DATA SCIENCE & MACHINE LEARNING-LAB CYCLE 2

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

CODE:

```
print("SJC22MCA-2007 : ANJALA MICHAEL")
print("Batch : MCA 2022-24")

import numpy as np
# Create a 3D array of float data type
#  we create a 2x3x4 array (2 planes, each with 3 rows and 4
columns)
array_3d = np.array([ [[1.5, 2.6, 3.7, 4.8],[5.1, 6.2, 7.3,
8.4],[9.5, 10.6, 11.7, 12.8]], [[13.0, 14.0, 15.0, 16.0],[17.0,
18.0, 19.0, 20.0],[21.0, 22.0, 23.0, 24.0]] ], dtype=float)
# Print the 3D array
print("3D Array:")
print(array_3d)
```

OUTPUT:

```
SJC22MCA-2007 : ANJALA MICHAEL
Batch : MCA 2022-24
3D Array:
[[[ 1.5  2.6  3.7  4.8]
  [ 5.1  6.2  7.3  8.4]
  [ 9.5 10.6 11.7 12.8]]

 [[13.  14.  15.  16. ]
  [17.  18.  19.  20. ]
  [21.  22.  23.  24. ]]]

Process finished with exit code 0
```

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

CODE:

```
print("SJC22MCA-2007 : ANJALA MICHAEL")
print("Batch : MCA 2022-24")
import numpy as np
# Create a 2D array (2x3) with complex data type
array_2d = np.array([ [1 + 2j, 3 + 4j, 5 + 6j], [7 + 8j, 9 + 10j,
11 + 12j] ], dtype=complex)
# Print the 2D array
print("2D Array:")
print(array_2d)
# Display the number of rows and columns
rows, columns = array_2d.shape
print("\nNumber of Rows:", rows)
print("Number of Columns:", columns)
# Display the dimensions of the array
dimensions = array_2d.ndim
print("\nDimensions of the Array:", dimensions)
# Reshape the array to 3x2
reshaped_array = array_2d.reshape(3, 2)
# Print the reshaped array
print("\nReshaped Array (3x2):")
print(reshaped_array)
```

OUTPUT:

```
SJC22MCA-2007 : ANJALA MICHAEL
Batch : MCA 2022-24
2D Array:
[[ 1. +2.j  3. +4.j  5. +6.j]
 [ 7. +8.j  9.+10.j 11.+12.j]]

Number of Rows: 2
Number of Columns: 3

Dimensions of the Array: 2

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

CODE:

```
print("SJC22MCA-2007 : ANJALA MICHAEL")
print("Batch : MCA 2022-24")

import numpy as np
# Create an uninitialized array
uninitialized_array = np.empty(shape=(2, 3))
print("Uninitialized Array:")
print(uninitialized_array)

# Create an array with all elements as 1
ones_array = np.ones(shape=(2, 3))
print("Array with All Elements as 1:")
print(ones_array)

# Create an array with all elements as 0
zeros_array = np.zeros(shape=(2, 3))
print("Array with All Elements as 0:")
print(zeros_array)
```

OUTPUT:

```
SJC22MCA-2007 : ANJALA MICHAEL
Batch : MCA 2022-24
Uninitialized Array:
[[1.63888689e-316 0.00000000e+000 6.94863598e-310]
 [6.94863595e-310 6.01346953e-154 2.06675028e+151]]
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

CODE:

```python
print("SJC22MCA-2007 : ANJALA MICHAEL")
print("Batch : MCA 2022-24")

import numpy as np
# Create a one-dimensional array with 10 elements using arange
one_dimensional_array = np.arange(10)
# a. Display the first 4 elements
first_4_elements = one_dimensional_array[:4]
# b. Display the last 6 elements
last_6_elements = one_dimensional_array[-6:]
# c. Display elements from index 2 to 7
elements_2_to_7 = one_dimensional_array[2:8]
# Display the results
print("Original Array:", one_dimensional_array)
print("a. First 4 elements:", first_4_elements)
print("b. Last 6 elements:", last_6_elements)
print("c. Elements from index 2 to 7:", elements_2_to_7)
```

