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

PYTHON PROGRAMMING LAB

Prepared by:

Name of Student:Husain mustufa hakim

Roll No:27

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: Husain

Roll Number: 27

Experiment No: 1.1

Title: *Write a program to compute Simple Interest.*

Theory:

- **Simple Interest SI is calculated using SI = (P x R x T) / 100 where P is the principal, R is the rate of interest, and T is the time (in years).**
- **The formula SI = (P x R x T) / 100 represents a straightforward linear relationship between principal, rate, and time in interest calculations.**

Code:

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


principal = float(input("Enter the Principal amount : "))
rate = float(input("Enter the annual interest rate : "))
time = float(input("Enter the maturity period (in years) : "))

print(f"Simple interest for {time} years : {(principal*time*rate)/
100.00}\nTotal Amount after {time} years : {principal +
((principal*time*rate)/100)}")
```

Output:

```
Enter the Principal amount : 1000
Enter the annual interest rate : 2
Enter the maturity period (in years) : 3
Simple interest for 3.0 years : 60.0
Total Amount after 3.0 years : 1060.0
```

Test Case:

1>

```
Enter the Principal amount : 12000.2
Enter the annual interest rate : 2.3
Enter the maturity period (in years) : 3.5
Simple interest for 3.5 years : 966.0161
Total Amount after 3.5 years : 12966.216100000001
```

2>

```
Enter the Principal amount : 1000
Enter the annual interest rate : 2.5
Enter the maturity period (in years) : 3
Simple interest for 3.0 years : 75.0
Total Amount after 3.0 years : 1075.0
```

Conclusion:

- *The Python program efficiently calculates Simple Interest based on user-provided values.*
- *It demonstrates the practical application of a mathematical formula in programming.*
- *Engaging with this program enhances skills in implementing mathematical concepts for real-world problem-solving.*

Experiment No: 1.2

Title:

Write a program to perform arithmetic, Relational operators.

Theory:

- *Arithmetic operators in Python, such as +, -, *, /, and %, perform basic mathematical operations like addition, subtraction, multiplication, division, and modulus.*
- *Relational operators like ==, !=, <, >, <=, and >= are used to compare values, producing Boolean results (True or False) based on the specified condition.*

Code:

```
#Write a program to perform arithmetic, Relational operators.
print("ABSOLUTE VALUE CALCULATOR")
num1 = int(input("Enter the number : "))
num2 = int(input("Enter second number  : "))

if(num1 > num2):
    print(f"{num1} is greater than {num2} \nSo , Absolute value = 
{num1 - num2}")
elif(num2 > num1):
    print(f"{num2} is greater than {num1} \nSo , Absolute value = 
{num2-num1}")
else:
    print(f"{num2} is equal to {num1} \nSo , Absolute value = 0")
```

Output:

```
ABSOLUTE VALUE CALCULATOR
Enter the number : 10
Enter second number  : 12
12 is greater than 10
So , Absolute value = 2
```

Test Case:

1>

```
ABSOLUTE VALUE CALCULATOR
Enter the number : 120
Enter second number  : 12
120 is greater than 12
So , Absolute value = 108
```

2>

```
ABSOLUTE VALUE CALCULATOR
Enter the number : 1300
Enter second number  : 1300
1300 is equal to 1300
So , Absolute value = 0
```

Conclusion:

- ***The Python program demonstrates the use of arithmetic operators for basic mathematical computations.***
- ***Integration of relational operators showcases the ability to make logical comparisons between values in a program.***
- ***Understanding and implementing these operators contribute to building foundational programming skills in Python.***

Experiment No:1.3

Title:

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

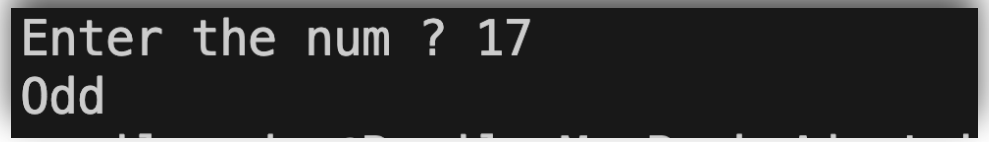
Theory:

- *An array containing the digits (0, 2, 4, 6, 8) is employed to check the least significant digit of a given number.*
- *The program utilizes this array to determine whether the given number is even or odd by checking if it ends with any of the specified digits.*

Code:

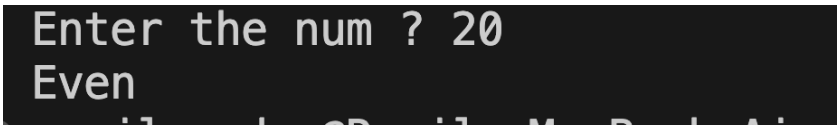
```
#Write a program to find whether a given no is even & odd.
ask = 10.2
while(ask!=int(ask)):
    try:
        ask = int(input("Enter the num ? "))
    except:
        print("Enter a integer num please ")

# print("Even") if ask % 2 == 0 else print("odd")


a = [0,2,4,6,8]
if(int(str(ask)[-1]) in a):
    print("Even")
else:
    print("Odd")
```

Output:

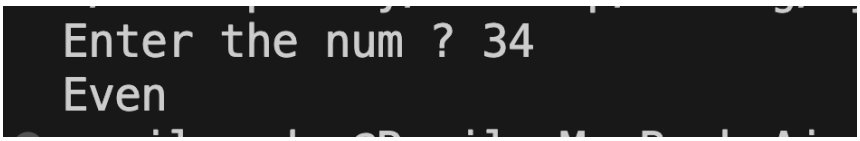


Test Case:

1>



2>



Conclusion:

- *The Python program creatively employs an array to identify even and odd numbers based on the last digit.*
- *This approach offers an alternative method to the traditional modulus operation for parity checks.*
- *By using arrays, the program showcases the flexibility and creativity in solving common programming challenges.*

Experiment No:1.4

Title:

Write a program to print first n natural number and their sum

Theory:

- *The program uses a for loop to iterate through the range of natural numbers up to a user-specified value.*
- *Inside the loop, it accumulates the sum of natural numbers and prints each number along with the intermediate sum.*
- *The output displays the series of natural numbers and their cumulative sum.*

Code:

```
ask = int(input("Enter the num till you want sum of? "))
sum = 0
for i in range(1,ask+1):
    sum += i

    if(i == ask):
        print(f"{i} = {sum}")
    else:
        print(i," + ",end = "")
```

Output:

```
Enter the num till you want sum of? 11
1  + 2  + 3  + 4  + 5  + 6  + 7  + 8  + 9  + 10  + 11 = 66
```

Test Case:

1>

```
Enter the num till you want sum of? 6
1  + 2  + 3  + 4  + 5  + 6 = 21
```

2>

```
Enter the num till you want sum of? 3
1  + 2  + 3 = 6
```

Conclusion:

- ***This Python program effectively generates and displays the first n natural numbers and their sum.***
- ***The loop structure simplifies the accumulation of the sum and sequential display of natural numbers.***
- ***Providing both individual numbers and the cumulative sum enhances the clarity of the program's output.***

Experiment No:1.5

Title:

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

Theory:

- *The program prompts the user to enter a sentence.*
- *It initializes empty lists to store unique vowels (*vow*) and consonants (*con*).*
- *The program iterates through each character in the entered sentence, categorizing them as vowels or consonants.*
- *It counts and prints the unique vowels and consonants along with their respective counts.*

Code:

```
# Write a program to determine whether the character entered is a
Vowel or not

ask = input("Enter the sentence ")
vow = []
con = []
vowcount = 0
concount = 0

for i in ask:
    if(i == " "):
        continue
    else:
        if(i.upper() in ["A","E","I","O","U"]):
```

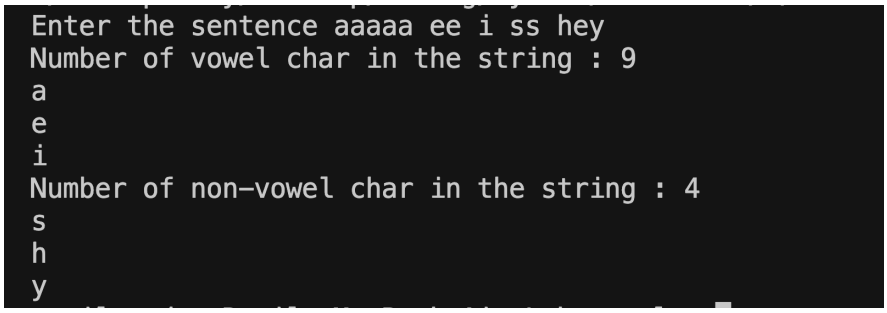
```
            if(i in vow):
                vowcount +=1
            else:
                vow.append(i)
        else:
            if(i in con):
                concount += 1
            else:
                con.append(i)

print(f"Number of vowel char in the string : {len(vow) +
vowcount}")
for i in vow:
    print(i)

print(f"Number of non-vowel char in the string : {len(con) +
concount}")
for i in con:
    print(i)
```

Output:

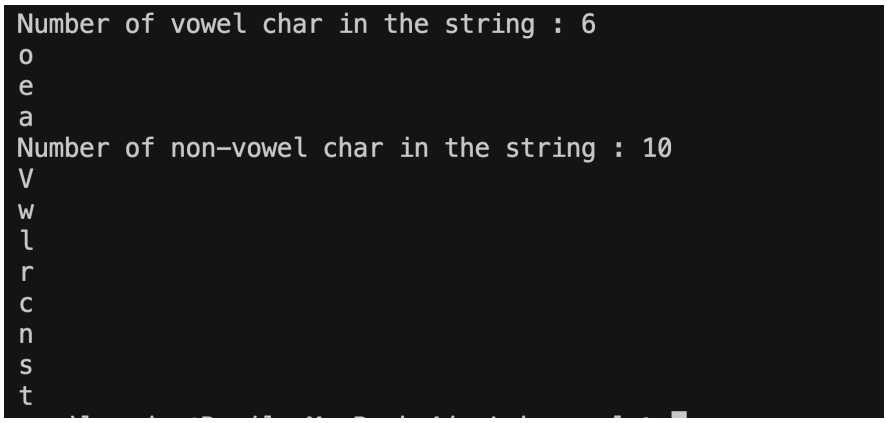
```
Enter the sentence Hey How are you
Number of vowel char in the string : 6
e
o
a
u
Number of non-vowel char in the string : 6
H
y
w
r
```

Test Case:

1>



2>



Conclusion:

- *This Python program effectively determines and categorizes vowels and consonants in the entered sentence.*
- *The use of lists (*vow *and* con*) helps store and display unique vowels and consonants.*
- *The program provides a clear count and list of both vowels and consonants, enhancing its functionality for character analysis.*

Experiment No:1.6

Title:

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

Theory:

- *The program prompts the user to enter a number to check if it's an Armstrong Number.*
- *It uses a for loop to iterate through each digit of the entered number, raising it to the power of the total number of digits and summing them.*
- *The program compares the computed sum with the original number to determine if it is an Armstrong Number.*

Code:

```
# Write a program to find whether given number is an Armstrong
Number.

ask = int(input("Enter the number to check if it's Armstrong : "))

sum = 0
for i in range(0,len(str(ask))):
    each = 1
    num = int(str(ask)[i])
    for j in range(len(str(ask))):
        each *= num
    sum += each
```

```
if(sum == ask):
    print("Armstrong")
else:
    print("Not")
```

Output:

```
Enter the number to check if it's Armstrong : 513
Not
```

Test Case:

1>

```
Enter the number to check if it's Armstrong : 153
Armstrong
```

2>

```
Enter the number to check if it's Armstrong : 1634
Armstrong
```

Conclusion:

- *This Python program effectively checks whether a given number is an Armstrong Number or not.*
- *Utilizing nested loops, the program calculates the sum of each digit raised to the power of the total digits.*
- *The program provides a clear output indicating whether the entered number is an Armstrong Number or not.*

Experiment No:1.7

Title:

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

Theory:

- *The program prompts the user to enter a number for which the factorial needs to be calculated.*
- *It utilizes a for loop to iterate through the range from 1 to the entered number, multiplying each iteration to calculate the factorial.*
- *The program then prints the computed factorial of the entered number.*

Code:

```
# Write a program using for loop to calculate factorial of a No.

ask = int(input("Enter the num : "))
fact = 1
for i in range(1,ask+1):
    fact *= i

print(fact)
```

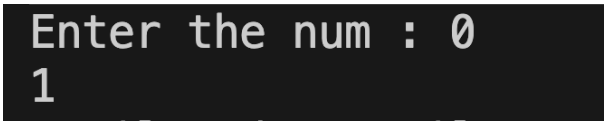
Output:

```
Enter the num : 5
120
```

Test Case:

1>

2>



Conclusion:

- *This Python program effectively calculates the factorial of a given number using a for loop.*
- *The use of a loop simplifies the iterative multiplication necessary for factorial computation.*
- *The program outputs the factorial, providing a straightforward solution for factorial calculations.*

Experiment No:1.8

Title:

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)

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

Theory:

1>

- *The program prompts the user to enter the number of rows for a pattern.*
- *It uses a for loop to iterate through the range of rows, printing asterisks in increasing quantity for each row.*

2>

- *The program prompts the user to enter the number of rows for a pattern.*
- *It uses a for loop to iterate through the range of rows, printing numbers in increasing quantity for each row.*

3>

- *The program prompts the user to enter the number of rows for a pattern.*
- *It uses a for loop to iterate through the range of rows, printing spaces and asterisks to create a triangular pattern.*

Code:

1>

```
# *
# * *
# * * *
# * * * *
# * * * * *
```

```
row = int(input("Enter the number of rows : "))

for i in range(1,row+1):
    print(i*"* ")
```

2>

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

```
ask = int(input("Enter the number of row : "))

for i in range(1,ask+1):
    print(i*f"{str(i)} ")
```

3>

```
#         *
#       * * *
#     * * * * *
#   * * * * * * *
# * * * * * * * * *
```

```
ask =  int(input("Ente the number of rows : "))

for i in range(1,ask+1):
    print((ask-i)*" ",i*"* ")
```

Output:

1>

```
Enter the number of rows : 4
*
* *
* * *
* * * *
```

2>

```
Enter the number of row : 4
1
2 2
3 3 3
4 4 4 4
```

3>

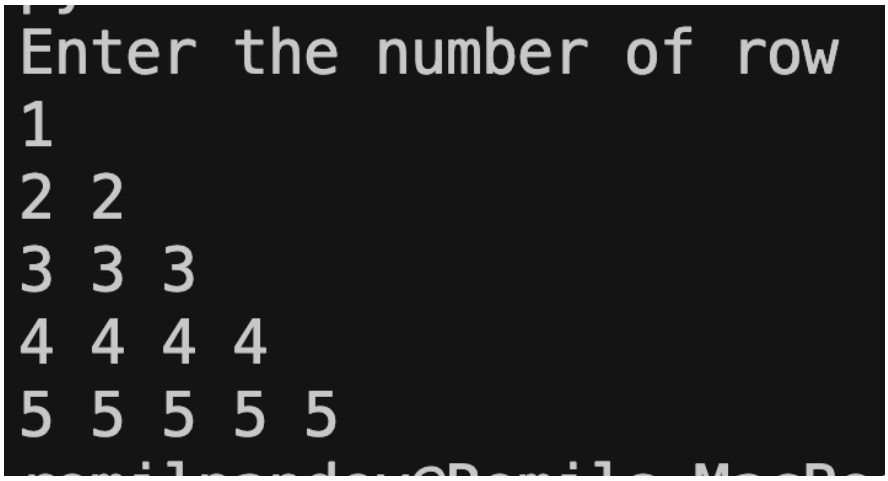
```
Ente the number of rows : 4
    *
   * *
  * * *
 * * * *
```

Test Case:

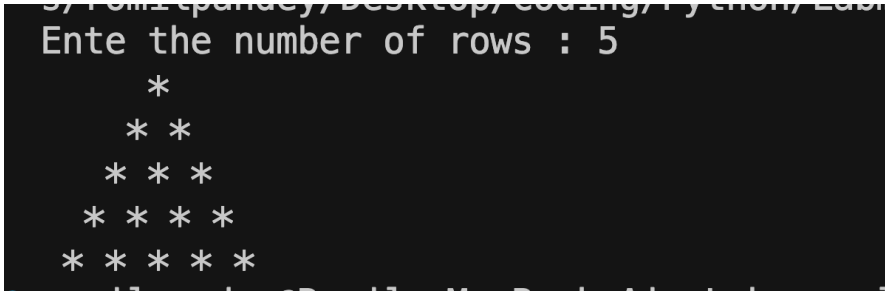
1>

```
Enter the number of rows : 6
*
* *
* * *
* * * *
* * * * *
* * * * * *
```

2>


Enter the number of row
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5

3>


Ente the number of rows : 5
*
* *
* * *
* * * *
* * * * *

Conclusion:

1>

- *This Python program generates a simple triangular pattern of asterisks based on the user-input number of rows.*
- *The loop efficiently controls the iteration and printing of asterisks for each row.*
- *The resulting output forms a visually appealing triangular pattern.*

2>

- *This Python program generates a pattern of numbers in a triangular format based on the user-input number of rows.*
- *The loop efficiently controls the iteration and printing of numbers for each row.*
- *The resulting output forms a visually appealing triangular pattern of increasing numbers.*

3>

- *This Python program generates a pattern of asterisks in a triangular format, with spaces creating a pyramid effect, based on the user-input number of rows.*
- *The loop efficiently controls the iteration and printing of spaces and asterisks for each row.*
- *The resulting output forms a visually appealing triangular pattern with an inverted pyramid of spaces and asterisks.*

Experiment No:2.1

Title:

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

Theory:

- *The program initializes a list a with the names of countries belonging to the BRICS group.*
- *It prompts the user to input a country name.*
- *The program checks if the entered country (in uppercase) is part of the BRICS group and provides the corresponding output.*

Code:

```
a = ["BRAZIL" , "RUSSIA" , "INDIA" , "CHINA" , "SOUTH AFRICA"]

ask = input("Enter the name")

if(ask.upper() in a):
    print(f"\n{ask} part Of BRICS")
else:
    print(f"\n{ask} Not Part of BRICS")
```

Output:

```
Enter the nameBrazil

Brazil part Of BRICS
```

Test Case:

1>

```
Enter the nameiNDiA

iNDiA part Of BRICS
```

2>

```
Enter the nameaskjnd

askjnd Not Part of BRICS
```

Conclusion:

- *This Python program efficiently checks whether a user-input country is part of the BRICS group.*
- *The use of the* `in` *operator simplifies the membership check within the list.*
- *The program outputs a clear message indicating whether the entered country is part of BRICS or not.*

Experiment No:2.2

Title:

Write a program to traverse a list in reverse order.
1.By using Reverse method.
2.By using slicing

Theory:

1>

- *The program prompts the user to enter a list of values separated by commas.*
- *It uses the* `reversed()` *function to traverse and print the elements of the list in reverse order.*

2>

- *The program prompts the user to enter a list of values separated by commas.*
- *It uses the slicing technique* `[::-1]` *to reverse the order of elements in the list and prints the result.*

Code:

1>

```
# Write a program to traverse a list in reverse order.
# 1.By using Reverse method.

l =  input("Enter list(sep val using \",\") : ").split(",")

for i in reversed(l):
    print(i,end=" ")
```

2>

```
l =  input("Enter list(sep val using \",\") : ").split(",")

print(l[::-1])
```

Output:

1>

```
Enter list(sep val using ",") : 1,2,3,4
4 3 2 1 %
```

2>

```
Enter list(sep val using ",") : 1,2,3,4
['4', '3', '2', '1']
```

Test Case:

1>

```
Enter list(sep val using ",") : 5,4,3,2,1
1 2 3 4 5 %
```

2>

Conclusion:

1>

- *This Python program efficiently traverses and prints the elements of the entered list in reverse order using the* `reversed()` *function.*
- *The use of the built-in function simplifies the process of iterating through the list in reverse.*
- *The program outputs the reversed list in a clear and concise manner.*

2>

- *This Python program efficiently reverses the order of elements in the entered list using slicing.*
- *The slicing technique* `[::-1]` *provides a concise and readable way to achieve list reversal.*
- *The program outputs the reversed list in a clear and straightforward manner.*

Experiment No:2.3

Title:

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

Theory:

- *The program prompts the user to enter an email address.*
- *It uses the* `split()` *function to divide the email address into a list containing the username and domain.*
- *The program then prints the extracted username and domain.*

Code:

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

ask = input("Enter the email ").split("@")

print(f"USERNAME : {ask[0]}\nDOMAIN : {ask[1]}")
```

