

1. Create a three dimensional array specifying float data type and print it.

```
print("\nNafeesath Neema\nRoll no:42\nbatch no:MCA-2022-24")
import numpy as np
depth = int(input("Enter the depth of the array: "))
rows = int(input("Enter the number of rows: "))
columns = int(input("Enter the number of columns: "))
three_dimensional_array = np.random.rand(depth, rows,
columns).astype(np.float32)
print("Generated 3D array:")
print(three_dimensional_array)
```

```
Nafeesath Neema
Roll no:42
batch no:MCA-2022-24
Enter the depth of the array: 12
Enter the number of rows: 2
Enter the number of columns: 2
Generated 3D array:
[[[0.03374756 0.37526283]
  [0.16418463 0.776146  ]]

 [[0.5469551  0.21093315]
  [0.6645558  0.86012506]]

 [[0.6183105  0.78761625]
  [0.69047785 0.09345458]]

 [[0.39935112 0.66432446]
  [0.13165326 0.85916185]]

 [[0.7489886  0.03847741]
  [0.01290835 0.8954838  ]]

 [[0.776423   0.62279344]
  [0.81224215 0.81705797]]

 [[0.271762   0.944755  ]
  [0.5860767  0.46948567]]

 [[0.4243815  0.3999472  ]
  [0.01959068 0.36435392]]

 [[0.68256974 0.30395514]
  [0.58696944 0.7973522  ]]

 [[0.85257185 0.03592451]
  [0.92388    0.16599247]]]
```

2. Create a 2 dimensional array (2X3) with elements belonging to complex data type and print it. Also display
- the no: of rows and columns
 - dimension of an array
 - reshape the same array to 3X2

```
print("\nNafeesath Neema\nRoll n0:42\nbatch no:MCA-2022-24")
import numpy as np
complex_array = np.array([[1 + 2j, 3 + 4j, 5 + 6j], [7 + 8j, 9 + 10j, 11 + 12j]], dtype=complex)
print("2D Array with Complex Data Type:")
print(complex_array)
num_rows, num_columns = complex_array.shape
array_dimensions = complex_array.ndim
reshaped_array = complex_array.reshape(3, 2)
print("\na. Number of Rows:", num_rows)
print("    Number of Columns:", num_columns)
print("b. Dimensions of the Array:", array_dimensions)
print("c. Reshaped Array (3x2):")
print(reshaped_array)
```

```
Nafeesath Neema
Roll n0:42
batch no:MCA-2022-24
2D Array with Complex Data Type:
[[ 1. +2.j  3. +4.j  5. +6.j]
 [ 7. +8.j  9.+10.j 11.+12.j]]

a. Number of Rows: 2
    Number of Columns: 3
b. Dimensions of the Array: 2
c. Reshaped Array (3x2):
[[ 1. +2.j  3. +4.j]
 [ 5. +6.j  7. +8.j]
 [ 9.+10.j 11.+12.j]]
```

3. Familiarize with the functions to create

- a) an uninitialized array
- b) array with all elements as 1,
- c) all elements as 0

```
print("\nNafeesath Neema\nRoll n0:42\nbatch no:MCA-2022-24")
import numpy as np
uninitialized_array = np.empty((2, 3))
print("Uninitialized Array:")
print(uninitialized_array)
ones_array = np.ones((2, 3))
print("Array with All Elements as 1:")
print(ones_array)
zeros_array = np.zeros((2, 3))
print("Array with All Elements as 0:")
print(zeros_array)
```

```
Nafeesath Neema
Roll n0:42
batch no:MCA-2022-24
Uninitialized Array:
[[4.65465816e-310  0.00000000e+000  6.91975379e-310]
 [6.91975464e-310  6.91975337e-310  6.91975469e-310]]
Array with All Elements as 1:
[[1.  1.  1.]
 [1.  1.  1.]]
Array with All Elements as 0:
[[0.  0.  0.]
 [0.  0.  0.]]
```