OUTPUT:

```
SJC22MCA-2007 : ANJALA MICHAEL
Batch : MCA 2022-24
Original Array: [0 1 2 3 4 5 6 7 8 9]
a. First 4 elements: [0 1 2 3]
b. Last 6 elements: [4 5 6 7 8 9]
c. Elements from index 2 to 7: [2 3 4 5 6 7]
```

5. Create an 1D array with arrange 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

CODE:

```
print("SJC22MCA-2007 : ANJALA MICHAEL")
print("Batch : MCA 2022-24")

import numpy as np
# Create a 1D array containing the first 15 even numbers as elements
even_numbers = np.arange(2, 31, 2)
# Elements from index 2 to 8 with step 2 using slice notation
slice_result = even_numbers[2:9:2]
# Last 3 elements of the array using negative index
last_3_elements = even_numbers[-3:]
# Alternate elements of the array
alternate_elements = even_numbers[::2]
# Display the last 3 alternate elements
last_3_alternate_elements = alternate_elements[-3:]
print("Original array:", even_numbers)
print("Elements from index 2 to 8 with step 2:", slice_result)
print("Last 3 elements of the array using negative index:",
last_3_elements)
print("Alternate elements of the array:", alternate_elements)
print("Last 3 alternate elements:", last_3_alternate_elements)
```

OUTPUT:

```
SJC22MCA-2007 : ANJALA MICHAEL
Batch : MCA 2022-24
Original array: [ 2  4  6  8 10 12 14 16 18 20 22 24 26 28 30]
Elements from index 2 to 8 with step 2: [ 6 10 14 18]
Last 3 elements of the array using negative index: [26 28 30]
Alternate elements of the array: [ 2  6 10 14 18 22 26 30]
Last 3 alternate elements: [22 26 30]
```

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)

CODE:

```
print("SJC22MCA-2007 : ANJALA MICHAEL")
print("Batch : MCA 2022-24")

import numpy as np
# Create a 2D array with 4 rows and 4 columns
two_dimensional_array = np.array([[1, 2, 3, 4],
                                  [5, 6, 7, 8],
                                  [9, 10, 11, 12],
                                  [13, 14, 15, 16]])
# Display all elements excluding the first row
excluding_first_row = two_dimensional_array[1:]
# Display all elements excluding the last column
excluding_last_column = two_dimensional_array[:, :-1]
# Display the elements of the 1st and 2nd column in the 2nd and 3rd row
column_1_2_in_row_2_3 = two_dimensional_array[1:3, 0:2]
# Display the elements of the 2nd and 3rd column
column_2_3 = two_dimensional_array[:, 1:3]
# Display the 2nd and 3rd element of the 1st row
elements_2_3_in_first_row = two_dimensional_array[0, 1:3]
# Display the elements from indices 4 to 10 in descending order
descending_order = two_dimensional_array.ravel()[::-1][4:11]
print("Original 2D array:\n", two_dimensional_array)
print("Elements excluding the first row:\n", excluding_first_row)
print("Elements excluding the last column:\n", excluding_last_column)
print("Elements of the 1st and 2nd column in the 2nd and 3rd row:\n",
column_1_2_in_row_2_3)
print("Elements of the 2nd and 3rd column:\n", column_2_3)
print("2nd and 3rd element of the 1st row:\n",
elements_2_3_in_first_row)
print("Elements from indices 4 to 10 in descending order:\n",
descending_order)
```

OUTPUT:

```
SJC22MCA-2007 : ANJALA MICHAEL
Batch : MCA 2022-24
Original 2D array:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
Elements excluding the first row:
 [[ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
Elements excluding the last column:
 [[ 1  2  3]
 [ 5  6  7]
 [ 9 10 11]
 [13 14 15]]
Elements of the 1st and 2nd column in the 2nd and 3rd row:
 [[ 5  6]
 [ 9 10]]
Elements of the 2nd and 3rd column:
 [[ 2  3]
 [ 6  7]
 [10 11]
 [14 15]]
2nd and 3rd element of the 1st row:
 [2 3]
Elements from indices 4 to 10 in descending order:
 [12 11 10  9  8  7  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

CODE:

```
print("SJC22MCA-2007 : ANJALA MICHAEL")
print("Batch : MCA 2022-24")

import numpy as np
# Create two 2D arrays
matrix1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
matrix2 = np.array([[9, 8, 7], [6, 5, 4], [3, 2, 1]])
# Add the two matrices
matrix_sum = matrix1 + matrix2
# Subtract the two matrices
matrix_diff = matrix1 - matrix2
# Multiply the individual elements of the matrices
matrix_product = matrix1 * matrix2
# Divide the elements of the matrices
matrix_divide = matrix1 / matrix2
# Perform matrix multiplication
matrix_multiply = np.dot(matrix1, matrix2)
# Display transpose of the matrix
matrix1_transpose = np.transpose(matrix1)
# Sum of diagonal elements of a matrix
diagonal_sum = np.trace(matrix1)
print("Matrix 1:\n", matrix1)
print("Matrix 2:\n", matrix2)
print("Matrix Sum:\n", matrix_sum)
print("Matrix Difference:\n", matrix_diff)
print("Matrix Element-wise Product:\n", matrix_product)
print("Matrix Element-wise Division:\n", matrix_divide)
print("Matrix Multiplication:\n", matrix_multiply)
print("Transpose of Matrix 1:\n", matrix1_transpose)
print("Sum of Diagonal Elements of Matrix 1:", diagonal_sum)
```

OUTPUT:

```
SJC22MCA-2007 : ANJALA MICHAEL
Batch : MCA 2022-24
Matrix 1:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Matrix 2:
 [[9 8 7]
 [6 5 4]
 [3 2 1]]
Matrix Sum:
 [[10 10 10]
 [10 10 10]
 [10 10 10]]
Matrix Difference:
 [[-8 -6 -4]
 [-2  0  2]
 [ 4  6  8]]

Matrix Element-wise Product:
 [[ 9 16 21]
 [24 25 24]
 [21 16  9]]
Matrix 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

CODE:

```
print("SJC22MCA-2007 : ANJALA MICHAEL")
print("Batch : MCA 2022-24")

import numpy as np
# Create a 1D array
array_1d = np.array([1, 2, 3, 4, 5])
print("\n\n1D Array before insertion:")
print(array_1d)
# Insert the value 6 at index 2
array_1d = np.insert(array_1d, 2, 6)
print("\n1D Array after insertion:")
print(array_1d)
# Create a 2D array
array_2d = np.array([[1, 2, 3],
                     [4, 5, 6]])
print("\nOriginal 2D Array:")
print(array_2d)
# Insert a row [7, 8, 9] at index 1 (between the existing rows)
array_2d = np.insert(array_2d, 1, [7, 8, 9], axis=0)
print("\n2D Array after insertions:")
print(array_2d)
```

OUTPUT:

```
SJC22MCA-2007 : ANJALA MICHAEL
Batch : MCA 2022-24


1D Array before insertion:
[1 2 3 4 5]

1D Array after insertion:
[1 2 6 3 4 5]

Original 2D Array:
[[1 2 3]
 [4 5 6]]

2D Array after insertions:
[[1 2 3]
 [7 8 9]
 [4 5 6]]
```

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

CODE:

```
print("SJC22MCA-2007 : ANJALA MICHAEL")
print("Batch : MCA 2022-24")

import numpy as np;
arr_id = np.array([1, 2, 3, 4, 5])
diagonal_matrix = np.diag(arr_id)
print("1D Array:")
print(arr_id)
print("\nDiagonal Matrix:")
print(diagonal_matrix)
arr_2d_square = np.array([[1, 2, 3],
                          [4, 5, 6],
                          [7, 8, 9]])
diagonal_elements = np.diag(arr_2d_square)
print("\n2D Square Matrix:")
print(arr_2d_square)
print("\nDiagonal Elements:")
print(diagonal_elements)
arr_2d_non_square = np.array([[1, 2, 3],
                              [4, 5, 6]])
diagonal_elements_non_square = np.diag(arr_2d_non_square)
print("\n2D Non-Square Matrix:")
print(arr_2d_non_square)
print("\nDiagonal Elements (Non-Square):")
print(diagonal_elements_non_square)
```

