S.No: 1	Exp. Name: ***Area of Circle - Algorithm and Flowchart***	**Date: 2025-01-23**

Aim:

Write a program to calculate the area of a circle and print the result as shown in the displayed test cases.

Constraints:

- Radius is in between the range **0.0 to 100.0** both are **inclusive**
- Input is in the form of float.

Follow the given instructions and write the code in the space provided.

Note: Take the pi value as **3.14.**

Source Code:

AreaOfCircle.py

```
rad = float(input("Enter the radius : "))
pi = 3.14
area = pi * rad * rad
if rad >= 0.0 and rad < 100.0:
        print("Area of circle =",format(area,".6f"))
else:
        print("Enter a positive value upto 100\n")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the radius :
0
Area of circle = 0.000000

Test Case - 2
User Output
Enter the radius :
-100
Enter a positive value upto 100

S.No: 2	Exp. Name: ***Area of the Square***	**Date: 2025-01-23**

Aim:

You are given a side **S** and your task is to find the Area of the square.

Note:

- The value of **S** is already provided using input function in uneditable mode.
- Your input and output layout must match the visible sample test case.

Source Code:

Areaofcircle.py

```
S=int(input("S:")) # Make use of the value of S read using the input function.
area = S * S
print(area)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
S:
2
4

ID: 24070521273
2024-28-CSE-D
Symbiosis Institute of Technology | SIT Nagpur

S.No: 3	Exp. Name: ***Area of the Rectangle***	Date: 2025-01-23

Aim:

You are given a length **L** and breadth **B** and your task is to find the Area of the Rectangle.

Note:

- The values of **L** and **B** are already provided using input function in uneditable mode.
- Your input and output layout must match the visible sample test case.

Source Code:

Areaofrect.py

```
L=int(input("L:"))
B=int(input("B:"))
# Make use of the values of L and B read using the input function.
area = L * B
print(area)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
L:
2
B:
3
6

S.No: 4	Exp. Name: ***Area of a Triangle***	Date: 2025-01-23

Aim:
Write a Python program to print the area of a triangle.

Sample Input & Output:
Base: 10
Height: 15
Area: 75

Note: Round off the result to two decimal places

Source Code:

triangle_area.py

```
base = float(input("Base: "))
height = float(input("Height: "))
area = 1/2 * (base * height)
print("Area:",format(area,".2f"))
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Base:
10
Height:
15
Area: 75.00

Test Case - 2
User Output
Base:
30.5
Height:
19.95
Area: 304.24

S.No: 5	Exp. Name: ***Finding roots of quadratic equation***	**Date: 2025-01-31**

Aim:

Write a Python program to find the roots of a quadratic equation by taking the coefficients from the user.

Note: Refer to the displayed test cases for input and output format.

Source Code:

roots.py

```
import cmath
a = float(input("a: "))
b = float(input("b: "))
c = float(input("c: "))
discriminant = (b**2) - (4*a*c)

if discriminant > 0:
        root1 = (-b + cmath.sqrt(discriminant)) / (2*a)

        root2 = (-b - cmath.sqrt(discriminant)) / (2*a)

        print(f"The roots are: {root1.real:.2f} and {root2.real:.2f}")
elif discriminant == 0:
        root = -b / (2*a)
        print(f"The root is: {root:.2f}")
else:
        root1 = (-b + cmath.sqrt(discriminant)) / (2*a)
        root2 = (-b - cmath.sqrt(discriminant)) / (2*a)

        real_part1 = f"{root1.real:.2f}"
        imag_part1 = f"{root1.imag:.2f}"
        real_part2 = f"{root2.real:.2f}"
        imag_part2 = f"{root2.imag:.2f}"

        if float(real_part1) == 0.0:
                real_part1 = "-0.00"

        print(f"The roots are: {real_part1}+{imag_part1}j and
{real_part2}{imag_part2}j")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
a:
3
b:
33
c:
0
The roots are: 0.00 and -11.00

Test Case - 2
User Output
a:
3
b:
0
c:
1
The roots are: -0.00+0.58j and -0.00-0.58j

Test Case - 3
User Output
a:
1
b:
2
c:
1
The root is: -1.00

S.No: 6	Exp. Name: ***Largest of three numbers***	Date: 2025-01-28

Aim:

Write a Python program to find the largest of three numbers.

Note: Follow the input and output layout mentioned in the visible test cases.

Source Code:

large.py

