Contents

[1. Write a program to implement Bresenham’s line drawing algorithm. 3](#_Toc132027529)

[1.1. Introduction 3](#_Toc132027530)

[1.2. Code 3](#_Toc132027531)

[1.3. Output 5](#_Toc132027532)

[2. Write a program to implement the mid-point circle drawing algorithm. 6](#_Toc132027533)

[2.1. Introduction 6](#_Toc132027534)

[2.2. Code 6](#_Toc132027535)

[2.3. Output 8](#_Toc132027536)

[3. Write a program to clip a line using Cohen and Sutherland line clipping algorithm. 9](#_Toc132027537)

[3.1. Introduction 9](#_Toc132027538)

[3.2. Code 10](#_Toc132027539)

[3.3. Output 12](#_Toc132027540)

[4. Write a program to clip a polygon using Sutherland Hodgeman algorithm 13](#_Toc132027541)

[4.1. Introduction 13](#_Toc132027542)

[4.2. Code 13](#_Toc132027543)

[4.3. Output 17](#_Toc132027544)

[5. Write a program to fill a polygon using scan line fill algorithm 18](#_Toc132027545)8

[5.1. Introduction](#_Toc132027546) 18

[5.2. Code 18](#_Toc132027547)

[5.3. Output 20](#_Toc132027548)

[6. Write a program to apply various 2D transformations on a 2D object (use homogenous coordinates).](#_Toc132027549) 21

[6.1. Introduction](#_Toc132027550) 21

[6.2. Code 22](#_Toc132027551)

[6.3. Output 24](#_Toc132027552)

[7. Write a program to apply Various 3D transformations on a 3D Object.](#_Toc132027553) 26

[7.1. Introduction 26](#_Toc132027554)

[7.2. Code 26](#_Toc132027555)

[7.3. Output 30](#_Toc132027556)

[8. Write a program to draw Hermite/Bezier Curve. 30](#_Toc132027557)

[8.1. Introduction 30](#_Toc132027558)

[8.2. Code 31](#_Toc132027559)

[8.3. Output 31](#_Toc132027560)

# Write a program to implement Bresenham’s line drawing algorithm.

## Introduction

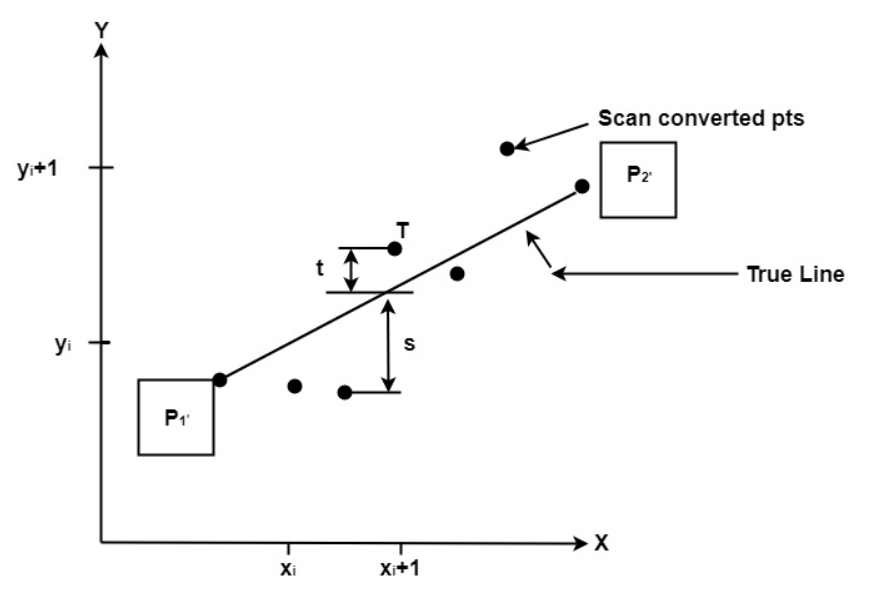
Bresenham's line algorithm is used for scan converting a line. It was developed by Bresenham. It is an efficient method because it involves only integer addition, subtractions, and multiplication operations. These operations can be performed very rapidly so lines can be generated quickly.

In this method, next pixel selected is that one who has the least distance from true line.The method works as follows:

Assume a pixel P1'(x1',y1'),then select subsequent pixels as we work our may to the night, one pixel position at a time in the horizontal direction toward P2'(x2',y2').

Once a pixel in choose at any step,The next pixel is

1. Either the one to its right (lower-bound for the line)
2. One top its right and up (upper-bound for the line)

The line is best approximated by those pixels that fall the least distance from the path between P1',P2'.

## Code

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 cordinates are :- ",x)

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

plt.plot(x,y)

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

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

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

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

drawline(X1,X2,Y1,Y2)

## Output

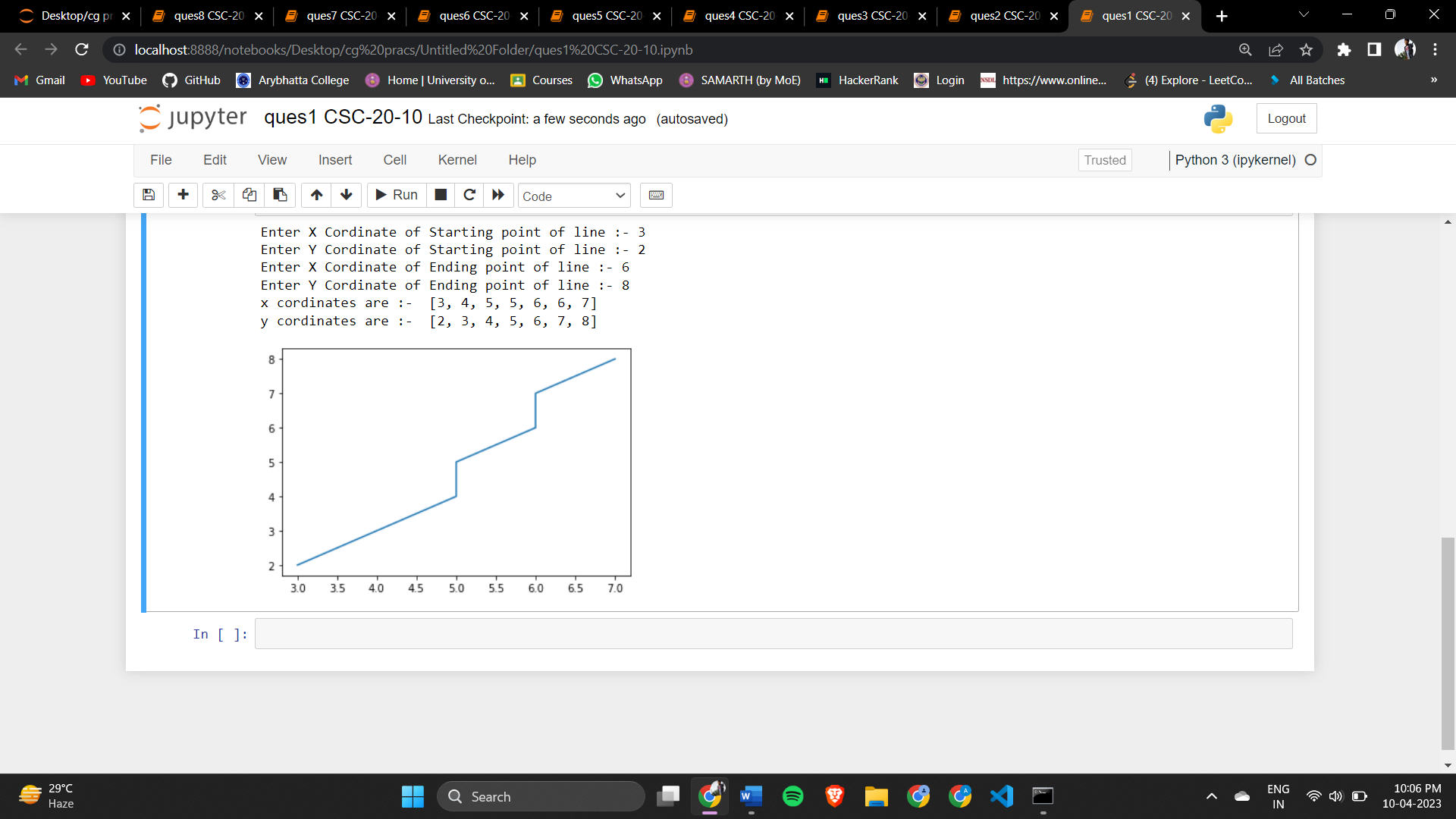
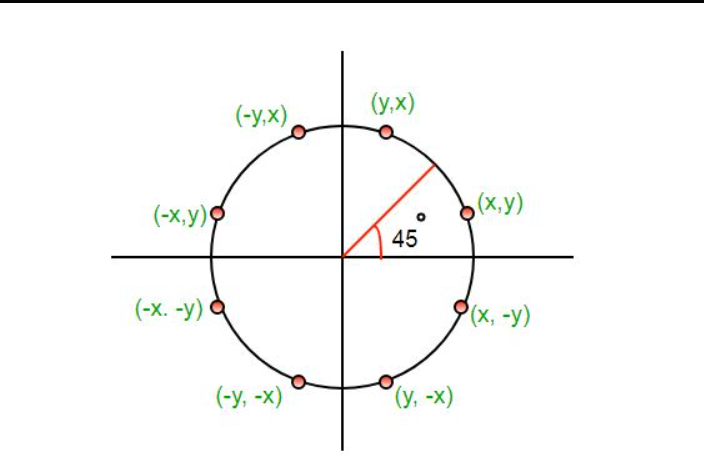


