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

PYTHON PROGRAMMING LAB

Prepared by:

Name of Student: **ROSHNI RAI**

Roll No: **08**

Batch: 2023-27

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Exp.	List of Experiment No
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: Piyush Kumar Singh Roll Number: 43

Experiment No:1.1

Title: Write a program to compute Simple Interest.

Theory:

Where *P* is the principal amount, *R* is the rate of interest, and *T* is the time in years.

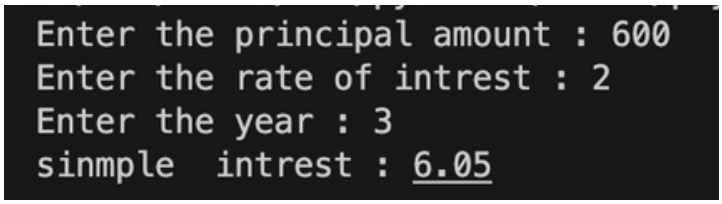
Code:-

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

p=float(input("Enter the principal amount : "))
r=float(input("Enter the rate of intrest : "))
t=float(input("Enter the year : "))

simple_intreat=(p+r+t)/100
print(f"sinmple intrest : {simple_intreat}")
```

Output:-

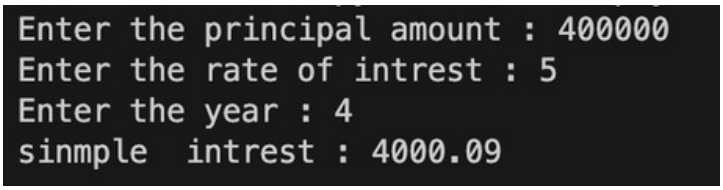
```
Enter the principal amount : 5000
Enter the rate of intrest : 5
Enter the year : 2
sinmple  intrest : 50.07
```

Caption



Caption

Test Case:-



Caption

Conclusion:-

- Simple Interest is straightforward to compute with the provided formula.
- The program accurately determines the Simple Interest based on user-input principal, rate, and time.

Experiment No:1.2

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

Theory:

- Arithmetic operators (+, -, *, /) perform basic mathematical operations.
- Relational operators (<, >, <=, >=, ==, !=) compare values and return true or false.

Code:

```
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))

# Arithmetic operations
addition_result = num1 + num2
subtraction_result = num1 - num2
multiplication_result = num1 * num2
division_result = num1 / num2
modulus_result = num1 % num2

print(f"Arithmetic Operations:")
print(f"{num1} + {num2} = {addition_result}")
print(f"{num1} - {num2} = {subtraction_result}")
print(f"{num1} * {num2} =
{multiplication_result}")
print(f"{num1} / {num2} = {division_result}")
print(f"{num1} % {num2} = {modulus_result}")

# Relational operations
equal_to = num1 == num2
not_equal_to = num1 != num2
```

```
greater_than = num1 > num2
less_than = num1 < num2
greater_than_or_equal_to = num1 >= num2
less_than_or_equal_to = num1 <= num2
```

```
print("\nRelational Operations:")
print(f"{num1} == {num2} : {equal_to}")
print(f"{num1} != {num2} : {not_equal_to}")
print(f"{num1} > {num2} : {greater_than}")
print(f"{num1} < {num2} : {less_than}")
```

```
Enter the first number: 10
Enter the second number: 9
Arithmetic Operations:
10.0 + 9.0 = 19.0
10.0 - 9.0 = 1.0
10.0 * 9.0 = 90.0
10.0 / 9.0 = 1.1111111111111112
10.0 % 9.0 = 1.0

Relational Operations:
10.0 == 9.0 : False
10.0 != 9.0 : True
10.0 > 9.0 : True
10.0 < 9.0 : False
10.0 >= 9.0 : True
10.0 <= 9.0 : False
```

Caption

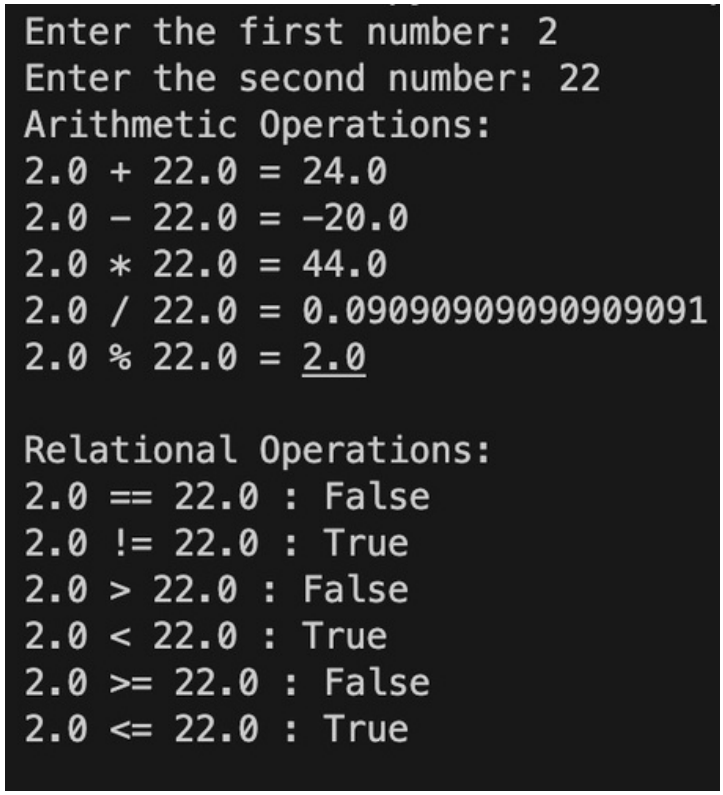
```
print(f"{num1} >= {num2} : {greater_than_or_equal_to}")
print(f"{num1} <= {num2} : {less_than_or_equal_to}")
```

Output:-

```
Enter the first number: 5
Enter the second number: 6
Arithmetic Operations:
5.0 + 6.0 = 11.0
5.0 - 6.0 = -1.0
5.0 * 6.0 = 30.0
5.0 / 6.0 = 0.8333333333333334
5.0 % 6.0 = 5.0

Relational Operations:
5.0 == 6.0 : False
5.0 != 6.0 : True
5.0 > 6.0 : False
5.0 < 6.0 : True
5.0 >= 6.0 : False
5.0 <= 6.0 : True
```

Caption

Test Case: -



Caption

Conclusion:-

- The program successfully handles arithmetic calculations.
- Relational operators effectively compare values, aiding decision-making in the program.

Experiment No:1.3

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

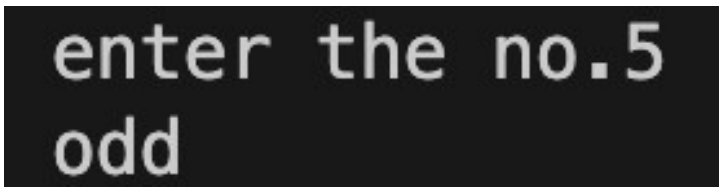
Theory:

- An even number is divisible by 2 without a remainder.
- An odd number has a remainder when divided by 2.

Code:

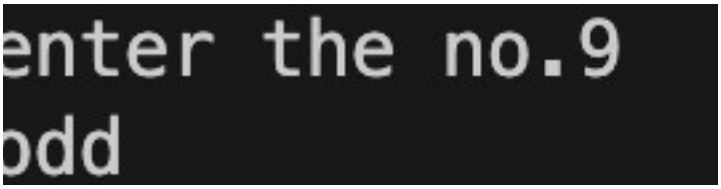
```
n=int(input("enter the no."))

if(n%2==0):
        print("even")
else:
print("odd")
```

Output:

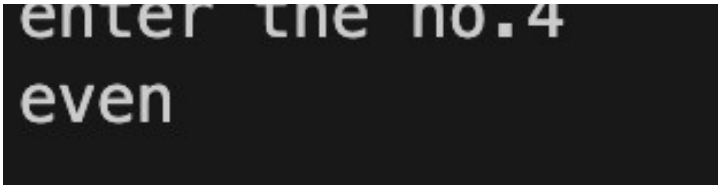


Caption

Test Case:



Caption



Caption

Conclusion:-

- The program accurately determines if a given number is even or odd.
- It utilizes the modulo operator to check for divisibility by 2, providing a clear result.

Experiment No:1.4

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

Theory:

- Natural numbers are positive integers starting from 1 up to n.
- The sum of the first n natural numbers is calculated using the formula

Code:-

```
# Write a program to print first n natural number & their sum.

n = int(input("Enter the value of n: "))

sum_of_numbers = 0

print(f"First {n} natural numbers:") for i in range(1, n + 1):
print(i, end=" ")
sum_of_numbers += i

print(f"\nSum of the first {n} natural numbers: {sum_of_numbers}")
```

Output:

```
Enter the value of n: 10
First 10 natural numbers:
1 2 3 4 5 6 7 8 9 10
Sum of the first 10 natural numbers: 55
```

Caption

Test Case:

```
Enter the value of n: 20
First 20 natural numbers:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Sum of the first 20 natural numbers: 210
```