4. Create an one dimensional array using arange function containing 10 elements.

Display

- a. First 4 elements
- b. Last 6 elements
- c. Elements from index 2 to 7

```
print("\nNafeesath Neema\nRoll no:42\nbatch no:MCA-2022-24")
import numpy as np
user_input = []
for i in range(10):
    element = float(input(f"Enter element {i + 1}: "))
    user_input.append(element)
arr = np.array(user_input)
print("a. First 4 Elements:")
print(arr[:4])
print("\nb. Last 6 Elements:")
print(arr[-6:])
start_index = int(input("Enter the start index (2 for example): "))
end_index = int(input("Enter the end index (7 for example): "))
print("\nc. Elements from Index {} to {}".format(start_index, end_index))
print(arr[start_index:end_index + 1])
```

5. Create an 1D array with arange containing first 15 even numbers as elements
 - a. Elements from index 2 to 8 with step 2(also demonstrate the same using slice function)
 - b. Last 3 elements of the array using negative index
 - c. Alternate elements of the array
 - d. Display the last 3 alternate elements
6. Create a 2 Dimensional array with 4 rows and 4 columns.
 - a. Display all elements excluding the first row
 - b. Display all elements excluding the last column
 - c. Display the elements of 1 st and 2 nd column in 2 nd and 3 rd row
 - d. Display the elements of 2 nd and 3 rd column
 - e. Display 2 nd and 3 rd element of 1 st row
 - f. Display the elements from indices 4 to 10 in descending order(use -values)

```
import numpy as np
print("\nNafeesath Neema\nRoll no:42\nbatch no:MCA-2022-24")
arr1d = np.arange(2, 32, 2)
print("1D Array with the first 15 even numbers as elements:")
print(arr1d)

# Using slicing
slice_result_a = arr1d[2:9:2]
print("\na. Elements from index 2 to 8 with step 2:")
print(slice_result_a)

# Using negative index
last_3_elements = arr1d[-3:]
print("\nb. Last 3 elements of the array using negative index:")
print(last_3_elements)

# Using slicing with step 2
alternate_elements = arr1d[::2]
print("\nc. Alternate elements of the array:")
print(alternate_elements)

# Using negative index for the last 3 elements and slicing with step 2
last_3_alternate_elements = arr1d[-3::2]
print("\nd. Display the last 3 alternate elements:")
print(last_3_alternate_elements)

arr2d = np.arange(1, 17).reshape(4, 4)
print("\n2D Array with 4 rows and 4 columns:")
print(arr2d)

# Using slicing to exclude the first row
exclude_first_row = arr2d[1:]
print("\na. Display all elements excluding the first row:")
```

```
print(exclude_first_row)

# Using slicing to exclude the last column
exclude_last_column = arr2d[:, :-1]
print("\nb. Display all elements excluding the last column:")
print(exclude_last_column)

# Using slicing to select specific rows and columns
selected_elements = arr2d[1:3, 0:2]
print("\nc. Display the elements of the 1st and 2nd column in the 2nd and 3rd row:")
print(selected_elements)

# Using slicing to select specific columns
selected_columns = arr2d[:, 1:3]
print("\nd. Display the elements of the 2nd and 3rd column:")
print(selected_columns)

# Using slicing to select elements in the 1st row
selected_elements_row1 = arr2d[0, 1:3]
print("\ne. Display the 2nd and 3rd element of the 1st row:")
print(selected_elements_row1)
```

Output

```
a. Display all elements excluding the first row:  
[[ 5  6  7  8]  
 [ 9 10 11 12]  
[13 14 15 16]]  
  
b. Display all elements excluding the last column:  
[[ 1  2  3]  
 [ 5  6  7]  
 [ 9 10 11]  
[13 14 15]]  
  
c. Display the elements of the 1st and 2nd column in the 2nd and 3rd row:  
[[ 5  6]  
 [ 9 10]]  
  
d. Display the elements of the 2nd and 3rd column:  
[[ 2  3]  
 [ 6  7]  
[10 11]  
[14 15]]  
  
e. Display the 2nd and 3rd element of the 1st row:  
[2 3]  
  
f. Display the elements from indices 4 to 10 in descending order:  
[22 20 18 16 14 12 10]
```

