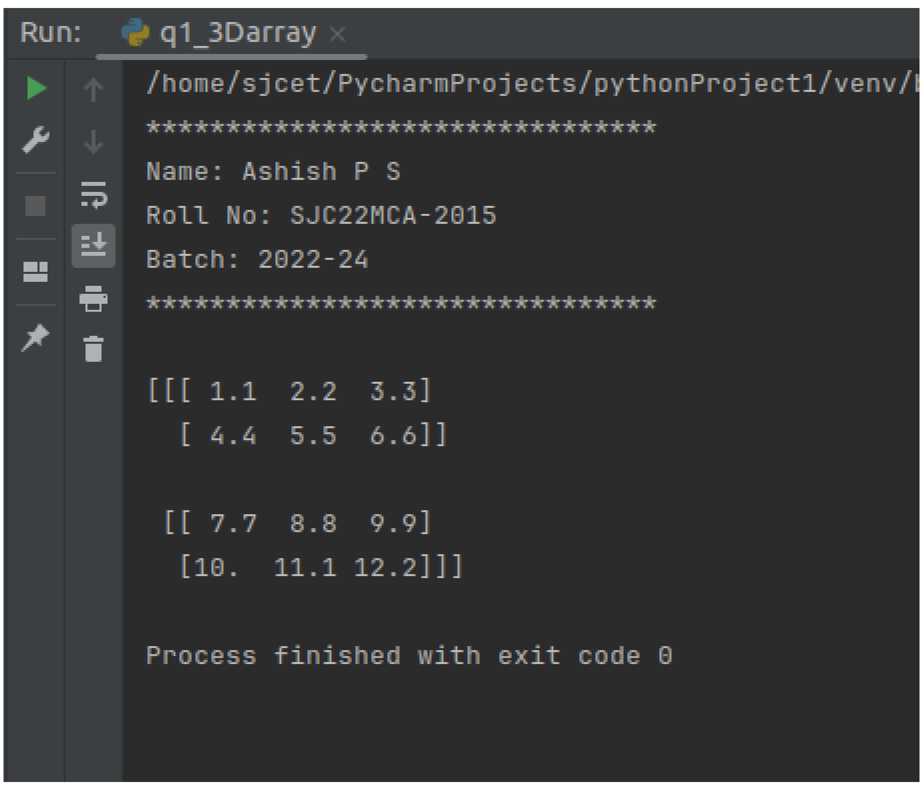
**1. Create a three dimensional array specifying float data type and print it.**

```
import numpy as np

print("********************************")
print("Name: Ashish P S")
print("Roll No: SJC22MCA-2015")
print("Batch: 2022-24")
print("********************************")
print()
array_3d = np.array([[[1.1, 2.2, 3.3], [4.4, 5.5, 6.6]],
                     [[7.7, 8.8, 9.9], [10.0, 11.1, 12.2]]], dtype=float)

print(array_3d)
```




