LAB CYCLE 2

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

CODE:

```
import numpy as np
print("University No: SJC22MCA-2021 \nName: Christin Benny \nBatch: S3 MCA \n________________________________________\n")
array_3d = np.array([
    [
        [1.0, 2.0, 3.0, 4.0],
        [5.0, 6.0, 7.0, 8.0],
        [9.0, 10.0, 11.0, 12.0]
    ],
    [
        [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("3D Array:")
print(array_3d)
```

OUTPUT

```
/home/sjcet/anaconda3/envs/untitled/bin/python /home/sjcet/christinmca22/Data Science/Cycle 2/q1.py
University No: SJC22MCA-2021
Name: Christin Benny
Batch: S3 MCA
--------------------------------------

3D 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.]]]

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:

```
import numpy as np
print("University No: SJC22MCA-2021 \nName: Christin Benny \nBatch: S3 MCA \n________________________________________\n")
complex_array = np.array([
   [1 + 2j, 2 + 3j, 3 + 4j],
   [4 + 5j, 5 + 6j, 6 + 7j]
], dtype=complex)

print("2D Array:")
print(complex_array)

rows, columns = complex_array.shape

print("Number of rows:", rows)
print("Number of columns:", columns)
```

print("Array dimension:", complex_array.shape)

reshaped_array = complex_array.reshape(3, 2)

print("Reshaped 3x2 Array:")
print(reshaped_array)

OUTPUT

```
/home/sjcet/anaconda3/envs/untitled/bin/python /home/sjcet/christinmca22/Data Science/Cycle 2/q2.py
University No: SJC22MCA-2021
Name: Christin Benny
Batch: S3 MCA
--------------------------------------

2D Array:
[[1.+2.j 2.+3.j 3.+4.j]
 [4.+5.j 5.+6.j 6.+7.j]]
Number of rows: 2
Number of columns: 3
Array dimension: (2, 3)
Reshaped 3x2 Array:
[[1.+2.j 2.+3.j]
 [3.+4.j 4.+5.j]
 [5.+6.j 6.+7.j]]

Process finished with exit code 0
```

3. Familiarize with the functions to create

a) an uninitialized array

b) array with all elements as 1,

c) all elements as 0

CODE:

import numpy as np

print("University No: SJC22MCA-2021 \nName: Christin Benny \nBatch: S3 MCA \n_______________________________________\n")

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)
```

<u>OUTPUT</u>

```
/home/sjcet/anaconda3/envs/untitled/bin/python /home/sjcet/christinmca22/Data Science/Cycle 2/q3.py
University No: SJC22MCA-2021
Name: Christin Benny
Batch: S3 MCA
--------------------------------------

Uninitialized Array:
[[6.89990592e-310 6.89990592e-310 6.89990552e-310]
 [6.89990551e-310 6.89990545e-310 6.89989667e-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.]]

Process finished with exit code 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

<u>CODE:</u>

```
import numpy as np
```

```
print("University No: SJC22MCA-2021 \nName: Christin Benny \nBatch: S3 MCA
\n______________________________________\n")

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

```
/home/sjcet/anaconda3/envs/untitled/bin/python /home/sjcet/christinmca22/Data Science/Cycle 2/q4.py
University No: SJC22MCA-2021
Name: Christin Benny
Batch: S3 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]

Process finished with exit code 0
```

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

CODE:

```
import numpy as np

print("University No: SJC22MCA-2021 \nName: Christin Benny \nBatch: S3 MCA \n_______________________________________\n")

even_numbers = np.arange(2, 32, 2)
print(even_numbers)

elements_from_2_to_8_step_2 = even_numbers[2:9:2]

print("a. Elements from index 2 to 8 with step 2:", elements_from_2_to_8_step_2)

last_3_elements = even_numbers[-3:]

print("b. Last 3 elements of the array using negative index:", last_3_elements)

alternate_elements = even_numbers[::2]

print("c. Alternate elements of the array:", alternate_elements)

last_3_alternate_elements = even_numbers[-1::-2][:3]

print("d. Last 3 alternate elements of the array:", last_3_alternate_elements)
```

OUTPUT

```
/home/sjcet/anaconda3/envs/untitled/bin/python /home/sjcet/christinmca22/Data Science/Cycle 2/q5.py
University No: SJC22MCA-2021
Name: Christin Benny
Batch: S3 MCA
--------------------------------------

[ 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. Last 3 alternate elements of the array: [30 26 22]

Process finished with exit code 0
```

