COMPUTER GRAPHICS

PRACTICALFILE

# Department of Computer Science

# Aryabhatta College

# University of Delhi

Session*:2023-2024*

**INDEX**

|  |  |  |
| --- | --- | --- |
| Sno | Practical Questions | Page No |
| 1 | Write a program to implement DDA and Bresenham’s line drawing algorithm**.** | 3 |
| 2 | Write a program to implement mid-point circle drawing algorithm. | 5 |
| 3 | Write a program to clip a line using Cohen and Sutherland line clipping algorithm. | 8 |
| 4 | Write a program to clip a polygon using Sutherland Hodgeman algorithm. | 11 |
| 5 | Write a program to fill a polygon using Scan line fill algorithm | 16 |
| 6 | Write a program to apply various 2D transformations on a 2D object (use homogenous Coordinates). | 19 |
| 7 | Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projection on it. | 23 |
| 8 | Write a program to draw Hermite /Bezier curve | 28 |

**Q1. Write a program to implement DDA and Bresenham’s line drawing algorithm.**

import matplotlib.pyplot as plt

def drawline(x1,x2,y1,y2):

dy=y2-y1

dx=x2-x1

m=dy/dx

x=[]

y=[]

p=(2\*dy)-dx

xnext=x1

ynext=y1

x.append(xnext)

y.append(ynext)

while(xnext<x2 or ynext<y2):

if(m<1):

xnext=x1+1

if(p<0):

ynext=y1

p=p+(2\*dy)

else:

ynext=y1+1

p=p+(2\*(dy-dx))

else:

ynext=y1+1

if(p<0):

xnext=x1

p=p+(2\*dx)

else:

xnext=x1+1

p=p+(2\*(dx-dy))

x.append(xnext)

y.append(ynext)

x1=xnext

y1=ynext

print("x coordinates are :-",x)

print("y coordinates are :-",y)

plt.plot(x,y)

plt.show()

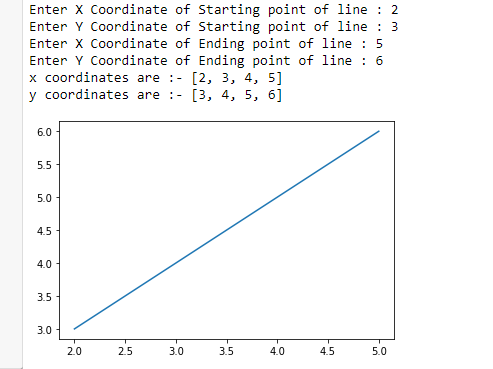
X1=int(input("Enter X Coordinate of Starting point of line : "))

Y1=int(input("Enter Y Coordinate of Starting point of line : "))

X2=int(input("Enter X Coordinate of Ending point of line : "))

Y2=int(input("Enter Y Coordinate of Ending point of line : "))

drawline(X1,X2,Y1,Y2)

**Output**

Q2. Write a program to implement mid-point circle drawing algorithm.

import matplotlib.pyplot as plt

def draw\_circle(x\_center, y\_center, radius):

points = []

x = radius

y = 0

# Initial points#

points.append((x + x\_center, y + y\_center))

if radius > 0:

points.append((x + x\_center, -y + y\_center))

points.append((y + x\_center, x + y\_center))

points.append((-y + x\_center, x + y\_center))

P = 1 - radius

while x > y:

y += 1

if P <= 0:

P = P + 2\*y + 1

else:

x -= 1

P = P + 2\*y - 2\*x + 1

if x < y:

break

points.append((x + x\_center, y + y\_center))

points.append((-x + x\_center, y + y\_center))

points.append((x + x\_center, -y + y\_center))

points.append((-x + x\_center, -y + y\_center))

if x != y:

points.append((y + x\_center, x + y\_center))

points.append((-y + x\_center, x + y\_center))

points.append((y + x\_center, -x + y\_center))

points.append((-y + x\_center, -x + y\_center))

return points

X0=int(input("Enter X coordinates of Center of Circle : "))

Y0=int(input("Enter Y coordinates of Center of Circle : "))