OUTPUT:

```
SJC22MCA-2007 : ANJALA MICHAEL
Batch : MCA 2022-24
1D Array:
[1 2 3 4 5]

Diagonal Matrix:
[[1 0 0 0 0]
 [0 2 0 0 0]
 [0 0 3 0 0]
 [0 0 0 4 0]
 [0 0 0 0 5]]

2D Square Matrix:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Diagonal Elements:
[1 5 9]

2D Non-Square Matrix:
[[1 2 3]
 [4 5 6]]

Diagonal Elements (Non-Square):
[1 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

CODE:

```
print("SJC22MCA-2007 : ANJALA MICHAEL")
print("Batch : MCA 2022-24")
import numpy as np;
matrix_size = 3
matrix = np.random.randint(10,20, size=(matrix_size, matrix_size))
print("Original Matrix:")
print(matrix)
if np.linalg.matrix_rank(matrix) == matrix_size:
   inverse_matrix = np.linalg.inv(matrix)
   print("\nInverse Matrix:")
   print(inverse_matrix)
else:
   print("\nThe matrix is not invertible (its rank is less than the
size).")
rank = np.linalg.matrix_rank(matrix)
print("\nRank of the Matrix:", rank)
determinant = np.linalg.det(matrix)
print("\nDeterminant of the Matrix:", determinant)
matrix_1d = matrix.flatten()
print("\nMatrix as 1D Array:")
print(matrix_1d)
eigenvalues, eigenvectors = np.linalg.eig(matrix)
print("\nEigenvalues:")
print(eigenvalues)
print("\nEigenvectors:")
print(eigenvectors)
```

OUTPUT:

```
SJC22MCA-2007 : ANJALA MICHAEL
Batch : MCA 2022-24
Original Matrix:
[[10 18 13]
 [19 15 13]
 [15 13 12]]

Inverse Matrix:
[[-0.05555556  0.23737374 -0.1969697 ]
 [ 0.16666667  0.37878788 -0.59090909]
 [-0.11111111 -0.70707071  0.96969697]]

Rank of the Matrix: 3
```

```
Determinant of the Matrix: -198.0

Matrix as 1D Array:
[10 18 13 19 15 13 15 13 12]

Eigenvalues:
[42.86453975 -6.5678454   0.70330565]

Eigenvectors:
[[-0.55860192 -0.7970779  -0.19342861]
 [-0.63167823  0.54335781 -0.50681664]
 [-0.53753744  0.26349405  0.84007278]]
```

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.

CODE:

```
print("SJC22MCA-2007 : ANJALA MICHAEL")
print("Batch : MCA 2022-24")

import numpy as np
X = np.array([[1, 2,3 ],
              [4, 5, 6],
              [7, 8, 9]])
X_cube_multiply = np.multiply(X, np.multiply(X, X))
X_cube_operator = X * X * X
X_cube_power = np.power(X, 3)
X_cube_double_star = X ** 3
identity_matrix = np.identity(X.shape[0])
X_power_2 = np.power(X, 2)
X_power_3 = np.power(X, 3)
X_power_4 = np.power(X, 4)

print("Original Matrix X:")
print(X)

print("\nCubed Matrix (Method 1 - multiply()):")
print(X_cube_multiply)

print("\nCubed Matrix (Method 2 - * operator):")
print(X_cube_operator)

print("\nCubed Matrix (Method 3 - power()):")
print(X_cube_power)

print("\nCubed Matrix (Method 4 - ** operator):")
print(X_cube_double_star)

print("\nIdentity Matrix:")
print(identity_matrix)

print("\nMatrix to Different Powers:")
print("X^2:")
print(X_power_2)
print("\nX^3:")
print(X_power_3)
print("\nX^4:")
print(X_power_4)
```

OUTPUT:

```
SJC22MCA-2007 : ANJALA MICHAEL
Batch : MCA 2022-24
Original Matrix X:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Cubed Matrix (Method 1 - multiply()):
[[  1   8  27]
 [ 64 125 216]
 [343 512 729]]

Cubed Matrix (Method 2 - * operator):
[[  1   8  27]
 [ 64 125 216]
 [343 512 729]]

Cubed Matrix (Method 3 - power()):
[[  1   8  27]
 [ 64 125 216]
 [343 512 729]]

Cubed Matrix (Method 4 - ** operator):
[[  1   8  27]
 [ 64 125 216]
 [343 512 729]]
```

```
Identity Matrix:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

Matrix to Different Powers:
X^2:
[[ 1  4  9]
 [16 25 36]
 [49 64 81]]

X^3:
[[  1   8  27]
 [ 64 125 216]
 [343 512 729]]

X^4:
[[   1   16   81]
 [ 256  625 1296]
 [2401 4096 6561]]
```

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

CODE:

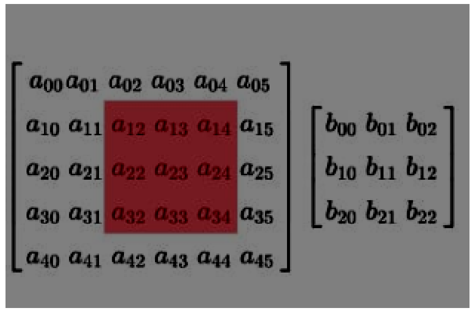
```
print("SJC22MCA-2007 : ANJALA MICHAEL")
print("Batch : MCA 2022-24")

import numpy as np;
X = np.array([[1, 2],
              [3, 4]])
Y = np.random.rand(*X.shape)
result = X * 2 + 2 * Y
print(result)
```

OUTPUT:

```
SJC22MCA-2007 : ANJALA MICHAEL
Batch : MCA 2022-24
[[2.28956727 5.04670424]
 [7.14639277 8.0837156 ]]
```

13. 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



CODE:

```
print("SJC22MCA-2007 : ANJALA MICHAEL")
print("Batch : MCA 2022-24")

import numpy as np
# Define matrix A with dimensions 5x6
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]])
print("Matrix A is : ")
print(A)

# Define matrix B with dimensions 3x3
B = np.array([[1,2,3,],[4,5,6],[7,8,9]])
print("Matrix B is : ")
print(B)

# Extract a 3x3 sub-matrix from A, for example, the top-left sub-matrix
sub_matrix = A[:3, :3]
print("The sub matrix is ")
print(sub_matrix)

# Multiply the sub-matrix with matrix B
result = np.dot(sub_matrix,B)
print("Matrix after multiplication with the sub matrix of A an d matrix
B")
print(result)

# Replace the extracted sub-matrix in A with the result
A[:3, :3] = result
print("Matrix A after the operation:")
print(A)
```

OUTPUT:

```
SJC22MCA-2007 : ANJALA MICHAEL
Batch : MCA 2022-24
Matrix A is :
[[ 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]]
Matrix B is :
[[1 2 3]
 [4 5 6]
 [7 8 9]]
The sub matrix is
[[ 1  2  3]
 [ 7  8  9]
 [13 14 15]]
Matrix after multiplication with the sub matrix of A an d matrix B
[[ 30  36  42]
 [102 126 150]
 [174 216 258]]
Matrix A after the operation:
[[ 30  36  42   4   5   6]
 [102 126 150  10  11  12]
 [174 216 258  16  17  18]
 [ 19  20  21  22  23  24]
 [ 25  26  27  28  29  30]]
```

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

CODE:

```
print("SJC22MCA-2007 : ANJALA MICHAEL")
print("Batch : MCA 2022-24")
import numpy as np
# Define matrices A, B, and C
A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])
B = np.array([[9, 8, 7],
              [6, 5, 4],
              [3, 2, 1]])
C = np.array([[10, 5, 9],
              [20, 15, 19],
              [30, 2, 29]])
# Perform matrix multiplication: (A * B) * C
result = np.dot(np.dot(A, B), C)
# Display the result
print("Matrix A:")
print(A)
print("Matrix B:")
print(B)
print("Matrix C:")
print(C)
print("Result of (A * B) * C:")
print(result)
```

OUTPUT:

```
SJC22MCA-2007 : ANJALA MICHAEL
Batch : MCA 2022-24
Matrix A:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Matrix B:
[[9 8 7]
 [6 5 4]
 [3 2 1]]
Matrix C:
[[10  5  9]
 [20 15 19]
 [30  2 29]]
Result of (A * B) * C:
[[1320  546 1248]
 [3840 1563 3633]
 [6360 2580 6018]]
```

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

CODE:

```
print("SJC22MCA-2007 : ANJALA MICHAEL")
print("Batch : MCA 2022-24")
import numpy as np
# Function to check if a matrix is symmetric
def is_symmetric(matrix):
   return (matrix == matrix.T).all()

# Function to check if a matrix is skew-symmetric
def is_skew_symmetric(matrix):
   return (matrix == -matrix.T).all()

# Define a matrix
matrix = np.array([[0, 1, -2],
                  [-1, 0, 3],
                  [2, -3, 0]])
# Check if the matrix is symmetric or skew-symmetric
if is_symmetric(matrix):
   print("The matrix is symmetric.")
elif is_skew_symmetric(matrix):
   print("The matrix is skew-symmetric.")
else:
   print("The matrix is neither symmetric nor skew-symmetric.")
```

OUTPUT:

```
SJC22MCA-2007 : ANJALA MICHAEL
Batch : MCA 2022-24
The matrix is skew-symmetric.
```

16. 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

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

$$A = \begin{bmatrix} 2 & 1 & -2 \\ 3 & 0 & 1 \\ 1 & 1 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -3 \\ 5 \\ -2 \end{bmatrix}$$

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

CODE:

```
print("SJC22MCA-2007 : ANJALA MICHAEL")
print("Batch : MCA 2022-24")

import numpy as np
# Define matrix A and vector b
A = np.array([[2, 1,-2],[3,0,1],[1,1,-1]])
b = np.array([-3,5,-2])
# Solve for X using the solve() function
X = np.linalg.solve(A, b)
print("Matrix A:")
print(A)
print("Vector b:")
print(b)
print("Solution for X:")
print(X)
```

OUTPUT:

```
SJC22MCA-2007 : ANJALA MICHAEL
Batch : MCA 2022-24
Matrix A:
[[ 2  1 -2]
 [ 3  0  1]
 [ 1  1 -1]]
Vector b:
[-3  5 -2]
Solution for X:
[ 1. -1.  2.]
```

17. 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.

Use the function: **numpy.linalg.svd()**

Singular value Decomposition

Matrix decomposition, also known as matrix factorization, involves describing a given matrix using its constituent elements.
The Singular-Value Decomposition, or SVD for short, is a matrix decomposition method for reducing a matrix to its constituent parts in order to make certain subsequent matrix calculations simpler. This approach is commonly used in reducing the no: of attributes in the given data set.

The SVD of mxn matrix A is given by the formula $A = U\Sigma V^T$

CODE:

```
print("SJC22MCA-2007 : ANJALA MICHAEL")
print("Batch : MCA 2022-24")

import numpy as np
A = np.array([[5, 27, 32], [14, 53, 62], [67, 88, 19]])
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)
```

OUTPUT:

```
SJC22MCA-2007 : ANJALA MICHAEL
Batch : MCA 2022-24
Original Matrix A :
[[ 5 27 32]
 [14 53 62]
 [67 88 19]]

Singular Values :
[135.69712478  52.97059904   1.18573314]

Reconstructed Matrix A_hat :
[[ 5. 27. 32.]
 [14. 53. 62.]
 [67. 88. 19.]]
```