name = input("Enter student name: ")
            department = input("Enter department (pgdm/btech): ")
            admission_system.admit_student(name, department)
            print("Student admitted successfully.")

        elif choice == "2":
            department = input("Enter department to count students:
")
            count = admission_system.count_students(department)
            print(f"Number of students in {department} department:
{count}")

        elif choice == "3":
            admission_system.sort_students()
            print("\nSorted Student List:")
            for student in admission_system.students:
                print(f"{student.name} - {student.department}")

        elif choice == "4":
            print("Exiting program.")
            break

        else:
            print("Invalid choice. Please enter a valid option.")
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

It illustrates the concept of classes and objects in object-oriented programming, including the use of constructors.

Experiment No:4.4

Title:

Class Store for Product Records and Billing:

Theory:

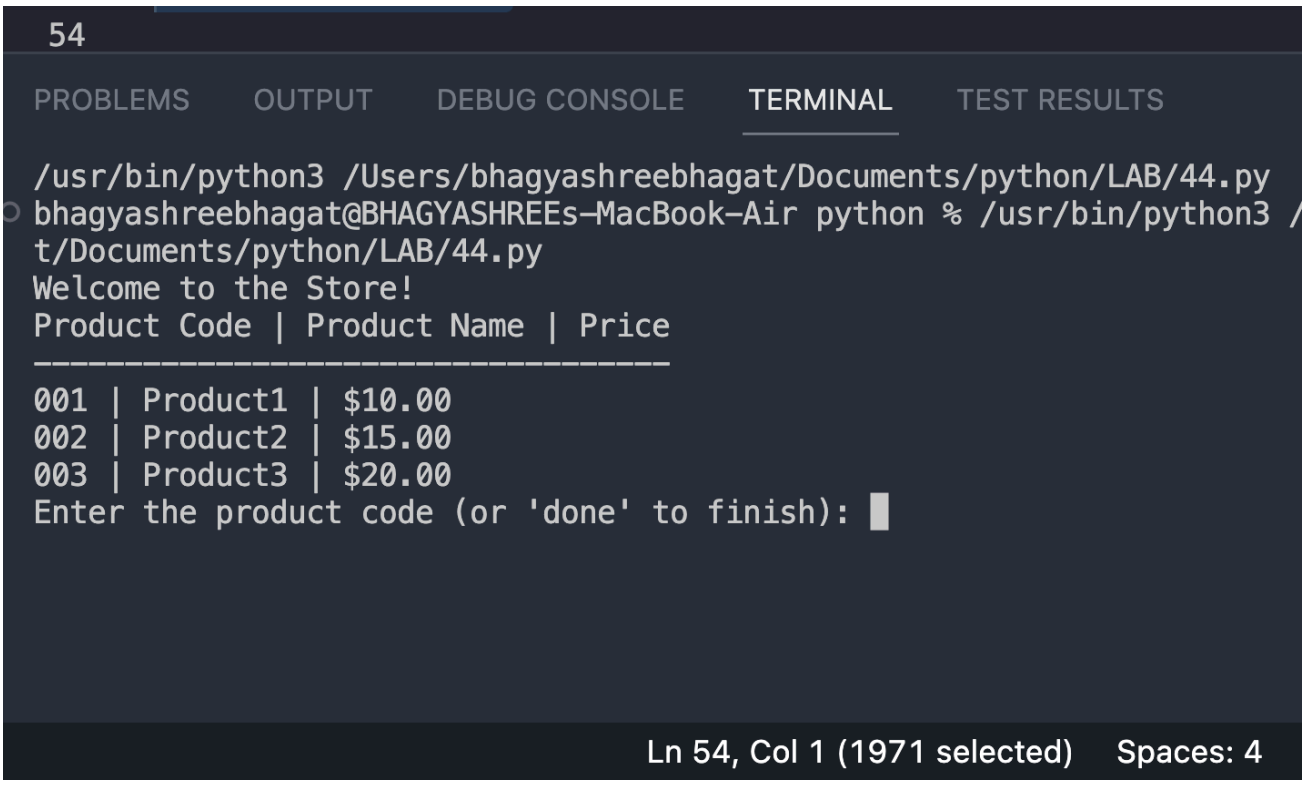
This program creates a class "Store" to manage product records, display a menu, take quantity input, generate a bill, and display the total amount.

Code:

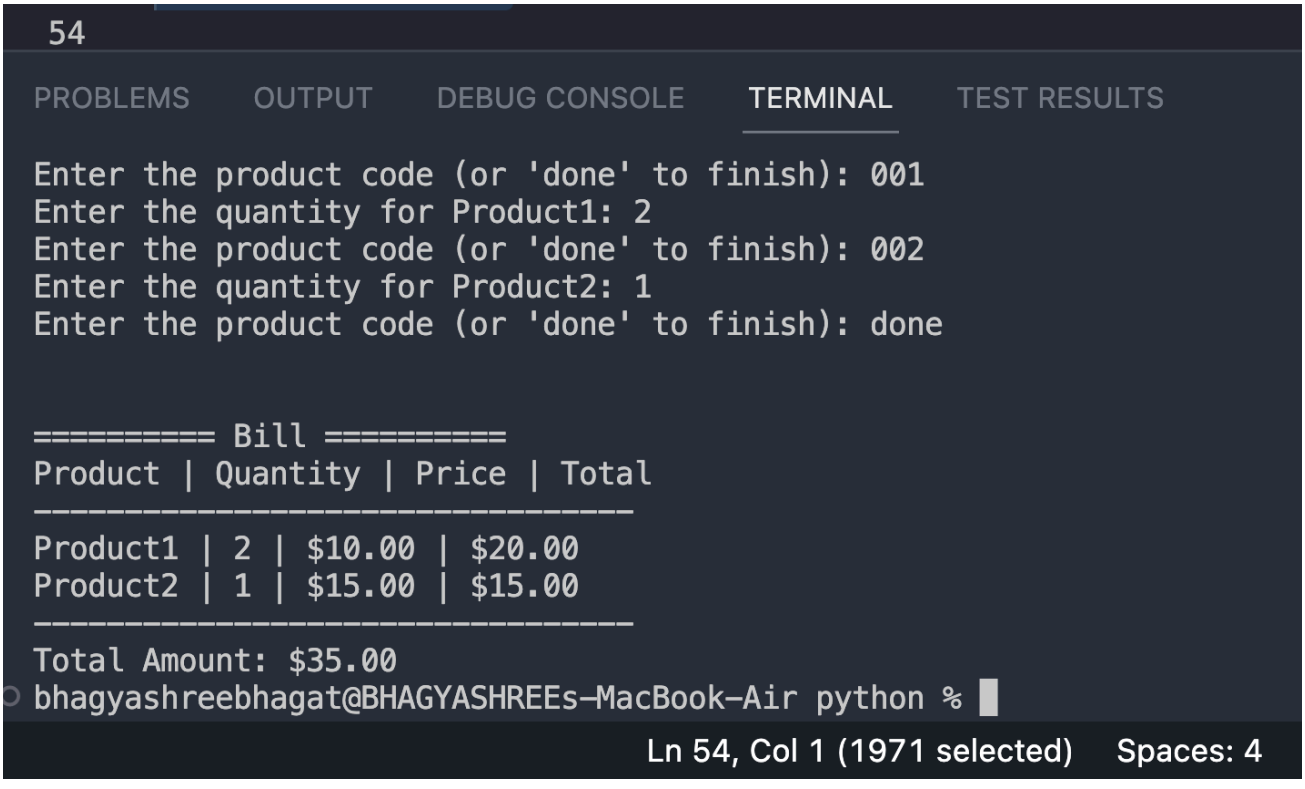
```
class Store:
    def __init__(self):
        self.products = {
            '001': {'name': 'Product1', 'price': 10.0},
            '002': {'name': 'Product2', 'price': 15.0},
            '003': {'name': 'Product3', 'price': 20.0},
        }
        self.cart = {}

    def display_menu(self):
        print("Product Code | Product Name | Price")
        print("-----------------------------------")
        for code, info in self.products.items():
            print(f"{code} | {info['name']} | $
{info['price']:.2f}")

    def take_order(self):
        while True:
            code = input("Enter the product code (or 'done' to
finish): ")
            if code.lower() == 'done':
                break

            if code in self.products:
                quantity = int(input(f"Enter the quantity for
{self.products[code]['name']}: "))
                if code in self.cart:
                    self.cart[code]['quantity'] += quantity
                else:
                    self.cart[code] = {'name': self.products[code]
['name'], 'price': self.products[code]['price'], 'quantity':
quantity}
```