```
# Type Content here...
num1=float(input("Enter the first number: "))
num2=float(input("Enter the second number: "))
num3=float(input("Enter the third number: "))
largest=max(num1,num2,num3)
print(f"Largest number: {largest:.1f}")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the first number:
5
Enter the second number:
12
Enter the third number:
8
Largest number: 12.0

Test Case - 2
User Output
Enter the first number:
-3
Enter the second number:
-8
Enter the third number:

-1
Largest number: -1.0

Test Case - 3
User Output
Enter the first number:
6
Enter the second number:
6
Enter the third number:
6
Largest number: 6.0

S.No: 7	Exp. Name: ***Problem Solving - Operators***	Date: 2025-01-28

Aim:

Write a program to read **temperature** in **Celsius** and print the **temperature** in **fahrenheit**.

Input Format:

- The user is prompted to enter a temperature (float) in Celsius.

Output Format:

- The program outputs the equivalent temperature in Fahrenheit.

Hint: The formula for conversion is **F = 1.8 * Temperature_in_Celsius + 32.0.**

Source Code:

OperatorPractice2.py

```
celsius=float(input("celsius: "))
fahrenheit=1.8*celsius+32.0
print(f"fahrenheit: {fahrenheit:}")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
celsius:
23.30
fahrenheit: 73.94

Test Case - 2
User Output
celsius:
45.3
fahrenheit: 113.53999999999999

S.No: 8	Exp. Name: ***Program to perform union, intersection, and difference operations on sets.***	**Date: 2025-01-30**

Aim:

Write a Python program to perform union, intersection and different operations on Set A and Set B.

Input format:

- The first input prompts for a space-separated list of integers for Set A
- The second input prompts for a space-separated list of integers for Set B

Output format:

- First displays the union of both sets
- Second line displays the intersection of both sets
- Third line displays the difference between Set A and Set B

Source Code:

setoperations.py

```
a = list(map(int,input("Set A: ").split()))
A = set(a)
b = list(map(int,input("Set B: ").split()))
B = set(b)

union_result = A.union(B)
intersection_result = A.intersection(B)
difference_result = A.difference(B)

print(f"Union:  {union_result}")
print(f"Intersection:  {intersection_result}")
print(f"Difference:  {difference_result}")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Set A:
0 2 4 6 8
Set B:
1 2 3 4 5

Union: {0, 1, 2, 3, 4, 5, 6, 8}
Intersection: {2, 4}
Difference: {0, 8, 6}

Test Case - 2
User Output
Set A:
10 11 22 33 44 55
Set B:
15 16 17 18 19 14
Union: {33, 10, 11, 44, 14, 15, 16, 17, 18, 19, 22, 55}
Intersection: set()
Difference: {33, 10, 11, 44, 22, 55}

S.No: 9	Exp. Name: ***Leap year***	Date: 2025-01-30

Aim:

Write a Python program to check if a given year is a leap year or not.

Sample Input and Output -1 :

```
Enter a year: 2020
2020 is a leap year
```

Sample Input and Output -2 :

```
Enter a year: 2006
2006 is not a leap year
```

Source Code:

leapYear.py

```
year = int(input("Enter a year: "))
if(year % 4 == 0 and year% 100 !=0) or (year % 400 == 0):
        print(f"{year} is a leap year")
else:
        print(f"{year} is not a leap year")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter a year:
1994
1994 is not a leap year

Test Case - 2
User Output
Enter a year:
2024
2024 is a leap year

S.No: 10	Exp. Name: ***Display the Grade***	Date: 2025-01-31

Aim:

Write a Python program to calculate the average marks for 5 subjects. The program should prompt the user to input the marks for each subject. After receiving the input, it should compute the average marks and then determine the corresponding grade based on the following grading system:

- A: 90 - 100
- B: 80 - 89
- C: 70 - 79
- D: 60 - 69
- F: Below 60

The program should display the average marks up to 2 decimal places and the assigned grade.

Source Code:

gradecalc.py