Figure 1. The output of Bresenham’s line drawing algorithm.

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

## Introduction

The **mid-point** circle drawing algorithm is an algorithm used to determine the points needed for rasterizing a circle. 

We use the **mid-point** algorithm to calculate all the perimeter points of the circle in the **first octant** and then print them along with their mirror points in the other octants. This will work because a circle is symmetric about its centre. 

The algorithm is very similar to the [Mid-Point Line Generation Algorithm](https://www.geeksforgeeks.org/mid-point-line-generation-algorithm/). Here, only the boundary condition is different.

For any given pixel (x, y), the next pixel to be plotted is either **(x, y+1)**or**(x-1, y+1).** This can be decided by following the steps below.

1. Find the mid-point **p** of the two possible pixels i.e (x-0.5, y+1)
2. If **p** lies inside or on the circle perimeter, we plot the pixel (x, y+1), otherwise if it’s outside we plot the pixel (x-1, y+1)

## Code

‘from graphics import \*

import time

def drawcircle(x0,y0,r):

xi=0

yi=r

xtemp=[]

ytemp=[]

p=3-(2\*r)

while(xi<=yi):

print("(",xi,",",yi,")")

xtemp.append(xi)

xtemp.append(yi)

ytemp.append(yi)

ytemp.append(xi)

xnext=xi+1

if(p<0):

ynext=yi

p=p+(4\*xnext)+6

else:

ynext=yi-1

p=p+(4\*(xnext-ynext))+10

xi=xnext

yi=ynext

x1=xtemp

y1=ytemp

x=[]

y=[]

for i in xtemp:

x.append(i)

x.append(i)

x.append(-i)

x.append(-i)

for i in ytemp:

y.append(i)

y.append(-i)

y.append(-i)

y.append(i)

win = GraphWin('Ques2:- Mid Point Circle Drawing Algorithm by CSC/20/50', 600, 480)

for i in range(len(x)):

pt = Point(x[i]+x0,y[i]+y0)

pt.draw(win)

time.sleep(10)

win.close()

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

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

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

drawcircle(X0,Y0,R)

## Output

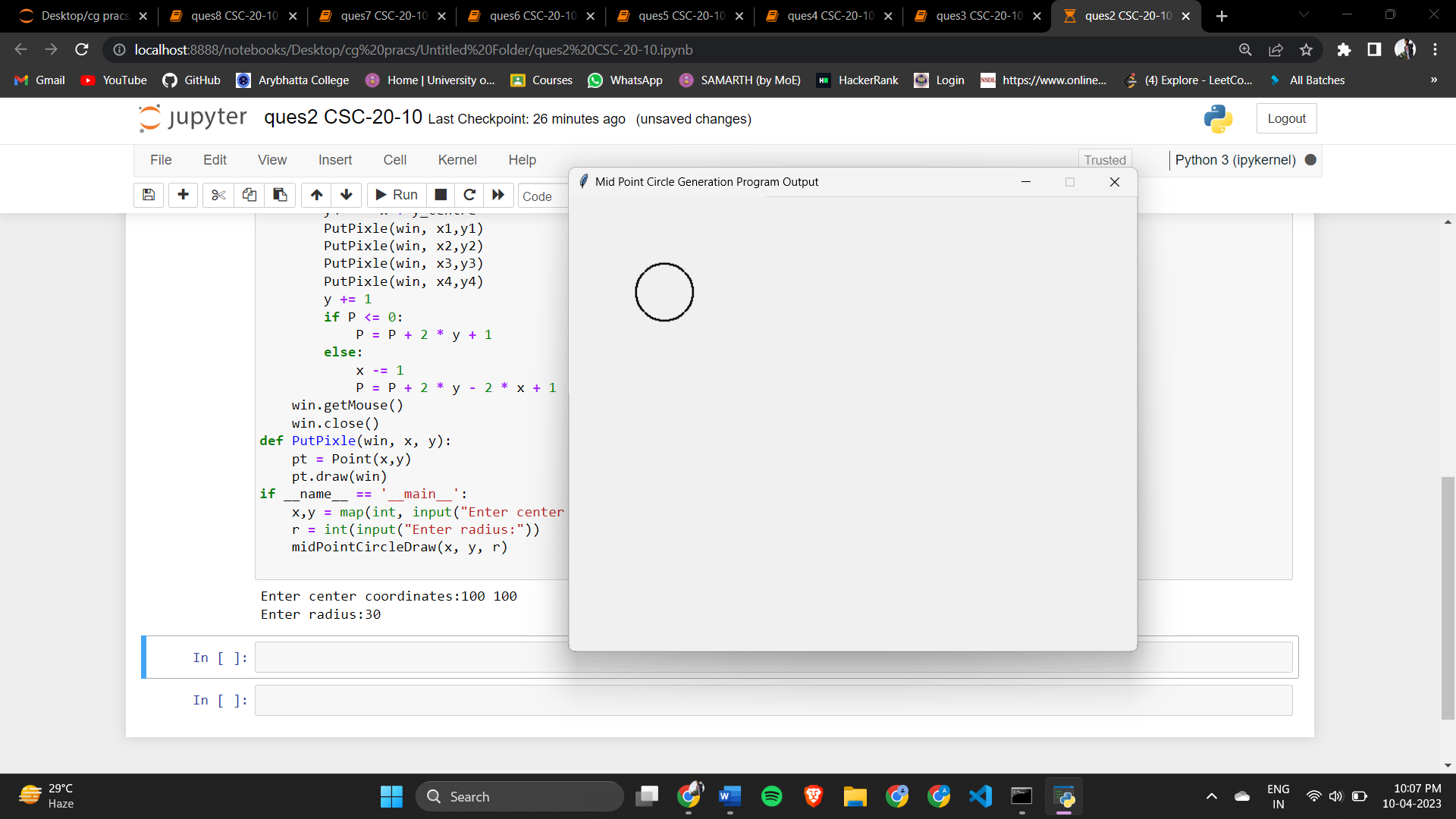


Figure 2. The output of Mid Point Circle Drawing algorithm.