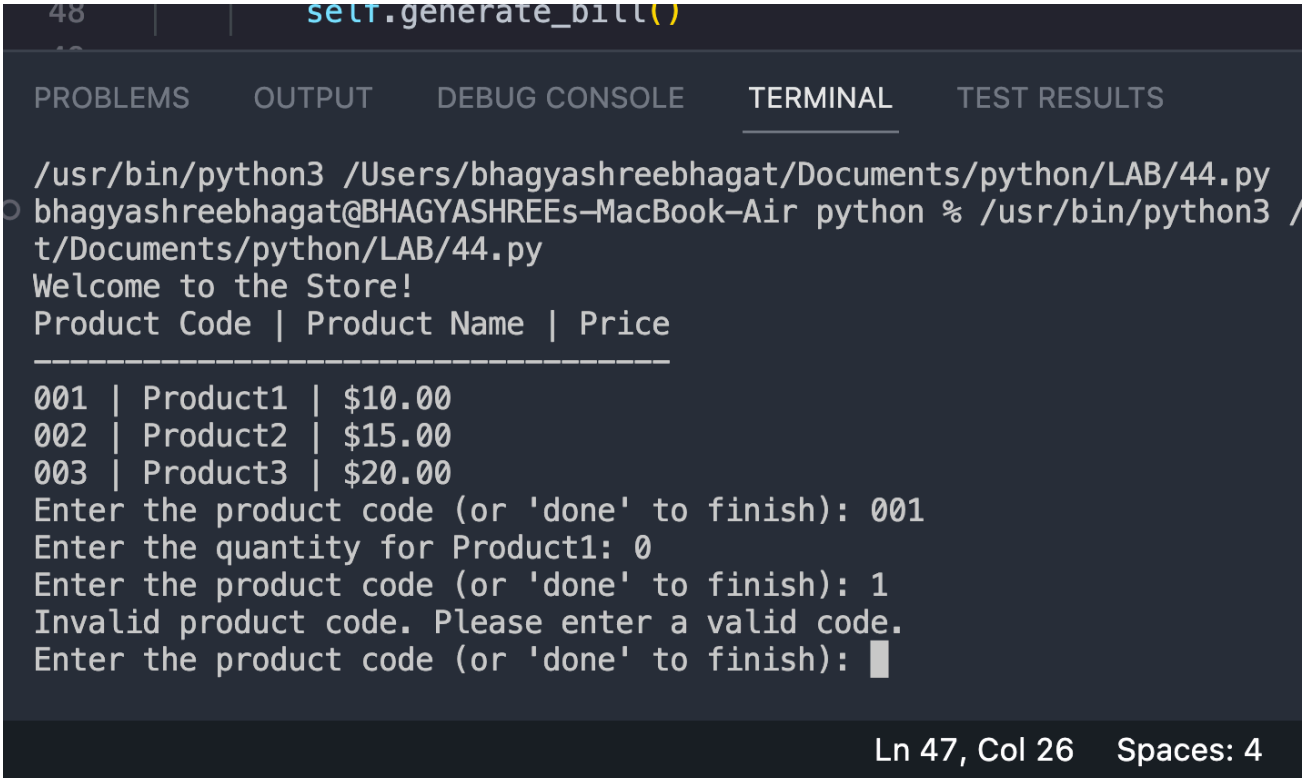
```
            else:
                print("Invalid product code. Please enter a valid
code.")

    def generate_bill(self):
        total_amount = 0.0
        print("\n\n========== Bill ==========")
        print("Product | Quantity | Price | Total")
        print("-------------------------------")
        for code, item in self.cart.items():
            total_item_price = item['price'] * item['quantity']
            total_amount += total_item_price
            print(f"{item['name']} | {item['quantity']} | $
{item['price']:.2f} | ${total_item_price:.2f}")

        print("-------------------------------")
        print(f"Total Amount: ${total_amount:.2f}")

    def run(self):
        print("Welcome to the Store!")
        self.display_menu()
        self.take_order()
        self.generate_bill()


if __name__ == "__main__":
    my_store = Store()
    my_store.run()
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

It provides a practical example of creating a class to model a store's inventory and billing system.

Experiment No:4.5

Title:

Input for Addition of Two Numbers Using Single Inheritance:

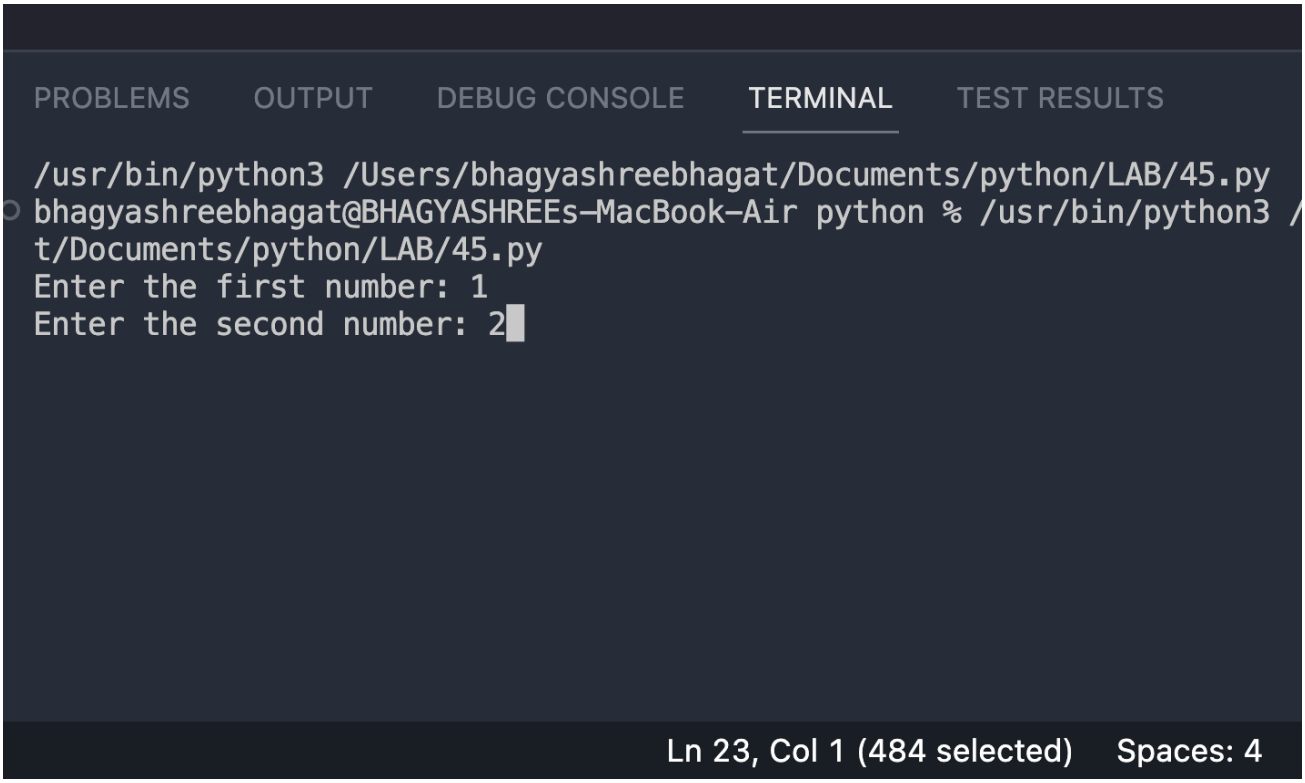
Theory:

The program demonstrates single inheritance by creating a child class for the addition of two numbers.

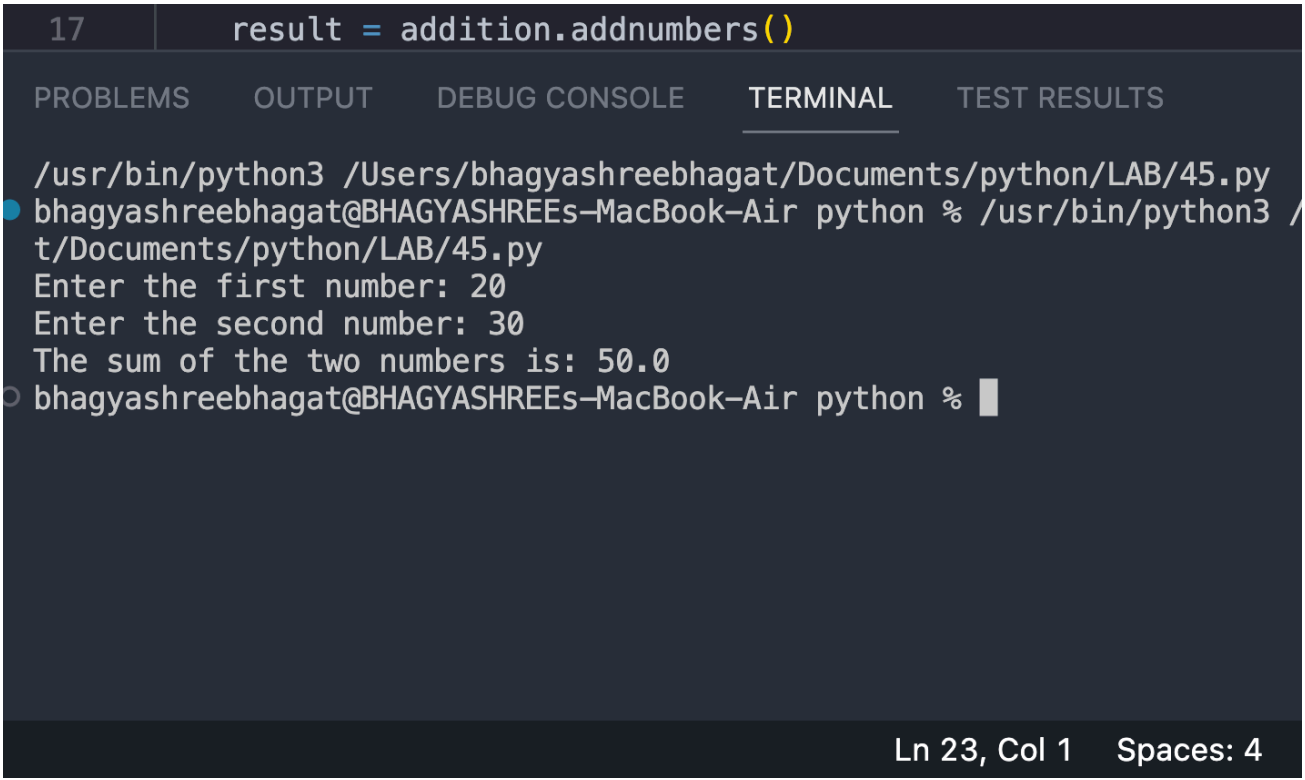
Code:

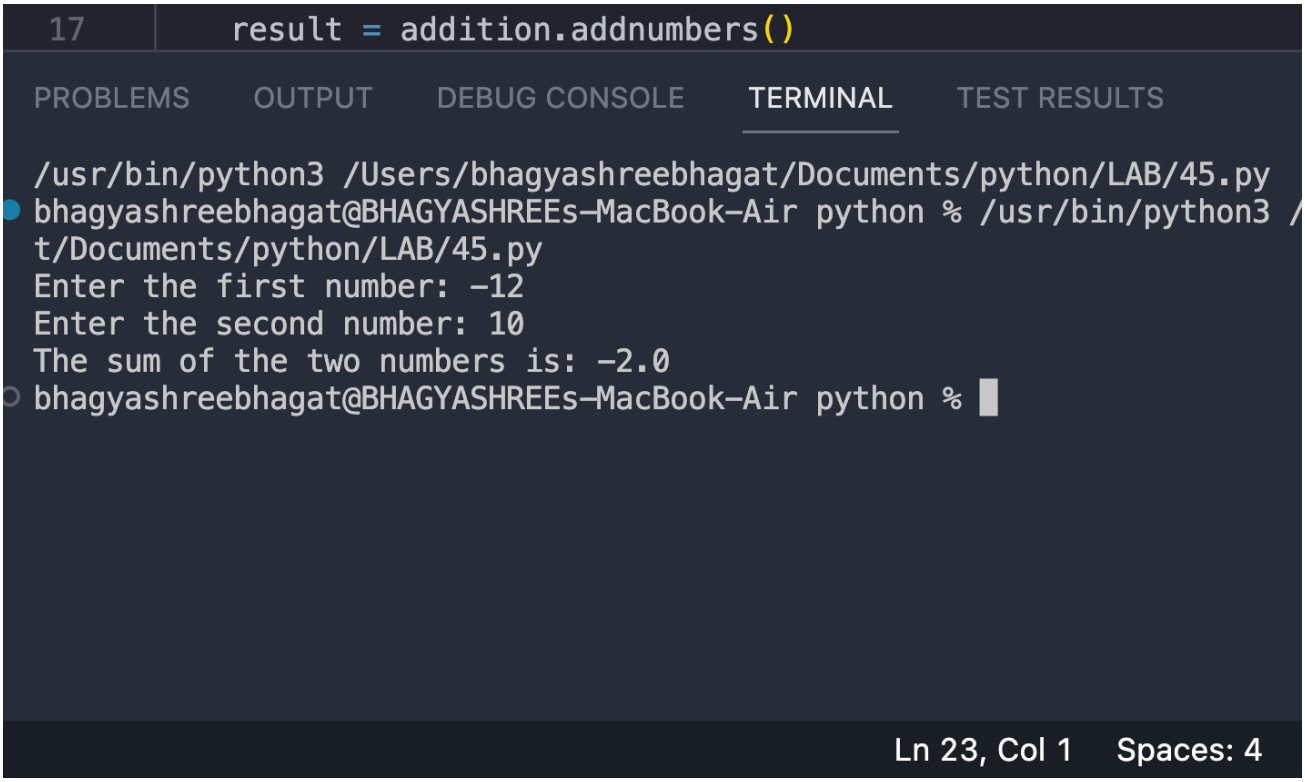
```
class Input:
    def getinput(self):
        num1 = float(input("Enter the first number: "))
        num2 = float(input("Enter the second number: "))
        return num1, num2

class Addition(Input):
    def addnumbers(self):
        num1, num2 = self.getinput()
        result = num1 + num2
        return result

def main():
    addition = Addition()
    result = addition.addnumbers()
    print(f"The sum of the two numbers is: {result}")

if __name__ == "__main__":
    main()
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

It showcases the concept of inheritance and how it can be used to reuse code in object-oriented programming.

Experiment No:4.6

Title:

Create Two Base Classes LU and ITM and One Derived Class

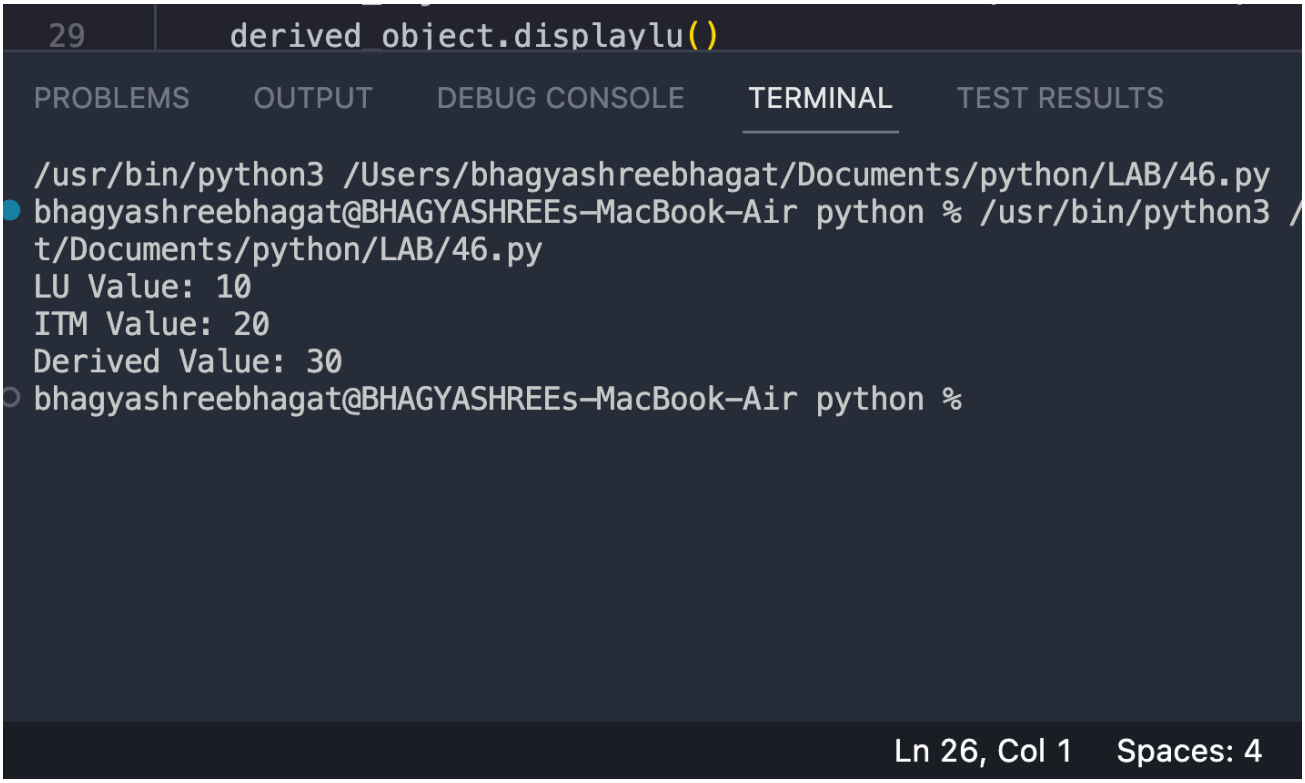
Theory:

This program implements multiple inheritance with two base classes (LU and ITM) and one derived class.