```
marks = []
for i in range(5):
        mark=float(input(f"subject {i+1}: "))
        marks.append(mark)

average_marks= sum(marks)/len(marks)
grade=""

if 90<= average_marks <= 100:
        grade="A"
elif 80<= average_marks < 89:
        grade="B"
elif 70<= average_marks < 79:
        grade="C"
elif 60<= average_marks <= 69:
        grade="D"
else:
        grade="F"
print(f"Average Marks: {average_marks:.2f}")
print(f"Grade: {grade}")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
subject 1:
67.8
subject 2:
89.7
subject 3:
90.5
subject 4:
90.0
subject 5:
98.0
Average Marks: 87.20
Grade: B

Test Case - 2
User Output
subject 1:
89.50
subject 2:
91.50
subject 3:
92.0
subject 4:
97.45
subject 5:
89.7
Average Marks: 92.03
Grade: A

S.No: 11	Exp. Name: ***Check whether the date is valid or not***	**Date: 2025-02-06**

Aim:

Write a Python program that prompts the user to input a date (year, month, and day) and checks if it is a valid date. If the entered date is valid, the program should increment the date by one day and display the incremented date. The program should take into account leap years when determining the number of days in February.

Source Code:

validdate.py

```
from datetime import date, timedelta
def is_valid_date(year,month,day):
        try:
                valid_date=date(year,month,day)
                return True
        except ValueError:
                return False
def increment_date(year,month,day):
        given_date = date(year,month,day)
        next_day=given_date+timedelta(days=1)
        return next_day
try:
        year = int(input("year: "))
        month = int(input("month: "))
        day = int(input("day: "))

        if is_valid_date(year,month,day):
                print("valid")
                next_date = increment_date(year,month,day)
                print(f"incremented date: {next_date}")
        else:
                print("invalid")

except ValueError:
        print("Invalid input !Please enter valid integers for
year,month,and day.")
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output
year:
2024
month:
12
day:
7
valid
incremented date: 2024-12-08

Test Case - 2
User Output
year:
2023
month:
13
day:
30
invalid

Test Case - 3
User Output
year:
2021
month:
2
day:
29
invalid

S.No: 12	Exp. Name: ***Factorial of a given number.***	**Date: 2025-01-31**

Aim:

Write a python program to print factorial of given number.

Note: If user enters a negative number as input prompt the message "Enter a positive number"

Source Code:

factorial.py

```
def factorial(n):
        if n==0 or n==1:
                return 1
        else:
                return n * factorial(n-1)
num=int(input("Enter a number : "))
if num<0:
        print("Enter a positive number")
else:
        print("Factorial of given number is :", factorial(num))
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter a number :
5
Factorial of given number is : 120

Test Case - 2
User Output
Enter a number :
-20
Enter a positive number

S.No: 13	Exp. Name: ***Print the following pattern***	Date: 2025-01-31

Aim:

Write a Python program to print the following pattern.

Sample Input and Output:

```
Enter a number : 6
* * * * * *
* * * * *
* * * *
* * *
* *
*
```

Source Code:

pattern2.py

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

for i in range(num,0,-1):
        print("* "*i)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter a number :
5
* * * * *
* * * *
* * *
* *
*

Test Case - 2
User Output

Enter a number :
6
* * * * * *
* * * * *
* * * *
* * *
* *
*

Test Case - 3
User Output
Enter a number :
3
* * *
* *
*

S.No: 14	Exp. Name: ***Combinations of all the digits.***	**Date: 2025-02-06**

Aim:

Write a Python program that prompts the user to input three digits (0-9) and checks if the entered digits are valid. If the digits are valid, the program generates all possible combinations of these three digits and prints them. Each combination is formed by arranging the digits in different orders. If the input is not valid (digits are not between 0 and 9), the program should display as "Invalid".