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

## Introduction

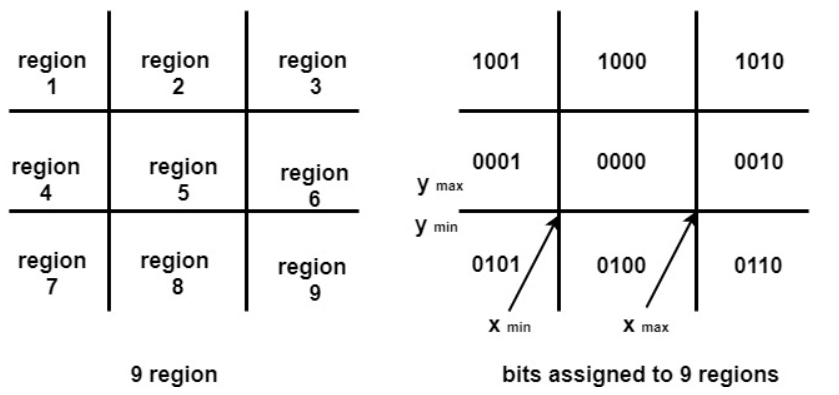
In the algorithm, first of all, it is detected whether line lies inside the screen or it is outside the screen. All lines come under any one of the following categories:

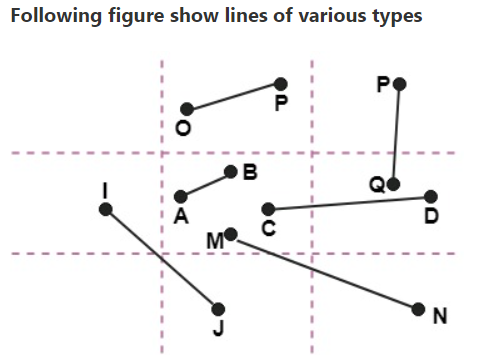
1. Visible
2. Not Visible
3. Clipping Case

**1. Visible:** If a line lies within the window, i.e., both endpoints of the line lies within the window. A line is visible and will be displayed as it is.

**2. Not Visible:** If a line lies outside the window it will be invisible and rejected. Such lines will not display. If any one of the following inequalities is satisfied, then the line is considered invisible.

**3. Clipping Case:** If the line is neither visible case nor invisible case. It is considered to be clipped case. First of all, the category of a line is found based on nine regions given below. All nine regions are assigned codes. Each code is of 4 bits. If both endpoints of the line have end bits zero, then the line is considered to be visible.



The center area is having the code, 0000, i.e., region 5 is considered a rectangle window. 

Line AB is the visible case  
Line OP is an invisible case  
Line PQ is an invisible line  
Line IJ are clipping candidates  
Line MN are clipping candidate  
Line CD are clipping candidate

## Code

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 by CSC/20/50')

plt.plot(xlist,ylist)

xmin,ymin=input("Enter Lower Bound Cordinates of Interaction Area :- ").split()

xmax,ymax=input("Enter Upper Bound Cordinates of Interaction Area :- ").split()

x1,y1=input("Enter Starting Cordinates of line :- ").split()

x2,y2=input("Enter Ending Cordinates of line :- ").split()