**2. Create a 2 dimensional array (2X3) with elements belonging to complex data type**

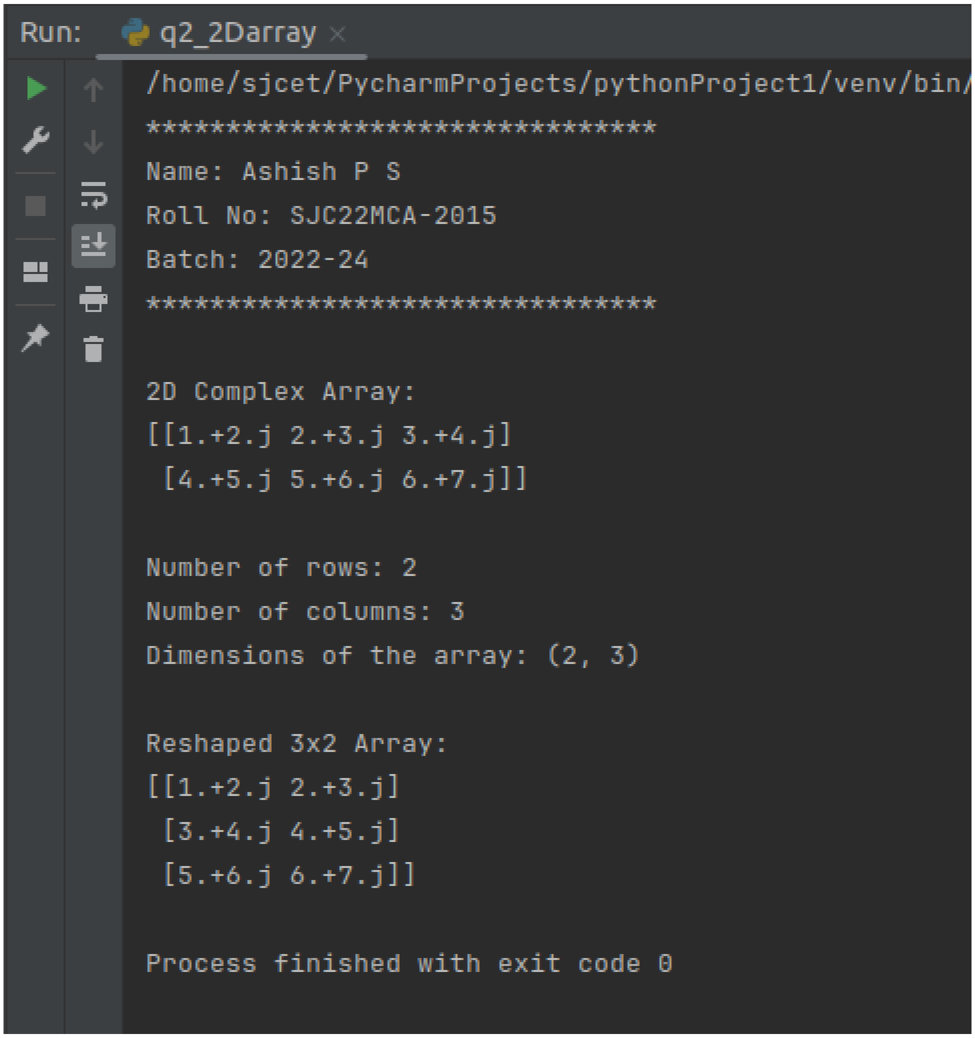
**and print it. Also display**

**a. the no: of rows and columns**

**b. dimension of an array**

**c. reshape the same array to 3X2**

```
import numpy as np

print("********************************")
print("Name: Ashish P S")
print("Roll No: SJC22MCA-2015")
print("Batch: 2022-24")
print("********************************")
print()
# Create a 2D array (2x3) with complex data type
array_2d_complex = np.array([[1 + 2j, 2 + 3j, 3 + 4j],
                             [4 + 5j, 5 + 6j, 6 + 7j]], dtype=complex)

# Print the 2D complex array
print("2D Complex Array:")
print(array_2d_complex)

# a. Get the number of rows and columns
num_rows, num_cols = array_2d_complex.shape

# b. Get the dimensions of the array
dimensions = array_2d_complex.shape

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

# c. Reshape the array to 3x2
array_reshaped = array_2d_complex.reshape(3, 2)

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




**3. Familiarize with the functions to create**

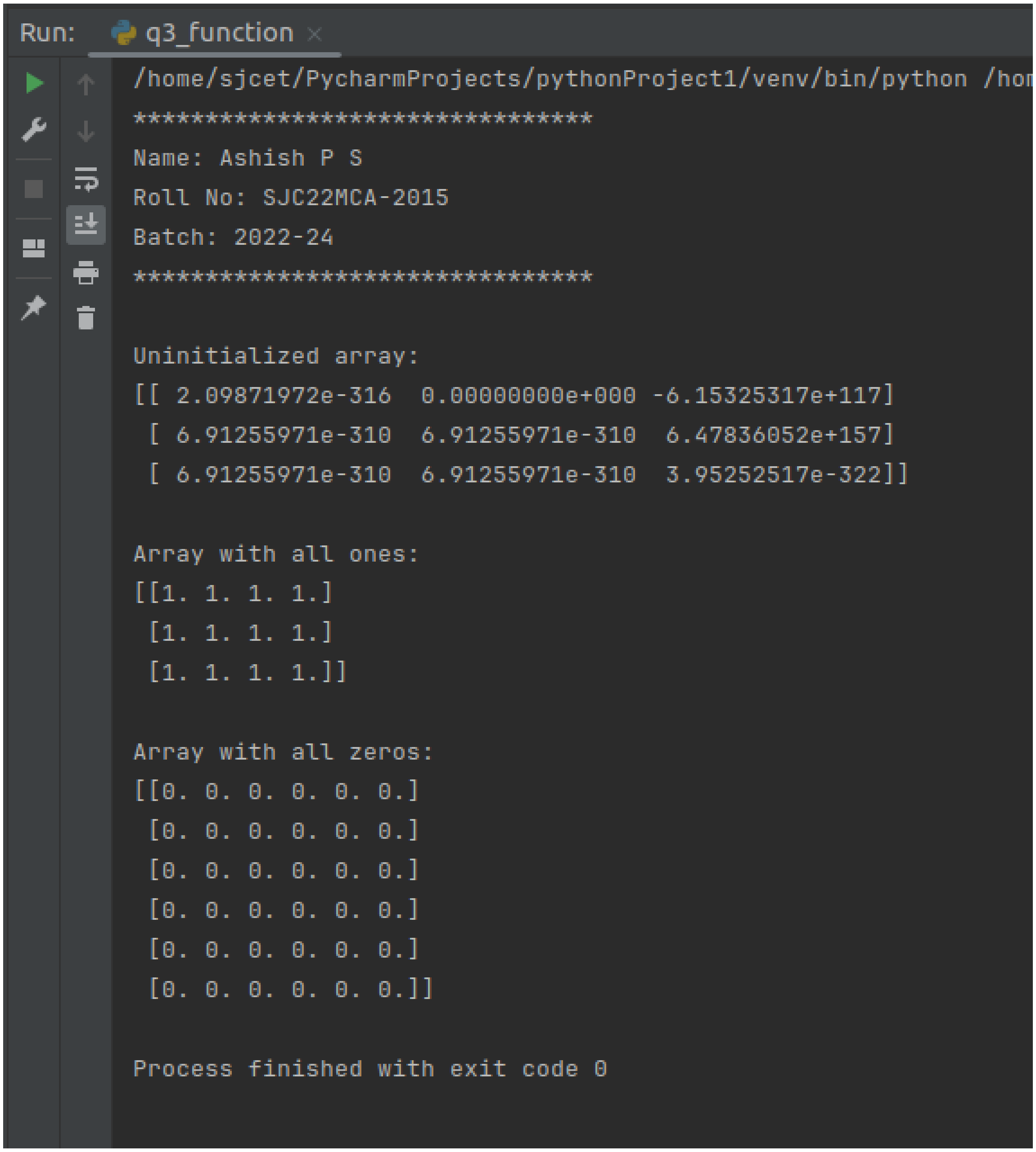
**a) an uninitialized array**

**b) array with all elements as 1,**

**c) all elements as 0**

```
import numpy as np

print("********************************")
print("Name: Ashish P S")
print("Roll No: SJC22MCA-2015")
```