Source Code:

combinations.py

```
import itertools
a = int (input("digit1 (0-9): "))
b = int (input("digit2 (0-9): "))
c = int (input("digit3 (0-9): "))

if(0 <= a <= 9) and (0 <= b <= 9) and (0 <= c <= 9):
        com = itertools.permutations([a,b,c])
        for combo in com :
                print(''.join(map(str,combo)))

else:
        print("Invalid")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
digit1 (0-9):
1
digit2 (0-9):
2
digit3 (0-9):
3
123
132
213
231

312
321

Test Case - 2
User Output
digit1 (0-9):
3
digit2 (0-9):
2
digit3 (0-9):
10
Invalid

S.No: 15	Exp. Name: ***Write a Python program to perform multiplication of two matrices***	**Date: 2025-02-13**

Aim:

Write a Python program to perform **multiplication** of two matrices.

Sample Input and Output-1:

```
Enter values for matrix - A
Number of rows, m = 2
Number of columns, n = 2
Entry in row: 1 column: 1
Entry in row: 1 column: 2
Entry in row: 2 column: 1
Entry in row: 2 column: 2
Enter values for matrix - B
Number of rows, m = 2
Number of columns, n = 2
Entry in row: 1 column: 1
Entry in row: 1 column: 2
Entry in row: 2 column: 1
Entry in row: 2 column: 2
Matrix - A = [[1, 2], [3, 4]]
Matrix - B = [[1, 2], [3, 4]]
Matrix - A * Matrix- B = [[7, 10], [15, 22]]
```


ID: 2407052127 3

Symbiosis Institute of Technology | SIT Nagpur

Sample Input and Output-2:

```
Enter values for matrix - A
Number of rows, m = 2
Number of columns, n = 3
Entry in row: 1 column: 1
Entry in row: 1 column: 2
Entry in row: 1 column: 3
Entry in row: 2 column: 1
Entry in row: 2 column: 2
Entry in row: 2 column: 3
Enter values for matrix - B
Number of rows, m = 2
Number of columns, n = 3
Entry in row: 1 column: 1
Entry in row: 1 column: 2
Entry in row: 1 column: 3
Entry in row: 2 column: 1
Entry in row: 2 column: 2
Entry in row: 2 column: 3
Matrix - A = [[1, 2, 3], [4, 5, 6]]
Matrix - B = [[1, 2, 3], [4, 5, 6]]
Cannot multiply the two matrices. Incorrect dimensions.
Matrix - A * Matrix- B = None
```

Source Code:

Lab11c.py

```
def matmult(A, B):
        rows_A = len(A)
        cols_A = len(A[0])

        rows_B = len(B)
        cols_B = len(B[0])

        if cols_A != rows_B :
                print("Cannot multiply the two matrices. Incorrect
dimensions.")
                return None

        result = []
        for i in range(rows_A):
                row = []
                for j in range(cols_B):
                        row.append(0)
                result.append(row)

        for i in range(rows_A):
                for j in range(cols_B):
                        for k in range(cols_A):
                                result[i][j] += A[i][k] * B[k][j]
        return result

def readmatrix(name = ''):
        print(f"Enter values for {name}")
        rows = int(input(f"Number of rows, m = "))
        cols = int(input(f"Number of columns, n = "))

        matrix = []

        for i in range(rows):
                rows = []
                for j in range(cols):
                        print(f"Entry in row: {i+1} column: {j+1}")
                        value = int(input())
                        rows.append(value)
                matrix.append(rows)
        return matrix
""" """
matrixa = readmatrix('matrix - A')
matrixb = readmatrix('matrix - B')
print("Matrix - A =", matrixa)