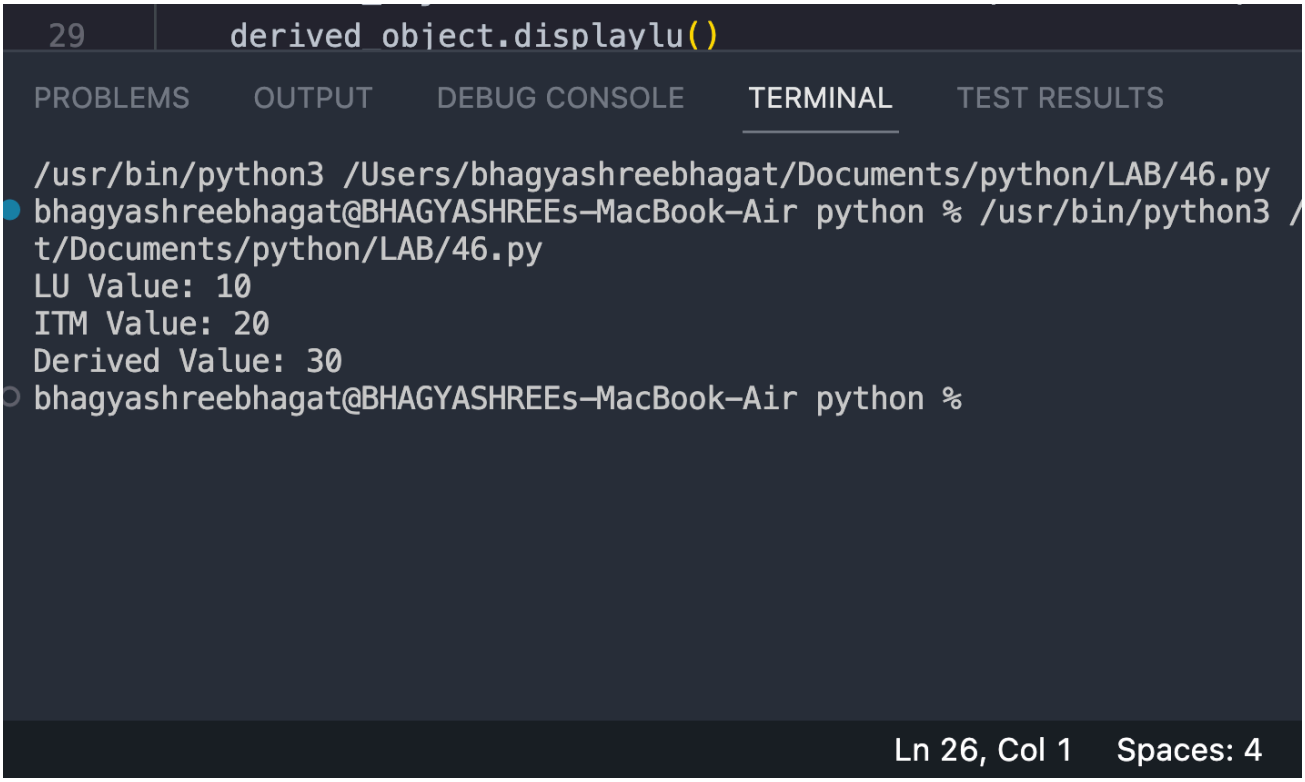
Code:

```
class LU:
    def __init__(self, luvalue):
        self.luvalue = luvalue

    def displaylu(self):
        print(f"LU Value: {self.luvalue}")


class ITM:
    def __init__(self, itmvalue):
        self.itmvalue = itmvalue

    def displayitm(self):
        print(f"ITM Value: {self.itmvalue}")


class DerivedClass(LU, ITM):
    def __init__(self, luvalue, itmvalue, derivedvalue):
        LU.__init__(self, luvalue)
        ITM.__init__(self, itmvalue)
        self.derivedvalue = derivedvalue

    def displayderived(self):
        print(f"Derived Value: {self.derivedvalue}")


if __name__ == "__main__":
    derived_object = DerivedClass(luvalue=10, itmvalue=20,
derivedvalue=30)
    derived_object.displaylu()
    derived_object.displayitm()
    derived_object.displayderived()
```

Output: (screenshot)



Test Case: Any two (screenshot)



Conclusion:

It illustrates how a class can inherit from multiple base classes, promoting code reuse and modularity.

Experiment No:4.7

Title:

Implement Multilevel Inheritance:

Theory:

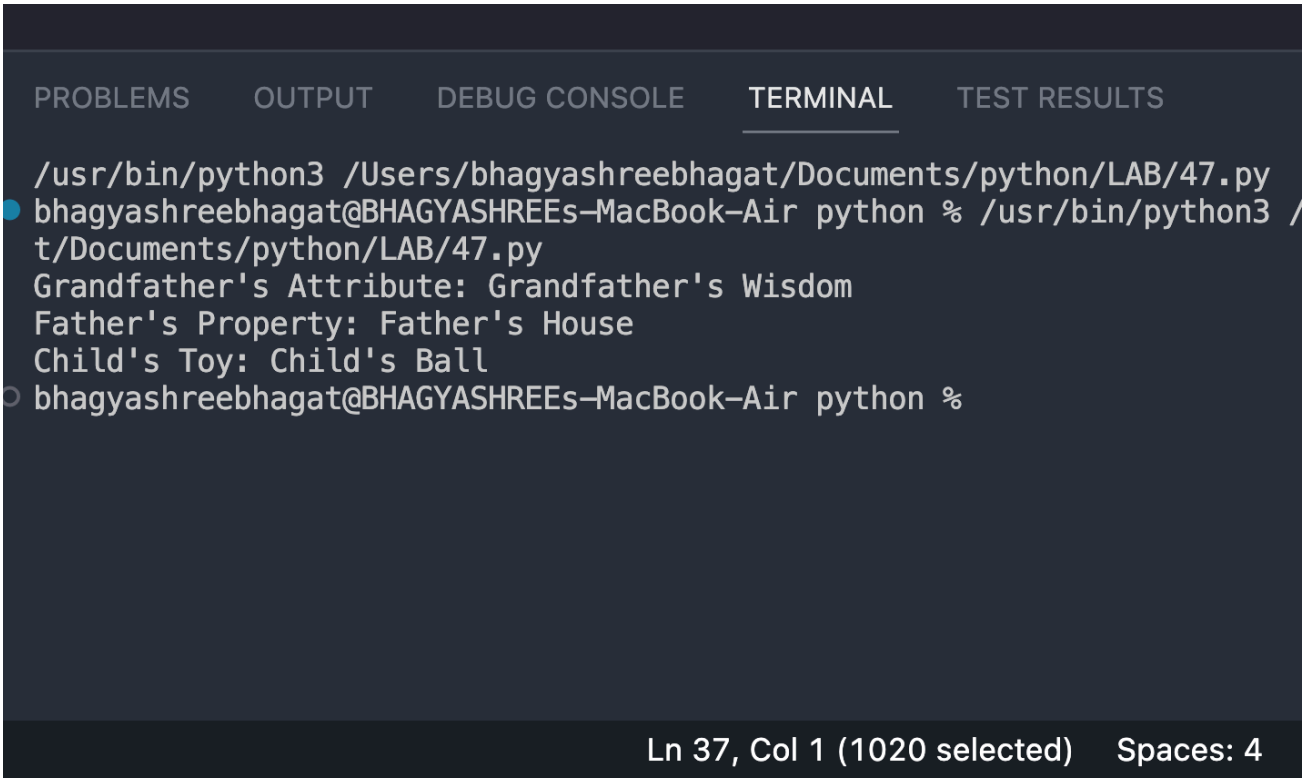
The program uses multilevel inheritance to show property inheritance from grandfather to child.

Code:

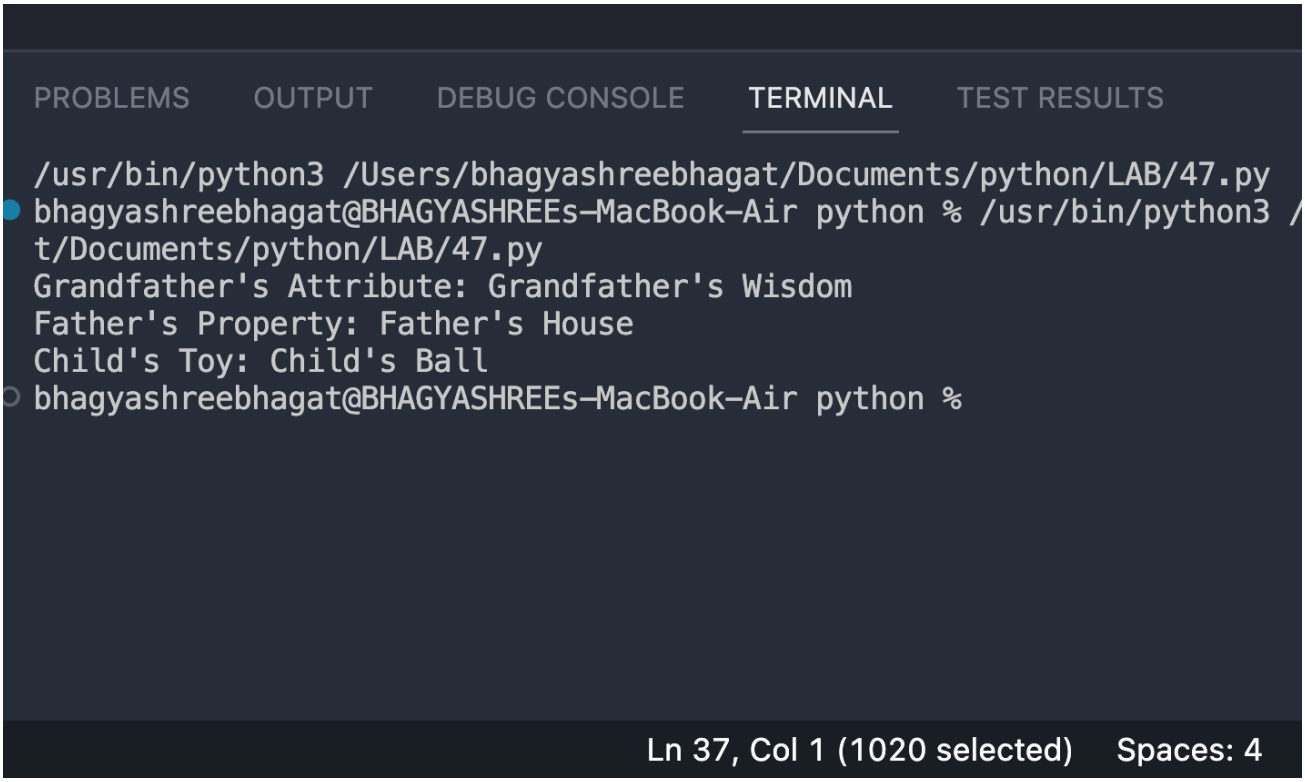
```
class Grandfather:
    def __init__(self, attribute):
        self.attribute = attribute

    def displaygrandfatherattribute(self):
        print(f"Grandfather's Attribute: {self.attribute}")


class Father(Grandfather):
    def __init__(self, attribute, propertyfather):
        super().__init__(attribute)
        self.propertyfather = propertyfather

    def displayfatherproperty(self):
        print(f"Father's Property: {self.propertyfather}")


class Child(Father):
    def __init__(self, attribute, propertyfather, toy):
        super().__init__(attribute, propertyfather)
        self.toy = toy

    def displaychildtoy(self):
        print(f"Child's Toy: {self.toy}")


if __name__ == "__main__":
    grandfatherattribute = "Grandfather's Wisdom"
    fatherproperty = "Father's House"
    childtoy = "Child's Ball"

    childobject = Child(grandfatherattribute, fatherproperty,
childtoy)
```

```
    childobject.displaygrandfatherattribute()
    childobject.displayfatherproperty()
    childobject.displaychildtoy()
```

Output: (screenshot)



Test Case: Any two (screenshot)



Conclusion:

It demonstrates the hierarchical organization of classes in a multilevel inheritance structure.

