**CYCLE 2**

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

**Code:**

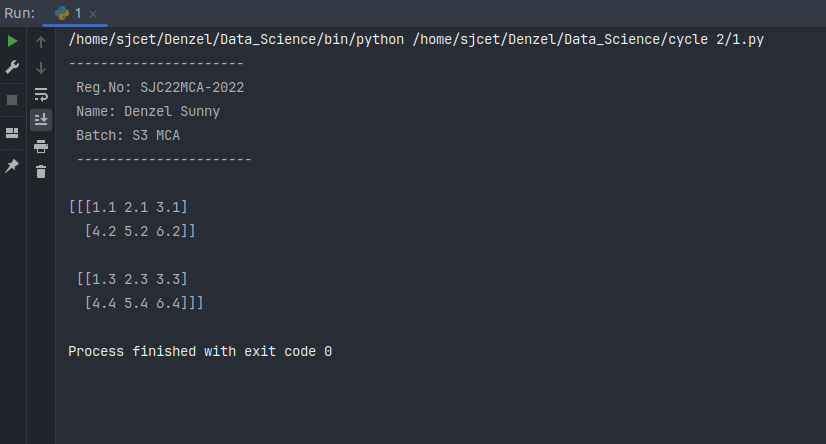
import numpy as np

print("----------------------\n Reg.No: SJC22MCA-2022\n Name: Denzel Sunny\n Batch: S3 MCA\n ----------------------\n")

arr = np.array([[[1.1, 2.1, 3.1], [4.2, 5.2, 6.2]], [[1.3, 2.3, 3.3], [4.4, 5.4, 6.4]]], dtype='f')

print(arr)

**Output:**



**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("----------------------\n Reg.No: SJC22MCA-2022\n Name: Denzel Sunny\n Batch: S3 MCA\n ----------------------\n")

complex\_array = np.array([[1+2j, 2+3j, 3+4j], [4+5j, 5+6j, 6+7j]], dtype=complex)

print("2D complex array: ")

print(complex\_array)

rows, cols = complex\_array.shape

print(f"\nNumber of rows: {rows}")

print(f"Number of columns: {cols}")

dimensions = complex\_array.ndim

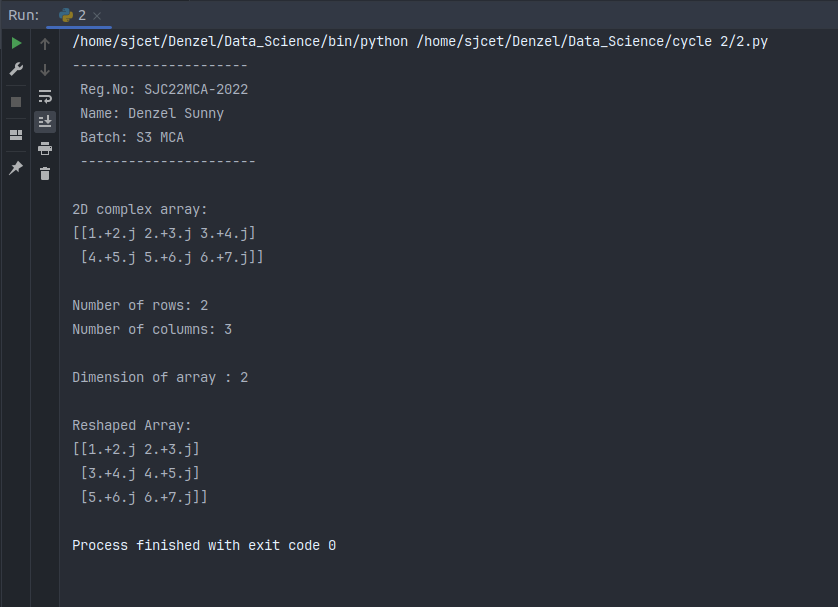
print(f"\nDimension of array : {dimensions}")

reshaped\_array = complex\_array.reshape(3,2)

print("\nReshaped Array: ")

print(reshaped\_array)

**Output:**



**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("----------------------\n Reg.No: SJC22MCA-2022\n Name: Denzel Sunny\n Batch: S3 MCA\n ----------------------\n")

uninitialized\_array = np.empty((3, 3))

print("uninitialized Array: ")

print(uninitialized\_array)

ones\_array = np.ones((2, 4))

print("\nArray with all elements as 1: ")