print("Matrix - B =", matrixb)
```

```
print("Matrix - A * Matrix- B =", matmult(matrixa, matrixb))
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter values for matrix - A
Number of rows, m =
2
Number of columns, n =
2
Entry in row: 1 column: 1
1
Entry in row: 1 column: 2
2
Entry in row: 2 column: 1
3
Entry in row: 2 column: 2
4
Enter values for matrix - B
Number of rows, m =
2
Number of columns, n =
2
Entry in row: 1 column: 1
1
Entry in row: 1 column: 2
2
Entry in row: 2 column: 1
3
Entry in row: 2 column: 2
4
Matrix - A = [[1, 2], [3, 4]]
Matrix - B = [[1, 2], [3, 4]]
Matrix - A * Matrix- B = [[7, 10], [15, 22]]

Test Case - 2
User Output
Enter values for matrix - A
Number of rows, m =
2
Number of columns, n =
3
Entry in row: 1 column: 1
1
Entry in row: 1 column: 2
2
Entry in row: 1 column: 3
3
Entry in row: 2 column: 1
4
Entry in row: 2 column: 2
5
Entry in row: 2 column: 3
6
Enter values for matrix - B
Number of rows, m =
3
Number of columns, n =
2
Entry in row: 1 column: 1
1
Entry in row: 1 column: 2
2
Entry in row: 2 column: 1
3
Entry in row: 2 column: 2
4
Entry in row: 3 column: 1
5
Entry in row: 3 column: 2
6
Matrix - A = [[1, 2, 3], [4, 5, 6]]
Matrix - B = [[1, 2], [3, 4], [5, 6]]
Matrix - A * Matrix- B = [[22, 28], [49, 64]]

Test Case - 3
User Output
Enter values for matrix - A
Number of rows, m =
3
Number of columns, n =
2
Entry in row: 1 column: 1
1
Entry in row: 1 column: 2
2
Entry in row: 2 column: 1
3
Entry in row: 2 column: 2
3
Entry in row: 3 column: 1
2
Entry in row: 3 column: 2
1
Enter values for matrix - B
Number of rows, m =
2
Number of columns, n =
1
Entry in row: 1 column: 1
1
Entry in row: 2 column: 1
2
Matrix - A = [[1, 2], [3, 3], [2, 1]]
Matrix - B = [[1], [2]]
Matrix - A * Matrix- B = [[5], [9], [4]]

S.No: 16	Exp. Name: ***Prime numbers***	Date: 2025-02-06

Aim:

Write a Python program that prints prime numbers less than **n** which represents the upper limit.

Source Code:

primeNumbers.py

```
def is_prime(num):
        if num<2:
                return False
        for i in range(2,int(num**0.5)+1):
                if num%i==0:
                        return False
        return True
n=int(input("Enter upper limit: "))
for num in range(2,n):
        if is_prime(num):
                print(num)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter upper limit:
20
2
3
5
7
11
13
17
19

Test Case - 2

User Output
Enter upper limit:
36
2
3
5
7
11
13
17
19
23
29
31

S.No: 17	Exp. Name: ***Program to count the number of vowels in a given string using sets.***	**Date: 2025-02-06**

Aim:

Write a program to count the number of vowels using sets in a given string.

Input Format:

- Prompts the user to enter the string.

Output Format:

- Print the count of vowels present in the string.

Source Code:

noofvowels.py

```
def vowel_count(str):
        count = 0
        vowel = set("aeiouAEIOU")
        for char in str:
                if char in vowel:
                        count += 1
        print(count)
str = input()
vowel_count(str)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Hello World
3

Test Case - 2
User Output
Rhythm
0

ID: 24070521273
2024-28-CSE-D
Symbiosis Institute of Technology | SIT Nagpur

Test Case - 3
User Output
CodeTantra
4

S.No: 18	Exp. Name: ***Palindrome checker***	Date: 2025-02-06

Aim:

Write and execute a Python program to find whether a given string is a palindrome or not.

Note: A palindrome is a string that reads the same backward as forward.
For example, "racecar" is a palindrome because it reads the same in both directions, while "hello" is not.