Experiment No:4.8

Title:

Design Library Catalogue System Using Inheritance:

Theory:

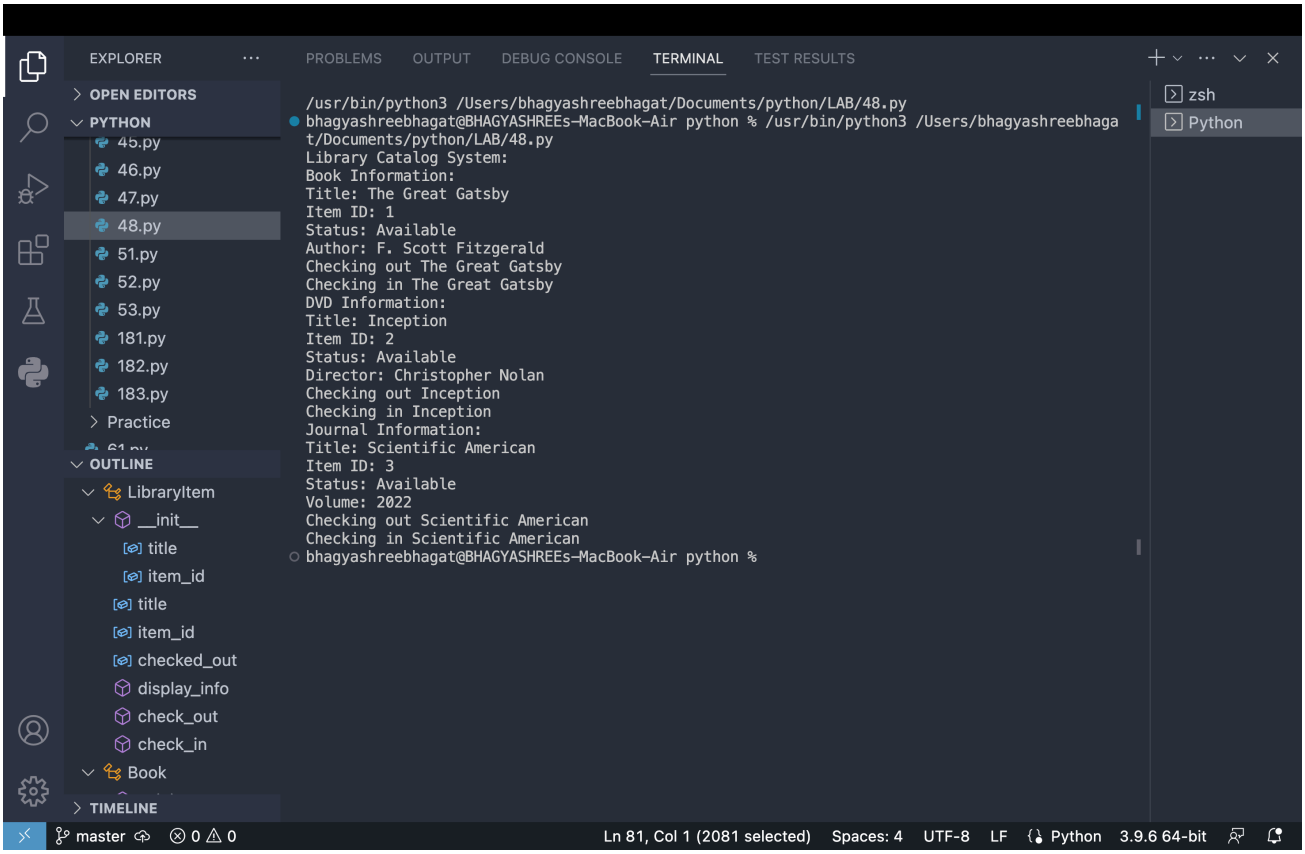
The program designs a library catalogue system using inheritance with a base class "LibraryItem" and derived classes "Book," "DVD," and "Journal."

Code:

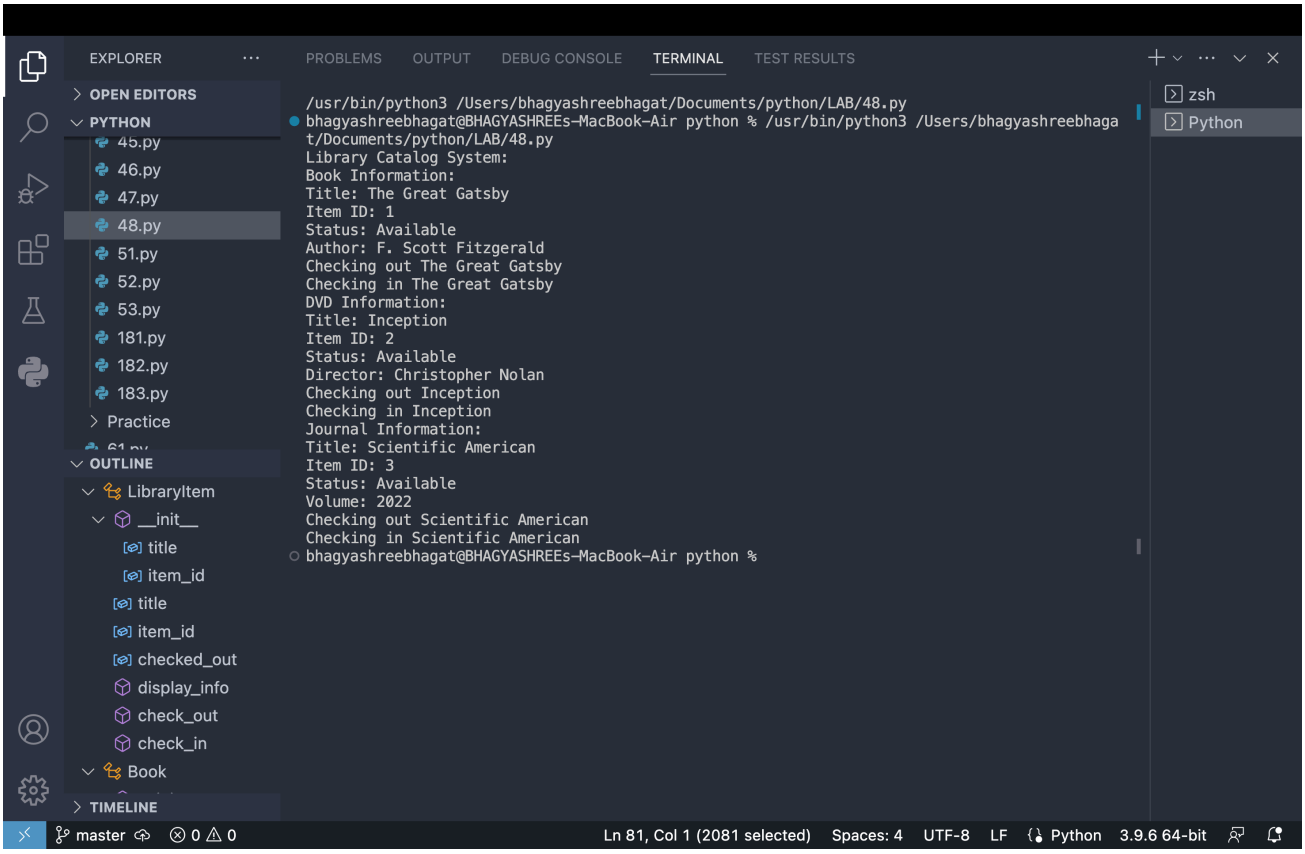
```
class LibraryItem:
    def __init__(self, title, item_id):
        self.title = title
        self.item_id = item_id
        self.checked_out = False

    def display_info(self):
        print(f"Title: {self.title}")
        print(f"Item ID: {self.item_id}")
        print("Status: Checked Out" if self.checked_out else
"Status: Available")

    def check_out(self):
        if not self.checked_out:
            print(f"Checking out {self.title}")
            self.checked_out = True
        else:
            print(f"{self.title} is already checked out")

    def check_in(self):
        if self.checked_out:
            print(f"Checking in {self.title}")
            self.checked_out = False
        else:
            print(f"{self.title} is already checked in")


class Book(LibraryItem):
    def __init__(self, title, item_id, author):
        super().__init__(title, item_id)
        self.author = author

    def display_info(self):
        print("Book Information:")
        super().display_info()
        print(f"Author: {self.author}")
```

```python
class DVD(LibraryItem):
    def __init__(self, title, item_id, director):
        super().__init__(title, item_id)
        self.director = director

    def display_info(self):
        print("DVD Information:")
        super().display_info()
        print(f"Director: {self.director}")


class Journal(LibraryItem):
    def __init__(self, title, item_id, volume):
        super().__init__(title, item_id)
        self.volume = volume

    def display_info(self):
        print("Journal Information:")
        super().display_info()
        print(f"Volume: {self.volume}")


def main():
    book = Book("The Great Gatsby", 1, "F. Scott Fitzgerald")
    dvd = DVD("Inception", 2, "Christopher Nolan")
    journal = Journal("Scientific American", 3, 2022)

    print("Library Catalog System:")
    book.display_info()
    book.check_out()
    book.check_in()

    dvd.display_info()
    dvd.check_out()
    dvd.check_in()

    journal.display_info()
    journal.check_out()
    journal.check_in()


if __name__ == "__main__":
    main()
```

Output: (screenshot)



Test Case: Any two (screenshot)



Conclusion:

It exemplifies the use of inheritance and method overriding to create a system with a structured hierarchy for library items.

Experiment No:5.1

Title:

Create my_module for Addition of Two Numbers:

Theory:

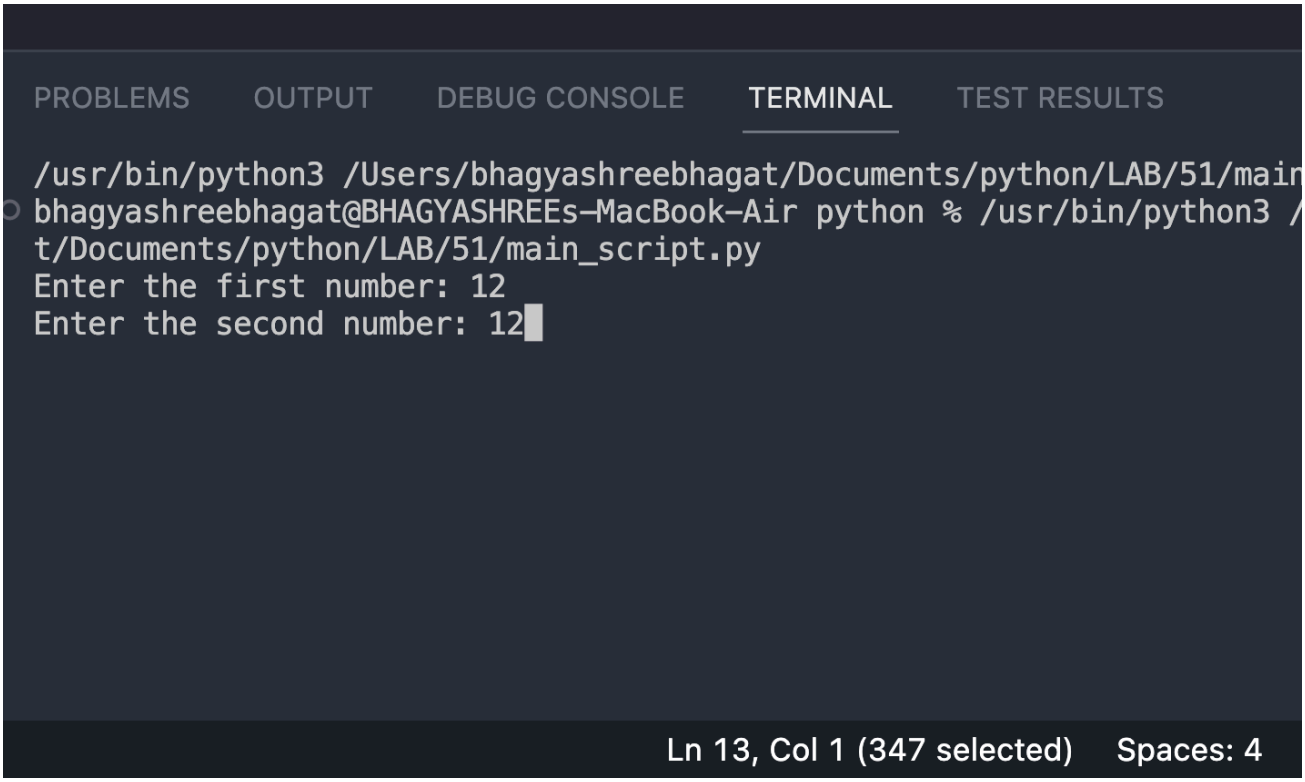
This program writes a module (my_module.py) for the addition of two numbers and imports it into the main script.

