1.1

```
a = complex(input("Enter 1st complex no.: "))
b = complex(input("Enter 2nd complex no.: "))
print("Given complex numbers are: 1.", a, " 2.", b)
print("Addition of two complex numbers: ", a, "+", b, "=", a+b)
print("Subtraction of two complex numbers: ", a, "-", b, "=", a-b)
print("Multiplication of two complex numbers:", a, "*", b, "=", a*b)
print("Division of two complex numbers:", a, "/", b, "=", a/b)
```

1.2

```
a = complex(input("Enter 1st complex no.: "))
b = complex(input("Enter 2nd complex no.: "))
print("The conjugate of the given complex numbers are: \nIn 1: ", a.conjugate(), "\nIn 2: ", b.conjugate())
```

1.3

```
import matplotlib.pyplot as plt

S = {3+3j, 2+1j}
x = [z.real for z in S]
y = [z.imag for z in S]

plt.plot(x, y, 'o')  # 'o' for points
plt.axis([-6, 6, -6, 6])
plt.show()
```

1.4

```
import matplotlib.pyplot as plt

angle = int(input("Enter angle: "))

if angle == 90:
  S1 = {2*x for x in {3+2j}}  # Adjusting the set to create a simple complex number
  print(S1)
  x = [z.real for z in S1]
  y = [z.imag for z in S1]
  plt.plot(x, y, 'o')
  plt.axis([-6, 6, -6, 6])
  plt.show()

elif angle == 180:
  S1 = {x-1 for x in {3+2j}}  # Adjusting the set for a different transformation
  print(S1)
  x = [z.real for z in S1]
  y = [z.imag for z in S1]
  plt.plot(x, y, 'o')
  plt.axis([-6, 6, -6, 6])
  plt.show()
```

```
elif angle == 270:
    S1 = {x-1j for x in {3+2j}}  # Another transformation
    print(S1)
    x = [z.real for z in S1]
    y = [z.imag for z in S1]
    plt.plot(x, y, 'o')
    plt.axis([-6, 6, -6, 6])
    plt.show()

else:
    print("Invalid angle.")
```

2

```
import numpy as np

# Part A: Displaying 2 matrices
a = np.array([[1, 2], [3, 4]])
print("First matrix is: \n", a)

b = np.array([[1, 2, 3], [-4, -5, 0], [7, 8, 9]])
print("Second matrix is: \n", b)

# Part B: Retrieving rows and specific elements
a = np.array([[1, 2, 3], [11, 12, 13], [12, 23, 34]])
print("\nMatrix: \n", a)
print("1st row of matrix: ", a[0, :])
print("2nd row of matrix: ", a[1, :])
print("3rd row of matrix: ", a[2, :])
print("2nd element of 3rd row of matrix: ", a[2, 1])
print("3rd element of 2nd row of matrix: ", a[1, 2])

# Part C: Retrieving columns of a matrix
a = np.array([[1, 2, 3], [11, 12, 13], [12, 23, 34]])
print("\nMatrix: \n", a)
print("1st column of matrix: ", a[:, 0])
print("2nd column of matrix: ", a[:, 1])
print("3rd column of matrix: ", a[:, 2])
```

3

3.1

```
# Part A: Input matrix dimensions and elements from the user
rows = int(input("Enter the number of rows: "))
columns = int(input("Enter the number of columns: "))

matrix = []
print("Enter the matrix elements row-wise:")
for i in range(rows):
    row = [int(x) for x in input().split()]  # Splitting and converting input to integers
    matrix.append(row)
```

```
print("Matrix:")
for row in matrix:
    print(row)
```

3.2

```
# Part B: Matrix operations using NumPy
import numpy as np

a = np.array([[1, 2, 3], [11, 12, 13], [12, 23, 34]])
print("Matrix:\n", a, "\n")

# Retrieving rows
print("1st row of matrix: ", a[0, :])
print("2nd row of matrix: ", a[1, :])
print("3rd row of matrix: ", a[2, :])

# Retrieving columns
print("1st column of matrix: ", a[:, 0])
print("2nd column of matrix: ", a[:, 1])
print("3rd column of matrix: ", a[:, 2])
```

3.3

```
import numpy as np

a = 5  # Scalar value
M = np.array([[1, 2], [3, 4]])  # Matrix

print("Matrix:\n", M)

# Scalar multiplication
b = a * M
print("\nScalar multiplication of the matrix is:\n", b)
```