Input format:

- The input should be a string

Output format:

- If the given string is a palindrome, print "palindrome", otherwise, print "not a palindrome"

Source Code:

palindrome.py

```
# Type Content here...
def check_palindrome(input_string):
        cleaned_string=input_string.replace(" ","").lower()
        if cleaned_string==cleaned_string[::-1]:
                return "palindrome"
        else:
                return "not a palindrome"
input_string = input()
result=check_palindrome(input_string)
print(result)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
mam
palindrome

Test Case - 2

User Output
sir
not a palindrome

S.No: 19	Exp. Name: ***Remove Punctuations***	Date: 2025-02-06

Aim:

Write a Python program that takes a sentence as input, removes punctuations from the sentence, and displays the modified sentence.

Source Code:

punctuation.py

```
import string
def remove_punctuation(sentence):
        translator = str.maketrans('','', string.punctuation)
        return sentence.translate(translator)
input_sentence=input()
modified_sentence=remove_punctuation(input_sentence)
print(modified_sentence)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
I love Coding!!
I love Coding

Test Case - 2
User Output
I have 3 chocolates.
I have 3 chocolates

ID: 24070521273
2024-28-CSE-D
Symbiosis Institute of Technology | SIT Nagpur

S.No: 20	Exp. Name: ***Reverse of a string***	Date: 2025-02-06

Aim:

Write and execute a Python program to find the reverse of a string.

Input format:

- The input should be a string

Output format:

- The output should be reverse of a string

Source Code:

reverseString.py

```
def reverse_string(input_string):
        return input_string[::-1]
input_string=input()
reversed_string=reverse_string(input_string)
print(reversed_string)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
CodeTantra
artnaTedoC

Test Case - 2
User Output
Programming
gnimmargorP

S.No: 21	Exp. Name: ***Program to find the sum of digits of a number***	**Date: 2025-02-27**

Aim:

Write a program to print the sum of digits of a number.

For example: If the number is **1234**, then the sum of digits, **1 + 2 + 3 + 4 = 10** should be printed.

Sample Input and Output:

```
num: 454545
sum: 27
```

Source Code:

Sumofdigs.py

```
def sum(num):
        sum = 0
        while num> 0:
                sum += num % 10
                num //=10
        return sum

n = int(input("num: "))
print("sum:",sum(n))
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
num:
101
sum: 2

Test Case - 2
User Output

num:
454545
sum: 27

Test Case - 3
User Output
num:
363
sum: 12

S.No: 22	Exp. Name: ***Sum of the Digits of a Number using Recursion***	**Date: 2025-03-01**

Aim:

Take an integer **n** from the user.Your task is to Write a program to find out the sum of the digits of the given number using the process of recursion. Print the result as shown in the Test cases.

- The program defines the **Sumof()** function.
- In the main program it takes the input **n** and sends it to the **Sumof()** function.
- The **Sumof()** function contains base and recursive criterion.

Constraints:

1 <= integer <= 10^6

Sample Test Case:

4532 ----> Input integer

14 ----> Sum of the digits of the given number (4+5+3+2 = 14)

Source Code:

sumofdigits.py

```
'''
Complete the given function using recursive approach,
and also write the driver code test the functionality,
and pass all the visible and hidden test cases.

'''
def Sumof(n):
        if n<10 :
                return n
        return(n%10)+Sumof(n//10)
n=int(input())
print(Sumof(n))

# take user input and add the function call
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output

4532
14

Test Case - 2
User Output
109
10

Test Case - 3
User Output
56
11

S.No: 23	Exp. Name: ***Phone book manager***	**Date: 2025-03-18**

Aim:

Write a Python program that functions as a simple phone book manager with a menu-driven interface. The program prompts the user to choose from the following options:

1. Add Contact
2. Remove Contact
3. Display
4. Quit

The program uses a dictionary to store contact information, where the contact name serves as the key and the associated phone number as the value. The user can add a contact, remove a contact, display the current contacts, or exit the program.

Source Code:

phonenumber_db.py

ID: 24070521273