```
Enter the value of n: 22
First 22 natural numbers:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
Sum of the first 22 natural numbers: 253
```

Caption

Caption

Conclusion:-

- The program effectively prints the first n natural numbers.

- It accurately calculates and displays the sum of the first n natural numbers.

Experiment No:1.5

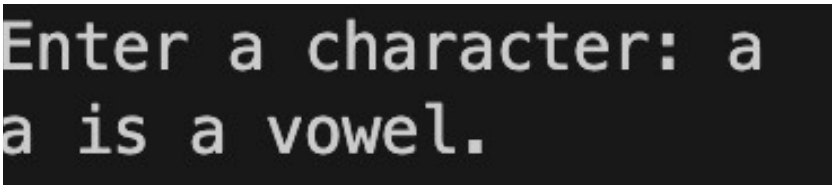
Title: Write a program to determine whether the character entered is a Vowel or not?

Theory:

- Vowels are the characters 'a', 'e', 'i', 'o', 'u' in the English alphabet.
- The program uses conditional statements to check if the entered character is a vowel.

Code:-

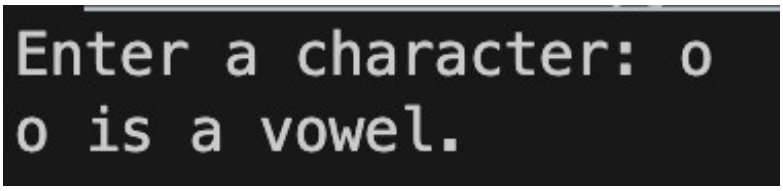
```
# Write a program to determine whether the character entered is a Vowel or not

char = input("Enter a character: ")

if char.lower() in ['a', 'e', 'i', 'o', 'u']:
print(f"{char} is a vowel.")
else:
print(f"{char} is not a vowel.")
```



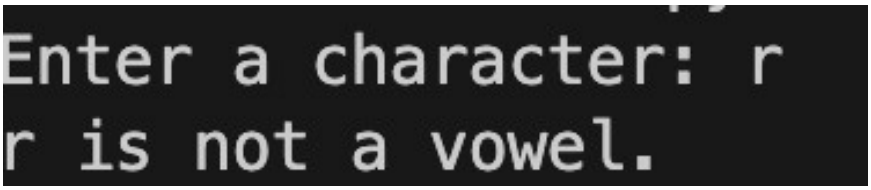
Caption

Output:

Test Case:



Caption



Caption

Conclusion:-

- The program correctly identifies whether the entered character is a vowel.
- It employs simple if-else conditions for a clear and accurate decision.

Experiment No:1.6

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

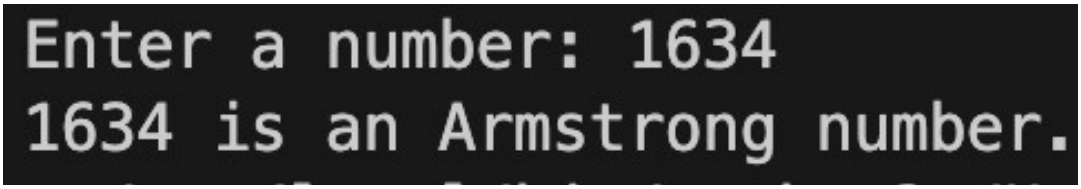
Theory:

- An Armstrong Number is equal to the sum of its own digits each raised to the power of the number of digits.
- The program extracts digits, raises them to the power, and checks for equality.

Code:-

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

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

num_str = str(num)
num_digits = len(num_str)

armstrong_sum = sum(int(digit) ** num_digits for digit in num_str)

if armstrong_sum == num:
    print(f"{num} is an Armstrong number.")
else:
    print(f"{num} is not an Armstrong number.")
```

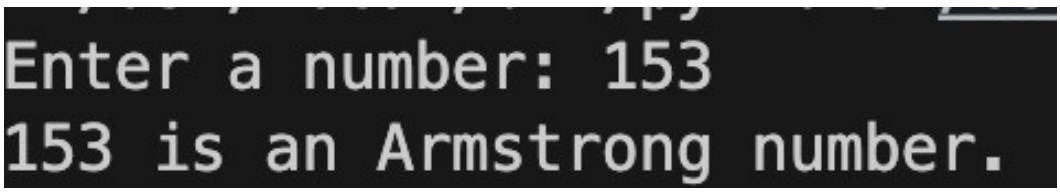
Output:



Caption

Test Case:

Caption



Caption

Conclusion:-

- The program accurately determines if the given number is an Armstrong Number.
- It uses a loop to extract digits, calculates the sum, and validates against the original number.

Experiment No:1.7

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

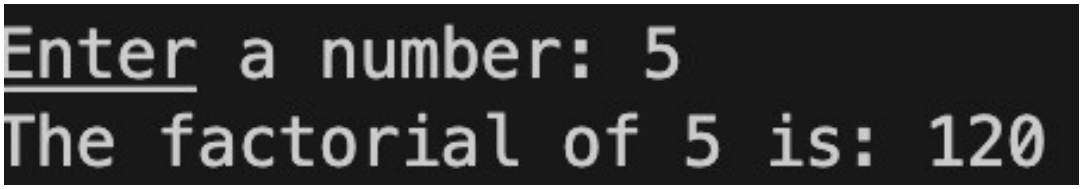
Theory:

- The factorial of a number is the product of all positive integers up to that number.
- A for loop iterates through numbers to calculate the factorial.

Code:

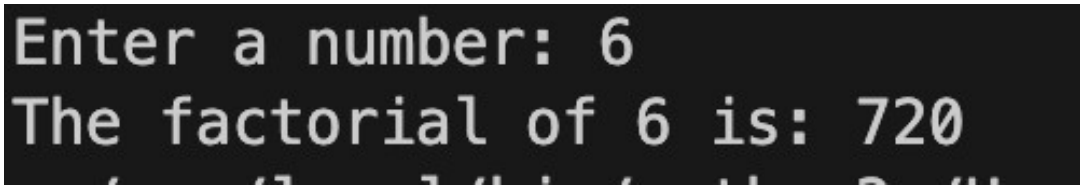
```
# Write a program using for loop to calculate factorial of a No.

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

factorial = 1

for i in range(1, num + 1):
factorial *= i

print(f"The factorial of {num} is: {factorial}")
```

Output:

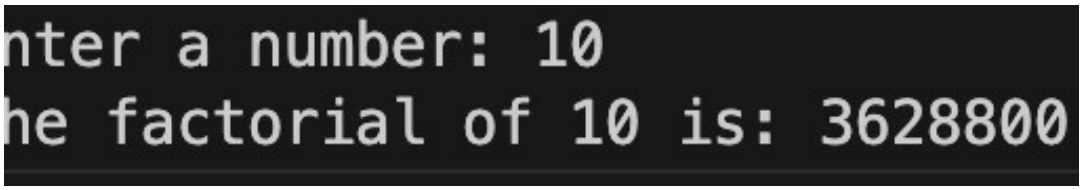


Caption

Test Case:



Caption



Caption

Conclusion:-

- The program successfully calculates the factorial of the given number.
- It uses a for loop to iterate and multiply, providing an accurate factorial result.

Experiment No:1.8

Title: Write a program to print the following pattern

Theory:

- i) Each row in the pattern consists of '*' repeated in increasing order.
- ii) Numerical values in each row increase sequentially, repeating the same digit.
- iii) The number of '*' in each row increases, forming a pyramid shape.

Code:

```
# Write a program to print the following pattern
# i) ii) iii)
# * 1 *
# * * 2 2 * * * # * * * 3 3 3 * * * * *
# * * * * 4 4 4 4 * * * * * * *
# * * * * * 5 5 5 5 5 * * * * * * * * *
```

```
n=int(input("Enter the no. : "))
```

```
for i in range (1,n+1,1):
for j in range(i):
print(" * ",end=" ")
print(" ")
```

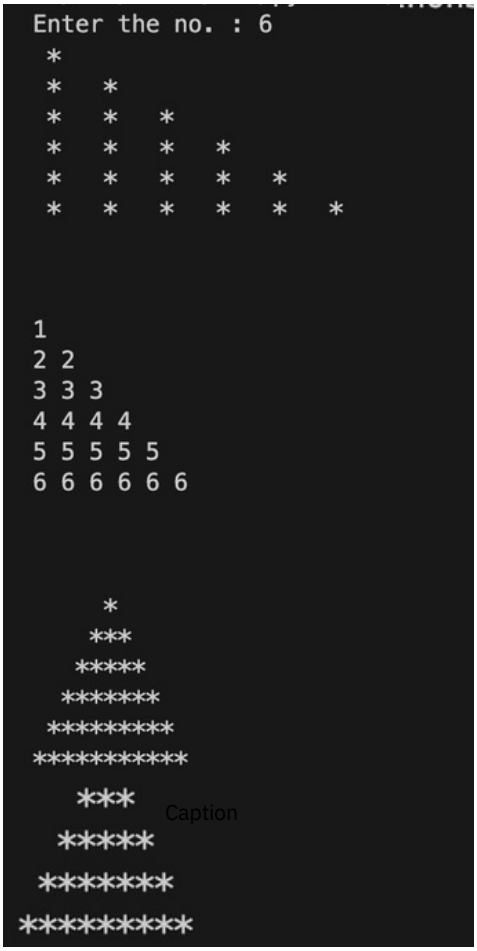
```
print("\n\n")

for i in range (1,n+1,1):
        for j in range(i):
                print( i ,end=" ")
print(" ")


print("\n\n")


for i in range(1,n+1):
            for j in range(n,i,-1):
print(" ",end="")
for k in range(2*i-1):
print("*",end="")

print("")
```

Output:

Test Case:

Conclusion:-

- i) The program successfully prints a pyramid pattern of '*'.
- ii) It accurately displays a pyramid pattern with repeating numerical values.
- iii) The program correctly generates a pyramid pattern with an increasing number of '*'.



Caption

Caption

Experiment No:2.1

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

Theory:

A list named **brics_countries** is created to store BRICS nations: Brazil, Russia, India, China, and South Africa.