R=int(input("Enter Radius of Circle : "))

points=draw\_circle(X0,Y0,R)

x\_coords, y\_coords = zip(\*points)

# Plotting

plt.figure()

plt.plot(x\_coords, y\_coords, 'ro') # 'ro' plots red dots

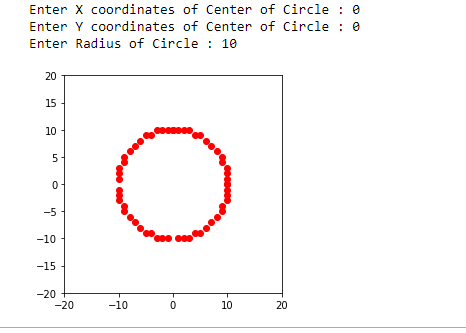
plt.xlim(-20, 20)

plt.ylim(-20, 20)

plt.gca().set\_aspect('equal', adjustable='box') # Ensure the aspect ratio is equal to make the circle look perfect

plt.show()

**Output**

****

**Q3.Write a program to clip a line using Cohen and Sutherland line clipping algorithm.**

import matplotlib.pyplot as plt

def check\_points(xmin,ymin,xmax,ymax,x1,y1):

if(x1>=xmin and x1<=xmax):

if(y1>=ymin and y1<=ymax):

return True

else:

return False

else:

return False

def get\_xy(xmin,ymin,xmax,ymax,x1,y1,x2,y2):

xlist=[]

ylist=[]

m = (y2-y1)/(x2-x1)

plt.ylim(ymin,ymax)

plt.xlim(xmin,xmax)

if(check\_points(xmin,ymin,xmax,ymax,x1,y1) and check\_points(xmin,ymin,xmax,ymax,x2,y2)):

print(" Nothing to Clip : Line Lies in the Interaction Region")

elif((not check\_points(xmin,ymin,xmax,ymax,x1,y1)) and (not check\_points(xmin,ymin,xmax,ymax,x2,y2))):

print(" Sorry but Line doesn't Lies in the Interaction Region")

else:

if(check\_points(xmin,ymin,xmax,ymax,x1,y1)):

A=x1

B=y1

C=x2

D=y2

else:

A=x2

B=y2

C=x1

D=y1

print("x1: ",A,", y1: ",B)

xlist.append(A)

ylist.append(B)

if(C<=xmin): #left

x=xmin

y=B+(m\*(xmin-A))

elif(C>=xmax):

#right

x=xmax

y=B+(m\*(xmax-A))

elif(D<=ymin): #bottom

y=ymin

x=((ymin-B)/m)+A

else: #top

y=ymax

x=((ymax-B)/m)+A

xlist.append(x)

ylist.append(y)

print("x2: ",x,", Y2: ",y)

plt.title('Cohen Sutherland Line Clipping Algorithm')

plt.plot(xlist,ylist)

xmin=input("Enter min x : ")

ymin=input("Enter min y : ")

xmax=input("Enter max x : ")

ymax=input("Enter max y : ")

x1=input("Enter Starting Coordinates of X : ")

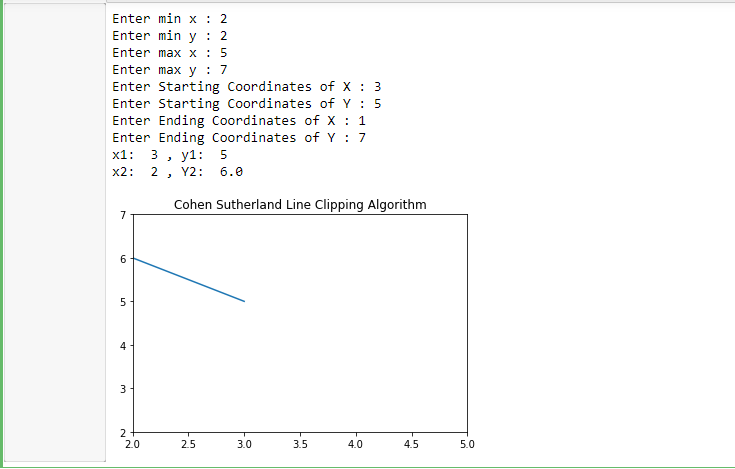
y1=input("Enter Starting Coordinates of Y : ")

x2=input("Enter Ending Coordinates of X : ")

y2=input("Enter Ending Coordinates of Y : ")

get\_xy(int(xmin),int(ymin),int(xmax),int(ymax),int(x1),int(y1),int(x2),int(y2))