Nafeesath Neema

Roll no:42

batch no:MCA-2022-24

1D Array with the first 15 even numbers as elements:

```
[ 2  4  6  8 10 12 14 16 18 20 22 24 26 28 30]
```

a. Elements from index 2 to 8 with step 2:

```
[ 6 10 14 18]
```

b. Last 3 elements of the array using negative index:

```
[26 28 30]
```

c. Alternate elements of the array:

```
[ 2  6 10 14 18 22 26 30]
```

d. Display the last 3 alternate elements:

```
[26 30]
```

2D Array with 4 rows and 4 columns:

```
[[ 1  2  3  4]
```

```
 [ 5  6  7  8]
```

```
 [ 9 10 11 12]
```

```
[13 14 15 16]]
```

a. Display all elements excluding the first row:

```
[[ 5  6  7  8]
```

```
 [ 9 10 11 12]
```

```
[13 14 15 16]]
```

b. Display all elements excluding the last column:

```
[[ 1  2  3]
```

```
 [ 5  6  7]
```

```
 [ 9 10 11]
```

```
[13 14 15]]
```

c. Display the elements of the 1st and 2nd column in the 2nd and 3rd row:

```
[[ 5  6]
```


7. Create two 2D arrays using array object and
 - a. Add the 2 matrices and print it
 - b. Subtract 2 matrices
 - c. Multiply the individual elements of matrix
 - d. Divide the elements of the matrices
 - e. Perform matrix multiplication
- f. Display transpose of the matrix
- g. Sum of diagonal elements of a matrix

```
import numpy as np
print("\nNafeesath Neema\nRoll no:42\nbatch no:MCA-2022-24")
matrix1 = np.array([[1, 2, 3],
                    [4, 5, 6],
                    [7, 8, 9]])

matrix2 = np.array([[9, 8, 7],
                    [6, 5, 4],
                    [3, 2, 1]])

# a. Add the two matrices and print the result
matrix_sum = matrix1 + matrix2
print("Matrix Addition:")
print(matrix_sum)

# b. Subtract the two matrices and print the result
matrix_diff = matrix1 - matrix2
print("\nMatrix Subtraction:")
print(matrix_diff)

# c. Multiply the individual elements of the matrices
elementwise_product = matrix1 * matrix2
print("\nElement-wise Multiplication:")
print(elementwise_product)

# d. Divide the elements of the matrices (assuming no division by zero)
elementwise_division = matrix1 / matrix2
print("\nElement-wise Division:")
print(elementwise_division)

# e. Perform matrix multiplication (dot product)
matrix_product = np.dot(matrix1, matrix2)
print("\nMatrix Multiplication:")
print(matrix_product)
```

```
# f. Display the transpose of a matrix
matrix1_transpose = np.transpose(matrix1)
print("\nTranspose of Matrix 1:")
print(matrix1_transpose)

# g. Calculate the sum of diagonal elements of a matrix
diagonal_sum = np.trace(matrix1)
print("\nSum of Diagonal Elements of Matrix 1:", diagonal_sum)
```

```
Nafeesath Neema
Roll no:42
batch no:MCA-2022-24
Matrix Addition:
[[10 10 10]
 [10 10 10]
 [10 10 10]]

Matrix Subtraction:
[[-8 -6 -4]
 [-2  0  2]
 [ 4  6  8]]

Element-wise Multiplication:
[[ 9 16 21]
 [24 25 24]
 [21 16  9]]

Element-wise Division:
[[0.11111111 0.25      0.42857143]
 [0.66666667 1.        1.5       ]
 [2.33333333 4.        9.        ]]

Matrix Multiplication:
[[ 30  24  18]
 [ 84  69  54]
 [138 114  90]]

Transpose of Matrix 1:
[[1 4 7]
 [2 5 8]
 [3 6 9]]

Sum of Diagonal Elements of Matrix 1: 15
```