Output:

```
  * * * *
  * * * * *
husainhakim@Husains-MacBook-Air python  %
```

Test Case:

```
Enter the email name@example.com
USERNAME : name
DOMAIN : example.com
romilpandey@Romils-MacBook-Air Labmanual % /opt/homebrew/bin/pyt
s/romilpandey/Desktop/Coding/Python/Labmanual/2/3scanemail.py
Enter the email yourname@yourdomain.com
USERNAME : yourname
DOMAIN : yourdomain.com
```

Conclusion:

- *This Python program efficiently extracts the username and domain from the entered email address using the* `split()` *function.*
- *The use of string manipulation simplifies the process of parsing the email components.*
- *The program outputs the username and domain in a clear and concise manner.*

Experiment No:2.4

Title:

Write a program to create a list of tuples from given list having number and add its cube in tuple.h i/p: c= [2,3,4,5,6,7,8,9]

Theory:

- *The program prompts the user to enter a list of numbers separated by commas.*
- *It uses a list comprehension to create a list of tuples, where each tuple contains a number from the input list and its cube.*
- *The program then iterates through the list of tuples and prints each tuple, displaying the number and its corresponding cube.*

Code:

```
#Write a program to create a list of tuples from given list having
number and
# add its cube in tuple.h
# i/p: c= [2,3,4,5,6,7,8,9]

c = [int(i) for i in input("Enter the list of num (seperated by \",
\")").split(",")]

tlist = [(i,i**3) for i in c]
print("All the number and it's cube")
for i in tlist:
    print(i)
```

Output:

```
Enter the list of num (seperated by ",")1,2,3
All the number and it's cube
(1, 1)
(2, 8)
(3, 27)
```

Test Case:

```
Enter the list of num (seperated by ",")4,5,6
All the number and it's cube
(4, 64)
(5, 125)
(6, 216)
```

Conclusion:

- *This Python program efficiently generates a list of tuples, each containing a number and its cube, based on the user-input list.*
- *List comprehension simplifies the creation of tuples, enhancing the readability of the code.*
- *The program outputs a clear list of tuples, displaying each number along with its cube.*

Experiment No:2.5

Title:

Write a program to compare two dictionaries in Python?(By using == operator)

Theory:

- *The program allows users to input key-value pairs for two dictionaries.*
- *It uses a while loop to repeatedly prompt users for dictionary entries until they choose to stop.*
- *The program then compares the two dictionaries using the == operator to determine if they are identical.*

Code:

```
# Write a program to compare two dictionaries in Python?
# (By using == operator)
d = {}
user = "y"
print("Dictionary No. - 1")
while(user.upper() == "Y"):
    key = input("Enter the key : ")
    value = input(f"Enter the value for the key {key} : ")
    d[key] = value
    user = input("Do you want to add another?(y/n)")

d1 = {}
user1 = "y"
```

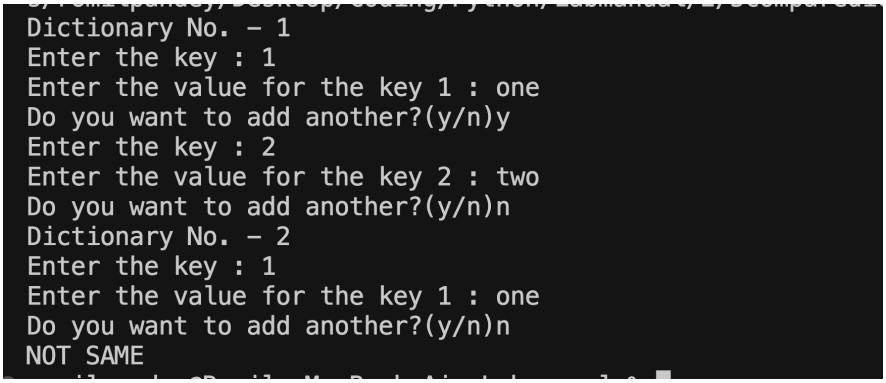
```
print("Dictionary No. - 2")
while(user1.upper() == "Y"):
    key1 = input("Enter the key : ")
    value1 = input(f"Enter the value for the key {key1} : ")
    d1[key1] = value1
    user1 = input("Do you want to add another?(y/n)")

if(d1 == d):
    print("SAME")
else:
    print("NOT SAME")
```

Output:

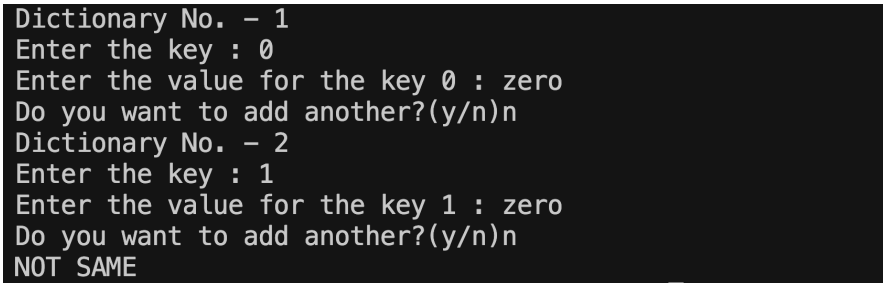
```
Dictionary No. - 1
Enter the key : 1
Enter the value for the key 1 : one
Do you want to add another?(y/n)y
Enter the key : 2
Enter the value for the key 2 : two
Do you want to add another?(y/n)n
Dictionary No. - 2
Enter the key : 1
Enter the value for the key 1 : one
Do you want to add another?(y/n)y
Enter the key : 2
Enter the value for the key 2 : two
Do you want to add another?(y/n)n
SAME
```

Test Case:

1>


Dictionary No. - 1
Enter the key : 1
Enter the value for the key 1 : one
Do you want to add another?(y/n)y
Enter the key : 2
Enter the value for the key 2 : two
Do you want to add another?(y/n)n
Dictionary No. - 2
Enter the key : 1
Enter the value for the key 1 : one
Do you want to add another?(y/n)n
NOT SAME

2>


Dictionary No. - 1
Enter the key : 0
Enter the value for the key 0 : zero
Do you want to add another?(y/n)n
Dictionary No. - 2
Enter the key : 1
Enter the value for the key 1 : zero
Do you want to add another?(y/n)n
NOT SAME

Conclusion:

- *This Python program efficiently compares two dictionaries entered by users using the == operator.*
- *The use of loops and user input makes the program interactive and adaptable to varying dictionary sizes.*
- *The program outputs whether the two dictionaries are the same or different based on the comparison result.*

Experiment No:2.6

Title:

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

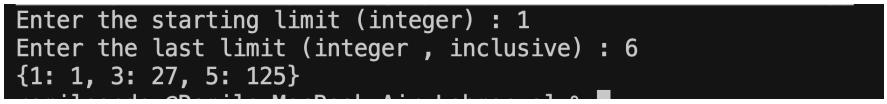
Theory:

- *The program prompts the user to input the starting and ending limits for a range.*
- *It adjusts the starting limit to the next odd number if necessary.*
- *Using a for loop, the program populates a dictionary with the cubes of odd numbers within the specified range.*

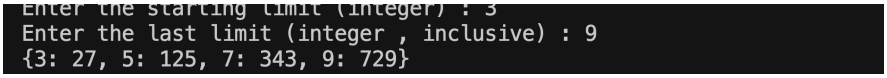
Code:

```
# Write a program that creates dictionary of cube of odd numbers in
the range.


d = {}

lowerrange = int(input("Enter the starting limit (integer) : "))
upperrange = int(input("Enter the last limit (integer ,
inclusive) : "))

lowerrange += 1 if lowerrange % 2 == 0 else 0

for i in range (lowerrange,upperrange+1,2):
    d[i] = i**3

print(d)
```

Output:



Test Case:



Conclusion:

- *This Python program efficiently creates a dictionary containing the cubes of odd numbers within a specified range.*
- *Adjusting the starting limit ensures that the range begins with an odd number, maintaining consistency.*
- *The program outputs the resulting dictionary, providing a clear mapping of odd numbers to their cubes.*

Experiment No:2.7

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:

- *The program manipulates a given list a using various list slicing operations.*
- *List slicing involves extracting specific portions of a list, enabling different operations on elements.*

Code:

1>

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

2>

```
a= [10,20,30,40,50,60,70,80,90,100]
print(a[3])
```

3>

```
a= [10,20,30,40,50,60,70,80,90,100]
print(a[0:5:1])
```

4>

```
a= [10,20,30,40,50,60,70,80,90,100]
print(a[-7:-(len(a)-2):-1])
```

5>

```
a= [10,20,30,40,50,60,70,80,90,100]
ele = int(input("Enter the element : "))
a.append(ele)
print(a)
```

6>

```
a = [int(i) for i in input("Enter the elements(\",\") : 
").split(",")]
#bubble sort

for i in range(0,len(a)):
    for j in range(i,len(a)):
        if(a[i] > a[j]):
            temp = a[i]
            a[i] = a[j]
            a[j] = temp
print(a)
```

7>

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

a.pop()

print(a)
```

8>

```
a = [10,20,30,40,50,60,70,80,90,100]
print(a)
ele = int(input("Enter the element to delete"))
a.remove(ele)
print(a)
```

9>

```
a= [10,20,30,40,50,60,70,80,90,100]
print(a)
index = int(input("Enter the index to remove "))
```

```
a.pop(index)
print(a)
```

10>

```
a= [10,20,10,30,50,50,50,20,90,100,30,40,50,60,70,80,90,100]
print(a)
ele = int(input("Enter the element to check occurences : "))
count = 0
for i in a:
    if(i == ele):
        count += 1

print(count)
```

11>

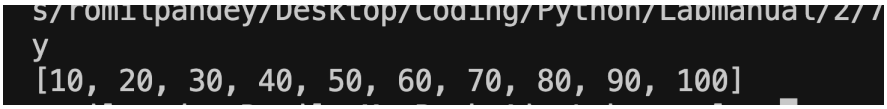
```
a= [10,20,30,40,50,60,70,80,90,100]
print(a)
l = [int(i) for i in input("Enter the elements to extend(seperated
by \",\") : ").split(",")]

a.extend(l)
print(a)
```

12>

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

print(f"Original List {a}")

#duplicate
l = []
for i in range(len(a)-1,-1,-1):
    l.append(a[i])
print("Duplicate reversed list : ",l)
#slicereverse

print("SLICED REVERSE : " , a[::-1])

#reversing the list

for i in range(0,int((len(a)/2.00)+0.5)):
    temp = a[i]
    a[i] = a[len(a)-i-1]
    a[len(a)-i-1] = temp

print("Reversed list : ",a)
```

Output:

1>



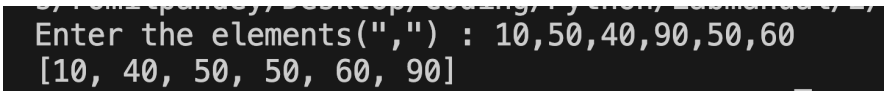
2>

3>

4>

5>

6>



7>

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

8>

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Enter the element to delete60
[10, 20, 30, 40, 50, 70, 80, 90, 100]
```

9>

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Enter the index to remove 3
[10, 20, 30, 50, 60, 70, 80, 90, 100]
```

10>

```
[10, 20, 10, 30, 50, 50, 50, 20, 90, 100, 30, 40, 50, 60, 70, 80, 90, 100]
Enter the element to check occurences : 50
4
```

11>

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Enter the elements to extend(seperated by ",") : 110,120,130
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130]
```

12>

```
Original List [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Duplicate reversed list :  [100, 90, 80, 70, 60, 50, 40, 30, 20, 10]
SLICED REVERSE :  [100, 90, 80, 70, 60, 50, 40, 30, 20, 10]
Reversed list :  [100, 90, 80, 70, 60, 50, 40, 30, 20, 10]
```

Test Case:

1>

```
y
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 120, 110]
```

5>

```
Enter the element : 120
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 120]
```

6>

```
Enter the elements(",") : 50,40,30,20
[20, 30, 40, 50]
```

8>

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Enter the element to delete30
[10, 20, 40, 50, 60, 70, 80, 90, 100]
```

9>

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Enter the index to remove 2
[10, 20, 40, 50, 60, 70, 80, 90, 100]
```

10>

```
[10, 20, 10, 30, 50, 50, 50, 20, 90, 100, 30, 40, 50, 60, 70, 80, 90, 100]
Enter the element to check occurences : 20
2
```

11>

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Enter the elements to extend(seperated by ",") : 1,23,22424,2
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 1, 23, 22424, 2]
```

12>

```
Original List [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Duplicate reversed list :  [100, 90, 80, 70, 60, 50, 40, 30, 20, 10]
SLICED REVERSE :  [100, 90, 80, 70, 60, 50, 40, 30, 20, 10]
Reversed list :  [100, 90, 80, 70, 60, 50, 40, 30, 20, 10]
```

Conclusion:

- *The complete list is printed, showcasing all elements in their original order.*
- *The 4th element is printed, demonstrating individual element access.*
- *The list from the 0th to the 4th index is printed, displaying a subset of the original list.*
- *Elements from the -7th to the 3rd position are printed, illustrating negative indexing.*
- *An element is appended to the list, showcasing the* `append()` *operation.*
- *The list is sorted, reordering elements in ascending order.*
- *An element is popped from the list, demonstrating the* `pop()` *operation.*
- *A specified element is removed from the list using the* `remove()` *operation.*
- *An element is inserted at a specified index, showcasing the* `insert()` *operation.*
- *The occurrence of a specified element is counted using the* `count()` *operation.*
- *The list is extended with additional elements, illustrating the* `extend()` *operation.*
- *The list is reversed, changing the order of elements in-place using the* `reverse()` *operation.*

Experiment No: 3.1

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:

1>

- *The program initializes a list* `a` *with some initial elements.*
- *It prompts the user to input additional elements for a new list* `l`*.*
- *The program extends the original list* `a` *by using the += operator to concatenate it with the new list* `l`*.*

2>

- *It prompts the user to input an additional element* `ele`*.*
- *The program uses the* `append()` *method to add the entered element to the end of the list*

3>

- *It prompts the user to input additional elements for a new list* `l`*.*
- *The program uses the* `extend()` *method to add all elements from the new list* `l` *to the end of the original list* `a`

Code:

1>

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

print(a)
l = [int(i) for i in input("Enter the elements (sep by \",\") : 
").split(",")]

a+=l
print(a)
```

2>

```
a = [10,20,30,40,50,60]
print(a)
```

```
ele = [int(i)for i in (input("Enter the element to add to list (sep
by \",\")").split(","))]

for i in ele:
    a.append(i)
print(a)
```

3>

```
a = [10,20,30,40,50,60,70]
print(a)
l = [int(i) for i in input("Enter list ot extend in the end :
").split(",")]

a.extend(l)
print(a)
```

Output:

1>

```
[10, 20, 30, 40, 50, 60, 70]
Enter the elements (sep by ",") : 80,90,100
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

2>

```
 [10, 20, 30, 40, 50, 60]
 Enter the element to add to list (sep by ",")70,80,90
 [10, 20, 30, 40, 50, 60, 70, 80, 90]
```

3>

```
[10, 20, 30, 40, 50, 60, 70]
Enter list ot extend in the end : 80,90,100
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

Test Case:

1>

```
[10, 20, 30, 40, 50, 60, 70]
Enter the elements (sep by ",") : 54,643,4352,2,1
[10, 20, 30, 40, 50, 60, 70, 54, 643, 4352, 2, 1]
```

2>

```
[10, 20, 30, 40, 50, 60]
Enter the element to add to list (sep by ",")23,32,6,7
[10, 20, 30, 40, 50, 60, 23, 32, 6, 7]
```

3>

```
[10, 20, 30, 40, 50, 60, 70]
Enter list ot extend in the end : 5,32,67,8
[10, 20, 30, 40, 50, 60, 70, 5, 32, 67, 8]
```

Conclusion:

1>

- *This Python program efficiently extends a list using the += operator for concatenation.*
- *User input allows for dynamic expansion of the list based on user preferences.*
- *The program outputs the extended list, providing a clear demonstration of the list extension process.*

2>

- *This Python program efficiently adds an element to the end of a list using the* `append()` *method.*
- *User input allows for dynamic updating of the list based on the element provided.*
- *The program outputs the modified list, demonstrating the addition of the specified element.*

3>

- *This Python program efficiently extends a list by adding all elements from another list using the method.*
- *User input allows for dynamic expansion of the list based on user preferences.*
- *The program outputs the extended list, providing a clear demonstration of the list extension process.*

Experiment No: 3.2

Title:

Write a program to add two matrices.

Theory:

- *The program prompts the user to input the dimensions (rows and columns) for two matrices.*
- *It uses nested loops to input the elements for each matrix based on the provided dimensions.*
- *The program then adds the corresponding elements from both matrices to create a new matrix.*
- *The resulting matrix is printed, demonstrating the addition of two matrices.*

Code:

```
# Write a program to add two matrices.

rc1 = [int(i) for i in input("Enter rows and columns for matrice
1 : ").split(",")]
l2=[]
row1 = rc1[0]
column1 = rc1[1]

for i in range(0,row1):
    rowele = []
    for j in range(0,column1):
        a = int(input(f"Enter the element for {i+1} row and {j+1}
column "))
        rowele.append(a)
    l2.append(rowele)

l1 = []
rc2 = [int(i) for i in input("Enter rows and columns for matrice
2 : ").split(",")]

row2 = rc2[0]
column2 = rc2[1]

for i in range(0,row2):
    rowele = []
    for j in range(0,column2):
        a = int(input(f"Enter the element for {i+1} row and {j+1}
column "))
        rowele.append(a)
    l1.append(rowele)

row = row1 if row1<row2 else row2
column = column1 if column1<column2 else column2
```

```
l = []
for i in range(0,row):
    rowele = []
    for j in range(0,column):
        a = l1[i][j] + l2[i][j]
        rowele.append(a)
    l.append(rowele)

print(l)
```

Output:

```
Enter rows and columns for matrice 1 : 2,2
Enter the element for 1 row and 1 column 10
Enter the element for 1 row and 2 column 10
Enter the element for 2 row and 1 column 10
Enter the element for 2 row and 2 column 10
Enter rows and columns for matrice 2 : 2,2
Enter the element for 1 row and 1 column 10
Enter the element for 1 row and 2 column 10
Enter the element for 2 row and 1 column 10
Enter the element for 2 row and 2 column 10
[[20, 20], [20, 20]]
```

Test Case:

```
Enter rows and columns for matrice 1 : 3,3
Enter the element for 1 row and 1 column 10
Enter the element for 1 row and 2 column 10
Enter the element for 1 row and 3 column 10
Enter the element for 2 row and 1 column 10
Enter the element for 2 row and 2 column 1
Enter the element for 2 row and 3 column 0
Enter the element for 3 row and 1 column 2
Enter the element for 3 row and 2 column 3
Enter the element for 3 row and 3 column 4
Enter rows and columns for matrice 2 : 3,3
Enter the element for 1 row and 1 column 20
Enter the element for 1 row and 2 column 23
Enter the element for 1 row and 3 column 324
Enter the element for 2 row and 1 column 45
Enter the element for 2 row and 2 column 645
Enter the element for 2 row and 3 column 43
Enter the element for 3 row and 1 column 5
Enter the element for 3 row and 2 column 3
Enter the element for 3 row and 3 column 5
[[30, 33, 334], [55, 646, 43], [7, 6, 9]]
```

Conclusion:

- *This Python program efficiently adds two matrices based on user input for dimensions and matrix elements.*
- *The use of nested loops allows for the input of matrix elements and their addition.*
- *The program outputs the resulting matrix, showcasing the addition of the two input matrices.*

Experiment No: 3.3

Title:

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

Theory:

- *The program prompts the user to enter a list of elements separated by commas.*
- *It uses a for loop to iterate through the elements of the input list.*
- *The program checks for the presence of each element in a new list* `l`*, and if not present, appends it to* `l`*.*
- *The resulting list* `l` *contains distinct elements from the input list.*

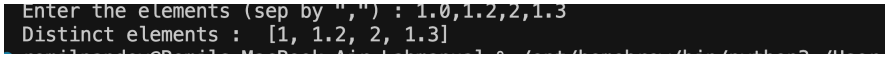
Code:

```
a = [float(i) for i in input("Enter the elements (sep by \",\") :
").split(",")]
l = []
for i in a:
    if(i not in l):
        if(int(i) == i):
            l.append(int(i))
        else:
            l.append(i)

print("Distinct elements : ", l)
```

Output:

```
Enter the elements (sep by ",") : 10,23,23,11,12,12
Distinct elements :  [10, 23, 11, 12]
```

Test Case:



Conclusion:

- *This Python program efficiently identifies and extracts distinct elements from a given list.*
- *The use of a new list ensures that only unique elements are included.*
- *The program outputs the distinct elements, providing a clear demonstration of the removal of duplicates.*

Experiment No: 4.1

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

- *The program defines a class* `Employee` *with an* `__init__` *method to initialize employee details such as ID, name, and salary.*
- *It includes a method* `get_employee_details` *to retrieve the details in a formatted string.*
- *The class also has a* `print_employee_details` *method to directly print the details.*
- *An instance* `employee1` *is created with specific details (employee ID, name, and salary).*
- *The program demonstrates two ways to access and display employee details: using the* `get_employee_details` *method and the* `print_employee_details` *method.*

CODE:

```
# Write a program to Create Employee Class & add methods to get
employee
# details & print.