**Output**

****

**Q4.Write a program to clip a polygon using Sutherland Hodgeman algorithm.**

import matplotlib.pyplot as plt

from shapely.geometry import Polygon

def draw\_graph(xlist, ylist, V, title):

plt.plot(xlist, ylist, c="red", label="Required Area")

x = []

y = []

for i in V:

x.append(i[0])

y.append(i[1])

plt.plot(x, y, c="blue", label="Polygon")

plt.show()

def check\_points(xmin, ymin, xmax, ymax, x1, y1):

if (x1 >= xmin and x1 <= xmax):

if (y1 >= ymin and y1 <= ymax):

return True

else:

return False

else:

return False

def get\_xy(xmin, ymin, xmax, ymax, x1, y1, x2, y2):

m = (y2 - y1) / (x2 - x1)

if (check\_points(xmin, ymin, xmax, ymax, x1, y1)):

A = x1

B = y1

C = x2

D = y2

else:

A = x2

B = y2

C = x1

D = y1

if (C <= xmin):

# left

x = xmin

y = B + (m \* (xmin - A))

elif (C >= xmax):

# right

x = xmax

y = B + (m \* (xmax - A))

elif (D <= ymin):

# bottom

y = ymin

x = ((ymin - B) / m) + A

else:

# top

y = ymax

x = ((ymax - B) / m) + A

return (x, y)

def clip\_using\_sutherland\_hodgman(x1, y1, x2, y2, V):

check = []

v1 = []

for i in V:

check.append(check\_points(x1, y1, x2, y2, i[0], i[1]))

for i in range(len(check) - 1):

if (check[i] == False):

if (check[i + 1] == True):

print("Outside to inside")

x, y = get\_xy(x1, y1, x2, y2, V[i][0], V[i][1], V[i + 1][0], V[i + 1][1])

v1.append((x, y))

else:

print("outside to outside")

elif (check[i]):

if (check[i + 1] == True):

print("Inside to Inside")

v1.append((V[i][0], V[i][1]))

else:

print("Inside to outside")

v1.append((int(V[i][0]), int(V[i][1])))

x, y = get\_xy(x1, y1, x2, y2, V[i][0], V[i][1], V[i + 1][0], V[i + 1][1])

v1.append((x, y))

print(v1)

return v1

x1, y1 = map(int, input("Enter Lower bound of Cropped Area: ").split())

x2, y2 = map(int, input("Enter Upper bound of Cropped Area: ").split())

xlist = [x1, x1, x2, x2, x1]

ylist = [y1, y2, y2, y1, y1]

count = int(input("Enter Number of vertices in a Polygon: "))

V = []

print("Start entering Points:")

for i in range(count):

print("Enter Coordinates of", i + 1, "Vertex:")

x, y = map(int, input().split())

V.append((x, y))

