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

PYTHON PROGRAMMING LAB

Prepared by:

Name of Student: **Jeevan Naidu**

Roll No: **15**

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: Jeevan Naidu

Roll Number: 15

Experiment No: 01.01

Title: Write a program to compute Simple Interest.

Theory:

$$\text{Simple Interest (SI)} = \frac{\text{Principal} \times \text{Rate} \times \text{Time}}{100}.$$

- Principal (P):The initial amount of money.

- Rate of Interest (R): The percentage of interest charged or earned.

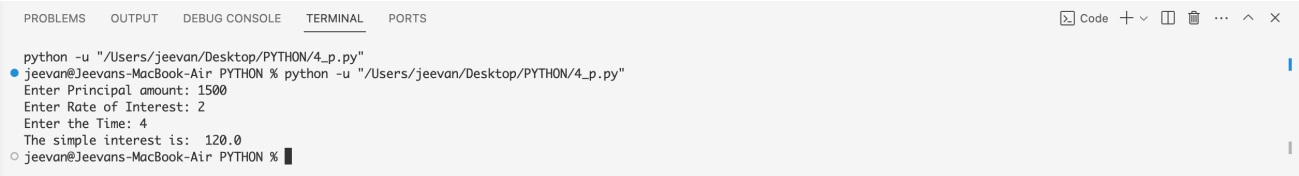
- Time (T): The duration for which the interest is calculated.

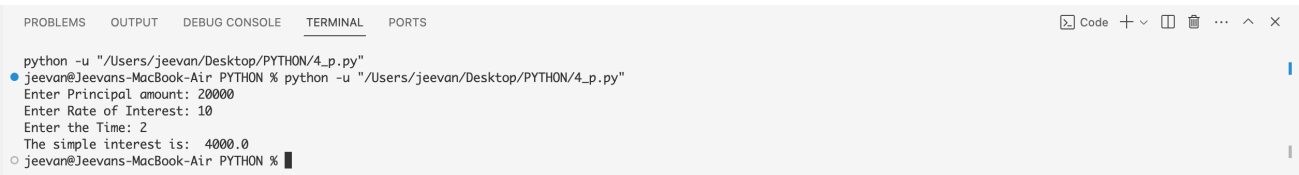
The program takes user input for these three values, calculates the simple interest, and then displays the result.

Code:

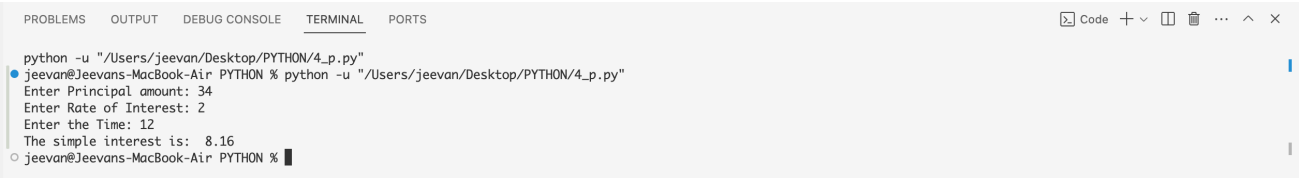
```
#WAP to compute simple interest
p=float(input("Enter Principal amount: "))
r=float(input("Enter Rate of Interest: "))
t=float(input("Enter the Time: "))
si=((p*r*t)/100)
print("The simple interest is: ",si)
```

Output: (screenshot)





Test Case: Any two (screenshot)



Conclusion:

This Python program provides a simple yet effective way to calculate the simple interest for a financial transaction. It demonstrates user input handling, basic arithmetic operations, and the use of variables in Python. The formula used for simple interest is a fundamental concept in financial mathematics, making this program a practical example for beginners to understand basic mathematical computations in Python.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 01.02

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

Theory:

In Python, arithmetic operations are fundamental mathematical operations that can be performed on numeric data types. These operations include addition, subtraction, multiplication, division, modulus, exponentiation, and floor division.

Relational operations, on the other hand, are used to compare two values and return a Boolean result indicating the relationship between them. Common relational operators include greater than (`>`), less than (`<`), equal to (`==`), greater than or equal to (`>=`), less than or equal to (`<=`), and not equal to (`!=`).

1. Accepting User Input:

   - The program starts by accepting two operands (`num1` and `num2`) and an operation choice (`num_op`) from the user.

2. Performing Arithmetic Operations:

   - Based on the operation choice, the program performs the corresponding arithmetic operation (addition, subtraction, multiplication, division, modulus, exponentiation, or floor division).

3. Performing Relational Operations:

   - If the operation choice is a relational operation (greater than, less than, or equal to), the program compares `num1` and `num2` and prints the result.

4. Handling Special Cases:

   - Special checks are included to avoid division by zero in the cases of regular division (`/`) and floor division (`//`).

5. Displaying the Result:

   - The program displays the result of the chosen operation or a message indicating an invalid operation choice.

Code:

```
# WAP to perform arithmetic and relational operations accepting two operands.
```

```
# Accepting two operands and operation choice
from the user
num1 = float(input("Enter the first number:
"))
num2 = float(input("Enter the second number:
"))
num_op = int(input("Enter the operation\n(1)
Addition\n(2) Subtraction\n(3)
Multiplication\n(4) Division\n(5)
Modulus\n(6) Exponent\n(7) Floor\n(8) Greater
Than\n(9) Less Than\n(10) Equal To\n"))

# Performing the selected operation and
displaying the result
if num_op == 1:
    print(num1, " + ", num2, " = ", num1 +
num2)
elif num_op == 2:
    print(num1, " - ", num2, " = ", num1 -
num2)
elif num_op == 3:
    print(num1, " * ", num2, " = ", num1 *
num2)
elif num_op == 4:
    if num2 != 0:  # Avoid division by zero
```

```
        print(num1, " / ", num2, " = ",
num1 / num2)
    else:
        print("Cannot divide by zero.")
elif num_op == 5:
    print(num1, " % ", num2, " = ", num1 %
num2)
elif num_op == 6:
    print(num1, " ^ ", num2, " = ", num1 **
num2)
elif num_op == 7:
    if num2 != 0:  # Avoid floor division by
zero
        print(num1, " // ", num2, " = ", num1
// num2)
    else:
        print("Cannot perform floor division
by zero.")
elif num_op == 8:
    print(num1, " > ", num2, " is ", num1 >
num2)
elif num_op == 9:
    print(num1, " < ", num2, " is ", num1 <
num2)
elif num_op == 10:
    print(num1, " == ", num2, " is ", num1 ==
num2)
```

```
else:
    print("Invalid operation choice. Please enter a number from 1 to 10 for a valid operation.")
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

This Python program demonstrates the implementation of both arithmetic and relational operations. Users can input two operands and choose from a menu of operations, including common arithmetic calculations and relational comparisons. The program is designed to handle various scenarios, such as avoiding division by zero, and provides a clear output to the user based on their input.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 01.03

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

Theory:

The provided Python program is designed to determine whether a given input number is even or odd. In Python, the modulus operator (`%`) is used to find the remainder of the division of one number by another. If the remainder is zero when a number is divided by 2, then the number is even; otherwise, it is odd.

1. Accepting User Input:

- The program starts by prompting the user to input a number (`num`).

2. Checking for Even or Odd:

- Using an `if-else` statement, the program checks whether the entered number is divisible by 2 without a remainder (`num % 2 == 0`). If the condition is true, it prints that the number is even; otherwise, it prints that the number is odd.

3. Displaying the Result:

- The program outputs a clear message indicating whether the input number is even or odd.

Code:

```
#WAP to find input number is even or odd
num=int(input("Enter a number: "))
if num%2==0:
    print(num,"is an even number")
else:
    print(num,"is odd number")
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

This Python program provides a simple yet effective way to determine whether a given number is even or odd. It leverages the modulus operator to check for divisibility by 2 and uses an `if-else` statement for decision-making. The program concludes by printing a message that communicates the parity of the entered number to the user. This type of program is commonly used in introductory programming courses to illustrate basic control flow and conditionals.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 01.04

Title: Write a program to print first n natural number & their sum.

Theory:

The provided Python program is designed to print the first n natural numbers and calculate their sum. It demonstrates the use of a function, a loop, and variables for this purpose.

1. Function Definition:

   - The program defines a function `print_natural_numbers_and_sum` that takes an argument `n`.

2. Initialization:

   - Inside the function, it initializes a variable `sum_of_numbers` to store the sum of natural numbers.

3. Printing and Sum Calculation:

   - Using a `for` loop, it iterates from 1 to `n` (inclusive) and prints each natural number.
   - It calculates the sum of natural numbers during the iteration.

4. Displaying Results:

   - The function then prints the sum of the first `n` natural numbers.

5. User Input and Function Call:

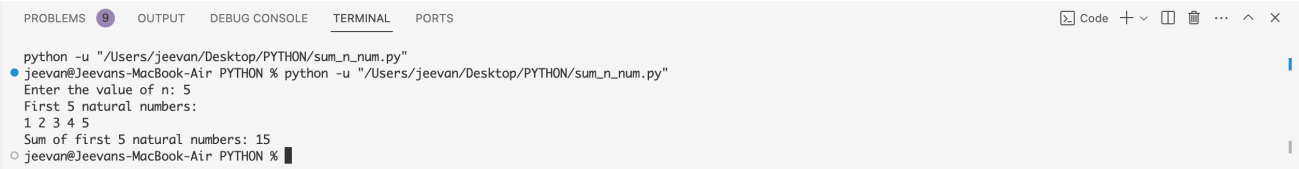
   - The user is prompted to input the value of `n`.
   - The function is called with the user-provided input.

Code:

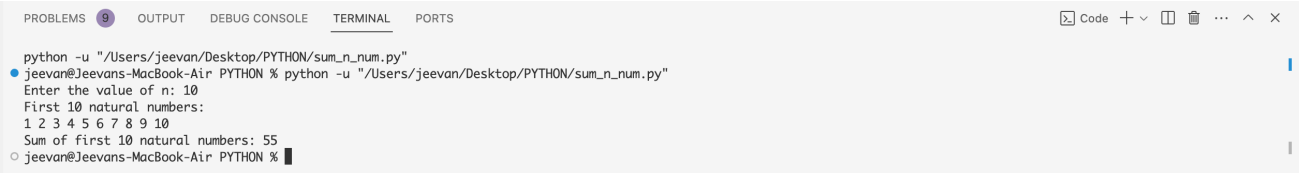
```
# Program to print first n natural numbers and their sum
```

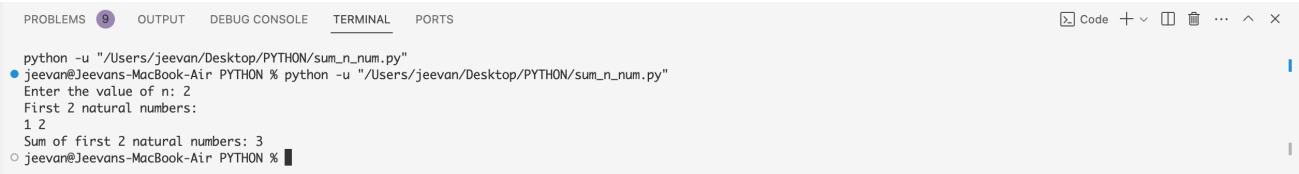
```python
# Function to print first n natural numbers and calculate their sum
def print_natural_numbers_and_sum(n):
    # Initialize variables
    sum_of_numbers = 0

    # Print first n natural numbers and calculate their sum
    print("First", n, "natural numbers:")
    for i in range(1, n + 1):
        print(i, end=" ")

        # Calculate sum
        sum_of_numbers += i

    # Print the sum of natural numbers
    print("\nSum of first", n, "natural numbers:", sum_of_numbers)

# Input: Number of natural numbers to print
n = int(input("Enter the value of n: "))

# Call the function
print_natural_numbers_and_sum(n)
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

This Python program provides a reusable function to print the first n natural numbers and calculate their sum. It leverages a loop to iterate through the natural numbers and a variable (`sum_of_numbers`) to accumulate their sum. The program demonstrates modular code organization through function usage and is an illustrative example for beginners learning Python programming.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 01.05

Title: Write a program to determine whether the character entered is a Vowel or not

Theory:

1. Function Definition:

- The program defines a function `is_vowel` that checks whether a given character is a vowel. It uses a string of vowels and checks if the character is present in that string.

2. User Input:

- The user is prompted to enter a character (`user_char`).

3. Input Validation:

- The program checks if the entered input is a single alphabetic character.

4. Vowel Check:

- If the entered character is a single alphabet, the program calls the `is_vowel` function to determine whether it is a vowel or not.

5. Displaying Results:

- The program prints the result based on whether the entered character is a vowel or not.

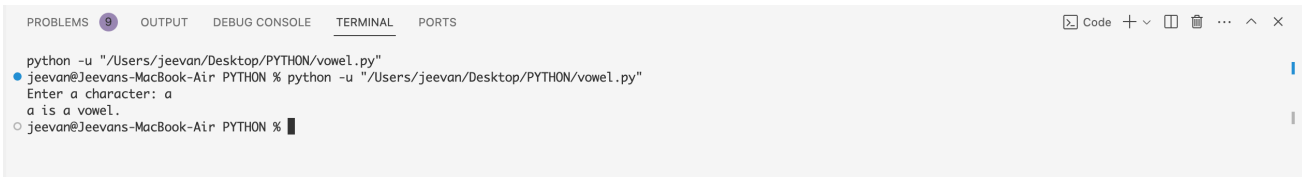
Code:

```
# Program to determine whether a character is a vowel or not

# Function to check if a character is a vowel
def is_vowel(char):
```

```
    vowels = "AEIOUaeiou"
    return char in vowels

# Input: User enters a character
user_char = input("Enter a character: ")

# Check if the entered character is a vowel
if len(user_char) == 1 and
user_char.isalpha():
    if is_vowel(user_char):
        print(user_char, "is a vowel.")
    else:
        print(user_char, "is not a vowel.")
else:
    print("Please enter a single alphabetic
character.")
```

Output: (screenshot)

Test Case: Any two (screenshot)



Conclusion:

This Python program provides a modular function (`is_vowel`) to determine whether a character is a vowel. It incorporates input validation to ensure a single alphabetic character is entered by the user. The program serves as a simple yet effective example of conditional statements and function usage in Python. It is suitable for educational purposes and beginners learning Python programming.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 01.06

Title: Write a program to find whether given number is an Armstrong Number.

Theory:

1. Function Definition:

- The program defines a function `is_armstrong_number` that checks whether a given number is an Armstrong number. It calculates the sum of cubes of individual digits raised to the power of the total number of digits.

2. User Input:

- The user is prompted to enter a number (`user_num`).

3. Input Validation:

- The program checks if the entered input is a non-negative integer.

4. Armstrong Check:

- If the entered number is a non-negative integer, the program calls the `is_armstrong_number` function to determine whether it is an Armstrong number or not.

5. Displaying Results:

- The program prints the result based on whether the entered number is an Armstrong number or not.

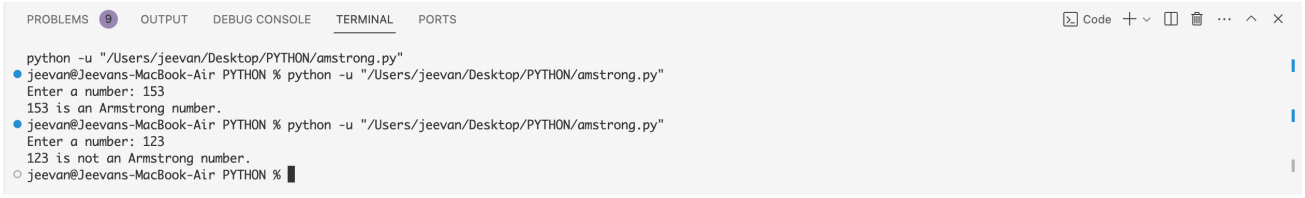
Code:

```
# Program to check if a number is an
Armstrong number

# Function to check if a number is an
Armstrong number
```

```python
def is_armstrong_number(num):
    order = len(str(num))
    sum_of_cubes = sum(int(digit) ** order
for digit in str(num))
    return num == sum_of_cubes

# Input: User enters a number
user_num = int(input("Enter a number: "))

# Check if the entered number is an Armstrong
number
if user_num >= 0:
    if is_armstrong_number(user_num):
        print(user_num, "is an Armstrong
number.")
    else:
        print(user_num, "is not an Armstrong
number.")
else:
    print("Please enter a non-negative
integer.")
```

Output: (screenshot)

Test Case: Any two (screenshot)



Conclusion:

This Python program checks whether a given number is an Armstrong number using a function that encapsulates the logic for the Armstrong check. It incorporates input validation to ensure the user enters a non-negative integer. The program demonstrates a practical application of mathematical concepts in programming and serves as an example for educational purposes.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 01.07

Title: Write a program using for loop to calculate factorial of a No.

Theory:

1. Function Definition:

- The program defines a function `calculate_factorial` to compute the factorial of a given number ((num)) using a `for` loop.

2. User Input:

- The user is prompted to enter a number ((user_num)).

3. Input Validation:

- The program checks if the entered input is a non-negative integer.

4. Factorial Calculation:

- If the entered number is a non-negative integer, the program calls the `calculate_factorial` function to calculate the factorial.

5. Displaying Results:

- The program prints the calculated factorial.

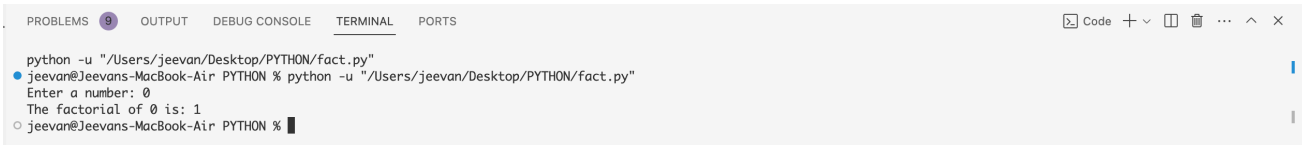
Code:

```
# Program to calculate the factorial of a number using a for loop
```

```python
# Function to calculate factorial
def calculate_factorial(num):
    factorial = 1
    for i in range(1, num + 1):
        factorial *= i
    return factorial

# Input: User enters a number
user_num = int(input("Enter a number: "))

# Check if the entered number is non-negative
if user_num >= 0:
    result = calculate_factorial(user_num)
    print(f"The factorial of {user_num} is: {result}")
else:
    print("Please enter a non-negative integer.")
```

Output: (screenshot)

Test Case: Any two (screenshot)



Conclusion:

This Python program demonstrates a common mathematical operation—calculating the factorial of a number. It uses a `for` loop to iteratively compute the product of positive integers up to the given number. The program includes input validation to ensure the user enters a non-negative integer, enhancing its robustness. This example showcases the use of loops for repetitive tasks and is educational for those learning Python programming.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 01.08.01

Title: i)

```
*
* *
* * *
* * * *
* * * * *
```

Theory:

1. User Input:
   - The program prompts the user to enter the number of rows (`num_rows`) for the right-angled triangle.

2. Outer Loop (`for i in range(1, num_rows + 1):`):
   - This loop iterates over each row from 1 to the specified number of rows.

3. Inner Loop (`for j in range(i):`):
   - For each row, an inner loop prints '*' characters. The loop runs `i` times in each row.

4. Printing '*' Characters:
   - The `print("*", end=" ")` statement prints '*' characters horizontally on the same line.

5. Newline Character (`print()`):
   - After printing '*' characters for each row, a newline character is added (`print()`) to move to the next line.

Code:

```python
# Program to print a right-angled triangle pattern

# Input: Number of rows
num_rows = int(input("Enter the number of rows: "))

# Loop to iterate over each row
for i in range(1, num_rows + 1):
    # Inner loop to print '*' in each row
    for j in range(i):
        print("*", end=" ")
    # Move to the next line after printing each row
    print()
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

This Python program illustrates the use of nested loops to create a right-angled triangle pattern of '*' characters. The outer loop controls the number of rows, and the inner loop prints the '*' characters in each row. This pattern is a common example in programming for understanding loop structures and pattern printing. It encourages user interaction by allowing them to specify the number of rows, promoting flexibility and customization in the pattern.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 01.08.02

Title: ii)

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

Theory:

- Input:

  - The user is prompted to enter the number of rows for the equilateral triangle pattern.

- Outer Loop:

  - The outer loop (`for i in range(1, num_rows + 1)`) iterates through each row of the triangle.

- Inner Loop:

  - The inner loop (`for j in range(1, i + 1)`) prints the value of `i` in each column of the current row.

- Printing:

  - The `print(i, end=" ")` statement prints the value of `i` in each column, and the `end=" "` argument ensures that the numbers are printed horizontally.

- Newline Character:

  - After printing each row, a `print()` statement without any arguments adds a newline character to move to the next line.

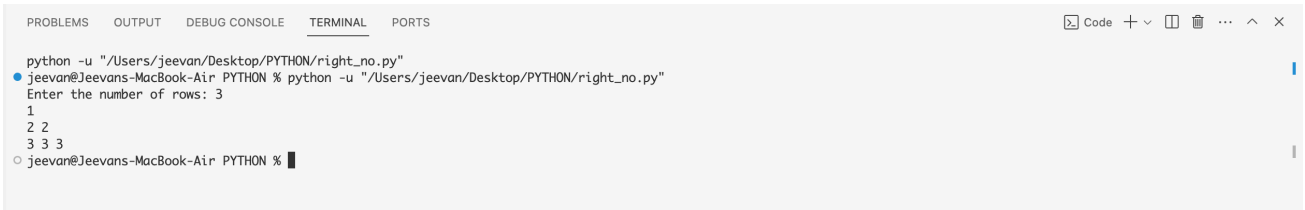
Code:

```
# Program to print an equilateral triangle pattern with numbers
```

```python
# Input: Number of rows
num_rows = int(input("Enter the number of rows: "))

# Loop to iterate over each row
for i in range(1, num_rows + 1):
    # Inner loop to print numbers in each row
    for j in range(1, i + 1):
        print(i, end=" ")
    # Move to the next line after printing each row
    print()
```

Output: (screenshot)

Test Case: Any two (screenshot)



Conclusion:

This program effectively demonstrates the use of nested loops to generate an equilateral triangle pattern with numbers based on the user-specified number of rows. The resulting pattern is displayed on the console, providing a clear visual representation of the equilateral triangle with repeated numbers.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 01.08.03

Title: iii)

```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

Theory:

- Outer Loop (`for i in range(1, 5):`):

  - This loop iterates through each row of the pyramid, and `i` represents the current row number.

- First Inner Loop (`for k in range(5, i, -1):`):

  - This loop is responsible for printing the leading spaces before the asterisks in each row. It starts with the maximum number of spaces (5) and decrements as `i` increases.

- Second Inner Loop (`for j in range(0, i):`):

  - This loop prints the ascending sequence of asterisks in each row.

- Third Inner Loop (`for l in range(i-1, 0, -1):`):

  - This loop prints the descending sequence of asterisks after the ascending sequence in each row.

- Printing (`print('  ', end='')` and `print('* ', end='')`):

  - The `print('  ', end='')` statement prints two spaces without moving to the next line, creating the space between the leading spaces and the asterisks.

  - The `print('* ', end='')` statement prints an asterisk followed by a space without moving to the next line.

- Newline Character (`print()`):

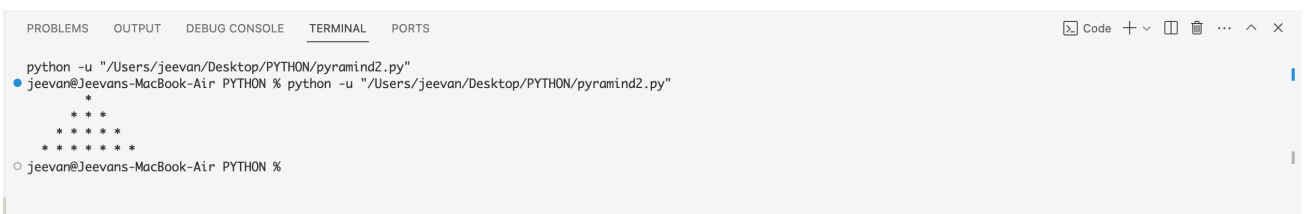
  - After printing the leading spaces and asterisks for each row, a `print()` statement without any arguments adds a newline character to move to the next line for the next row.

Code:

```
for i in range(1,5):
    for k in range(5,i,-1):
        print('  ',end='')
    for j in range(0,i):
        print('* ',end='')
    for l in range(i-1,0,-1):
        print('* ',end='')
    print()
```

Output: (screenshot)

Test Case: Any two (screenshot)



Conclusion:

This code successfully generates a pyramid pattern using asterisks, with each row having an ascending sequence of asterisks followed by a descending sequence. The number of rows in the pyramid is determined by the range of the outer loop. The pattern is displayed on the console, creating a visually appealing pyramid structure.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 02.01

Title: Write a program that define the list of defines the list of define countries that are in BRICS.

Theory:

This Python program is designed to check whether a given country is a member of BRICS (Brazil, Russia, India, China, South Africa) or not. It uses a list (`brics`) to store the names of BRICS member countries. The user is prompted to input the name of a country, and the program checks if the entered country is present in the BRICS list.

- List Definition (`brics=['BRAZIL','RUSSIA','INDIA','CHINA','SOUTH AFRICA']`):

  - This line defines a list named `brics` containing the names of BRICS member countries. The names are stored in uppercase to make the comparison case-insensitive.

- User Input (`a=input("Enter the name of country: ").upper()`):

  - The user is prompted to enter the name of a country, and the input is converted to uppercase using the `upper()` method for consistent comparison.

- Loop to Check Membership (`for i in range(5):`):

  - This loop iterates through each element (country) in the `brics` list.

  - The loop compares the user-input country with each element in the list.

  - If a match is found, it sets `k` to 'found', prints a message indicating membership, and exits the loop using `break`.

- Checking and Output (`if k!='found':`):

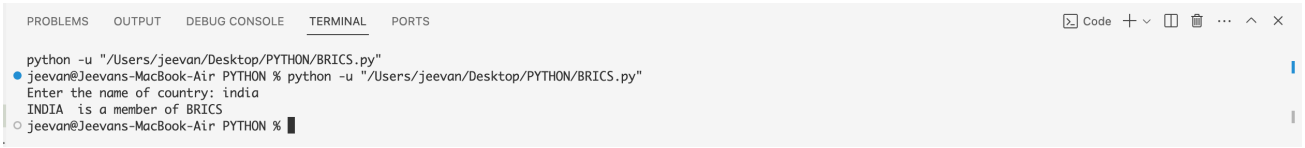
  - If `k` is not set to 'found' after the loop, it means that the entered country is not in the BRICS list, and the program prints a message indicating non-membership.

Code:

```
'''WAP that defines a list of countries that are member of BRICS
check wether the country is member of brics or not'''
brics=['BRAZIL','RUSSIA','INDIA','CHINA','SOUTH AFRICA']

a=input("Enter the name of country: ").upper()

for i in range(5):
    k='not found'
    if a==brics[i]:
        k='found'
        print(a, " is a member of BRICS")
        break

if k!='found':
    print(a, " is not a member of BRICS")
```

Output: (screenshot)

Test Case: Any two (screenshot)



Conclusion:

This program effectively checks whether a user-input country is a member of BRICS and provides the corresponding output. It demonstrates the use of lists, user input, and loops for comparison. The uppercase conversion ensures case-insensitive matching. The program delivers a clear message about BRICS membership based on the user's input.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 02.02

Title: Write a program to traverse a list in reverse order.

1.By using Reverse method.
2.By using slicing

Theory:

1. Using Reverse Method:

- The `reversed()` function returns a reversed iterator of the given iterable (in this case, the list).
- The `list()` function is used to convert the iterator back to a list.
- The reversed list is printed.

2. Using Slicing:

- Slicing with `[::-1]` creates a new list with elements in reverse order.
- The reversed list is printed.

Code:

```
# Method 1: Using Reverse Method
def reverse_method(my_list):
    reversed_list = list(reversed(my_list))
    print("Using Reverse Method:",
reversed_list)

# Method 2: Using Slicing
def slicing(my_list):
    reversed_list = my_list[::-1]
```

```
    print("Using Slicing:", reversed_list)

# Input: List of numbers
numbers_list = [1, 2, 3, 4, 5]

# Call the functions to traverse in reverse order
reverse_method(numbers_list)
slicing(numbers_list)
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

The provided Python program demonstrates two methods for traversing a list in reverse order:

- Method 1: Using Reverse Method: Utilizes the `reversed()` function to create a reversed iterator, converted back to a list.

- Method 2: Using Slicing: Uses slicing with `[::-1]` to generate a new list with elements in reverse order.

Both methods achieve the same result, offering flexibility based on personal preference or specific use cases. The program is concise, efficient, and provides a clear illustration of the two approaches.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 02.03

Title: Write a program that scans the email address and forms a tuple of username and domain.

Theory:

1. Function `extract_email_parts`:

- The function takes an email address as input.

- It uses the `split('@')` method to split the email address into parts using '@' as the separator.

- Checks if the email has the correct format by verifying that it contains only one '@'.

- Returns a tuple of username and domain if the format is correct; otherwise, returns `(None, None)`.

2. User Input and Result Display:

- The user is prompted to enter an email address using the `input` function.

- The `extract_email_parts` function is called with the user-provided email address to obtain the username and domain.

- The program checks if both username and domain are not `None` before displaying them.

- If the email format is invalid, it displays an error message.

Code:

```
def extract_email_parts(email):
    # Split the email address into username and domain using '@' as the separator
    parts = email.split('@')
```

```
    # Check if the email has the correct
format (contains only one '@')
    if len(parts) == 2:
        username, domain = parts
        return username, domain
    else:
        return None, None

# Input: User enters an email address
user_email = input("Enter your email address:
")

# Extract username and domain from the email
address
username, domain =
extract_email_parts(user_email)

# Display the result
if username and domain:
    print("Username:", username)
    print("Domain:", domain)
else:
    print("Invalid email address format.
Please enter a valid email.")
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

The Python program demonstrates a modular approach to extract and display the username and domain parts of an email address. It includes error handling to manage cases where the email format is incorrect, providing a user-friendly experience.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 02.07

Title: 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.**

Theory:

Lists in Python are versatile data structures that allow you to store and manipulate collections of items. The provided program demonstrates various list operations:

1. Printing the Complete List: The program starts by printing the complete list.

2. Accessing the 4th Element: It then prints the 4th element of the list using list indexing.

3. Slicing the List: The program prints a sublist by slicing the list from the 0th to the 4th index.

4. Negative Indexing and Slicing: The program showcases the use of negative indexing to slice the list from the -7th to the 3rd element.

5. Appending an Element: An element (110) is appended to the list using the `append` method.

6. Sorting the List: The elements of the list are sorted in ascending order using the `sort` method.

7. Popping an Element: The last element is popped from the list using the `pop` method.

8. Removing a Specified Element: The element 60 is removed from the list using the `remove` method.

9. Inserting an Element at a Specified Index: The program inserts the element 25 at index 2 using the `insert` method.

10. Counting Occurrences: The program counts the occurrences of the element 30 using the `count` method.

11. Extending the List: The list is extended with elements [120, 130, 140] using the `extend` method.

12. Reversing the List: Finally, the list is reversed using the `reverse` method.

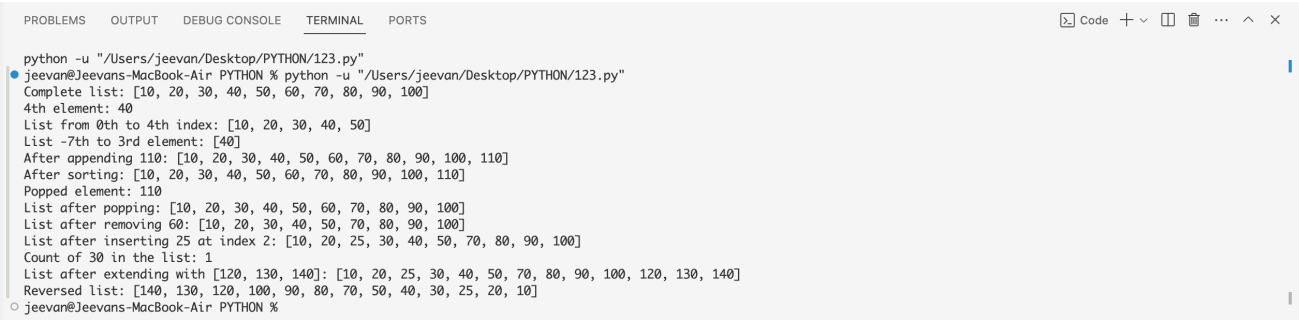
Code:

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

# Print complete list
print("Complete list:", a)

# Print 4th element of the list
print("4th element:", a[3])

# Print list from 0th to 4th index
print("List from 0th to 4th index:", a[0:5])

# Print list -7th to 3rd element
```

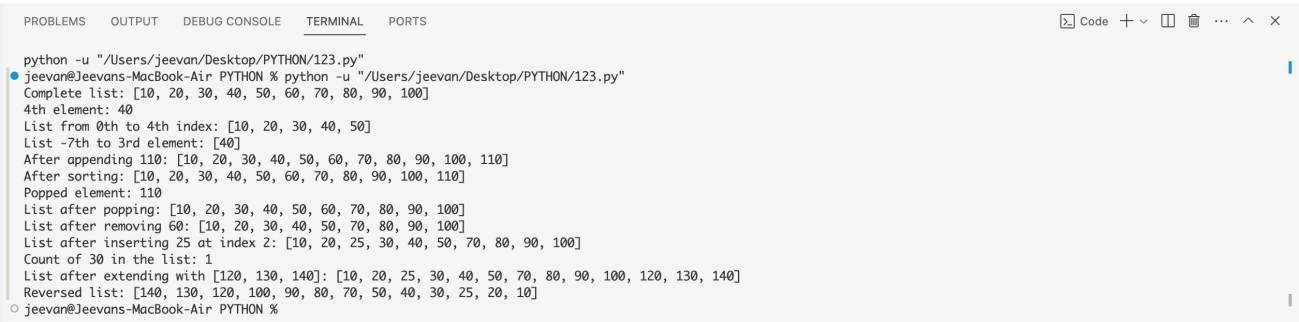
```
print("List -7th to 3rd element:", a[-7:4])

# Appending an element to the list
a.append(110)
print("After appending 110:", a)

# Sorting the elements of the list
a.sort()
print("After sorting:", a)

# Popping an element
popped_element = a.pop()
print("Popped element:", popped_element)
print("List after popping:", a)

# Removing a specified element
a.remove(60)
print("List after removing 60:", a)

# Entering an element at a specified index
a.insert(2, 25)
print("List after inserting 25 at index 2:",
a)

# Counting the occurrence of a specified
element
count_30 = a.count(30)
```

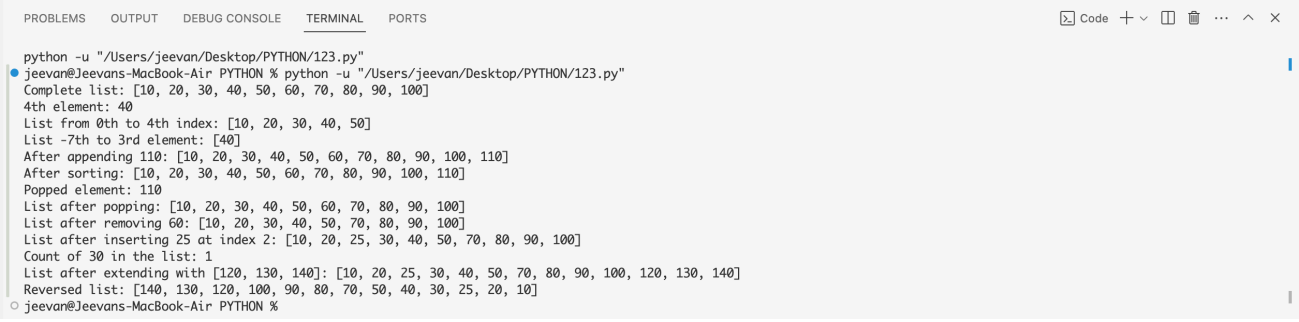
```
print("Count of 30 in the list:", count_30)

# Extending the list
a.extend([120, 130, 140])
print("List after extending with [120, 130, 140]:", a)

# Reversing the list
a.reverse()
print("Reversed list:", a)
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

This program covers a range of operations that can be performed on lists in Python. It provides insights into accessing elements, slicing, appending, sorting, popping, removing, inserting, counting, extending, and reversing a list. These operations are fundamental in handling and manipulating lists efficiently.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 02.05

Title: Write a program to compare two dictionaries in Python?

(By using == operator)

Theory:

In this Python program, we have defined a function `compare_dictionaries` that takes two dictionaries as arguments and compares them using the `==` operator. The `==` operator checks if the key-value pairs in both dictionaries are the same.

We then create two example dictionaries, `dict_a` and `dict_b`, with the same key-value pairs. The `compare_dictionaries` function is called with these dictionaries, and the result is stored in the variable `result`.

Finally, the program prints whether the dictionaries are equal or not based on the value of the `result` variable.

Code:

```
# Function to compare two dictionaries
def compare_dictionaries(dict1, dict2):
    return dict1 == dict2

# Example dictionaries
dict_a = {'a': 1, 'b': 2, 'c': 3}
dict_b = {'a': 1, 'b': 2, 'c': 3}

# Compare dictionaries using the function
result = compare_dictionaries(dict_a, dict_b)
```

```
# Display the result
if result:
    print("The dictionaries are equal.")
else:
    print("The dictionaries are not equal.")
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

This program demonstrates a simple way to compare two dictionaries in Python using the `==` operator. If the dictionaries have the same key-value pairs, the program prints "The dictionaries are equal," otherwise, it prints "The dictionaries are not equal."

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 02.06

Title: Write a program that creates dictionary of cube of odd numbers in the range.

Theory:

1. Function Definition (`create_odd_cubes_dictionary`):

- The function takes two parameters, `start` and `end`, representing the range of numbers.
- It initializes an empty dictionary called `odd_cubes_dict`.
- It iterates through the range from `start` to `end + 1` and checks if each number is odd.
- For each odd number, it calculates the cube and adds an entry to the dictionary with the odd number as the key and its cube as the value.
- The final dictionary is returned.

2. User Input:

- The user is prompted to enter the start and end values of the range.

3. Function Call and Output Display:

- The `create_odd_cubes_dictionary` function is called with the user-provided range.
- The resulting dictionary is stored in the variable `result_dict`.
- The program displays the dictionary as the output.

