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

Code

```
print("Name: Aswathy Chandran")
print("Reg No: SJC22MCA-2016")
print("Batch: 22-24")
print()
import numpy as np
shape = (3, 3, 3)
d_type = np.float32
my_3d_array = np.random.rand(*shape).astype(d_type)

print(my_3d_array)
```

Output

```
/home/sjcet/PycharmProjects/pythonProject/venv/bin/python /home/sjcet/PycharmProjects/
Name: Aswathy Chandran
Reg No: SJC22MCA-2016
Batch: 22-24

[[[0.11443481 0.90174955 0.14218792]
  [0.75207806 0.7446035  0.42370817]
  [0.19990502 0.9838276  0.94932806]]

 [[0.5363345  0.23515585 0.9659123 ]
  [0.68101084 0.47537005 0.48414797]
  [0.3966246  0.5653436  0.6498678 ]]

 [[0.06672744 0.88656497 0.66475195]
  [0.99261963 0.16235617 0.41194278]
  [0.5830807  0.09784301 0.80568075]]]

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("Name: Aswathy Chandran")
print("Reg No: SJC22MCA-2016")
print("Batch: 22-24")
print()

```
import numpy as np

complex_array = np.array([[7 + 2j, 1 + 4j, 5 + 3j],
                           [9 + 6j, 8 + 10j, 11 + 12j]], dtype=complex
print("2D Array:")
print(complex_array)

num_rows, num_columns = complex_array.shape

dimensions = complex_array.ndim

print("\nNumber of rows:", num_rows)
print("   Number of columns:", num_columns)
print("\nDimensions of the array:", dimensions)

reshaped_array = complex_array.reshape(3, 2)

print("\nReshaped Array (3x2):")
print(reshaped_array)
```

Output

```
/home/sjcet/PycharmProjects/pythonProject/venv/bin/python /home/sjcet/PycharmProjects/
Name: Aswathy Chandran
Reg No: SJC22MCA-2016
Batch: 22-24

2D Array:
[[ 7. +2.j  1. +4.j  5. +3.j]
 [ 9. +6.j  8.+10.j 11.+12.j]]

Number of rows: 2
   Number of columns: 3

Dimensions of the array: 2

Reshaped Array (3x2):
[[ 7. +2.j  1. +4.j]
 [ 5. +3.j  9. +6.j]
 [ 8.+10.j 11.+12.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

```
print("Name: Aswathy Chandran")
print("Reg No: SJC22MCA-2016")
print("Batch: 22-24")
print()
import numpy as np

uninitialized_array = np.empty((3, 3))
```

```
ones_array = np.ones((3, 4))
zeros_array = np.zeros((6, 6))
print("Uninitialized array:")
print(uninitialized_array)

print("\nArray with all ones:")
print(ones_array)

print("\nArray with all zeros:")
print(zeros_array)
```

Output

```
/home/sjcet/PycharmProjects/pythonProject/venv/bin/python /home/sjcet/PycharmProjects/
Name: Aswathy Chandran
Reg No: SJC22MCA-2016
Batch: 22-24

Uninitialized array:
[[6.91555983e-310 6.91555983e-310 4.84095796e-317]
 [6.91555020e-310 4.84097377e-317 6.91555020e-310]
 [4.84098958e-317 4.68814544e-317 3.95252517e-322]]

Array with all ones:
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]

Array with all zeros:
[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 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

Code

```
print("Name: Aswathy Chandran")
print("Reg No: SJC22MCA-2016")
print("Batch: 22-24")
print()
import numpy as np

my_array = np.arange(10)

print("First 4 elements:")
print(my_array[:4])

print("\nLast 6 elements:")
print(my_array[-6:])

print("\nElements from index 2 to 7:")
print(my_array[2:8])
```

Output

```
/home/sjcet/PycharmProjects/pythonProject/venv/bin/python /home/sjcet/PycharmProjects
Name: Aswathy Chandran
Reg No: SJC22MCA-2016
Batch: 22-24

First 4 elements:
[0 1 2 3]

Last 6 elements:
[4 5 6 7 8 9]

