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

PYTHON PROGRAMMING LAB

Prepared by:

Name of Student: Rafe Shaikh

Roll No: 150096723018

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 ofeach 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: Rafe Shaikh

Roll Number: 150096723018

Experiment No:1

Title:1.1 Write a program to compute Simple Interest.

Theory:

1.User Input:

The program utilises the input function to interact with the user and collect information. It prompts the user to enter the principal amount, rate of interest, and time.

2.Data Types and Variables:

It uses the float data type for storing decimal values, such as the principal amount, rate of interest, and time. Variables like principal_amount, rate_of_interest, time, and simple_interest are declared and assigned values.

3.Arithmetic Operations:

The program employs arithmetic operations for the Simple Interest calculation. Multiplication and division operations are used to apply the formula:

Simple Interest: Principal Amount x Rate of interest x time / 100

4.Formula Implementation:

The program translates the mathematical formula for Simple Interest into executable code, demonstrating how to express real-world mathematical concepts in a programming context.

Print Statement:

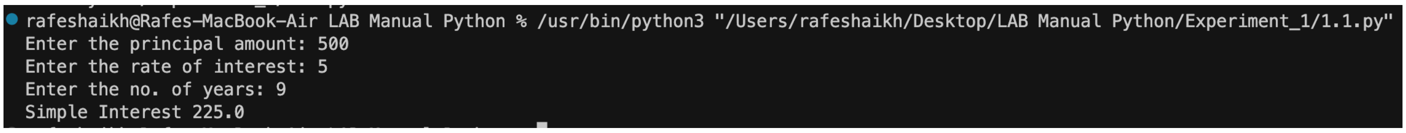
The print statement is used to display the computed Simple Interest to the user.

Code:

```
#Write a program to compute Simple Interest.
```

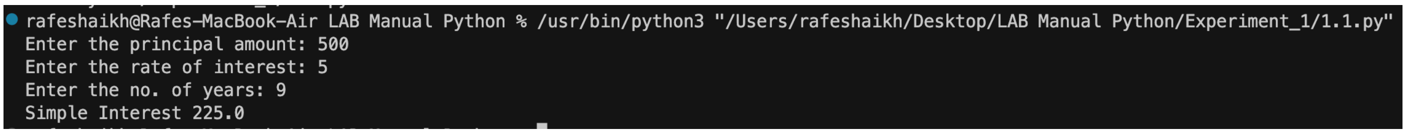
```python
principal_amount=float(input("Enter the principal amount: "))
rate_of_interest=float(input("Enter the rate of interest: "))
time=float(input("Enter the no. of years: "))
simple_interest=(principal_amount*rate_of_interest*time)/100
print("Simple Interest",simple_interest)
```

Output: (screenshot):

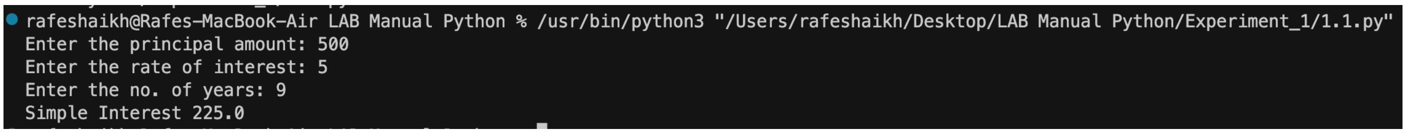


```
rafeshaikh@Rafes-MacBook-Air LAB Manual Python % /usr/bin/python3 "/Users/rafeshaikh/Desktop/LAB Manual Python/Experiment_1/1.1.py"
Enter the principal amount: 500
Enter the rate of interest: 5
Enter the no. of years: 9
Simple Interest 225.0
```

Test Case: Any two (screenshot)



```
rafeshaikh@Rafes-MacBook-Air LAB Manual Python % /usr/bin/python3 "/Users/rafeshaikh/Desktop/LAB Manual Python/Experiment_1/1.1.py"
Enter the principal amount: 500
Enter the rate of interest: 5
Enter the no. of years: 9
Simple Interest 225.0
```



```
rafeshaikh@Rafes-MacBook-Air LAB Manual Python % /usr/bin/python3 "/Users/rafeshaikh/Desktop/LAB Manual Python/Experiment_1/1.1.py"
Enter the principal amount: 500
Enter the rate of interest: 5
Enter the no. of years: 9
Simple Interest 225.0
```

Conclusion:The Simple Interest calculation program demonstrates fundamental concepts in programming, including user input, data types, variables, arithmetic operations, and the translation of mathematical formulas into code. It provides a practical example of how Python can be used for basic financial calculations. This program is a starting point for individuals learning programming and serves as a foundation for more complex financial applications. Understanding and implementing such programs enhance one's ability to solve real-world problems using programming skills.

Name of Student: Rafe Shaikh

Roll Number: 150096723018

Experiment No:1

Title:1.2 Write a program to perform arithmetic, Relational operators.

Theory:

Code:

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

Name of Student: Rafe Shaikh

Roll Number: 150096723018

Experiment No:1

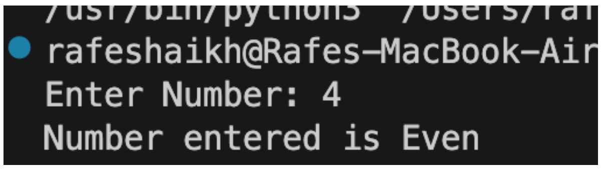
Title:1.3 Write a program to find whether a given no is even & odd.

Theory:The program checks whether a given number is even or odd. It uses the modulus operator (%) to determine if the remainder when dividing the input number by 2 is zero. If the remainder is zero, the number is considered even; otherwise, it is considered odd.

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

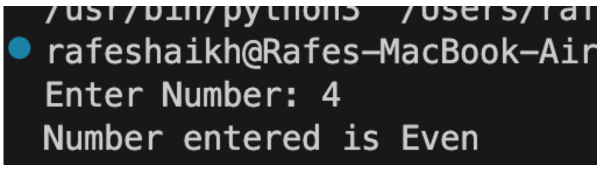
```python
n=int(input("Enter Number: "))
if n%2==0:
    print("Number entered is Even")
else:
    print("Number entered is Odd")
```

Output: (screenshot)

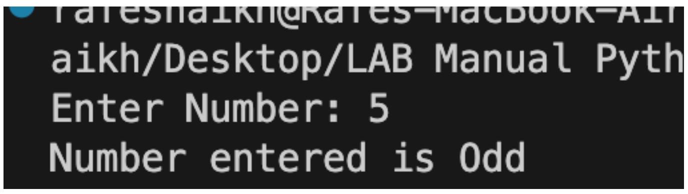


```
/usr/bin/python3 /Users/raf
● rafeshaikh@Rafes-MacBook-Air
  Enter Number: 4
  Number entered is Even
```

Test Case: Any two (screenshot)



```
/usr/bin/python3 /Users/raf
● rafeshaikh@Rafes-MacBook-Air
  Enter Number: 4
  Number entered is Even
```



```
● rafeshaikh@Rafes-MacBook-Air
  aikh/Desktop/LAB Manual Pyth
  Enter Number: 5
  Number entered is Odd
```

Conclusion: **The program effectively determines whether the entered number is even or odd and provides a clear output message. It showcases the use of a basic conditional statement (`if-else`) and the modulus operator to perform a common mathematical check. This type of program is useful for basic number classification and is a fundamental concept in introductory programming courses.**

Name of Student: Rafe Shaikh

Roll Number: 150096723018

Experiment No:1

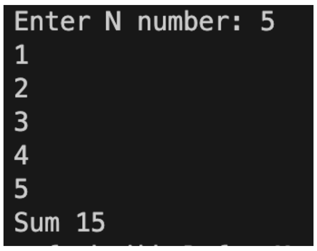
Title:1.4 Write a program to print first n natural number & their sum.

Theory:The program prints the first `n` natural numbers and calculates their sum. It uses a `for` loop to iterate through the range of numbers from 1 to `n` (inclusive). Inside the loop, each number is printed, and its value is added to the variable `sum`. Finally, the total sum is displayed.

Code:# Write a program to print first n natural number &amp; their sum.

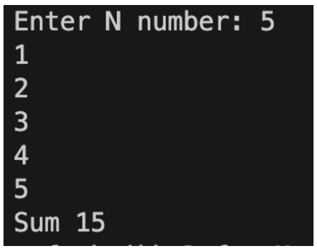
```
n=int(input("Enter N number: "))
sum=0
for i in range(1,n+1):
    print(i)
    sum=sum+i
print("Sum",sum)
```

Output: (screenshot)

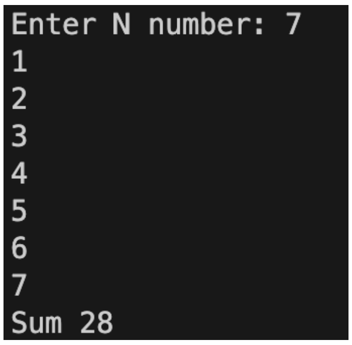


```
Enter N number: 5
1
2
3
4
5
Sum 15
```

Test Case: Any two (screenshot)



```
Enter N number: 5
1
2
3
4
5
Sum 15
```



```
Enter N number: 7
1
2
3
4
5
6
7
Sum 28
```

Conclusion:The program successfully prints the first `n` natural numbers and calculates their sum. It demonstrates the use of a `for` loop for iteration, the accumulation of values in a variable (`sum`), and the display of results using the `print` statement. This type of program is common in introductory programming to illustrate the basics of looping and variable manipulation. It provides a simple yet fundamental example for users learning Python programming.

Name of Student: Rafe Shaikh

Roll Number: 150096723018

Experiment No:1

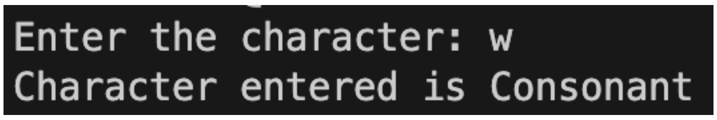
Title:1.5 Write a program to determine whether the character entered is a Vowel or not

Theory:The provided Python program determines whether a character entered by the user is a vowel or a consonant. It uses a list of vowels (`['a', 'e', 'i', 'o', 'u']`) and the `if-else` statement to check if the entered character is present in the list. The program also converts the user input to lowercase to handle both uppercase and lowercase characters uniformly.

Code:# Write a program to determine whether the character entered is a Vowel or not.

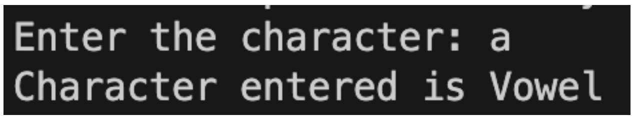
```
list=['a','e','i','o','u']
ask=input("Enter the character: ").lower()
if ask in list:
    print("Character entered is Vowel")
else:
    print("Character entered is Consonant")
```

Output: (screenshot)

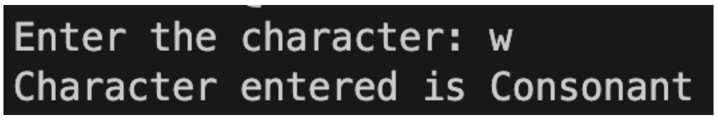


```
Enter the character: w
Character entered is Consonant
```

Test Case: Any two (screenshot)



```
Enter the character: a
Character entered is Vowel
```



```
Enter the character: w
Character entered is Consonant
```

Conclusion: **The program effectively determines whether the entered character is a vowel or a consonant. It demonstrates the use of a list to store specific characters, the `if-else` statement for conditional branching, and the conversion of user input to lowercase for case-insensitive comparison. This program is a simple example of character classification and is commonly used in introductory programming courses.**

Name of Student: Rafe Shaikh

Roll Number: 150096723018

Experiment No:1

Title: 1.6 Write a program to find whether given number is an Armstrong Number.

Theory:

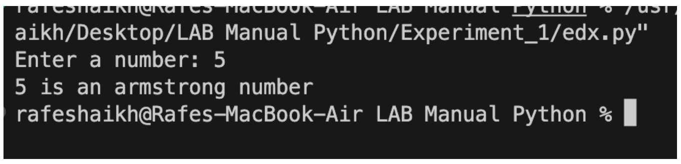
A positive number is called an Armstrong number of order n if abcd=a^n+b^n+c^n+d^n. Eg- 153(3 digits are there, therefore, order=3), therefore, 1^3+5^3+3^3=153. Therefore, 153 is an Armstrong number of order 3.

Code:

Code:

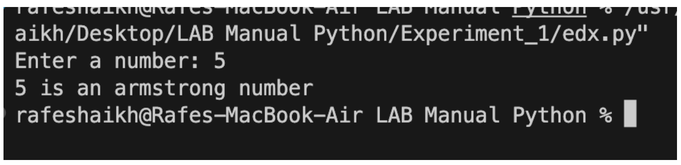
```
#WAP to check whether a given number is an armstrong number
a=int(input("Enter a number: "))
b=len(str(a))
c=a
sum=0
while c!=0:
    d=c%10
    sum+=d**b
    c//=10
if sum==a:
    print(a,"is an armstrong number")
else:
    print(a,"is not an armstrong number")
```

Output: (screenshot)



```
rafeshaikh@Rafes-MacBook-Air LAB Manual Python % /usr/
aikh/Desktop/LAB Manual Python/Experiment_1/edx.py"
Enter a number: 5
5 is an armstrong number
rafeshaikh@Rafes-MacBook-Air LAB Manual Python %
```

Test Case: Any two (screenshot)



```
rafeshaikh@Rafes-MacBook-Air LAB Manual Python % /usr/
aikh/Desktop/LAB Manual Python/Experiment_1/edx.py"
Enter a number: 5
5 is an armstrong number
rafeshaikh@Rafes-MacBook-Air LAB Manual Python %
```

Conclusion: Hence, calculating the order of the user given number(using len function), and adding the digits of the number raised to the power of the order(using while loop), and checking whether the sum is equal to the number(using relational operator ==) and printing the appropriate message(using if else statement).

Name of Student: Rafe Shaikh

Roll Number: 150096723018

Experiment No:1

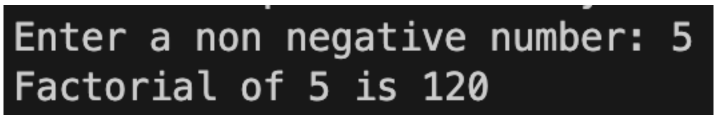
Title:1.7 Write a program using for loop to calculate factorial of a No.

Theory:The program calculates the factorial of a non-negative integer entered by the user using a `for` loop. The program includes conditional statements to handle cases where the entered number is negative or zero.

Code:

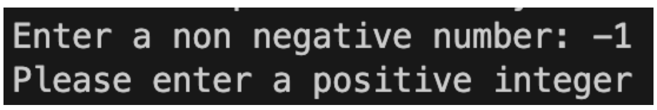
```
# Write a program using for loop calculate factorial of a number
n=int(input("Enter a non negative number: "))
factorial=1
if n<0:
    print("Please enter a positive integer")
elif n==0:
    print("The factorial of 0 is 1")
else:
    for i in range(1,n+1):
        factorial=factorial*i
    print(f"Factorial of {n} is {factorial}")
```

Output: (screenshot):

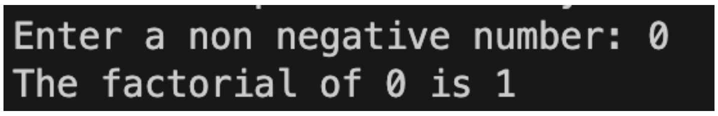


```
Enter a non negative number: 5
Factorial of 5 is 120
```

Test Case: Any two (screenshot):



```
Enter a non negative number: -1
Please enter a positive integer
```



```
Enter a non negative number: 0
The factorial of 0 is 1
```

Conclusion: **The program successfully calculates the factorial of a non-negative integer entered by the user using a `for` loop. It incorporates conditional statements to handle cases where the entered number is negative or zero. The program is a practical example of using loops for iterative calculations and includes proper handling for edge cases.**

Name of Student: Rafe Shaikh

Roll Number: 150096723018

Experiment No:1

Title:1.8 Write a program to print the following pattern

i)

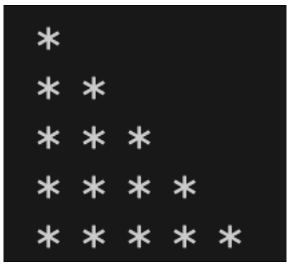
```
*
* *
* * *
* * * *
* * * * *
```

Theory:The provided Python code uses nested loops to print a pattern of asterisks (stars) in the shape of a right-angled triangle. The outer loop controls the number of rows, and the inner loop prints asterisks for each row.

Code:

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

Output: (screenshot)



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

Test Case: Any two (screenshot):

Conclusion:

The code generates a right-angled triangle pattern of asterisks. The number of asterisks in each row corresponds to the row number. For example, the first row has one asterisk, the second row has two asterisks, and so on, up to n rows. The nested loop structure efficiently handles the printing of the pattern, demonstrating a common use of nested loops in programming for creating various patterns.

