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

CODE:

```
print("SJC22MCA-2024 : EMMANUEL.A")
print("BATCH : MCA")
import numpy as np

arr = np.array([[[2.1, 17.3], [45.7, 78.4]], [[88.5, 92.2], [60.5, 76.8]],
[[76.5, 33.5], [20.3, 18.2]]], dtype='float')
print(arr)
```

OUTPUT:

```
SJC22MCA-2024 : EMMANUEL.A
BATCH : MCA
[[[ 2.1 17.3]
  [45.7 78.4]]

 [[88.5 92.2]
  [60.5 76.8]]

 [[76.5 33.5]
  [20.3 18.2]]]
```

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-2023 : EMMANUEL.A")
print("BATCH : MCA")
import numpy as np
x = np.array([[1+2j, 2+3j, 3+4j], [4+1j, 5+2j, 6+3j]], dtype=complex)
print (x)
rows,columns=x.shape
print("number of rows",rows)
print("number of columns",columns)
```

```
print("array dimension is",x.shape)
reshape=x.reshape(3,2)
print("reshaped array is")
print(reshape)
```

OUTPUT:

```
SJC22MCA-2023 : EMMANUEL.A
BATCH : MCA
[[1.+2.j 2.+3.j 3.+4.j]
 [4.+1.j 5.+2.j 6.+3.j]]
number of rows 2
number of columns 3
```

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-2024 : EMMANUEL.A ")
print("BATCH : MCA")
import numpy as np
uninitialized_array = np.empty(shape=(2, 3))
print("Uninitialized Array:")
print(uninitialized_array)
ones_array = np.ones(shape=(2, 3))
print("Array with All Elements as 1:")
print(ones_array)
zeros_array = np.zeros(shape=(2, 3))
print("Array with All Elements as 0:")
print(zeros_array)
```

OUTPUT:

```
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:

```
print("SJC22MCA-2024 : EMMANUEL.A")
print("BATCH : MCA")
import numpy as np
one_dimensional_array = np.arange(10)
first_4_elements = one_dimensional_array[:4]
last_6_elements = one_dimensional_array[-6:]
elements_2_to_7 = one_dimensional_array[2:8]
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-2024 : EMMANUEL.A
BATCH : MCA
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-2024 : EMMANUEL")
print("BATCH : MCA")
import numpy as np
array=np.arange(2,32,2)
print(array)
step_size=array[2:9:2]
print("a.index 2 to 8 with step 2:",step_size)
result_slice=array[2:9:2]
print("a.1 using slice function:",result_slice)
last_3_elements = array[-3:]
print("b. Last 3 elements:", last_3_elements)
result=array[::2]
print("c.alternate elements:",result)
result=array[-1::-2][:3]
print("d.last 3 alternative elements:",result)
```

OUTPUT:

```
SJC22MCA-2024 : EMMANUEL
BATCH : MCA
[ 2  4  6  8 10 12 14 16 18 20 22 24 26 28 30]
a.index 2 to 8 with step 2: [ 6 10 14 18]
a.1 using slice function: [ 6 10 14 18]
b. Last 3 elements: [26 28 30]
c.alternate elements: [ 2  6 10 14 18 22 26 30]
d.last 3 alternative elements: [30 26 22]
```

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-2023 : EMMANUEL")
print("BATCH : MCA")
import numpy as np
twodim_array = np.array([
    [1, 2, 3, 4],
```

```
    [5, 6, 7, 8],
    [9, 10, 11, 12],
    [13, 14, 15, 16]
])
result=twodim_array[1:]
print("a.elements excluding the first row:")
print(result)
result=twodim_array[:,:-1]
print("b.elements excluding the last column:")
print(result)
result=twodim_array[1:3,0:2]
print("c.elements of 1 st and 2 nd column in 2 nd and 3 rd row")
print(result)
result=twodim_array[:,1:3]
print("d.elements of 2 nd and 3 rd column")
print(result)
result=twodim_array[0,1:3]
print("e.2 nd and 3 rd element of 1 st row")
print(result)
result=twodim_array.flatten()[10:3:-1]
print("f.elements from indices 4 to 10 in descending order")
print(result)
```