```
print("Batch: 2022-24")
print("*******************************")
print()
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)
```



```
Run:    q3_function
/home/sjcet/PycharmProjects/pythonProject1/venv/bin/python /hom
********************************
Name: Ashish P S
Roll No: SJC22MCA-2015
Batch: 2022-24
********************************

Uninitialized array:
[[ 2.09871972e-316  0.00000000e+000 -6.15325317e+117]
 [ 6.91255971e-310  6.91255971e-310  6.47836052e+157]
 [ 6.91255971e-310  6.91255971e-310  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**

import numpy as np

print("********************************")

```python
print("Name: Ashish P S")
print("Roll No: SJC22MCA-2015")
print("Batch: 2022-24")
print("********************************")
print()

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])
```

```
Run:    q4_1Darray
/home/sjcet/PycharmProjects/pythonProject1/venv/bin/
********************************
Name: Ashish P S
Roll No: SJC22MCA-2015
Batch: 2022-24
********************************

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**

```
import numpy as np

print("********************************")
print("Name: Ashish P S")
print("Roll No: SJC22MCA-2015")
print("Batch: 2022-24")
print("********************************")
print()

# Create a 1D array with the first 15 even numbers using arange
even_numbers = np.arange(2, 31, 2)

# a. Elements from index 2 to 8 with step 2
subset_a = even_numbers[2:9:2]  # Using array slicing
print("a. Elements from index 2 to 8 with step 2:")
print(subset_a)

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

# c. Alternate elements of the array
alternate_elements = even_numbers[::2]
print("\nc. Alternate elements of the array:")
print(alternate_elements)

# d. Display the last 3 alternate elements
last_3_alternate_elements = alternate_elements[-3:]
print("\nd. Last 3 alternate elements:")
print(last_3_alternate_elements)
```

```
Run:   q5_1Darray_arange ×
/home/sjcet/PycharmProjects/pythonProject1/venv/bin/pyth
********************************
Name: Ashish P S
Roll No: SJC22MCA-2015
Batch: 2022-24
********************************

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:
[22 26 30]

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)**

```
import numpy as np

print("********************************")
print("Name: Ashish P S")
print("Roll No: SJC22MCA-2015")
print("Batch: 2022-24")
```

```
print("********************************")
print()

# Create a 2D array with 4 rows and 4 columns
array_2d = np.array([
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12],
    [13, 14, 15, 16]
])
print("4x4 2D array is :")
print("\n[1, 2, 3, 4]\n[5, 6, 7, 8]\n[9, 10, 11, 12]\n[13, 14, 15, 16]")


print("\n\na. All elements excluding the first row:")
print(array_2d[1:])


print("\nb. All elements excluding the last column:")
print(array_2d[:, :-1])


print("\nc. Elements of the 1st and 2nd column in the 2nd and 3rd row:")
print(array_2d[1:3, 0:2])


print("\nd. Elements of the 2nd and 3rd column:")
print(array_2d[:, 1:3])


print("\ne. 2nd and 3rd element of the 1st row:")
print(array_2d[0, 1:3])


print("\nf. Elements from indices 4 to 10 in descending order:")
print(array_2d.flatten()[10:3:-1])
```

Run: q6_4x4_2Darray

```
/home/sjcet/PycharmProjects/pythonProject1/venv/bin/python /home/sjcet
********************************
Name: Ashish P S
Roll No: SJC22MCA-2015
Batch: 2022-24
********************************

4x4 2D array is :

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


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

b. All elements excluding the last column:
[[ 1  2  3]
 [ 5  6  7]
 [ 9 10 11]
 [13 14 15]]

c. Elements of the 1st and 2nd column in the 2nd and 3rd row:
[[ 5  6]
 [ 9 10]]
```

```
d. Elements of the 2nd and 3rd column:
[[ 2  3]
 [ 6  7]
 [10 11]
 [14 15]]

e. 2nd and 3rd element of the 1st row:
[2 3]

f. Elements from indices 4 to 10 in descending order:
[11 10  9  8  7  6  5]

Process finished with exit code 0
```

**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("********************************")
print("Name: Ashish P S")
print("Roll No: SJC22MCA-2015")
print("Batch: 2022-24")
print("********************************")
print()

# Create two 2D arrays
array1 = np.array([[1, 2], [3, 4]])
array2 = np.array([[5, 6], [7, 8]])
```

```
addition_result = array1 + array2
print("a. Addition of the two matrices:")
print(addition_result)


subtraction_result = array1 - array2
print("\nb. Subtraction of the two matrices:")
print(subtraction_result)


elementwise_multiply_result = array1 * array2
print("\nc. Elementwise multiplication of the two matrices:")
print(elementwise_multiply_result)


elementwise_divide_result = array1 / array2
print("\nd. Elementwise division of the two matrices:")
print(elementwise_divide_result)


matrix_multiply_result = np.dot(array1, array2)
print("\ne. Matrix multiplication of the two matrices:")
print(matrix_multiply_result)

transpose_result = array1.T
print("\nf. Transpose of the first matrix:")
print(transpose_result)


diagonal_sum = np.trace(array1)
print("\ng. Sum of diagonal elements of the first matrix:")
print(diagonal_sum)
```

Run: q7_2Darray_object

```
/home/sjcet/PycharmProjects/pythonProject1/venv/bin/pytho
********************************
Name: Ashish P S
Roll No: SJC22MCA-2015
Batch: 2022-24
********************************

a. Addition of the two matrices:
[[ 6  8]
 [10 12]]

b. Subtraction of the two matrices:
[[-4 -4]
 [-4 -4]]

c. Elementwise multiplication of the two matrices:
[[ 5 12]
 [21 32]]

d. Elementwise division of the two matrices:
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
```