Code:

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

print("BRICS Countries:")
for country in brics_countries:
print(country)
```

Output

Caption

Test Case:

Caption

Conclusion:-

- The program successfully defines and displays the list of BRICS countries.
- It's a straightforward representation of the BRICS alliance member nations.

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:

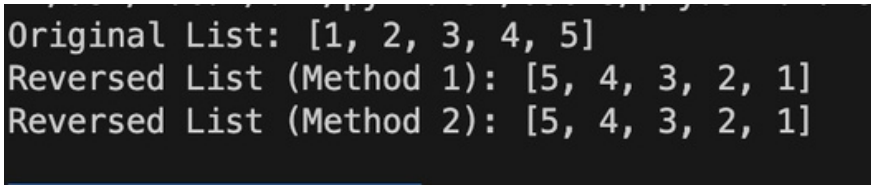
- The **reverse()** method is applied to the list, which reverses its elements in-place.
- Slicing with **[::-1]** is utilized to traverse the list in reverse order.

Code:

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


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


reversed_list_method1 = list(sample_list)
reversed_list_method1.reverse()


# Method 2: Using slicing
reversed_list_method2 = sample_list[::-1]
```

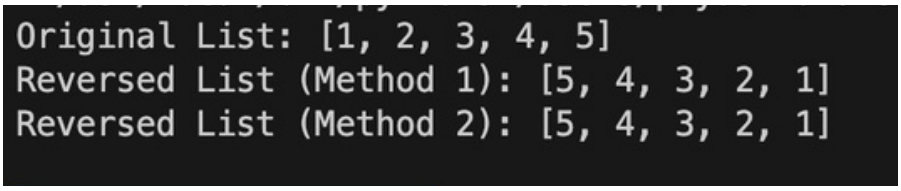
```
print("Original List:", sample_list)

print("Reversed List (Method 1):", reversed_list_method1)

print("Reversed List (Method 2):", reversed_list_method2)
```

Output:



Caption

Test Case:



Caption

Conclusion:-

- The program effectively reverses the list using the **reverse()** method.
- Slicing ensures a simple and efficient way to traverse the list in reverse.

Experiment No:2.3

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

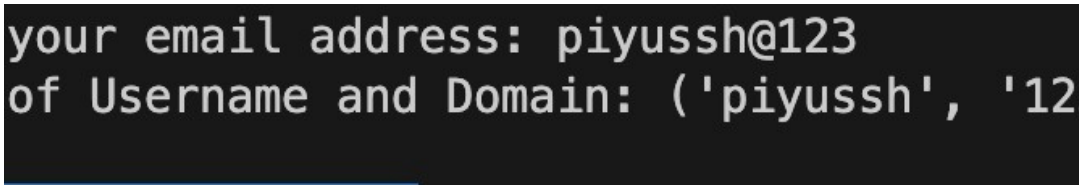
Theory:

he program takes an email address as input, splits it into username and domain using the '@' symbol, and creates a tuple with these components.

Code:

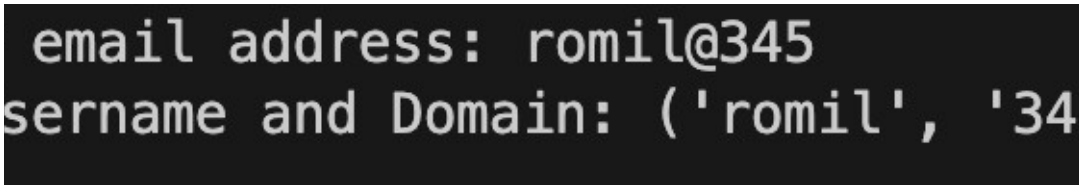
```
# Write a program that scans the email address and forms a tuple of username
# and domain.

email = input("Enter your email address: ")

username, domain = email.split('@')

email_tuple = (username, domain)

print("Tuple of Username and Domain:", email_tuple)
```

Output:

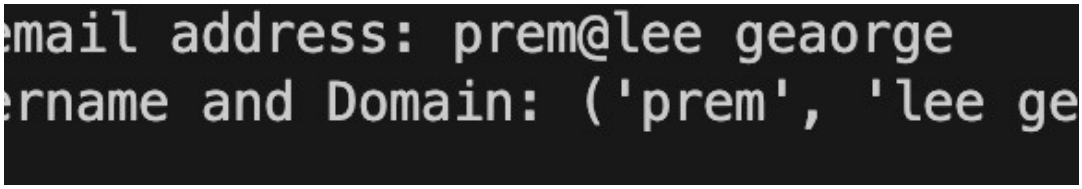


Caption

Test Case:



Caption



Caption

Conclusion:-

- The program successfully forms a tuple representing the username and domain of the provided email address.
- It uses basic string manipulation to achieve this.

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.

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

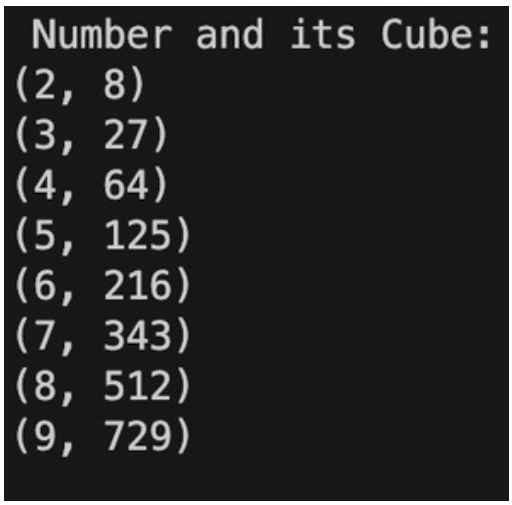
Theory:

The program utilizes list comprehension to create tuples with a number and its cube from a given list.

Code:

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

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

tuple_list = [(num, num**3) for num in c]

print(" Number and its Cube:")
for i in tuple_list:
print(i)
```

Output:



Caption

Test case :

```
 Number and its Cube:
(2, 8)
(3, 27)
(4, 64)
(5, 125)
(6, 216)
(7, 343)
(8, 512)
(9, 729)
```

Caption

Conclusion:-

- Successfully generates a list of tuples representing numbers and their cubes.
- Utilizes a concise and effective approach with list comprehension.

Experiment No:2.5

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

Theory:

The program compares two dictionaries in Python using the == operator, which checks if both dictionaries have the same key-value pairs.

Code:

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

dict1 = {'a': 1, 'b': 2, 'c': 3} dict2 = {'a':
1, 'b': 2, 'c': 3}

if dict1 == dict2:
        print("The dictionaries are equal.")
else:
        print("The dictionaries are not equal.")
```

Output: (screenshot)

Caption

Conclusion:

- The program accurately determines if two dictionaries are equal based on their key-value pairs using the == operator.
- It's a straightforward method for dictionary comparison.

Experiment No:2.6

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

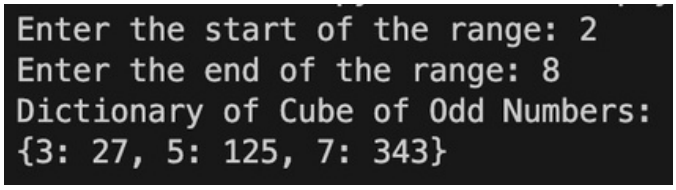
Theory:

The program utilizes dictionary comprehension to create a dictionary with cubes of odd numbers in a specified range.

Code:

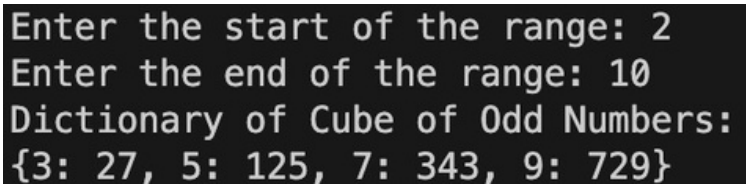
```
# Write a program that creates dictionary of cube of odd numbers in the range.

start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))

odd_cubes_dict = {num: num**3 for num in range(start_range, end_range + 1) if num % 2 != 0}

print("Dictionary of Cube of Odd Numbers:")
print(odd_cubes_dict)
```

Output:

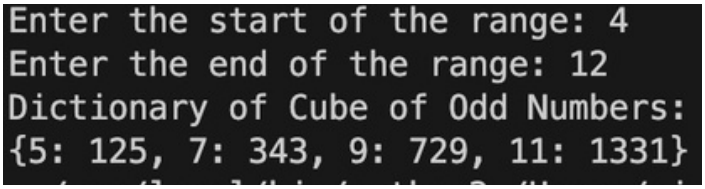


Caption

Test Case:



Caption



Caption

Conclusion:

- successfully generates a dictionary containing cubes of odd numbers within the given range.
- The program provides a concise solution using dictionary comprehension.

Experiment No:2.7

Title: Write a program for various list slicing operation.

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

Print Complete list

Print 4th element of list

Print list from0th to 4th index.

Print list -7th to 3rd element

Appending an element to list.

Sorting the element of list.

Popping an element.

Removing Specified element.

Entering an element at specified index.

Counting the occurrence of a specified element. Extending list.

Reversing the list.

Theory:

The program showcases list slicing and operations: accessing, slicing, appending, sorting, popping, removing, inserting, counting, extending, and reversing a list.