Name of Student: Rafe Shaikh

Roll Number: 150096723018

Experiment No:1

Title:1.8 Write a program to print the following pattern

1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

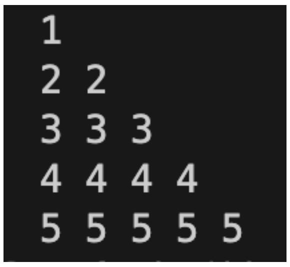
Theory:The provided Python code uses nested loops to print a pattern of numbers in the shape of a right-angled triangle. The outer loop controls the number of rows, and the inner loop prints numbers for each row.

Code:# Define the number of rows for the pattern

```
num_rows = 5
```

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

Output: (screenshot)



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

Test Case: Any two (screenshot)

Conclusion: The code generates a right-angled triangle pattern of numbers. The numbers in each row are repeated based on the row number. For example, the first row has one 1, the second row has two 2s, and so on, up to the specified number of rows (`num_rows`). The nested loop structure efficiently handles the printing of the pattern. This pattern demonstrates the repetition of numbers in a triangular form, which is a common use of nested loops for creating visual patterns in programming.

Name of Student: Rafe Shaikh

Roll Number: 150096723018

Experiment No:1

Title:1.8 Write a program to print the following pattern

iii)

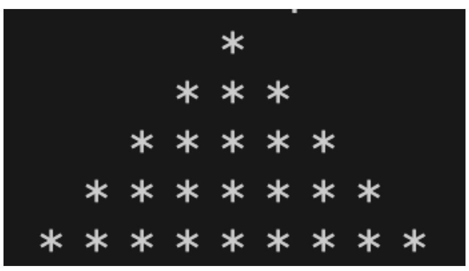
```
*
* * *
* * * * *
* * * * * * *
* * * * * * * * *
```

Theory:The code creates a pyramid pattern of asterisks (stars) using nested loops. The outer loop controls the number of rows, and the inner loops print leading spaces and asterisks for each row, resulting in a pyramid shape.

Code:

```
# Define the number of rows for the pyramid
num_rows = 5

# Outer loop to iterate over each row
for i in range(1, num_rows + 1):
    # Print leading spaces
    for j in range(num_rows - i):
        print(" ", end=" ")

    # Inner loop to print asterisks for each row
    for k in range(2*i - 1):
        print("*", end=" ")

    # Move to the next line after each row
    print()
```

Output: (screenshot):



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

Test Case: Any two (screenshot)

Conclusion:The provided Python code creates a pyramid pattern of asterisks (stars) using nested loops. The outer loop controls the number of rows, and the inner loops print leading spaces and asterisks for each row, resulting in a pyramid shape.

Name of Student: Rafe Shaikh

Roll Number: 18

Experiment No:2

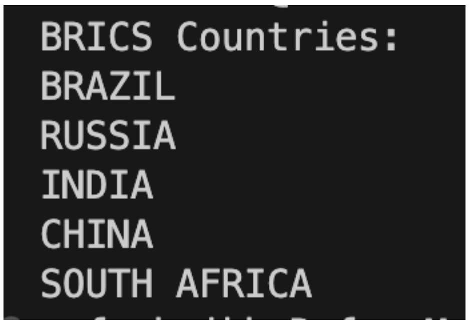
Title:2.1 Write a program that define the list of defines the list of define countries that are in BRICS.

Theory:The code prints a list of BRICS countries. It uses a `for` loop to iterate through the elements of the list and prints each country on a new line.

Code:

```
list=['BRAZIL','RUSSIA','INDIA','CHINA','SOUTH AFRICA']
print("BRICS Countries:")
for country in list:
    print(country)
```

Output: (screenshot):



```
BRICS Countries:
BRAZIL
RUSSIA
INDIA
CHINA
SOUTH AFRICA
```

Test Case: Any two (screenshot)

Conclusion: The code effectively prints the names of BRICS countries from the provided list. It demonstrates the use of a `for` loop to iterate through the elements of a list and the `print` statement to display the information. This type of program is common in handling lists and can be used for displaying various types of information stored in a collection.

Name of Student: Rafe Shaikh

Roll Number: 18

Experiment No:2

Title:2.2 Write a program to traverse a list in reverse order.

1.By using Reverse method.

2.By using slicing

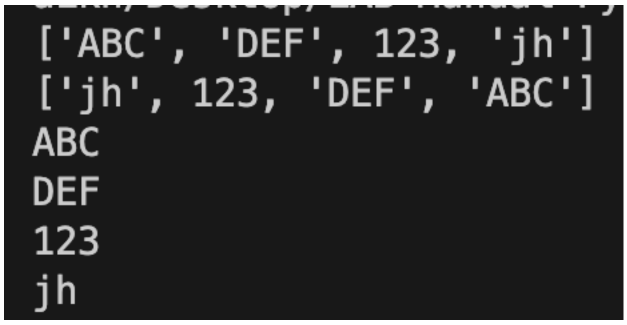
Theory: The `reverse()` method is used to reverse the order of elements in a list in-place.

Slicing with `[::-1]` is a common way to iterate through a list in reverse order.

Code:

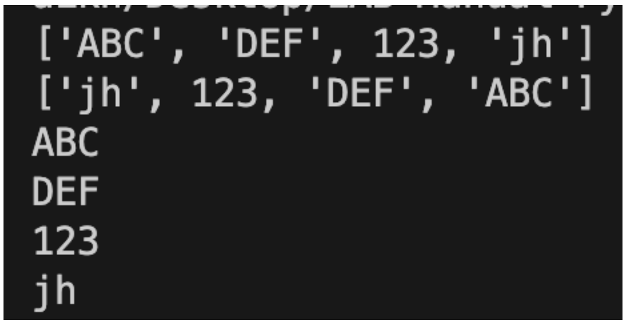
```
list=['ABC','DEF',123,'jh']
print(list)
list.reverse()
print(list)
for i in list[::-1]:
    print(i)
```

Output: (screenshot):



```
['ABC', 'DEF', 123, 'jh']
['jh', 123, 'DEF', 'ABC']
ABC
DEF
123
jh
```

Test Case: Any two (screenshot)



```
['ABC', 'DEF', 123, 'jh']
['jh', 123, 'DEF', 'ABC']
ABC
DEF
123
jh
```

Conclusion: The original list is printed.

- The list is reversed in-place using the `reverse()` method and printed again.
- The elements of the reversed list are printed individually in reverse order using a for loop with slicing.

Name of Student: Rafe Shaikh

Roll Number: **18**

Experiment No:2

Title:2.3 Write a program that scans the email address and forms a tuple of username and domain.

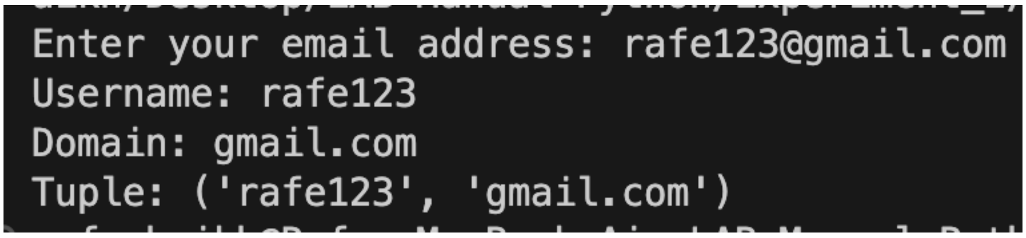
Theory: Using split() function with @ as a parameter to split the email address given by user and store it in a tuple. Then print the first element of the tuple as username and the second element from the same tuple as domain.

Code:

```
# 2.3 Write a program that scans the email address and forms a 
tuple of username
# and domain.

# Get user input for the email address
email_address = input("Enter your email address: ")

# Split the email address into username and domain
username, domain = email_address.split("@")

# Form a tuple with username and domain
```

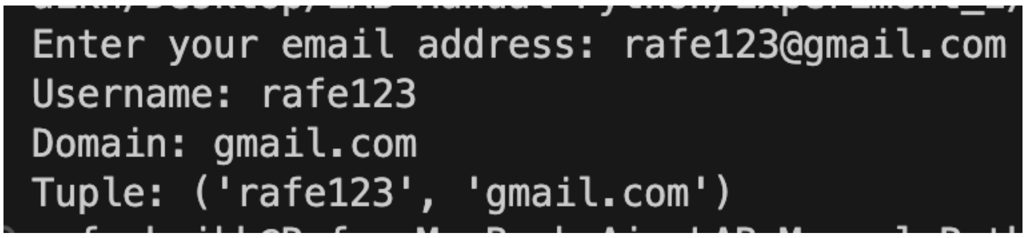
```
email_tuple = (username, domain)

# Print the tuple
print("Username:", email_tuple[0])
print("Domain:", email_tuple[1])
print("Tuple:", email_tuple)
```

Output: (screenshot)



```
Enter your email address: rafe123@gmail.com
Username: rafe123
Domain: gmail.com
Tuple: ('rafe123', 'gmail.com')
```

Test Case: Any two (screenshot)



```
Enter your email address: rafe123@gmail.com
Username: rafe123
Domain: gmail.com
Tuple: ('rafe123', 'gmail.com')
```

Conclusion: Hence, using split() function to split the email address in username and domain and printing them after storing them in a tuple.

Name of Student: Rafe Shaikh

Roll Number: 18

Experiment No:2

Title: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]

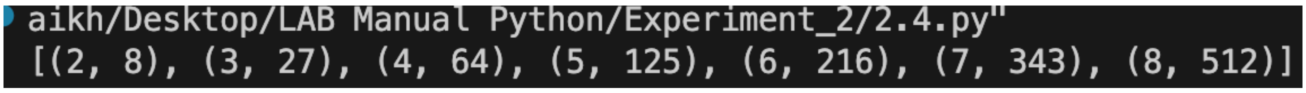
Theory:

Using list comprehension, make another list with tuples as its elements. Each tuple would have the number from original list and it's cube using arithmetic operator(**).

Code:# i/p: c= [2,3,4,5,6,7,8,9]

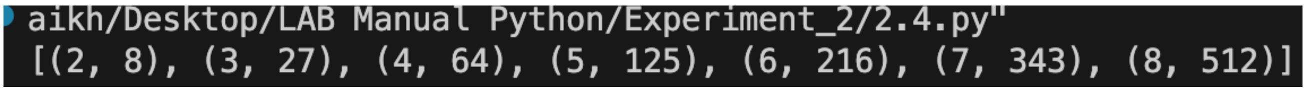
```
c=[2,3,4,5,6,7,8]
tuple_list=[]
for i in c:
    tuple_list.append((i,i**3))

print(tuple_list)
```

Output: (screenshot)



```
aikh/Desktop/LAB Manual Python/Experiment_2/2.4.py"
[(2, 8), (3, 27), (4, 64), (5, 125), (6, 216), (7, 343), (8, 512)]
```

Test Case: Any two (screenshot)



```
aikh/Desktop/LAB Manual Python/Experiment_2/2.4.py"
[(2, 8), (3, 27), (4, 64), (5, 125), (6, 216), (7, 343), (8, 512)]
```

Conclusion: Hence, using list comprehension to make a list of tuples containing the values and their cube using ** operator from the original list.

Name of Student:Rafe Shaikh

Roll Number: 18

Experiment No:2

Title: 2.5 Write a program to compare two dictionaries in Python?

(By using == operator)

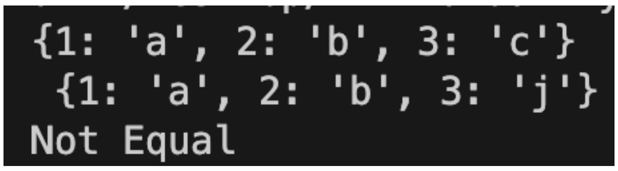
Theory:

Dictionary is a datatype in Python which stores information as key-value pairs, where keys are unique and are used to distinguish between values. Using relational operator(==) to check whether two dictionaries have same key-value pairs or not.

Code:

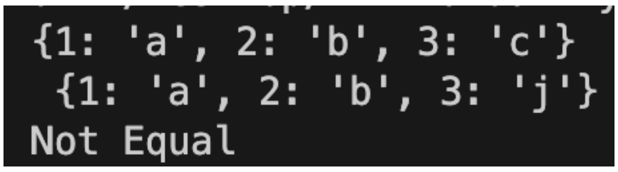
```python
dict1={1:'a',2:'b',3:'c'}
dict2={1:'a',2:'b',3:'j'}
print(dict1,"\n",dict2)
if dict1==dict2:
    print("Equal")
else:
    print("Not Equal")
```

Output: (screenshot):

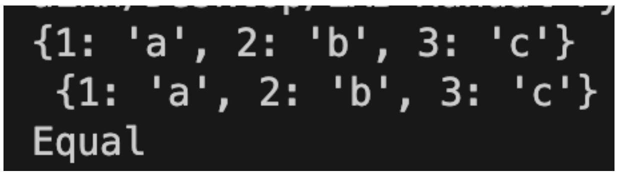


```
{1: 'a', 2: 'b', 3: 'c'}
 {1: 'a', 2: 'b', 3: 'j'}
Not Equal
```

Test Case: Any two (screenshot)



```
{1: 'a', 2: 'b', 3: 'c'}
 {1: 'a', 2: 'b', 3: 'j'}
Not Equal
```



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

Conclusion: Hence, using == operator to check whether two dictionaries have same key-value pairs or not and printing appropriate messages using if else statements.

Name of Student:Rafe Shaikh

Roll Number: 18

Experiment No:2

Title:2.6 Write a program that creates dictionary of cube of odd numbers in the range.

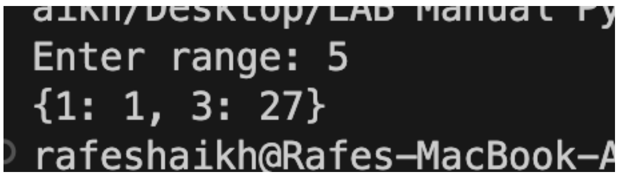
Theory:

Using for loop to iterate over every number from 1 till range given by user and using if statement to check whether a number is odd or even using modulus operator(%) and making the number as key and its cube as a value, and printing the dictionary in the end.

Code:

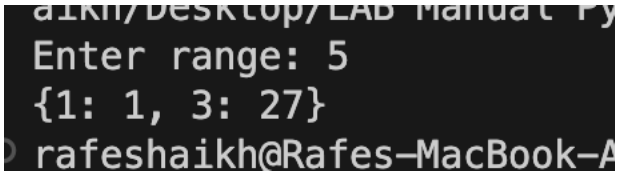
```
#WAP that creates dictionary of cube of odd numbers in the range.
a=int(input("Enter range: "))
b={}
for i in range(1,a):
    if i%2!=0:
        b[i]=i**3
print(b)
```

Output: (screenshot)

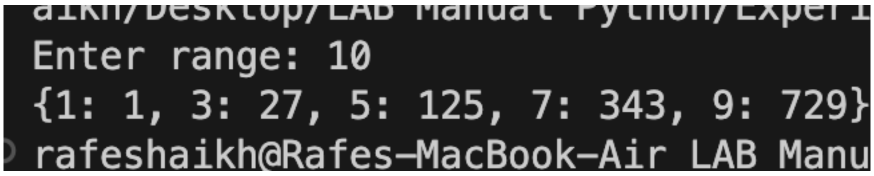


```
aikh/Desktop/LAB Manual Py
Enter range: 5
{1: 1, 3: 27}
rafeshaikh@Rafes-MacBook-A
```

Test Case: Any two (screenshot)



```
aikh/Desktop/LAB Manual Py
Enter range: 5
{1: 1, 3: 27}
rafeshaikh@Rafes-MacBook-A
```



```
aikh/Desktop/LAB Manual Python/Experi
Enter range: 10
{1: 1, 3: 27, 5: 125, 7: 343, 9: 729}
rafeshaikh@Rafes-MacBook-Air LAB Manu
```

Conclusion:

Hence, using for loop and if statement to create a dictionary of odd numbers as keys and their cube as values and printing the dictionary.

Name of Student:Rafe Shaikh

Roll Number: 18

Experiment No:2

Title: 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.