```
e. Matrix multiplication of the two matrices:
[[19 22]
 [43 50]]

f. Transpose of the first matrix:
[[1 3]
 [2 4]]

g. Sum of diagonal elements of the first matrix:
5

Process finished with exit code 0
```

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

```
import numpy as np

print("********************************")
print("Name: Ashish P S")
print("Roll No: SJC22MCA-2015")
print("Batch: 2022-24")
print("********************************")
print()

# Create a 1D array
arr_1d = np.array([1, 2, 3, 4, 5])

# Insert an element (e.g., 10) at a specific index (e.g., index 2)
arr_1d_inserted = np.insert(arr_1d, 2, 10)

print("Original 1D Array:")
print(arr_1d)

print("\n1D Array after Insertion:")
print(arr_1d_inserted)


# Create a 2D array
arr_2d = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

# Insert a row at a specific index (e.g., index 1)
new_row = np.array([10, 11, 12])
arr_2d_inserted = np.insert(arr_2d, 1, new_row, axis=0)

print("Original 2D Array:")
print(arr_2d)

print("\n2D Array after Row Insertion:")
print(arr_2d_inserted)
```

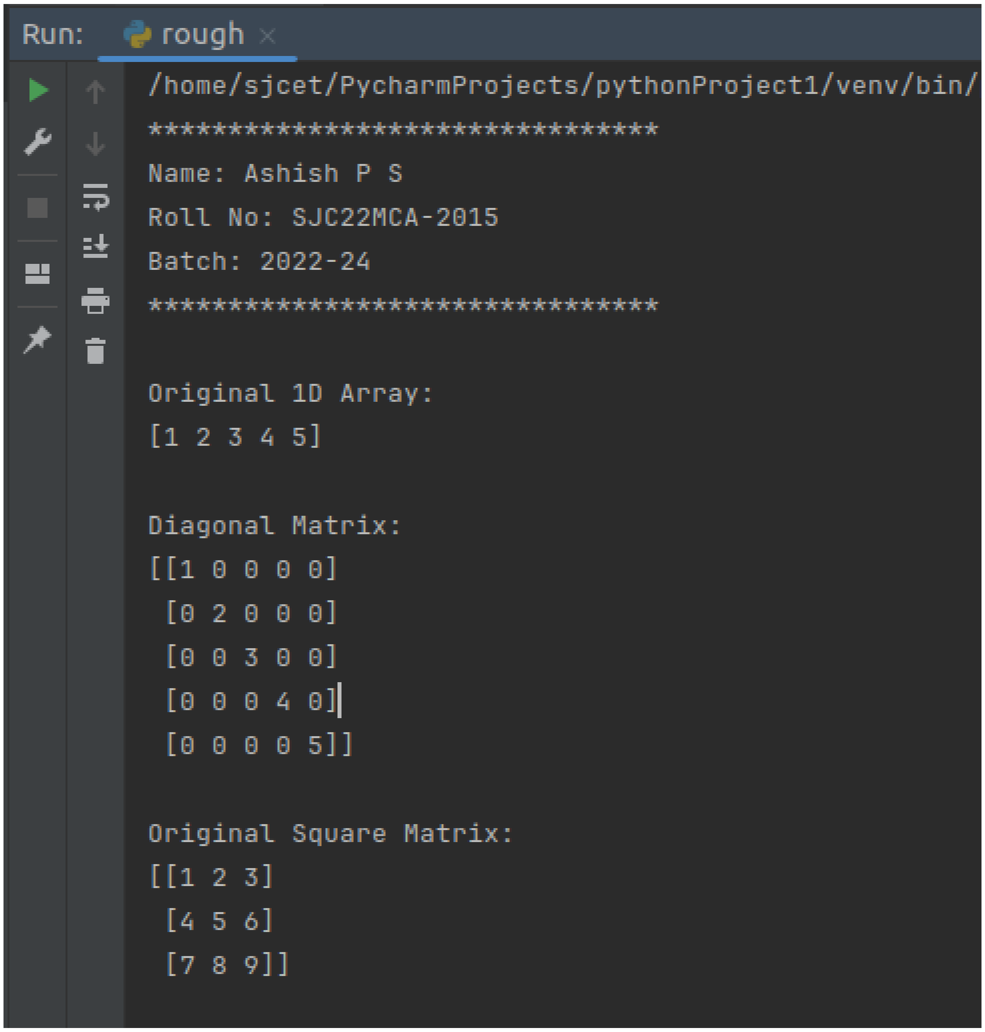
Run: q8_insert_func

```
/home/sjcet/PycharmProjects/pythonProject1/venv/bin/python
********************************
Name: Ashish P S
Roll No: SJC22MCA-2015
Batch: 2022-24
********************************

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

1D Array after Insertion:
[ 1  2 10  3  4  5]
Original 2D Array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

2D Array after Row 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)**

```
import numpy as np
print("********************************")
print("Name: Ashish P S")
print("Roll No: SJC22MCA-2015")
print("Batch: 2022-24")
print("********************************")
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)




```
/home/sjcet/PycharmProjects/pythonProject1/venv/bin/
********************************
Name: Ashish P S
Roll No: SJC22MCA-2015
Batch: 2022-24
********************************

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**

```
import numpy as np
print("********************************")
print("Name: Ashish P S")
print("Roll No: SJC22MCA-2015")
print("Batch: 2022-24")
print("********************************")
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)
```