Code:

```
# 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.


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

# i. Print Complete list
print("\n\ni. Complete List:", a)

# ii. Print 4th element of list
print("\nii. 4th Element of the List:", a[3])

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

# iv. Print list -7th to 3rd element
print("\niv. List from -7th to 3rd Element:",
a[-7:4])

# v. Appending an element to list
a.append(110)
print("\nv. List after Appending 110:", a)
```

```
# vi. Sorting the elements of the list
a.sort()
print("\nvi. Sorted List:", a)

# vii. Popping an element
popped_element = a.pop()
print("\nvii. Popped Element:", popped_element)
print(" List after Popping:", a)

# viii. Removing a specified element
specified_element = 60
a.remove(specified_element)
print("\nviii. List after Removing",
specified_element, ":", a)

# ix. Inserting an element at specified index
index_to_insert = 2
element_to_insert = 15
a.insert(index_to_insert, element_to_insert)
print("\nix. List after Inserting", element_to_insert,
"at Index", index_to_insert, ":", a)

# x. Counting the occurrence of a specified
element
element_to_count = 20
count = a.count(element_to_count)
print("\nx. Count of", element_to_count, "in the
List:", count)

# xi. Extending list
extended_list = [120, 130, 140]
a.extend(extended_list)
print("\nxi. List after Extending with",
extended_list, ":", a)
```

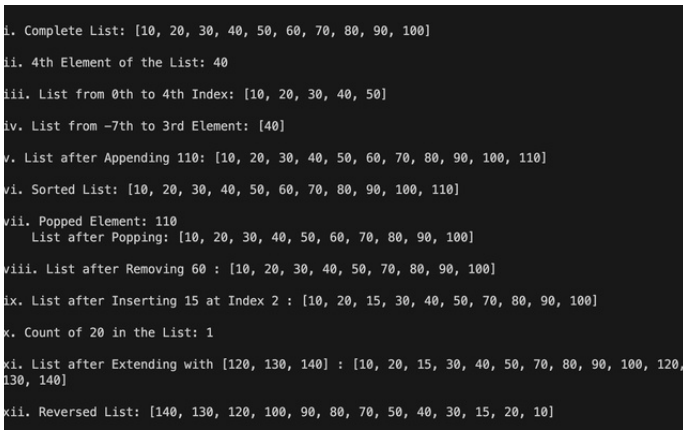
```python
# xii. Reversing the list
a.reverse()
print("\nxii. Reversed List:", a)
```

Output:

```
i. Complete List: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

ii. 4th Element of the List: 40

iii. List from 0th to 4th Index: [10, 20, 30, 40, 50]

iv. List from -7th to 3rd Element: [40]

v. List after Appending 110: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]

vi. Sorted List: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]

vii. Popped Element: 110
    List after Popping: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

viii. List after Removing 60 : [10, 20, 30, 40, 50, 70, 80, 90, 100]

ix. List after Inserting 15 at Index 2 : [10, 20, 15, 30, 40, 50, 70, 80, 90, 100]

x. Count of 20 in the List: 1

xi. List after Extending with [120, 130, 140] : [10, 20, 15, 30, 40, 50, 70, 80, 90, 100, 120,
130, 140]

xii. Reversed List: [140, 130, 120, 100, 90, 80, 70, 50, 40, 30, 15, 20, 10]
```

Caption

Test Case:



Caption

- The program performs various list operations, including slicing, appending, sorting, popping, removing, inserting, counting, extending, and reversing.
- Each operation is executed successfully, demonstrating diverse list manipulation techniques.

Experiment No:3.1

Title: Write a program to extend a list in python by using given approach.

i. By using + operator.

ii. By using Append ()

iii. By using extend ()

Theory:

- i) The + operator combines two lists to extend the original list.
- ii) The **append()** method adds a single element
- (list)
- to the end.

  iii) The **extend()** method adds individual elements to the end.

Code:

```
# 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 ()


original_list = [1, 2, 3]
```

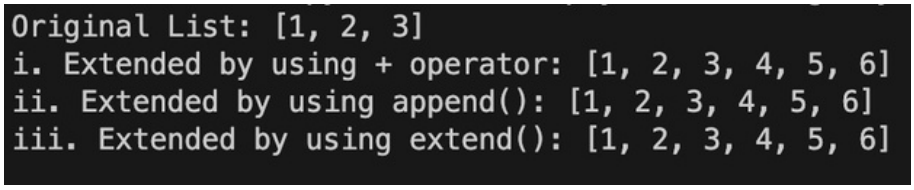
```
# i. By using + operator
list_extended_by_plus_operator = original_list + [4, 5, 6]

# ii. By using Append()
list_extended_by_append = original_list.copy()
list_extended_by_append.append(4)
list_extended_by_append.append(5)
list_extended_by_append.append(6)

# iii. By using extend()
list_extended_by_extend = original_list.copy()
list_extended_by_extend.extend([4, 5, 6])


print("Original List:", original_list) print("i. Extended by using + operator:", list_extended_by_plus_operator)
print("ii. Extended by using append():", list_extended_by_append)
print("iii. Extended by using extend():", list_extended_by_extend)
```

Output:

```
Original List: [1, 2, 3]
i. Extended by using + operator: [1, 2, 3, 4, 5, 6]
ii. Extended by using append(): [1, 2, 3, 4, 5, 6]
iii. Extended by using extend(): [1, 2, 3, 4, 5, 6]
```

Caption

Test Case:



Caption

Conclusion:

- i) + operator merges lists, creating a new extended list.
- ii) append() adds a sublist as a single element.
- iii) extend() adds elements individually, extending the list.

Experiment No:3.2

Title: Write a program to add two matrices.

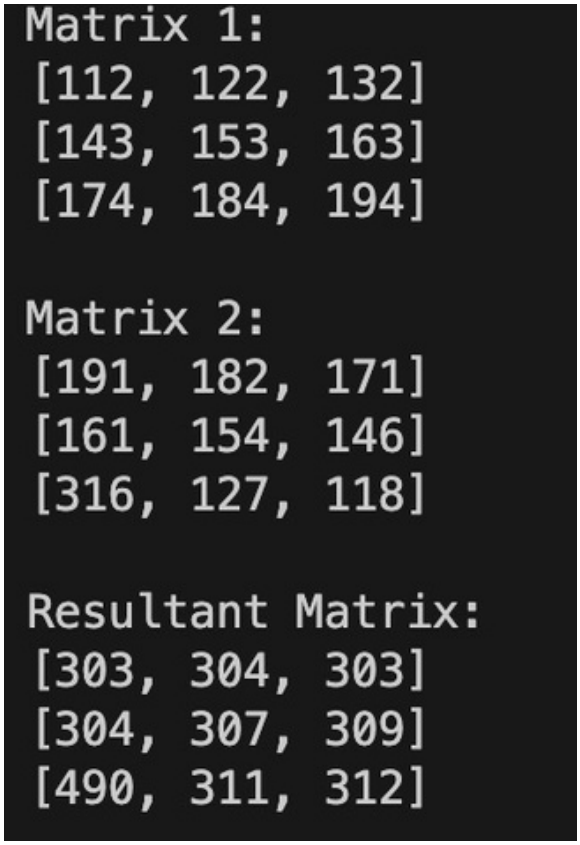
Theory:

Matrices are added element-wise, combining corresponding elements from two matrices to create a new matrix.

Code:

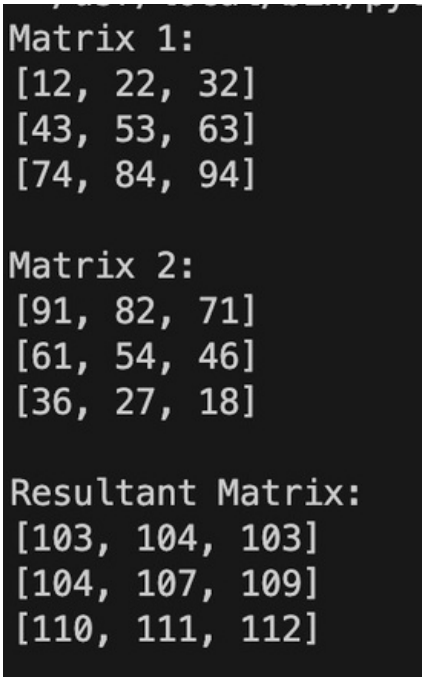
```
# Write a program to add two matrices.

matrix1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
matrix2 = [[9, 8, 7], [6, 5, 4], [3, 2, 1]]

# Add matrices
result_matrix = [[matrix1[i][j] + matrix2[i][j]
for j in range(len(matrix1[0]))] for i in
range(len(matrix1))]

# Print the matrices and the result
print("Matrix 1:")
for row in matrix1:
print(row)

print("\nMatrix 2:")
for row in matrix2:
print(row)

print("\nResultant Matrix:")
for row in result_matrix:
print(row)
```

Output:

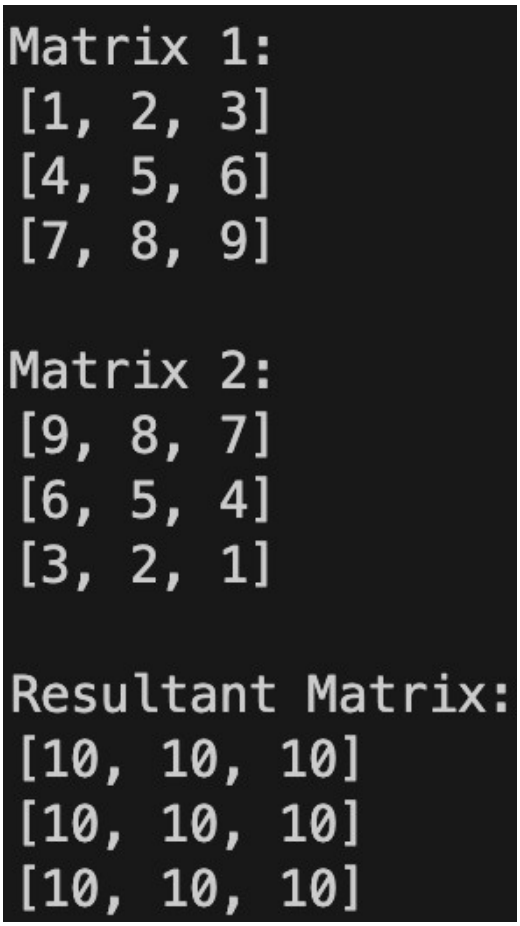


Caption

Test Case:



Caption


Matrix 1:
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
Matrix 2:
[9, 8, 7]
[6, 5, 4]
[3, 2, 1]
Resultant Matrix:
[10, 10, 10]
[10, 10, 10]
[10, 10, 10]

Caption

Conclusion:

- The program accurately adds two matrices, producing the result in a new matrix.
- Utilizes nested loops for efficient element-wise addition.