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

## Output

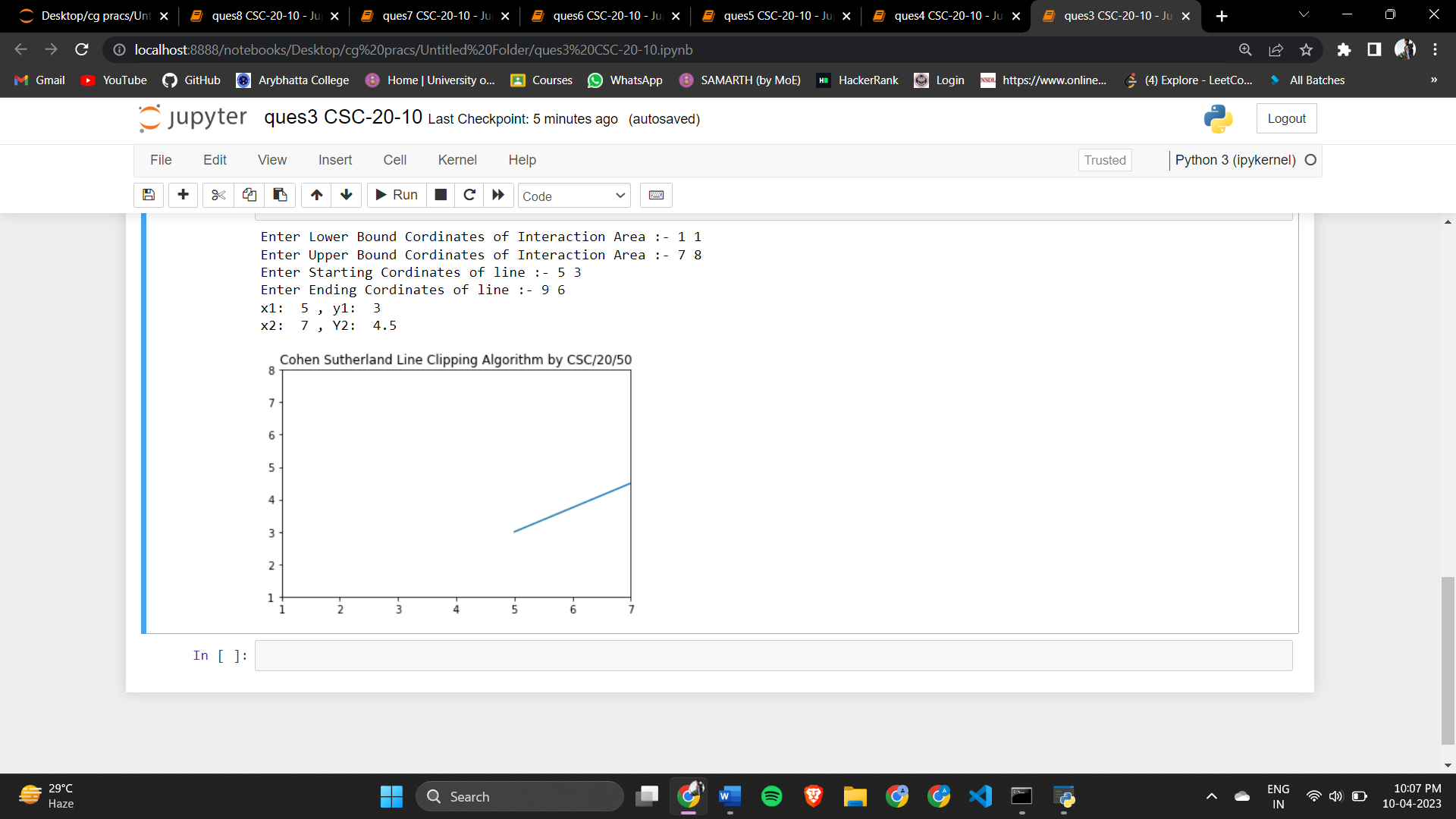


Figure 3. The output of Cohen and Sutherland Line Clipping Algorithms

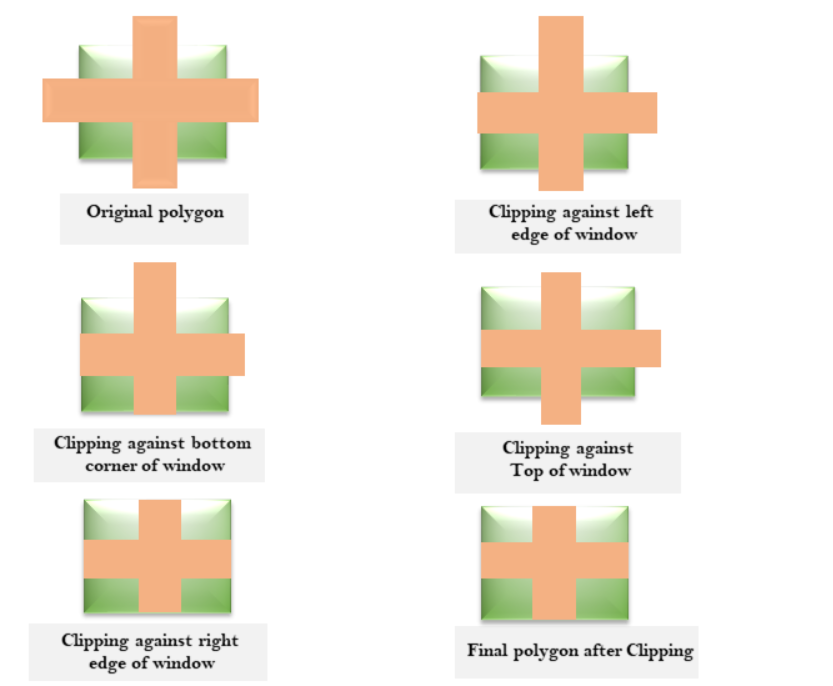
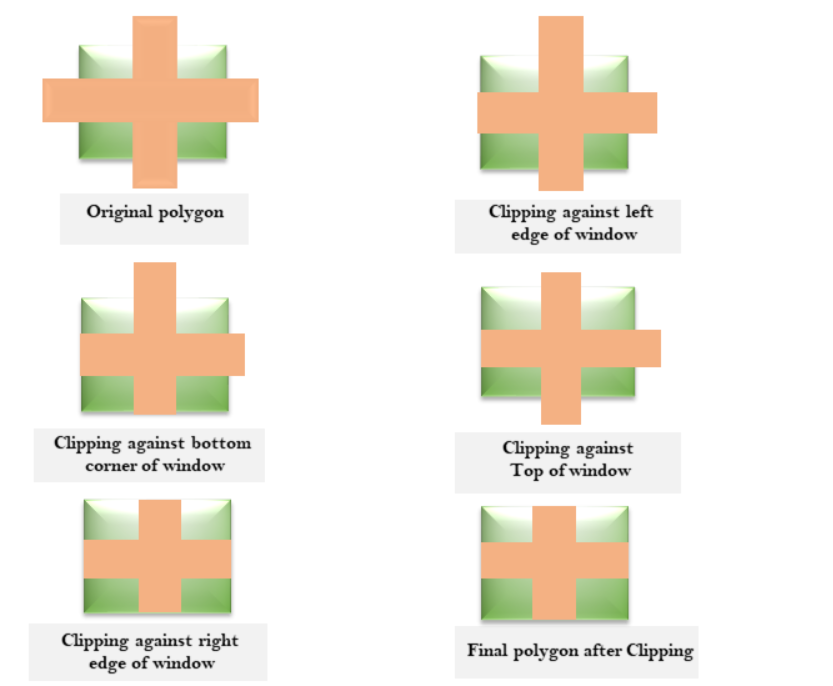
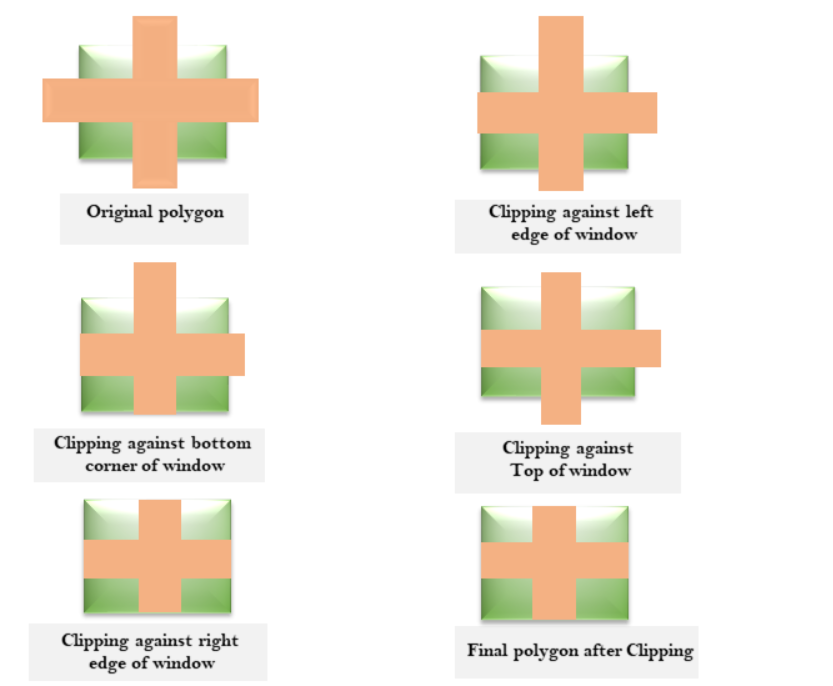
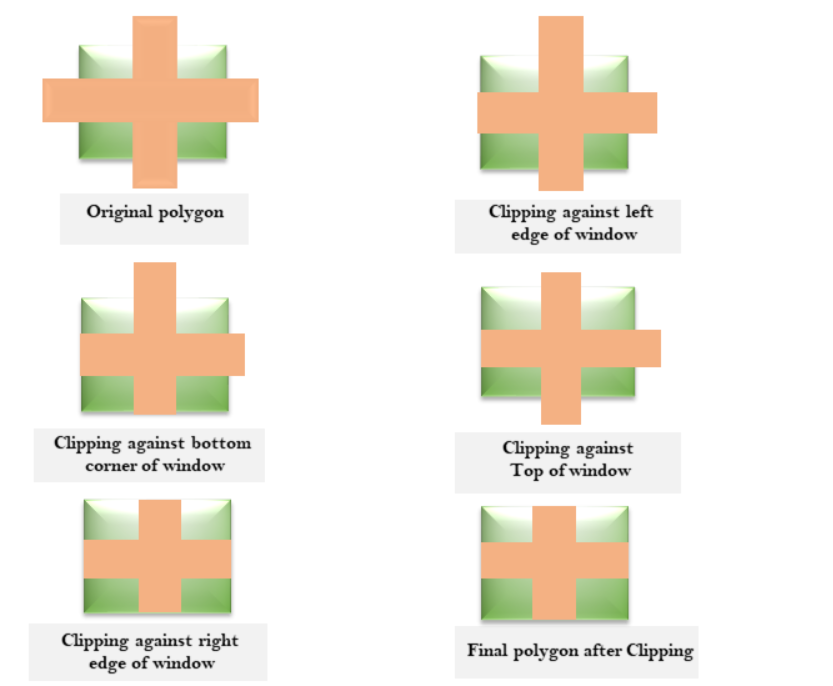
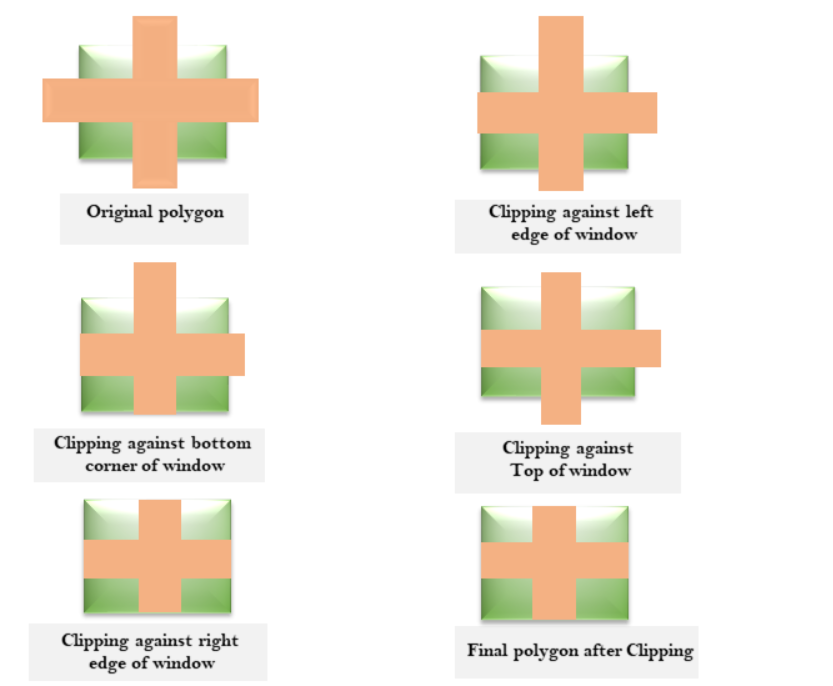
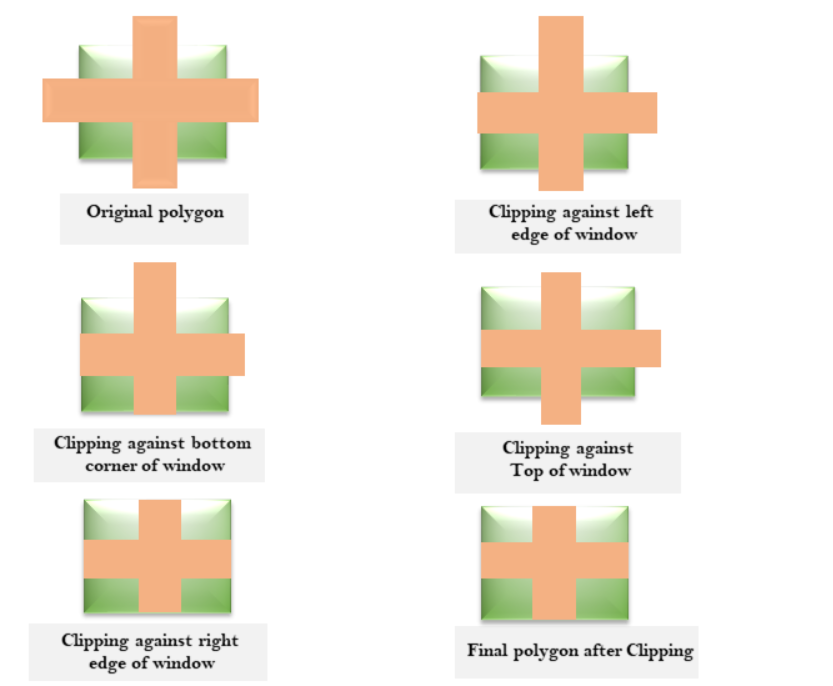
# Write a program to clip a polygon using Sutherland Hodgeman algorithm