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):
        return f"Employee ID: {self.emp_id}\nEmployee Name:
{self.emp_name}\nEmployee Salary: {self.emp_salary}"

    def print_employee_details(self):
        print(self.get_employee_details())


employee1 = Employee(emp_id=101, emp_name="John Doe",
emp_salary=50000)
```

```
details = employee1.get_employee_details()
print("Employee Details (using get_employee_details method):\n",
details)

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

OUTPUT:

```
Employee Details (using get_employee_details method):
 Employee ID: 101
Employee Name: John Doe
Employee Salary: 50000

Employee Details (using print_employee_details method):
Employee ID: 101
Employee Name: John Doe
Employee Salary: 50000
```

TEST CASE:

```
ython /pythonlabmanual.py/factorial.py"
Employee Details (using get_employee_details method):
 Employee ID: 101
Employee Name: John Doe
Employee Salary: 50000

Employee Details (using print_employee_details method):
Employee ID: 101
Employee Name: John Doe
```

```
Employee Details (using get_employee_details method):
 Employee ID: 101
Employee Name: Your name
Employee Salary: Your Salary

Employee Details (using print_employee_details method):
Employee ID: 101
Employee Name: Your name
Employee Salary: Your Salary
```

CONCLUSION:

- *This Python program showcases the use of a class* `Employee` *to model and manage employee details.*
- *The class encapsulates data and behavior related to an employee.*
- *The program outputs employee details using two different methods, providing flexibility in how information is accessed and displayed.*

Experiment No: 4.2

TITLE:
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,
THEORY:

- *The program defines a function* `get_user_details` *that takes a combination of positional arguments (*`args`*) and keyword arguments (*`**kwargs`*).*
- *It checks if the required information (name, email, and age) is provided either as keyword arguments or as positional arguments.*
- *The function prints the user details based on the provided information.*

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):
    if 'name' in kwargs and 'email' in kwargs and 'age' in kwargs:
        name = kwargs['name']
        email = kwargs['email']
        age = kwargs['age']
    elif len(args) == 3:
        name, email, age = args
    else:
        print("Invalid input. Provide either name, email, and age as keyword arguments or as positional arguments.")
        return

    print("User Details:")
    print(f"Name: {name}")
    print(f"Email: {email}")
    print(f"Age: {age}")
```

```
get_user_details(name="John Doe", email="john@example.com", age=25)
```

OUTPUT:

```
s/romilpandey/Desktop/Coding/Python/Labmanual/4/
User Details:
Name: SUJAL
Email: sujal@example.com
Age: 25
```

TEST CASE:

```
User Details:
Name: Prem
Email: ashlin@example.com
Age: 25
```

```
User Details:
Name: Blah blah
Email: blahblah@example.com
Age: 25
```

CONCLUSION:

- *This Python program demonstrates the use of a function with a flexible parameter list to receive user details.*
- *The function allows users to provide input using a combination of positional and keyword arguments.*
- *The program outputs the user details, providing a clear example of how to handle input flexibility in functions.*

Experiment No: 4.3

TITLE:
*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,*
THEORY:

- *The program defines a class* `details` *to manage student admission details.*
- *The class includes a class variable* `count` *to keep track of the total admissions and separate counters* `bcount` *and* `pcount` *for BTech and PGDM admissions, respectively.*
- *The class has methods* `getdata` *to input student details and* `display` *to display the details based on the chosen department.*
- *The program uses a list* `objs` *to store instances of the* `details` *class and manages student admissions based on user input.*

CODE:

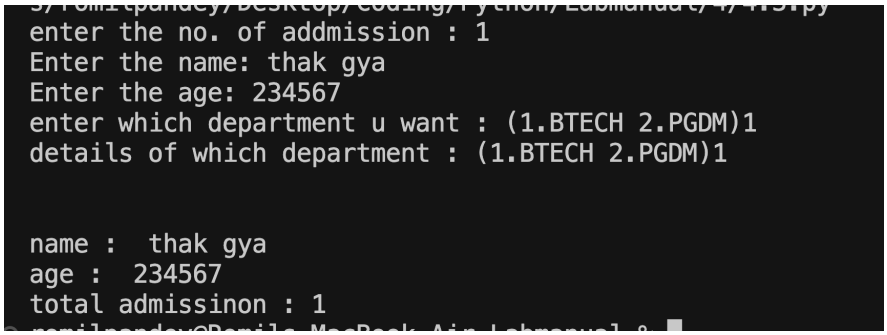
```
# Write a program to admit the students in the different
# Departments(pgdm/btech)and count the students. (Class, Object and
Constructor).

class details:
    count=0
    bcount=0
    pcount=0
    def getdata(self):
        name=input("Enter the name: ")
        age=int(input("Enter the age: "))

        self.name=name
        self.age=age
        d=int(input("enter which department u want : (1.BTECH
2.PGDM)"))
        self.d=d

        details.count+=1

        if self.d==1:
            details.bcount+=1
        else:
            details.pcount+=1
```

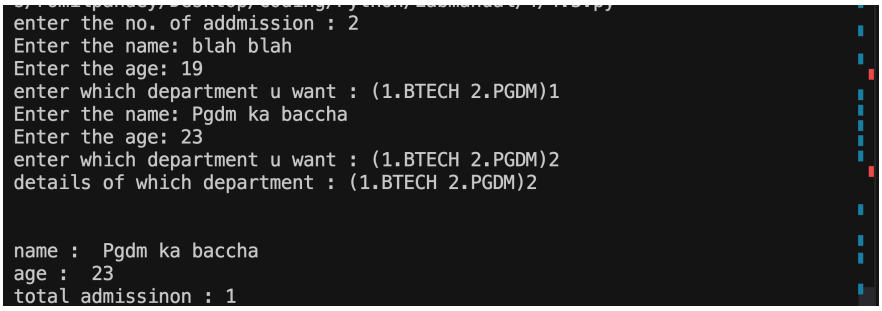
```
    def display(self):
        if self.d==1:
            print('\n\nname : ',self.name,'\nage : ',self.age)
            print(f"total admissinon : {details.bcount}")

        elif self.d==2:
            print('\n\nname : ',self.name,'\nage : ',self.age)
            print(f"total admissinon : {details.pcount}")

objs=list()

n=int(input("enter the no. of addmission : "))

for i in range(n):
    objs.append(details())

for i in range(n):
    objs[i].getdata()

e=int(input("details of which department : (1.BTECH 2.PGDM)"))

if e==1:
    for i in range(n):
        if objs[i].d==1:
            objs[i].display()
else:
    for i in range(n):
        if objs[i].d==2:
            objs[i].display()
```

OUTPUT:

```
enter the no. of addmission : 1
Enter the name: sujal
Enter the age: 18
enter which department u want : (1.BTECH 2.PGDM)1
details of which department : (1.BTECH 2.PGDM)1


name :  sujal
age :  18
total admissinon : 1
```

TEST CASE:





CONCLUSION:

- *This Python program demonstrates the use of classes, objects, and constructors to manage student admission details.*
- *The program efficiently counts and displays student admissions in different departments (BTech and PGDM).*
- *The use of a list of objects allows for the dynamic management of student details based on user input.*

Experiment No: 4.4

TITLE:

THEORY:

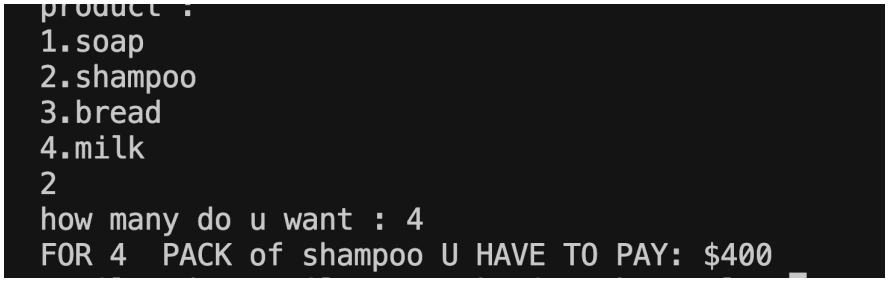
- *The program defines a class* `store` *to manage product records and generate bills.*
- *The class includes private attributes* `__itemCode` *and* `__price` *to store product details.*
- *The* `setdata` *method allows the user to input product details such as the type of product and quantity.*
- *The* `getdata` *method calculates and displays the total cost based on the entered product and quantity.*

CODE:

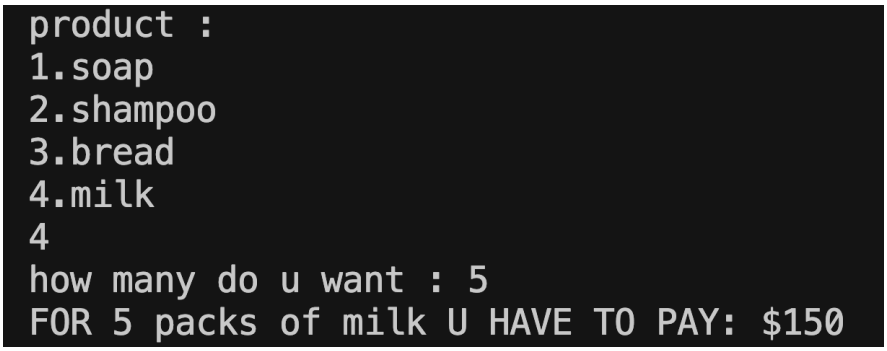
```
# Write a program that has a class store which keeps the record of
code and
# price of product display the menu of all product and prompt to
enter the quantity of
# each item required and finally generate the bill and display the
total amount.


class store:
    __itemcode=0
    __price=0

    def setdata(self):

        a=int(input("product : \n1.soap
\n2.shampoo\n3.bread\n4.milk\n"))
        self.a=a

        if self.a==1:
            self.__itemCode=1
            price=self.__price=10
            self.pirce=price
            n=int(input("how many do u want : "))
            self.n=n
```