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.

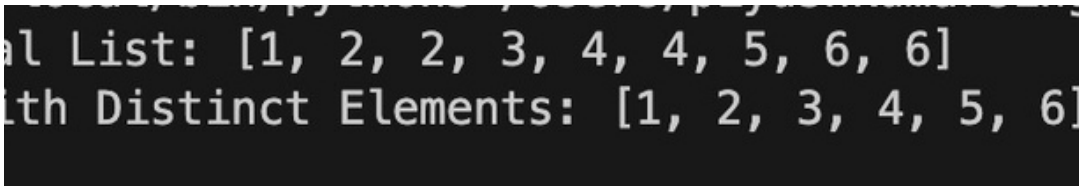
Theory:

The function employs the set data structure to obtain unique elements from a given list.

Code:

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

original_list = [1, 2, 2, 3, 4, 4, 5, 6, 6]

unique_list = list(set(original_list))

print("Original List:", original_list)
print("List with Distinct Elements:", unique_list)
```

Output:



Caption

Conclusion:

- Successfully generates a new list with distinct elements from the original list.
- Utilizing set ensures uniqueness, simplifying the removal of duplicates.

Experiment No:3.4

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

Theory:

A perfect number is one whose sum of proper divisors (excluding itself) equals the number.

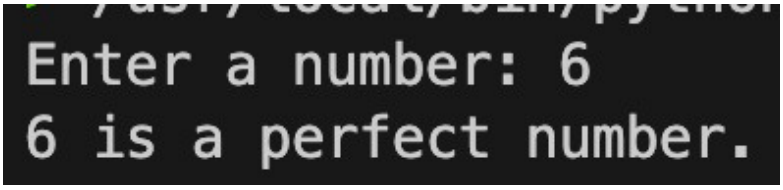
Code:

```
# Write a program to Check whether a number
is perfect or not.


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

if number <= 0:
print(f"{number} is not a perfect number.")
else:
        divisor_sum = sum([i for i in range(1, number)
if number % i == 0])
if divisor_sum == number:
print(f"{number} is a perfect number.")
else:
print(f"{number} is not a perfect
number.")
```

Output:

Caption

Test Case:



Caption

Caption

Conclusion:

- The program effectively checks if a given number is perfect
- It iterates through divisors, calculates their sum, and compares it to the original number.

Experiment No:3.5

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

string_test= 'Today is My Best Day'

Theory:

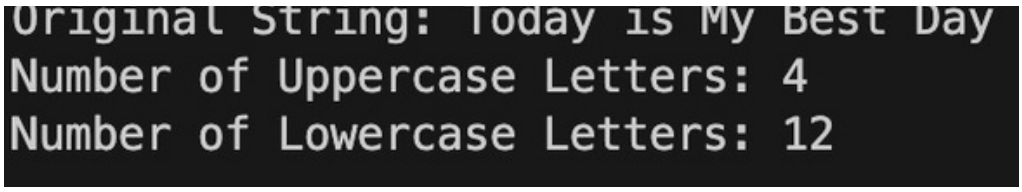
The function counts the number of upper and lower-case letters in a given string by iterating through each character and checking its case.

Code:

```
# 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'

def count_upper_lower(string):
upper_count = 0
lower_count = 0

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

return upper_count, lower_count
```

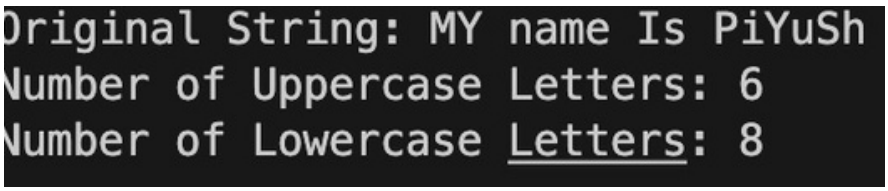
```
# Example usage:
string_test = 'Today is My Best Day'
upper, lower = count_upper_lower(string_test)

print("Original String:", string_test)
print("Number of Uppercase Letters:", upper)
print("Number of Lowercase Letters:", lower)
```

Output:



Caption

Test Case:



Caption

Conclusion:

- The function accurately counts the number of upper and lower-case letters in the provided string.

- It uses simple character case checks to achieve this.

Experiment No:4.1

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

Theory:

- The program defines an Employee class to encapsulate employee details.
- Methods are added to get employee details and print them.

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
```

Caption

Conclusion:

- The Employee class provides an organized structure for storing employee information.
- Methods facilitate easy access to employee details and printing.

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 utilizes a function with a combination of positional arguments (*args) and keyword arguments (**kwargs) to receive user input for name, email, and age.

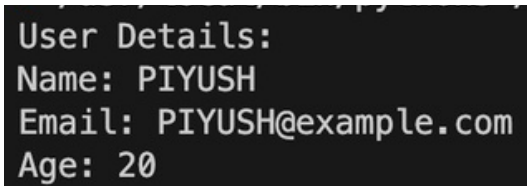
Code:

```
# 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}")
```

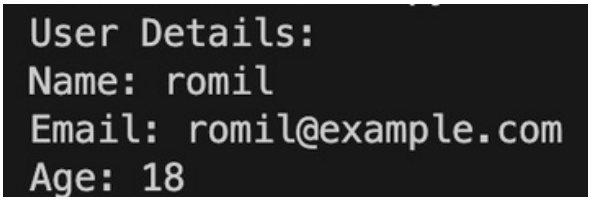
```
get_user_details(name="John    Doe",
email="john@example.com", age=25)
```

Output:

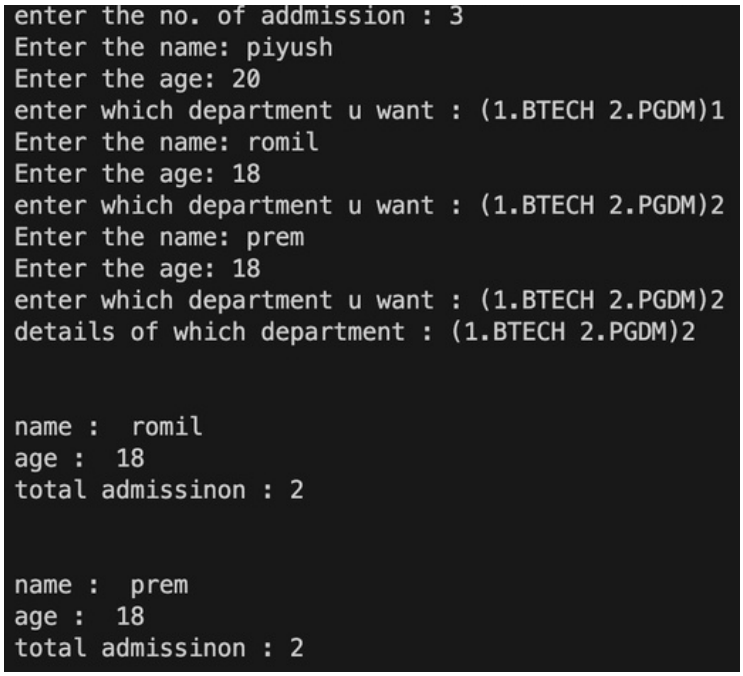


Caption

Test Case:



Caption



Caption

Conclusion:

- This approach allows flexibility in function calls, enhancing readability and accommodating various input scenarios.
- It simplifies user input handling in the program.

Experiment No:4.3

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

Theory:

The program employs classes, objects, and constructors to model students and their admission into different departments (PGDM/BTech).

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: piyush k. singh
Enter the age: 20
enter which department u want : (1.BTECH 2.PGDM)1
details of which department : (1.BTECH 2.PGDM)2
```

Caption

Test Case:

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


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

Caption

Conclusion:

- Object-oriented principles are applied, enhancing code organization and reusability.

- The program accurately counts and manages student admissions based on the specified departments.

Experiment No:4.4

Title:

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

Theory:

The program utilizes a class named 'Store' to manage product records, display a menu, take user input for quantities, and generate a bill with total amount.

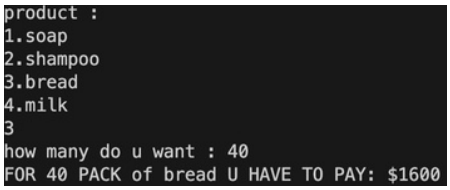
Code:

```
# 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}")
```

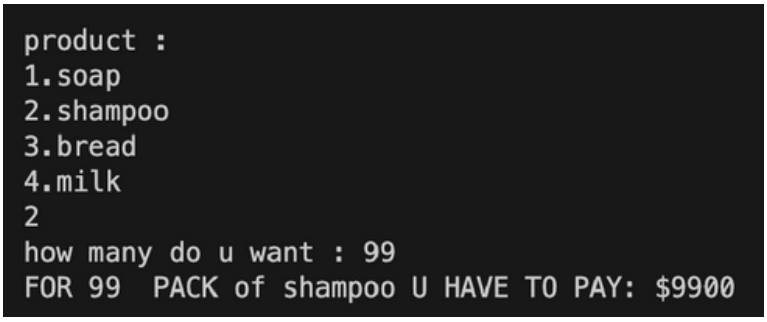
```
if self.a==2:
                    print(f"FOR {self.n} PACK of shampoo
U HAVE TO PAY: ${self.n*self.pirce}")

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:



Caption

Test Case:



Capttiion

Conclusion:

- Object-oriented approach enhances code organization for managing store operations.
- The program successfully calculates and displays the total amount based on user input quantities.