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:

```
import numpy as np


print("University No: SJC22MCA-2021 \nName: Christin Benny \nBatch: S3 MCA
\n____________________________________________\n")


array_2d = np.array([[1, 2, 3, 4],
                     [5, 6, 7, 8],
                     [9, 10, 11, 12],
                     [13, 14, 15, 16]])
print(array_2d)
```

```python
a_result = array_2d[1:, :]
print("\nAll elements excluding the first row: ")
print(a_result)

b_result = array_2d[:, :-1]
print("\nAll elements excluding the last column: ")
print(b_result)

c_result = array_2d[1:3, 0:2]
print("\nElements of the 1st and 2nd column in the 2nd and 3rd row: ")
print(c_result)

d_result = array_2d[:, 1:3]
print("\nElements of the 2nd and 3rd column: ")
print(d_result)

e_result = array_2d[0, 1:3]
print("\n2nd and 3rd element of the 1st row: ")
print(e_result)

f_result = array_2d[1:3, 0:3].flatten()[::-1]
print("\nElements from indices 4 to 10 in descending order: ")
print(f_result)
```

OUTPUT

```
/home/sjcet/anaconda3/envs/untitled/bin/python /home/sjcet/christinmca22/Data Science/Cycle 2/6.py
University No: SJC22MCA-2021
Name: Christin Benny
Batch: S3 MCA
--------------------------------------

[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]

All elements excluding the first row:
[[ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]

All 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:
[11 10  9  7  6  5]
```

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:

```
import numpy as np

print("University No: SJC22MCA-2021 \nName: Christin Benny \nBatch: S3 MCA \n_______________________________________\n")


matrix1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

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


addition_result = matrix1 + matrix2

print("a. Addition Result:")

print(addition_result)


subtraction_result = matrix1 - matrix2

print("b. Subtraction Result:")

print(subtraction_result)


multiplication_result = matrix1 * matrix2

print("c. Multiplication Result:")

print(multiplication_result)


epsilon = 1e-15

division_result = np.divide(matrix1, matrix2 + epsilon)

print("d. Division Result:")

print(division_result)
```

```
matrix_multiplication_result = np.dot(matrix1, matrix2)
print("e. Matrix Multiplication Result:")
print(matrix_multiplication_result)

transpose_result = np.transpose(matrix1)
print("f. Transpose of the Matrix:")
print(transpose_result)

diagonal_sum = np.trace(matrix1)
print("g. Sum of Diagonal Elements:")
print(diagonal_sum)
```

OUTPUT

```
/home/sjcet/anaconda3/envs/untitled/bin/python /home/sjcet/christinmca22/Data Science/Cycle 2/7.py
University No: SJC22MCA-2021
Name: Christin Benny
Batch: S3 MCA
--------------------------------------

a. Addition Result:
[[10 10 10]
 [10 10 10]
 [10 10 10]]
b. Subtraction Result:
[[-8 -6 -4]
 [-2  0  2]
 [ 4  6  8]]
c. Multiplication Result:
[[ 9 16 21]
 [24 25 24]
 [21 16  9]]
d. Division Result:
[[0.11111111 0.25       0.42857143]
 [0.66666667 1.         1.5       ]
 [2.33333333 4.         9.        ]]
e. Matrix Multiplication Result:
[[ 30  24  18]
 [ 84  69  54]
 [138 114  90]]
f. Transpose of the Matrix:
[[1 4 7]
 [2 5 8]
 [3 6 9]]
g. Sum of Diagonal Elements:
15
```

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

CODE:

```
import numpy as np

print("University No: SJC22MCA-2021 \nName: Christin Benny \nBatch: S3 MCA
\n__________________________________________\n")


arr1d = np.array([1, 2, 3, 4, 5])

inserted_arr = np.insert(arr1d, 2, 6)


print("Original 1D Array:")

print(arr1d)
```

```
print("\n1D Array after Insertion:")
print(inserted_arr)

import numpy as np


arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

inserted_arr = np.insert(arr2d, 1, [10, 11, 12], axis=0)

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

print("\n2D Array after Insertion:")
print(inserted_arr)
```

OUTPUT

```
/home/sjcet/anaconda3/envs/untitled/bin/python /home/sjcet/christinmca22/Data Science/Cycle 2/8.py
University No: SJC22MCA-2021
Name: Christin Benny
Batch: S3 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  3]
 [10 11 12]
 [ 4  5  6]
 [ 7  8  9]]

Process finished with exit code 0
```

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

<u>CODE:</u>

```
import numpy as np

print("University No: SJC22MCA-2021 \nName: Christin Benny \nBatch: S3 MCA
\n_______________________________________\n")

A = np.array([1, 2, 3, 4, 5])

D = np.diag(A)

print("Original 1D Array:")

print(A)

print("\nDiagonal Matrix:")

print(D)
```

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

D_square = np.diag(B)

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

print("\nDiagonal Elements:")
print(D_square)