## Introduction

It is performed by processing the boundary of polygon against each window corner or edge. First of all entire polygon is clipped against one edge, then resulting polygon is considered, then the polygon is considered against the second edge, so on for all four edges. **Four possible situations while processing**

1. If the first vertex is an outside the window, the second vertex is inside the window. Then second vertex is added to the output list. The point of intersection of window boundary and polygon side (edge) is also added to the output line.
2. If both vertexes are inside window boundary. Then only second vertex is added to the output list.
3. If the first vertex is inside the window and second is an outside window. The edge which intersects with window is added to output list.
4. If both vertices are the outside window, then nothing is added to output list.

Following figures shows original polygon and clipping of polygon against four windows



## Code

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

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

print("x2: ",C,", y2: ",D)

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\_hongeman(x1,y1,x2,y2,V):

check=[]

v1=[]

for i in V:

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

print(check)

print(V)

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)

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

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

draw\_graph(xlist,ylist,v1,"Cropped Data is :- ")

import matplotlib.pyplot as plt

from shapely.geometry import Polygon

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

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

xlist=[int(x1),int(x1),int(x2),int(x2),int(x1)]

ylist=[int(y1),int(y2),int(y2),int(y1),int(y1)]

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

V=[]

print("Start entering Points :- ")

for i in range(count):

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

x,y=input().split()

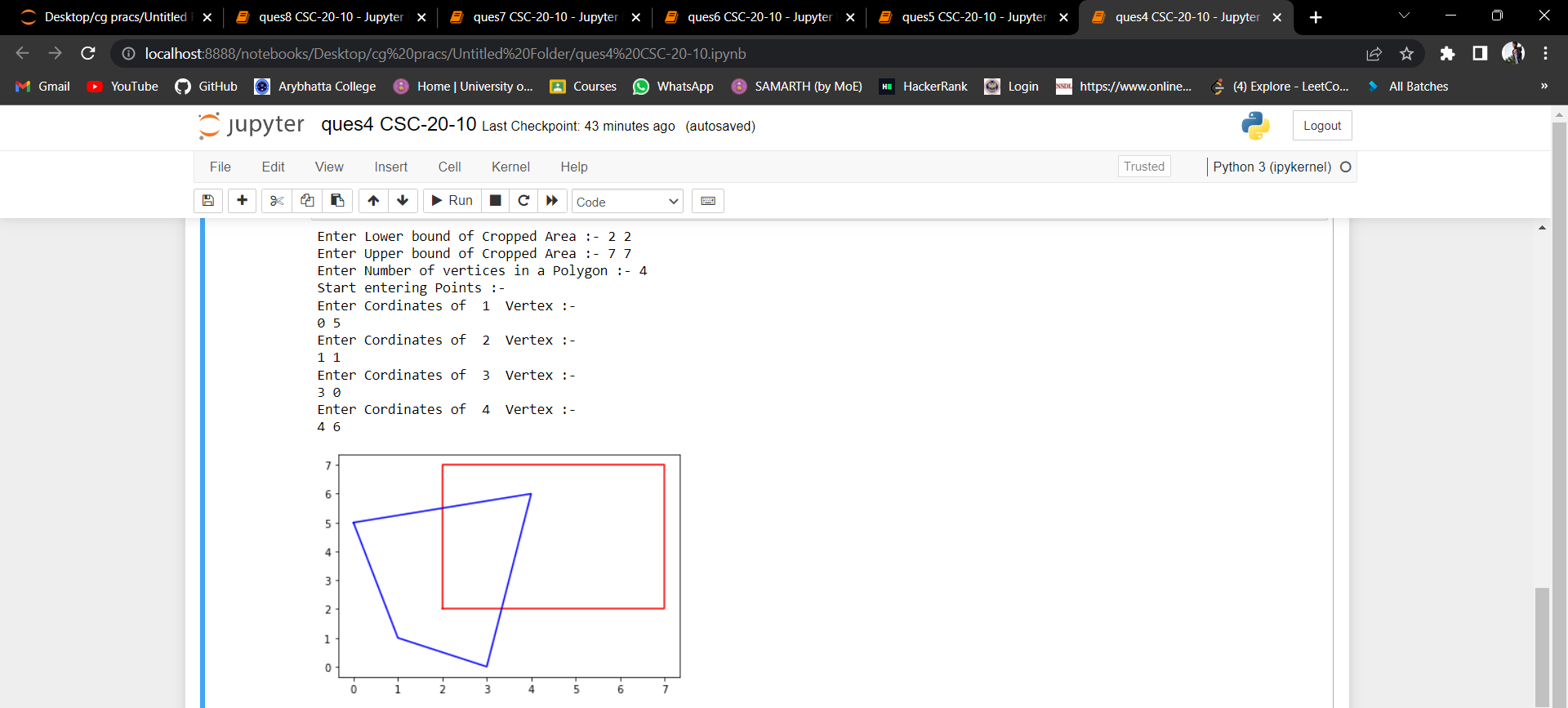
V.append((int(x),int(y)))