Code:

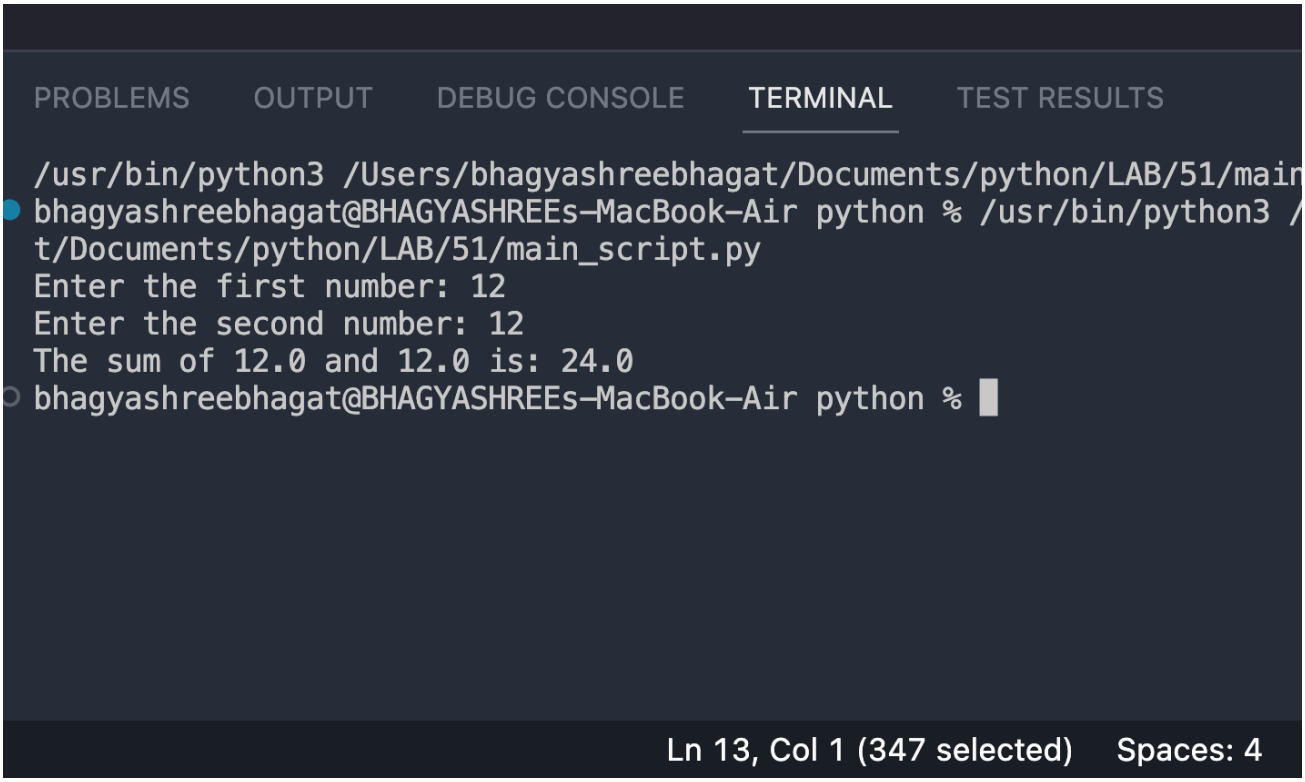
```python
# main_script.py
import my_module

# Taking user input for two numbers
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))

# Using the add_numbers function from the imported module
result = my_module.add_numbers(num1, num2)

# Displaying the result
print(f"The sum of {num1} and {num2} is: {result}")
```

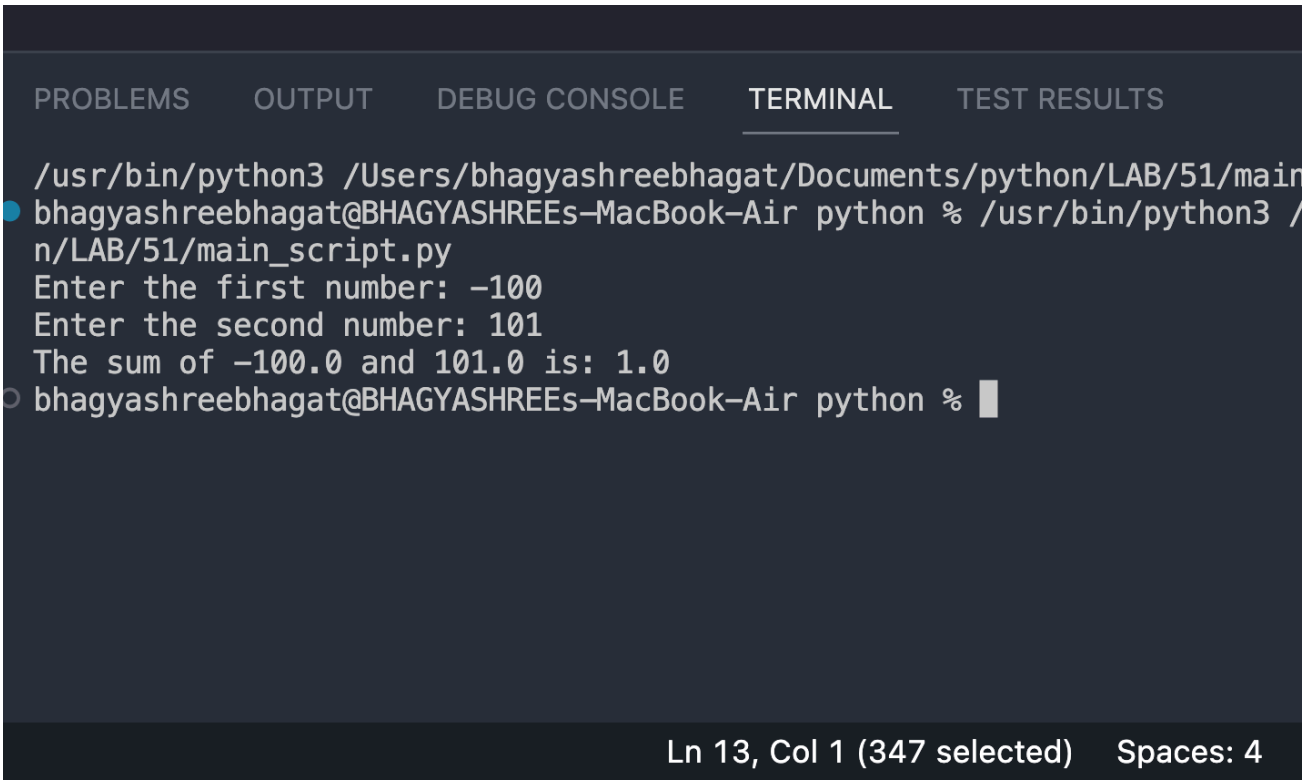
```python
def add_numbers(num1, num2):
    return num1 + num2
```

Output: (screenshot)



Test Case: Any two (screenshot)


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
TEST RESULTS
/usr/bin/python3 /Users/bhagyashreebhagat/Documents/python/LAB/51/main
bhagyashreebhagat@BHAGYASHREEs-MacBook-Air python % /usr/bin/python3 /
t/Documents/python/LAB/51/main_script.py
Enter the first number: 12
Enter the second number: 12
The sum of 12.0 and 12.0 is: 24.0
bhagyashreebhagat@BHAGYASHREEs-MacBook-Air python %
Ln 13, Col 1 (347 selected)
Spaces: 4


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
TEST RESULTS
/usr/bin/python3 /Users/bhagyashreebhagat/Documents/python/LAB/51/main
bhagyashreebhagat@BHAGYASHREEs-MacBook-Air python % /usr/bin/python3 /
n/LAB/51/main_script.py
Enter the first number: -100
Enter the second number: 101
The sum of -100.0 and 101.0 is: 1.0
bhagyashreebhagat@BHAGYASHREEs-MacBook-Air python %
Ln 13, Col 1 (347 selected)
Spaces: 4

Conclusion:

It introduces the concept of modular programming, demonstrating how to organize code into separate modules for reusability.

Experiment No:5.2

Title:

Create Bank Module for Balance, Withdraw, and Deposit:

Theory:

The program designs a bank module (bank_module.py) to perform operations such as checking the balance, withdrawing, and depositing money, and imports the module into the main script.

Code:

```
class BankAccount:
    def __init__(self, account_holder, balance=0.0):
        self.account_holder = account_holder
        self.balance = balance

    def check_balance(self):
        return self.balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
```

```
            print(f"Deposit of ${amount} successful. New balance: $
{self.balance}")
        else:
            print("Invalid deposit amount. Please enter a positive
amount.")

    def withdraw(self, amount):
        if amount > 0 and amount <= self.balance:
            self.balance -= amount
            print(f"Withdrawal of ${amount} successful. New
balance: ${self.balance}")
        else:
            print("Invalid withdrawal amount or insufficient
funds.")
```

```
from bank_module import BankAccount

def main():
    account_holder_name = input("Enter account holder's name: ")
    initial_balance = float(input("Enter initial balance: "))

    account = BankAccount(account_holder_name, initial_balance)

    while True:
        print("\nBank Operations:")
        print("1. Check Balance")
        print("2. Deposit")
        print("3. Withdraw")
        print("4. Exit")

        choice = input("Enter your choice (1/2/3/4): ")

        if choice == '1':
            print(f"Current Balance: ${account.check_balance()}")
        elif choice == '2':
            deposit_amount = float(input("Enter the deposit amount:
"))
            account.deposit(deposit_amount)
        elif choice == '3':
            withdrawal_amount = float(input("Enter the withdrawal
amount: "))
            account.withdraw(withdrawal_amount)
        elif choice == '4':
```

```
            print("Exiting program. Goodbye!")
            break
        else:
            print("Invalid choice. Please enter 1, 2, 3, or 4.")

if __name__ == "__main__":
    main()
```

Output: (screenshot)

Test Case: Any two (screenshot)

```
bhagyashreebhagat@BHAGYASHREEs-MacBook-Air python % /usr/bin/python3 /
n/LAB/BankAccount/main_script.py
Enter account holder's name: bhagyashree
Enter initial balance: 1000

Bank Operations:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice (1/2/3/4): 1
Current Balance: $1000.0

Bank Operations:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice (1/2/3/4):
```

```
Bank Operations:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice (1/2/3/4): 2
Enter the deposit amount: 1000
Deposit of $1000.0 successful. New balance: $2000.0

Bank Operations:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice (1/2/3/4):
```

Conclusion:

It illustrates how to create modular components for banking operations and integrate them into a main script.

Experiment No:5.3

Title:

Create a Package "cars" with Different Modules:

Theory:

This program creates a package named "cars" with modules (BMW, AUDI, NISSAN) having classes and functionality, and imports them into the main file "cars."