Run: q10_sqmatrix_randint

```
/home/sjcet/PycharmProjects/pythonProject1/venv/bin/pytho
*********************************
Name: Ashish P S
Roll No: SJC22MCA-2015
Batch: 2022-24
*********************************

Random Square Matrix:
[[ 3 10  3]
 [ 2 10  8]
 [ 2  5  9]]

Inverse Matrix:
[[ 0.5  -0.75  0.5 ]
 [-0.02  0.21 -0.18]
 [-0.1   0.05  0.1 ]]

Rank of the Matrix: 3

Determinant of the Matrix: 100.00000000000004
```

```
Matrix as a 1D Array:
[ 3 10  3  2 10  8  2  5  9]

Eigenvalues:
[17.75515847+0.j         2.12242076+1.0618362j  2.12242076-1.0618362j]

Eigenvectors:
[[ 0.55223953+0.j          0.94767823+0.j          0.94767823-0.j        ]
 [ 0.66334359+0.j         -0.00905943+0.1433295j  -0.00905943-0.1433295j ]
 [ 0.50498195+0.j         -0.24702282-0.14233864j -0.24702282+0.14233864j]]

Process finished with exit code 0
```

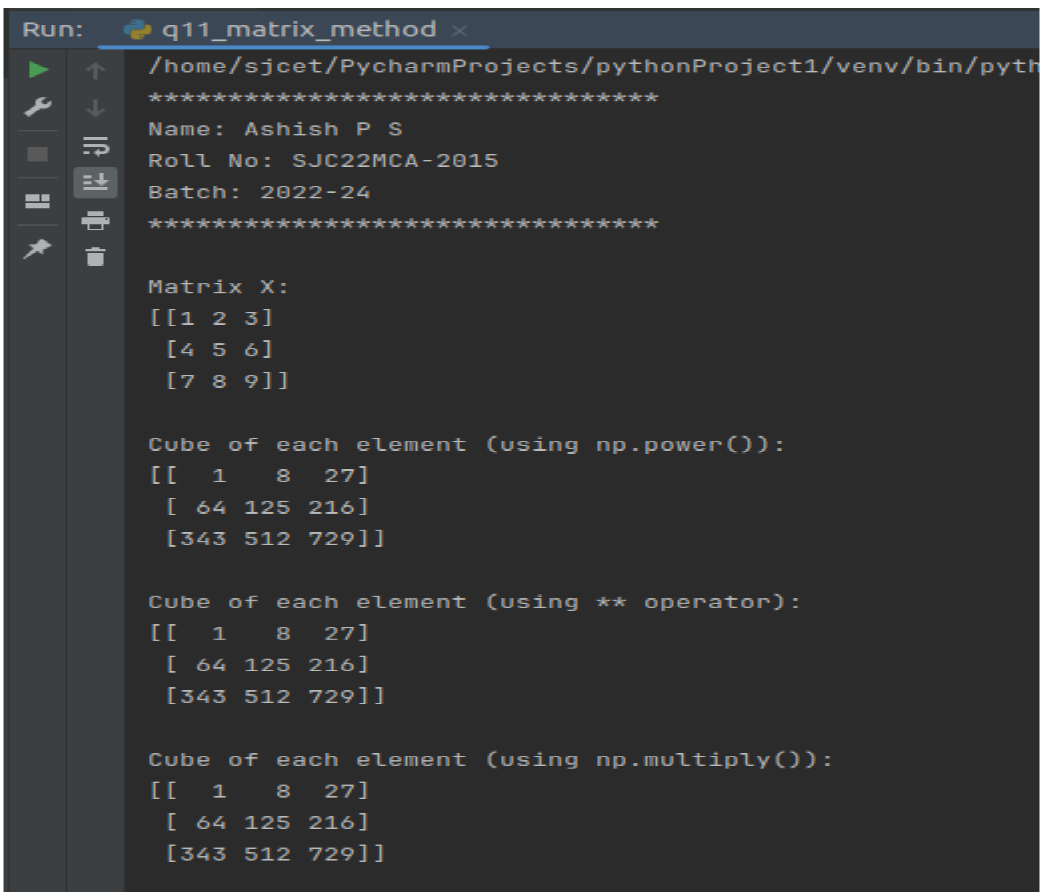
**11.. Create a matrix X with suitable rows and columns**