```
        elif self.a==2:
            self.__itemCode=2
            price=self.__price=100
            self.pirce=price
            n=int(input("how many do u want : "))
            self.n=n

        elif self.a==3:
            self.__itemCode=3
            price=self.__price=40
            self.pirce=price
            n=int(input("how many do u want : "))
            self.n=n

        elif self.a==4:
            self.__itemCode=4
            price=self.__price=30
            self.pirce=price
            n=int(input("how many do u want : "))
            self.n=n

        else:
            print("INVALID INPUT")


    def getdata(self):

        if self.a==1:
            print(f"FOR {self.n}  PACKS of sope U HAVE TO PAY: $
{self.n*self.pirce}")

        if self.a==2:
            print(f"FOR {self.n}  PACK of shampoo U HAVE TO PAY: $
{self.n*self.pirce}")

        if self.a==3:
            print(f"FOR {self.n} PACK of bread U HAVE TO PAY: $
{self.n*self.pirce}")

        if self.a==4:
            print(f"FOR {self.n} packs of milk U HAVE TO PAY: $
{self.n*self.pirce}")


obj=store()
obj.setdata()
obj.getdata()
```

OUTPUT:



TEST CASE:



CONCLUSION:

- *This Python program demonstrates the use of a class to manage a store's product records and generate bills.*
- *The class efficiently sets and retrieves product details, calculating the total cost based on user input.*
- *The program outputs the total cost for the selected product and quantity, providing a clear billing system.*

Experiment No: 4.5

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

THEORY:

- *The program defines a parent class* `parent` *with a method* `add` *to perform addition.*
- *The child class* `child` *inherits from the parent class and includes a method* `takevalue` *to take input for two numbers.*
- *The child class utilizes the* `add` *method from the parent class to perform addition.*

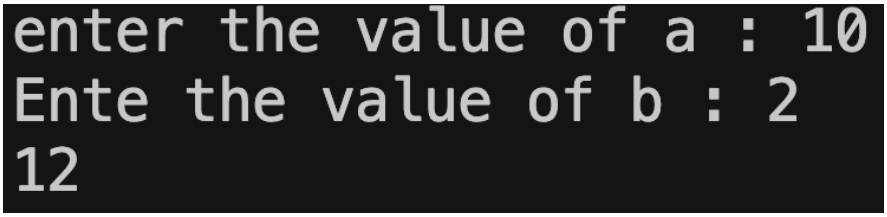
CODE:

```
class parent:
    def add(self):
        print(self.a+self.b)

class child(parent):
    def takevalue(self):
        self.a=int(input("enter the value of a : "))
        self.b=int(input("Ente the value of b : "))

c=child()
c.takevalue()
c.add()
```

OUTPUT:

```
enter the value of a : 10
Ente the value of b : 20
30
```

TEST CASE:



CONCLUSION:

- *This Python program demonstrates single inheritance, where the child class inherits from the parent class.*
- *The program efficiently takes input for two numbers in the child class and performs addition using the* add *method from the parent class.*
- *The use of inheritance allows for code reuse and logical organization of functionality.*

Experiment No:4.6

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

THEORY:

- *The program defines three classes:* `Lu`, `ITM`, *and* `Child`.
- *Both* `Lu` *and* `ITM` *classes contain attributes related to subjects, trainers, and durations for courses.*
- *The* `Child` *class inherits from both* `Lu` *and* `ITM` *classes.*
- *The* `Child` *class has methods* `setsub` *to set the subjects based on user input and* `getsub` *to display information about the selected subjects, including trainers and durations.*

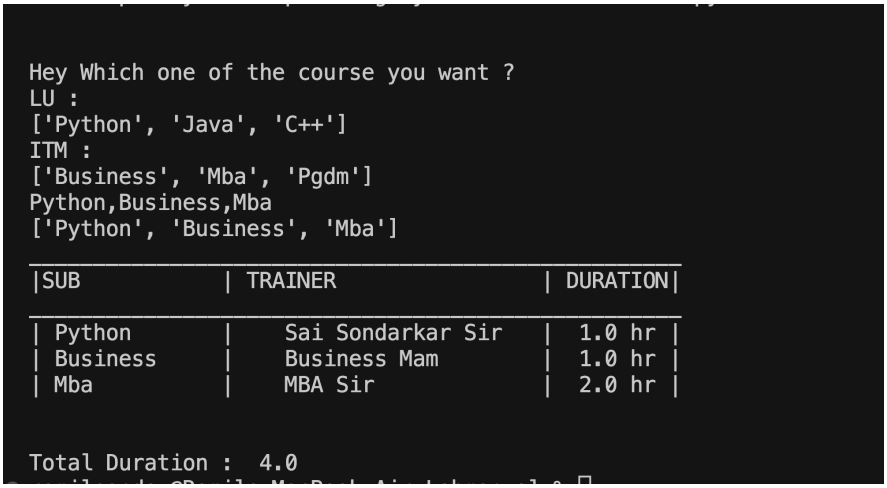
CODE:

```
class Lu:
    def __init__(self):
        self._sub = ["Python","Java","C++"]
        self._trainer = ["Sai Sondarkar Sir","Sumit Sir","Sheetal
Ma'am"]
        self._duration = [1.0,2.0,1.5]

class ITM:
    def __init__(self):
        self._sub1 = ["Business" ,"Mba" ,"Pgdm" ]
        self._trainer1 = ["Business Mam" , "MBA Sir" , "Pgdm Mam"]
        self._duration1 = [1.0,2.0,3.0]


class Child(ITM,Lu):

    def __init__(self):
        Lu.__init__(self)
        ITM.__init__(self)

    def setsub(self):
        print("\n\nHey Which one of the course you want ? ","LU :
",self._sub,"ITM : ",self._sub1,sep="\n")
        self.subj = (input()).split(",")
        for i in range(0,len(self.subj)):
```

```
            a = ""
            for j in range(0,len(self.subj[i])):

                if(self.subj[i][j] == self.subj[i][j].upper()):
                    if(not(j == 0)):
                        a += self.subj[i][j].lower()
                    else:
                        a+= self.subj[i][j]
                else:
                    if(j == 0):
                        a += self.subj[i][j].upper()
                    else:
                        a += self.subj[i][j]
            self.subj[i] = a

        print(self.subj)



    def getsub(self):
        totalhour=  0

print("_____________________________________________\n|SUB\t
| TRAINER\t\t| DURATION|
\n_____________________________________________")


        for i in self.subj:

            index = None
            if(i in self._sub ):
                index = self._sub.index(i)
                print("|" , i ,((8-len(i))*" "),"   |   " ,
self._trainer[index], ((17-len(self._trainer[index]))*" "), " |
",self._duration[index],"hr" , "|")
                totalhour += self._duration[index]
            else:
                index = self._sub1.index(i)
                print("|" , i ,((8-len(i))*" "),"   |   " ,
self._trainer1[index] ,((17-len(self._trainer1[index]))*" "), " |
",self._duration1[index],"hr", "|")
                totalhour += self._duration1[index]

        print("\n\nTotal Duration : ",totalhour)
```

```
a = Child()
a.setsub()
a.getsub()
```

OUTPUT:



TEST CASE:

```
s/romilpandey/Desktop/Coding/Python/Labmanual/474.6.py


Hey Which one of the course you want ?
LU :
['Python', 'Java', 'C++']
ITM :
['Business', 'Mba', 'Pgdm']
Java,C++
['Java', 'C++']
_____________________________________________
|SUB           | TRAINER                | DURATION|
_____________________________________________
| Java         |    Sumit Sir           |  2.0 hr |
| C++          |    Sheetal Ma'am       |  1.5 hr |


Total Duration :  3.5
romilpandey@Romils-MacBook-Air Labmanual %
```

```
Hey Which one of the course you want ?
LU :
['Python', 'Java', 'C++']
ITM :
['Business', 'Mba', 'Pgdm']
Business,Mba,Java
['Business', 'Mba', 'Java']
_____________________________________________________
|SUB           | TRAINER                | DURATION|
_____________________________________________________
| Business     |    Business Mam        |  1.0 hr |
| Mba          |    MBA Sir             |  2.0 hr |
| Java         |    Sumit Sir           |  2.0 hr |


Total Duration :  5.0
```

CONCLUSION:

- *This Python program demonstrates multiple inheritance, where* `DerivedClass` *inherits from both* `LU` *and* `ITM` *classes.*
- *The use of* `super()` *allows the derived class to access and invoke methods from both parent classes.*
- *The program creates an object of* `DerivedClass`*, displays information using the overridden* `display_info` *method, and invokes the overridden* `method`*.*

Experiment No: 4.7

TITLE:
Write a program to implement Multilevel inheritance, Grandfather→Father-→Child to show property inheritance from grandfather to child.

THEORY:

- *The program defines four classes:* `Grandfather`, `Father`, `Husband`, *and* `Child`.
- *Each class represents a family member with specific attributes related to inheritance and purchases.*
- `Father` *and* `Child` *classes inherit from the* `Grandfather` *class, and* `Child` *also inherits from* `Husband`.
- *The classes have initialization methods (*`__init__`*) to set the attributes based on inheritance and purchases.*
- *The* `Child` *class overrides the* `__init__` *method to collect additional information for the child.*
- *The* `Child` *class has a* `getdata` *method to display information about the child's assets.*

CODE:

```
class Grandfather:
    def __init__(self):
        self.gfname = " John  "
        self.sername = " wick"
        self.gfinherita = 5000
        self.gfperchased = 5000
        self.gfasset = self.gfinherita + self.gfperchased
```

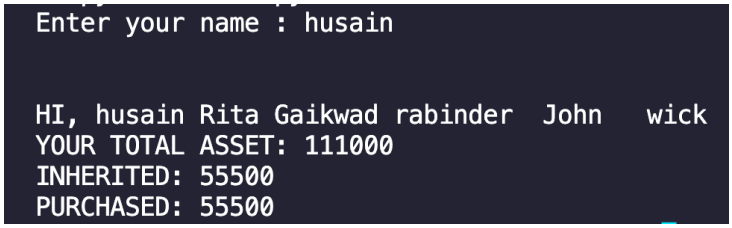
```
class Father(Grandfather):
    def __init__(self):
        super(Father, self).__init__()
        self.fname = " rabinder " + self.gfname + self.sername
        self.finherita = 500
```