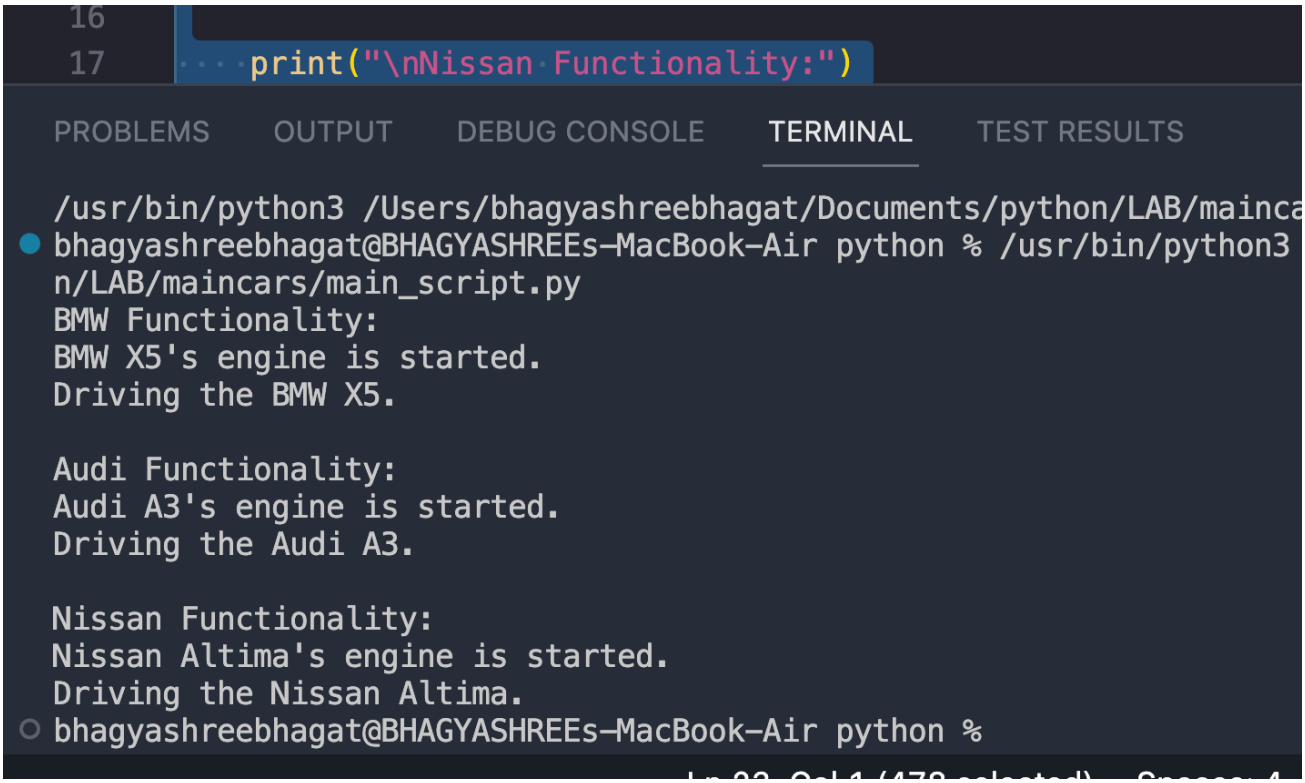
Code:

```python
# audi.py

class Audi:
    def __init__(self, model):
        self.model = model

    def start_engine(self):
        print(f"Audi {self.model}'s engine is started.")

    def drive(self):
        print(f"Driving the Audi {self.model}.")
```

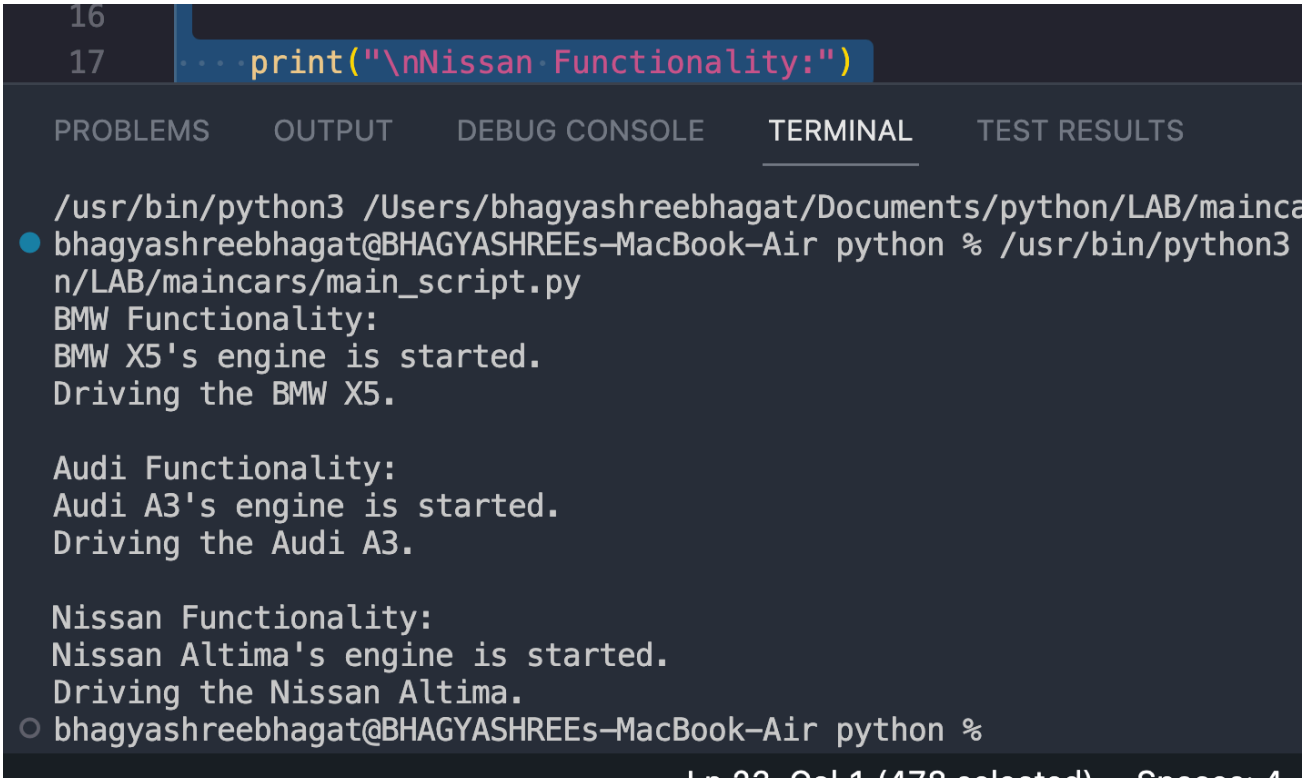
```python
# bmw.py

class BMW:
    def __init__(self, model):
        self.model = model

    def start_engine(self):
        print(f"BMW {self.model}'s engine is started.")

    def drive(self):
        print(f"Driving the BMW {self.model}.")



class Nissan:
    def __init__(self, model):
        self.model = model

    def start_engine(self):
        print(f"Nissan {self.model}'s engine is started.")

    def drive(self):
        print(f"Driving the Nissan {self.model}.")



# main_script.py
from cars import bmw, audi, nissan

def main():
    bmw_car = bmw.BMW(model="X5")
    audi_car = audi.Audi(model="A3")
    nissan_car = nissan.Nissan(model="Altima")

    print("BMW Functionality:")
    bmw_car.start_engine()
    bmw_car.drive()

    print("\nAudi Functionality:")
    audi_car.start_engine()
    audi_car.drive()

    print("\nNissan Functionality:")
    nissan_car.start_engine()
    nissan_car.drive()

if __name__ == "__main__":
```

```
main()
```

Output: (screenshot)



Test Case: Any two (screenshot)



Conclusion:It demonstrates the organization of code into packages and modules, promoting modularity and code organization.

Experiment No:6.1

Title:

Implement Multithreading:

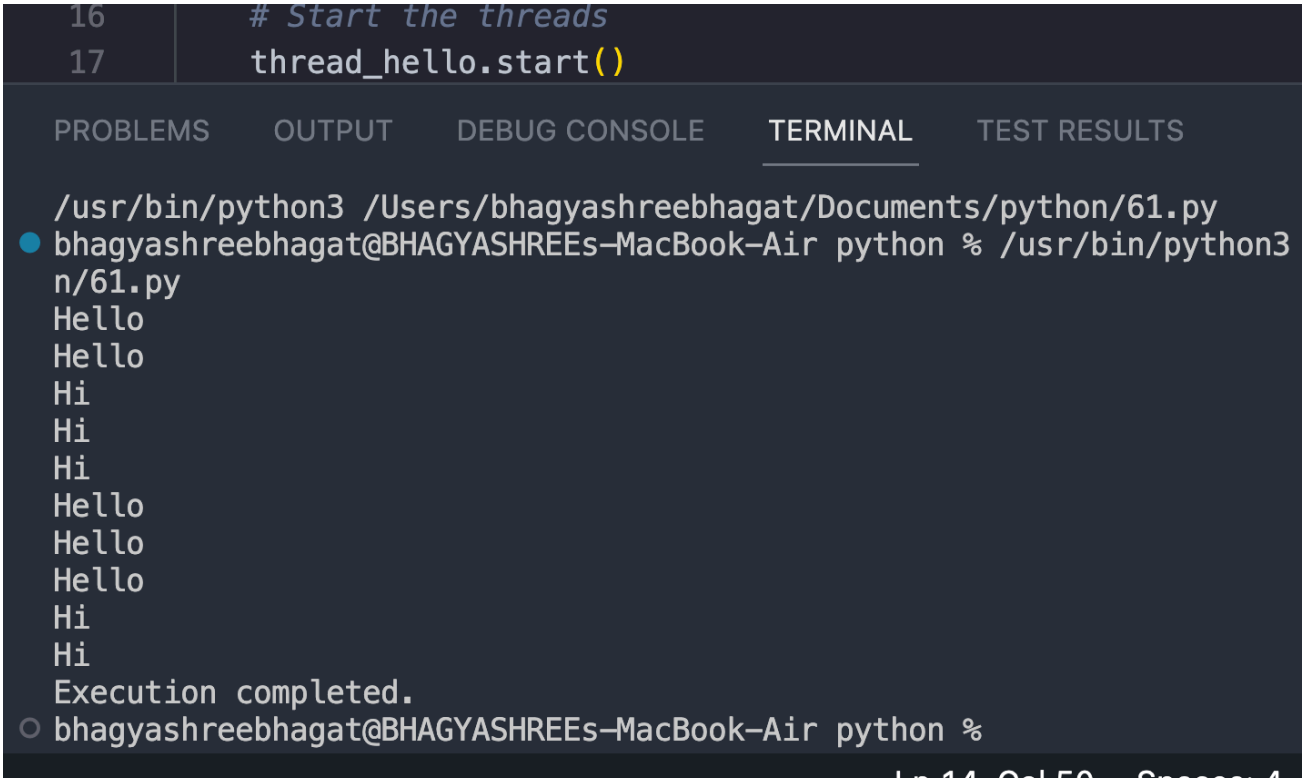
Theory:

The program uses the threading module to implement multithreading, printing "Hello" with one thread and "Hi" with another thread.

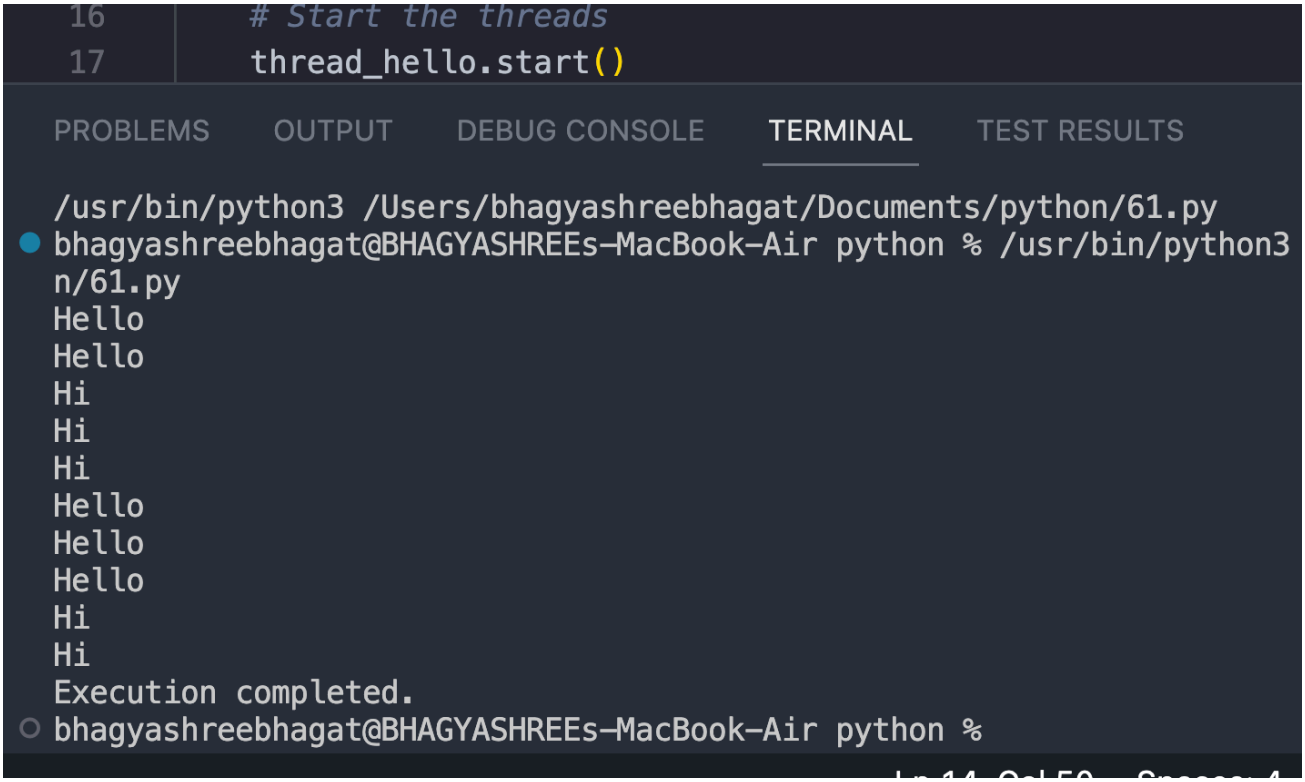
Code:

```
import threading

def print_hello():
    for _ in range(5):
        print("Hello")

def print_hi():
    for _ in range(5):
        print("Hi")

if __name__ == "__main__":
    # Create two threads
    thread_hello = threading.Thread(target=print_hello)
    thread_hi = threading.Thread(target=print_hi)

    # Start the threads
    thread_hello.start()
    thread_hi.start()

    # Wait for both threads to finish
    thread_hello.join()
    thread_hi.join()

    print("Execution completed.")
```

Output: (screenshot)



Test Case: Any two (screenshot)



Conclusion:

It illustrates the concurrent execution of multiple threads, providing an introduction to multithreading in Python.