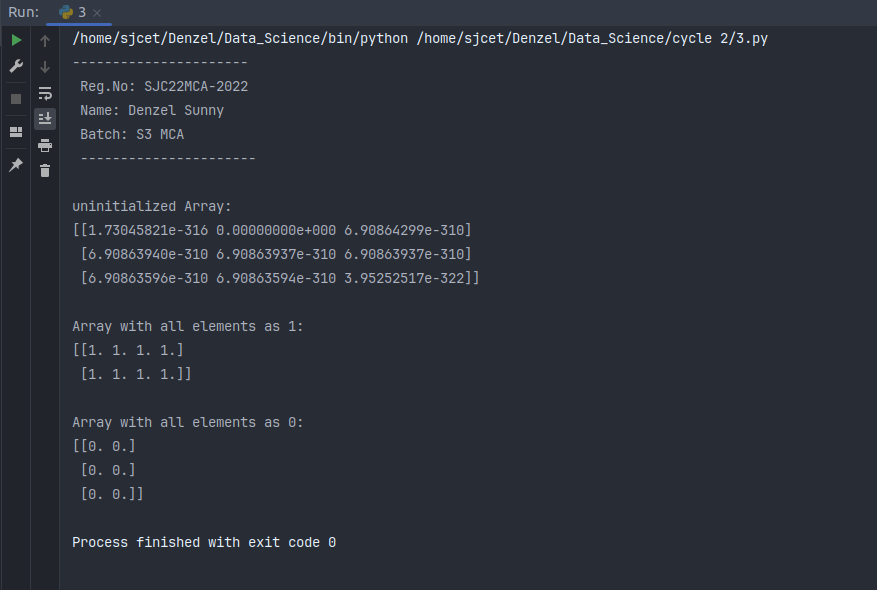
print(ones\_array)

zeros\_array = np.zeros((3, 2))

print("\nArray with all elements as 0: ")

print(zeros\_array)

**Output:**



**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:**

import numpy as np

print("----------------------\n Reg.No: SJC22MCA-2022\n Name: Denzel Sunny\n Batch: S3 MCA\n ----------------------\n")

arr = np.arange(10)

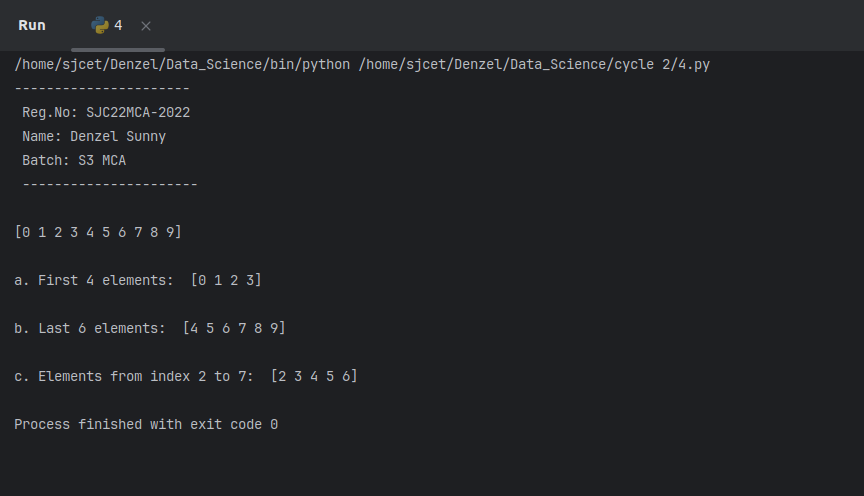
print(arr)

print("\na. First 4 elements: ", arr[:4])

print("\nb. Last 6 elements: ", arr[-6:])

print("\nc. Elements from index 2 to 7: ", arr[2:7])

**Output:**



**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("----------------------\n Reg.No: SJC22MCA-2022\n Name: Denzel Sunny\n Batch: S3 MCA\n ----------------------\n")

print("Array with first 15 even numbers: ")

even\_numbers = np.arange(2, 32, 2)

print(even\_numbers)

subset\_a = even\_numbers[2:9:2]

print("\na. Elements from index 2 to 8 with step 2: ", subset\_a)

subset\_b = even\_numbers[-3:]

print("\nb. Last 3 elements of the array using negative index: ", subset\_b)

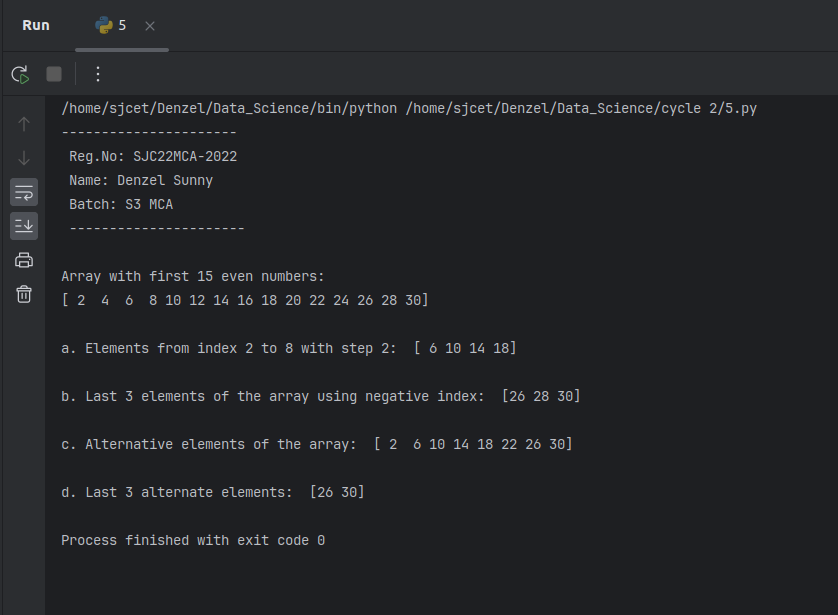
subset\_c = even\_numbers[::2]

print("\nc. Alternative elements of the array: ", subset\_c)

subset\_d = even\_numbers[-3::2]

print("\nd. Last 3 alternate elements: ", subset\_d)