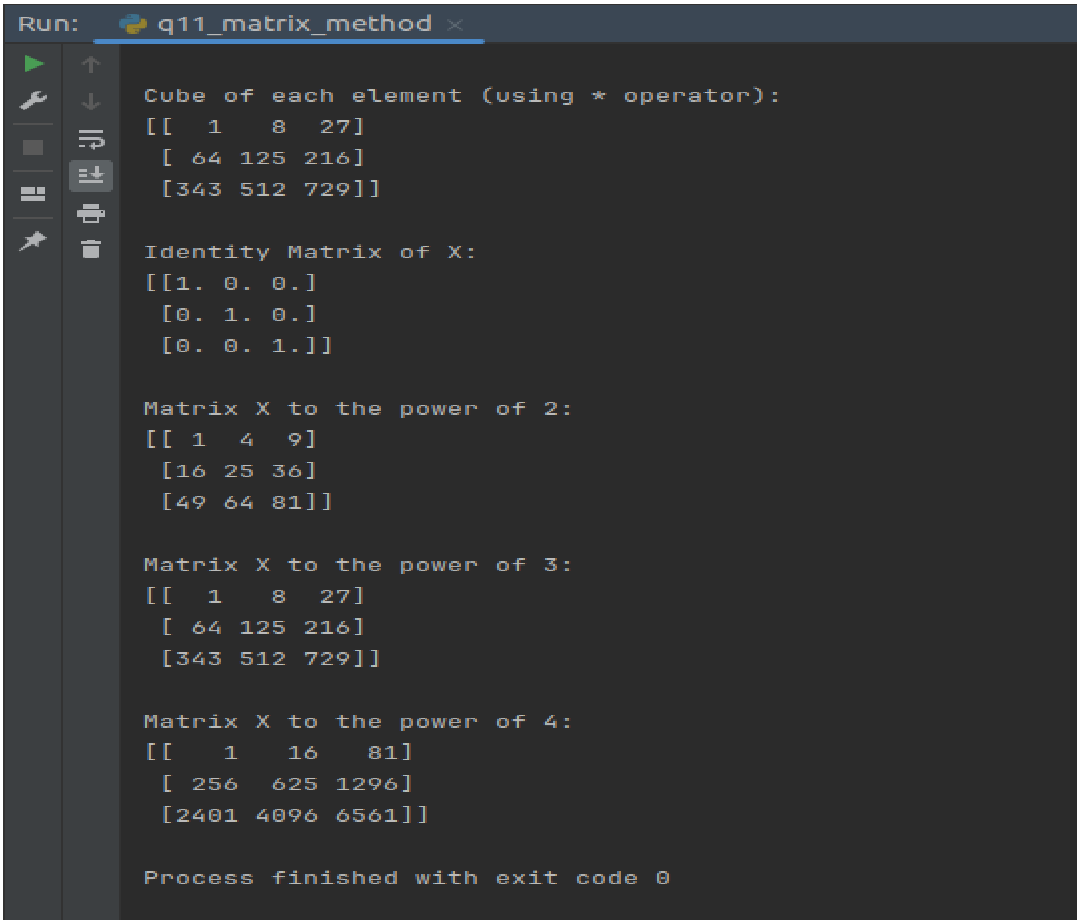
**i) Display the cube of each element of the matrix using different methods(use multiply(), *, power(),**)**

**ii) Display identity matrix of the given square matrix.**
**iii) Display each element of the matrix to different powers.**

```
import numpy as np

print("********************************")
print("Name: Ashish P S")
print("Roll No: SJC22MCA-2015")
print("Batch: 2022-24")
print("********************************")
print()

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.power() to calculate the cube
cubed_matrix1 = np.power(X, 3)

# Using the ** operator to calculate the cube
cubed_matrix2 = X ** 3

# Using np.multiply() to calculate the cube
cubed_matrix3 = np.multiply(X, np.multiply(X, X))

# Using the * operator to calculate the cube
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)

# ii) Display the 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
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])
```



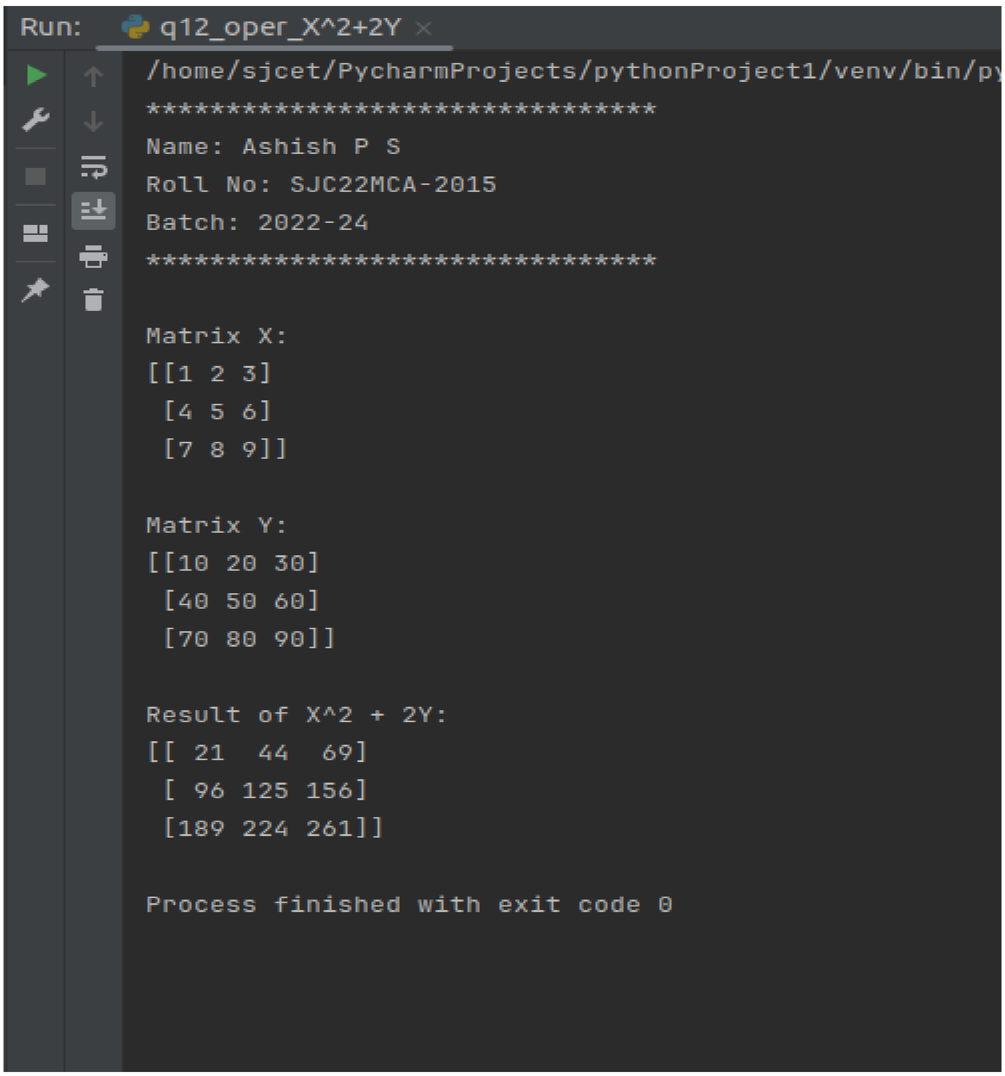


```
Run:    q11_matrix_method

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
```