8. Demonstrate the use of insert() function in 1D and 2D array

```
import numpy as np
print("\nNafeesath Neema\nRoll no:42\nbatch no:MCA-2022-24")
# Input the elements of the 1D array
arr1d = np.array([int(x) for x in input("Enter elements of the 1D array
separated by spaces: ").split()])

# Input the element and position to insert
element = int(input("Enter the element to insert: "))
position = int(input("Enter the position to insert (0-based index): "))

# Insert the element at the specified position
arr1d_inserted = np.insert(arr1d, position, element)

print("Original 1D array:")
print(arr1d)

print("\n1D array after inserting the element:")
print(arr1d_inserted)
rows = int(input("Enter the number of rows: "))
cols = int(input("Enter the number of columns: "))

# Input the elements of the 2D array
print("Enter elements of the 2D array row by row:")
arr2d = np.array([[int(x) for x in input().split()] for _ in range(rows)])

# Input the row, column, and element to insert
row_index = int(input("Enter the row index to insert: "))
col_index = int(input("Enter the column index to insert: "))
element = int(input("Enter the element to insert: "))

# Insert the element at the specified position in the 2D array
arr2d_inserted = np.insert(arr2d, row_index, element, axis=0)
arr2d_inserted = np.insert(arr2d_inserted, col_index, element, axis=1)

print("Original 2D array:")
print(arr2d)

print("\n2D array after inserting the element:")
print(arr2d_inserted)
```

Nafeesath Neema

Roll no:42

batch no:MCA-2022-24

Enter elements of the 1D array separated by spaces: 1 3 4 5

Enter the element to insert: 56

Enter the position to insert (0-based index): 2

Original 1D array:

[1 3 4 5]

1D array after inserting the element:

[1 3 56 4 5]

Enter the number of rows: 3

Enter the number of columns: 2

Enter elements of the 2D array row by row:

1

4

5

Enter the row index to insert: 2

Enter the column index to insert: 1

Enter the element to insert: 34

Original 2D array:

[[1]

[4]

[5]]

2D array after inserting the element:

[[1 34]

[4 34]

[34 34]

[5 34]]

9. Demonstrate the use of diag() function in 1D and 2D array.(use both square matrix and matrix with different dimensions)

```
import numpy as np
print("\nNafeesath Neema\nRoll no:42\nbatch no:MCA-2022-24")

# User input for a 1D array
arr_1d = np.array(list(map(int, input("Enter elements for 1D array separated by
spaces: ").split()))))

# Using diag() (should be empty for 1D array)
diag_1d = np.diag(arr_1d)

print("\nOriginal 1D Array:")
print(arr_1d)

print("\nDiagonal Elements (empty for 1D array):")
print(diag_1d)

# User input for a square 2D array
n = int(input("Enter the size of the square matrix: "))
square_matrix = np.array([list(map(int, input().split())) for _ in range(n)])

# Using diag() to extract diagonal elements of the square matrix
diag_square_matrix = np.diag(square_matrix)

print("\nOriginal Square Matrix:")
print(square_matrix)

print("\nDiagonal Elements of Square Matrix:")
print(diag_square_matrix)
```

Nafeesath Neema

Roll no:42

batch no:MCA-2022-24

Enter elements for 1D array separated by spaces: 1 2 3 4

Original 1D Array:

[1 2 3 4]

Diagonal Elements (empty for 1D array):

[[1 0 0 0]

[0 2 0 0]

[0 0 3 0]

[0 0 0 4]]

Enter the size of the square matrix: 3

1 2 3

4 5 6

3 4 5

Original Square Matrix:

[[1 2 3]

[4 5 6]

[3 4 5]]

Diagonal Elements of Square Matrix:

[1 5 5]