**Output:**



**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("----------------------\n Reg.No: SJC22MCA-2022\n Name: Denzel Sunny\n Batch: 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)

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: ")

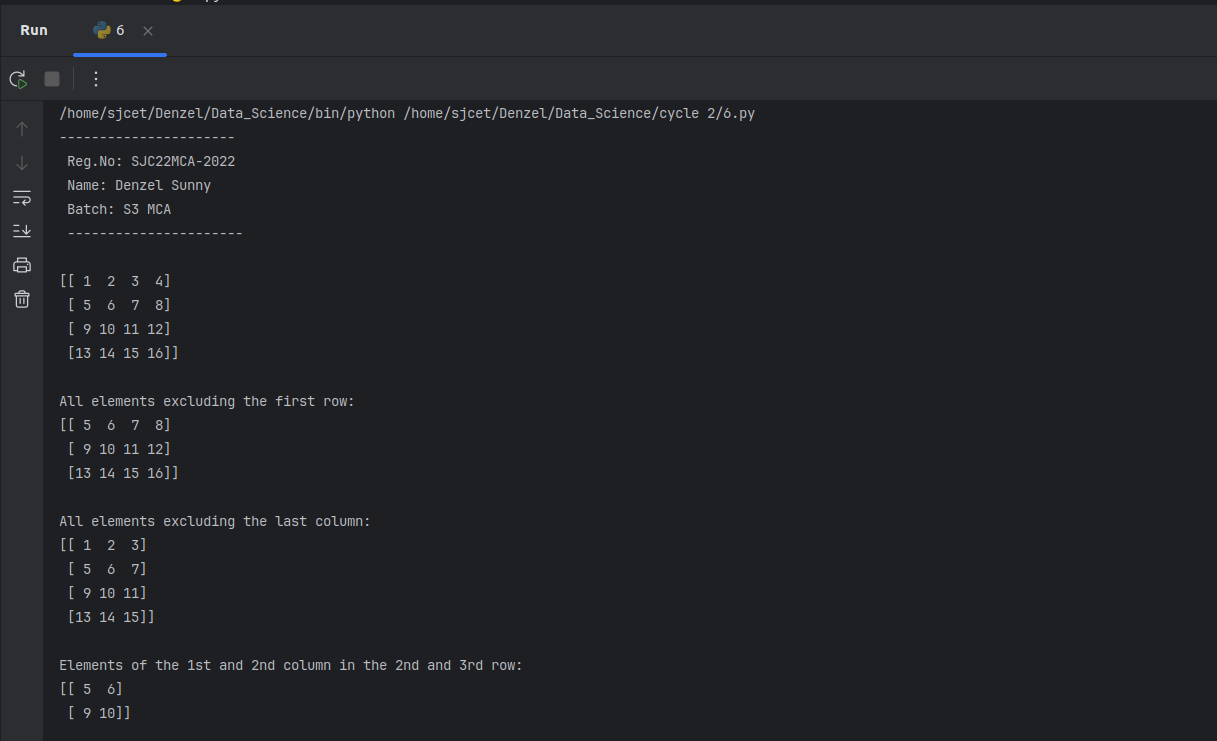
print(e\_result)

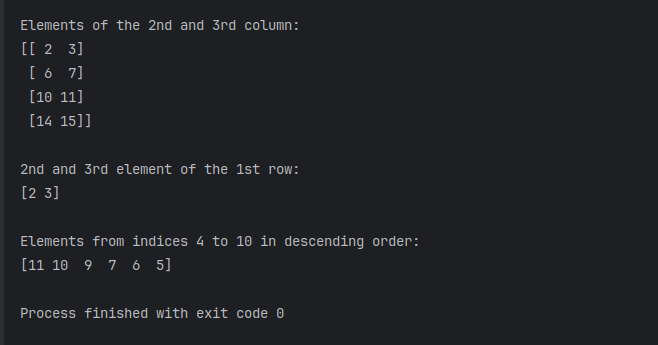
f\_result = array\_2d[1:3, 0:3].flatten()[::-1]

print("\nElements from indices 4 to 10 in descending order: ")

print(f\_result)

**Output:**





**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("----------------------\n Reg.No: SJC22MCA-2022\n Name: Denzel Sunny\n Batch: 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]])

print("matrix 1: \n",matrix1,"\nmatrix 2: \n", matrix2)

add = matrix1 + matrix2

print("\naddition result: ")

print(add)

sub = matrix1 - matrix2

print("\nSubtraction result: ")

print(sub)

element\_multiplicaiton = matrix1 \* matrix2

print("\nElementwise multiplicaiton: ")

print(element\_multiplicaiton)

mask = (matrix2 != 0)

div = np.divide(matrix1, matrix2, where=mask)

print("\nDivision result: ")

print(div)

transpose = matrix1.T

print("\nTranspose of Matrix: ")

