**Name:- Milind Kailas Tajane**

**Roll No:- CS061**

**Date:-\_\_\_\_\_\_\_\_\_\_\_\_**

**Practical No:6**

# ----------------------------------------------------------------

**AIM:- Write a python program to find sum & multiplication of two matrices implemented using list.**

**---------------------------------------------------------------------------------------------------------------**

**CODE:-**

def add\_matrices(matrix1, matrix2):

*# Check if dimensions are the same*

*if* len(matrix1) *!=* len(matrix2) *or* len(matrix1[0]) *!=* len(matrix2[0]):

*raise* ValueError("Matrices must have the same dimensions for addition.")

    result *=* []

*for* i *in* range(len(matrix1)):

        row *=* []

*for* j *in* range(len(matrix1[0])):

            row.append(matrix1[i][j] *+* matrix2[i][j])

        result.append(row)

*return* result

def multiply\_matrices(matrix1, matrix2):

*# Check if matrices can be multiplied*

*if* len(matrix1[0]) *!=* len(matrix2):

*raise* ValueError("Number of columns in the first matrix must be equal to the number of rows in the second matrix.")

    result *=* []

*for* i *in* range(len(matrix1)):

        row *=* []

*for* j *in* range(len(matrix2[0])):

            sum\_product *=* 0

*for* k *in* range(len(matrix2)):

                sum\_product *+=* matrix1[i][k] *\** matrix2[k][j]

            row.append(sum\_product)

        result.append(row)

*return* result

*# Example usage:*

matrix1 *=* [

    [1, 2, 3],

    [4, 5, 6],

]

matrix2 *=* [

    [7, 8, 9],

    [1, 2, 3],

]

*# Matrix Addition*

sum\_result *=* add\_matrices(matrix1, matrix2)

print("Sum of matrices:")

*for* row *in* sum\_result:

    print(row)

*# Matrix Multiplication*

matrix3 *=* [

    [1, 2],

    [3, 4],

    [5, 6],

]

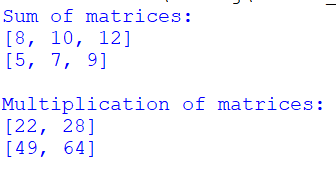
multiplication\_result *=* multiply\_matrices(matrix1, matrix3)

print("\nMultiplication of matrices:")

*for* row *in* multiplication\_result:

    print(row)

**Output:-**



**Name:- Milind Kailas Tajane**

**Roll No:- CS061**

**Date:-\_\_\_\_\_\_\_\_\_\_\_\_**

**Practical No:7**

# ----------------------------------------------------------------

**AIM:- Write a program to implement function decorator to display cube of a number.**

**----------------------------------------------------------------------------------------------------------------**

**CODE:-**

# Define the decorator

def cube\_decorator(func):

def wrapper(num):

result = func(num)

print(f"The cube of {num} is {result}")

return result

return wrapper

# Use the decorator on a function

@cube\_decorator

def cube(num):

return num \*\* 3

# Test the function

if \_\_name\_\_ == "\_\_main\_\_":

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

cube(number)

**Output:-**



**Name:- Milind Kailas Tajane**

**Roll No:- CS061**

**Date:-\_\_\_\_\_\_\_\_\_\_\_\_**

**Practical No: 8**

# ----------------------------------------------------------------

**AIM:- Write a program to implement generator function to display square of numbers from 1 to 10 .**

**----------------------------------------------------------------------------------------------------------------**

**CODE:-**

# Define the generator function

def square\_generator():

for num in range(1, 11):

yield num \*\* 2

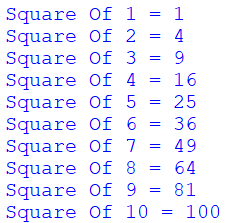
# Use the generator function

if \_\_name\_\_ == "\_\_main\_\_":

for line\_number, square in enumerate(square\_generator(), start=1):

print(f"Square Of {line\_number} = {square}")

**Output:-**

****

**Name:- Milind Kailas Tajane**

**Roll No:- CS061**

**Date:-\_\_\_\_\_\_\_\_\_\_\_\_**

**Practical No: 9**

# ----------------------------------------------------------------

**AIM:- WAP to implement tower of Hanoi.**

**----------------------------------------------------------------------------------------------------------------**

**CODE:-**

def tower\_of\_hanoi(n, source, destination, auxiliary):

if n == 1:

print(f"Move disk 1 from {source} to {destination}")

return

# Move n-1 disks from source to auxiliary, using destination as auxiliary.

tower\_of\_hanoi(n - 1, source, auxiliary, destination)

# Move the nth disk from source to destination.

print(f"Move disk {n} from {source} to {destination}")

# Move the n-1 disks from auxiliary to destination, using source as auxiliary.

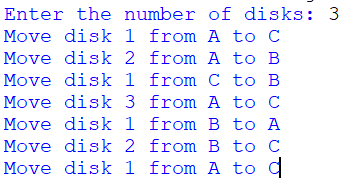
tower\_of\_hanoi(n - 1, auxiliary, destination, source)

# Example usage

num\_disks = int(input("Enter the number of disks: "))

tower\_of\_hanoi(num\_disks, "A", "C", "B")

**Output:-**



**Name:- Milind Kailas Tajane**

**Roll No:- CS061**

**Date:-\_\_\_\_\_\_\_\_\_\_\_\_**

**Practical No: 10**

# ----------------------------------------------------------------

**AIM:- Write A Program to display date in following format “Friday, 23 April 2017”.**

**----------------------------------------------------------------------------------------------------------------**

**CODE:-**

from datetime import datetime

def display\_formatted\_date(date\_input):

"""

Displays the date in the format: 'Friday, 23 April 2017'

Parameters:

date\_input (str): Input date string in 'YYYY-MM-DD' format.

"""

try:

# Parse the input date string

date\_object = datetime.strptime(date\_input, '%Y-%m-%d')

# Format the date in the desired format

formatted\_date = date\_object.strftime('%A, %d %B %Y')

print(f"Formatted Date: {formatted\_date}")

except ValueError:

print("Invalid date format. Please use 'YYYY-MM-DD' format.")

# Example usage

input\_date = input("Enter a date (YYYY-MM-DD): ")

display\_formatted\_date(input\_date)

**Output:-**