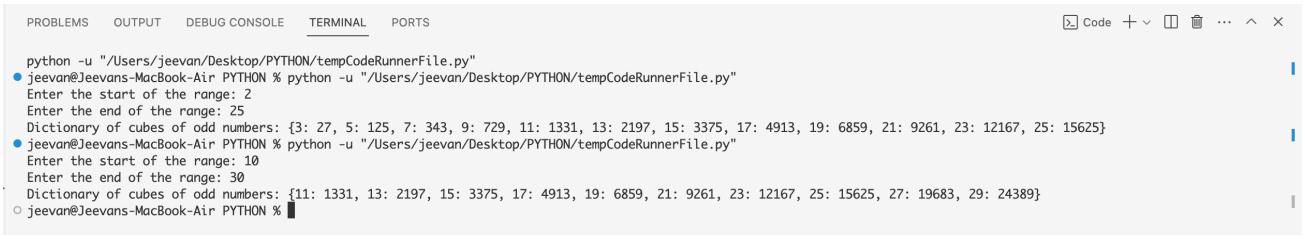
Code:

```
# Function to create a dictionary of cubes of odd numbers in a given range
def create_odd_cubes_dictionary(start, end):
```

```
    odd_cubes_dict = {}

    for num in range(start, end + 1):
        if num % 2 != 0:  # Check if the
number is odd
            odd_cubes_dict[num] = num ** 3  #
Cube of the odd number

    return odd_cubes_dict

# Input: User enters the range
start_range = int(input("Enter the start of
the range: "))
end_range = int(input("Enter the end of the
range: "))

# Create the dictionary of cubes of odd
numbers in the given range
result_dict =
create_odd_cubes_dictionary(start_range,
end_range)

# Display the result
print("Dictionary of cubes of odd numbers:",
result_dict)
```

Output: (screenshot)

Test Case: Any two (screenshot)



Conclusion:

This program showcases the creation of a dictionary that captures the cubes of odd numbers within a specified numerical range. The user's input defines the range, and the program efficiently generates and displays the corresponding dictionary.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 02.04

Title: 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]

Theory:

1. Function Definition (`create_tuples_with_cubes`):

- The function takes a single parameter, `input_list`, which is the list of numbers.

- It uses a list comprehension to create a list of tuples, where each tuple consists of a number from the input list and its cube.

- The resulting list of tuples is returned.

2. Given Input List (`c`):

- The provided input list contains the numbers `[2, 3, 4, 5, 6, 7, 8, 9]`.

3. Function Call and Output Display:

- The `create_tuples_with_cubes` function is called with the input list `c`.

- The resulting list of tuples is stored in the variable `result_list`.

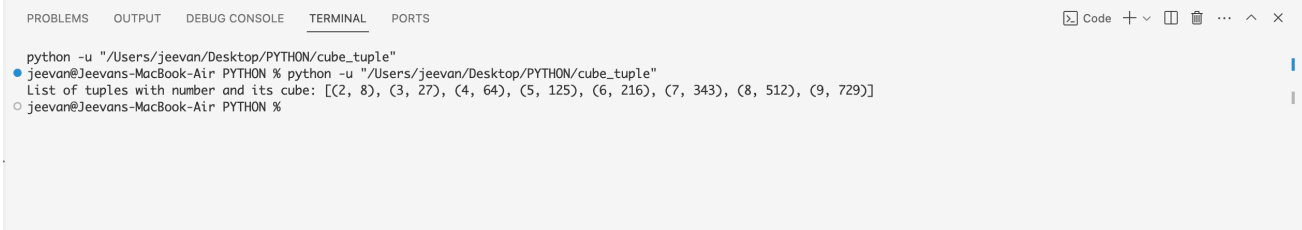
- The program displays the list of tuples as the output.

Code:

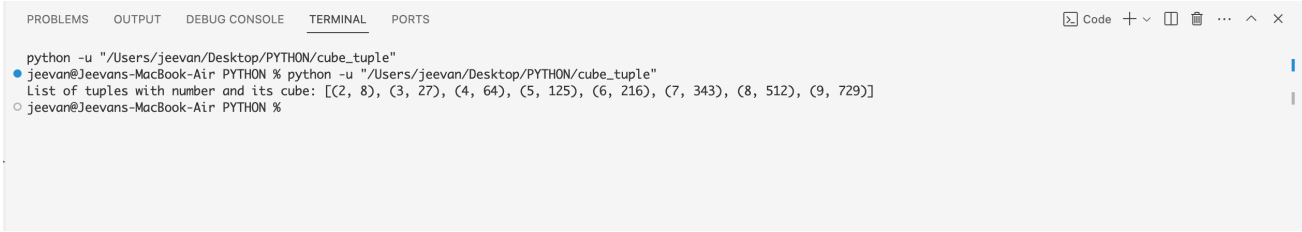
```
# Function to create a list of tuples with number and its cube
def create_tuples_with_cubes(input_list):
    tuples_list = [(num, num ** 3) for num in input_list]
```

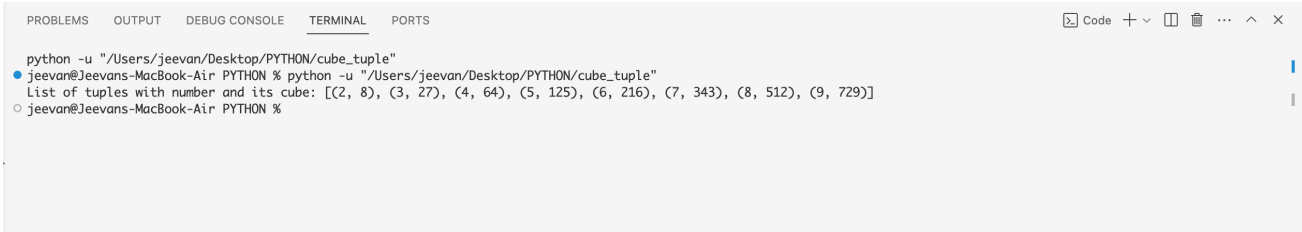
```
    return tuples_list

# Given input list
c = [2, 3, 4, 5, 6, 7, 8, 9]

# Create list of tuples with number and its cube
result_list = create_tuples_with_cubes(c)

# Display the result
print("List of tuples with number and its cube:", result_list)
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

This program showcases the use of list comprehension to efficiently generate a list of tuples. Each tuple contains a number from the given list (`c`) and its corresponding cube. The resulting data structure provides a clear and concise representation of the numbers and their cubes.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 03.01

Title: 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 ()

Theory:

1. Using + Operator:

- The `+` operator facilitates the concatenation of two lists.

- In this scenario, the original list `[1, 2, 3]` is combined with the list `[4, 5, 6]` using the `+` operator, resulting in a new list `[1, 2, 3, 4, 5, 6]`.

2. Using Append():

- The `append()` method is employed to append a single element (in this case, the list `[7, 8, 9]`) to the end of the original list.

- Subsequently, the original list is modified to `[1, 2, 3, [7, 8, 9]]`.

3. Using Extend():

- The `extend()` method is utilized to add individual elements from an iterable (the list `[4, 5, 6]`) to the end of the original list.

- Following the extension, the original list transforms into `[1, 2, 3, 4, 5, 6]`.

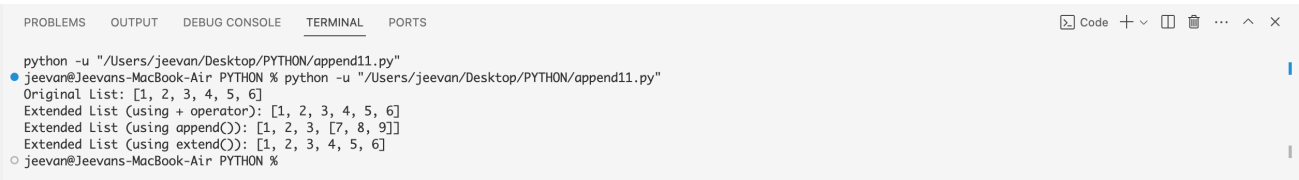
Code:

```
# Given list
original_list = [1, 2, 3]

# i. By using + operator
extended_list_1 = original_list + [4, 5, 6]
```

```python
# ii. By using Append()
original_list.append([7, 8, 9])
extended_list_2 = original_list

# iii. By using extend()
original_list = [1, 2, 3]  # Resetting original list
original_list.extend([4, 5, 6])

# Displaying the results
print("Original List:", original_list)
print("Extended List (using + operator):", extended_list_1)
print("Extended List (using append()):", extended_list_2)
print("Extended List (using extend()):", original_list)
```

Output: (screenshot)

Test Case: Any two (screenshot)



Conclusion:

- The program effectively demonstrates various techniques to extend a list:
  - Leveraging the `+` operator for list concatenation.
  - Utilizing the `append()` method to append a single element.
  - Applying the `extend()` method to add elements from an iterable.
- Mastery of these methods is essential for manipulating lists in Python.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 03.02

Title: Write a program to add two matrices.

Theory:

This Python program defines a function `add_matrices` to add two matrices. The function takes two matrices as input, checks if they have the same dimensions, and returns the sum of the matrices if possible. The main part of the program demonstrates the usage of this function by adding two example matrices (`matrix_a` and `matrix_b`). The result is displayed if the addition is possible; otherwise, a message indicating that the matrices have different dimensions is printed.

Code:

```
# Function to add two matrices
def add_matrices(matrix1, matrix2):
    result_matrix = []

    # Check if the matrices have the same dimensions
    if len(matrix1) == len(matrix2) and len(matrix1[0]) == len(matrix2[0]):
        for i in range(len(matrix1)):
            row_result = []
            for j in range(len(matrix1[0])):
                # Sum the corresponding elements of the two matrices
                row_result.append(matrix1[i][j] + matrix2[i][j])
```

```
            result_matrix.append(row_result)

        return result_matrix
    else:
        return None  # Return None if matrices have different dimensions

# Input matrices
matrix_a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
matrix_b = [[9, 8, 7], [6, 5, 4], [3, 2, 1]]

# Add matrices
result = add_matrices(matrix_a, matrix_b)

# Display the result
if result:
    print("Matrix A:")
    for row in matrix_a:
        print(row)

    print("\nMatrix B:")
    for row in matrix_b:
        print(row)

    print("\nSum of Matrices A and B:")
    for row in result:
        print(row)
```

```
else:
    print("Matrices have different dimensions. Addition not possible.")
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

Matrix addition is a fundamental operation in linear algebra, and this program provides a clear example of how to perform it in Python. Understanding matrix operations is crucial in various fields, including computer graphics, scientific computing, and machine learning.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 03.03

Title: Write a Python function that takes a list and returns a new list with distinct elements from the first list.

Theory:

The `remove_duplicates` function takes a list as input and iterates through its elements. It checks whether each element is already present in the `distinct_list`. If not, it appends the element to the `distinct_list`. This ensures that the resulting list only contains distinct elements.

Code:

```
def remove_duplicates(input_list):
    """
    Function to remove duplicates from a list.

    Parameters:
    - input_list: The input list with possible duplicate elements.

    Returns:
    - A new list with distinct elements from the input list.
    """
    distinct_list = []
    for item in input_list:
        if item not in distinct_list:
```

```
            distinct_list.append(item)
    return distinct_list

# Example usage:
original_list = [1, 2, 2, 3, 4, 4, 5, 6, 6]
result_list =
remove_duplicates(original_list)
print("Original List:", original_list)
print("List with Distinct Elements:",
result_list)
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

Removing duplicates from a list is a common task in programming, especially when dealing with data processing or ensuring uniqueness in a collection of items. The function provided demonstrates a straightforward approach to achieve this in Python.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 03.04

Title: Write a program to Check whether a number is perfect or not.

Theory:

The function takes an input number and follows these steps:

1. Check if the number is less than or equal to 0. If so, return False, as negative numbers and zero are not perfect numbers.

2. Iterate through potential divisors from 2 to the square root of the given number.

3. For each divisor found, update the sum of divisors. If a divisor is different from its corresponding pair, add the pair to the sum.

4. Check if the sum of divisors is equal to the original number.

5. Return True if the number is a perfect number, and False otherwise.

Code:

```
def is_perfect_number(number):
    """
    Function to check whether a number is a perfect number.

    Parameters:
    - number: The input number to be checked.

    Returns:
```

```
    - True if the number is a perfect number,
False otherwise.
    """
    if number <= 0:
        return False

    # Find divisors and calculate the sum
    divisors_sum = 1
    for i in range(2, int(number**0.5) + 1):
        if number % i == 0:
            divisors_sum += i
            if i != number // i:
                divisors_sum += number // i

    # Check if the sum of divisors is equal
to the original number
    return divisors_sum == number

# Example usage:
user_number = int(input("Enter a number: "))
result = is_perfect_number(user_number)

if result:
    print(user_number, "is a perfect
number.")
else:
```

```
    print(user_number, "is not a perfect number.")
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

The function provides a basic implementation for determining whether a given number is a perfect number. Users can input a number, and the program will output whether the number is a perfect number or not based on the defined criteria.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 03.05

Title: 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'

Theory:

The provided Python code defines a function `count_upper_lower` that takes a string as input and counts the number of upper- and lower-case letters in the string. The function uses a for loop to iterate through each character in the string and checks whether it is an upper-case or lower-case letter using the `isupper()` and `islower()` string methods.

The counts are accumulated, and the function returns a tuple containing the count of upper-case letters and lower-case letters.