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

```
import numpy as np

print("*********************************")
print("Name: Ashish P S")
print("Roll No: SJC22MCA-2015")
print("Batch: 2022-24")
print("*********************************")
print()

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

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

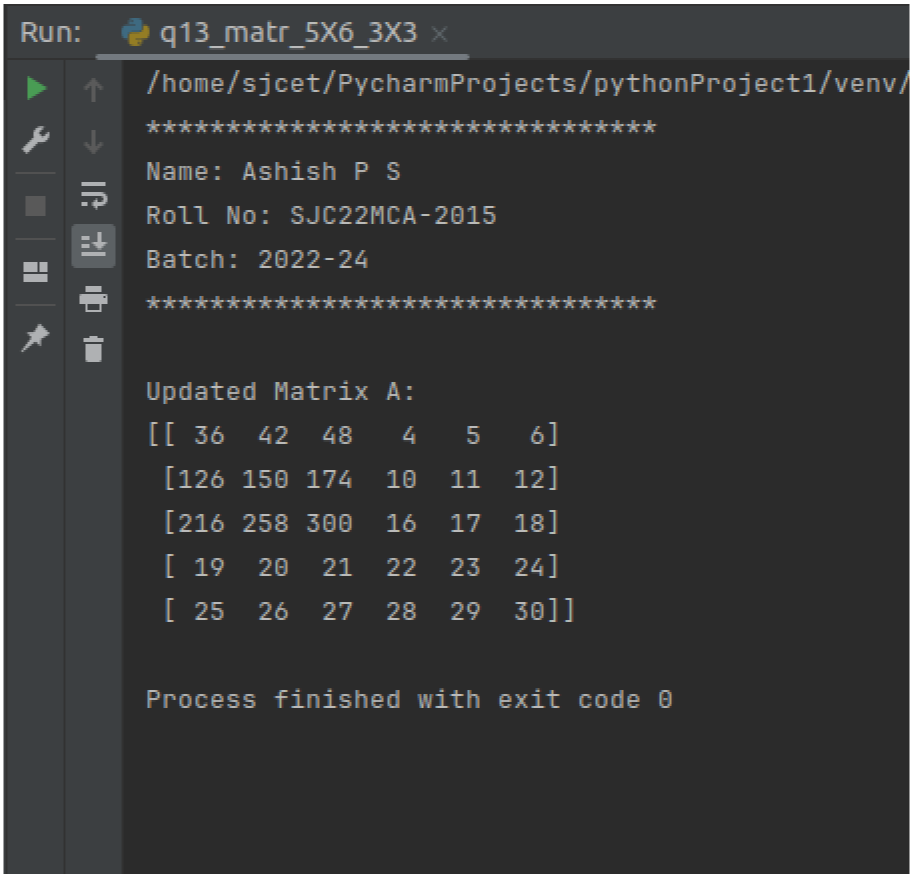
```
              [70, 80, 90]])

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)
```



**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**

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

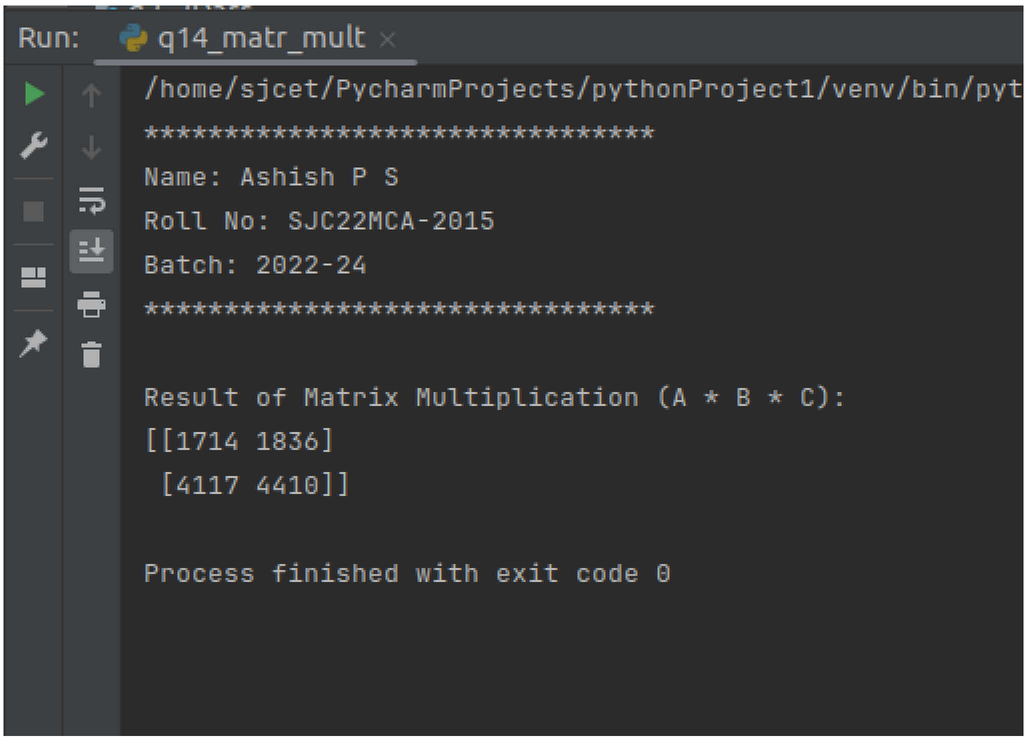
```
import numpy as np

print("********************************")
print("Name: Ashish P S")
print("Roll No: SJC22MCA-2015")
print("Batch: 2022-24")
print("********************************")
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