Theory:

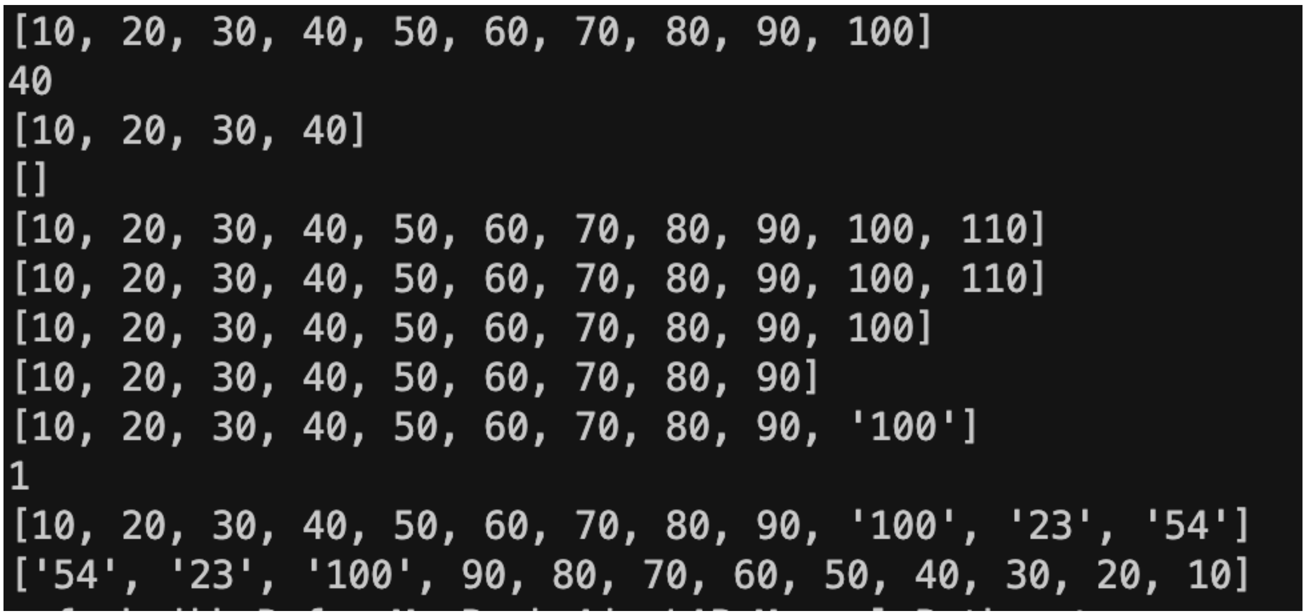
Using index slicing to print specific elements, predefined functions such as append() to add an element in the list, sort() to sort the list in ascending or descending order, pop() to remove and element in the list, insert() to add an element in the list at a specific index, remove() to remove a specific element from the list, for loop to check the occurrence of an element, extend() to add multiple elements in the list, reverse() to reverse the list.

Code:

```
# 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
print(a)

# ii. Print 4th element of list
print(a[3])

# iii. Print list from0th to 4th index.
print(a[0:4])

# iv. Print list -7th to 3rd element
print(a[-7:2])

# v. Appending an element to list.
a.append(110)
print(a)

# vi. Sorting the element of list.
a.sort()
print(a)

# vii. Popping an element.
a.pop()
print(a)

# viii. Removing Specified element.
a.remove(100)
print(a)
```

```
# ix. Entering an element at specified index.
a.insert(9,'100')
print(a)

# x. Counting the occurrence of a specified element.
x=a.count('100')
print(x)

# xi. Extending list.
b=['23','54']
a.extend(b)
print(a)

# xii. Reversing the list.
a.reverse()
print(a)
```

Output: (screenshot)

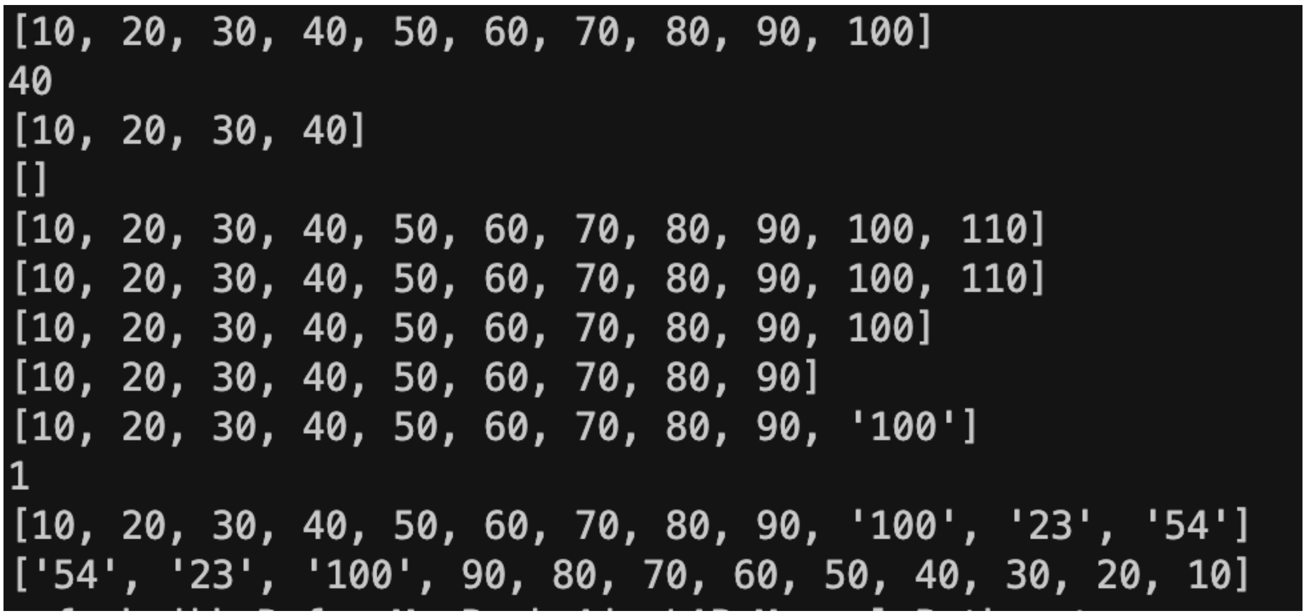


```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
40
[10, 20, 30, 40]
[]
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[10, 20, 30, 40, 50, 60, 70, 80, 90]
[10, 20, 30, 40, 50, 60, 70, 80, 90, '100']
1
[10, 20, 30, 40, 50, 60, 70, 80, 90, '100', '23', '54']
['54', '23', '100', 90, 80, 70, 60, 50, 40, 30, 20, 10]
```

Test Case: Any two (screenshot)

Conclusion:

Hence, using for loop, slicing, and various predefined list functions to add, remove, sort and reverse a list.



```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
40
[10, 20, 30, 40]
[]
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[10, 20, 30, 40, 50, 60, 70, 80, 90]
[10, 20, 30, 40, 50, 60, 70, 80, 90, '100']
1
[10, 20, 30, 40, 50, 60, 70, 80, 90, '100', '23', '54']
['54', '23', '100', 90, 80, 70, 60, 50, 40, 30, 20, 10]
```

Name of Student:Rafe Shaikh

Roll Number: 18

Experiment No:3

Title: 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 ()

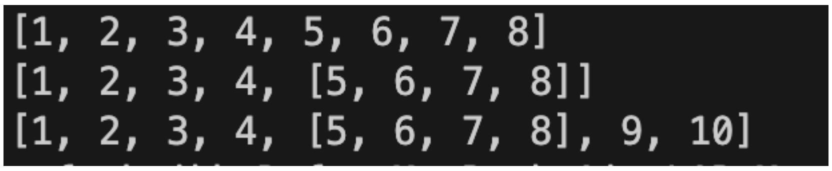
Theory: + operator is used to add a list to another list. Append() is used to add an element at the end of the list. Extend() is used to add multiple elements at the end of the list.

Code:

```
# Write a program to extend a list in python by using given
approach.
list1=[1,2,3,4]
list2=[5,6,7,8]
extend=[9,10]

# i. By using + operator.

plus=list1+list2
print(plus)
```

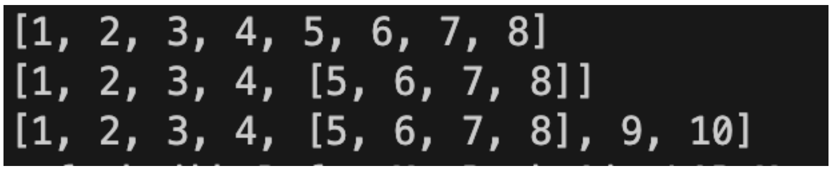
```
# ii. By using Append ()

list1.append(list2)
print(list1)

# iii. By using extend ()

list1.extend(extend)
print(list1)
```

Output: (screenshot)



```
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, [5, 6, 7, 8]]
[1, 2, 3, 4, [5, 6, 7, 8], 9, 10]
```

Test Case: Any two (screenshot)



```
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, [5, 6, 7, 8]]
[1, 2, 3, 4, [5, 6, 7, 8], 9, 10]
```

Conclusion:

Hence, using for loop and if statement to create a dictionary of odd numbers as keys and their cube as values and printing the dictionary.

Name of Student:Rafe Shaikh

Roll Number: 18

Experiment No:3

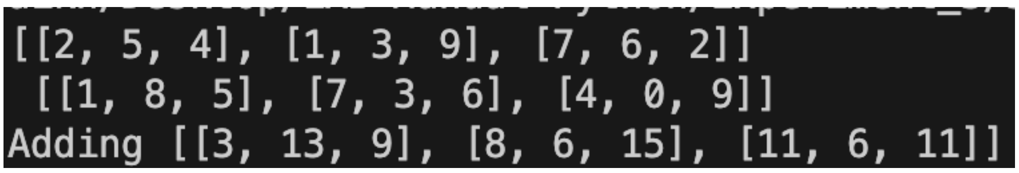
Title: 3.2 Write a program to add two matrices.

Theory: A matrix is a nested list in Python(a list of lists). It consists of two lists which have three lists inside each of them. Using nested for loop, one for each of the two lists, add elements from the two matrices and store the sum in another matrix of same dimension and print it.

Code:

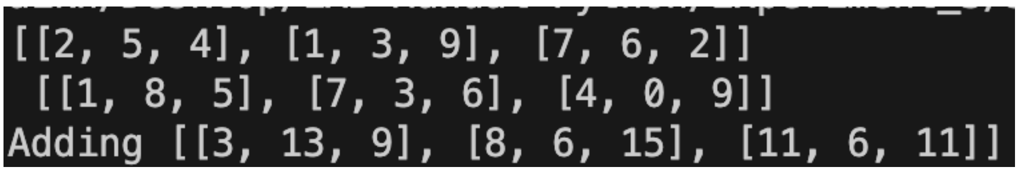
```python
a=[[2,5,4],[1,3,9],[7,6,2]]
b=[[1,8,5],[7,3,6],[4,0,9]]
c=[[0,0,0],[0,0,0],[0,0,0]]
for i in range(len(a)):
    for j in range(len(b)):
        c[i][j]=a[i][j]+b[i][j]
print(c)
```

Output: (screenshot)



```
[[2, 5, 4], [1, 3, 9], [7, 6, 2]]
 [[1, 8, 5], [7, 3, 6], [4, 0, 9]]
Adding [[3, 13, 9], [8, 6, 15], [11, 6, 11]]
```

Test Case: Any two (screenshot)



```
[[2, 5, 4], [1, 3, 9], [7, 6, 2]]
 [[1, 8, 5], [7, 3, 6], [4, 0, 9]]
Adding [[3, 13, 9], [8, 6, 15], [11, 6, 11]]
```

Conclusion:

Hence, using nested for loop, add the elements of two nested lists and store the sum in another nested list.

Name of Student:Rafe Shaikh

Roll Number: 18

Experiment No:3

Title: 3.3 Write a Python function that takes a list and returns a new list with distinct elements from the first list.

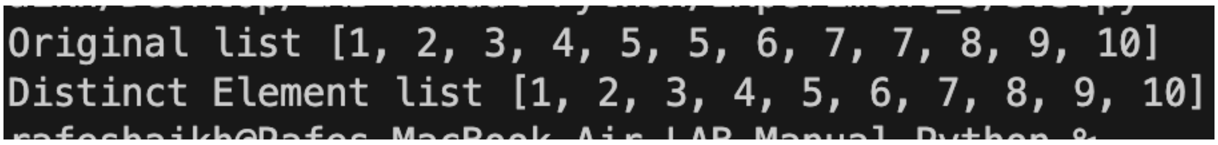
Theory:

Making a user defined function distin() taking a user list as a parameter and then making an empty list and using a for loop to iterate through the user list and using membership operator(in) to check whether the element is present in the empty list too. If it is not present, append it in the list and check the next element for the same.

Code:

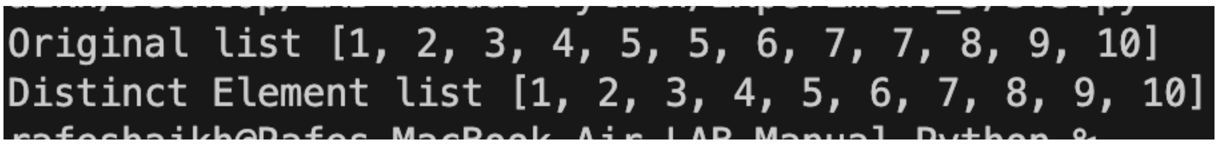
```python
list1=[1,2,3,4,5,5,6,7,7,8,9,10]
list2=[]

for i in list1:
    if i not in list2:
        list2.append(i)

print("Original list",list1)
print("Distinct Element list",list2)
```

Output: (screenshot)



```
Original list [1, 2, 3, 4, 5, 5, 6, 7, 7, 8, 9, 10]
Distinct Element list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

TestCase: Any two (screenshot)



```
Original list [1, 2, 3, 4, 5, 5, 6, 7, 7, 8, 9, 10]
Distinct Element list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Conclusion: Hence, by making a function, using nested for loop to check whether each element in the list is only present once and appending it to another list and then printing the other list at the end.

Name of Student:Rafe Shaikh

Roll Number: 18

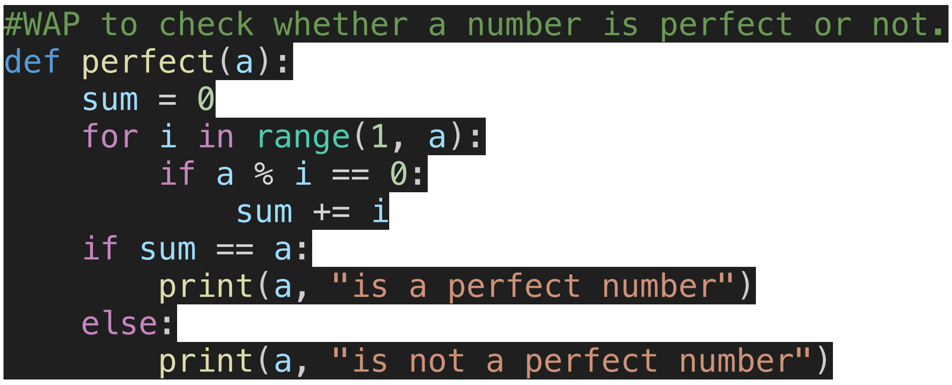
Experiment No:3

Title: 3.4 Write a program to Check whether a number is perfect or not.

Theory:

A number is a perfect number if the sum of its divisors(excluding the number itself) is equal to the number itself. Eg- 6 is a perfect number as divisors of 6 are 1,2,3. Sum- 1+2+3=6 which is equal to the number itself.

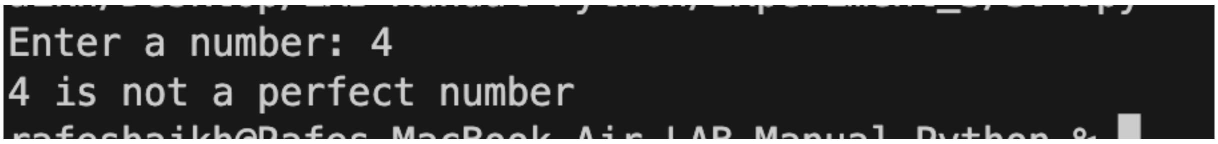
Code:



```python
#WAP to check whether a number is perfect or not.
def perfect(a):
    sum = 0
    for i in range(1, a):
        if a % i == 0:
            sum += i
    if sum == a:
        print(a, "is a perfect number")
    else:
        print(a, "is not a perfect number")
```

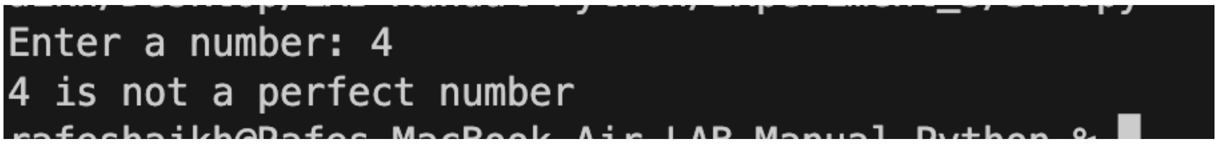
```
a = int(input("Enter a number: "))
perfect(a)
```

Output: (screenshot):

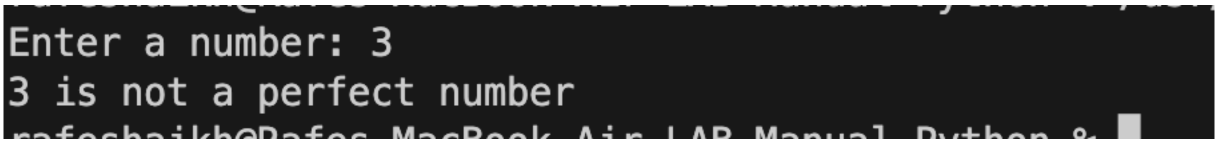


```
Enter a number: 4
4 is not a perfect number
```

Test Case: Any two (screenshot)



```
Enter a number: 4
4 is not a perfect number
```



```
Enter a number: 3
3 is not a perfect number
```

Conclusion:

Hence, using a for loop to iterate over each number from 1 to number itself(exclusive) and checking whether it is a divisor of the number using if else statement and adding it in sum variable and checking afterwards if the sum is equal to the number and printing the appropriate message using if else statement.