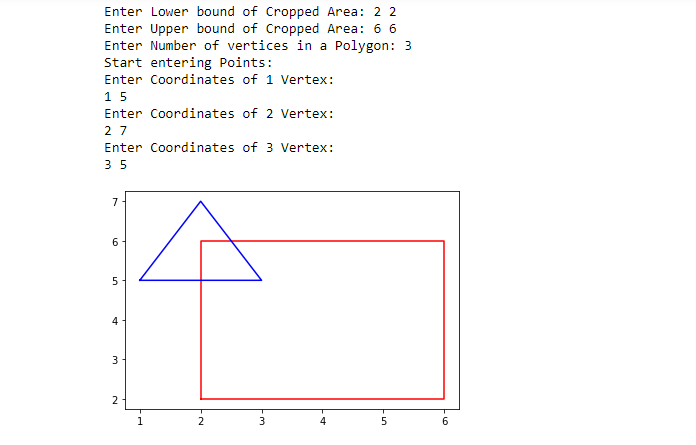
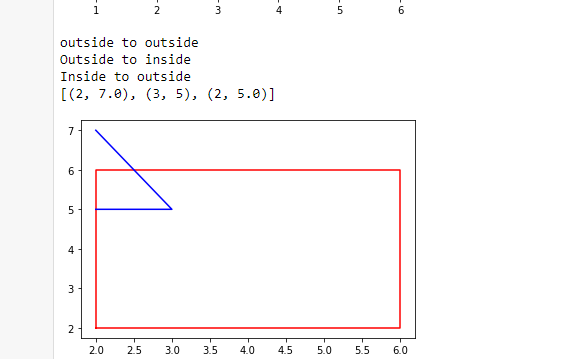
V.append((V[0][0], V[0][1]))

draw\_graph(xlist, ylist, V, "Actual Data is: ")

V1 = clip\_using\_sutherland\_hodgman(x1, y1, x2, y2, V)

draw\_graph(xlist, ylist, V1, "Cropped Data is: ")

Output



**Q5 Write a program to fill a polygon using Scan line fill algorithm**

import matplotlib.pyplot as plt

def draw\_graph(V):

x = []

y = []

for i in V:

x.append(i[0])

y.append(i[1])

plt.title("Before Applying Scan Line Algorithm")

plt.plot(x, y, c="blue", label="Polygon")

plt.show()

return max(y), min(y), max(x), min(x)

def get\_range(V):

y2, y1, x2, x1 = draw\_graph(V)

x = []

y = []

for i in range(len(V) - 1):

x1 = V[i][0]

y1 = V[i][1]

x2 = V[i + 1][0]

y2 = V[i + 1][1]

x3, y3 = calpoints(x1, y1, x2, y2, x, y)

x = x3

y = y3

for i in range(int(len(y) / 2)):

if i != int(len(y) / 2) and i != 0:

plt.plot([x[i], x[y.index(y[i], i + 1)]], [y[i], y[i]], c="red")

plt.title("After Applying Scan Line Algorithm")

plt.show()

def calpoints(x1, y1, x2, y2, xlist, ylist):

m = (y2 - y1) / (x2 - x1)

c = y1 - (m \* x1)

for i in range(abs(y2 - y1) \* 10):

if x1 <= x2:

y = (i / 10) + y1

x = (y - c) / m

xlist.append(x)

ylist.append(y)

else:

y = y1 - (i / 10)

x = (y - c) / m

xlist.append(x)

ylist.append(y)

return xlist, ylist

count = int(input("Enter Number of vertices in a Polygon: "))

V = []

print("Start entering Points:")

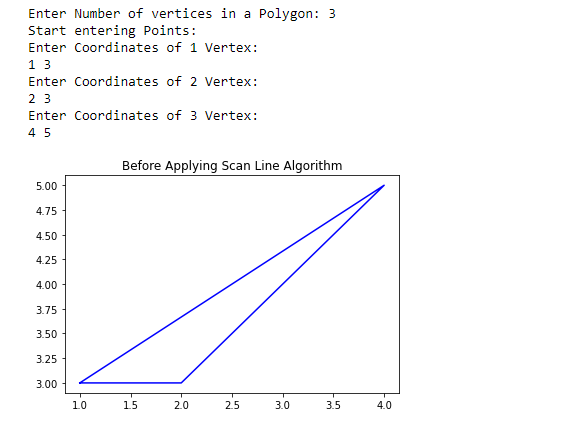
for i in range(count):

print("Enter Coordinates of", i + 1, "Vertex:")

x, y = map(int, input().split())

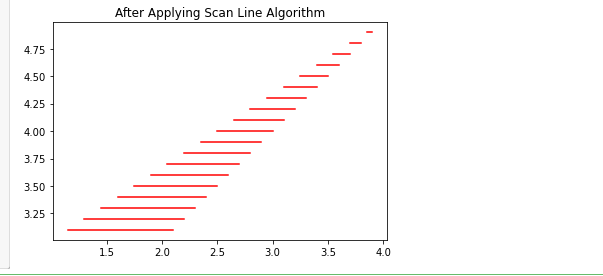
V.append((x, y))