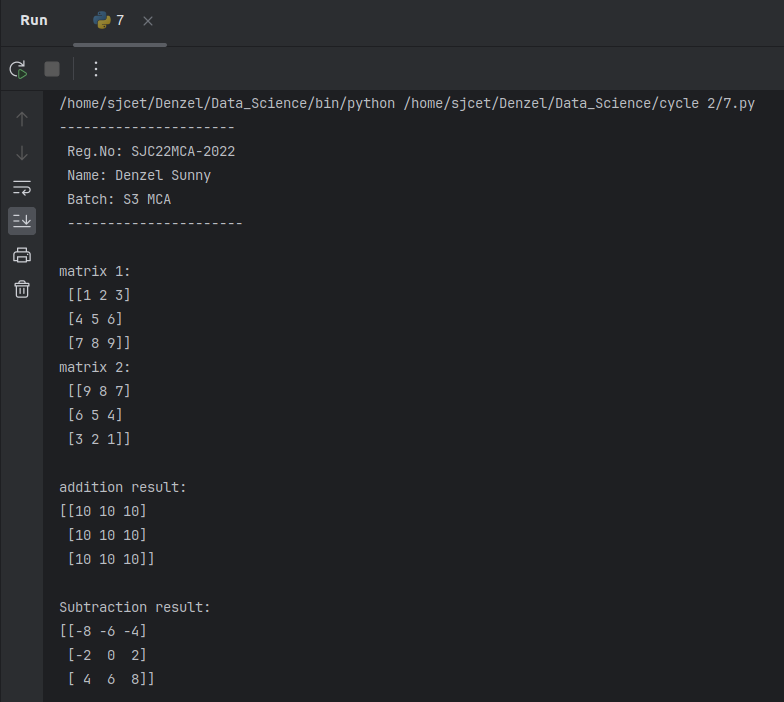
print(transpose)

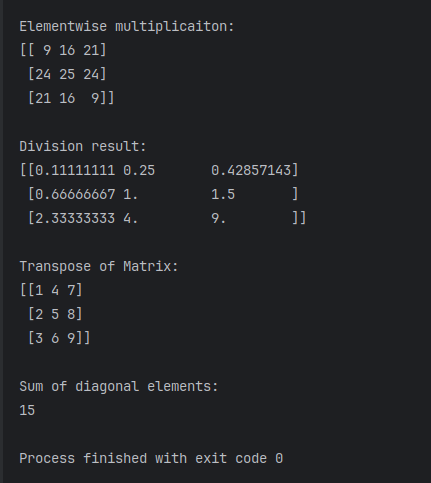
diagonal = np.trace(matrix1)

print("\nSum of diagonal elements: ")

print(diagonal)

**Output:**





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

**Code:**

import numpy as np

print("----------------------\n Reg.No: SJC22MCA-2022\n Name: Denzel Sunny\n Batch: S3 MCA\n ----------------------\n")

arr1 = np.array([1, 2, 3, 4, 5])

# Insert element at specific index (15 at index 2)

arr1\_insert = np.insert(arr1, 2, 15)

print("Original array: ",arr1)

print("After insertion: ",arr1\_insert)

arr2 = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

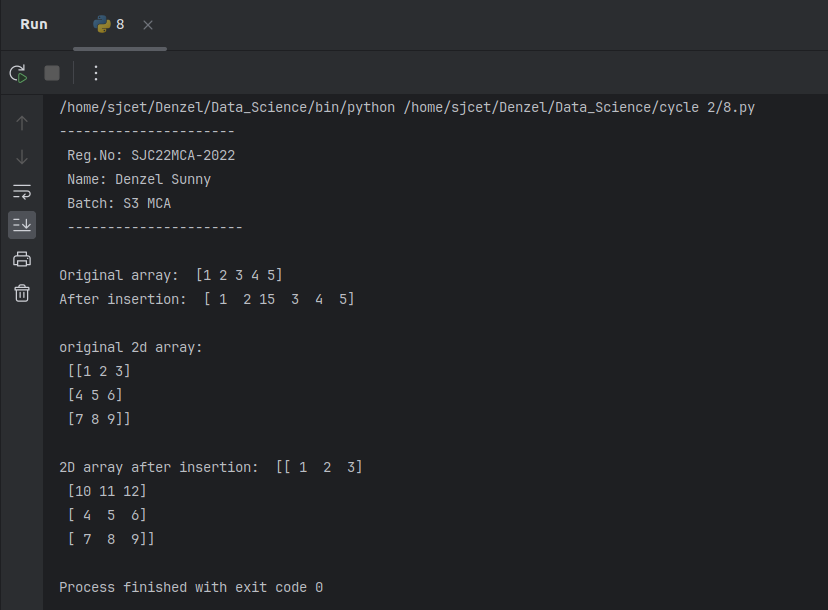
# insert at specific row index

arr2\_inserted = np.insert(arr2, 1, [10, 11, 12], axis=0)

print("\noriginal 2d array: \n", arr2)

print("\n2D array after insertion: ", arr2\_inserted)