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

```
print("Name: Aswathy Chandran")
print("Reg No: SJC22MCA-2016")
print("Batch: 22-24")
print()
import numpy as np
even_numbers = np.arange(2, 32, 2)
print(even_numbers)
print()
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)
print()
last_3_elements = even_numbers[-3:]

print("b. Last 3 elements of the array using negative index:", last_3_elements)
print()
alternate_elements = even_numbers[::2]

print("c. Alternate elements of the array:", alternate_elements)
print()
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/PycharmProjects/pythonProject/venv/bin/python /home/sjcet/PycharmProjects/
Name: Aswathy Chandran
Reg No: SJC22MCA-2016
Batch: 22-24

[ 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("Name: Aswathy Chandran")
print("Reg No: SJC22MCA-2016")
print("Batch: 22-24")
print()
matrix1 = np.array([[51, 82, 37], [14, 20, 62], [7, 10, 77]])
```

```
matrix2 = np.array([[5, 43, 22], [9, 12, 0], [32, 52, 71]])

matrix_sum = matrix1 + matrix2
print("Sum of the two matrices:")
print(matrix_sum)

matrix_diff = matrix1 - matrix2
print("\nDifference of the two matrices:")
print(matrix_diff)

matrix_product = matrix1 * matrix2
print("\nElement-wise product of the two matrices:")
print(matrix_product)

with np.errstate(divide='ignore', invalid='ignore'):
    matrix_division = np.true_divide(matrix1, matrix2)
    matrix_division[~np.isfinite(matrix_division)] = np.nan
print("\nElement-wise division of the two matrices:")
print(matrix_division)

matrix_mult = np.dot(matrix1, matrix2)
print("\nMatrix multiplication of the two matrices:")
print(matrix_mult)


matrix1_transpose = np.transpose(matrix1)
print("\nTranspose of matrix1:")
print(matrix1_transpose)

diagonal_sum = np.trace(matrix1)
print("\nSum of diagonal elements of matrix1:")
print(diagonal_sum)
```

Output

```
/home/sjcet/PycharmProjects/pythonProject/venv/bin/python /home/sjcet/PycharmProjects/
Name: Aswathy Chandran
Reg No: SJC22MCA-2016
Batch: 22-24

Sum of the two matrices:
[[ 56 125  59]
 [ 23  32  62]
 [ 39  62 148]]

Difference of the two matrices:
[[ 46  39  15]
 [  5   8  62]
 [-25 -42   6]]

Element-wise product of the two matrices:
[[ 255 3526  814]
 [ 126  240    0]
 [ 224  520 5467]]

Element-wise division of the two matrices:
[[10.2         1.90697674  1.68181818]
 [ 1.55555556  1.66666667         nan]
 [ 0.21875     0.19230769  1.08450704]]

Matrix multiplication of the two matrices:
[[2177 5101 3749]
 [2234 4066 4710]
 [2589 4425 5621]]

Transpose of matrix1:
```

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("Name: Aswathy Chandran")
print("Reg No: SJC22MCA-2016")
print("Batch: 22-24")
print()

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("\nb. Subtraction Result:")
print(subtraction_result)

multiplication_result = matrix1 * matrix2
print("\nc. Multiplication Result:")
print(multiplication_result)

epsilon = 1e-15
division_result = np.divide(matrix1, matrix2 + epsilon)
print("\nd. Division Result:")
print(division_result)

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

```
transpose_result = np.transpose(matrix1)
print("\nf. Transpose of the Matrix:")
print(transpose_result)

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

Output

```
/home/sjcet/PycharmProjects/pythonProject/venv/bin/python /home/sjcet/PycharmProjects/
Name: Aswathy Chandran
Reg No: SJC22MCA-2016
Batch: 22-24