Name of Student:Rafe Shaikh

Roll Number: 18

Experiment No:3

Title: 3.5 Write a Python function that accepts a string and counts the number of upper-case and lower-case letters.

Theory:

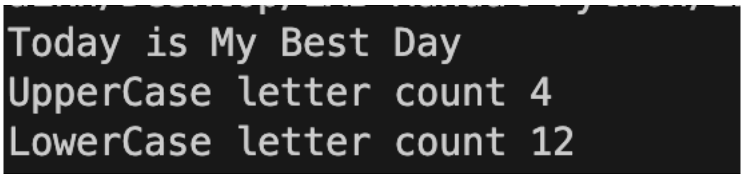
Using a for loop to iterate each character in the string and if elif statement and predefined functions isupper() and islower() to check whether a character is lowercase or uppercase and incrementing the value of the respective uppercase or lowercase counter by 1.

Code:

```
string_test= 'Today is My Best Day'
upper_count=0
lower_count=0

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

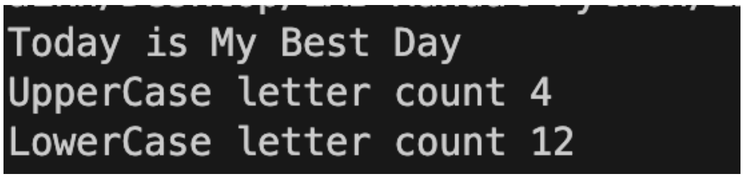
```
print("UpperCase letter count",upper_count)
print("LowerCase letter count",lower_count)
```

Output: (screenshot):



```
Today is My Best Day
UpperCase letter count 4
LowerCase letter count 12
```

Test Case: Any two (screenshot)



```
Today is My Best Day
UpperCase letter count 4
LowerCase letter count 12
```

Conclusion:

Hence, using a for loop to iterate over each character in the string and using isupper() and islower() to check the characters and increment the counter variables value by 1 using if elif statement.

Name of Student:Rafe Shaikh

Roll Number: 18

Experiment No:4

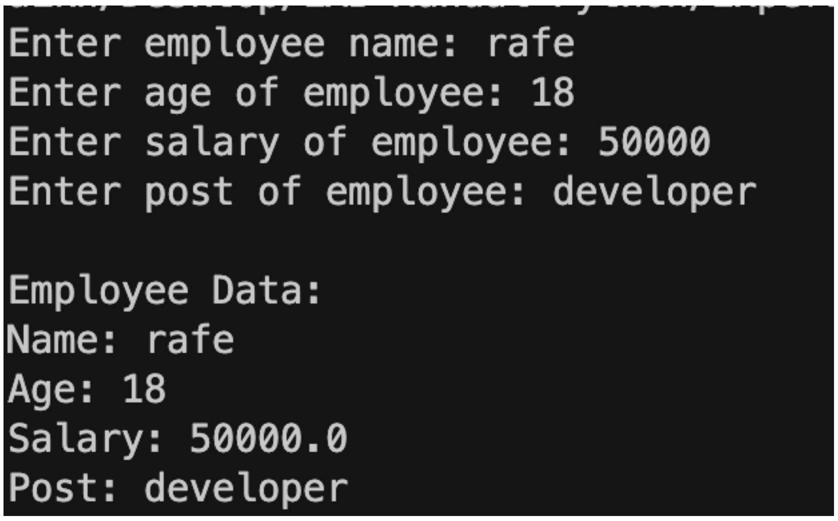
Title: 4.1 Write a program to Create Employee Class & add methods to get employee details & print.

Theory: A class is a blueprint for objects. It contains attributes and methods which the objects can access and use. An object is an instance of a class. Each class has its own copy of attributes and share the methods of the class.

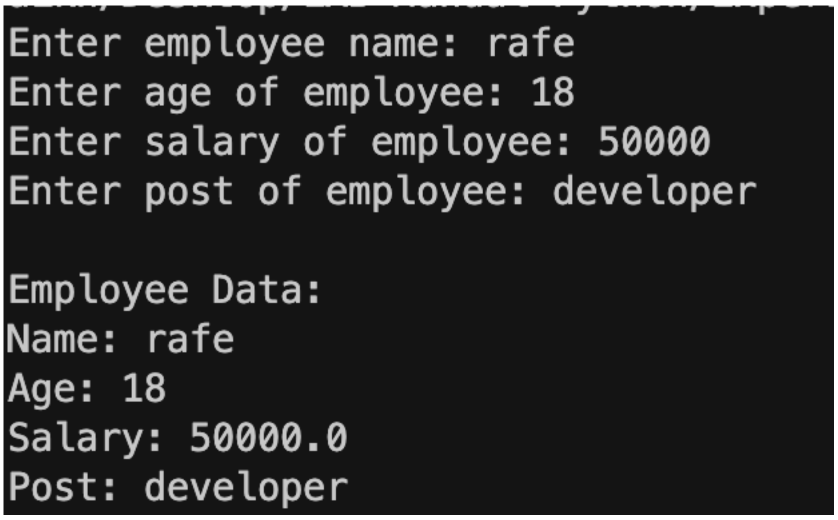
Code:

```
#WAP to Create Employee Class & add methods to get employee details
& print.
class Employee:
    __name=""
    __age=0
    __salary=0
    _post=""
    def setData(self):
        name=input("Enter employee name: ")
        self.__name=name
        age=int(input("Enter age of employee: "))
        self.__age=age
        salary=float(input("Enter salary of employee: "))
        self.__salary=salary
        post=input("Enter post of employee: ")
        self.__post=post
    def getData(self):
        print("\nEmployee Data:")
        print("Name:", self.__name)
        print("Age:", self.__age)
        print("Salary:", self.__salary)
        print("Post:", self.__post)
a=Employee()
a.setData()
a.getData()
```

Output: (screenshot)


```
Enter employee name: rafe
Enter age of employee: 18
Enter salary of employee: 50000
Enter post of employee: developer

Employee Data:
Name: rafe
Age: 18
Salary: 50000.0
Post: developer
```

Test Case: Any two (screenshot)



```
Enter employee name: rafe
Enter age of employee: 18
Enter salary of employee: 50000
Enter post of employee: developer

Employee Data:
Name: rafe
Age: 18
Salary: 50000.0
Post: developer
```

Conclusion:

Hence, using get and set data functions to take values from the user and then assigning the attributes of the object these values and printing them to the user.

Name of Student:Rafe Shaikh

Roll Number: 18

Experiment No:4

Title:

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

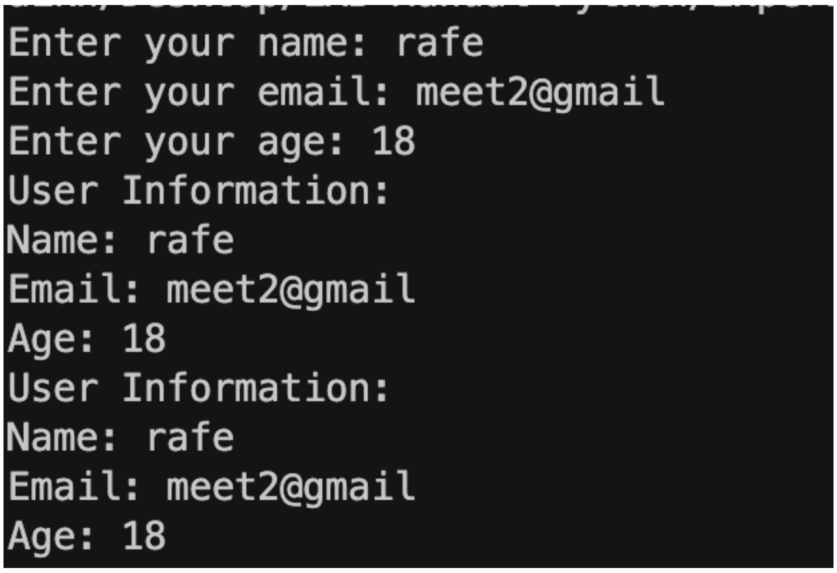
Theory:

While defining a function, we use *args and **kwargs as parameters when we don't know how many arguments the user will pass during function call. *args will take a tuple of positional arguments as parameter and work on it. **kwargs will take a dictionary of keyword arguments as parameter and assign the keys to keywords in arguments and the values as the values passed by the user in the arguments.

Code:

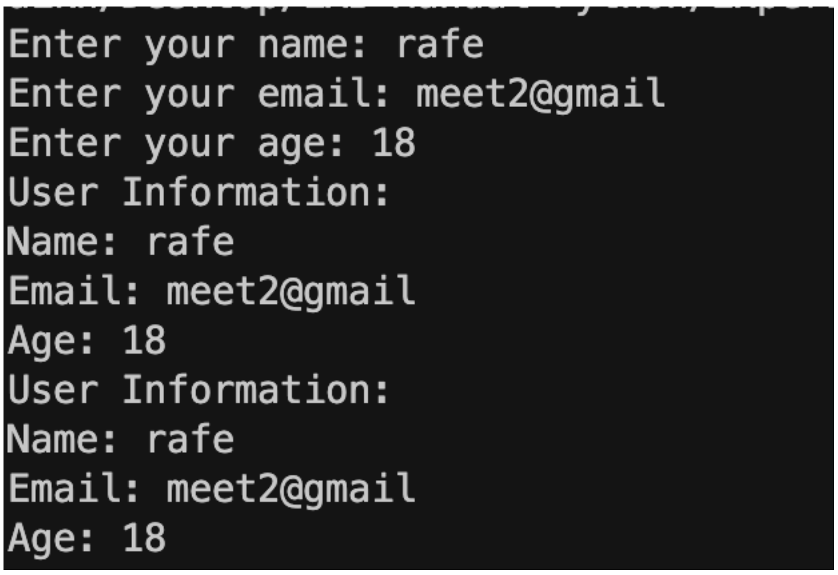
```python
# Define a function named get_user_info that takes both positional 
and keyword arguments
def get_user_info(*args, **kwargs):
    # Using positional arguments (args) for name, email, and age
    if len(args) >= 3:
        name = args[0]
        email = args[1]
        age = args[2]
    else:
        # If there are not enough positional arguments, use keyword 
arguments (kwargs)
        name = kwargs.get('name', '')
        email = kwargs.get('email', '')
        age = kwargs.get('age', 0)

    # Print user information
    print("User Information:")
    print(f"Name: {name}")
    print(f"Email: {email}")
    print(f"Age: {age}")

# Taking input from the user
user_name = input("Enter your name: ")
user_email = input("Enter your email: ")
user_age = int(input("Enter your age: "))

# Calling the function with positional arguments
get_user_info(user_name, user_email, user_age)

# Calling the function with keyword arguments
get_user_info(name=user_name, email=user_email, age=user_age)
```

Output: (screenshot):



```
Enter your name: rafe
Enter your email: meet2@gmail
Enter your age: 18
User Information:
Name: rafe
Email: meet2@gmail
Age: 18
User Information:
Name: rafe
Email: meet2@gmail
Age: 18
```

Test Case: Any two (screenshot)



```
Enter your name: rafe
Enter your email: meet2@gmail
Enter your age: 18
User Information:
Name: rafe
Email: meet2@gmail
Age: 18
User Information:
Name: rafe
Email: meet2@gmail
Age: 18
```

Conclusion: Hence, using dynamic argument passing(*args and **kwargs) to take multiple arguments from the user in a combination of positional and keyword arguments and printing them using a user defined function.

Name of Student:Rafe Shaikh

Roll Number: 18

Experiment No:4

Title: 4.3 Write a program to admit the students in the different Departments(pgdm/btech) and count the students. (Class, Object and Constructor).

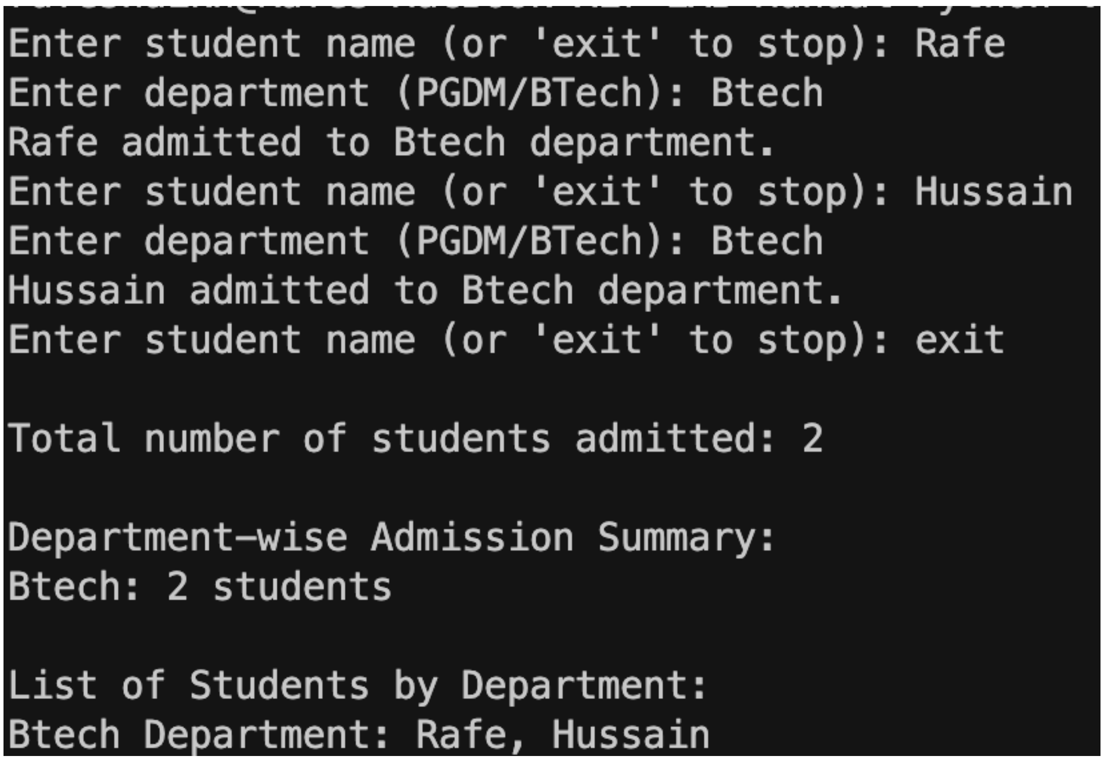
Theory: The provided Python program employs object-oriented programming concepts, including classes, objects, and constructors, to facilitate student admissions. The `Student` class encapsulates individual student attributes, and the `AdmissionManager` class manages the admission process, utilizing methods for admitting students, counting their numbers, and providing department-wise summaries. A while loop facilitates interactive user input for admitting students, and the program effectively employs data structures like lists and dictionaries to organize student data. Overall, the code demonstrates the encapsulation of behavior and data, enhancing modularity and readability in managing student admissions.

Code:

```python
# 4.3 Write a program to admit the students in the different
# Departments(pgdm/btech)and count the students. (Class, Object and
Constructor).