OUTPUT:

```
SJC22MCA-2023 : EMMANUEL
BATCH : MCA
a.elements excluding the first row:
[[ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
b.elements excluding the last column:
[[ 1  2  3]
 [ 5  6  7]
 [ 9 10 11]
 [13 14 15]]
c.elements of 1 st and 2 nd column in 2 nd and 3 rd row
[[ 5  6]
 [ 9 10]]
d.elements of 2 nd and 3 rd column
[[ 2  3]
 [ 6  7]
 [10 11]
 [14 15]]
```

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-2024 : EMMANUEL.A")
print("BATCH : MCA")
import numpy as np
```

```
matrix1 = np.array([[1, 2, 3],
                    [4, 5, 6],
                    [7, 8, 9]])
matrix2 = np.array([[9, 8, 7],
                    [6, 5, 4],
                    [3, 2, 1]])
result = matrix1 + matrix2
print("Matrix Addition:")
print(result)
result = matrix1 - matrix2
print("\nMatrix Subtraction:")
print(result)
result = matrix1 * matrix2
print("\nMatrix  Multiplication:")
print(result)
result = matrix1 / matrix2
print("\nMatrix  Division:")
print(result)
result = np.dot(matrix1, matrix2)
print("\nMatrix Multiplication:")
print(result)
result = matrix1.T
print("\nMatrix Transpose:")
print(result)
diagonal_sum = np.trace(matrix1)
print("\nSum of Diagonal Elements:")
print(diagonal_sum)
```

Output:

```
SJC22MCA-2024 : EMMANUEL.A
BATCH : MCA
Matrix Addition:
[[10 10 10]
 [10 10 10]
 [10 10 10]]

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

Matrix  Multiplication:
[[ 9 16 21]
 [24 25 24]
 [21 16  9]]

Matrix  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]]
```

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

CODE:

```
print("SJC22MCA-2023 : EMMANUEL")
print("BATCH : MCA")
import numpy as np
```

```python
arr1 = np.array([1, 2, 3, 4, 5])
new_arr1 = np.insert(arr1, 2, 6)
print("Original 1D Array:")
print(arr1)
print("\n1D Array After Insertion:")
print(new_arr1)
arr2d = np.array([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])
new_arr2d = np.insert(arr2d, 1, [10, 11, 12], axis=0)
new_arr2d = np.insert(new_arr2d, 2, [0, 0, 0, 0], axis=1)
print("Original 2D Array:")
print(arr2d)
print("\n2D Array After Insertion:")
print(new_arr2d)
```

OUTPUT:

```
SJC22MCA-2023 : EMMANUEL
BATCH : MCA
Original 1D Array:
[1 2 3 4 5]

1D Array After Insertion:
[1 2 6 3 4 5]
Original 2D Array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

2D Array After Insertion:
[[ 1  2  0  3]
 [10 11  0 12]
 [ 4  5  0  6]
 [ 7  8  0  9]]
```

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

CODE:

```
print("SJC22MCA-2023 : EMMANUEL")
print("BATCH : MCA")
import numpy as np
print("1D array:")
arr1d=np.array([1, 2, 3, 4, 5])
diagonal_matrix=np.diag(arr1d)
print("Original 1D Array:")
print(arr1d)
print("Diagonal Matrix:")
print(diagonal_matrix)
print("2D array:")
arr2=np.array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]])
diagonal_matrix=np.diag(arr2)
print("Original 1D Array:")
print(arr2)
print("Diagonal Matrix:")
print(diagonal_matrix)


square_matrix=np.array([[1, 2, 3],
                        [4, 5, 6],
                        [7, 8, 9]])
diagonal_elements=np.diag(square_matrix)
print("Original Square Matrix:")
print(square_matrix)
print("Diagonal Elements (1D Array):")
print(diagonal_elements)
non_square_matrix=np.array([[1, 2, 3],
                            [4, 5, 6]])
diagonal_elements=np.diag(non_square_matrix)
print("Original Non-Square Matrix:")
print(non_square_matrix)
print("Diagonal Elements (1D Array):")
print(diagonal_elements)
```

OUTPUT:

```
SJC22MCA-2023 : EMMANUEL
BATCH : MCA
1D array:
Original 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 array:
Original 1D Array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Diagonal Matrix:
[1 5 9]
Original Square Matrix:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Diagonal Elements (1D Array):
[1 5 9]
Original Non-Square Matrix:
[[1 2 3]
 [4 5 6]]
```

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-2023 : EMMANUEL.A" )
print("BATCH : MCA")
import numpy as np
matrix_size = 3
min_value = 1
max_value = 10
random_matrix = np.random.randint(min_value, max_value + 1, size=(matrix_size,
matrix_size))
print("Random Square Matrix:")
print(random_matrix)
inverse_matrix = np.linalg.inv(random_matrix)
print("\nInverse Matrix:")
print(inverse_matrix)
matrix_rank = np.linalg.matrix_rank(random_matrix)
print("\nRank of the Matrix:", matrix_rank)
matrix_determinant = np.linalg.det(random_matrix)
print("\nDeterminant of the Matrix:", matrix_determinant)
matrix_1d = random_matrix.flatten()
print("\nMatrix as 1D Array:")
print(matrix_1d)
eigenvalues, eigenvectors = np.linalg.eig(random_matrix)
print("\nEigenvalues:")
print(eigenvalues)
print("\nEigenvectors:")
print(eigenvectors)
```

OUTPUT:

```
SJC22MCA-2023 : EMMANUEL.A
BATCH : MCA
Random Square Matrix:
[[10  2  2]
 [ 1 10  8]
 [ 2  4  8]]

Inverse Matrix:
[[ 0.10344828 -0.01724138 -0.00862069]
 [ 0.01724138  0.1637931  -0.16810345]
 [-0.03448276 -0.07758621  0.2112069 ]]

Rank of the Matrix: 3

Determinant of the Matrix: 464.0000000000004

Matrix as 1D Array:
[10  2  2  1 10  8  2  4  8]

Eigenvalues:
[15.82319496  8.87131562  3.30548941]

Eigenvectors:
[[ 0.42951231  0.88930795  0.03893041]
 [ 0.75482173 -0.45488873 -0.76871526]
 [ 0.49574523 -0.04698527  0.63840526]]
```

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

<u>CODE:</u>

```python
print("SJC22MCA-2023:  EMMANUEL.A ")
print("BATCH : MCA")
import numpy as np
X = np.array([[1, 2, 3],
            [4, 5, 6],
            [7, 8, 9]])
# i) Display the cube of each element of the matrix using different methods
# Using np.multiply()
cubed_X1 = np.multiply(X, np.multiply(X, X))
# Using the * operator
cubed_X2 = X * X * X
# Using np.power()
cubed_X3 = np.power(X, 3)
# Using the ** operator
cubed_X4 = X ** 3
print("Cube of each element using np.multiply():")
print(cubed_X1)
print("\nCube of each element using * operator:")
print(cubed_X2)
print("\nCube of each element using np.power():")
print(cubed_X3)
print("\nCube of each element using ** operator:")
print(cubed_X4)
# ii) Display identity matrix of the given square matrix
identity_matrix = np.identity(X.shape[0])
print("\nIdentity Matrix of X:")
print(identity_matrix)
# iii) Display each element of the matrix to different powers
exponents = np.array([[2, 3, 4],
                    [3, 2, 1],
                    [0.5, 0.25, 0.1]])
powered_X = np.power(X, exponents)
print("\nMatrix X to Different Powers:")
print(powered_X)
# Create matrix Y with the same dimensions as X
Y = np.random.randint(1, 10, size=X.shape)
# Perform the operation X^2 + 2Y
result = np.power(X, 2) + 2 * Y
print("\nResult of X^2 + 2Y:")
print(result)
```