10. Create a square matrix with random integer values(use randint()) and use appropriate functions to find:

- i) inverse
- ii) rank of matrix
- iii) Determinant
- iv) transform matrix into 1D array
- v) eigen values and vectors

```
import numpy as np
print("\nNafeesath Neema\nRoll no:42\nbatch no:MCA-2022-24")
# User input for the size of the square matrix
size = int(input("Enter the size of the square matrix: "))

# User input for the elements of the matrix
print(f"Enter {size}x{size} matrix:")
user_matrix = np.array([list(map(int, input().split())) for _ in range(size)])

print("\nUser Input Matrix:")
print(user_matrix)

# i) Inverse of the matrix
try:
    inverse_matrix = np.linalg.inv(user_matrix)
    print("\nInverse of the Matrix:")
    print(inverse_matrix)
except np.linalg.LinAlgError:
    print("\nMatrix is singular, and its inverse does not exist.")

# ii) Rank of the matrix
rank_matrix = np.linalg.matrix_rank(user_matrix)
print("\nRank of the Matrix:", rank_matrix)

# iii) Determinant of the matrix
determinant_matrix = np.linalg.det(user_matrix)
print("\nDeterminant of the Matrix:", determinant_matrix)

# iv) Transform matrix into a 1D array
flattened_matrix = user_matrix.flatten()
print("\nMatrix as 1D Array:")
print(flattened_matrix)

# v) Eigenvalues and Eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(user_matrix)
print("\nEigenvalues:")
print(eigenvalues)
print("\nEigenvectors:")
print(eigenvectors)
```

Output

```
Nafeesath Neema
Roll no:42
batch no:MCA-2022-24
Enter the size of the square matrix: 2
Enter 2x2 matrix:
1 2
3 4

User Input Matrix:
[[1 2]
 [3 4]]

Inverse of the Matrix:
[[-2.   1. ]
 [ 1.5 -0.5]]

Rank of the Matrix: 2

Determinant of the Matrix: -2.0000000000000004

Matrix as 1D Array:
[1 2 3 4]

Eigenvalues:
[-0.37228132  5.37228132]

Eigenvectors:
[[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
```


- 11.. Create a matrix X with suitable rows and columns
- i) Display the cube of each element of the matrix using different methods(use multiply(), *, power(),**)
 - ii) Display identity matrix of the given square matrix.
 - iii) Display each element of the matrix to different powers.

```
import numpy as np
print("\nNafeesath Neema\nRoll no:42\nbatch no:MCA-2022-24")
# Take user input for the dimensions of the matrix
rows = int(input("Enter the number of rows: "))
cols = int(input("Enter the number of columns: "))

# Take user input to populate the matrix
X = np.empty((rows, cols), dtype=int)
for i in range(rows):
    for j in range(cols):
        X[i, j] = int(input(f"Enter the element at row {i+1}, column {j+1}: "))

# i) Display the cube of each element of the matrix using different methods
# Method 1: Using multiplication
cubed_matrix_multiply = X * X * X

# Method 2: Using the power function
cubed_matrix_power = np.power(X, 3)

# Method 3: Using the double asterisk (**) operator
cubed_matrix_double_asterisk = X ** 3

# Display the results
print("\nCube of each element using multiplication:\n", cubed_matrix_multiply)
print("\nCube of each element using power function:\n", cubed_matrix_power)
print("\nCube of each element using double asterisk (**):\n",
cubed_matrix_double_asterisk)

# ii) Display the identity matrix of the given square matrix
if rows == cols:
    identity_matrix = np.identity(rows)
    print("\nIdentity matrix of X:\n", identity_matrix)
else:
    print("\nX is not a square matrix, so it doesn't have an identity matrix.")

# iii) Display each element of the matrix to different powers
powers = [2, 3, 4] # Powers to raise each element to
powered_matrices = [np.power(X, p) for p in powers]

# Display each powered matrix
for i, p in enumerate(powers):
    print(f"\nMatrix raised to the power {p}:\n", powered_matrices[i])
```