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

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

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

## Output



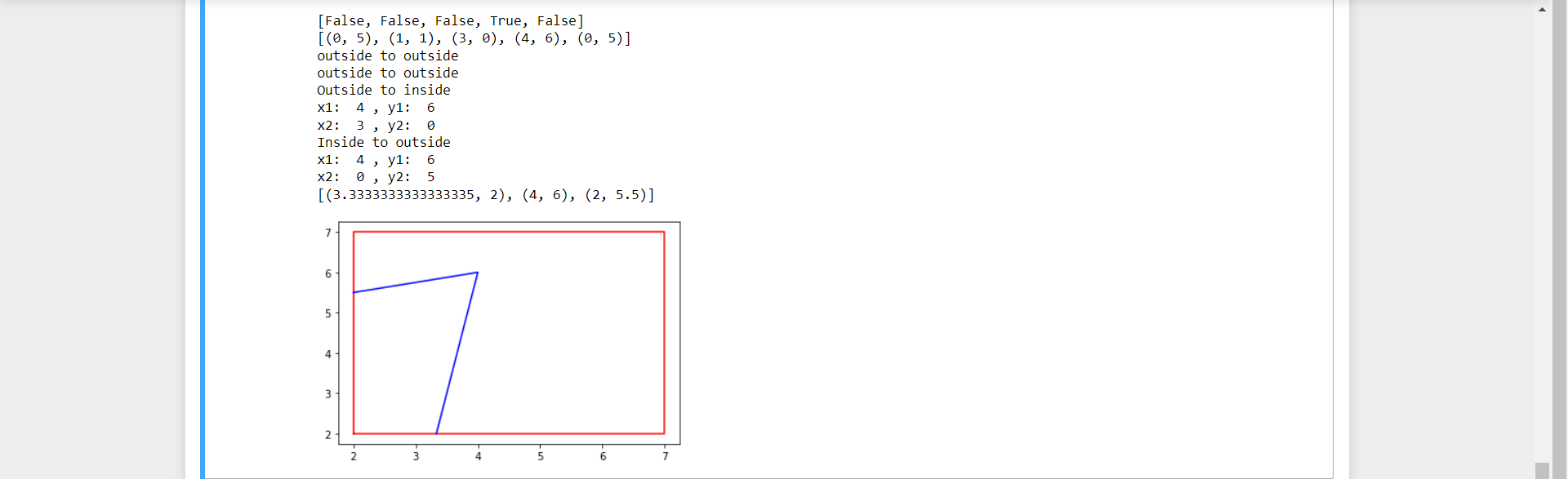


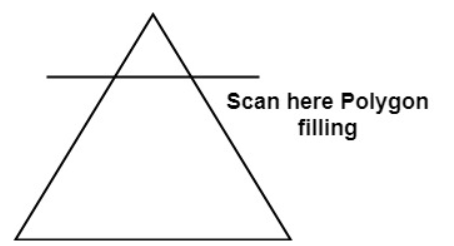
Figure 4. The output of Sutherland Hodgeman algorithm.

# Write a program to fill a polygon using scan line fill algorithm

## Introduction

This algorithm lines interior points of a polygon on the scan line and these points are done on or off according to requirement. The polygon is filled with various colors by coloring various pixels.

In above figure polygon and a line cutting polygon in shown. First of all, scanning is done. Scanning is done using raster scanning concept on display device. The beam starts scanning from the top left corner of the screen and goes toward the bottom right corner as the endpoint. The algorithms find points of intersection of the line with polygon while moving from left to right and top to bottom. The various points of intersection are stored in the frame buffer. The intensities of such points is keep high. Concept of coherence property is used. According to this property if a pixel is inside the polygon, then its next pixel will be inside the polygon.



## Code

def draw\_graph(V):

x=[]

y=[]

for i in V:

x.append(i[0])

y.append(i[1])

plt.title("Before Appling Scan line Algorithm")

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

plt.show()

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

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 Appling 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

import matplotlib.pyplot as plt

from graphics import \*

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

V=[]

print("Start entering Points :- ")

for i in range(count):

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

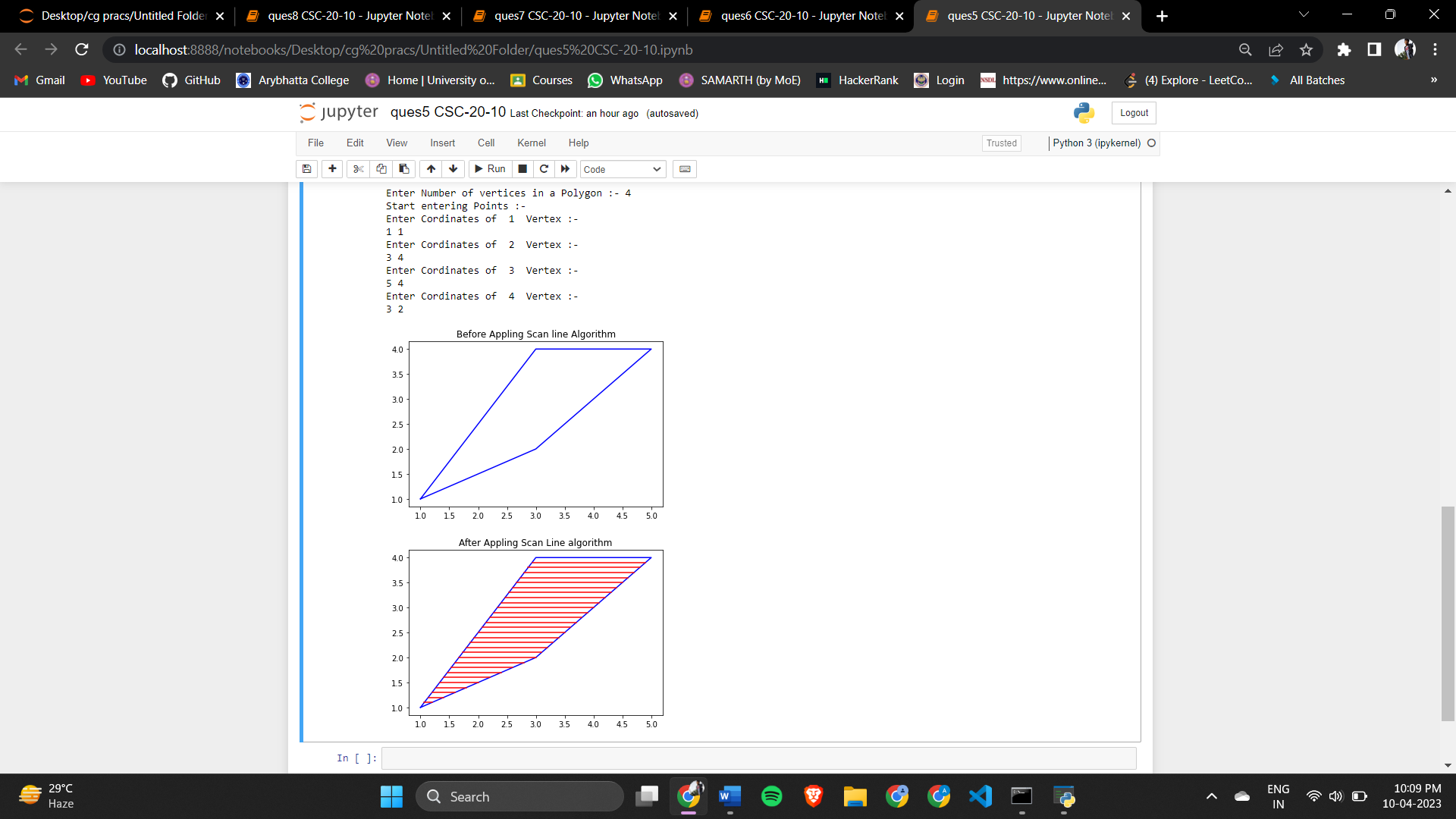
x,y=input().split()

V.append((int(x),int(y)))

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

get\_range(V)

## Output



**Figure** 5. The output of Scan line Fill algorithm.

# Write a program to apply various 2D transformations on a 2D object (use homogenous coordinates).

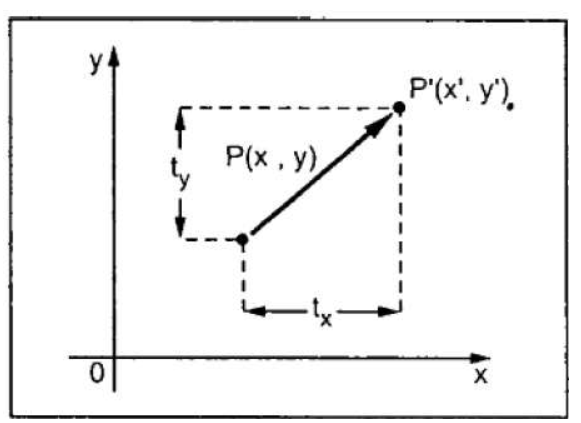
## Introduction

Transformation means changing some graphics into something else by applying rules. We can have various types of transformations such as translation, scaling up or down, rotation, etc. When a transformation takes place on a 2D plane, it is called 2D transformation.