Experiment No:4.5

Title:

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

Theory:

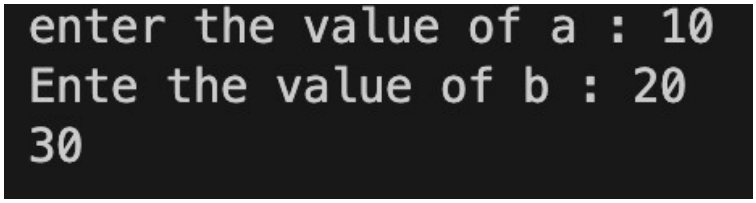
- Single inheritance is employed, where the **AddNumbers** class inherits from **InputNumbers** to reuse input methods for addition of two numbers.

Code:

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


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()
```

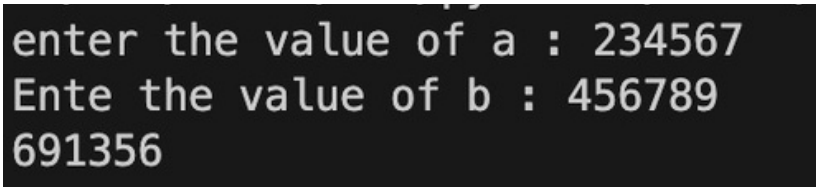
```
c.takevalue()
c.add()
```

Output:

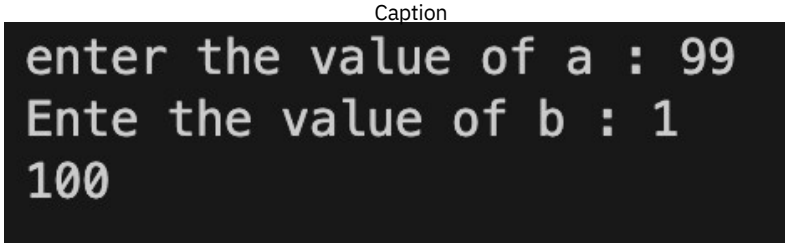


Caption

Test Case:



Caption



Caption

Conclusion:

- This approach enhances code reuse, promoting a clean and modular structure.

- The program successfully takes user input, performs addition, and displays the result.

Experiment No:4.6

Title:

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

Theory:

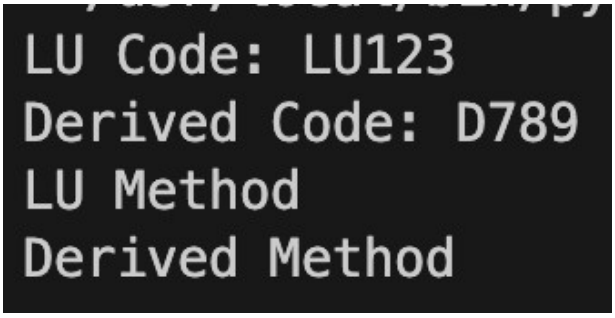
The program utilizes multiple inheritance with two base classes, **LU** and **ITM**, and one derived class inheriting from both.

Code:

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

class LU:
def __init__(self, lu_code):
self.lu_code = lu_code

def display_info(self):
print(f"LU Code: {self.lu_code}")

def method(self):
print("LU Method")


class ITM:
        def __init__(self, itm_code):
self.itm_code = itm_code
```

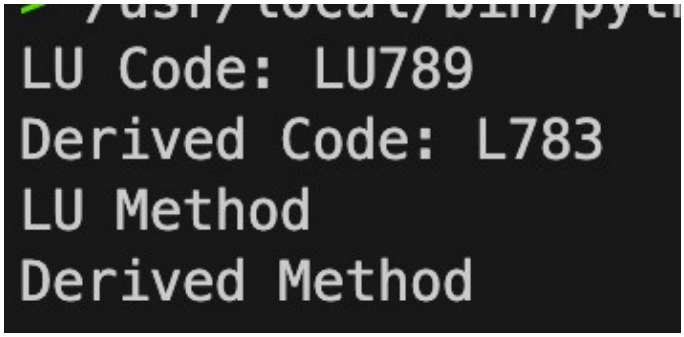
```
 def display_info(self):
                    print(f"ITM Code: {self.itm_code}")

def method(self):
print("ITM Method")


class DerivedClass(LU, ITM):
def __init__(self, lu_code, itm_code,
derived_code):
super().__init__(lu_code)
              self.derived_code = derived_code

def display_info(self):
                    super(DerivedClass, self).display_info()
print(f"Derived Code:
{self.derived_code}")

def method(self):
super(DerivedClass, self).method()
print("Derived Method")



derived_object    =    DerivedClass("LU123",    "ITM456",
"D789")

derived_object.display_info() derived_object.method()
```

Output:

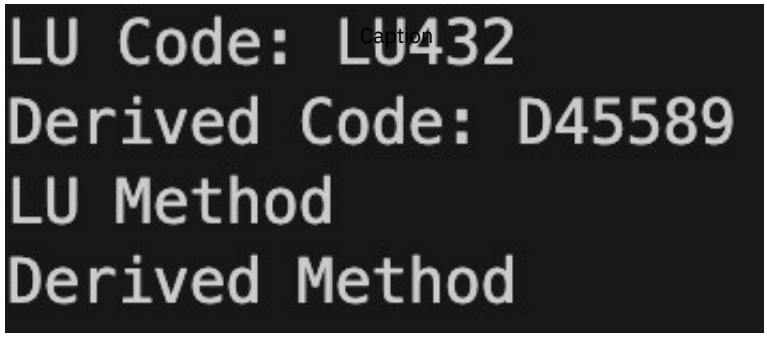


Caption

Test Case:



Caption



Caption

Conclusion:

- Multiple inheritance allows the derived class to inherit attributes and methods from both **LU** and **ITM**.
- The program demonstrates an efficient way to model relationships and share functionalities among classes.

Experiment No:4.7

Title:

Write a program to implement Multilevel inheritance,

Grandfather > Father >Child to show property inheritance from grandfather to child.

Theory:

Multilevel inheritance involves creating a chain of classes where a derived class inherits from a base class, and another class inherits from this derived class.

Code:

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

class Grandfather:
def __init__(self):
self.gfname = " Romil "
self.sername = " Pandey"
self.gfinherita = 5000
self.gfperchased = 5000
self.gfasset = self.gfinherita +
self.gfperchased


class Father(Grandfather):
def __init__(self):
                    super(Father, self).__init__()
```

```
 self.fname = " M.piyush " + self.gfname + self.sername
self.finherita = 500
self.fperchased = 500
self.fasset = self.finherita +
self.fperchased


class Husband:
def __init__(self):
            self.hname = " Ashlin"
self.hsername = " lee george"
            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 : prem


HI, prem Ashlin lee george M.piyush  Romil  Pandey
YOUR TOTAL ASSET: 111000
INHERITED: 55500
PURCHASED: 55500
```

Caption

Test Case:

```
Enter your name : Atharva


HI, Atharva  Ashlin lee george M.piyush  Romil  Pandey
YOUR TOTAL ASSET: 111000
INHERITED: 55500
PURCHASED: 55500
```

Caption

Conclusion:

- This programming structure promotes code reusability and establishes a clear hierarchy.

- The program successfully demonstrates property inheritance from Grandfather to Child in a multilevel inheritance scenario.

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 utilizes inheritance, creating a base class (**LibraryItem**) and derived classes (**Book**, **DVD**, **Journal**) for items in a library.

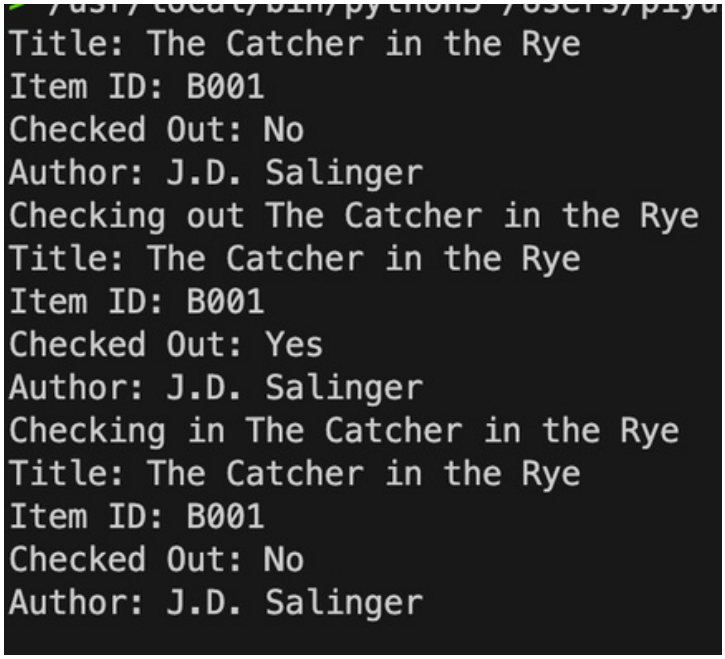
Code:

```
# 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)

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)
```

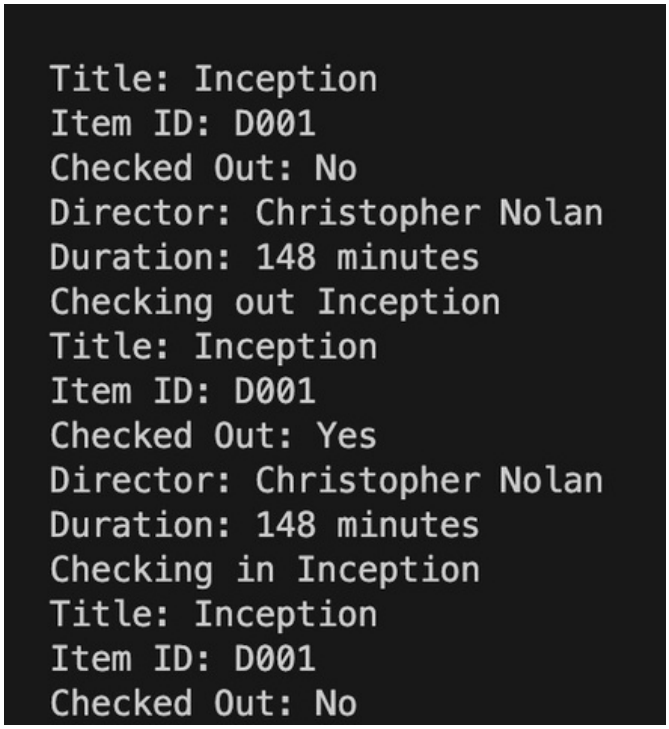
```
        self.director = director
        self.duration = duration

    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:



Caption

Test Case:



Caption

Conclusion:

- Inheritance provides a hierarchical structure, allowing common functionality in the base class and unique features in derived classes.
- Method overriding enhances the flexibility to customize behavior in each class, supporting a comprehensive library catalogue system with check-in and check-out functionality.