Code:

```
def count_upper_lower(string):
    """
    Function to count the number of upper- and lower-case letters in a given string.

    Parameters:
    - string: The input string.

    Returns:
    - Tuple containing the count of upper-case letters and lower-case letters.
```

```
    """
    upper_count = 0
    lower_count = 0

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

    return upper_count, lower_count

# Example usage:
string_test = 'Today is My Best Day'
result = count_upper_lower(string_test)

print("Count of upper-case letters:",
result[0])
print("Count of lower-case letters:",
result[1])
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

This function can be useful when there is a need to analyze the distribution of upper-case and lower-case letters in a given string. In the example usage with the string "Today is My Best Day," the function is called with the string, and the counts of upper-case and lower-case letters are printed. Users can incorporate or modify this function based on their specific requirements for letter case analysis in strings.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 04.01

Title: Write a program to Create Employee Class & add methods to get employee details & print.

Theory:

1. `get_employee_details`: Returns a tuple containing the employee's ID, name, and salary.

2. `print_employee_details`: Prints the employee's ID, name, and salary.

In the example usage, the program interacts with the user to input employee details such as ID, name, and salary. An `Employee` object (`employee1`) is then created using the provided inputs. The program retrieves the employee details as a tuple and prints them using the methods defined in the `Employee` class.

Code:

```
class Employee:
    def __init__(self, emp_id, emp_name, emp_salary):
        self.emp_id = emp_id
        self.emp_name = emp_name
        self.emp_salary = emp_salary

    def get_employee_details(self):
        """
        Method to get employee details.

        Returns:
```

```
        - Tuple containing employee ID,
employee name, and employee salary.
        """
        return self.emp_id, self.emp_name,
self.emp_salary

    def print_employee_details(self):
        """Method to print employee
details."""
        print("Employee ID:", self.emp_id)
        print("Employee Name:",
self.emp_name)
        print("Employee Salary:",
self.emp_salary)

# Example usage with user input:
# Taking inputs from the user
emp_id = input("Enter Employee ID: ")
emp_name = input("Enter Employee Name: ")
emp_salary = input("Enter Employee Salary: ")

# Creating an Employee object with user
inputs
employee1 = Employee(emp_id=emp_id,
emp_name=emp_name, emp_salary=emp_salary)

# Getting and printing employee details
```

```
details_tuple = employee1.get_employee_details()
print("\nEmployee Details (as tuple):", details_tuple)

print("\nEmployee Details (printed):")
employee1.print_employee_details()
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

This program illustrates the concept of encapsulation in object-oriented programming by using a class to model an employee and methods to access and display the employee's details. User input enables the creation of an `Employee` object with dynamic information, showcasing the flexibility and reusability of the class.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 04.02

Title: Write a program to take input as name, email & age from user using combination of keywords argument and positional arguments (*args andkwargs) using function**

Theory:

In this program, we have created a function called `get_user_details` that takes a combination of positional arguments (`*args`) and keyword arguments (`**kwargs`). The function is designed to collect user details and return them as a dictionary.

- The `*args` parameter allows us to accept a variable number of positional arguments, which are used to capture the user's name, email, and age.

- The `**kwargs` parameter allows us to accept additional keyword arguments, such as city and country.

- Inside the function, we check if there are at least three positional arguments provided (name, email, age). If so, we update the `user_details` dictionary with these details.

- We then update the `user_details` dictionary with any additional keyword arguments provided.

- The final result is a dictionary containing user details, including both positional and keyword arguments.

Code:

```
# 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
def get_user_details(*args, **kwargs):
    """
    Function to get user details.

    Parameters:
    - *args: Positional arguments (name,
email, age)
    - **kwargs: Keyword arguments

    Returns:
    - Dictionary containing user details
    """
    user_details = {}

    # Check if positional arguments are
provided
    if len(args) >= 3:
        user_details['name'] = args[0]
        user_details['email'] = args[1]
        user_details['age'] = args[2]

    # Update with keyword arguments
    user_details.update(kwargs)

    return user_details
```

```python
# Example usage:
name_input = input("Enter your name: ")
email_input = input("Enter your email: ")
age_input = int(input("Enter your age: "))

# Using the function with a combination of positional and keyword arguments
user_info = get_user_details(name_input, email_input, age_input, city='Example City', country='Example Country')

# Display the user details
print("\nUser Details:")
for key, value in user_info.items():
    print(f"{key}: {value}")
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

This program demonstrates the use of a flexible function that can handle a varying number of inputs from the user. By combining positional and keyword arguments, it provides a versatile way to collect user details and store them in a dictionary. The program concludes by displaying the user details using a loop that iterates through the dictionary items.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 04.03

Title: Write a program to admit the students in the different Departments(pgdm/ btech)and count the students. (Class, Object and Constructor).

Theory:

In this program, we have defined a `Student` class with class variables (`total_students`, `btech_students`, and `pgdm_students`) to keep track of the number of students and the number of students in each department. The class has an `__init__` method that initializes the instance variables (`name` and `department`) and increments the class variables based on the department of the student.

The `admit_students` function is responsible for taking user inputs for the number of students to admit, creating a list of `Student` objects based on user inputs, and displaying the details of admitted students along with the total count and counts for each department.

Code:

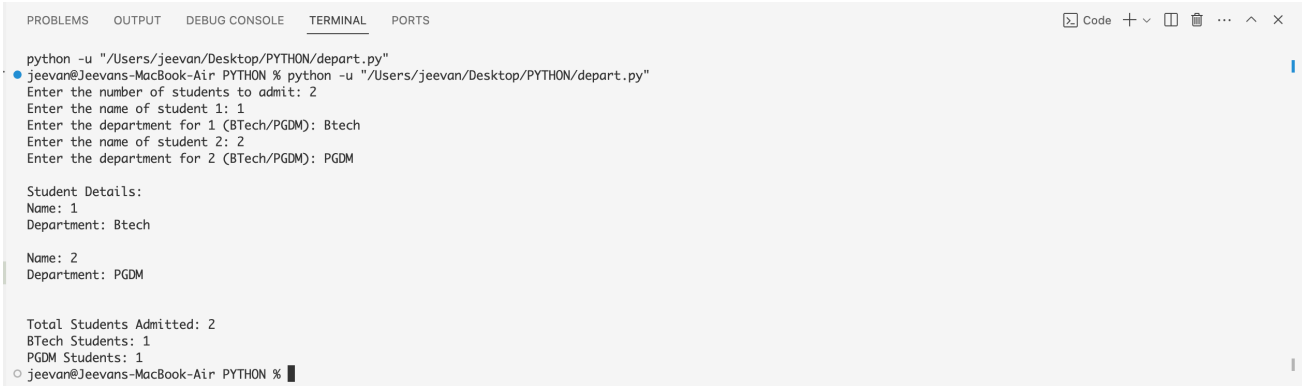
```
class Student:
    # Class variables to count the number of students for each department
    total_students = 0
    btech_students = 0
    pgdm_students = 0

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

```
        # Increment the total_students count
        Student.total_students += 1

        # Increment the department-specific
count
        if department.upper() == "BTECH":
            Student.btech_students += 1
        elif department.upper() == "PGDM":
            Student.pgdm_students += 1

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Department: {self.department}
\n")

# Function to admit students
def admit_students():
    # Number of students to admit
    num_students = int(input("Enter the
number of students to admit: "))

    # Creating a list to store student
objects
    students_list = []
```

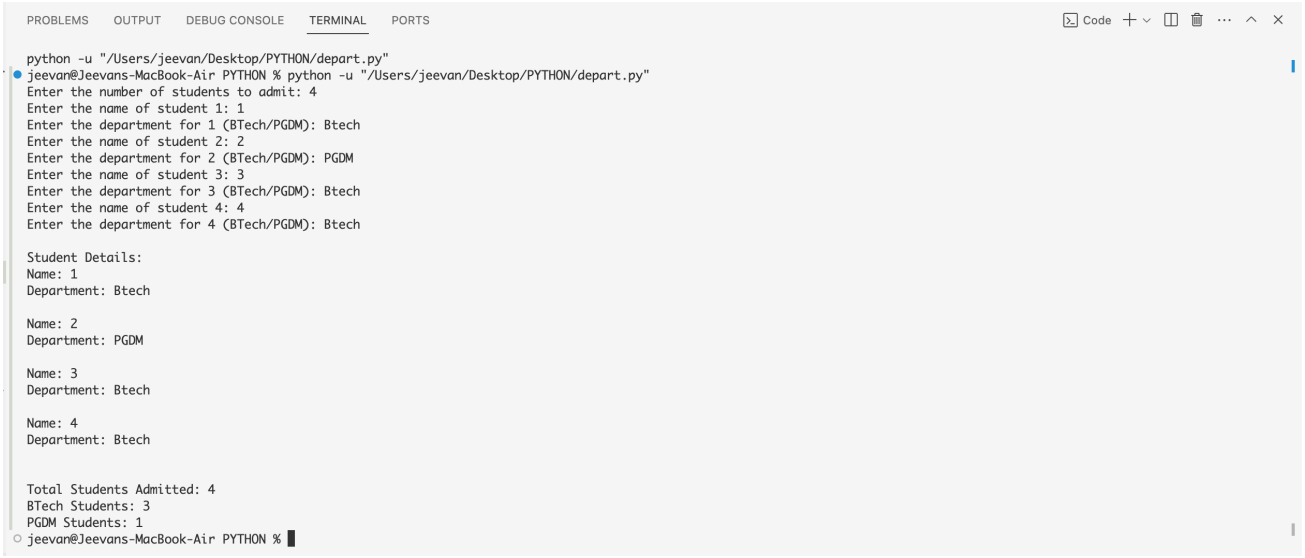
```
    # Loop to take user inputs for each
student
    for i in range(num_students):
        name = input(f"Enter the name of
student {i + 1}: ")
        department = input(f"Enter the
department for {name} (BTech/PGDM): ")

        # Creating an object of the Student
class and adding it to the list
        student = Student(name, department)
        students_list.append(student)

    # Displaying student details
    print("\nStudent Details:")
    for student in students_list:
        student.display_details()

    # Displaying the total number of students
    print(f"\nTotal Students Admitted:
{Student.total_students}")
    print(f"BTech Students:
{Student.btech_students}")
    print(f"PGDM Students:
{Student.pgdm_students}")

# Calling the function to admit students
```

`admit_students()`

Output: (screenshot)



Test Case: Any two (screenshot)



Conclusion:

This program demonstrates the use of a class to model students, class variables to maintain counts, and methods to interact with the class. It provides a structured way to manage student information and count the number of students in different departments. The use of user inputs allows for dynamic data entry, making the program flexible for various scenarios.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 04.04

Title: 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.

Theory:

1. Class and Object:

- The `Store` class is defined to represent a product in the store. It has attributes like `name`, `__code`, and `__price`.

- Objects of the `Store` class are created for each product in the `products` list.

2. Encapsulation (Private Variables):

- Encapsulation is utilized by marking the `__code` and `__price` attributes as private. This is done by prefixing them with double underscores. It restricts direct access to these attributes from outside the class.

3. Initialization (__init__ method):

- The `__init__` method is used for initializing the attributes of the `Store` class when an object is created.

4. Menu Display:

- The code displays a menu by iterating through the `products` list and printing each product's name and price.

5. User Input and List Manipulation:

- The program prompts the user to enter the quantity of each product. The quantities are stored in a list (`quantities`), maintaining the order of the products.

6. Bill Generation:

- The code calculates the subtotal for each product based on the entered quantity and the product's price. It then accumulates the total amount by summing up the subtotals.