**Output:**



**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("----------------------\n Reg.No: SJC22MCA-2022\n Name: Denzel Sunny\n Batch: S3 MCA\n ----------------------\n")

arr1 = np.array([1,2,3,4,5])

diagonal\_elements = np.diag(arr1)

print("Original Array: ", arr1)

print("Diagonal elements: ", diagonal\_elements)

arr2 = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

diag\_elments = np.diag(arr2)

print("Original 2d array: \n", arr2)

print("Diagonal elements: ", diag\_elments)

arr\_nonsqr = np.array([[1, 2, 3],

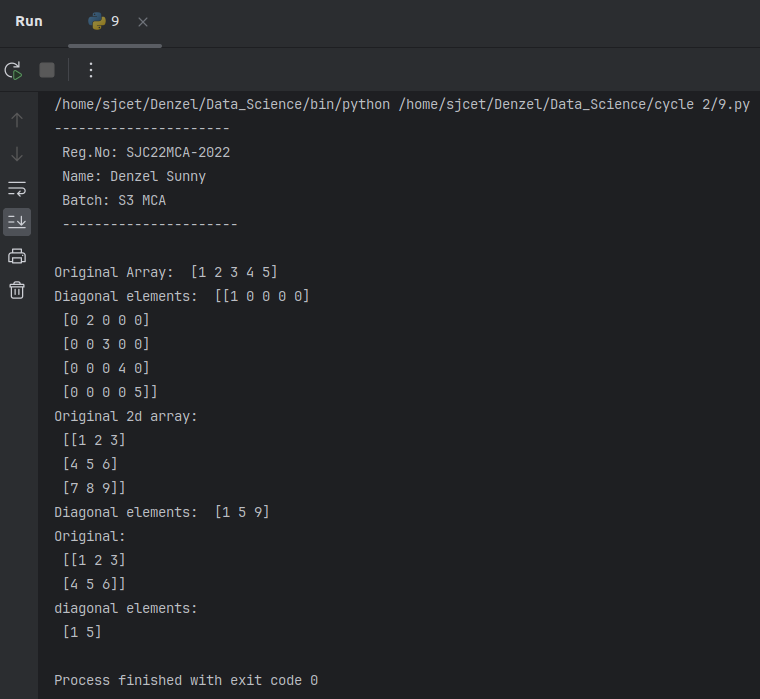
[4, 5, 6]])

diag\_nonsqr = np.diag(arr\_nonsqr)

print("Original: \n", arr\_nonsqr)

print("diagonal elements: \n", diag\_nonsqr)

**Output:**



**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("----------------------\n Reg.No: SJC22MCA-2022\n Name: Denzel Sunny\n Batch: S3 MCA\n ----------------------")

n = 4

matrix = np.random.randint(1, 10, (n, n))

try:

inverse\_matrix = np.linalg.inv(matrix)

except np.linalg.LinAlgError:

inverse\_matrix = None

print("The Matrix is not invertible.")

rank = np.linalg.matrix\_rank(matrix)

determinant = np.linalg.det(matrix)

matrix\_id = matrix.flatten()

eigenvalues, eigenvectors = np.linalg.eig(matrix)

print("Original matrix: ", matrix)

if inverse\_matrix is not None:

print("Inverse matrix: \n", inverse\_matrix)

print("Rank of Matrix: ", rank)

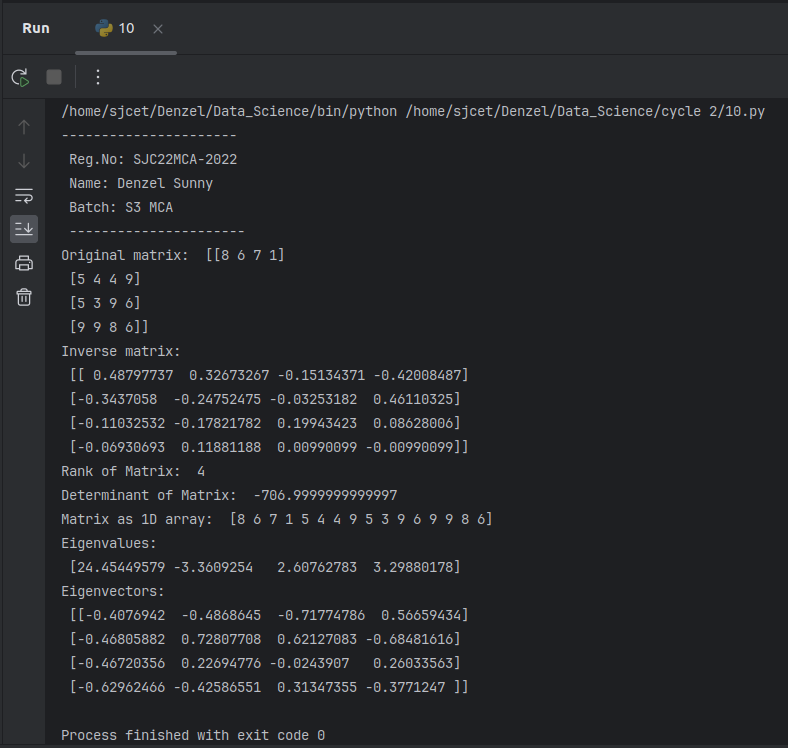
print("Determinant of Matrix: ", determinant)

print("Matrix as 1D array: ", matrix\_id)

print("Eigenvalues: \n", eigenvalues)

print("Eigenvectors: \n", eigenvectors)