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

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

    def admit_student(self, name, department):
        student = Student(name, department)
        self.students.append(student)
        print(f"{name} admitted to {department} department.")

    def count_students(self):
        total_students = len(self.students)
        print(f"\nTotal number of students admitted:
{total_students}")

        # Counting students in each department
        department_counts = {}
        for student in self.students:
            department_counts[student.department] =
department_counts.get(student.department, 0) + 1

        # Displaying the count for each department
        print("\nDepartment-wise Admission Summary:")
        for department, count in department_counts.items():
            print(f"{department}: {count} students")

        # Displaying the list of students for each department
        print("\nList of Students by Department:")
        for department in set([student.department for student in
self.students]):
            students_in_department = [student.name for student in
self.students if student.department == department]
            print(f"{department} Department: {',
'.join(students_in_department)}")
```

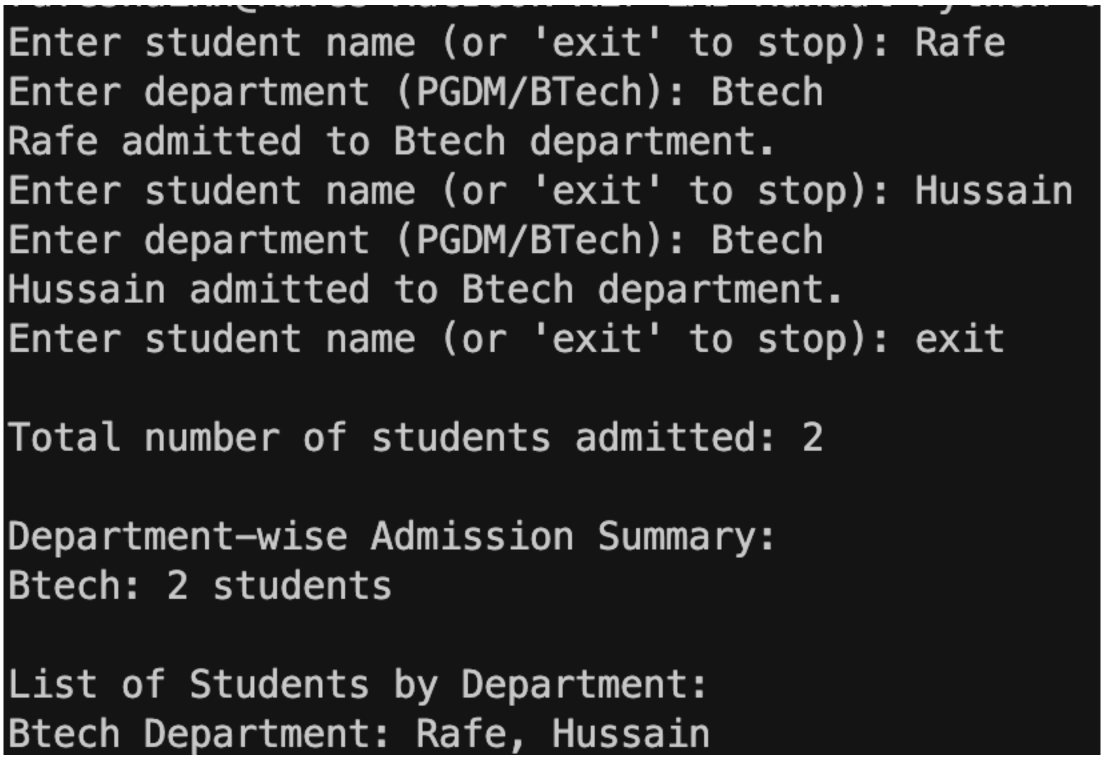
```python
# Creating an instance of the AdmissionManager class
admission_manager = AdmissionManager()

# Taking user input for admission
while True:
    user_name = input("Enter student name (or 'exit' to stop): ")
    if user_name.lower() == 'exit':
        break
    user_department = input("Enter department (PGDM/BTech): ")
    admission_manager.admit_student(user_name, user_department)

# Displaying the total number of students and department-wise summary
admission_manager.count_students()
```

Output: (screenshot):



```
Enter student name (or 'exit' to stop): Rafe
Enter department (PGDM/BTech): Btech
Rafe admitted to Btech department.
Enter student name (or 'exit' to stop): Hussain
Enter department (PGDM/BTech): Btech
Hussain admitted to Btech department.
Enter student name (or 'exit' to stop): exit

Total number of students admitted: 2

Department-wise Admission Summary:
Btech: 2 students

List of Students by Department:
Btech Department: Rafe, Hussain
```

Test Case: Any two (screenshot)



```
Enter student name (or 'exit' to stop): Rafe
Enter department (PGDM/BTech): Btech
Rafe admitted to Btech department.
Enter student name (or 'exit' to stop): Hussain
Enter department (PGDM/BTech): Btech
Hussain admitted to Btech department.
Enter student name (or 'exit' to stop): exit

Total number of students admitted: 2

Department-wise Admission Summary:
Btech: 2 students

List of Students by Department:
Btech Department: Rafe, Hussain
```

Conclusion:

In conclusion, the Python program effectively utilizes object-oriented principles to model student admissions, employing classes and objects to encapsulate student attributes and admission management functionalities. The constructor initializes necessary data structures, and user input is facilitated through a while loop. The use of methods and data structures enhances modularity and readability, allowing for the organized admission of students and providing insightful summaries. The program showcases a structured and interactive approach to student admission, promoting maintainability and ease of understanding in a real-world scenario.

Name of Student:Rafe Shaikh

Roll Number: 18

Experiment No:4

Title:

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.

Theory:

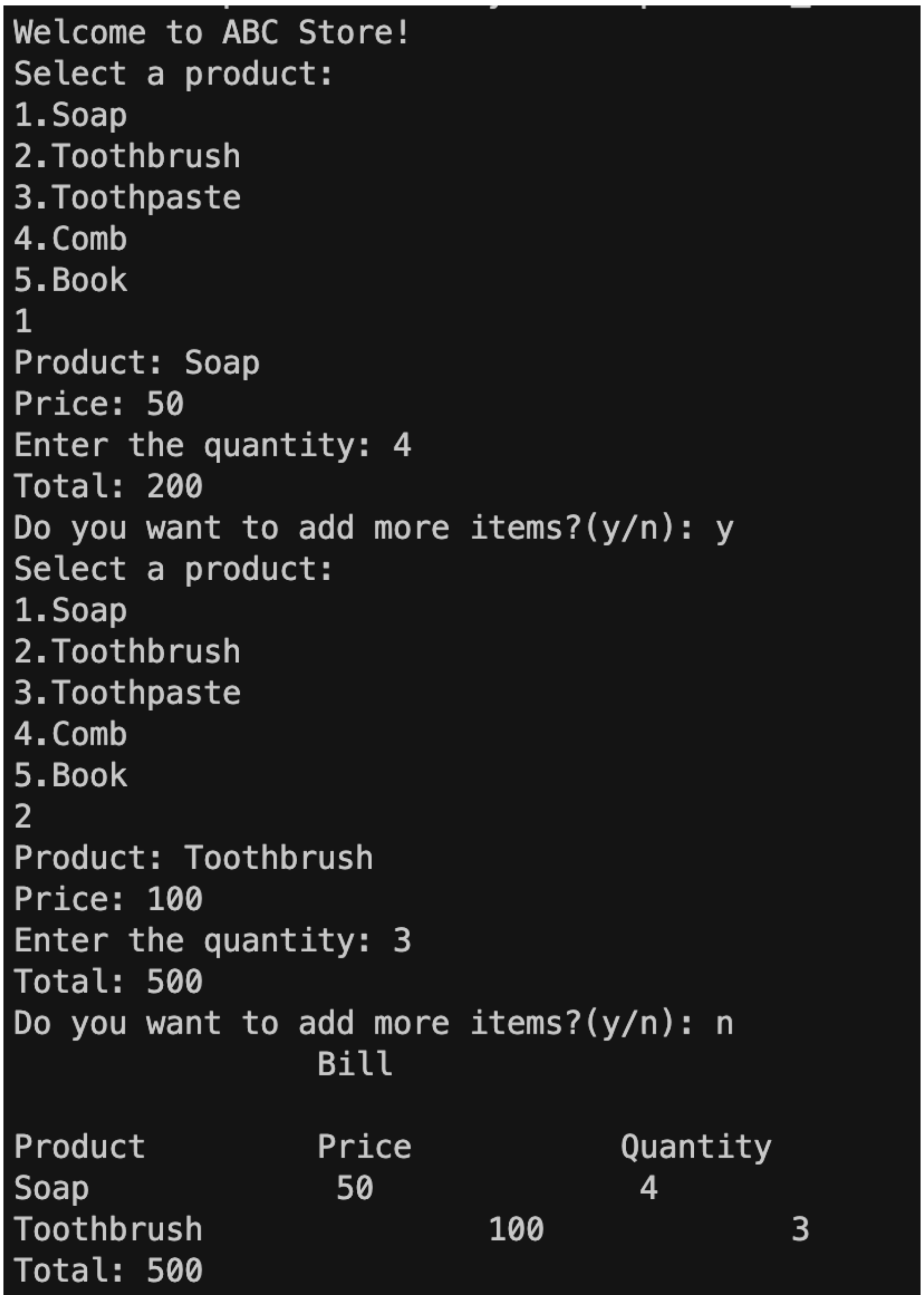
Using constructor to print a menu of items to the user and getting quantity of the product and printing the bill with total amount at the end.

Code:

```
#WAP 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.
class Store:
    __itemCode=0
    __price=0
    __total=0
    product=[]
    quantity=[]
    price=[]
    def __init__(self):
        print("Welcome to ABC Store!")
        while True:
            a=int(input("Select a product:
\n1.Soap\n2.Toothbrush\n3.Toothpaste\n4.Comb\n5.Book\n"))
            c=("Soap","Toothbrush","Toothpaste","Comb","Book")
            if a==1:
                self.__itemCode=1
                self.__price=50
            elif a==2:
                self.__itemCode=2
                self.__price=100
            elif a==3:
                self.__itemCode=3
                self.__price=200
            elif a==4:
                self.__itemCode=4
                self.__price=150
            elif a==5:
                self.__itemCode=5
                self.__price=500
            print("Product:",c[a-1],"\nPrice:",self.__price)
            self.product.append(c[a-1])
            self.price.append(self.__price)
            b=int(input("Enter the quantity: "))
            while b<=0:
                print("Invalid quantity")
                b=int(input("Enter the quantity: "))
            self.quantity.append(b)
            self.__total+=self.__price*b
            print("Total:",self.__total)
            d=input("Do you want to add more items?(y/n): ")
            if d.lower()=="n":
                break
        print("\t\tBill\n\nProduct\t\tPrice\t\tQuantity")
        for i in range(len(self.product)):

print(self.product[i],"\t\t",self.price[i],"\t\t",self.quantity[i],
"")
        print("Total:",self.__total)
obj=Store()
```

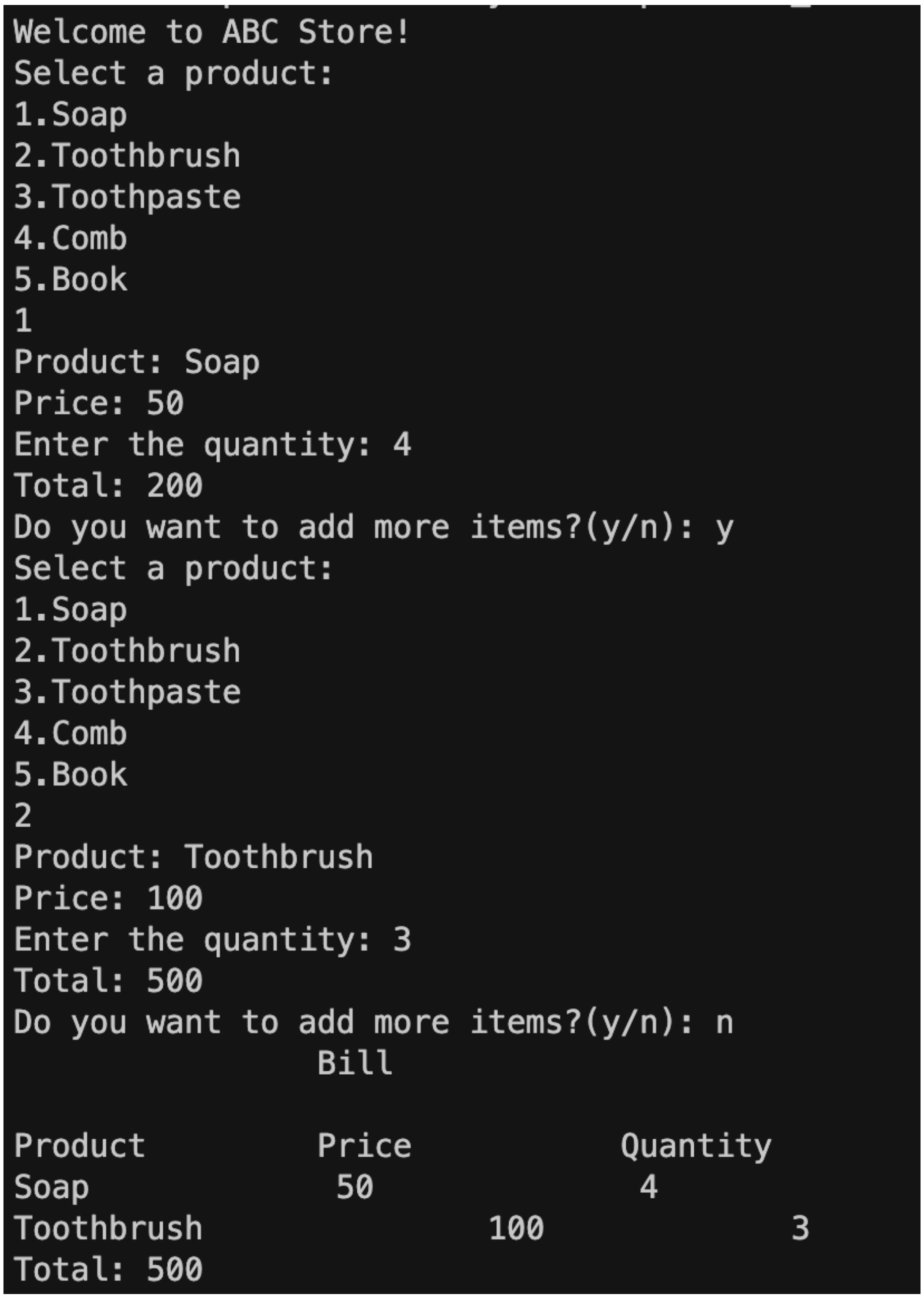
Output: (screenshot)

Test two



```
Welcome to ABC Store!
Select a product:
1.Soap
2.Toothbrush
3.Toothpaste
4.Comb
5.Book
1
Product: Soap
Price: 50
Enter the quantity: 4
Total: 200
Do you want to add more items?(y/n): y
Select a product:
1.Soap
2.Toothbrush
3.Toothpaste
4.Comb
5.Book
2
Product: Toothbrush
Price: 100
Enter the quantity: 3
Total: 500
Do you want to add more items?(y/n): n
                Bill

Product         Price           Quantity
Soap             50              4
Toothbrush              100             3
Total: 500
```

Case: Any (screenshot)



```
Welcome to ABC Store!
Select a product:
1.Soap
2.Toothbrush
3.Toothpaste
4.Comb
5.Book
1
Product: Soap
Price: 50
Enter the quantity: 4
Total: 200
Do you want to add more items?(y/n): y
Select a product:
1.Soap
2.Toothbrush
3.Toothpaste
4.Comb
5.Book
2
Product: Toothbrush
Price: 100
Enter the quantity: 3
Total: 500
Do you want to add more items?(y/n): n
                Bill

Product         Price          Quantity
Soap             50             4
Toothbrush              100             3
Total: 500
```

Conclusion: Hence, using constructor, printing a bill with total amount at the end.

Name of Student:Rafe Shaikh

Roll Number: 18

Experiment No:4

Title:

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

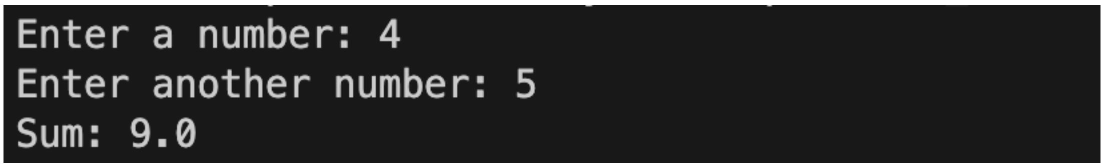
Theory:

Single inheritance is when there is a single parent class and a single child class which inherits the attributes and methods of the parent class. It reduces lines of code as instead of writing the same attributes and methods again in a new class, we can simply inherit them.

Code:

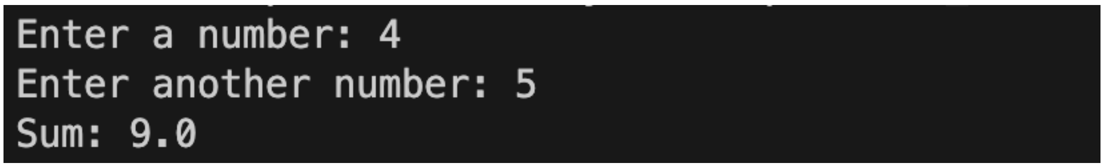
```python
#WAP to take input from user for addition of two numbers using
(single inheritance).
class Addition:
    def add(a,b):
        print("Sum:",a+b)
class Numbers(Addition):
    def getNumbers(self):
        c=float(input("Enter a number: "))
        d=float(input("Enter another number: "))
        Addition.add(c,d)
e=Numbers()
e.getNumbers()
```

Output: (screenshot)



```
Enter a number: 4
Enter another number: 5
Sum: 9.0
```

Test Case: Any two (screenshot)