```
        self.fperchased = 500
        self.fasset = self.finherita + self.fperchased


class Husband:
    def __init__(self):
        self.hname = " Rita"
        self.hsername = " Gaikwad"
        self.hinherita = 50000
        self.hperchased = 50000
        self.hasset = self.hinherita + self.hperchased


class Child(Father, Husband):
    def __init__(self):
        Father.__init__(self)
        Husband.__init__(self)
        self.cname = input("Enter your name : ")
        self.cname += self.hname + self.hsername + self.fname
        self.cinherita = self.finherita + self.gfinherita +
self.hinherita
        self.cperchased = self.fperchased + self.gfperchased +
self.hperchased
        self.casset = self.gfasset + self.fasset + self.hasset

    def getdata(self):
        print(f"\n\nHI, {self.cname}")
        print(f"YOUR TOTAL ASSET: {self.casset}")
        print(f"INHERITED: {self.cinherita}")
        print(f"PURCHASED: {self.cperchased}")


obj = Child()
obj.getdata()
```

OUTPUT:

```
Enter your name : husain


HI, husain Rita Gaikwad rabinder  John   wick
YOUR TOTAL ASSET: 111000
INHERITED: 55500
PURCHASED: 55500
```

TEST CASE:



CONCLUSION:

- *This Python program demonstrates multiple inheritance and attribute initialization in a family hierarchy.*
- *Each class represents a family member, and the attributes include inherited and purchased assets.*
- *The program creates an object of the* `Child` *class, collects information, and displays details about the child's assets.*

Experiment No:4.8

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:

- *The program defines three classes:* `LibraryItem`*,* `Book`*, and* `DVD`*.*
- `LibraryItem` *is the base class with attributes for title, item ID, and checked-out status, along with methods for displaying information, checking out, and checking in.*
- `Book` *and* `DVD` *are subclasses of* `LibraryItem` *with additional attributes (*`author` *for* `Book` *and* `director` *and* `duration` *for* `DVD`*).*
- *Each subclass overrides the* `display_info` *method to include the specific information for that type of library item.*

CODE:

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

    def display_info(self):
        print(f"Title: {self.title}")
        print(f"Item ID: {self.item_id}")
        print(f"Checked Out: {'Yes' if self.checked_out else
'No'}")

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

    def check_in(self):
```

```
        if self.checked_out:
            print(f"Checking in {self.title}")
            self.checked_out = False
        else:
            print(f"{self.title} is not checked out.")


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

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


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

    def display_info(self):
        super().display_info()
        print(f"Director: {self.director}")
        print(f"Duration: {self.duration} minutes")


# Example usage
book1 = Book("The Catcher in the Rye", "B001", "J.D. Salinger")
dvd1 = DVD("Inception", "D001", "Christopher Nolan", 148)

book1.display_info()
book1.check_out()
book1.display_info()
book1.check_in()
book1.display_info()

print("\n")

dvd1.display_info()
dvd1.check_out()
dvd1.display_info()
dvd1.check_in()
dvd1.display_info()
```

OUTPUT:

```
Author: J.D. Salinger


Title: Inception
Item ID: D001
Checked Out: No
Director: Christopher Nolan
Duration: 148 minutes
Checking out Inception
Title: Inception
Item ID: D001
Checked Out: Yes
Director: Christopher Nolan
Duration: 148 minutes
Checking in Inception
Title: Inception
Item ID: D001
Checked Out: No
Director: Christopher Nolan
```

TEST CASE:

```
Author: J.D. Salinger
Checking out The Catcher in the Rye
Title: The Catcher in the Rye
Item ID: B001
Checked Out: Yes
Author: J.D. Salinger
Checking in The Catcher in the Rye
Title: The Catcher in the Rye
Item ID: B001
Checked Out: No
Author: J.D. Salinger


Title: The Dark Knight
Item ID: D002
Checked Out: No
Director: Christopher Nolan
Duration: 148 minutes
Checking out The Dark Knight
Title: The Dark Knight
Item ID: D002
Checked Out: Yes
Director: Christopher Nolan
Duration: 148 minutes
Checking in The Dark Knight
Title: The Dark Knight
Item ID: D002
Checked Out: No
Director: Christopher Nolan
Duration: 148 minutes
```

CONCLUSION:

- *This Python program demonstrates the use of inheritance to model a library system with different types of items (books and DVDs).*
- *The base class* `LibraryItem` *provides common functionality, and subclasses* `Book` *and* `DVD` *extend it with additional attributes and methods.*
- *The example usage at the end showcases creating instances, displaying information, checking out, and checking in items in a library.*

Experiment No: 5.1

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

THEORY:

- *The provided code demonstrates the use of modules in Python.*
- *A module is a file containing Python definitions and statements that can be reused in other Python scripts.*

CODE:

my_module

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

```
Main
import my_module as mm

print(mm.add(1,2))
```

OUTPUT:

TEST CASE:

CONCLUSION:

- *Modules in Python help organize code into reusable components.*
- *Importing modules allows the reuse of functions and variables defined in those modules.*
- *The code showcases a simple example of module usage, importing a function and using it in the main script.*

Experiment No: 5.2

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:

- *The provided code represents a basic ATM simulation using a custom module named "Bank Module."*
- *The simulation allows users to perform operations such as withdrawal, checking balance, depositing money, and transferring money.*

CODE:

Bank Module - :

```
import time
```

```
import random
def get_account_number():
    while True:
        account_no = int(input("\nEnter your card number: \n"))
        if 10000000 <= account_no <= 99999999:
            return account_no
        else:
            print("Account number should be of 8 numbers :) \n\n")

def withdraw_money(balance):
    while True:
        money = float(input("\nEnter the money you want to withdraw
₹"))
        if money > balance:
            time.sleep(1)
            print("\nYour balance is lower than the amount you want
to withdraw")
        elif money < 100:
            print("Minimal amount should be ₹100")
        else:
            return money

def deposit_money(balance):
```

```
    money = float(input("\nEnter the amount you want to deposit ₹"))
    return balance + money

def transfer_money(balance, account_no):
    while True:
        money = float(input("\nEnter the amount you want to transfer : ₹"))
        ac = float(input("\nEnter the account you want to transfer to "))
        if int(ac) == int(account_no):
            print("\nCan't send money to yourself, can you? \n")
        elif not (10000000 <= ac < 99999999):
            print("\nAccount no. should be of 8 digits\n")
        elif money > balance or money < 100:
            if money > balance:
                print("Not enough money in your account \n")
            else:
                print("Minimal transfer amount is ₹100\n")
        else:
            time.sleep(2)
            balance -= money
            print(f"\nTransferred amount  ₹{money:.3f} To Account with account no: {int(ac)}\n")
            print(f"Your bank now has ₹{balance:.3f}")
            return balance
```

Main - :

```
import time

import random
from BANKmodule import *

def main():
    c = random.randint(1000, 10000)
    account_no = get_account_number()

    print("\n\nChecking your card status please wait :) \n")
    time.sleep(2)
    print("\nWELCOME TO ATM ")

    while True:
        print("\n\nWhat would you like to do?\n1. Withdrawal\n2. Check balance\n3. Deposit money\n4. Transfer money \n5. Cancel \n")
        n = int(input())
```

```
        if n == 1:
            c -= withdraw_money(c)
            time.sleep(2)
        elif n == 2:
            print(f"\nYour account has ₹{c:.3f}\n")
        elif n == 3:
            c = deposit_money(c)
            print("Successfully deposited ")

        elif n == 4:
            c = transfer_money(c, account_no)

        elif n == 5:
            print("\nTHANK YOU FOR CHOOSING US :) \n\n")
            return
        else:
            print("\nInvalid option :)\n")

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

OUTPUT:

```
ding/Python/Labmanual/5727atm.py

Enter your card number:
12345678


Checking your card status please wait :)


WELCOME TO ATM


What would you like to do?
1. Withdrawal
2. Check balance
3. Deposit money
4. Transfer money
5. Cancel

2

Your account has ₹9856.000



What would you like to do?
1. Withdrawal
2. Check balance
3. Deposit money
4. Transfer money
```

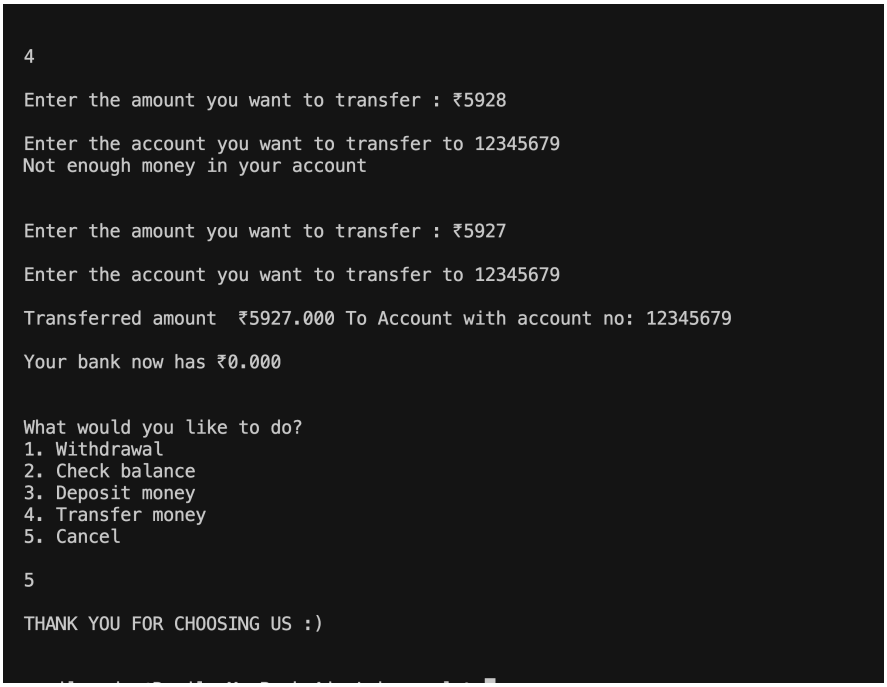
```
5. Cancel

4

Enter the amount you want to transfer : ₹1000

Enter the account you want to transfer to 12345678

Can't send money to yourself, can you?