7. Enumeration:

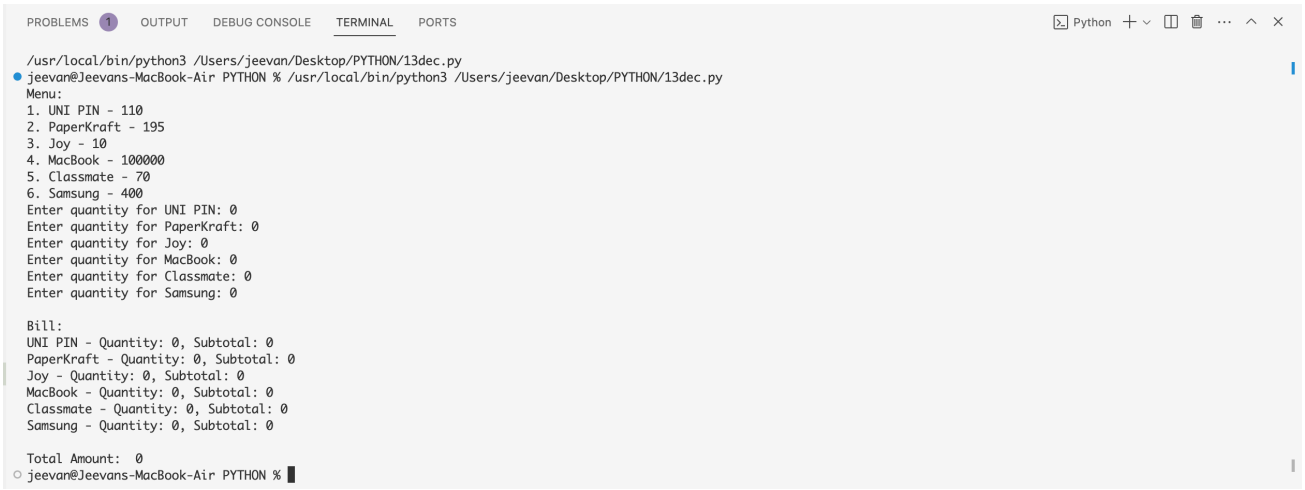
- The `enumerate` function is used to iterate over both the index and the corresponding product in the `products` list, starting from 1

Code:

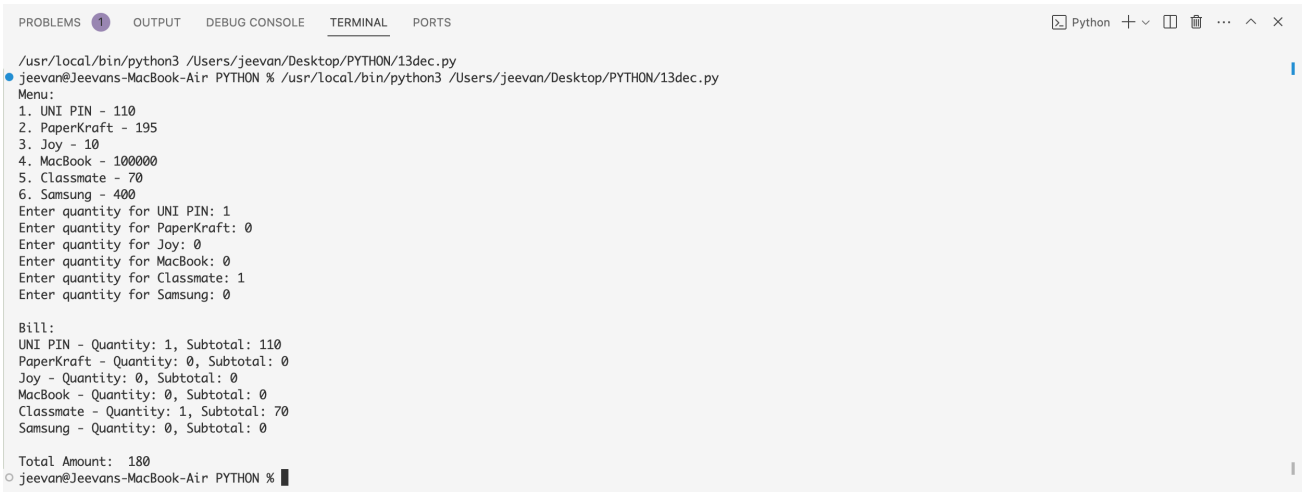
```
class Store:
    def __init__(self, name, code, price):
        self.name = name
        self.__code = code
        self.__price = price

# Create instances of the Store class for
each product
products = [
    Store("UNI PIN", "49 02778 915262", 110),
    Store("PaperKraft", "02252005", 195),
    Store("Joy", "123 321", 10),
    Store("MacBook", "AppleMac123", 100000),
    Store("Classmate", "Class123", 70),
    Store("Samsung", "Sam 123", 400)
]

# Display menu
print("Menu:")
for idx, product in enumerate(products,
start=1):
    print(f"{idx}. {product.name} -
{product._Store__price}")

# Prompt user for quantities
quantities = []
```

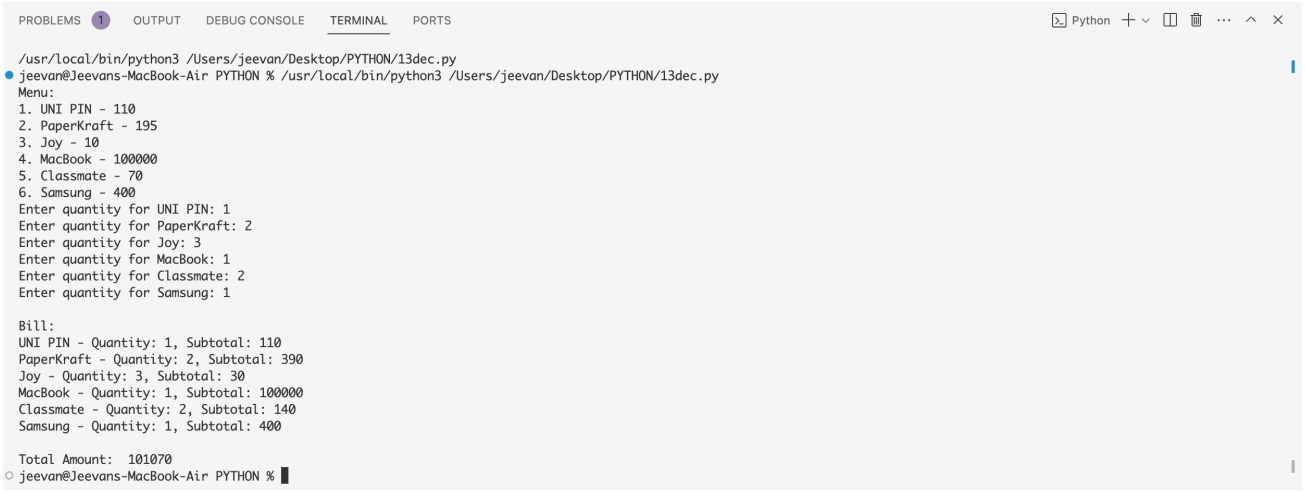
```
for idx, product in enumerate(products,
start=1):
    quantity = int(input(f"Enter quantity for
{product.name}: "))
    quantities.append(quantity)

# Generate and display the bill
total_amount = 0
print("\nBill:")
for idx, product in enumerate(products,
start=1):
    quantity = quantities[idx - 1]
    subtotal = quantity *
product._Store__price
    total_amount += subtotal
    print(f"{product.name} - Quantity:
{quantity}, Subtotal: {subtotal}")

print("\nTotal Amount: ", total_amount)
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

The provided code demonstrates the use of object-oriented programming (OOP) principles such as encapsulation and class instantiation to model a store system. It involves creating a class to represent products, displaying a menu, obtaining user input for quantities, and generating a bill. This program showcases the practical application of OOP concepts in a real-world scenario, making the code modular and easy to understand.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 04.05

Title: Write a program to take input from user for addition of two numbers using (single inheritance).

Theory:

1. Parent Class (`Parent`):

- The `Parent` class has a method named `add` that takes two parameters (`a` and `b`), adds them, and returns the result.

- The `add` method is designed to be used by the child class.

2. Child Class (`Child`):

- The `Child` class is derived from the `Parent` class, indicating a single inheritance relationship.

- It has a method called `takevalues`, which takes user input for two numbers, calls the `add` method from the parent class, and prints the result.

3. Object Creation and Method Invocation:

- An object `obj` of the `Child` class is created.

- The `takevalues` method of the `Child` class is invoked on this object.

4. User Input and Method Invocation:

- The `takevalues` method prompts the user to input two numbers (`a` and `b`).

- It then calls the `add` method from the parent class (`Parent`) with the entered values and prints the result.

Code:

```
#Single_Inheritance
class Parent():
    def add(self,a,b):
        self.a=a
```

```
        self.b=b
        return self.a+self.b
class Child(Parent):
    def takevalues(self):
        a=int(input("Enter the first number: "))
        b=int(input("Enter the second number: "))
        c=self.add(a,b)
        print(c)
obj=Child()
obj.takevalues()
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

This example demonstrates the basic concept of single inheritance, where the child class (`Child`) inherits the method `add` from its parent class (`Parent`). The child class extends the functionality by adding a method (`takevalues`) to obtain user input and use the inherited method for performing

addition. Single inheritance provides a way to create a hierarchy of classes, facilitating code reuse and maintaining a logical structure in object-oriented programming.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 04.06

Title: Write a program to create two base classes LU and ITM and one derived class. (Multiple inheritance).

Theory:

1. Base Classes (`LetsUpgrade` and `ITM`):

- `LetsUpgrade` and `ITM` are two base classes containing information about courses, instructors, and durations.

- Each class encapsulates a 2D list representing different aspects of the courses.

2. Derived Class (`Btech`):

- `Btech` is the derived class that inherits from both `LetsUpgrade` and `ITM`.

- It has an `__init__` method to initialize an empty list called `subjects` to store the chosen subjects.

3. Methods:

- `courses`: Displays available courses from both `LetsUpgrade` and `ITM`.

- `choice`: Takes user input for the number of subjects to choose and the subject numbers.

- `show_invoice`: Generates and displays an invoice for the selected subjects, including course name, instructor, and duration.

4. Subject Selection and Invoice Generation:

- The user is prompted to enter the number of subjects they want to choose and then input the subject numbers.

- The program then calculates the total duration of the selected subjects.

Code:

```
class LetsUpgrade:
    a = [["Python", "CPP", "Java"], ["Saurabh
sir", "Sai sir", "Prasad sir"], ["12:00",
"15:00", "02:00"]]

class ITM:
    b = [["Soft skills", "Business Studies",
"Entrepreneurship"], ["Sakshi ma'am",
"Sheetal ma'am", "Sumit sir"],
         ["16:00", "19:00", "23:00"]]

class Btech(LetsUpgrade, ITM):
    def __init__(self):
        self.subjects = []

    def courses(self):
        print("LetsUpgrade")
        for i in range(3):
            print(i + 1, "=",
LetsUpgrade.a[0][i], "--", LetsUpgrade.a[1]
[i], "--", LetsUpgrade.a[2][i])
        print()
        print("ITM")
        for i in range(3):
            print(i + 4, "=", ITM.b[0][i],
"--", ITM.b[1][i], "--", ITM.b[2][i])
```

```
    def choice(self):
        num_subjects = int(input("Enter the
number of subjects: "))
        for _ in range(num_subjects):
            subject_no = int(input("Enter the
subject no.: "))
            self.subjects.append(subject_no)

    def show_invoice(self):
        total_duration = 0
        print("\nInvoice:")
        for subject_no in self.subjects:
            if 1 <= subject_no <= 3:
                category = LetsUpgrade
                subject_index = subject_no -
1
            elif 4 <= subject_no <= 6:
                category = ITM
                subject_index = subject_no -
4
            else:
                print(f"Invalid subject no.:
{subject_no}")
                continue
```

```
            course_name = category.a[0]
[subject_index] if hasattr(category, 'a')
else category.b[0][subject_index]
            instructor = category.a[1]
[subject_index] if hasattr(category, 'a')
else category.b[1][subject_index]
            duration = category.a[2]
[subject_index] if hasattr(category, 'a')
else category.b[2][subject_index]
            total_duration +=
int(duration.split(':')[0]) * 60 +
int(duration.split(':')[1])

            print(f"Subject {subject_no}:
{course_name} - {instructor} - {duration}
hours")

        print(f"\nTotal Duration:
{total_duration // 60} hours {total_duration
% 60} minutes")


# Create an object of the derived class
obj = Btech()

# Access methods from base classes
obj.courses()
```

```
obj.choice()
obj.show_invoice()
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

This program illustrates the use of multiple inheritance in Python, allowing a derived class to inherit attributes and methods from more than one base class. It showcases the flexibility of OOP (Object-Oriented Programming) to model and organize code, providing a clear structure for handling related information. The program efficiently manages course details from two different sources and allows users to interactively choose subjects, providing them with a detailed invoice.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 04.07

Title: Write a program to implement Multilevel inheritance,

Grandfather→Father-→Child to show property inheritance from grandfather to child.

Theory:

1. `Grandfather` Class:

- Represents the grandfather in the family.

- Initializes attributes for the grandfather's name, assets, and properties.

2. `Father` Class (Inherits from `Grandfather`):

- Represents the father, inheriting from the `Grandfather` class.

- Initializes additional attributes for the father's name, increased assets, and properties.

3. `Child` Class (Inherits from `Father`):

- Represents the child, inheriting from the `Father` class.

- Takes user input for the child's name and initializes attributes for the child's full name.

- Prints a greeting and displays the total inherited assets and purchased properties.

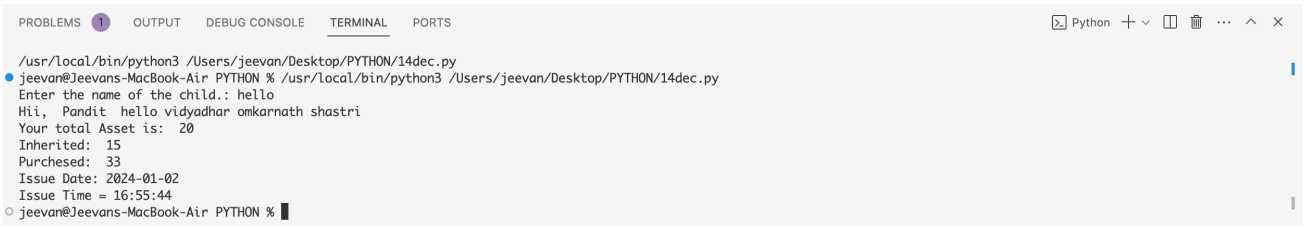
Code:

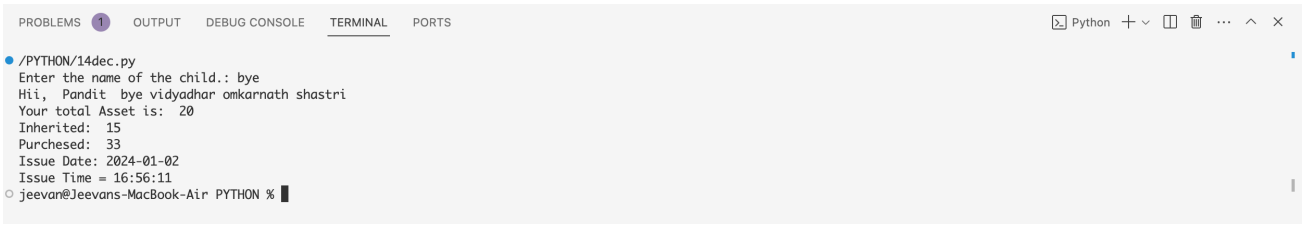
```
class Grandfather():
    def __init__(self):
        self.name="omkarnath shastri"
        self.asset=15
```