**Output:**



**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:**

import numpy as np

print("----------------------\n Reg.No: SJC22MCA-2022\n Name: Denzel Sunny\n Batch: S3 MCA\n ----------------------\n")

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]

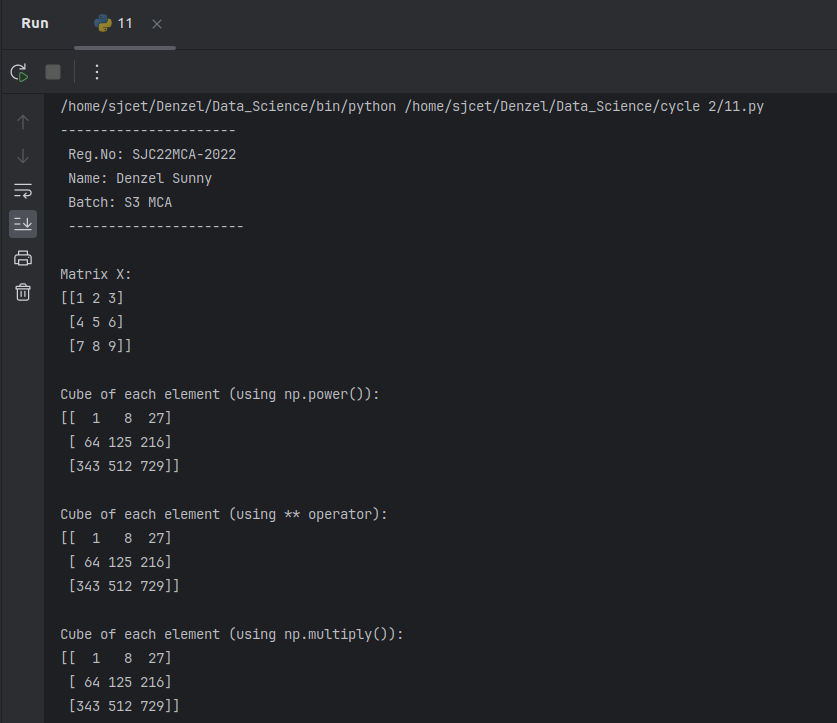
powered\_matrices = [np.power(X, exp) for exp in exponentials]

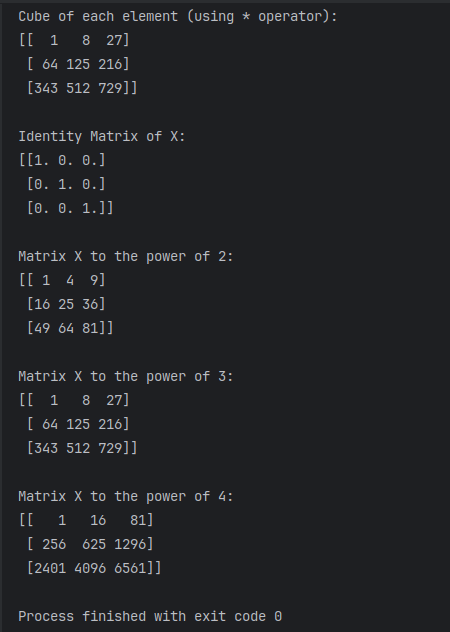
for i, exp in enumerate(exponentials):

print(f"\nMatrix X to the power of {exp}:")

print(powered\_matrices[i])

**Output:**





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

**Code:**

import numpy as np

print("----------------------\n Reg.No: SJC22MCA-2022\n Name: Denzel Sunny\n Batch: S3 MCA\n ----------------------\n")

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

[4, 5, 6],

[7, 8, 9]])

# Create matrix Y with the same dimensions as X

Y = np.random.randint(1, 10, X.shape)

# Perform the operation X^2 + 2Y

result = X\*\*2 + 2 \* Y

# Display the original matrices and the result

print("Matrix X:")

print(X)