C = np.array([[1, 2, 3],
              [4, 5, 6]])

D_nonsquare = np.diag(C)
print("\nOriginal Non-Square Matrix:")
print(C)

print("\nDiagonal Matrix from Non-Square Matrix:")
print(D_nonsquare)
```

OUTPUT

```
/home/sjcet/anaconda3/envs/untitled/bin/python /home/sjcet/christinmca22/Data Science/Cycle 2/9.py
University No: SJC22MCA-2021
Name: Christin Benny
Batch: S3 MCA
--------------------------------------

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]]

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

Diagonal Elements:
[1 5 9]

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

Diagonal Matrix from Non-Square Matrix:
[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:

```
import numpy as np
print("University No: SJC22MCA-2021 \nName: Christin Benny \nBatch: S3 MCA \n________________________________________\n")
matrix_size = 3
```

```
random_matrix = np.random.randint(1, 11, size=(matrix_size, matrix_size))

print("Random Square Matrix:")
print(random_matrix)

try:
    inverse_matrix = np.linalg.inv(random_matrix)
    print("\nInverse Matrix:")
    print(inverse_matrix)
except np.linalg.LinAlgError:
    print("\nInverse does not exist for this matrix.")

rank = np.linalg.matrix_rank(random_matrix)
print("\nRank of the Matrix:", rank)

determinant = np.linalg.det(random_matrix)
print("\nDeterminant of the Matrix:", determinant)

matrix_1d = random_matrix.flatten()
print("\nMatrix as a 1D Array:")
print(matrix_1d)

eigenvalues, eigenvectors = np.linalg.eig(random_matrix)
print("\nEigenvalues:")
print(eigenvalues)
print("\nEigenvectors:")
print(eigenvectors)
```

OUTPUT

```
/home/sjcet/anaconda3/envs/untitled/bin/python /home/sjcet/christinmca22/Data Science/Cycle 2/10.py
University No: SJC22MCA-2021
Name: Christin Benny
Batch: S3 MCA
--------------------------------------

Random Square Matrix:
[[4 2 1]
 [9 1 1]
 [4 3 5]]

Inverse Matrix:
[[-0.03921569  0.1372549  -0.01960784]
 [ 0.80392157 -0.31372549 -0.09803922]
 [-0.45098039  0.07843137  0.2745098 ]]

Rank of the Matrix: 3

Determinant of the Matrix: -51.0

Matrix as a 1D Array:
[4 2 1 9 1 1 4 3 5]

Eigenvalues:
[ 8.90832691  3.         -1.90832691]

Eigenvectors:
[[ 0.36799882  0.18257419  0.27324531]
 [ 0.51656026  0.36514837 -0.93004529]
 [ 0.773138   -0.91287093  0.24566797]]

Process finished with exit code 0
```

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.

iv) Create a matrix Y with same dimension as X and perform the operation X^2 +2Y

<u>CODE:</u>

```
import numpy as np

print("University No: SJC22MCA-2021 \nName: Christin Benny \nBatch: S3 MCA
\n________________________________________\n")

X = np.array([[1, 2, 3],

          [4, 5, 6],
```

```
              [7, 8, 9]])

cubed_matrix1 = np.power(X, 3)

cubed_matrix2 = X ** 3

cubed_matrix3 = np.multiply(X, X, X)

cubed_matrix4 = X * X * X

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

print("\nCube of each element (using np.power()):")
print(cubed_matrix1)

print("\nCube of each element (using ** operator):")
print(cubed_matrix2)

print("\nCube of each element (using np.multiply()):")
print(cubed_matrix3)

print("\nCube of each element (using * operator):")
print(cubed_matrix4)

identity_matrix = np.identity(X.shape[0])
print("\nIdentity Matrix of X:")
print(identity_matrix)
```

```
exponentials = [2, 3, 4]

powered_matrices = [np.power(X, exp) for exp in exponentials]

for i, exp in enumerate(exponentials):
    print(f"\nMatrix X to the power of {exp}:")
    print(powered_matrices[i])
```

```
/home/sjcet/anaconda3/envs/untitled/bin/python /home/sjcet/christinmca22/Data Science/Cycle 2/11.py
University No: SJC22MCA-2021
Name: Christin Benny
Batch: S3 MCA
--------------------------------------

Matrix X:
[[ 1  4  9]
 [16 25 36]
 [49 64 81]]

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]]

Cube of each element (using np.multiply()):
[[ 1  4  9]
 [16 25 36]
 [49 64 81]]

Cube of each element (using * operator):
[[     1     64    729]
 [  4096  15625  46656]
 [117649 262144 531441]]
```

```
Cube of each element (using np.multiply()):
[[ 1  4  9]
 [16 25 36]
 [49 64 81]]

Cube of each element (using * operator):
[[     1     64    729]
 [  4096  15625  46656]
 [117649 262144 531441]]

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

Matrix X to the power of 2:
[[   1   16   81]
 [ 256  625 1296]
 [2401 4096 6561]]

Matrix X to the power of 3:
[[     1     64    729]
 [  4096  15625  46656]
 [117649 262144 531441]]

Matrix X to the power of 4:
[[       1      256     6561]
 [   65536   390625  1679616]
 [ 5764801 16777216 43046721]]
```

```
import numpy as np


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



cubed_matrix1 = np.power(X, 3)


cubed_matrix2 = X ** 3


cubed_matrix3 = np.multiply(X, np.multiply(X, X))


cubed_matrix4 = X * X * X


print("Matrix X:")
```

```
print(X)

print("\nCube of each element (using np.power()):")
print(cubed_matrix1)

print("\nCube of each element (using ** operator):")
print(cubed_matrix2)

print("\nCube of each element (using np.multiply()):")
print(cubed_matrix3)

print("\nCube of each element (using * operator):")
print(cubed_matrix4)

identity_matrix = np.identity(X.shape[0])
print("\nIdentity Matrix of X:")
print(identity_matrix)

exponentials = [2, 3, 4]

powered_matrices = [np.power(X, exp) for exp in exponentials]

for i, exp in enumerate(exponentials):
    print(f"\nMatrix X to the power of {exp}:")
    print(powered_matrices[i])
```