3.4

```
import numpy as np

a = np.array([[12, 7], [4, 5], [3, 8]])  # Matrix

print("Matrix is:\n", a)

# Transpose of the matrix
print("\nTransposed matrix is:\n", a.transpose())
```

4

```
import numpy as np

# Input number of rows and columns
r = int(input("Enter the number of rows: "))
c = int(input("Enter the number of columns: "))

# Check if the matrix is square
```

```
if r != c:
    print("Matrix must be square to be invertible")
else:
    print("Enter the matrix elements row-wise:")

    # Inputting the matrix
    matrix = np.array([list(map(float, input().split())) for _ in range(r)])

    # Calculate the determinant
    det = np.linalg.det(matrix)

    # Check if the determinant is zero
    if det == 0:
        print("The matrix is not invertible (determinant is zero)")
    else:
        # Calculate the inverse of the matrix
        inverse_matrix = np.linalg.inv(matrix)
        print("The matrix is invertible.")
        print("Inverse of the matrix:")
        print(inverse_matrix)
```

5.A

```
import numpy as np


# Vector and matrix definitions

x = np.array([2, 1, 3])  # Corrected the vector definition

y = np.array([[4, 5], [6, 7], [8, 9]])  # Corrected the matrix definition


print("Vector:", x)

print("Matrix:\n", y)


# Vector-matrix multiplication

result = np.dot(x, y)

print("The vector-matrix multiplication is:\n", result)
```

5.B

```
import numpy as np


# Matrix definitions

x = np.array([[1, 2], [3, 4]])  # Corrected the matrix definition

y = np.array([[1, 2], [3, 4]])  # Corrected the matrix definition
```

```
print("1st matrix:\n", x)
print("2nd matrix:\n", y)

# Matrix-matrix multiplication
result = np.dot(x, y)
print("The matrix-matrix multiplication is:\n", result)
```

6.A GCD

```
import math

# Input two numbers
a = int(input("Enter a number: "))
b = int(input("Enter another number: "))

# Calculate GCD
gcd_value = math.gcd(a, b)

# Print the result
print("GCD of", a, "and", b, "is", gcd_value)
```

6.B

```
import math

def find_factors(N):
    factors = []
    for i in range(1, int(math.sqrt(N)) + 1):
        if N % i == 0:
            factors.append(i)
            if i != N // i:  # Avoid adding the square root twice
                factors.append(N // i)
    return sorted(factors)

def verify_m_and_n(N):
```

```python
    factors = find_factors(N)
    print(f"Factors of {N}: {factors}")

    for a in factors:
        for b in factors:
            if (a + b) % 2 == 0 and (b - a) % 2 == 0:
                m = (a + b) // 2
                n = (b - a) // 2
                if m**2 - n**2 == N:
                    print(f"Verification: {m}^2 - {n}^2 = {N}")
                    return

    print(f"No valid (m, n) pair found for N = {N}")

# Main function to input a number and verify m and n
N = int(input("Enter a Number: "))
verify_m_and_n(N)
```

7

```python
import numpy as np

def oprojection(of_vec, on_vec):
    x1 = np.array(of_vec)
    x2 = np.array(on_vec)

    print(f"1st vector a = {x1}")
    print(f"2nd vector b = {x2}")

    scalar = np.dot(x1, x2) / np.dot(x2, x2)
    vec = scalar * x2

    print("Projection of a on b = ", vec)
    print("Om Singh 46")
```

```
# Example usage
oprojection([2, 4], [8, 6])
```

8

```
import numpy as np

# Define the matrix A
A = np.array([[-2, 1], [12, -30]])

print("Matrix A:\n", A)

# Compute eigenvalues and eigenvectors
eigen_values, eigen_vectors = np.linalg.eig(A)

print("Eigenvalues of A are:", eigen_values)
print("Eigenvectors of A are:\n", eigen_vectors)
```

9

```
def How_echelon(matrix):
    Rows = len(matrix)
    Cols = len(matrix[0])

    for i in range(Rows):
        if matrix[i][i] != 0:
            divisor = matrix[i][i]
            for j in range(i, Cols):
                matrix[i][j] = matrix[i][j] / divisor
        for k in range(i + 1, Rows):
            factor = matrix[k][i]
            for j in range(i, Cols):
                matrix[k][j] = matrix[k][j] - factor * matrix[i][j]

    return matrix
```

```
A = [[2, 1, -1, -3], [4, 5, -3, -7], [2, 6, -1, 6]]


# Calling the function
ref_matrix = How_echelon(A)


print("Om Singh 46")
for row in ref_matrix:
    print(row)
```