Experiment No:7.1

Title: Use 'Weather API' to Print Temperature, Sunrise, and Sunset:

Theory:

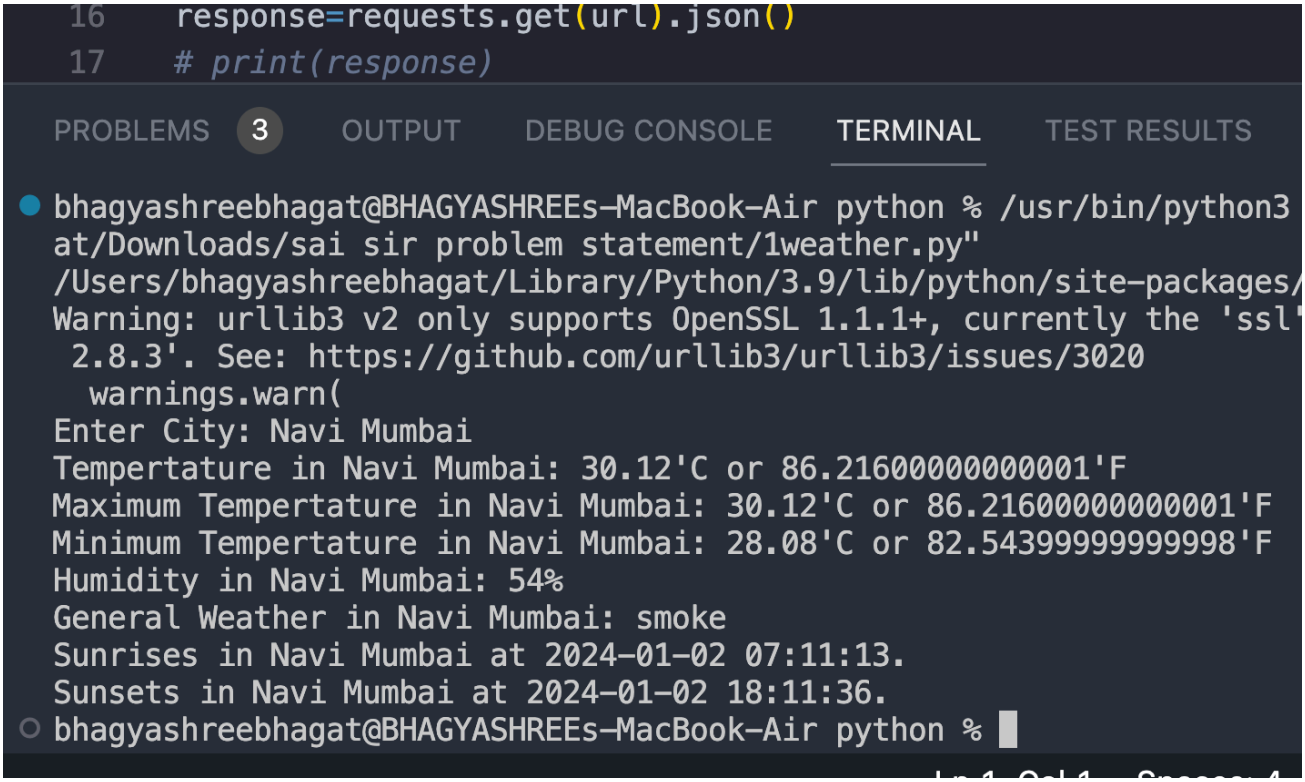
This program utilizes a weather API to print the temperature of a city and the sunrise and sunset times, considering the humidity of that area.

Code:

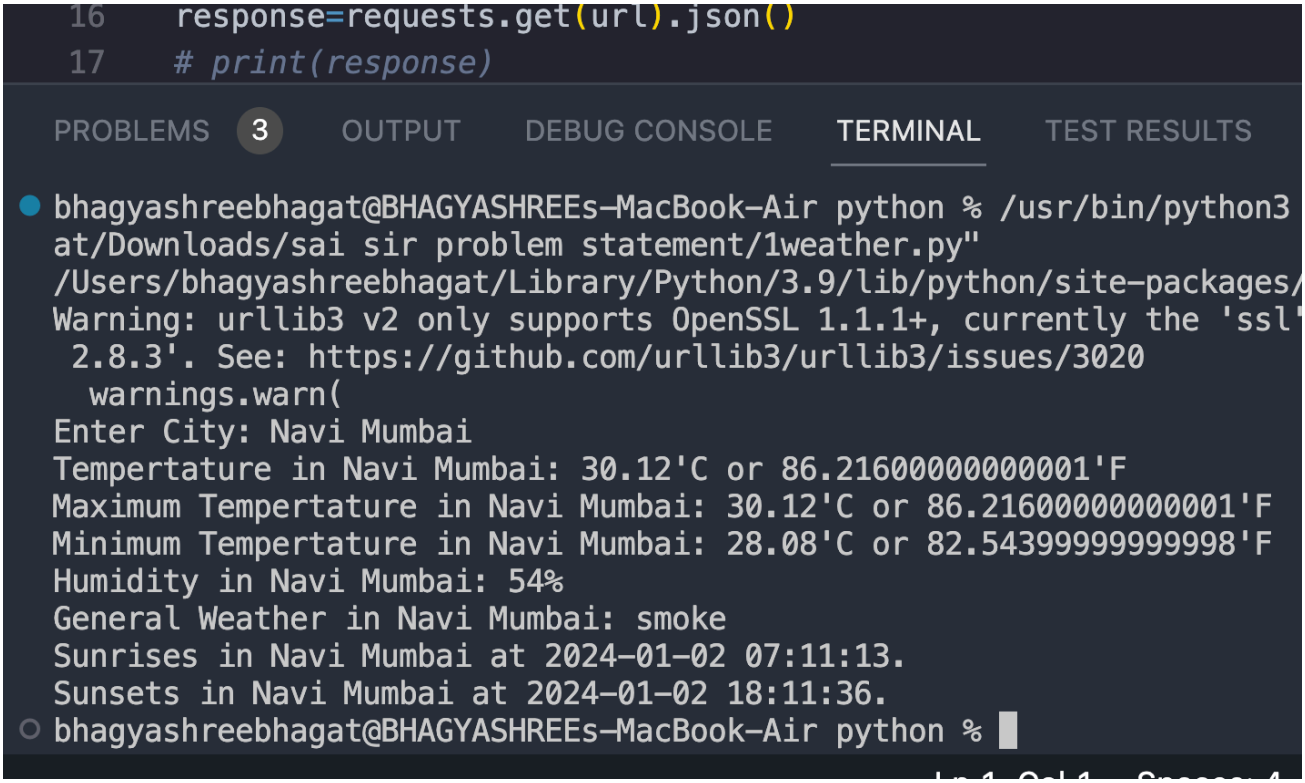
```
import datetime as dt
import requests

baseurl="http://api.openweathermap.org/data/2.5/weather?"

api_key="ed54f9d7486d2edeb84114f5c8b922f4"

city=input('Enter City: ')

def kel_to_cel_fahren(kelvin):
    celsius=kelvin-273
    fahrenheit=celsius*(9/5)+32
    return celsius,fahrenheit

url= baseurl + 'appid=' + api_key + '&q=' + city
response=requests.get(url).json()
# print(response)

temp_kelvin=response['main']['temp']
temp_celsius, temp_fahrenheit=kel_to_cel_fahren(temp_kelvin)
max_temp=response['main']['temp_max']
tempc,tempf=kel_to_cel_fahren(max_temp)
min_temp=response['main']['temp_min']
temc,temf=kel_to_cel_fahren(min_temp)
humidity=response['main']['humidity']
description=response['weather'][0]['description']
sunrise=dt.datetime.utcfromtimestamp(response['sys']['sunrise']
+response['timezone'])
sunset=dt.datetime.utcfromtimestamp(response['sys']['sunset']
+response['timezone'])
```