OUTPUT

```
/home/sjcet/anaconda3/envs/untitled/bin/python /home/sjcet/christinmca22/Data Science/Cycle 2/11 a.py
Matrix X:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

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]]

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]]

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

```
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]]

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

Matrix X to the power of 2:
[[ 1  4  9]
 [16 25 36]
 [49 64 81]]

Matrix X to the power of 3:
[[  1   8  27]
 [ 64 125 216]
 [343 512 729]]

Matrix X to the power of 4:
[[   1   16   81]
 [ 256  625 1296]
 [2401 4096 6561]]
```

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

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{bmatrix} \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$

<u>CODE:</u>

```
import numpy as np
print("University No: SJC22MCA-2021 \nName: Christin Benny \nBatch: S3 MCA
\n_______________________________________\n")
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

print("Updated Matrix A:")
print(A)
```

OUTPUT

```
/home/sjcet/anaconda3/envs/untitled/bin/python /home/sjcet/christinmca22/Data Science/Cycle 2/12 .py
University No: SJC22MCA-2021
Name: Christin Benny
Batch: S3 MCA
--------------------------------------

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]]

Process finished with exit code 0
```

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

CODE:

```
import numpy as np

print("University No: SJC22MCA-2021 \nName: Christin Benny \nBatch: S3 MCA
\n________________________________________\n")

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

```
/home/sjcet/anaconda3/envs/untitled/bin/python /home/sjcet/christinmca22/Data Science/Cycle 2/13 .py
University No: SJC22MCA-2021
Name: Christin Benny
Batch: S3 MCA
--------------------------------------

Result of Matrix Multiplication (A * B * C):
[[1714 1836]
 [4117 4410]]

Process finished with exit code 0
```

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

CODE:

```
import numpy as np
print("University No: SJC22MCA-2021 \nName: Christin Benny \nBatch: S3 MCA
\n_______________________________________\n")
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.")
```

OUTPUT

```
University No: SJC22MCA-2021
Name: Christin Benny
Batch: S3 MCA
--------------------------------------

The matrix is skew-symmetric (antisymmetric).

Process finished with exit code 0
```

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.

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

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

CODE:

```
import numpy as np
```

```
print("University No: SJC22MCA-2021 \nName: Christin Benny \nBatch: S3 MCA
\n_______________________________________\n")

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.")
```

OUTPUT

```
University No: SJC22MCA-2021
Name: Christin Benny
Batch: S3 MCA
--------------------------------------

Solution X:
[ 2.   0.8 -0.6]

Process finished with exit code 0
```

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 $A = U\Sigma V^T$

<u>CODE:</u>

```
import numpy as np


print("University No: SJC22MCA-2021 \nName: Christin Benny \nBatch: S3 MCA
\n________________________________________\n")


A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])
U, S, VT = np.linalg.svd(A)
reconstructed_A = np.dot(U, np.dot(np.diag(S), VT))


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

```
/home/sjcet/anaconda3/envs/untitled/bin/python /home/sjcet/christinmca22/Data Science/Cycle 2/16.py
University No: SJC22MCA-2021
Name: Christin Benny
Batch: S3 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.]]

Process finished with exit code 0
```