Transformations play an important role in computer graphics to reposition the graphics on the screen and change their size or orientation.

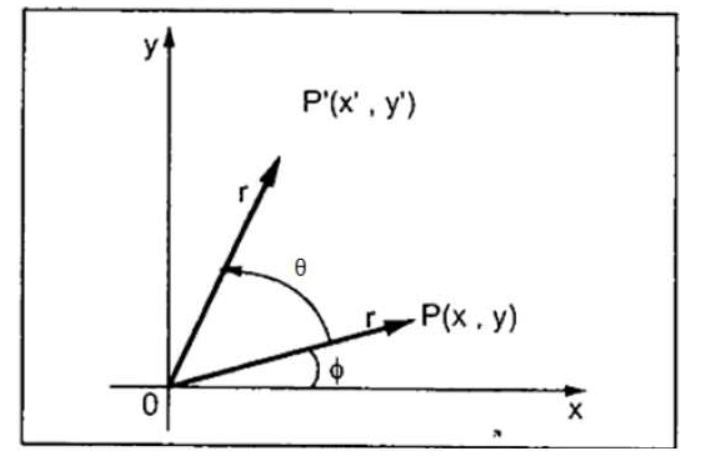
**Translation**

A translation moves an object to a different position on the screen. You can translate a point in 2D by adding translation coordinate (tx,ty) to the original coordinate X,Y to get the new coordinate X′,Y′.



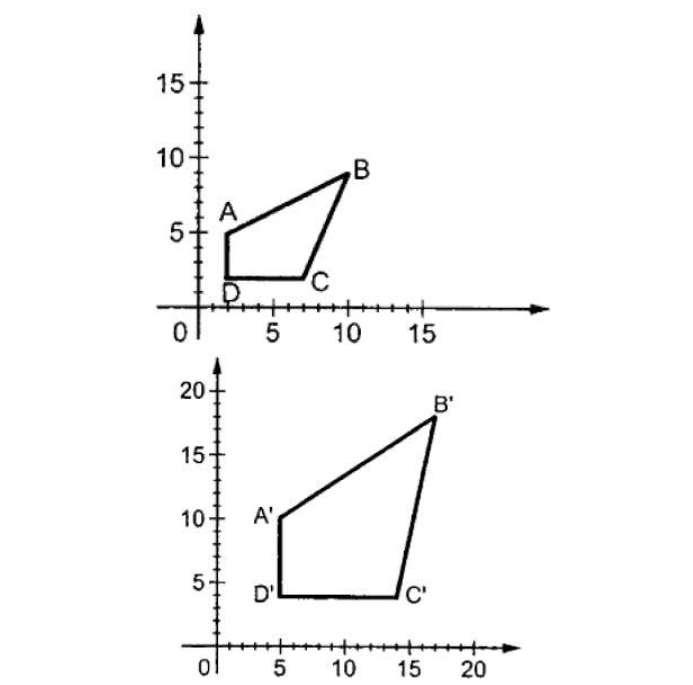
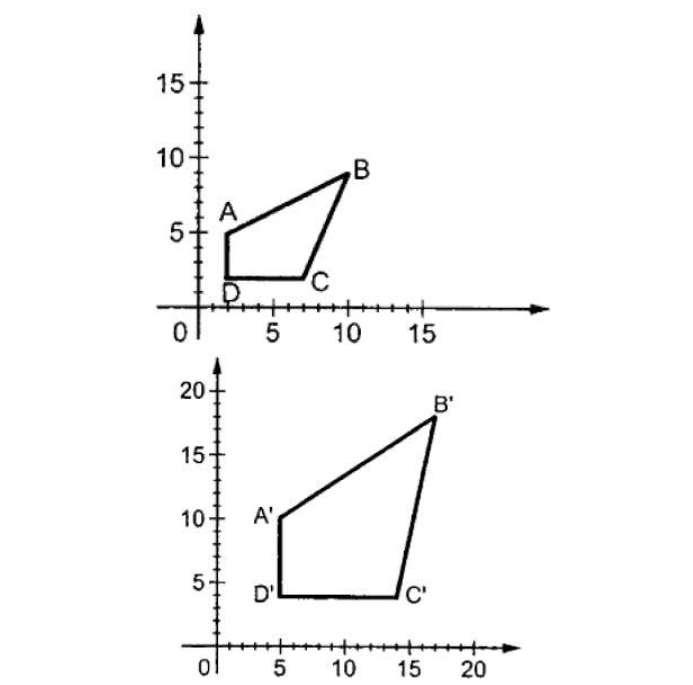
**Rotation**

In rotation, we rotate the object at particular angle θ  from its origin. From the following figure, we can see that the point P(X,Y) is located at angle φ from the horizontal X coordinate with distance r from the origin.Let us suppose you want to rotate it at the angle θ. After rotating it to a new location, you will get a new point P’ (X′,Y′).



**Scaling**

To change the size of an object, scaling transformation is used. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result.



## Code

import matplotlib.pyplot as plt

import numpy as np

import math

def draw\_graph(V,title):

x=[]

y=[]

for i in range(len(V)):

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

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

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)

x=np.dot(O[0],C)

y=np.dot(O[1],C)

return [x,y]

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 i in V:

print(i)

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

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 i in V:

print(i)

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

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 i in V:

print(i)

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

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

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

V=[]

for i in range(count):

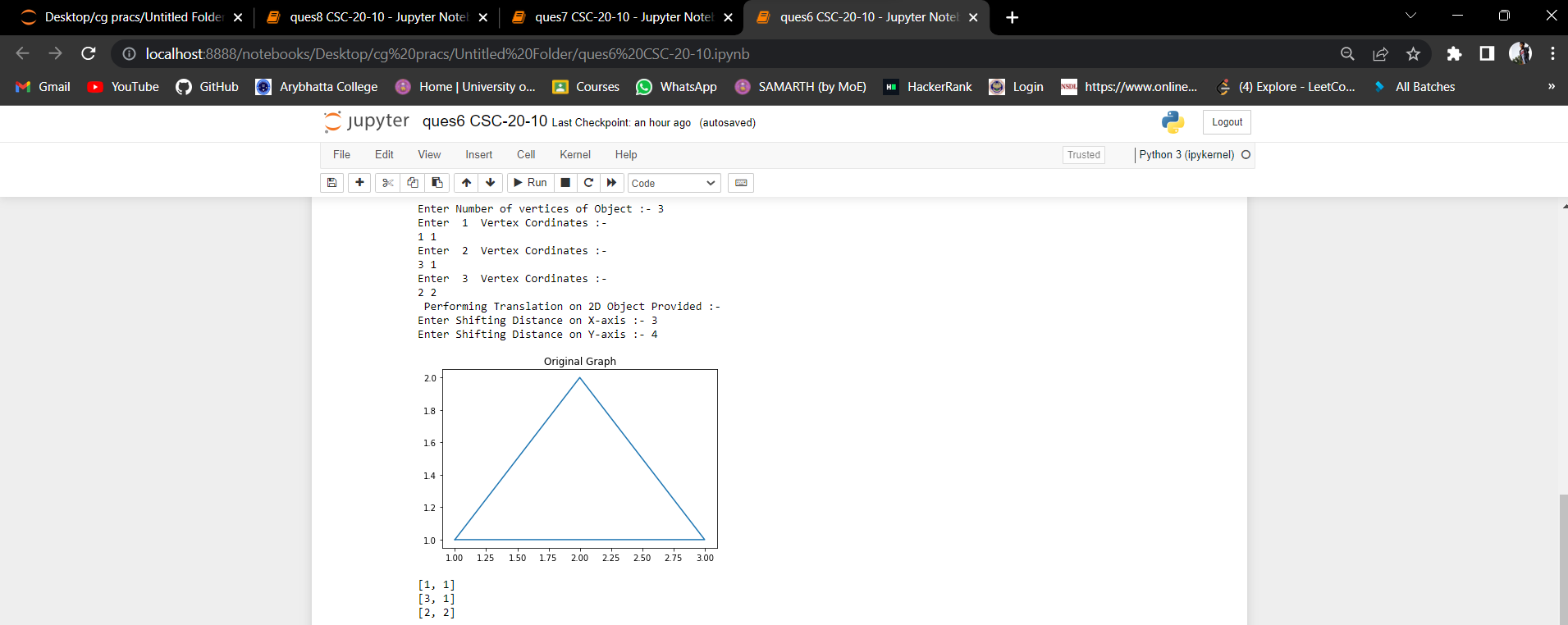
print("Enter ",i+1," Vertex Cordinates :- ")