OUTPUT:

```
SJC22MCA-2023:  EMMANUEL.A
BATCH : MCA
Cube of each element using np.multiply():
[[  1   8  27]
 [ 64 125 216]
 [343 512 729]]

Cube of each element using * operator:
[[  1   8  27]
 [ 64 125 216]
 [343 512 729]]

Cube of each element using np.power():
[[  1   8  27]
 [ 64 125 216]
 [343 512 729]]

Cube of each element using ** operator:
[[  1   8  27]
 [ 64 125 216]
 [343 512 729]]

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

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.

CODE:

```
rint("SJC22MCA-2023:    EMMANUEL.A")
```

```python
print("BATCH : MCA")
import numpy as np
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)
print("after multiply",result)
A[:3, :3] = result
print("Updated Matrix A:")
print(A)
```

OUTPUT:

```
SJC22MCA-2023:    EMMANUEL.A
BATCH : MCA
after multiply [[ 36  42  48]
 [126 150 174]
 [216 258 300]]
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]]
```

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

<u>CODE:</u>

```
print("SJC22MCA-2023: EMMANUEL.A")
print("BATCH : MCA")
import numpy as np
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)
```

OUTPUT:

```
SJC22MCA-2023: EMMANUEL.A
BATCH : MCA
Result of Matrix Multiplication (A * B * C):
[[1714 1836]
 [4117 4410]]
```

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

CODE:

```
print("SJC22MCA-2023:  EMMANUEL.A")
print("BATCH : MCA")
import numpy as np
def symmetric(matrix):
  transpose = np.transpose(matrix)
  return np.array_equal(matrix, transpose)
def 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 symmetric(matrix):
  print("The matrix is symmetric.")
elif skew_symmetric(matrix):
  print("The matrix is skew-symmetric (antisymmetric).")
else:
  print("The matrix is neither symmetric nor skew-symmetric.")
```

OUTPUT:

```
SJC22MCA-2023:  EMMANUEL.A
BATCH : MCA
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.

CODE:

```
print("SJC22MCA-2023:   EMMANUEL.A")
print("BATCH : MCA")
import numpy as np
A = np.array([[2, 1, -2],
             [3, 0, 1],
             [1, 1, -1]])
b = np.array([-3, 5, 2])
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.")
```

OUTPUT:

```
SJC22MCA-2023:   EMMANUEL.A
BATCH : MCA
Solution X:
[-2.96059473e-16  7.00000000e+00  5.00000000e+00]
```

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.
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

<u>CODE:</u>

```
print("SJC22MCA-2023: EMMANUEL.A")
print("BATCH : MCA")
import numpy as np
# Define the matrix A (replace with your own matrix)
A = np.array([[1, 2, 3],
            [4, 5, 6],
            [7, 8, 9]])
# Perform SVD on matrix A
U, S, VT = np.linalg.svd(A)
# Reconstruct the original matrix from the three matrices obtained in SVD
reconstructed_A = np.dot(U, np.dot(np.diag(S), VT))
# Print the original matrix A, SVD components, and the reconstructed matrix
print("Original Matrix A:")
print(A)
```

```
print("\nMatrix U:")
print(U)
print("\nSingular Values S:")
print(S)
print("\nMatrix VT (Transpose of V):")
print(VT)
print("\nSVD Reconstructed Matrix A:")
print(reconstructed_A)
```

OUTPUT:

```
SJC22MCA-2023: EMMANUEL.A
BATCH : MCA
Original Matrix A:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Matrix U:
[[-0.21483724  0.88723069  0.40824829]
 [-0.52058739  0.24964395 -0.81649658]
 [-0.82633754 -0.38794278  0.40824829]]

Singular Values S:
[1.68481034e+01 1.06836951e+00 4.41842475e-16]

Matrix VT (Transpose of V):
[[-0.47967118 -0.57236779 -0.66506441]
 [-0.77669099 -0.07568647  0.62531805]
 [-0.40824829  0.81649658 -0.40824829]]

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