Nafeesath Neema

Roll no:42

batch no:MCA-2022-24

Enter the number of rows: 2

Enter the number of columns: 2

Enter the element at row 1, column 1: 1

Enter the element at row 1, column 2: 2

Enter the element at row 2, column 1: 3

Enter the element at row 2, column 2: 4

Cube of each element using multiplication:

```
[[ 1  8]
```

```
[27 64]]
```

Cube of each element using power function:

```
[[ 1  8]
```

```
[27 64]]
```

Cube of each element using double asterisk (**):

```
[[ 1  8]
```

```
[27 64]]
```

Identity matrix of X:

```
[[1. 0.]
```

```
[0. 1.]]
```

Matrix raised to the power 2:

```
[[ 1  4]
```

```
[ 9 16]]
```

Matrix raised to the power 3:

```
[[ 1  8]
```

```
[27 64]]
```

Matrix raised to the power 4:

```
[[ 1 16]
```

```
[ 81 256]]
```

11. Create a matrix Y with same dimension as X and perform the operation $X^2 + 2Y$

```
import numpy as np
print("\nNafeesath Neema\nRoll no:42\nbatch no:MCA-2022-24")
m = int(input("Enter the number of rows (m): "))
n = int(input("Enter the number of columns (n): "))
X = np.zeros((m, n))
print("Enter values for matrix X:")
for i in range(m):
    for j in range(n):
        X[i, j] = float(input(f"X[{i+1},{j+1}]: "))
Y = np.zeros((m, n))
print("Enter values for matrix Y:")
for i in range(m):
    for j in range(n):
        Y[i, j] = float(input(f"Y[{i+1},{j+1}]: "))
result = X**2 + 2*Y
print("Matrix X:")
print(X)
print("\nMatrix Y:")
print(Y)
print("\nResult of X^2 + 2Y:")
print(result)
```

Nafeesath Neema

Roll no:42

batch no:MCA-2022-24

Enter the number of rows (m): 2

Enter the number of columns (n): 2

Enter values for matrix X:

X[1,1]: 1

X[1,2]: 3

X[2,1]: 4

X[2,2]: 5

Enter values for matrix Y:

Y[1,1]: 6

Y[1,2]: 7

Y[2,1]: 5

Y[2,2]: 3

Matrix X:

[[1. 3.]

[4. 5.]]

Matrix Y:

[[6. 7.]

[5. 3.]]

Result of $X^2 + 2Y$:

[[13. 23.]

[26. 31.]]

12. Define matrices A with dimension 5x6 and B with dimension 3x3. Extract a sub matrix of dimension 3x3 from A and multiply it with B. Replace the extracted sub matrix in A with the matrix obtained after multiplication

```
import numpy as np
print("\nNafeesath Neema\nRoll no:42\nbatch no:MCA-2022-24")
A = np.array([[1, 2, 3, 4, 5, 6],
              [7, 8, 9, 10, 11, 12],
              [13, 14, 15, 16, 17, 18],
              [19, 20, 21, 22, 23, 24],
              [25, 26, 27, 28, 29, 30]])

B = np.array([[2, 3, 4],
              [5, 6, 7],
              [8, 9, 10]])

submatrix_A = A[:3, :3]

result = np.dot(submatrix_A, B)

A[:3, :3] = result

# Display the updated matrix A
print("Updated Matrix A:")
print(A)
print("\nResult after replacing the submatrix in A:")
print(A)
```

Nafeesath Neema

Roll no:42

batch no:MCA-2022-24

Updated Matrix A:

```
[[ 36  42  48   4   5   6]
 [126 150 174  10  11  12]
 [216 258 300  16  17  18]
 [ 19  20  21  22  23  24]
 [ 25  26  27  28  29  30]]
```

Result after replacing the submatrix in A:

```
[[ 36  42  48   4   5   6]
 [126 150 174  10  11  12]
 [216 258 300  16  17  18]
 [ 19  20  21  22  23  24]
 [ 25  26  27  28  29  30]]
```