```
Enter a number: 4
Enter another number: 5
Sum: 9.0
```

Conclusion:

Hence, using single inheritance to take values from the user from method of child class and adding and printing the sum from method of parent class.

Name of Student:Rafe Shaikh

Roll Number: 18

Experiment No:4

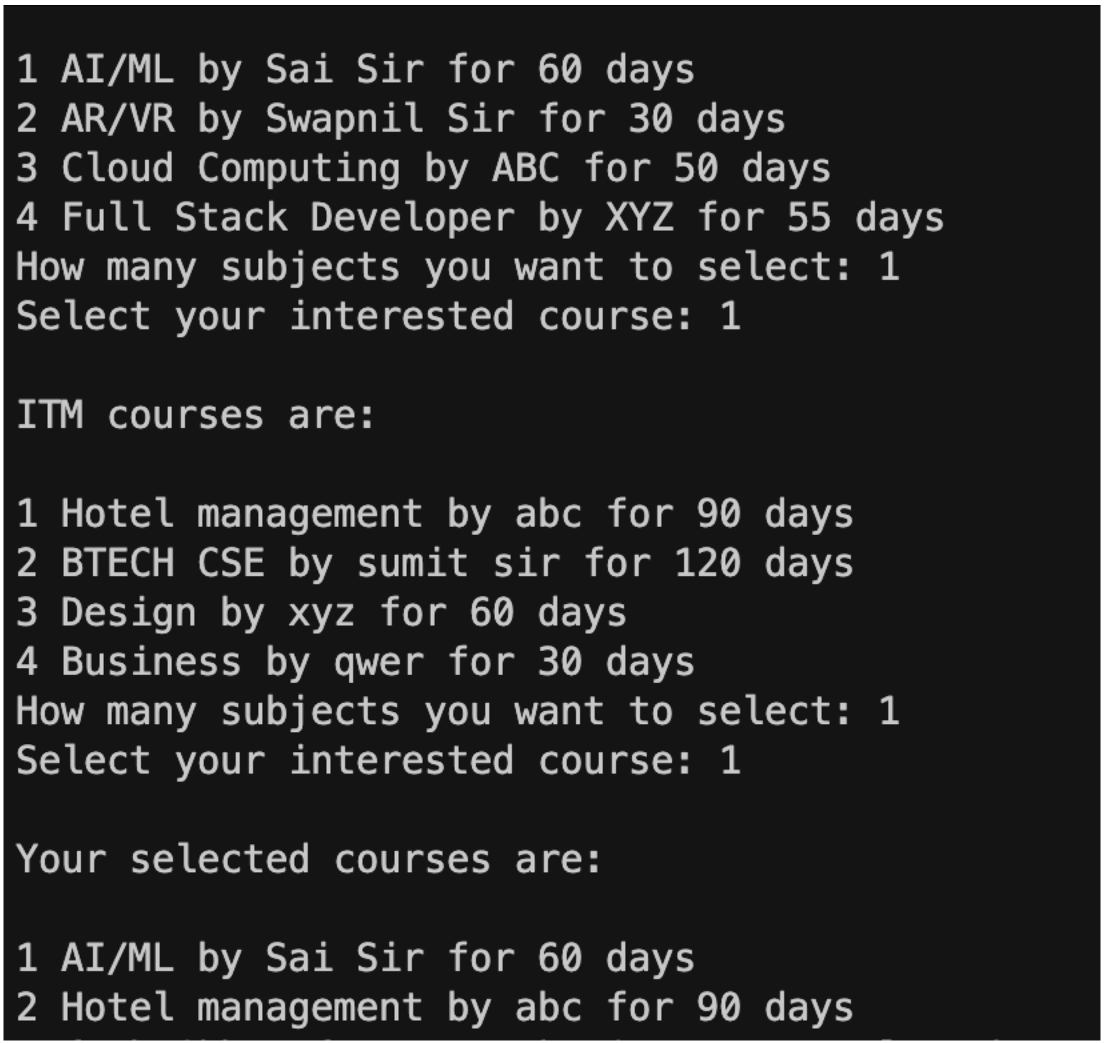
Title: 4.6 Write a program to create two base classes LU and ITM and one derived class. (Multiple inheritance).

Theory: Multiple inheritance is when a single child class inherits methods and attributes from multiple parent classes.

Code:

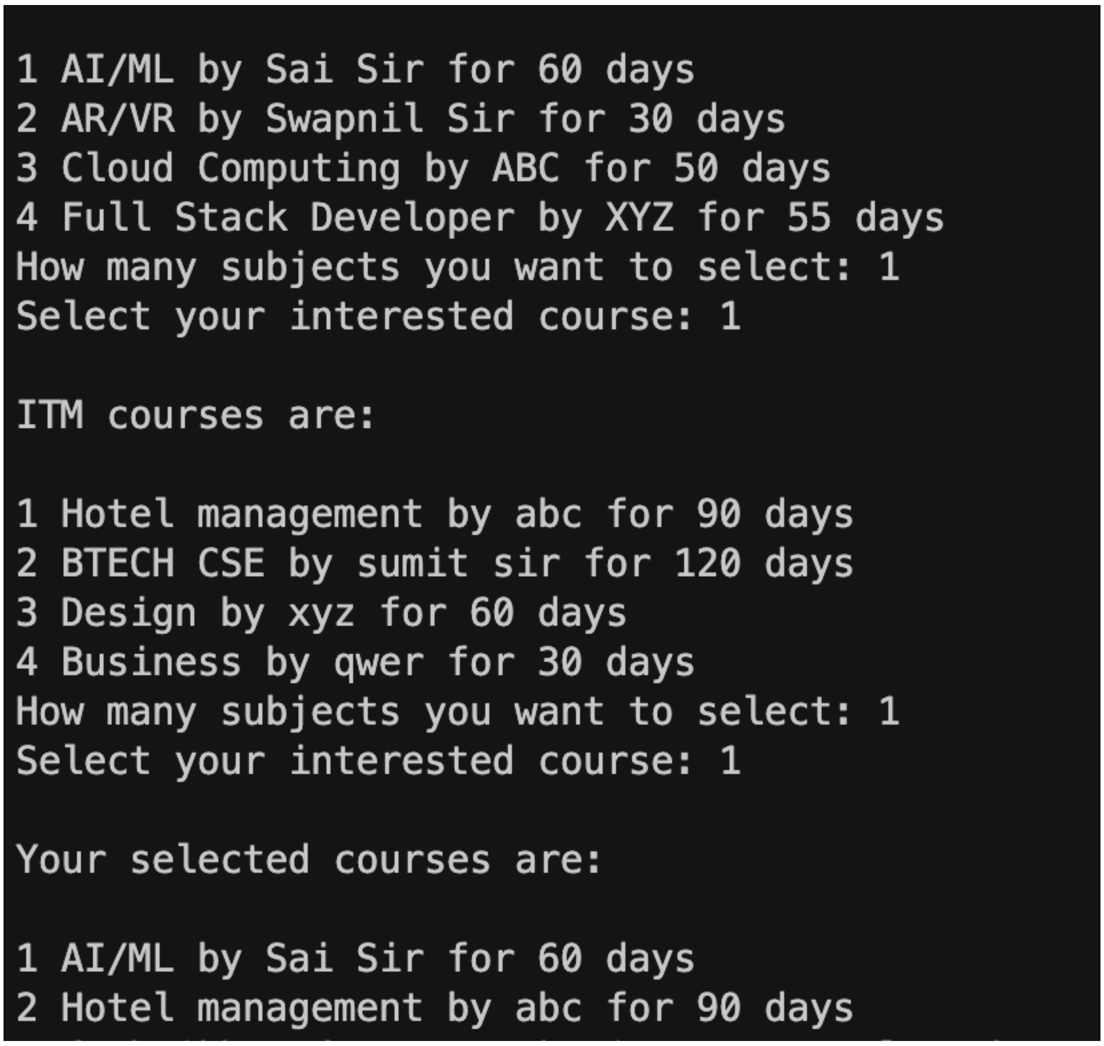
```
#WAP to create two base classes LU and ITM and one derived 
class(Multiple inheritance).
#multiple inheritance
class LU:
    subjects=["AI/ML","AR/VR","Cloud Computing","Full Stack 
Developer"]
    teachers=["Sai Sir","Swapnil Sir","ABC","XYZ"]
    duration=["60 days","30 days","50 days","55 days"]
class ITM:
    subjects=["Hotel management","BTECH CSE","Design","Business"]
    teachers=["abc","sumit sir","xyz","qwer"]
    duration=["90 days","120 days","60 days","30 days"]
class Btech(LU,ITM):
    print("LetsUpgrade courses are:\n")
    a=LU.subjects
    j=1
    for i in range(len(a)):

print(j,LU.subjects[i],"by",LU.teachers[i],"for",LU.duration[i])
        j+=1
    c=[]
    d=int(input("How many subjects you want to select: "))
    if d>4:
        print("Invalid number of subjects")
    else:
        for i in range(d):
            e=int(input("Select your interested course: "))
            c.append(e)
    print("\nITM courses are:\n")
    j=1
    b=ITM.subjects
    for i in range(len(b)):

print(j,ITM.subjects[i],"by",ITM.teachers[i],"for",ITM.duration[i])
        j+=1
    f=[]
    d=int(input("How many subjects you want to select: "))
    if d>4:
        print("Invalid number of subjects")
    else:
        for i in range(d):
            e=int(input("Select your interested course: "))
            f.append(e)
    j=1
    print("\nYour selected courses are:\n")
    for i in c:

print(j,LU.subjects[i-1],"by",LU.teachers[i-1],"for",LU.duration[i-
1])
        j+=1
    for i in f:

print(j,ITM.subjects[i-1],"by",ITM.teachers[i-1],"for",ITM.duration
[i-1])
        j+=1
obj=Btech()
```

Output:
(screenshot)



```
1 AI/ML by Sai Sir for 60 days
2 AR/VR by Swapnil Sir for 30 days
3 Cloud Computing by ABC for 50 days
4 Full Stack Developer by XYZ for 55 days
How many subjects you want to select: 1
Select your interested course: 1

ITM courses are:

1 Hotel management by abc for 90 days
2 BTECH CSE by sumit sir for 120 days
3 Design by xyz for 60 days
4 Business by qwer for 30 days
How many subjects you want to select: 1
Select your interested course: 1

Your selected courses are:

1 AI/ML by Sai Sir for 60 days
2 Hotel management by abc for 90 days
```

Test Case: Any two (screenshot)



```
1 AI/ML by Sai Sir for 60 days
2 AR/VR by Swapnil Sir for 30 days
3 Cloud Computing by ABC for 50 days
4 Full Stack Developer by XYZ for 55 days
How many subjects you want to select: 1
Select your interested course: 1

ITM courses are:

1 Hotel management by abc for 90 days
2 BTECH CSE by sumit sir for 120 days
3 Design by xyz for 60 days
4 Business by qwer for 30 days
How many subjects you want to select: 1
Select your interested course: 1

Your selected courses are:

1 AI/ML by Sai Sir for 60 days
2 Hotel management by abc for 90 days
```

Conclusion:

Hence, using multiple inheritance to show courses from multiple classes(LU and ITM) and print courses and their details which the user chooses.

Name of Student:Rafe Shaikh

Roll Number: 18

Experiment No: 4

Title: 4.7 Write a program to implement Multilevel inheritance, Grandfather->Father->Child to show property inheritance from grandfather to child.

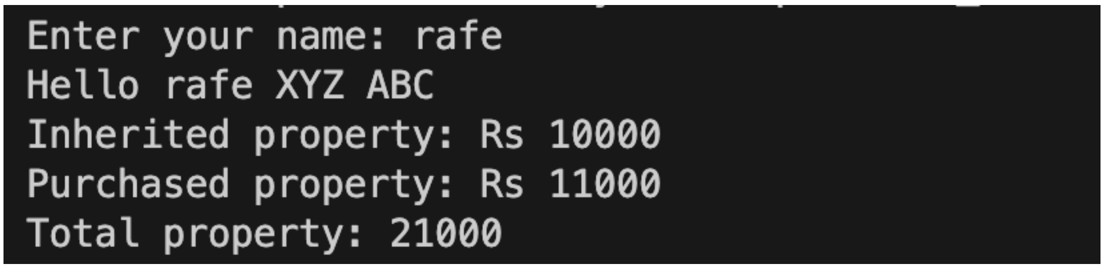
Theory:

Multilevel inheritance is when there is a parent class which has a child class which in turn has a child class of its own. The last child class inherits all the methods and attributes from its parent class as well as from the grandfather class.

Code:

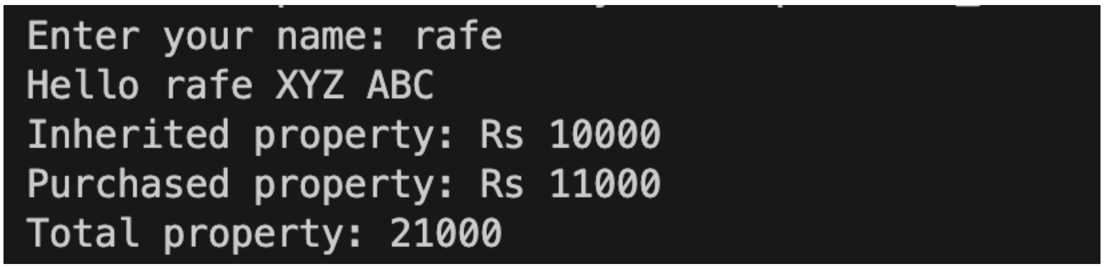
```
#WAP to implement Multilevel inheritance, Grandfather→Father-→Child
to show property inheritance from grandfather to child.
class Grandfather:
    def __init__(self):
        self.name="ABC"
        self.inherit=10000
        self.purchase=1000
class Father(Grandfather):
    def __init__(self):
        super().__init__()
        self.name=" XYZ "+self.name
        self.inherit=self.inherit
        self.purchase=10000+self.purchase
class Child(Father):
    def __init__(self,name):
        super().__init__()
        self.name=name+self.name
        self.inherit=self.inherit
        self.purchase=self.purchase
        print("Hello",self.name)
        print("Inherited property: Rs",self.inherit)
        print("Purchased property: Rs",self.purchase)
        print("Total property:",self.inherit+self.purchase)
a=input("Enter your name: ")
obj=Child(a)
```

Output: (screenshot)



```
Enter your name: rafe
Hello rafe XYZ ABC
Inherited property: Rs 10000
Purchased property: Rs 11000
Total property: 21000
```

Test Case: Any two (screenshot)



```
Enter your name: rafe
Hello rafe XYZ ABC
Inherited property: Rs 10000
Purchased property: Rs 11000
Total property: 21000
```

Conclusion:

Hence, using multilevel inheritance to take name from the user and calculate inherited and purchased property from grandfather and father for the child and their full name using methods and attributes from the grandfather and father class.

Name of Student:Rafe Shaikh

Roll Number: 18

Experiment No: 4

Title: 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)

Theory:

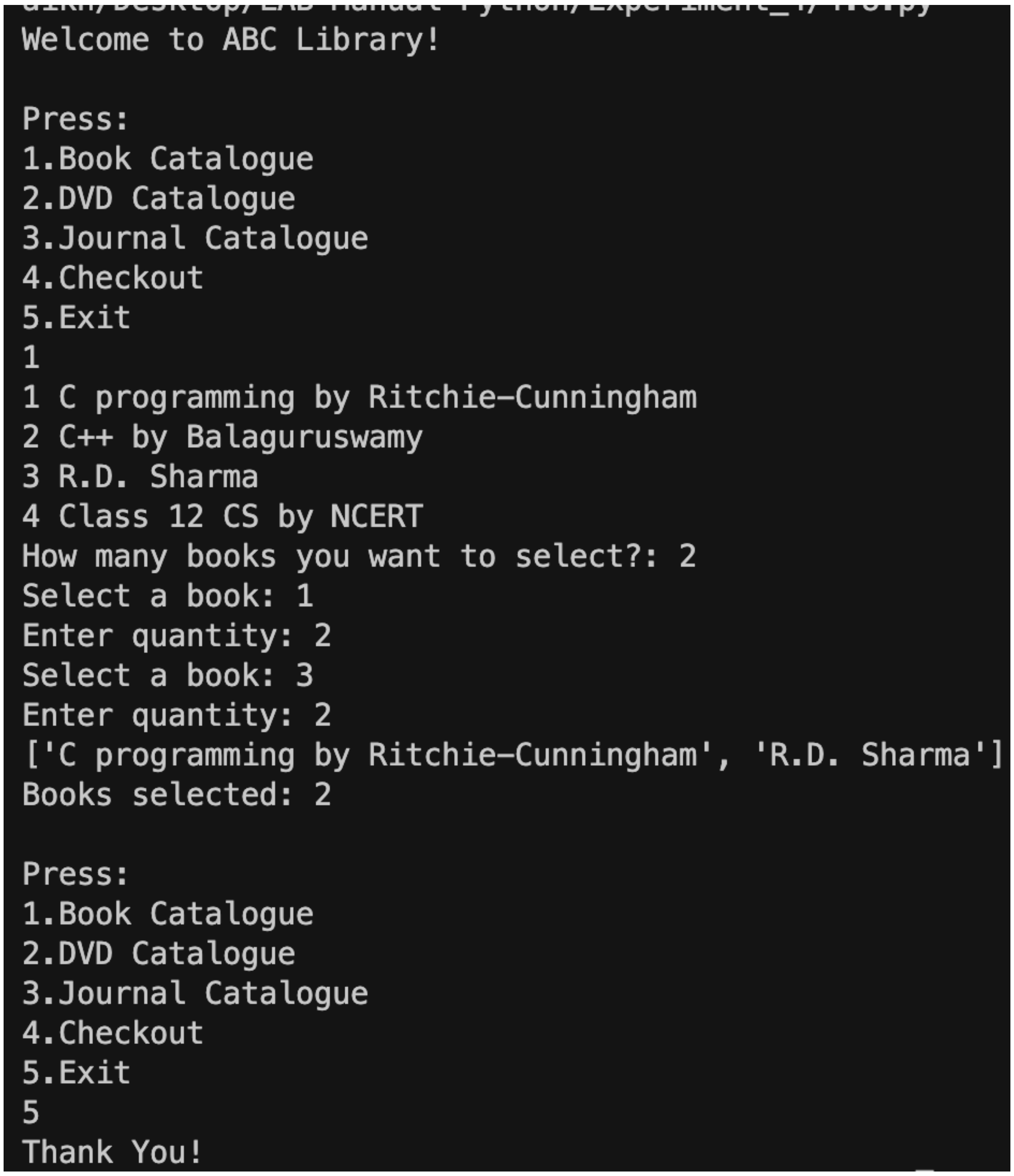
Method overriding is when there is a method of same name in both base class and derived class and when an object is created of derived class and method is called, derived class object is called and base class method is overridden.

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"]
```