Experiment No:5.1

Title:

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

Theory:

Modules in Python are files containing Python definitions and statements. They can be imported into other scripts to reuse code.

Code:

```
def add():
    a=int(input("Enter the 1st no. : "))
    b=int(input("Enter the 2st no. : "))
return a+b
```

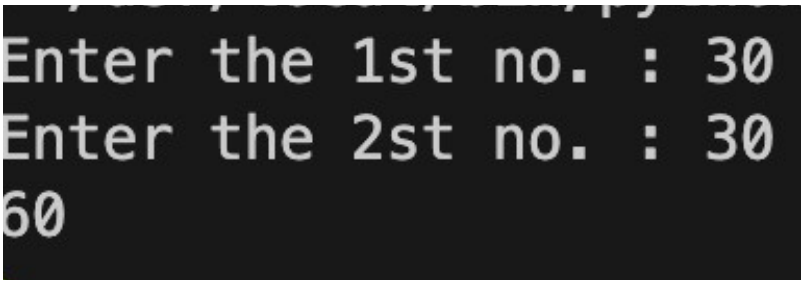
```
import my_module as

mm print(mm.add())
```

Output:

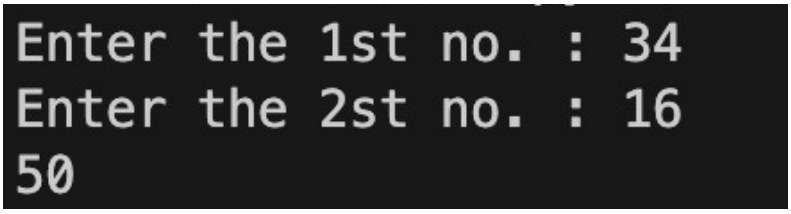
```
Enter the 1st no. : 4
Enter the 2st no. : 5
9
```

Caption

Test case :



Caption



Caption

Conclusion:

- Creating a module (**my_module**) allows the encapsulation of the addition function for reuse.

- Importing and using the module in the main script enhances code modularity and reusability.

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:

Modules in Python encapsulate code, promoting code organization and reusability. They can be imported into other scripts.

Code:

Module- import "me

```
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 ac == 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
```

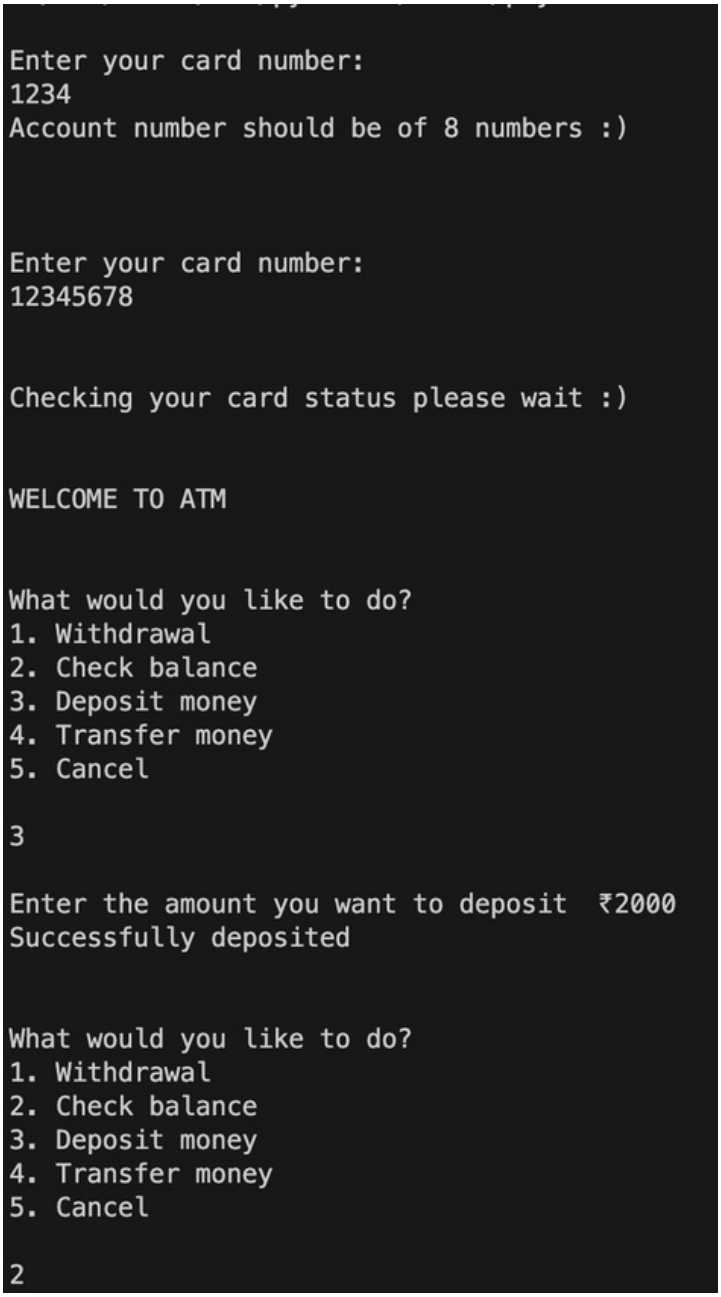
```
 print(f"\nTransferred amount ₹
{money:.3f} To Account with account no: {int(ac)}
\n")
print(f"Your bank now has ₹
{balance:.3f}")
return balance
```

Main- import time

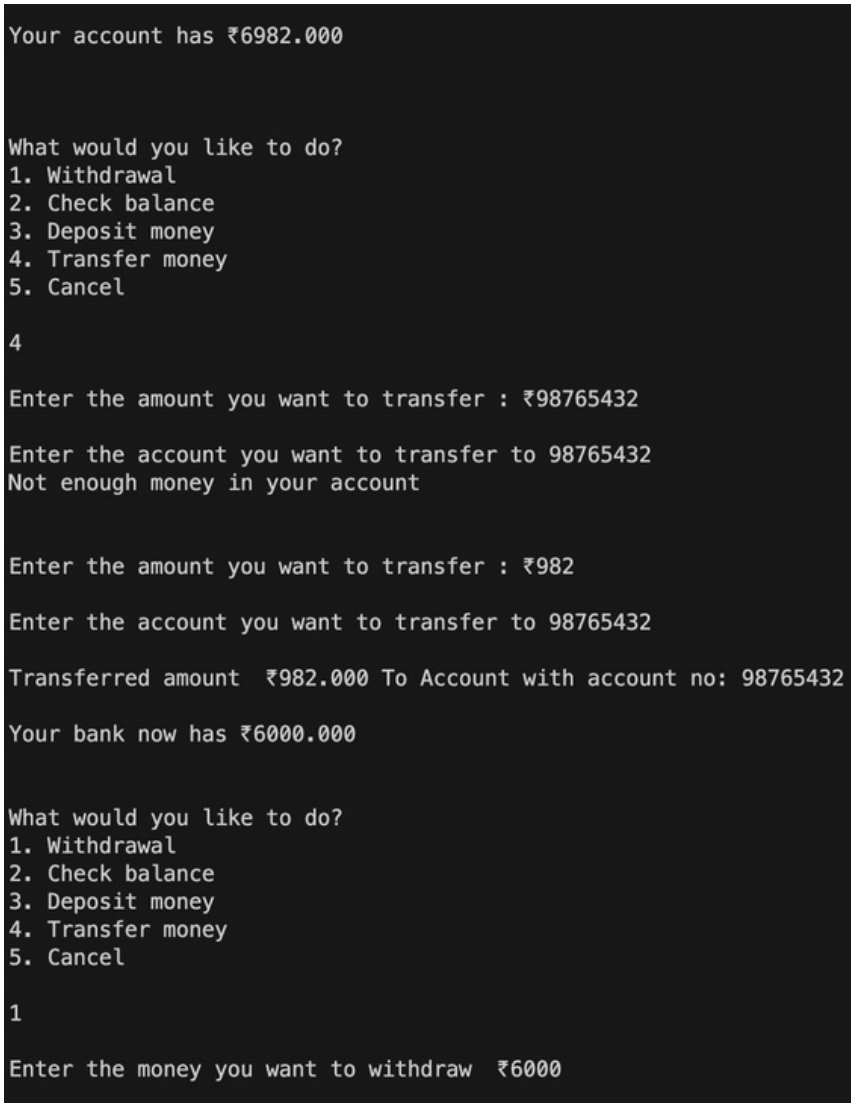
```
import random
from BANKmodule import *

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()
```