Enter the amount you want to transfer : ₹1000

Enter the account you want to transfer to 12345679

Transferred amount  ₹1000.000 To Account with account no: 12345679

Your bank now has ₹8856.000


What would you like to do?
1. Withdrawal
2. Check balance
3. Deposit money
4. Transfer money
5. Cancel

5

THANK YOU FOR CHOOSING US :)
```

```
What would you like to do?
1. Withdrawal
2. Check balance
3. Deposit money
```

```
ding/Python/Labmanual/5/2/atm.py

Enter your card number:
123456
Account number should be of 8 numbers :)



Enter your card number:
12345678


Checking your card status please wait :)


WELCOME TO ATM


What would you like to do?
1. Withdrawal
2. Check balance
3. Deposit money
4. Transfer money
5. Cancel

2

Your account has ₹6028.000
```

TEST CASE:

```
Your account has ₹6028.000



What would you like to do?
1. Withdrawal
2. Check balance
3. Deposit money
4. Transfer money
5. Cancel

6

Invalid option :)



What would you like to do?
1. Withdrawal
2. Check balance
3. Deposit money
4. Transfer money
5. Cancel

1

Enter the money you want to withdraw  ₹99
Minimal amount should be ₹100

Enter the money you want to withdraw  ₹101
4. Transfer money
5. Cancel

1

Enter the money you want to withdraw  ₹1000


What would you like to do?
1. Withdrawal
2. Check balance
3. Deposit money
4. Transfer money
5. Cancel

4

Enter the amount you want to transfer : ₹1000

Enter the account you want to transfer to 12345678
```


4
Enter the amount you want to transfer : ₹5928
Enter the account you want to transfer to 12345679
Not enough money in your account
Enter the amount you want to transfer : ₹5927
Enter the account you want to transfer to 12345679
Transferred amount ₹5927.000 To Account with account no: 12345679
Your bank now has ₹0.000
What would you like to do?
1. Withdrawal
2. Check balance
3. Deposit money
4. Transfer money
5. Cancel
5
THANK YOU FOR CHOOSING US :)

CONCLUSION:

- *The code demonstrates a simple ATM simulation with functionalities like withdrawal, balance check, deposit, and money transfer.*
- *It incorporates error handling for various scenarios, such as insufficient balance and invalid input.*
- *The modular approach of creating a custom "Bank Module" enhances code organization and reusability.*

Experiment No: 5.2

TITLE:

THEORY:

CODE:

Cars folder -> Init.py

```
from BMW import BMW
from AUDI import AUDI
from NISSAN import NISSAN
```

AUDI.PY

```
class AUDI:
    def __init__(self, model):
        self.model = model

    def start_engine(self):
        print(f"{self.model} is starting the engine.")
```

```
Enter the money you want to withdraw  ₹99
Minimal amount should be ₹100

Enter the money you want to withdraw  ₹101


What would you like to do?
1. Withdrawal
2. Check balance
3. Deposit money
4. Transfer money
5. Cancel

2

Your account has ₹5927.000



What would you like to do?
1. Withdrawal
2. Check balance
3. Deposit money
4. Transfer money
5. Cancel

4

Enter the amount you want to transfer : ₹5928

Enter the account you want to transfer to 12345679
```

```
    def drive(self):
        print(f"{self.model} is on the move.")
```

BMW.PY

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

    def start_engine(self):
        print(f"{self.model} is starting the engine.")

    def drive(self):
        print(f"{self.model} is on the move.")
```

NISSAN.PY

```
class NISSAN:
    def __init__(self, model):
        self.model = model

    def start_engine(self):
        print(f"{self.model} is starting the engine.")

    def drive(self):
        print(f"{self.model} is on the move.")
```

MAIN.PY

```
from cars import BMW, AUDI, NISSAN

def main():
    bmw_car = BMW("BMW X5")
    audi_car = AUDI("Audi A4")
    nissan_car = NISSAN("Nissan Altima")

    bmw_car.start_engine()
    bmw_car.drive()

    audi_car.start_engine()
    audi_car.drive()

    nissan_car.start_engine()
    nissan_car.drive()

if __name__ == "__main__":
```

```
main()
```

OUTPUT:

TEST CASE:

```
BMW X5 is starting the engine.
BMW X5 is on the move.
Audi A4 is starting the engine.
Audi A4 is on the move.
Nissan Altima is starting the engine.
Nissan Altima is on the move.
```

CONCLUSION:

- *The code demonstrates a simple representation of three car models using classes in Python.*
- *Each car class encapsulates its specific behavior (starting the engine and driving) through methods.*
- *By organizing the code into classes, it promotes code reuse and encapsulation, enhancing maintainability and readability.*

Experiment No:6

TITLE:

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

THEORY:

- *The provided code showcases the usage of the* `threading` *module in Python to create and run two threads concurrently.*
- *The code defines two functions,* `hey` *and* `hi`*, each printing a message.*
- *Two threads,* `t1` *and* `t2`*, are created with the* `Thread` *class from the* `threading` *module, each targeting one of the functions.*
- *The* `start` *method is called on each thread, initiating their execution concurrently.*

CODE:

```
import threading

def hey():
```

```
    print("Hey")
def hi():
    print("Hi")


t1 = threading.Thread(target=hey)
t2 = threading.Thread(target=hi)

t1.start()
t2.start()
```

OUTPUT:

CONCLUSION:

- *The code demonstrates a basic example of multithreading in Python using the* `threading` *module.*
- *Multithreading allows for the concurrent execution of different tasks, improving program efficiency.*
- *The* `start` *method initiates the execution of threads, and the functions* `hey` *and* `hi` *run concurrently.*

Experiment No: 7

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:

- *The provided code is a simple Python script that retrieves weather information for a given city using the OpenWeatherMap API.*
- *It uses the* `requests` *library to make HTTP requests to the OpenWeatherMap API and fetch weather details.*
- *The* `colorama` *library is used to add color to the console output.*

CODE:

```
import requests
from colorama import Fore

ask = input("Enter the city Name you want to know weather? ")
response = requests.get(f"http://api.openweathermap.org/geo/1.0/
direct?q={ask},IND&limit=2&appid=7cfa155d846b68092d142ae9a8f74534")

a = response.json()

while True:
    try:
        lat = a[0]['lat']
        lon = a[0]['lon']
        break
    except:
        print(f"{ask} City Not Found Please type the name correctly
:)")
        ask = input("Enter the city Name you want to know weather?
")
        response = requests.get(f"http://api.openweathermap.org/
geo/1.0/direct?
q={ask},IND&limit=2&appid=7cfa155d846b68092d142ae9a8f74534")
        a = response.json()


weather = requests.get(f"https://api.openweathermap.org/data/2.5/
weather?lat={lat}&lon={lon}
&units=metric&appid=7cfa155d846b68092d142ae9a8f74534")
weather_detail = weather.json()
```

```
print(f"\n{Fore.WHITE}Name : {Fore.GREEN}{ask}\t\t{Fore.WHITE}
Weather : {Fore.GREEN}{weather_detail['weather'][0]['main']} ,
{Fore.GREEN}{weather_detail['weather'][0]['description']}
\n{Fore.WHITE}Division  : {Fore.GREEN}{weather_detail['name']}
\t\t{Fore.WHITE}Temp (°C) : {Fore.GREEN}{weather_detail['main']
['temp']}\n{Fore.WHITE}Feels as if (°C): {Fore.GREEN}
{weather_detail['main']['feels_like']}\t\t{Fore.WHITE}Humidity :
{Fore.GREEN}{weather_detail['main']['humidity']}")
```

OUTPUT:

```
Enter the city Name you want to know weather? Navi Mumbai

Name : Navi Mumbai                Weather : Smoke , smoke
Division  : Navi Mumbai           Temp (°C) : 30.97
Feels as if (°C): 32.77           Humidity : 51
```

TEST CASE:

```
sunrise': 1704159710, 'sunset': 1704199325}, 'timezone': 19800, 'id': 8131499, 'name': 'Konkan Divis
ion', 'cod': 200}

Name : mumbai
Weather : Haze , haze
Division  : Konkan Division
Temp (°C) : 27
Feels like (°C): 28.15
Humidity : 61
husainhakim@Husains-MacBook-Air python  % /usr/bin/python3 "/Users/husainhakim/python /saisirprogram
s.py/weatherapi.py"
/Users/husainhakim/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:34: NotOpenSSLWar
ning: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL
 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
  warnings.warn(
Enter the city name to check the weather:-delhi
{'coord': {'lon': 77.2219, 'lat': 28.6517}, 'weather': [{'id': 701, 'main': 'Mist', 'description': '
mist', 'icon': '50d'}], 'base': 'stations', 'main': {'temp': 11.06, 'feels_like': 10.21, 'temp_min':
 11.06, 'temp_max': 11.06, 'pressure': 1020, 'humidity': 76}, 'visibility': 1100, 'wind': {'speed':
```

CONCLUSION:

- *The script demonstrates how to use the OpenWeatherMap API to fetch weather information for a specific city.*
- *It incorporates user input, error handling (checking if the city is found), and presents weather details in a formatted manner with colored text in the console.*

Experiment No: 7.2

TITLE:

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

THEORY:

- *The provided code is a Python script that interacts with the CoinGecko API to fetch real-time cryptocurrency prices.*
- *It uses the* `requests` *library to make HTTP requests to the CoinGecko API and retrieve cryptocurrency price information.*
- *The user is prompted to enter the name of a cryptocurrency, and the script displays its price in USD and INR.*

CODE:

```
import requests
api_id="CG-MVwfDPbzcds9onoar9oKSaqf"
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(response.status_code == 200):
        while int(input("Press any key to see updates or \"0\" to
exit")) != 0:
            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()
            for i in a:
                print("₹",a[i]['inr'])
        break
    else:
        print("Invalid currency name please retype")
```

OUTPUT:

```
Enter cryptocoin: Ruby
Press any key to see updates or "0" to exit1
₹ 3.88
Press any key to see updates or "0" to exit2
₹ 3.88
Press any key to see updates or "0" to exit3
₹ 3.88
Press any key to see updates or "0" to exit0
```

```
Enter cryptocoin: Ethereum
Press any key to see updates or "0" to exit1
₹ 190371
Press any key to see updates or "0" to exit2
₹ 190371
Press any key to see updates or "0" to exit3
₹ 190371
Press any key to see updates or "0" to exit0
```

```
Press any key to see updates or "0" to exit1
₹ 3542273
Press any key to see updates or "0" to exit2
₹ 3542273
Press any key to see updates or "0" to exit3
₹ 3542273
Press any key to see updates or "0" to exit2
₹ 3542273
Press any key to see updates or "0" to exit1
₹ 3542273
Press any key to see updates or "0" to exit1
₹ 3542273
Press any key to see updates or "0" to exit1
₹ 3542273
Press any key to see updates or "0" to exit1
₹ 3542273
```

TEST CASE:

CONCLUSION:

- *The script allows the user to check real-time cryptocurrency prices using the CoinGecko API.*
- *It provides an interactive way for the user to see updates continuously until they choose to exit by entering "0".*
- *The script handles errors by checking the status code of the API response and prompts the user to re-enter the cryptocurrency name if it is invalid.*