V.append((V[0][0], V[0][1]))



get\_range(V)

Output



**Q6Write a program to apply various 2D transformations on a 2D object (use homogenous Coordinates).**

import matplotlib.pyplot as plt

import numpy as np

import math

def draw\_graph(V, title):

x = [point[0] for point in V]

y = [point[1] for point in V]

x.append(x[0])

y.append(y[0])

plt.plot(x, y)

plt.title(title)

plt.show()

def multiply\_matrix(O, C):

C.append(1)

result = np.dot(O, C)

return [result[0], result[1]]

def Translation(V):

x = []

y = []

for i in range(len(V)):

x.append(V[i][0])

y.append(V[i][1])

print("Performing Translation on 2D Object Provided:")

dx = int(input("Enter Shifting Distance on X-axis: "))

dy = int(input("Enter Shifting Distance on Y-axis: "))

O = [[1, 0, dx], [0, 1, dy], [0, 0, 1]]

V1 = []

draw\_graph(V, "Original Graph")

for point in V:

V1.append(multiply\_matrix(O, point))

draw\_graph(V1, "Translation of 2D Object")

print(V1)

Scaling(V1)

def Scaling(V):

x = []

y = []

for i in range(len(V)):

x.append(V[i][0])

y.append(V[i][1])

print("Performing Scaling on 2D Object Provided:")

sx = float(input("Enter Scaling factor of X-axis: "))

sy = float(input("Enter Scaling factor of Y-axis: "))

O = [[sx, 0, 0], [0, sy, 0], [0, 0, 1]]

V1 = []

for point in V:

V1.append(multiply\_matrix(O, point))

draw\_graph(V1, "Scaling of 2D Object")

Rotation(V1)

def Rotation(V):

x = []

y = []

for i in range(len(V)):

x.append(V[i][0])

y.append(V[i][1])

print("Performing Rotation on 2D Object Provided:")

sign = int(input("Enter Rotation side (0: Anti Clockwise, 1: Clockwise): "))

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

if sign == 0:

O = [[math.cos(math.radians(theta)), -math.sin(math.radians(theta)), 0],

[math.sin(math.radians(theta)), math.cos(math.radians(theta)), 0],

[0, 0, 1]]

else:

O = [[math.cos(math.radians(theta)), math.sin(math.radians(theta)), 0],

[-math.sin(math.radians(theta)), math.cos(math.radians(theta)), 0],

[0, 0, 1]]

V1 = []

for point in V:

V1.append(multiply\_matrix(O, point))

draw\_graph(V1, "Rotation of 2D Object")

count = int(input("Enter Number of vertices of Object: "))

V = []

for i in range(count):

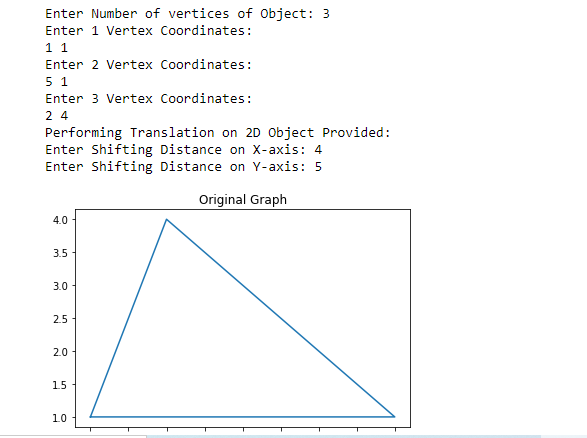
print("Enter", i+1, "Vertex Coordinates:")