13. Given 3 Matrices A, B and C. Write a program to perform matrix multiplication of the 3 matrices.

```
import numpy as np
print("\nNafeesath Neema\nRoll n0:42\nbatch no:MCA-2022-24")
A = np.array([[1, 2, 3],
              [4, 5, 6]])

B = np.array([[7, 8],
              [9, 10],
              [11, 12]])

C = np.array([[13, 14],
              [15, 16]])

result = np.dot(np.dot(A, B), C)

print("Result of Matrix Multiplication (A * B * C):")
print(result)
```

```
Nafeesath Neema
Roll n0:42
batch no:MCA-2022-24
Result of Matrix Multiplication (A * B * C):
[[1714 1836]
 [4117 4410]]
```


14. Write a program to check whether given matrix is symmetric or Skew Symmetric.

```
import numpy as np
print("\nNafeesath Neema\nRoll no:42\nbatch no:MCA-2022-24")
def is_symmetric(matrix):
    transpose = np.transpose(matrix)
    return np.array_equal(matrix, transpose)

def is_skew_symmetric(matrix):
    transpose = np.transpose(matrix)
    return np.array_equal(matrix, -transpose)

matrix = np.array([[0, 1, -2],
                   [-1, 0, 3],
                   [2, -3, 0]])

if is_symmetric(matrix):
    print("The matrix is symmetric.")
elif is_skew_symmetric(matrix):
    print("The matrix is skew-symmetric (antisymmetric).")
else:
    print("The matrix is neither symmetric nor skew-symmetric.")

def is_symmetric(matrix):
    transpose = np.transpose(matrix)
    return np.array_equal(matrix, transpose)

def is_skew_symmetric(matrix):
    transpose = np.transpose(matrix)
    return np.array_equal(matrix, -transpose)

matrix = np.array([[0, 1, -2],
                   [-1, 0, 3],
                   [2, -3, 0]])

if is_symmetric(matrix):
    print("The matrix is symmetric.")
elif is_skew_symmetric(matrix):
    print("The matrix is skew-symmetric (antisymmetric).")
else:
    print("The matrix is neither symmetric nor skew-symmetric.")
```

Nafeesath Neema

Roll no:42

batch no:MCA-2022-24

The matrix is skew-symmetric (antisymmetric).

The matrix is skew-symmetric (antisymmetric).

15. Given a matrix-vector equation $AX=b$. Write a program to find out the value of X using `solve()`, given A and b as below

$X=A^{-1} b$.

Note: Numpy provides a function called `solve` for solving such equations.

```
import numpy as np
print("\nNafeesath Neema\nRoll no:42\nbatch no:MCA-2022-24")
A = np.array([[2, 3, -1],
              [1, 2, 1],
              [3, 1, -2]])

b = np.array([7, 3, 8])

try:

    X = np.linalg.solve(A, b)

    print("Solution X:")
    print(X)
except np.linalg.LinAlgError:
    print("Matrix A is singular. The system of equations may not have a unique solution.")
```

```
Nafeesath Neema
Roll no:42
batch no:MCA-2022-24
Solution X:
[ 2.  0.8 -0.6]
```

16. Write a program to perform the SVD of a given matrix A. Also reconstruct the given matrix from the 3 matrices obtained after performing SVD.

```
import numpy as np
print("\nNafeesath Neema\nRoll n0:42\nbatch no:MCA-2022-24")
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

U, S, Vt = np.linalg.svd(A)

A_hat = U @ np.diag(S) @ Vt

print("Original Matrix A:")
print(A)
print("\nSingular Values:")
print(S)
print("\nReconstructed Matrix A_hat:")
print(A_hat)
```

```
Nafeesath Neema
Roll n0:42
batch no:MCA-2022-24
Original Matrix A:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Singular Values:
[1.68481034e+01 1.06836951e+00 3.33475287e-16]

Reconstructed Matrix A_hat:
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]
```