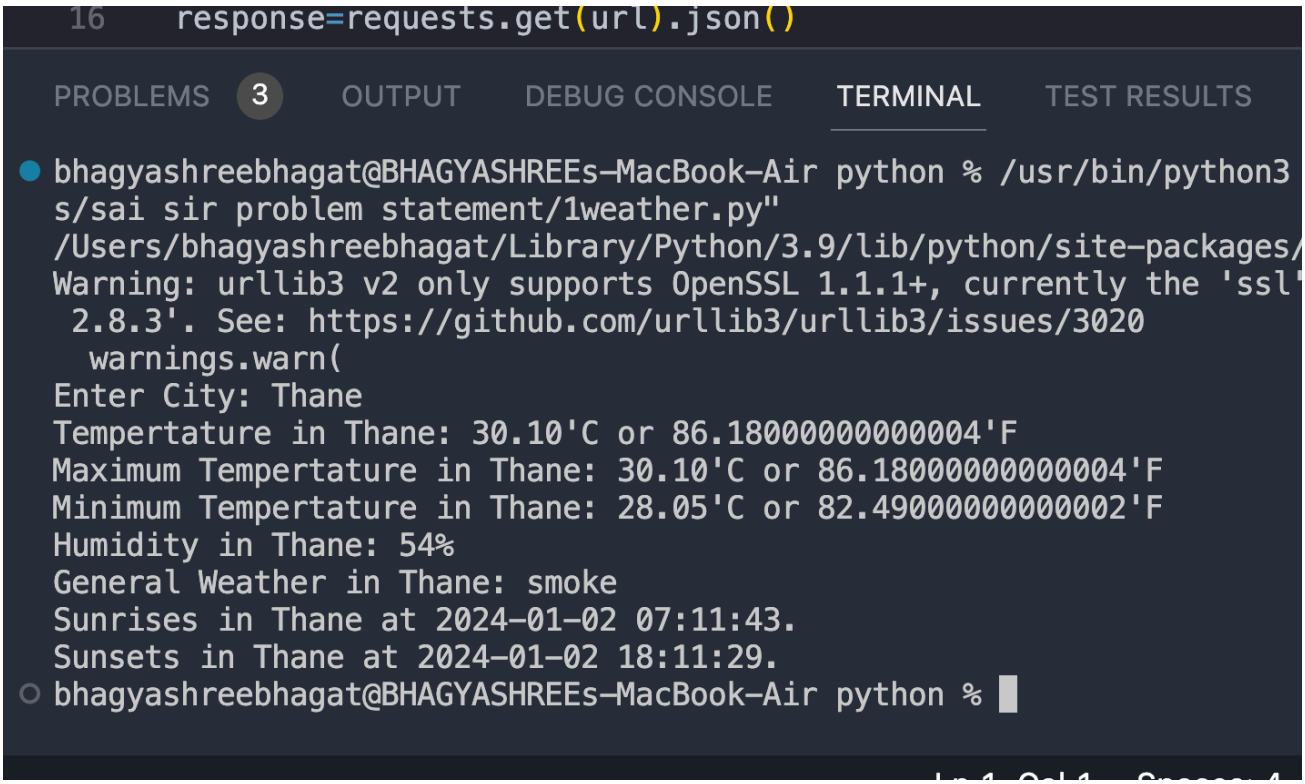
```
print(f"Tempertature in {city}: {temp_celsius:.2f}'C or
{temp_fahrenheit}'F")
print(f"Maximum Tempertature in {city}: {tempc:.2f}'C or
{tempf}'F")
print(f"Minimum Tempertature in {city}: {temc:.2f}'C or {temf}'F")
print(f"Humidity in {city}: {humidity}%")
print(f"General Weather in {city}: {description}")
print(f"Sunrises in {city} at {sunrise}.")
print(f"Sunsets in {city} at {sunset}.")
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

It showcases how to interact with external APIs to gather real-time data, integrating it into a Python program.

Experiment No:7.2

Title:

Use 'API' of Cryptocurrency:

Theory:

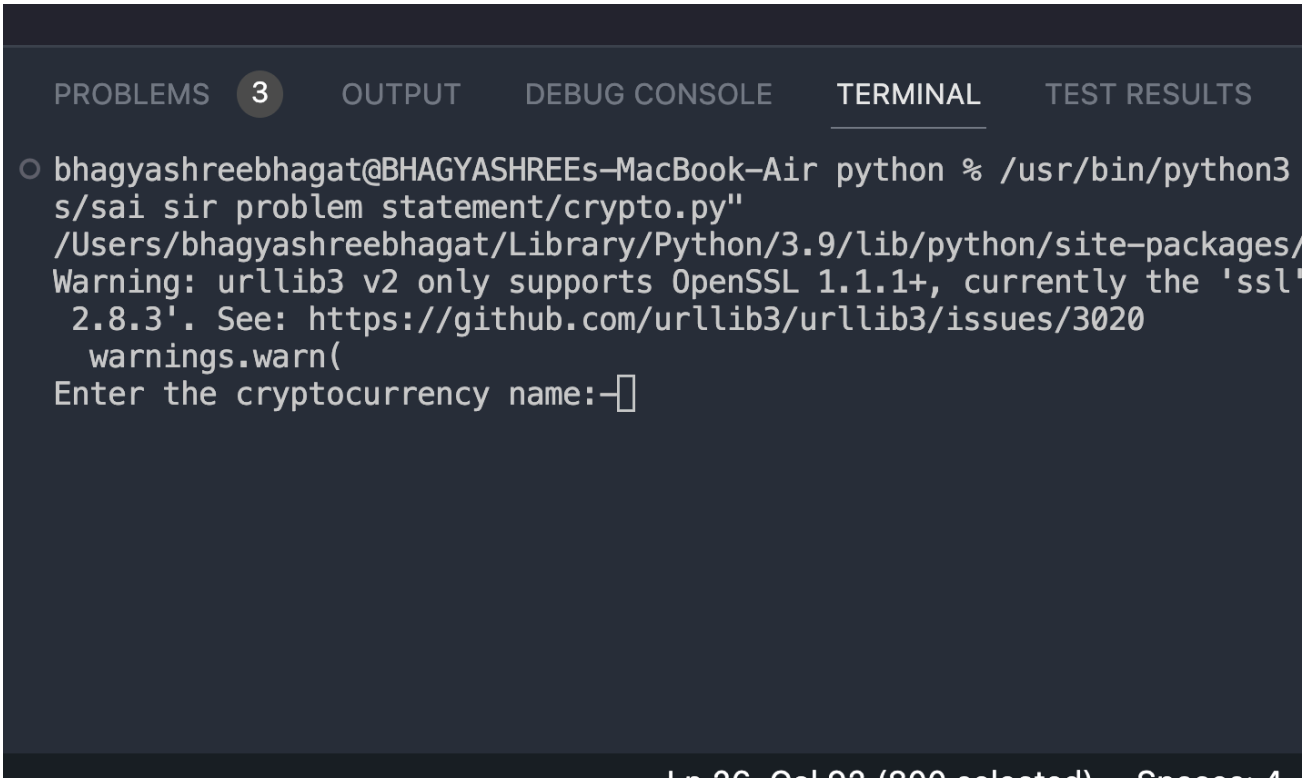
The program uses the API of a cryptocurrency to gather information.

Code:

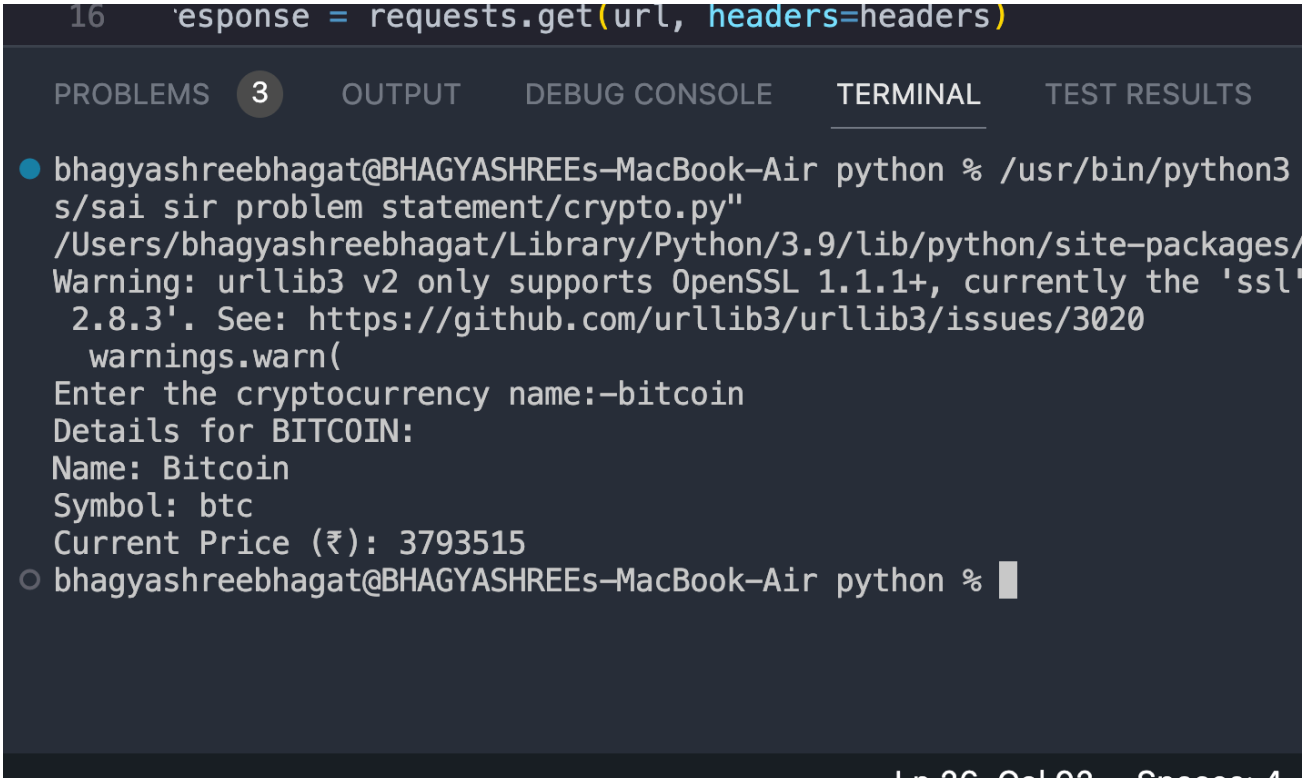
```
import requests

api_key="CG-hw5JqqeSPWRHKa2AdtbLLU3b"
base_url="https://api.coingecko.com/api/v3/coins/"
```

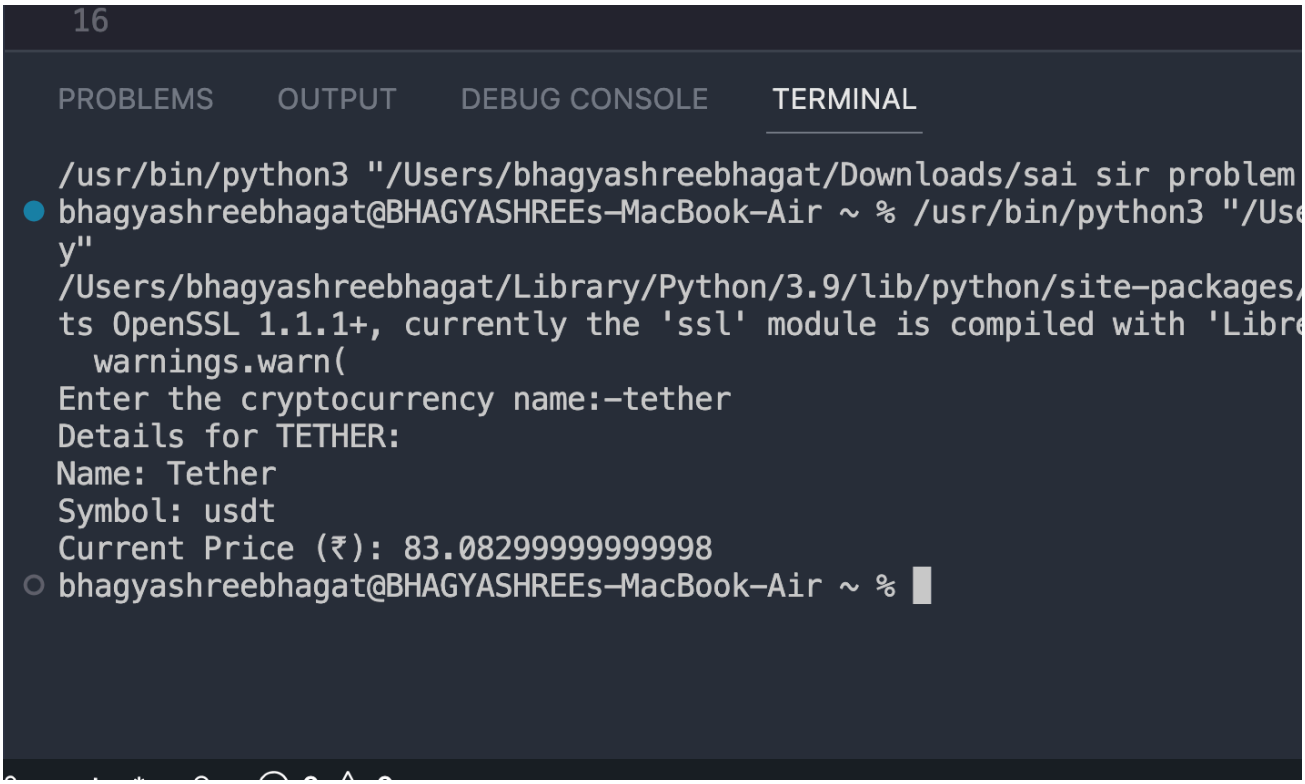
```
# .lower() method is used to convert the input to lowercase to
ensure consistency.
cryptocurrency = input("Enter the cryptocurrency name:-").lower()

url = f"{base_url}{cryptocurrency}"

headers = {
    "Content-Type": "application/json",
    "Authorization": f"Bearer {api_key}"
}

response = requests.get(url, headers=headers)

if response.status_code == 200:
    data = response.json()
    print(f"Details for {cryptocurrency.upper()}:")
    print("Name:", data['name'])
    print("Symbol:", data['symbol'])
    print("Current Price (₹):", data['market_data']
['current_price']['usd']*83)

else:
    print(f"Failed to fetch data for {cryptocurrency}. Status
code:", response.status_code)
```

Output: (screenshot)



Test Case: Any two (screenshot)


16 esponse = requests.get(url, headers=headers)
PROBLEMS 3
OUTPUT
DEBUG CONSOLE
TERMINAL
TEST RESULTS
bhagyashreebhagat@BHAGYASHREEs-MacBook-Air python % /usr/bin/python3
s/sai sir problem statement/crypto.py"
/Users/bhagyashreebhagat/Library/Python/3.9/lib/python/site-packages/
Warning: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl'
2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
warnings.warn(
Enter the cryptocurrency name:-bitcoin
Details for BITCOIN:
Name: Bitcoin
Symbol: btc
Current Price (₹): 3793515
bhagyashreebhagat@BHAGYASHREEs-MacBook-Air python %


16
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
/usr/bin/python3 "/Users/bhagyashreebhagat/Downloads/sai sir problem
bhagyashreebhagat@BHAGYASHREEs-MacBook-Air ~ % /usr/bin/python3 "/Use
y"
/Users/bhagyashreebhagat/Library/Python/3.9/lib/python/site-packages/
ts OpenSSL 1.1.1+, currently the 'ssl' module is compiled with 'Libre
warnings.warn(
Enter the cryptocurrency name:-tether
Details for TETHER:
Name: Tether
Symbol: usdt
Current Price (₹): 83.08299999999998
bhagyashreebhagat@BHAGYASHREEs-MacBook-Air ~ %

Conclusion:

It demonstrates how to retrieve data from a cryptocurrency API, providing insights into integrating external data sources into a Python program.