```
        self.property=20
class Father(Grandfather):
    def __init__(self):
        super().__init__()
        self.namefather="vidyadhar"+" "+
self.name
        self.assetfather=self.asset +5
        self.propertyfather=13
class Child(Father):
    def __init__(self):
        super().__init__()
        a=input("Enter the name of the
child.: ")
        self.namechild="Pandit "+" "+a+" "+
self.namefather
        print("Hii, ", self.namechild)
        print("Your total Asset is:
",self.assetfather)
        print("Inherited: ",self.asset)
        print("Purchesed: ",self.property +
self.propertyfather)


obj=Child()
```

Output: (screenshot)

Test Case: Any two (screenshot)





Conclusion:

1. Inheritance Hierarchy:

- The code demonstrates a simple inheritance hierarchy with three classes, representing different generations in a family.

2. Attribute Initialization:

- Each class initializes its own attributes in addition to calling the `super().__init__()` method to initialize attributes from the parent class.

3. User Interaction:

- The `Child` class takes user input for the child's name and displays a greeting along with information about inherited assets and purchased properties.

4. Inheritance Concept:

- The code illustrates the concept of inheritance, where child classes inherit attributes and behaviors from their parent classes. This promotes code reuse and establishes a hierarchical relationship among the classes.

5. Super() Function:

- The `super().__init__()` function is used to call the constructor of the parent class, ensuring that attributes from the parent class are properly initialized before adding attributes specific to the child class.

Overall, the code provides a simple example of how inheritance can be used to model relationships between family members in an object-oriented programming paradigm.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 04.08

Title: **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)**

Theory:

Method overriding is when there is a method of same name in both base class and derived class and when an object is created of derived class and method is called, derived class object is called and base class method is overridden.

Code:

```
#WAP 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)
class Library_item:
        _bookCount=0
        _dvdCount=0
```

```
        _journalCount=0
        quantity=[]
        cart=[]
class Book(Library_item):
    def __init__(self):
        super().__init__()
        def check_out_book(self):
            d=int(input("How many books you
want to select?: "))
            if d>4:
                print("Invalid choice")
            else:
                for i in range(d):
                    c=int(input("Select a
book: "))
                    f=int(input("Enter
quantity: "))

Library_item.quantity.append(f)
                    self.cart.append(a[c-1])

Library_item._bookCount+=1
                print(self.cart)
                print("Books
selected:",Library_item._bookCount)
```

```
        a=["C programming by Ritchie-
Cunningham","C++ by Balaguruswamy","R.D.
Sharma","Class 12 CS by NCERT"]
        b=1
        for i in range(len(a)):
            print(b,a[i])
            b+=1
        check_out_book(self)
class Dvd(Library_item):
    def __init__(self):
        super().__init__()
        def check_out_dvd(self):
            d=int(input("How many DVD's you
want to select?: "))
            for i in range(d):
                if d>4:
                    print("Invalid choice")
                else:
                    c=int(input("Select a
DVD: "))
                    f=int(input("Enter
quantity: "))

Library_item.quantity.append(f)
                    self.cart.append(a[c-1])
                    Library_item._dvdCount+=1
            print(self.cart)
```

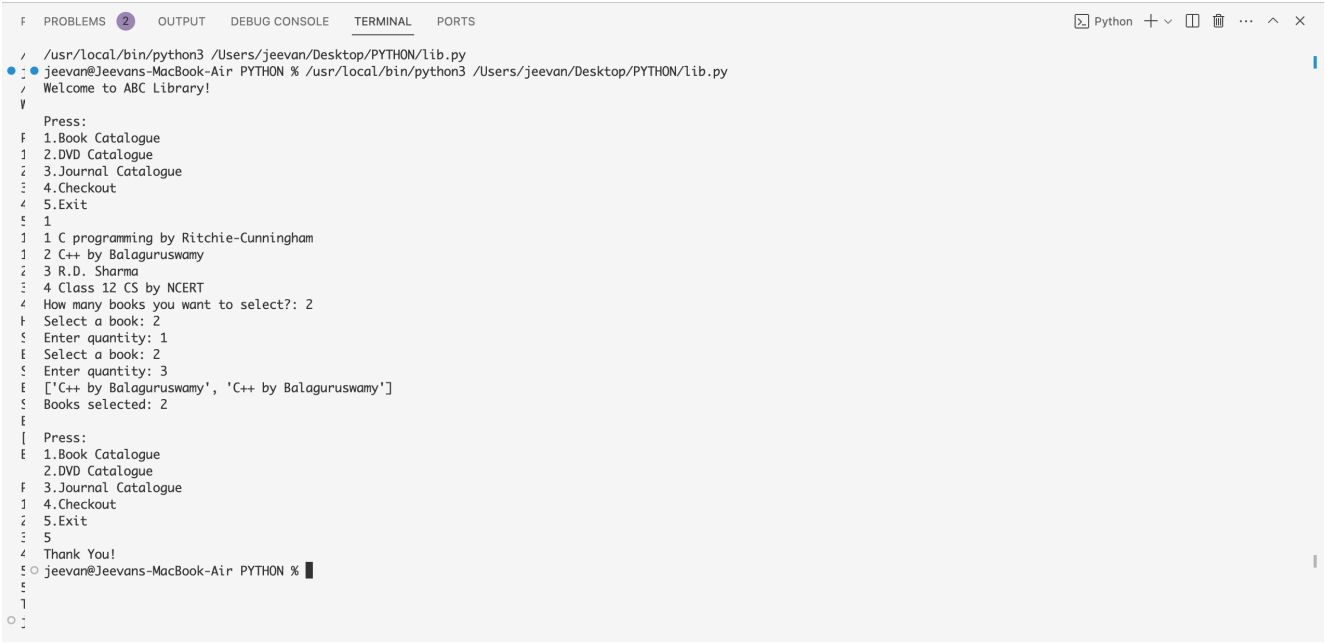
```
            print("DVDs
selected:",Library_item._dvdCount)
        a=["Avengers","Justice
League","Conjuring","ABC"]
        b=1
        for i in range(len(a)):
            print(b,a[i])
            b+=1
        check_out_dvd(self)
class Journal(Library_item):
    def __init__(self):
        super().__init__()
        def check_out_journal(self):
            d=int(input("How many Journal you
want to select?: "))
            if d>4:
                print("Invalid choice")
            else:
                for i in range(d):
                    c=int(input("Select a
journal: "))
                    f=int(input("Enter
quantity: "))

Library_item.quantity.append(f)
                    self.cart.append(a[c-1])
```

```
Library_item._journalCount+=1
                print(self.cart)
                print("Journals
selected:",Library_item._journalCount)
        a=["A Journal","XYZ Journal","ABC
Journal","QWERTY Journal"]
        b=1
        for i in range(len(a)):
            print(b,a[i])
            b+=1
        check_out_journal(self)
class Checkout(Library_item):
    def __init__(self):
        super().__init__()
        print("\nCheckout")
        j=1
        for i in range(len(self.cart)):

print(j,self.cart[i],"-",self.quantity[i])
            j+=1
        print("\nBooks
selected:",self._bookCount)
        print("Journals
selected:",self._journalCount)
        print("DVDs
selected:",self._dvdCount)
```

```
print("Total:",self._bookCount+self._journalC
ount+self._dvdCount)
objs=list()
print("Welcome to ABC Library!")
for i in range(100):
    a=int(input("\nPress:\n1.Book
Catalogue\n2.DVD Catalogue\n3.Journal
Catalogue\n4.Checkout\n5.Exit\n"))
    if a==1:
        objs.append(Book())
    elif a==2:
        objs.append(Dvd())
    elif a==3:
        objs.append(Journal())
    elif a==4:
        objs.append(Checkout())
    elif a==5:
        print("Thank You!")
        break
    else:
        print("Invalid choice")
        break
```

Output: (screenshot)

Test Case: Any two (screenshot)



Conclusion:

Hence, using single inheritance and method overriding, made a library catalogue system having checkout system for books, movies, and journals.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 05.01

Title: Write a program to create my_module for addition of two numbers and import it in main script.

Theory:

Module is a collection of functions. Its functions can be used in another program by importing the module.

Code:

```
class Addition:
        def add(self,x,y):
            print("Sum:",x+y)
```

```python
#WAP to create my_module for addition of two numbers and import it in main script.
import addition
a=addition.Addition()
b=float(input("Enter a number: "))
c=float(input("Enter another number: "))
a.add(b,c)
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

Hence, creating a module for addition of two values given by the user and printing it by importing the module in main program.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 05.02

Title: 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.

Theory:

Module is a collection of functions. Its functions can be used in another program by importing the module.

Code:

```
class ATM:
    __balance=50000
    __pin=0
```

```
    __name=""
    def __init__(self):
        print("Welcome to ABC Bank")
        self.__pin=int(input("Enter your PIN:
"))
        self.__name=input("Enter your name:
")
    def deposit(self):
        a=float(input("Enter amount to
deposit: "))
        if a<=0:
            print("Invalid amount\n")
        else:
            print("Successfully deposited
Rs",a,"cash\n")
            self.__balance+=a
    def withdraw(self):
        a=float(input("Enter amount to
withdraw: "))
        if a<=0 or a>self.__balance:
            print("Invalid amount\n")
        else:
            print("Successfully withdrew
Rs",a,"cash\n")
            self.__balance-=a
    def check(self):
        print("Balance: Rs", self.__balance)
```

```
        print(f"Your Account Number
XXXXXXXX{self.__pin}\n")

#WAP 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.
from bank import ATM
b=ATM()
while True:
    a=int(input("Press 1 to deposit\nPress 2
to withdraw\nPress 3 to check balance\nPress
4 to Exit\n"))
    if a==1:
        b.deposit()
    elif a==2:
        b.withdraw()
    elif a==3:
        b.check()
    elif a==4:
        print("Thank You")
        break
    else:
        print("Invalid choice\n")
```

Output: (screenshot)

```
PROBLEMS 2  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                    Python

/usr/local/bin/python3 /Users/jeevan/Desktop/PYTHON/bank/main.py
jeevan@Jeevans-MacBook-Air PYTHON % /usr/local/bin/python3 /Users/jeevan/Desktop/PYTHON/bank/main.py
Welcome to ABC Bank
Enter your PIN: 1234
Enter your name: sad
Press 1 to deposit
Press 2 to withdraw
Press 3 to check balance
Press 4 to Exit
1
Enter amount to deposit: 200
Successfully deposited Rs 200.0 cash

Press 1 to deposit
Press 2 to withdraw
Press 3 to check balance
Press 4 to Exit
3
Balance: Rs 50200.0
Your Account Number XXXXXXXX1234

Press 1 to deposit
Press 2 to withdraw
Press 3 to check balance
Press 4 to Exit
```

Test Case: Any two (screenshot)

```
PROBLEMS 2  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                    Python

/usr/local/bin/python3 /Users/jeevan/Desktop/PYTHON/bank/main.py
jeevan@Jeevans-MacBook-Air PYTHON % /usr/local/bin/python3 /Users/jeevan/Desktop/PYTHON/bank/main.py
Welcome to ABC Bank
Enter your PIN: 1234
Enter your name: asdfg
Press 1 to deposit
Press 2 to withdraw
Press 3 to check balance
Press 4 to Exit
4
Thank You
jeevan@Jeevans-MacBook-Air PYTHON %
```

```
PROBLEMS 2  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                    Python

/usr/local/bin/python3 /Users/jeevan/Desktop/PYTHON/bank/main.py
jeevan@Jeevans-MacBook-Air PYTHON % /usr/local/bin/python3 /Users/jeevan/Desktop/PYTHON/bank/main.py
Welcome to ABC Bank
Enter your PIN: 345
Enter your name: dfdg
Press 1 to deposit
Press 2 to withdraw
Press 3 to check balance
Press 4 to Exit
1
Enter amount to deposit: 500
Successfully deposited Rs 500.0 cash

Press 1 to deposit
Press 2 to withdraw
Press 3 to check balance
Press 4 to Exit
3
Balance: Rs 50500.0
Your Account Number XXXXXXXX345

Press 1 to deposit
Press 2 to withdraw
Press 3 to check balance
Press 4 to Exit
4
Thank You
jeevan@Jeevans-MacBook-Air PYTHON %
```

Conclusion:

Hence, creating a module for bank ATM with functions for deposit, withdraw, and checking bank balance and asking the user for deposit, withdraw or check bank balance and calling the user specified function using while loop.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 05.03

Title: 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.