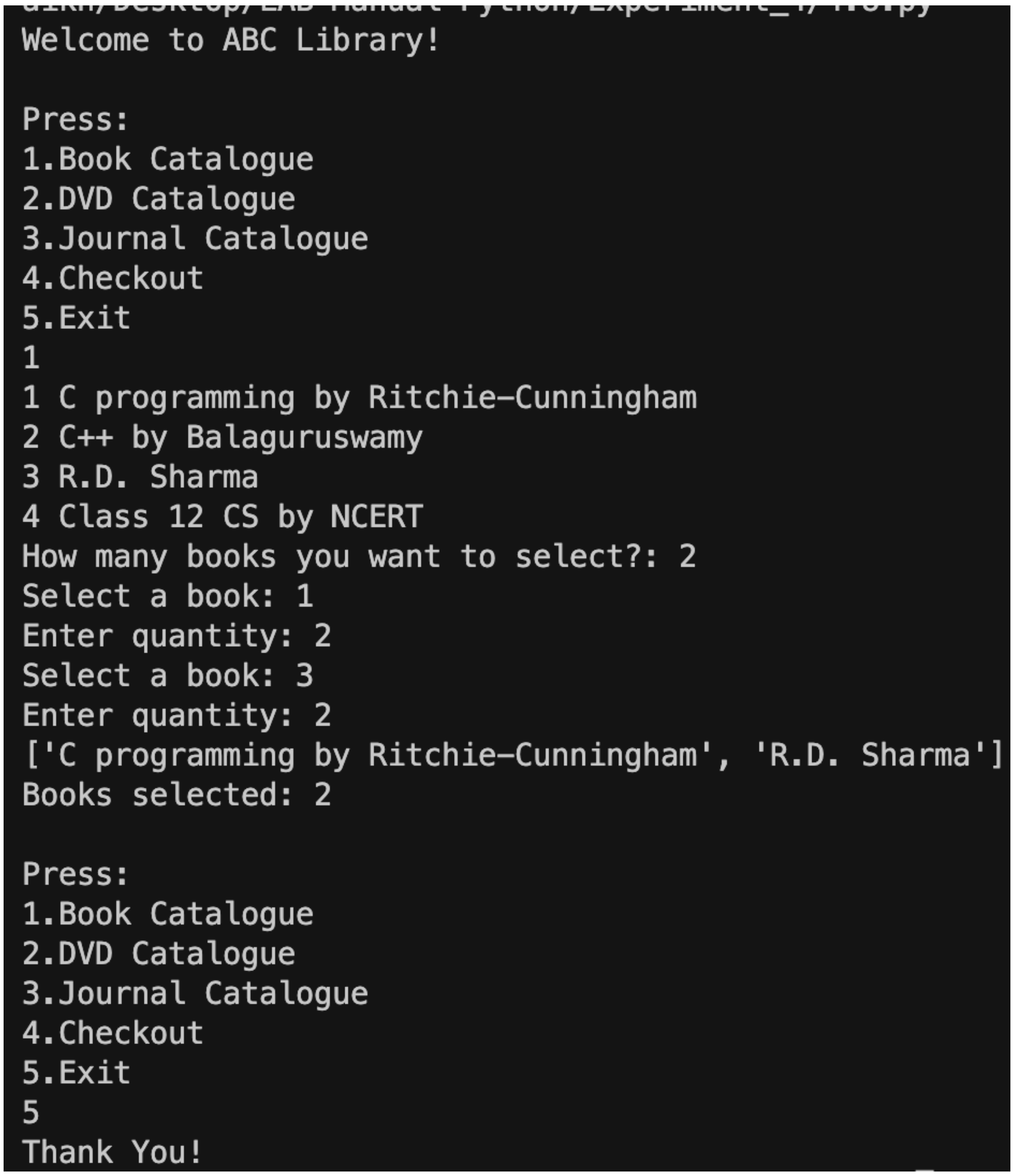
```
        b=1
        for i in range(len(a)):
            print(b,a[i])
            b+=1
        check_out_journal(self)
class Checkout(Library_item):
    def __init__(self):
        super().__init__()
        print("\nCheckout")
        j=1
        for i in range(len(self.cart)):
            print(j,self.cart[i],"-",self.quantity[i])
            j+=1
        print("\nBooks selected:",self._bookCount)
        print("Journals selected:",self._journalCount)
        print("DVDs selected:",self._dvdCount)

print("Total:",self._bookCount+self._journalCount+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)



```
Welcome to ABC Library!

Press:
1.Book Catalogue
2.DVD Catalogue
3.Journal Catalogue
4.Checkout
5.Exit
1
1 C programming by Ritchie-Cunningham
2 C++ by Balaguruswamy
3 R.D. Sharma
4 Class 12 CS by NCERT
How many books you want to select?: 2
Select a book: 1
Enter quantity: 2
Select a book: 3
Enter quantity: 2
['C programming by Ritchie-Cunningham', 'R.D. Sharma']
Books selected: 2

Press:
1.Book Catalogue
2.DVD Catalogue
3.Journal Catalogue
4.Checkout
5.Exit
5
Thank You!
```

Test Case: Any two (screenshot)



```
Welcome to ABC Library!

Press:
1.Book Catalogue
2.DVD Catalogue
3.Journal Catalogue
4.Checkout
5.Exit
1
1 C programming by Ritchie-Cunningham
2 C++ by Balaguruswamy
3 R.D. Sharma
4 Class 12 CS by NCERT
How many books you want to select?: 2
Select a book: 1
Enter quantity: 2
Select a book: 3
Enter quantity: 2
['C programming by Ritchie-Cunningham', 'R.D. Sharma']
Books selected: 2

Press:
1.Book Catalogue
2.DVD Catalogue
3.Journal Catalogue
4.Checkout
5.Exit
5
Thank You!
```

Conclusion:

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

Name of Student:Rafe Shaikh

Roll Number: **18**

Experiment No: 5

Title: 5.1 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:

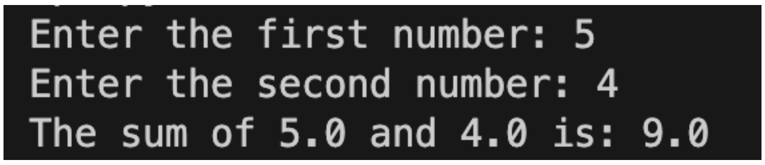
my_module.py

```python
# my_module.py

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

main_script.py

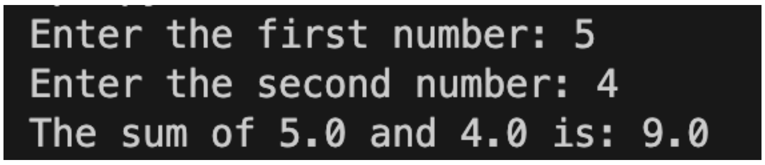
```python
# main_script.py
import my_module

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

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

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

Output: (screenshot)

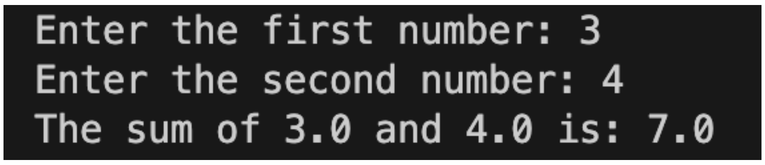


```
Enter the first number: 5
Enter the second number: 4
The sum of 5.0 and 4.0 is: 9.0
```

Test Case: Any two (screenshot)



```
Enter the first number: 5
Enter the second number: 4
The sum of 5.0 and 4.0 is: 9.0
```



```
Enter the first number: 3
Enter the second number: 4
The sum of 3.0 and 4.0 is: 7.0
```

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: Lakshya Duhoon

Roll Number: **46**

Experiment No: 5

Title: 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.

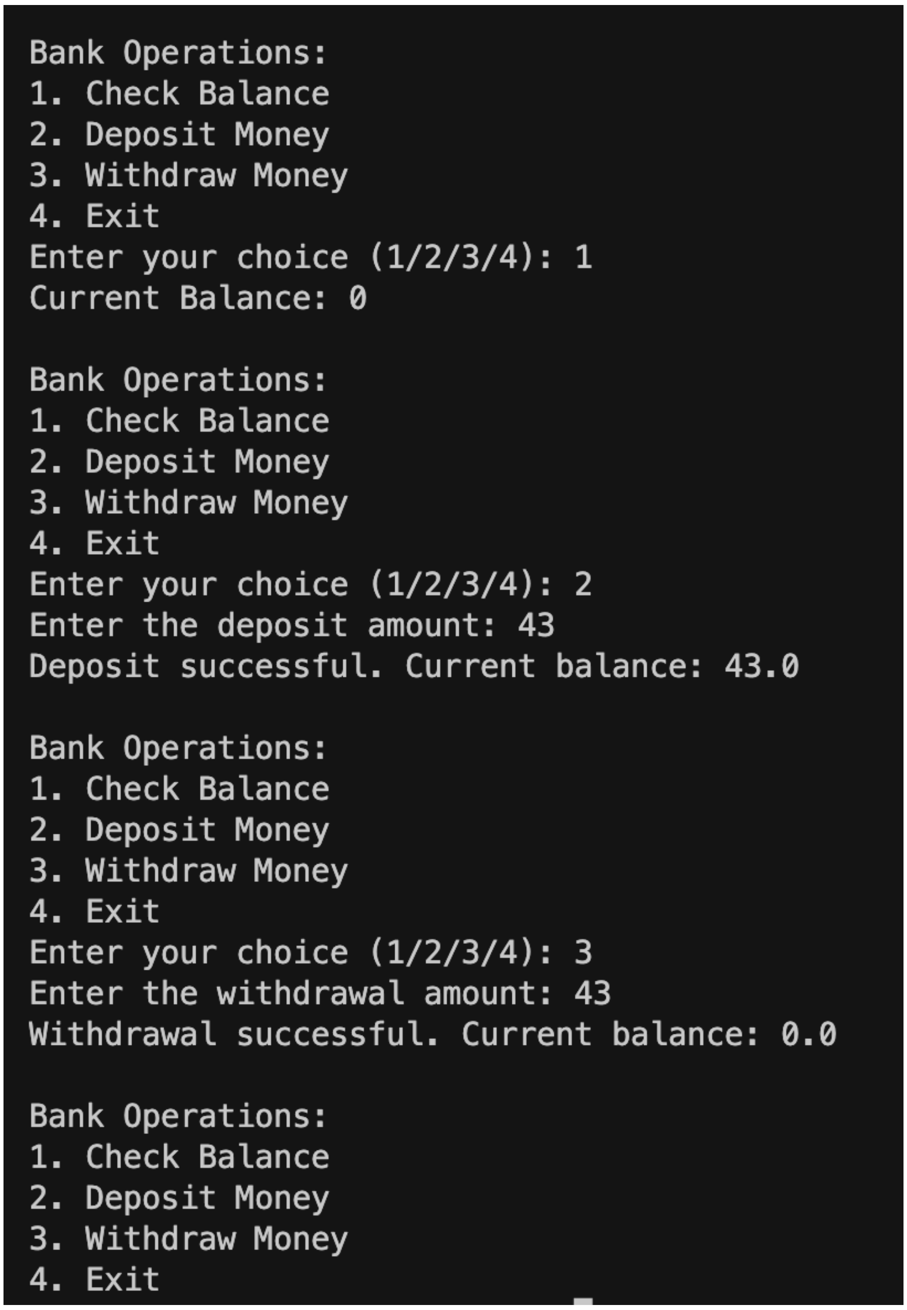
Theory:

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

Code:

bank_module.py -

```
# bank_module.py

class BankAccount:
    def __init__(self, balance=0):
        self.balance = balance

    def check_balance(self):
        return self.balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            return f"Deposit successful. Current balance: 
{self.balance}"
        else:
            return "Invalid deposit amount. Please enter a positive 
value."

    def withdraw(self, amount):
        if amount > 0 and amount <= self.balance:
            self.balance -= amount
            return f"Withdrawal successful. Current balance: 
{self.balance}"
        elif amount <= 0:
            return "Invalid withdrawal amount. Please enter a 
positive value."
        else:
            return "Insufficient funds for withdrawal."
```

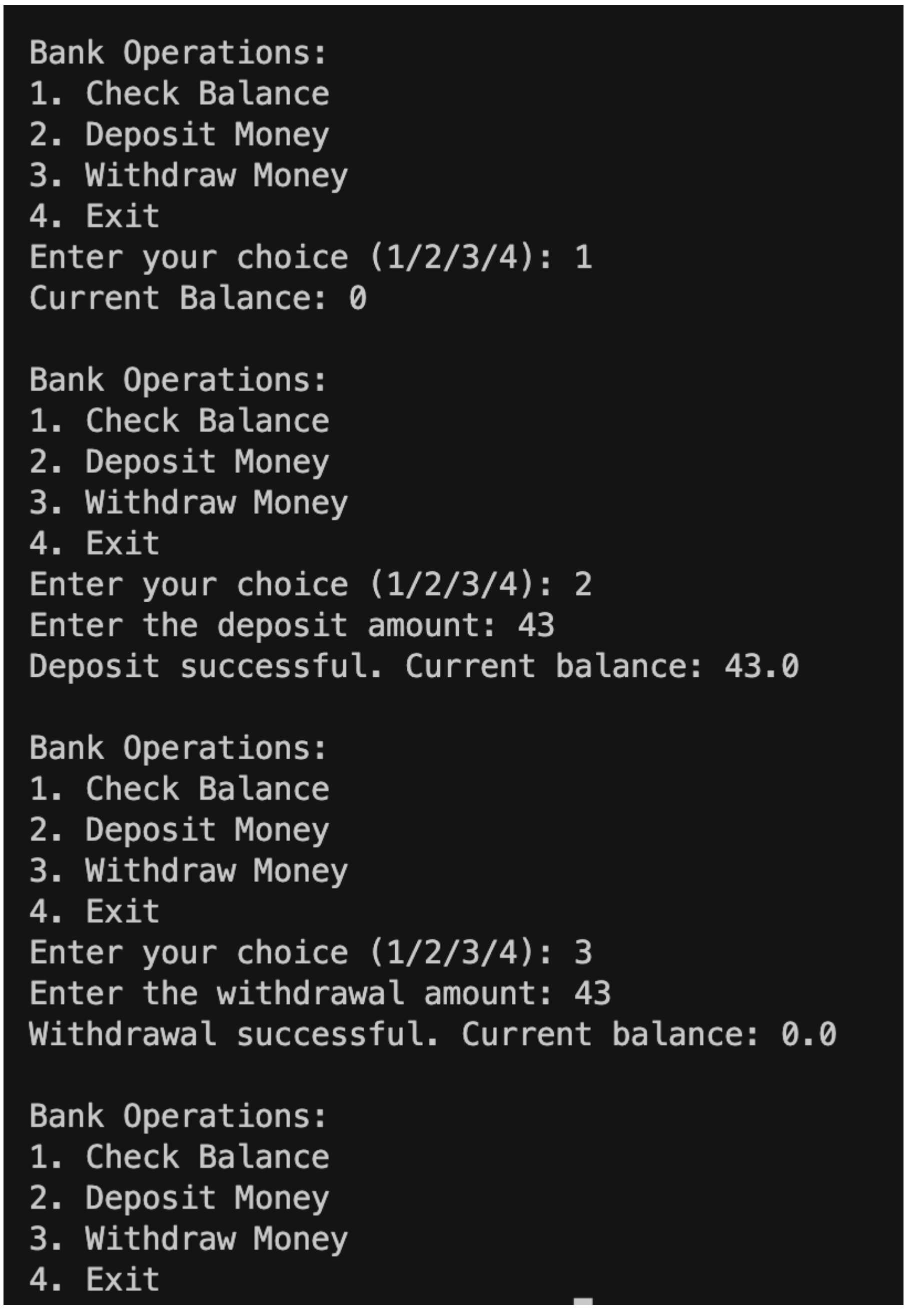


```
Bank Operations:
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
Enter your choice (1/2/3/4): 1
Current Balance: 0

Bank Operations:
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
Enter your choice (1/2/3/4): 2
Enter the deposit amount: 43
Deposit successful. Current balance: 43.0

Bank Operations:
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
Enter your choice (1/2/3/4): 3
Enter the withdrawal amount: 43
Withdrawal successful. Current balance: 0.0

Bank Operations:
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
```

main_script.py-

```
# main_script.py
from bank_module
import BankAccount

def main():
    # Creating a
bank account
    account =
BankAccount()

    while True:

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

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

        if choice
== '1':
            #
Checking balance

print(f"Current
Balance:
{account.check_balance()}")
        elif choice == '2':
            # Depositing money
            deposit_amount = float(input("Enter the deposit amount:
"))
            print(account.deposit(deposit_amount))
        elif choice == '3':
            # Withdrawing money
            withdraw_amount = float(input("Enter the withdrawal
amount: "))
            print(account.withdraw(withdraw_amount))
        elif choice == '4':
# Exiting the program
print("Exiting the Bank Operations. Thank you!")
break
        else:
print("Invalid choice. Please enter a valid option (1/2/3/4).")