# Display the updated matrix A
print("Updated Matrix A:")
print(A)
```




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

```
import numpy as np

print("********************************")
print("Name: Ashish P S")
print("Roll No: SJC22MCA-2015")
print("Batch: 2022-24")
print("********************************")
print()

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

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

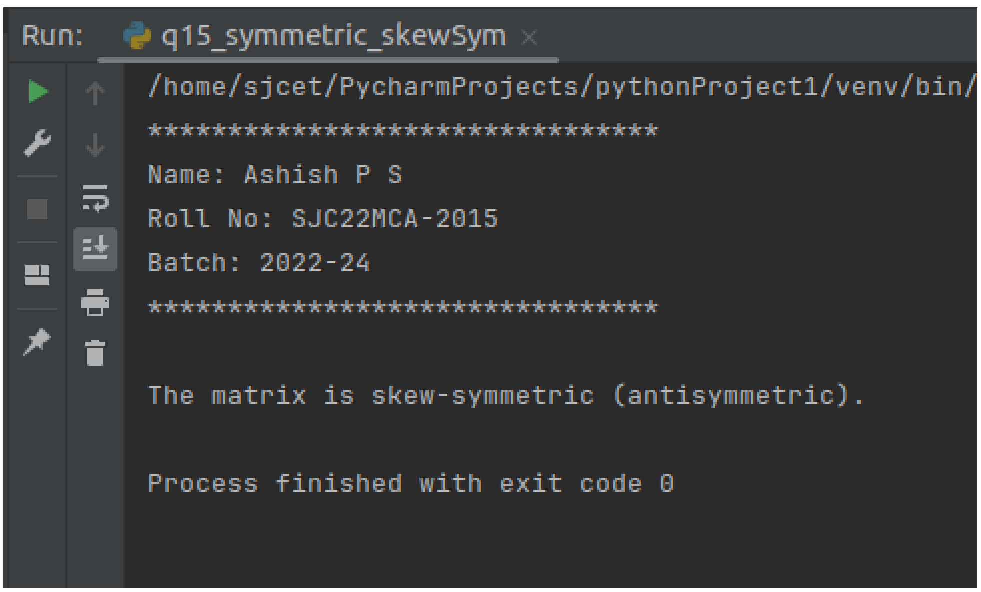
```
C = np.array([[13, 14],
              [15, 16]])

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

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




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

```
import numpy as np

print("********************************")
print("Name: Ashish P S")
print("Roll No: SJC22MCA-2015")
print("Batch: 2022-24")
print("********************************")
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.")
```



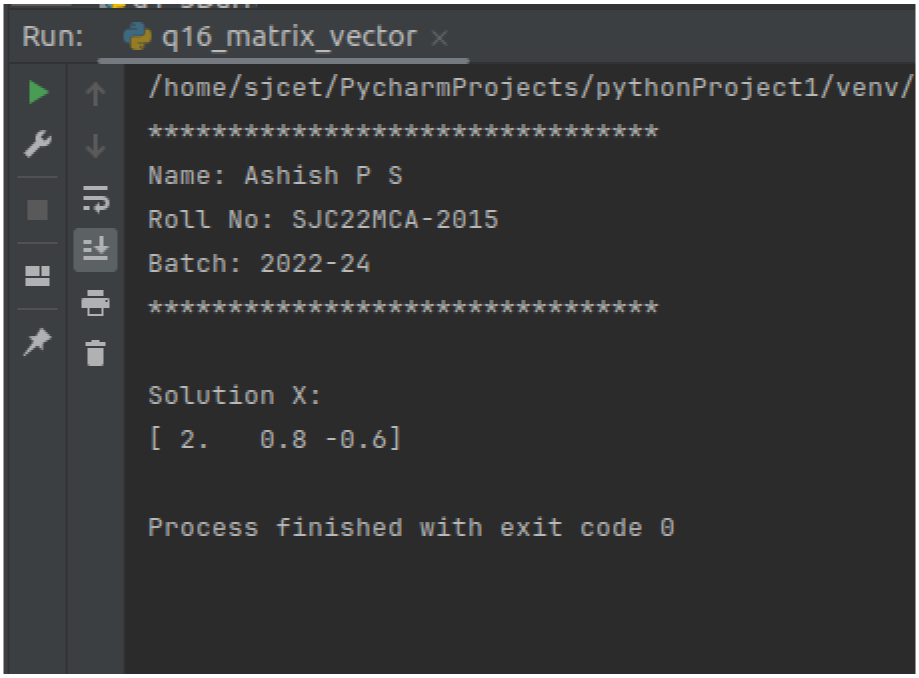

**16. Given a matrix-vector equation AX=b. Write a program to find out the value of X using solve(), given A and b as below**

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

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

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

```
import numpy as np

print("********************************")
print("Name: Ashish P S")
print("Roll No: SJC22MCA-2015")
print("Batch: 2022-24")
print("********************************")
print()

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.")
```




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** $A = U\Sigma V^T$

```
import numpy as np
print("********************************")
print("Name: Ashish P S")
print("Roll No: SJC22MCA-2015")
print("Batch: 2022-24")
print("********************************")
```

```
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)
```

```
Run:   q17_SVD
/home/sjcet/PycharmProjects/pythonProject1/venv/bin/python
********************************
Name: Ashish P S
Roll No: SJC22MCA-2015
Batch: 2022-24
********************************

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
```