Theory:

A package is a collection of modules. It is a folder containing modules and __init__.py file to let python treat the folder as a package. Each module has methods for drive and start engine.

Code:

```
class Audi:
    def __init__(self, model):
        self.model = model
    def start_engine(self):
        print(f"Audi {self.model} engine
started.")
    def drive(self):
        print(f"Driving the Audi
{self.model}.")

class BMW:
    def __init__(self, model):
        self.model = model
    def start_engine(self):
        print(f"BMW {self.model} engine
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} engine
started.")
    def drive(self):
        print(f"Driving the Nissan
{self.model}.")

#WAP 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.
from cars.bmw import BMW
from cars.audi import Audi
from cars.nissan import Nissan

bmw_car = BMW(model="X5")
bmw_car.start_engine()
bmw_car.drive()
print()
audi_car = Audi(model="A4")
audi_car.start_engine()
audi_car.drive()
print()
nissan_car = Nissan(model="Altima")
nissan_car.start_engine()
nissan_car.drive()
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

Hence, creating a package containing modules for different car models, with each module containing methods for each car model.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 06.01

Title: Write a program to implement Multithreading. Printing “Hello” with one thread & printing “Hi” with another thread.

Theory:

Multithreading is used for multitasking as multiple processes are run in parallel at the same time. Each process is given a thread to run on and all the threads are joined together to create a main thread at the end of execution. Sleep is a function in time module which is used to suspend the execution of a thread for a specified number of seconds.

Code:

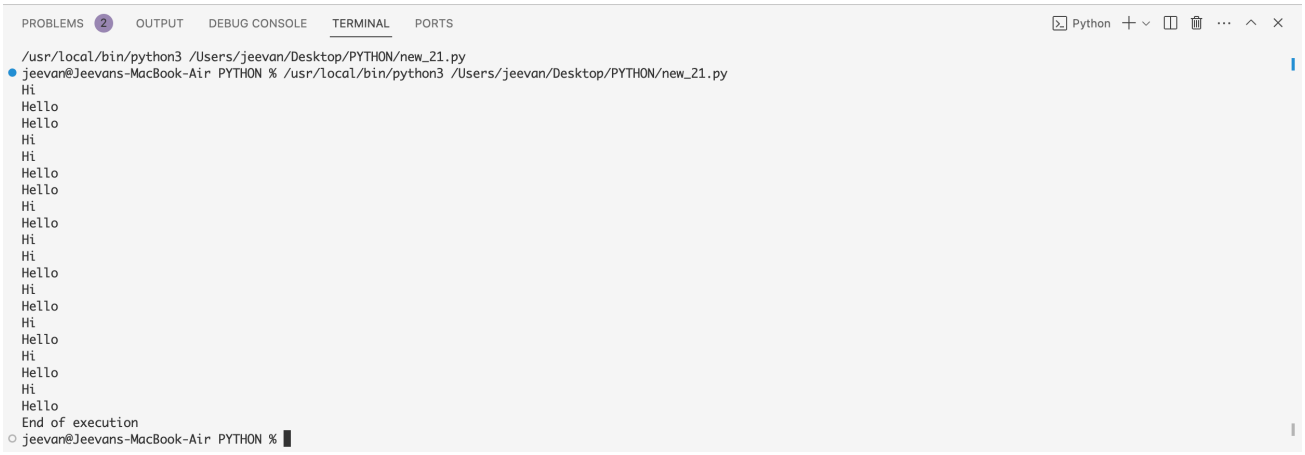
```python
#WAP to implement Multithreading. Printing "Hello" with one thread & printing "Hi" with another thread.
from threading import Thread
from time import sleep
def hi():
    for i in range(10):
        print("Hi")
        sleep(0.5)
def hello():
    for i in range(10):
        print("Hello")
        sleep(0.5)
t1=Thread(target=hi)
t2=Thread(target=hello)
t1.start()
t2.start()
t1.join()
t2.join()
print("End of execution")
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

Hence, by using multithreading module in python to run multiple processes at once in parallel with each process getting their own threads to run on and joining them all at the end of execution of program. Also using sleep function from time module to let the processes on the threads run after a specified interval of time.



Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 07.01

Title: 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.

Theory:

1. OpenWeatherMap API:

- The code utilizes the OpenWeatherMap API to obtain real-time weather data for a specified city.

- The API key is used for authentication and authorization to access the weather information.

2. Kelvin to Celsius and Fahrenheit Conversion:

- The temperature data obtained from the API is in Kelvin. The `kelvin_to_celsius_fahrenheit` function is used to convert the temperature to both Celsius and Fahrenheit scales.

3. Weather Data Retrieval:

- The program extracts relevant weather information, such as temperature, feels-like temperature, wind speed, humidity, general weather description, sunrise, and sunset times.

4. Date and Time Handling:

- The `datetime` module is employed to convert Unix timestamps received from the API into human-readable date and time formats.

- The local time is adjusted based on the city's timezone.

Code:

```
import datetime as dt
import requests

BASE_URl = "http://api.openweathermap.org/
data/2.5/weather?"
API_KEY = "ef374978325e8ff38db840f71f5f2d09"
CITY = input("City: ")

def kelvin_to_celsius_fahrenheit(kelvin):
    celsius = kelvin -273.15
```

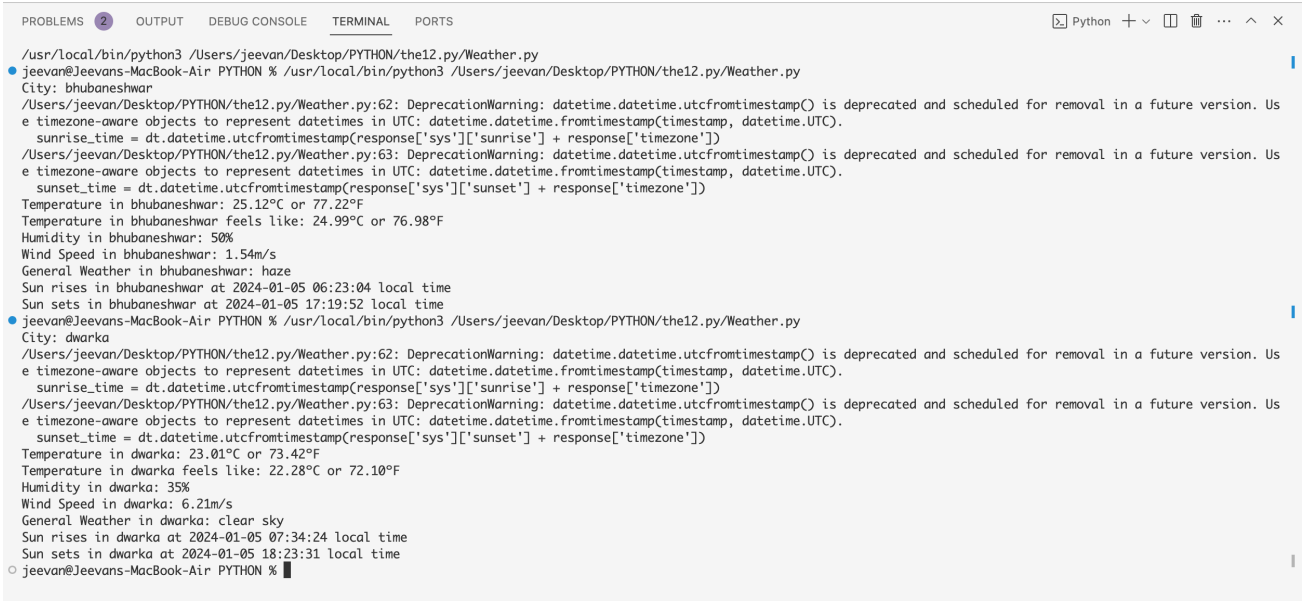
```
    fahrenheit = celsius * (9/5) + 32
    return celsius, fahrenheit

url = BASE_URl + "appid=" + API_KEY + "&q="
+CITY

response = requests.get(url).json()


temp_kelvin = response['main']['temp']
temp_celsius, temp_fahrenheit =
kelvin_to_celsius_fahrenheit(temp_kelvin)
feels_like_kelvin = response['main']
['feels_like']
feels_like_celsius, feels_like_fahrenheit =
kelvin_to_celsius_fahrenheit(feels_like_kelvi
n)
wind_speed = response['wind']['speed']
humidity = response['main']['humidity']
description = response['weather'][0]
['description']
sunrise_time =
dt.datetime.utcfromtimestamp(response['sys']
['sunrise'] + response['timezone'])
sunset_time =
dt.datetime.utcfromtimestamp(response['sys']
['sunset'] + response['timezone'])
```

```
print(f"Temperature in {CITY}: {temp_celsius:.2f}°C or {temp_fahrenheit:.2f}°F")
print(f"Temperature in {CITY} feels like: {feels_like_celsius:.2f}°C or {feels_like_fahrenheit:.2f}°F")
print(f"Humidity in {CITY}: {humidity}%")
print(f"Wind Speed in {CITY}: {wind_speed}m/s")
print(f"General Weather in {CITY}: {description}")
print(f"Sun rises in {CITY} at {sunrise_time} local time")
print(f"Sun sets in {CITY} at {sunset_time} local time")
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:



In conclusion, this Python program serves as a practical tool for retrieving and displaying real-time weather information for a specified city using the OpenWeatherMap API. Key features of the program include temperature conversion, providing both Celsius and Fahrenheit scales, as well as presenting additional weather details such as humidity, wind speed, and sunrise/sunset times in a user-friendly format.

Name of Student: Jeevan Naidu

Roll Number: 15

Experiment No: 07.02

Title: Write a program to use the 'API' of crypto currency.

Theory:

Cryptocurrencies are digital or virtual currencies that use cryptography for security and operate on a decentralized network of computers. Prices of cryptocurrencies are highly volatile and can change rapidly. The code you provided uses the CoinGecko API to fetch real-time prices of a specified cryptocurrency in both USD and INR.

1. API (Application Programming Interface):

   - An API allows different software applications to communicate with each other.

   - In this case, the CoinGecko API is used to get cryptocurrency prices.

2. Cryptocurrency Price Fetching:

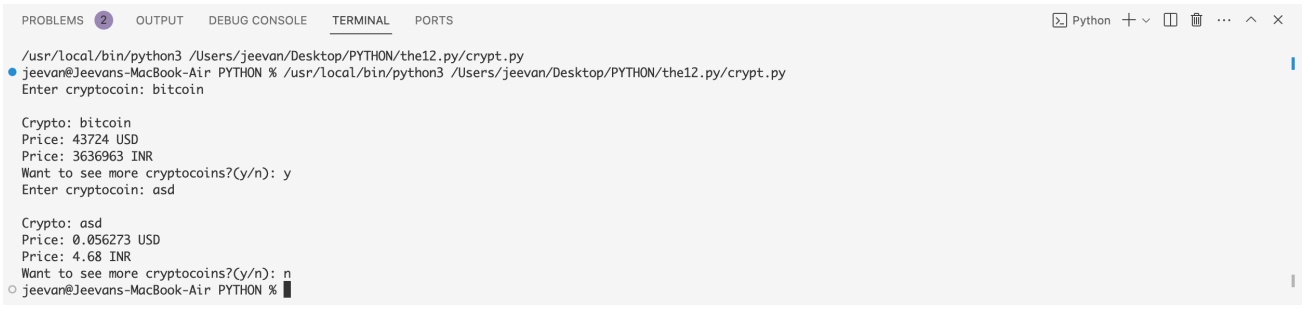
- The code fetches real-time cryptocurrency prices in USD and INR using the CoinGecko API.

3. User Interaction:

- The program interacts with the user by taking input for the cryptocurrency they want to check and asking if they want to check more.

Code:

```
#WAP to use the API of crypto currency
import requests
api_id="CG-iUeKqns6CWFzks6qxkfymPQV"
while True:
    coin=input("Enter cryptocoin: ")
    response = requests.get(f"https://
api.coingecko.com/api/v3/simple/price?
ids={coin}
&vs_currencies=usd,inr&x_cg_demo_api_key={api
_id}")
    a=response.json()
    if coin in a:
        print("\nCrypto:",coin)
        print("Price:",a[coin]['usd'],"USD")
        print("Price:",a[coin]['inr'],"INR")
    else:
        print("Invalid cryptocoin!")
    b=input("Want to see more cryptocoins?(y/
n): ")
```




```
    if b.lower() == "n":
        break
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

In conclusion, this Python program provides a simple interface for users to check real-time prices of cryptocurrencies using the CoinGecko API. It demonstrates how APIs can be leveraged to access external data, making it a practical tool for individuals interested in monitoring cryptocurrency prices. The infinite loop ensures continuous user interaction until the user decides to exit, making it a user-friendly tool for cryptocurrency enthusiasts and investors.