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

Output: (screenshot)



```
Bank Operations:
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
Enter your choice (1/2/3/4): 1
Current Balance: 0

Bank Operations:
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
Enter your choice (1/2/3/4): 2
Enter the deposit amount: 43
Deposit successful. Current balance: 43.0

Bank Operations:
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
Enter your choice (1/2/3/4): 3
Enter the withdrawal amount: 43
Withdrawal successful. Current balance: 0.0

Bank Operations:
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
```

Test Case: Any two (screenshot)

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: Rafe Shaikh

Roll Number: **18**

Experiment No: 5

Title: 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.

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:

audi.py -

```
class Audi:
    def __init__(self, model):
```

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

bmw.py-

```python
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}.")
```

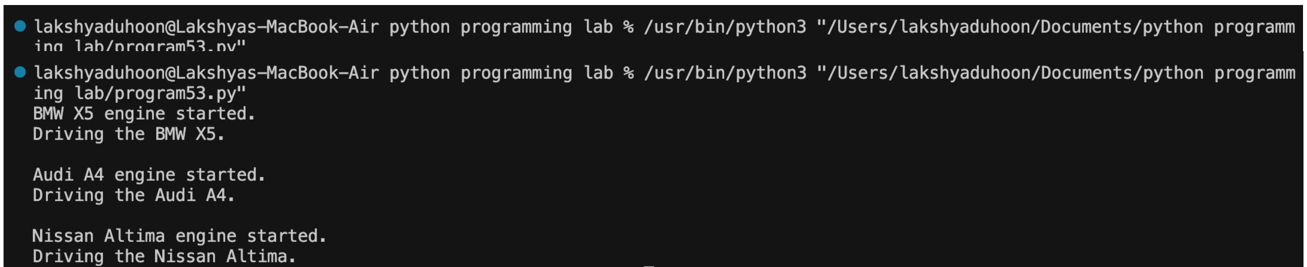
nissan.py-

```python
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}.")
```

main program-

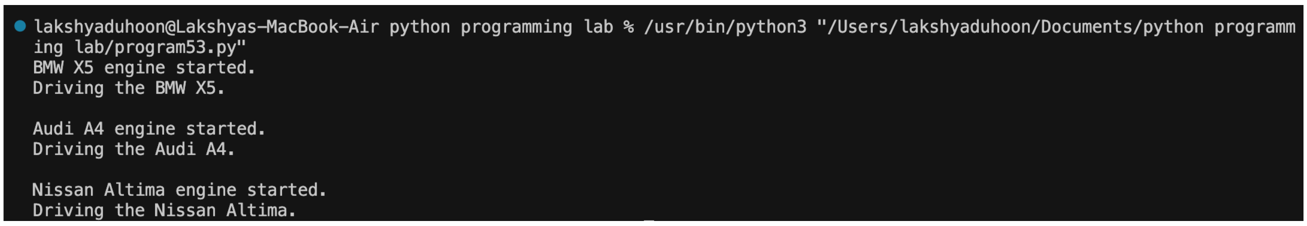
```python
#WAP to create a package with name cars and add different modules
(such as BMW, AUDI, NISSAN) having classes and functionality and
import them in main file cars.
from cars.bmw import BMW
from cars.audi import Audi
from cars.nissan import Nissan

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)



```
● lakshyaduhoon@Lakshyas-MacBook-Air python programming lab % /usr/bin/python3 "/Users/lakshyaduhoon/Documents/python programm
  ing lab/program53.py"
● lakshyaduhoon@Lakshyas-MacBook-Air python programming lab % /usr/bin/python3 "/Users/lakshyaduhoon/Documents/python programm
  ing lab/program53.py"
  BMW X5 engine started.
  Driving the BMW X5.

  Audi A4 engine started.
  Driving the Audi A4.

  Nissan Altima engine started.
  Driving the Nissan Altima.
```

Test Case: Any two (screenshot)



```
● lakshyaduhoon@Lakshyas-MacBook-Air python programming lab % /usr/bin/python3 "/Users/lakshyaduhoon/Documents/python programm
  ing lab/program53.py"
  BMW X5 engine started.
  Driving the BMW X5.

  Audi A4 engine started.
  Driving the Audi A4.

  Nissan Altima engine started.
  Driving the Nissan Altima.
```

Conclusion:

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

Name of Student:Rafe Shaikh

Roll Number: 18

Experiment No:6

Title: 6.1 Write a program to implement Multithreading. Printing "Hello" with one thread & printing "Hi" with another thread.

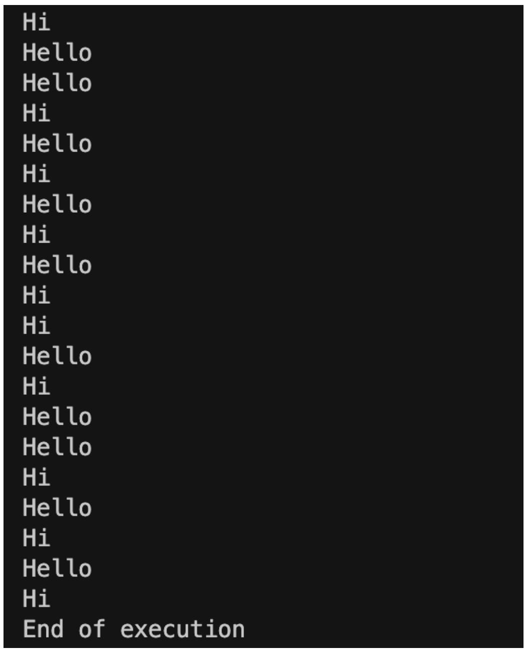
Theory:

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:

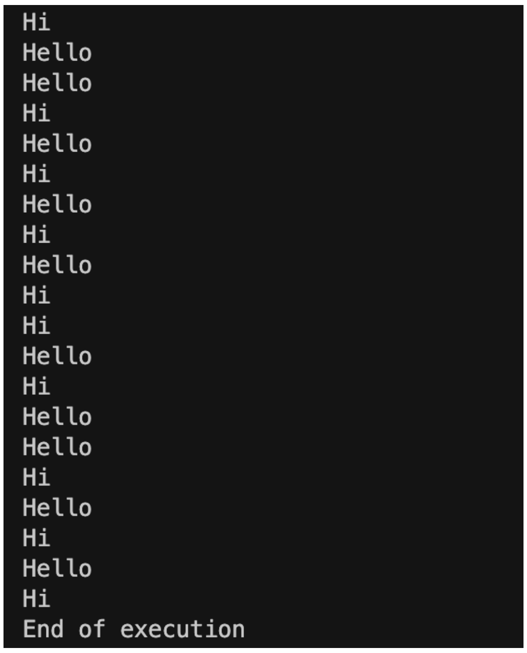
```
#WAP to implement Multithreading. Printing "Hello" with one thread
& printing "Hi" with another thread.
from threading import Thread
from time import sleep
def hi():
    for i in range(10):
        print("Hi")
        sleep(0.5)
def hello():
    for i in range(10):
        print("Hello")
        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)



```
Hi
Hello
Hello
Hi
Hello
Hi
Hello
Hi
Hello
Hi
Hi
Hello
Hi
Hello
Hello
Hi
Hello
Hi
Hello
Hi
End of execution
```

Test Case: Any two (screenshot)



```
Hi
Hello
Hello
Hi
Hello
Hi
Hello
Hi
Hello
Hi
Hi
Hello
Hi
Hello
Hello
Hi
Hello
Hi
Hello
Hi
End of execution
```

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:Rafe Shaikh

Roll Number: 18

Experiment No:7

Title:7.1 Write a program to use ‘weather API’ and print temperature of any city, also print the sunrise and sunset times for the same humidity of that area.

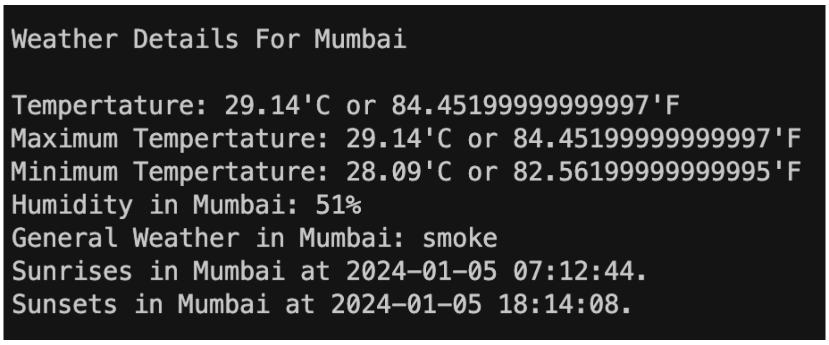
Theory: Requests module is used to connect a website with our program and an API key is required to get information from any website. API key is used for authentication of a user for getting any information from a website. Datetime module is used to convert Unix Epoch time to IST time for user convenience.

Code:import datetime as dt

```
import requests
```

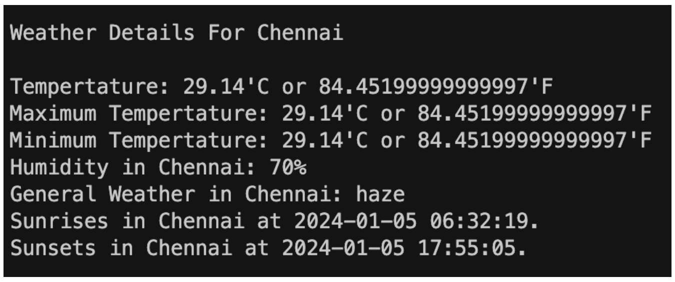
```
base_url="http://api.openweathermap.org/data/2.5/weather?"
api_key="233925bd1ab52f415a1273455bd44334"
city=input("Enter the city name: ")
def kel_to_cel_fahren(kelvin):
    celsius = kelvin - 273
    fahrenheit = celsius * (9/5) + 32
    return celsius, fahrenheit

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

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

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

Output: (screenshot)

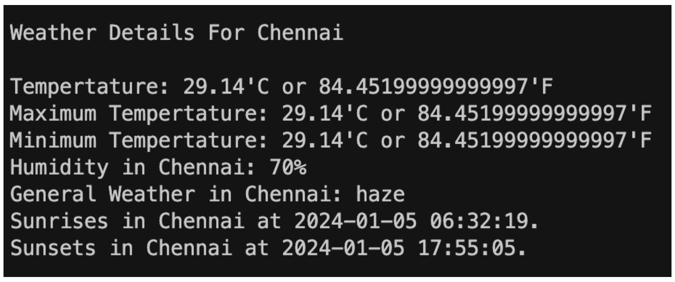


```
Weather Details For Mumbai

Tempertature: 29.14'C or 84.45199999999997'F
Maximum Tempertature: 29.14'C or 84.45199999999997'F
Minimum Tempertature: 28.09'C or 82.56199999999995'F
Humidity in Mumbai: 51%
General Weather in Mumbai: smoke
Sunrises in Mumbai at 2024-01-05 07:12:44.
Sunsets in Mumbai at 2024-01-05 18:14:08.
```

Test Case: Any two (screenshot)



```
Weather Details For Chennai

Tempertature: 29.14'C or 84.45199999999997'F
Maximum Tempertature: 29.14'C or 84.45199999999997'F
Minimum Tempertature: 29.14'C or 84.45199999999997'F
Humidity in Chennai: 70%
General Weather in Chennai: haze
Sunrises in Chennai at 2024-01-05 06:32:19.
Sunsets in Chennai at 2024-01-05 17:55:05.
```



```
Weather Details For Chennai

Tempertature: 29.14'C or 84.45199999999997'F
Maximum Tempertature: 29.14'C or 84.45199999999997'F
Minimum Tempertature: 29.14'C or 84.45199999999997'F
Humidity in Chennai: 70%
General Weather in Chennai: haze
Sunrises in Chennai at 2024-01-05 06:32:19.
Sunsets in Chennai at 2024-01-05 17:55:05.
```

Conclusion:

Hence, using API of openweather to get weather details of user given city and printing the weather details with the help of requests module and date time module to convert Unix epoch time to IST time.

Name of Student:Rafe Shaikh

Roll Number: 18

Experiment No:7

Title:7.2 Write a program to use the 'API' of crypto currency.

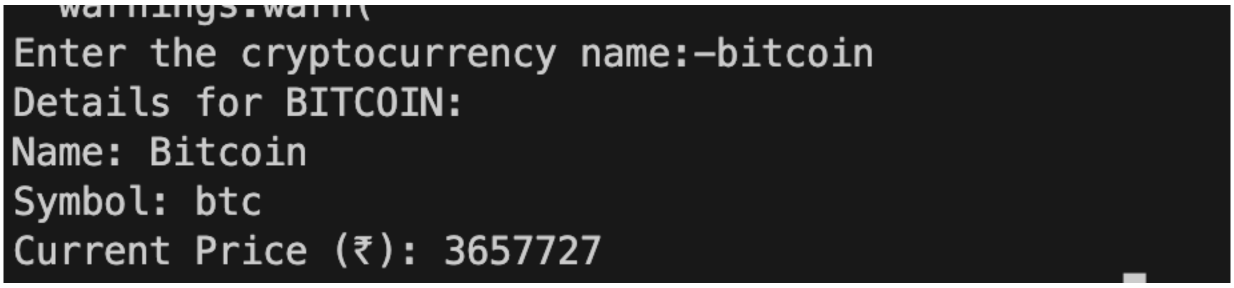
Theory:

Requests module is used to connect a website with our program and an API key is required to get information from any website. API key is used for authentication of a user for getting any information from a website.

Code:

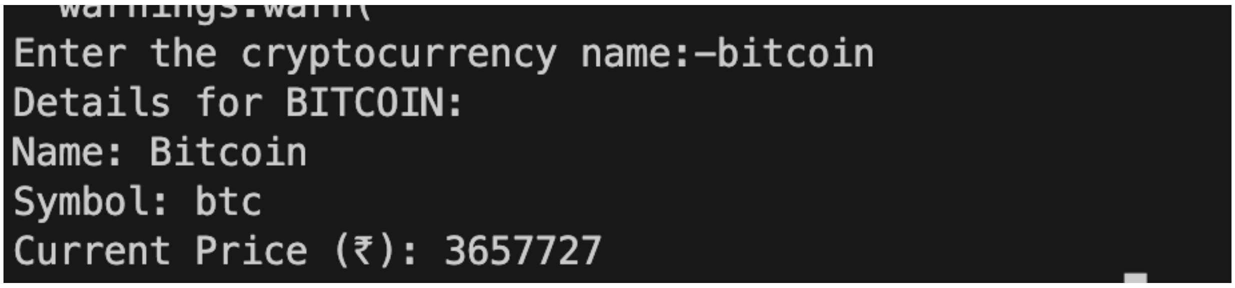
```
import requests

api_key="CG-Xk36brt3X2HFj3JYGJZpbFXP"
base_url="https://api.coingecko.com/api/v3/coins/"


cryptocurrency = input("Enter the cryptocurrency name:-").lower()

url = f"{base_url}{cryptocurrency}"

response = requests.get(url)

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

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

Output: (screenshot)

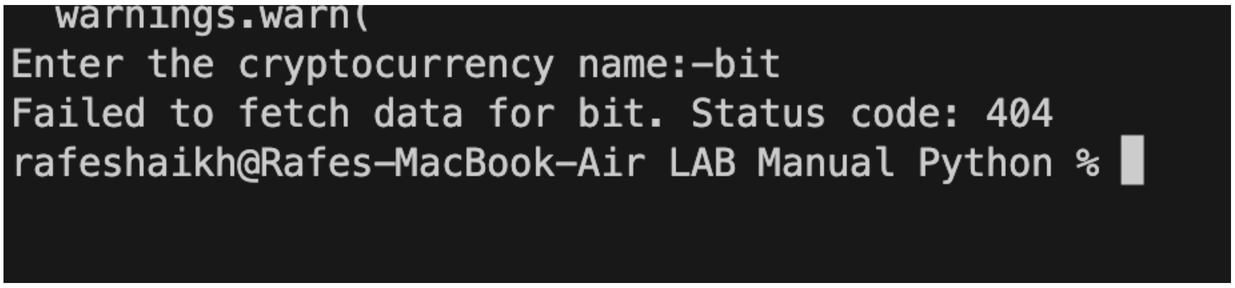


```
  warnings.warn(
Enter the cryptocurrency name:-bitcoin
Details for BITCOIN:
Name: Bitcoin
Symbol: btc
Current Price (₹): 3657727
```

Test Case: Any two (screenshot)



```
  warnings.warn(
Enter the cryptocurrency name:-bitcoin
Details for BITCOIN:
Name: Bitcoin
Symbol: btc
Current Price (₹): 3657727
```



```
  warnings.warn(
Enter the cryptocurrency name:-bit
Failed to fetch data for bit. Status code: 404
rafeshaikh@Rafes-MacBook-Air LAB Manual Python % 
```

Conclusion: Hence, using API of CoinGecko to get price of cryptocurrency given by the user and printing INR and USD price and asking for more crypto till the user chooses the option to terminate the program using while loop.