```
phone_book = {}
while True:
        print("1. Add Contact")
        print("2. Remove Contact")
        print("3. Display")
        print("4. Quit")
        choice = input("Enter choice (1-4): ")

        if choice == '1':
                name=input("Name: ")
                phone=input("Phone number: ")
                phone_book[name]=phone

        elif choice=='2':
                name=input("Name: ")
                if name in phone_book:
                        del phone_book[name]
                else:
                        print("Not found")
        elif choice=='3':
                if not phone_book:
                        print("{}")
                else:
                        print(phone_book)
        elif choice=='4':
                break
        else:
                print("Invalid")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1. Add Contact
2. Remove Contact
3. Display
4. Quit
Enter choice (1-4):
3

```
{}
1. Add Contact
2. Remove Contact
3. Display
4. Quit
Enter choice (1-4):
1
Name:
Aman
Phone number:
8900004500
1. Add Contact
2. Remove Contact
3. Display
4. Quit
Enter choice (1-4):
1
Name:
Arjun
Phone number:
966969669
1. Add Contact
2. Remove Contact
3. Display
4. Quit
Enter choice (1-4):
3
{'Aman': '8900004500', 'Arjun': '966969669'}
1. Add Contact
2. Remove Contact
3. Display
4. Quit
Enter choice (1-4):
2
Name:
Aanya
Not found
1. Add Contact
2. Remove Contact
3. Display
4. Quit
```

Enter choice (1-4):
2
Name:
Arjun
1. Add Contact
2. Remove Contact
3. Display
4. Quit
Enter choice (1-4):
3
{'Aman': '8900004500'}
1. Add Contact
2. Remove Contact
3. Display
4. Quit
Enter choice (1-4):
7
Invalid
1. Add Contact
2. Remove Contact
3. Display
4. Quit
Enter choice (1-4):
4

S.No: 24	Exp. Name: ***Write a python program to define a module to find Fibonacci Numbers and import the module to another program.***	**Date: 2025-03-18**

Aim:

Write a python program to define a module to find Fibonacci Numbers and import the module to another program.

Aim:

- To create a Python program that generates a Fibonacci sequence up to a given maximum value.

Algorithm:

Step 1: Import the fibonacci_module.
Step 2: Accept an integer input from the user as the maximum value (n).
Step 3: If n is greater than 0:
- Generate the Fibonacci sequence up to n using the generate_fibonacci_sequence() function from the fibonacci_module.
- Print the generated Fibonacci series.

Step 4: If n is not greater than 0, print "Please enter a positive integer".
Step 5: End the program.

Note: The fibonacci_module contains the generate_fibonacci_sequence() function to generate the Fibonacci sequence up to a specified maximum value.

Source Code:

fibonacci_program.py

```
import fibonacci_module
def main():
        try:
                n=int(input("Enter the max value: "))
                if n>0:

fib_series=fibonacci_module.generate_fibonacci_sequence(n)
                        print(f"Fibonacci series upto {n} :")
                        print(" ".join(map(str,fib_series)),end=" ")
                else:
                        print("Please enter a positive integer")
        except ValueError:
                print("Invalid input! Please enter a valid integer.")
if __name__ == "__main__":
        main()
#write your code here
```

fibonacci_module.py

```
def generate_fibonacci_sequence(count):
        if count<=0:
                return[]
        elif count ==1:
                return[0]
        elif count==2:
                return[0,1]
        sequence = generate_fibonacci_sequence(count-1)
        sequence.append(sequence[-1] + sequence[-2])
        return sequence
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the max value:
10
Fibonacci series upto 10 :
0 1 1 2 3 5 8 13 21 34

Test Case - 2
User Output
Enter the max value:
-9
Please enter a positive integer

S.No: 25	Exp. Name: ***Sum of the Complex numbers***	**Date: 2025-03-20**

Aim:

Implement a Python program using a class named **Complex** to perform operations on complex numbers. The class has the following methods:

1. **initComplex():** A method that takes user input for the real and imaginary parts to initialize a complex number.
2. **display():** A method that displays the complex number in the form **"a + bi".**
3. **sum():** A method that computes the sum of two complex numbers and stores the result in the current instance.

The program creates three instances of the **Complex** class, initializes two complex numbers, displays them, computes their sum, and displays the result.

Source Code:

complex_class.py

ID: 24070521273
2024-28-CSE-D
Symbiosis Institute of Technology | SIT Nagpur