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]]
```

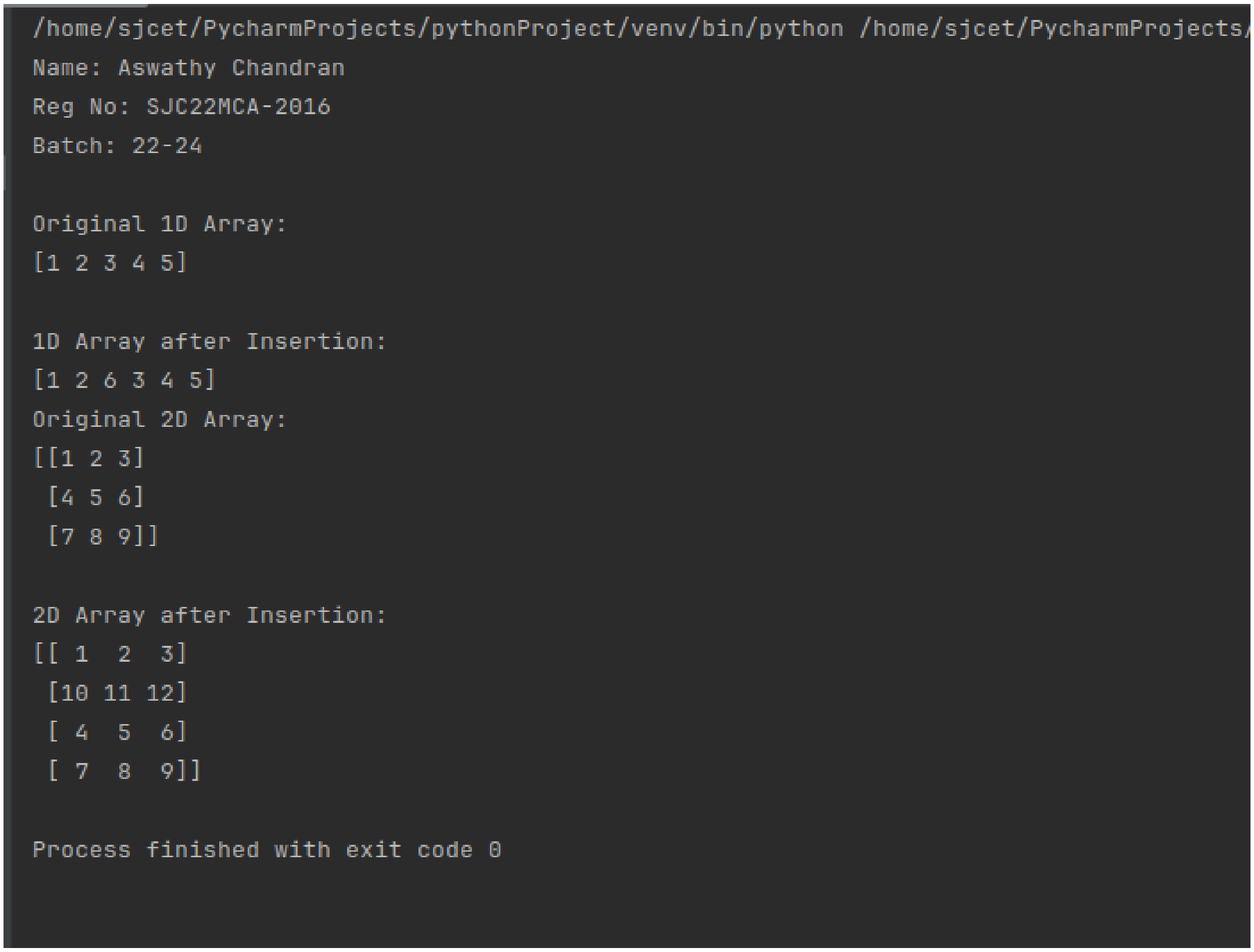
```
f. Transpose of the Matrix:
[[1 4 7]
 [2 5 8]
 [3 6 9]]

g. Sum of Diagonal Elements:
15

Process finished with exit code 0
```

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

Code

```
print("Name: Aswathy Chandran")
print("Reg No: SJC22MCA-2016")
print("Batch: 22-24")
print()
import numpy as np
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)

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/PycharmProjects/pythonProject/venv/bin/python /home/sjcet/PycharmProjects/
Name: Aswathy Chandran
Reg No: SJC22MCA-2016
Batch: 22-24

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)

Code

```
import numpy as np
print("Name: Aswathy Chandran")
print("Reg No: SJC22MCA-2016")
print("Batch: 22-24")
print()
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/PycharmProjects/pythonProject/venv/bin/python /home/sjcet/PycharmProjects/p
Name: Aswathy Chandran
Reg No: SJC22MCA-2016
Batch: 22-24

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]

Process finished with exit code 0
```

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("Name: Aswathy Chandran")
print("Reg No: SJC22MCA-2016")
print("Batch: 22-24")
print()
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/PycharmProjects/pythonProject/venv/bin/python /home/sjcet/PycharmProjects/
Name: Aswathy Chandran
Reg No: SJC22MCA-2016
Batch: 22-24