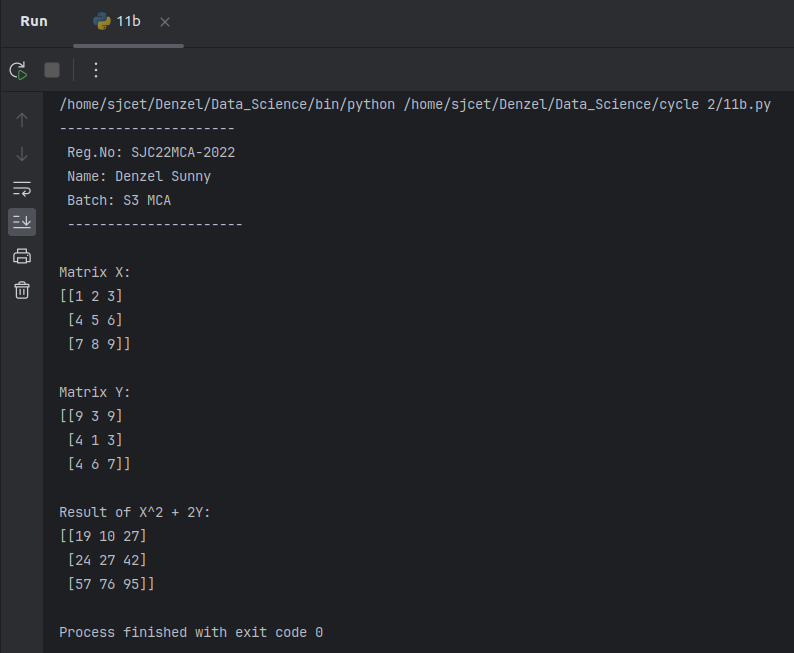
print("\nMatrix Y:")

print(Y)

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

print(result)

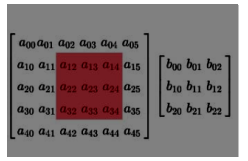
**Output:**



**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("----------------------\n Reg.No: SJC22MCA-2022\n Name: Denzel Sunny\n Batch: 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, 4, 6],

[1, 3, 5],

[7, 8, 9]])

print("Matrix A:")

print(A)

print("\nMatrix B:")

print(B)

# Extract a sub-matrix of dimension 3x3 from A

sub\_matrix\_A = A[:3, :3]

# Multiply the sub-matrix with matrix B

result\_matrix = np.dot(sub\_matrix\_A, B)

# Replace the extracted sub-matrix in A with the result matrix

A[:3, :3] = result\_matrix

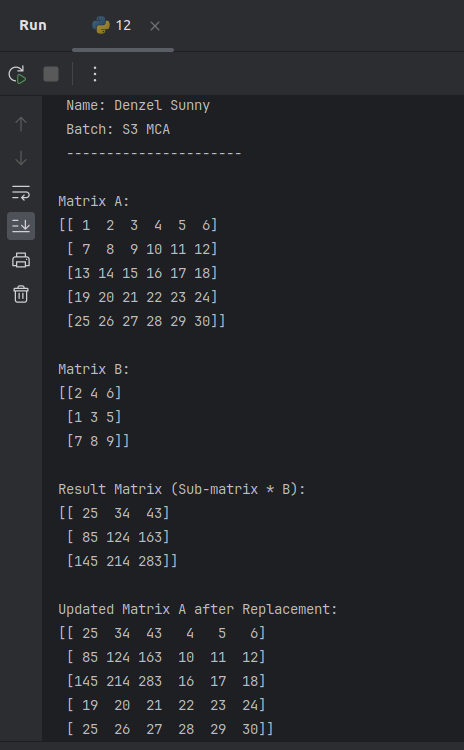
print("\nResult Matrix (Sub-matrix \* B):")

print(result\_matrix)

print("\nUpdated Matrix A after Replacement:")

print(A)

**Output:**



**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("----------------------\n Reg.No: SJC22MCA-2022\n Name: Denzel Sunny\n Batch: 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],

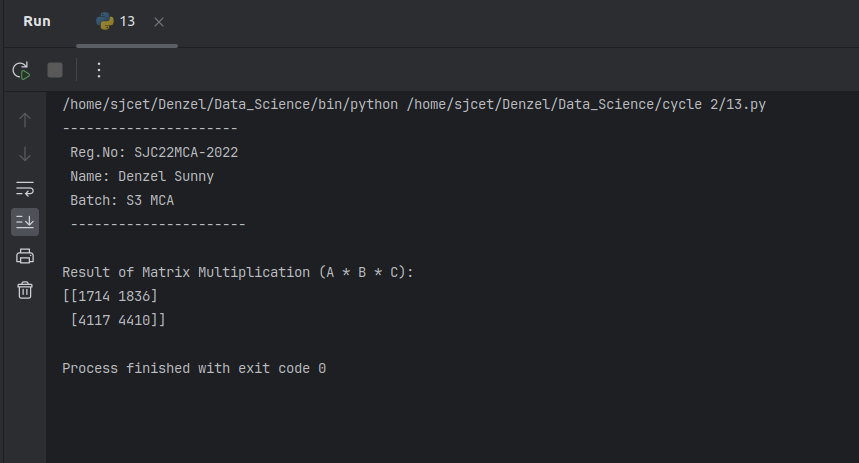
[15, 16]])

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

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

print(result)

**Output:**



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

**Code:**

import numpy as np

print("----------------------\n Reg.No: SJC22MCA-2022\n Name: Denzel Sunny\n Batch: 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]])

print(matrix)

if is\_symmetric(matrix):

print("The matrix is symmetric.")