```
class Complex ():
        def __init__(self):
                self.real=0
                self.imaginary=0

        def initComplex(self):
                self.real=int(input("Real Part: "))
                self.imaginary=int(input("Imaginary Part: "))

        def display(self):
                if self.imaginary>=0:
                        print(f"{self.real}+{self.imaginary}i")
                else:
                        print(f"{self.real}{self.imaginary}i")

        def sum(self,c1,c2):
                self.real=c1.real+c2.real
                self.imaginary=c1.imaginary+c2.imaginary
c1 = Complex()
c2 = Complex()
c3 = Complex()
print("complex number 1")
c1.initComplex()
c1.display()
print("complex number 2")
c2.initComplex()
c2.display()
print("Sum:",end=" ")


c3.sum(c1,c2)
c3.display()
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
complex number 1
Real Part:
3
Imaginary Part:
4

3+4i
complex number 2
Real Part:
-9
Imaginary Part:
5
-9+5i
Sum: -6+9i

Test Case - 2
User Output
complex number 1
Real Part:
-5
Imaginary Part:
0
-5+0i
complex number 2
Real Part:
-8
Imaginary Part:
0
-8+0i
Sum: -13+0i

S.No: 26	Exp. Name: ***Display the car details***	Date: 2025-03-20

Aim:

Follow the instructions to create a Python program modeling cars with specific types: Car1 and Car2. Begin by defining a base class **Car** with attributes **brand**, **price**, **model,** and **color**. Subsequently, create two derived classes, **Car1** and **Car2**, both inheriting from the Car class. Each derived class should introduce its attributes, and:

- Implement a method **display_details** in the base class Car to print the common attributes (brand, price, model, color).
- Override the **display_details** method in each derived class (Car1 and Car2) to print the brand, price, model, and color respectively.

Input Format:

For Car1:

- The first line contains the brand, price, model, and color of the Car1, separated by spaces.

For Car2:

- The first line contains the brand, price, model, and color of the Car2, separated by spaces.

Output Format:

- The first four lines should display information about the Car1, including the brand, price, model, and color.
- The second four lines should display information about the Car2, including the brand, price, model, and color.

Note:

- Price must be a positive float.
- Refer to the displayed test cases for better understanding.
- For simplicity, code for reading inputs has already been provided.

Source Code:

carDetails.py

```
class Car:
        def __init__(self,brand,price,model,color):
                self.brand= brand
                self.price= price
                self.model= model
                self.color= color

        def display_details(self):
                print(f"{self.brand}")
                print(f"{self.price}")
                print(f"{self.model}")
                print(f"{self.color}")
def get_car_details():
        brand = input("brand: ")
        price = float(input("price: "))
        model = input("model: ")
        color = input("color: ")
        return brand, price, model, color

print("Details for Car 1:")
car1_brand, car1_price, car1_model, car1_color = get_car_details()
car1=Car(car1_brand,car1_price,car1_model,car1_color)
print("Details for Car 2:")
car2_brand, car2_price, car2_model, car2_color = get_car_details()
car2=Car(car2_brand,car2_price,car2_model,car2_color)
print("Car 1 Details:")
car1.display_details()
print("Car 2 Details:")
car2.display_details()
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Details for Car 1:
brand:
Nano
price:
120000
model:
Magic

color:
Yellow
Details for Car 2:
brand:
Innova
price:
200000
model:
xu
color:
White
Car 1 Details:
Nano
120000.0
Magic
Yellow
Car 2 Details:
Innova
200000.0
xu
White