Random Square Matrix:
[[ 5  7  5]
 [ 3  1  3]
 [10  4  6]]

Inverse Matrix:
[[-9.3750000e-02 -3.4375000e-01  2.5000000e-01]
 [ 1.8750000e-01 -3.1250000e-01  4.4408921e-17]
 [ 3.1250000e-02  7.8125000e-01 -2.5000000e-01]]

Rank of the Matrix: 3

Determinant of the Matrix: 64.00000000000003

Matrix as a 1D Array:
[ 5  7  5  3  1  3 10  4  6]

Eigenvalues:
[15.06902003+0.j         -1.53451001+1.37564648j -1.53451001-1.37564648j]

Eigenvectors:
[[ 0.57833349+0.j         -0.52882824+0.23037485j -0.52882824-0.23037485j]
 [ 0.28621959+0.j         -0.08529529-0.31898106j -0.08529529+0.31898106j]
 [ 0.76394288+0.j          0.74715722+0.j          0.74715722-0.j        ]]

Process finished with exit code 0
```

11.a

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("Name: Aswathy Chandran")
print("Reg No: SJC22MCA-2016")
print("Batch: 22-24")
print()

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/PycharmProjects/pythonProject/venv/bin/python /home/sjcet/PycharmProjects/
Name: Aswathy Chandran
Reg No: SJC22MCA-2016
Batch: 22-24

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]]
```

```
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]]

Process finished with exit code 0
```

11b

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

Code

```
import numpy as np
print("Name: Aswathy Chandran")
print("Reg No: SJC22MCA-2016")
print("Batch: 22-24")
print()
X = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])
```

```
Y = np.array([[10, 20, 30],
              [40, 50, 60],
              [70, 80, 90]])

result = np.power(X, 2) + 2 * Y

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

print("\nMatrix Y:")
print(Y)

print("\nResult of X^2 + 2Y:")
print(result)
```

<u>**Output**</u>

```
/home/sjcet/PycharmProjects/pythonProject/venv/bin/python /home/sjcet/PycharmProjects
Name: Aswathy Chandran
Reg No: SJC22MCA-2016
Batch: 22-24

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

Matrix Y:
[[10 20 30]
 [40 50 60]
 [70 80 90]]

Result of X^2 + 2Y:
[[ 21  44  69]
 [ 96 125 156]
 [189 224 261]]

Process finished with exit code 0
```

**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

```
import numpy as np
print("Name: Aswathy Chandran")
print("Reg No: SJC22MCA-2016")
print("Batch: 22-24")
print()
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/PycharmProjects/pythonProject/venv/bin/python /home/sjcet/PycharmProjects/
Name: Aswathy Chandran
Reg No: SJC22MCA-2016
Batch: 22-24

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]]

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("Name: Aswathy Chandran")
print("Reg No: SJC22MCA-2016")
print("Batch: 22-24")
print()
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/PycharmProjects/pythonProject/venv/bin/python /home/sjcet/PycharmProjects/
Name: Aswathy Chandran
Reg No: SJC22MCA-2016
Batch: 22-24

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("Name: Aswathy Chandran")
print("Reg No: SJC22MCA-2016")
print("Batch: 22-24")
print()
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

```
/home/sjcet/PycharmProjects/pythonProject/venv/bin/python /home/sjcet/PycharmProjects/
Name: Aswathy Chandran
Reg No: SJC22MCA-2016
Batch: 22-24

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

Code

```
import numpy as np
print("Name: Aswathy Chandran")
print("Reg No: SJC22MCA-2016")
print("Batch: 22-24")
print()
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.")
```

<u>Output</u>

```
/home/sjcet/PycharmProjects/pythonProject/venv/bin/python /home/sjcet/PycharmProjects/
Name: Aswathy Chandran
Reg No: SJC22MCA-2016
Batch: 22-24

Solution X:
[ 1. -1.  2.]

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 $A = U\Sigma V^T$

Code

```
import numpy as np
print("Name: Aswathy Chandran")
print("Reg No: SJC22MCA-2016")
print("Batch: 22-24")
print()
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)
```

Output

```
/home/sjcet/PycharmProjects/pythonProject/venv/bin/python /home/sjcet/PycharmProjects/
Name: Aswathy Chandran
Reg No: SJC22MCA-2016
Batch: 22-24

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

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

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

Process finished with exit code 0
```