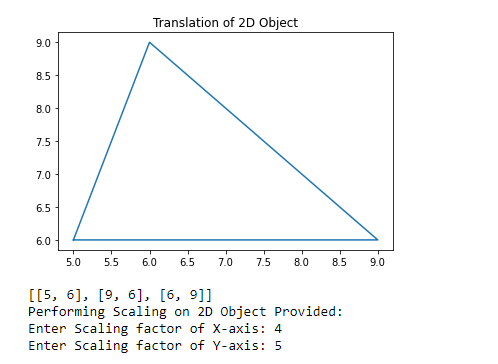
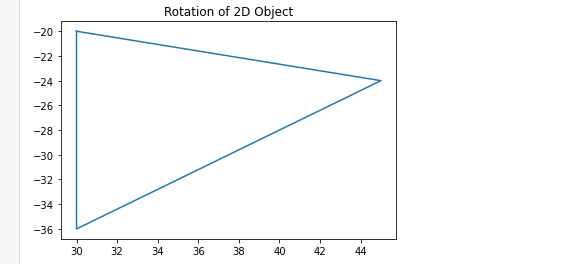
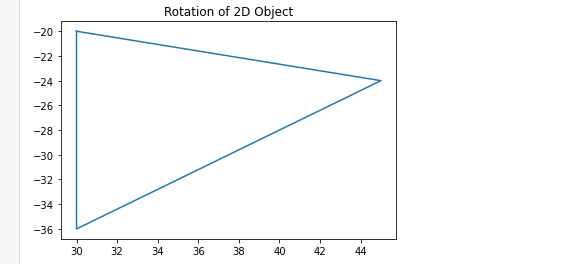
x, y = map(int, input().split())

V.append([x, y])

Translation(V)

Output





**Q7.Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projection on it.**

import numpy as np

import math

def multiply\_matrix(O, C):

C\_homogeneous = C + [1] # Append 1 to represent homogeneous coordinate

result = np.dot(O, C\_homogeneous)

return result[:3].tolist() # Extract only x, y, z coordinates

def Translation(V):

x = []

y = []

z = []

for i in range(len(V)):

x.append(V[i][0])

y.append(V[i][1])

z.append(V[i][2])

print("Performing Translation on 3D Object Provided:")

dx = int(input("Enter Shifting Distance on X-axis: "))

dy = int(input("Enter Shifting Distance on Y-axis: "))

dz = int(input("Enter Shifting Distance on Z-axis: "))

O = [[1, 0, 0, dx],

[0, 1, 0, dy],

[0, 0, 1, dz],

[0, 0, 0, 1]]

V1 = []

for point in V:

V1.append(multiply\_matrix(O, point))

print(V1)

Scaling(V1)

def Scaling(V):

x = []

y = []

z = []

for i in range(len(V)):

x.append(V[i][0])

y.append(V[i][1])

z.append(V[i][2])

print("Performing Scaling on 3D Object Provided:")

sx = float(input("Enter Scaling factor of X-axis: "))

sy = float(input("Enter Scaling factor of Y-axis: "))

sz = float(input("Enter Scaling factor of Z-axis: "))

O = [[sx, 0, 0, 0],

[0, sy, 0, 0],

[0, 0, sz, 0],

[0, 0, 0, 1]]

V1 = []

for point in V:

V1.append(multiply\_matrix(O, point))

print(V1)

Rotation(V1)

def Rotation(V):

x = []

y = []

z = []

for i in range(len(V)):

x.append(V[i][0])

y.append(V[i][1])

z.append(V[i][2])

print("Performing Rotation on 3D Object Provided:")

sign = int(input("Enter Rotation side (0: Anti Clockwise, 1: Clockwise): "))

theta = int(input("Enter Rotation angle (in degrees): "))

Axis = input("Enter Rotation's Axis (x, y, or z): ")

if sign == 0:

if Axis == 'x':

O = [[1, 0, 0, 0],

[0, math.cos(math.radians(theta)), -math.sin(math.radians(theta)), 0],

[0, math.sin(math.radians(theta)), math.cos(math.radians(theta)), 0],

[0, 0, 0, 1]]

elif Axis == 'y':

O = [[math.cos(math.radians(theta)), 0, math.sin(math.radians(theta)), 0],

[0, 1, 0, 0],

[-math.sin(math.radians(theta)), 0, math.cos(math.radians(theta)), 0],

[0, 0, 0, 1]]

else:

O = [[math.cos(math.radians(theta)), -math.sin(math.radians(theta)), 0, 0],

[math.sin(math.radians(theta)), math.cos(math.radians(theta)), 0, 0],

[0, 0, 1, 0],

[0, 0, 0, 1]]

else:

if Axis == 'x':

O = [[1, 0, 0, 0],

[0, math.cos(math.radians(theta)), math.sin(math.radians(theta)), 0],

[0, -math.sin(math.radians(theta)), math.cos(math.radians(theta)), 0],

[0, 0, 0, 1]]

elif Axis == 'y':

O = [[math.cos(math.radians(theta)), 0, -math.sin(math.radians(theta)), 0],

[0, 1, 0, 0],

[math.sin(math.radians(theta)), 0, math.cos(math.radians(theta)), 0],

[0, 0, 0, 1]]

else:

O = [[math.cos(math.radians(theta)), math.sin(math.radians(theta)), 0, 0],

[-math.sin(math.radians(theta)), math.cos(math.radians(theta)), 0, 0],

[0, 0, 1, 0],

[0, 0, 0, 1]]

V1 = []

for point in V:

V1.append(multiply\_matrix(O, point))

print(V1)

count = int(input("Enter Number of vertices of Object: "))

V = []

for i in range(count):

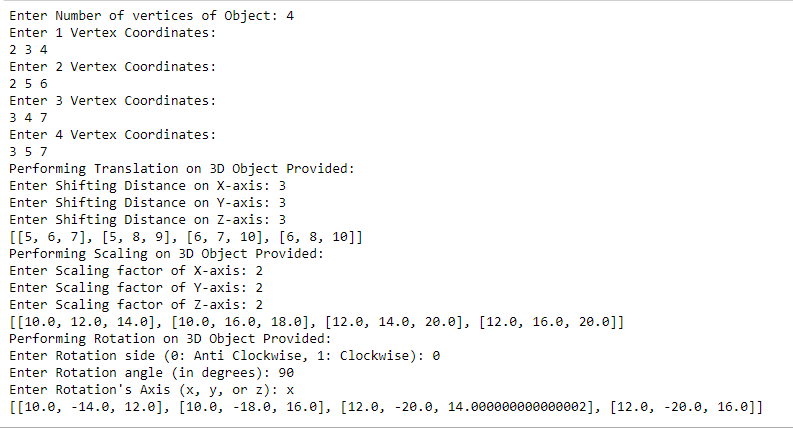
print("Enter", i + 1, "Vertex Coordinates:")

x, y, z = map(int, input().split())

V.append([x, y, z])

Translation(V)

Output



**Q8Write a program to draw Hermite /Bezier curve**

import numpy as np

import matplotlib.pyplot as plt

def h00(t):

return 2\*t\*\*3 - 3\*t\*\*2 + 1

def h10(t):

return t\*\*3 - 2\*t\*\*2 + t

def h01(t):

return -2\*t\*\*3 + 3\*t\*\*2

def h11(t):

return t\*\*3 - t\*\*2

P0 = np.array([5, 0])

P1 = np.array([1, 1])

V0 = np.array([1, 10])

V1 = np.array([2, 1])

t\_v = np.linspace(0, 1, 100)

Bezier\_pts = np.zeros((2, len(t\_v)))

for i, t in enumerate(t\_v):

Bezier\_pts[:, i] = (1-t)\*\*3 \* P0 + 3\*t\*(1-t)\*\*2 \* (P0 + V0) + 3\*t\*\*2\*(1-t) \* (P1 - V1) + t\*\*3 \* P1

Hermite\_pts = np.zeros((2, len(t\_v)))

for i, t in enumerate(t\_v):

Hermite\_pts[:, i] = h00(t) \* P0 + h10(t) \* V0 + h01(t) \* P1 + h11(t) \* V1

plt.plot(Bezier\_pts[0], Bezier\_pts[1], label="Bezier")

plt.plot(Hermite\_pts[0], Hermite\_pts[1], label="Hermite")

plt.scatter([P0[0], P1[0]], [P0[1], P1[1]], label="Control points")

plt.legend()

plt.show()

Output