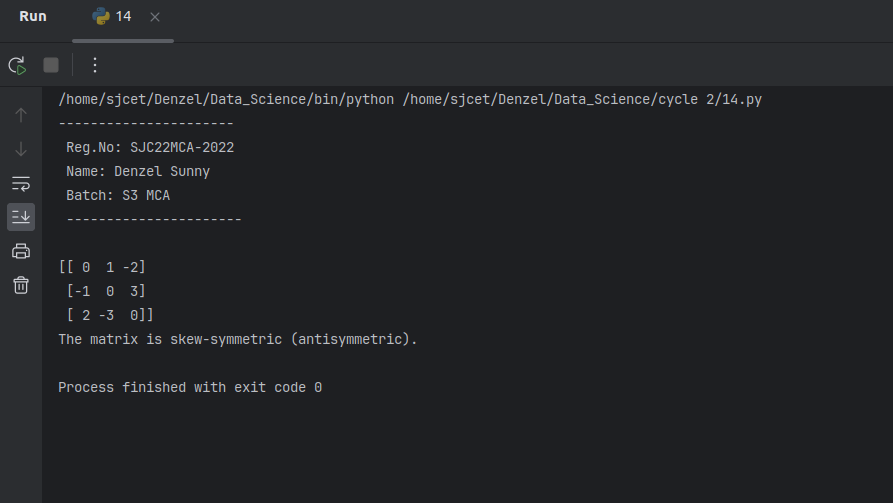
elif is\_skew\_symmetric(matrix):

print("The matrix is skew-symmetric (antisymmetric).")

else:

print("The matrix is neither symmetric nor skew-symmetric.")

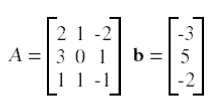
**Output:**



**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.**



**Code:**

import numpy as np

print("----------------------\n Reg.No: SJC22MCA-2022\n Name: Denzel Sunny\n Batch: S3 MCA\n ----------------------\n")

# Define matrix A and vector b

A = np.array([[2, 1],

[1, 3]])

b = np.array([5, 7])

# Use np.linalg.solve() to find X

X = np.linalg.solve(A, b)

# Display the solution X

print("Matrix A:")

print(A)

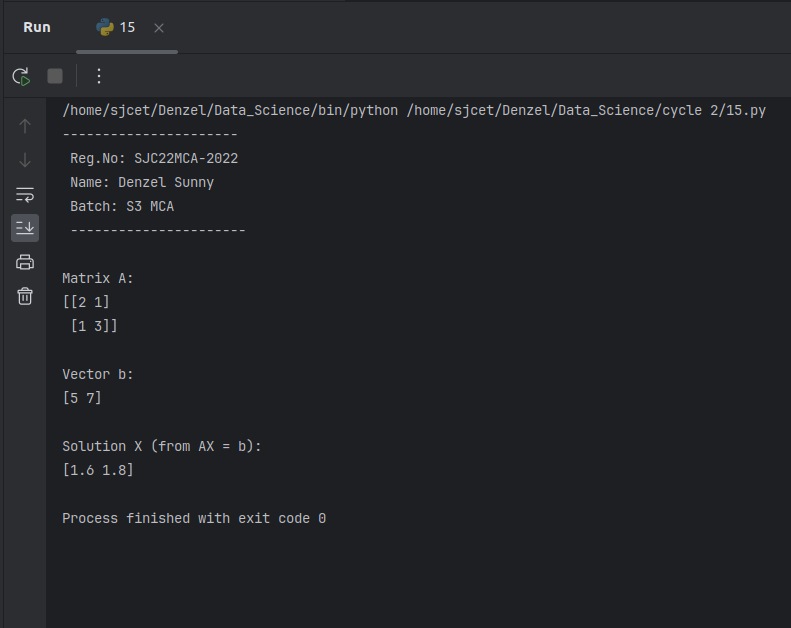
print("\nVector b:")

print(b)

print("\nSolution X (from AX = b):")

print(X)

**Output:**



**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.**



**Code:**

import numpy as np

print("----------------------\n Reg.No: SJC22MCA-2022\n Name: Denzel Sunny\n Batch: S3 MCA\n ----------------------\n")

# Define the matrix A

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

[4, 5, 6],

[7, 8, 9]])

# Perform SVD

U, S, Vt = np.linalg.svd(A)

# Reconstruct the matrix A from the three matrices obtained after SVD

reconstructed\_A = U @ np.diag(S) @ Vt

# Display the original matrix A

print("Matrix A:")

print(A)

# Display the matrices U, S, and Vt

print("\nMatrix U:")

print(U)

print("\nMatrix S (Diagonal Matrix):")

print(S)

print("\nMatrix Vt (Transpose of V):")

print(Vt)

# Display the reconstructed matrix A

print("\nReconstructed Matrix A:")

print(reconstructed\_A)

**Output:**