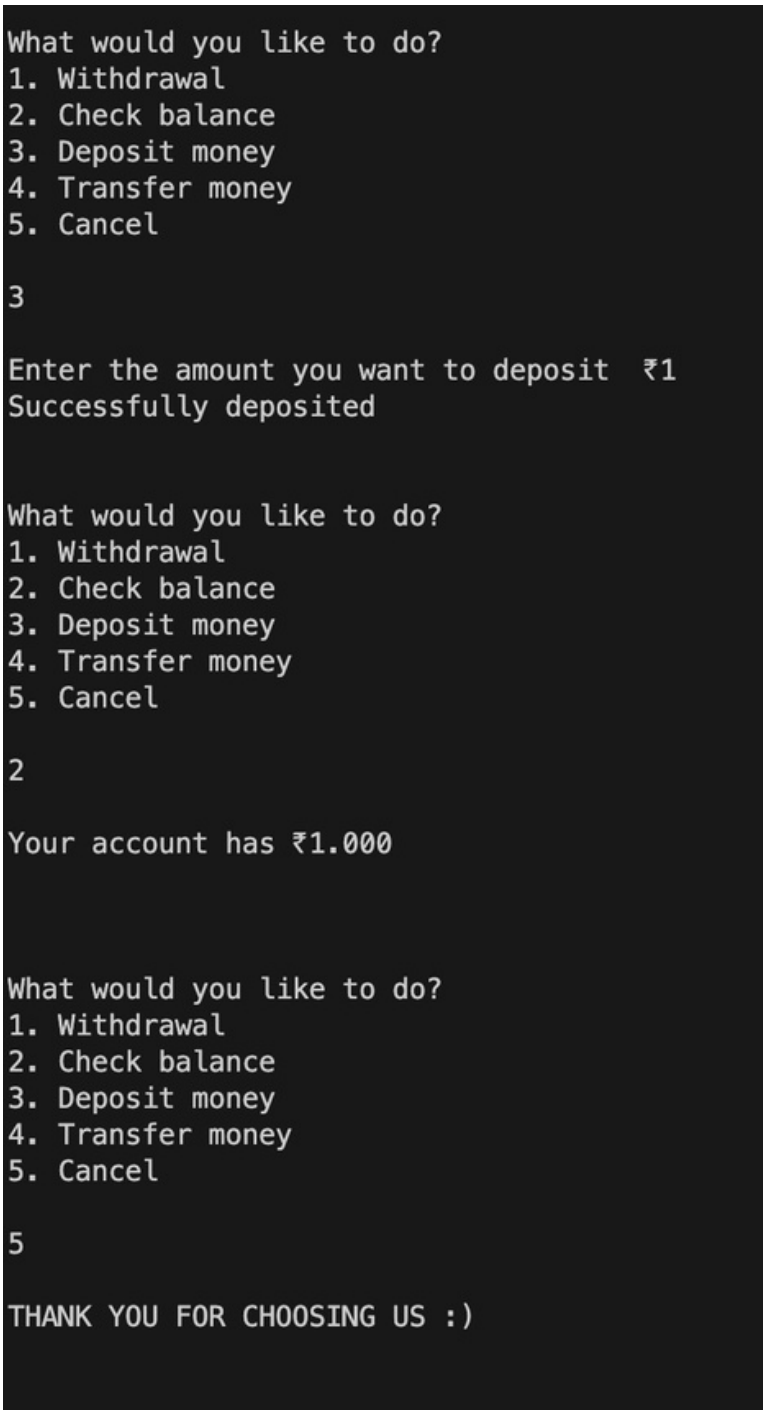
Output:


Enter your card number:
1234
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
3
Enter the amount you want to deposit ₹2000
Successfully deposited
What would you like to do?
1. Withdrawal
2. Check balance
3. Deposit money
4. Transfer money
5. Cancel
2

Caption


Your account has ₹6982.000
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 : ₹98765432
Enter the account you want to transfer to 98765432
Not enough money in your account
Enter the amount you want to transfer : ₹982
Enter the account you want to transfer to 98765432
Transferred amount ₹982.000 To Account with account no: 98765432
Your bank now has ₹6000.000
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 ₹6000

Caption


What would you like to do?
1. Withdrawal
2. Check balance
3. Deposit money
4. Transfer money
5. Cancel
3
Enter the amount you want to deposit ₹1
Successfully deposited
What would you like to do?
1. Withdrawal
2. Check balance
3. Deposit money
4. Transfer money
5. Cancel
2
Your account has ₹1.000
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 :)

Caption

Conclusion:

- The Bank Module provides a structured approach for basic banking operations.

- Importing and using the module in the main script facilitates easy access to bank account functionalities, enhancing code modularity and readability.

Experiment No:5.3

Title: Write a program to create a package with name cars and add different modules (such as BMW, AUDI, NISSAN) having classes and functionality and import them in main file cars.

Theory:

In Python, a package is a way of organizing related modules into a single directory hierarchy. Modules within a package can be accessed using the dot notation.

Code:

```
class AUDI:
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.")



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

def start_engine(self):
```

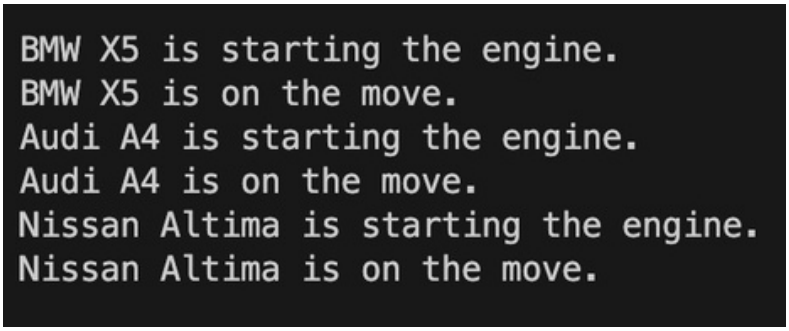
```
        print(f"{self.model} is starting the
engine.")

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

```
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.")
```

```
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:



Caption

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.
```

Caption

Conclusion:

- The program exemplifies the concept of packaging with the **cars** package containing modules for various car classes.
- Importing and utilizing these modules in the main script promotes code modularity, readability, and reusability.

Experiment No:6

Title:

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

Theory:

Multithreading is a concurrent execution mechanism where multiple threads operate independently within the same process, sharing resources like memory space.

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:

Caption

Test Case:

Caption

Conclusion:

- The program exemplifies multithreading by printing "Hello" and "Hi" concurrently.
- Multithreading enhances program efficiency by allowing simultaneous execution of tasks, improving responsiveness and performance.

Experiment No:7.1

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:

APIs (Application Programming Interfaces) allow different software systems to communicate with each other. Weather APIs provide access to weather-related data.

Code:

```
API_KEY='007cf1373b288f507576cc72315c98c8'

import requests
import datetime

city=input("Enter the city : ")

response=requests.get(f"https://
api.openweathermap.org/data/2.5/weather?q={city}
&APPID={API_KEY}&units=Metric")

a=response.json()
print(a)

if 'message' in a:
print("city not found !")

else:
print("\nCity:",city)
```

```
print("Temperature:",a['main']['temp'],"C")
print("Humidity:",a['main']['humidity'])

print("Sunrise(IST):",datetime.datetime.fromtimestamp(a['sys']['sunrise']))

print("Sunset(IST):",datetime.datetime.fromtimestamp(a['sys']['sunset']))
```

Output:

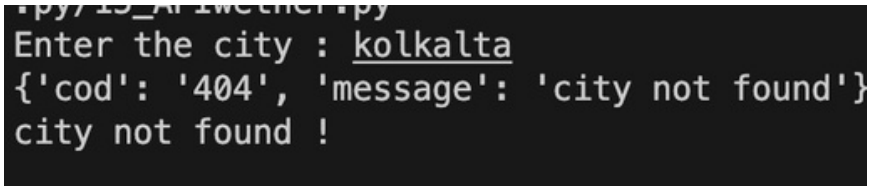
```
City: mumbai
Temperature: 30.99 C
Humidity: 48
Sunrise(IST): 2023-12-28 07:10:01
Sunset(IST): 2023-12-28 18:09:27
```

Caption

Test Case:

```
City: patna
Temperature: 23.96 C
Humidity: 78
Sunrise(IST): 2023-12-28 06:34:12
Sunset(IST): 2023-12-28 17:07:05
```

Caption



Caption

Conclusion:

- The program utilizes the OpenWeatherMap API to fetch and display temperature, sunrise, sunset, and humidity for a given city.
- Using APIs enables developers to integrate external data seamlessly into their applications, enhancing functionality and user experience.

Experiment No:7.2

Title:

Write a program to use the ‘API’ of crypto currency.

Theory:

Cryptocurrency APIs provide a way for developers to access real-time data about cryptocurrencies, including prices, market data, and other relevant information.

Code:

```
API_KEY='CG-Kjmr47XUisC8wTQ75jsf7wAS'

import requests

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_KEY}
")

a=response.json()

if coin in a:
print(a)
print("\nCrypto:",coin)
print("Price:",a[coin]['usd'],"USD")
print("Price:",a[coin]['inr'],"INR")

else:
```

```
print("Invalid cryptocoin!")
        b=input("Want to see more cryptocoins?(y/n):
")
if b.lower() == "n":
break
```

Output:

Caption

Test Case:

Caption

Conclusion:

- The program demonstrates how to use a cryptocurrency API (CoinGecko) to retrieve and display the current price of a specified cryptocurrency.

- APIs offer a powerful means for developers to incorporate dynamic data into applications, enhancing functionality and keeping information up-to-date.

THANK YOU