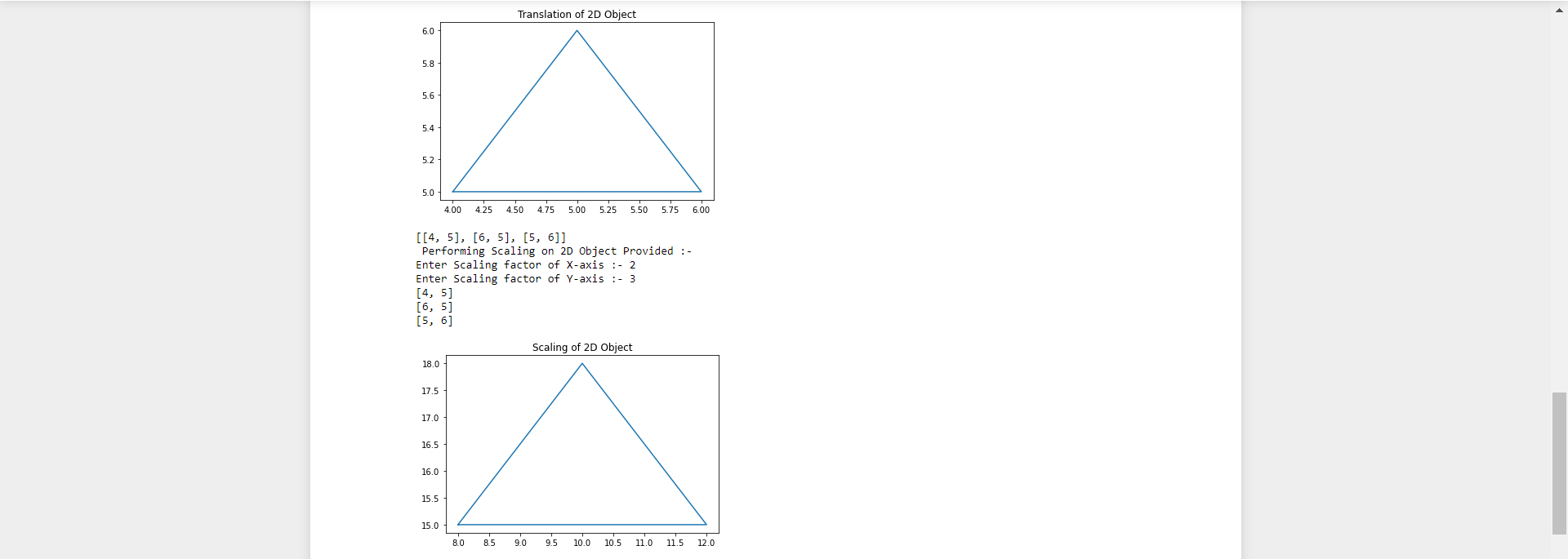
x,y=input().split()

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

Translation(V)

## Output





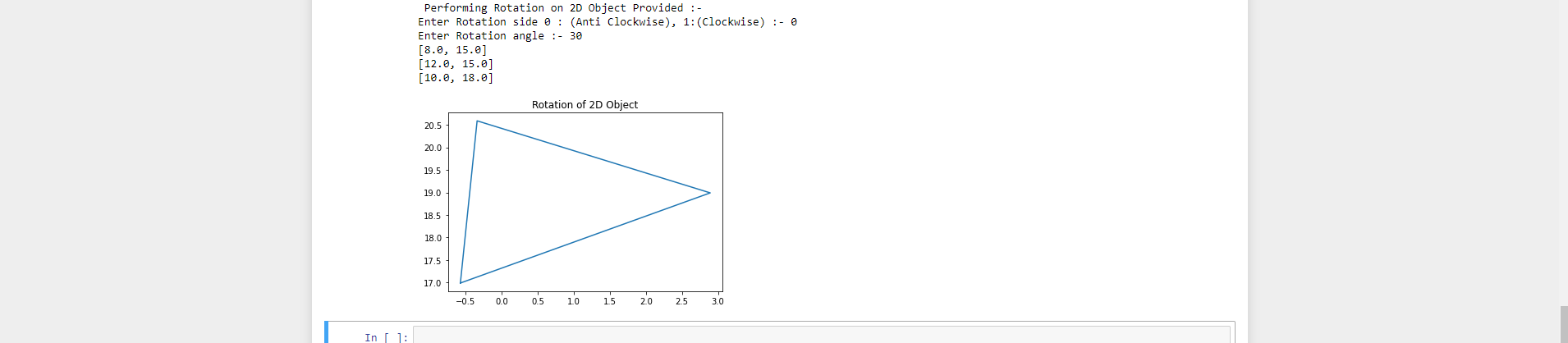


Figure 6. The output of 2D Transformation on a 2D Object.

# Write a program to apply Various 3D transformations on a 3D Object.

## Introduction

3-D Transformation: In very general terms a 3D model is a mathematical representation of a physical entity that occupies space. In more practical terms, a 3D model is made of a description of its shape and a description of its color appearance.3-D Transformation is the process of manipulating the view of a three-D object with respect to its original position by modifying its physical attributes through various methods of transformation like Translation, Scaling, Rotation, Shear, etc.

Properties of 3-D Transformation:

1. Lines are preserved,
2. Parallelism is preserved,
3. Proportional distances are preserved.

One main categorization of a 3D object’s representation can be done by considering whether the surface or the volume of the object is represented:

Boundary-based: the surface of the 3D object is represented. This representation is also called b-rep. Polygon meshes, implicit surfaces, and parametric surfaces, which we will describe in the following, are common representations of this type.

Volume-based: the volume of the 3D object is represented. Voxels and Constructive Solid Geometry (CSG) Are commonly used to represent volumetric data.

Types of Transformations:

1. Translation
2. Scaling
3. Rotation

## Code

import matplotlib.pyplot as plt

import numpy as np

import math

def multiply\_matrix(O,C):

C.append(1)

x=np.dot(O[0],C)

y=np.dot(O[1],C)

z=np.dot(O[2],C)

return [x,y,z]

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 i in V:

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

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 i in V:

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

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 :- "))

Axis=input("Enter Rotation's Axis :- ")

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 i in V:

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

print(V1)

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

V=[]

for i in range(count):

print("Enter ",i+1," Vertex Cordinates :- ")

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

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

Translation(V)

## Output

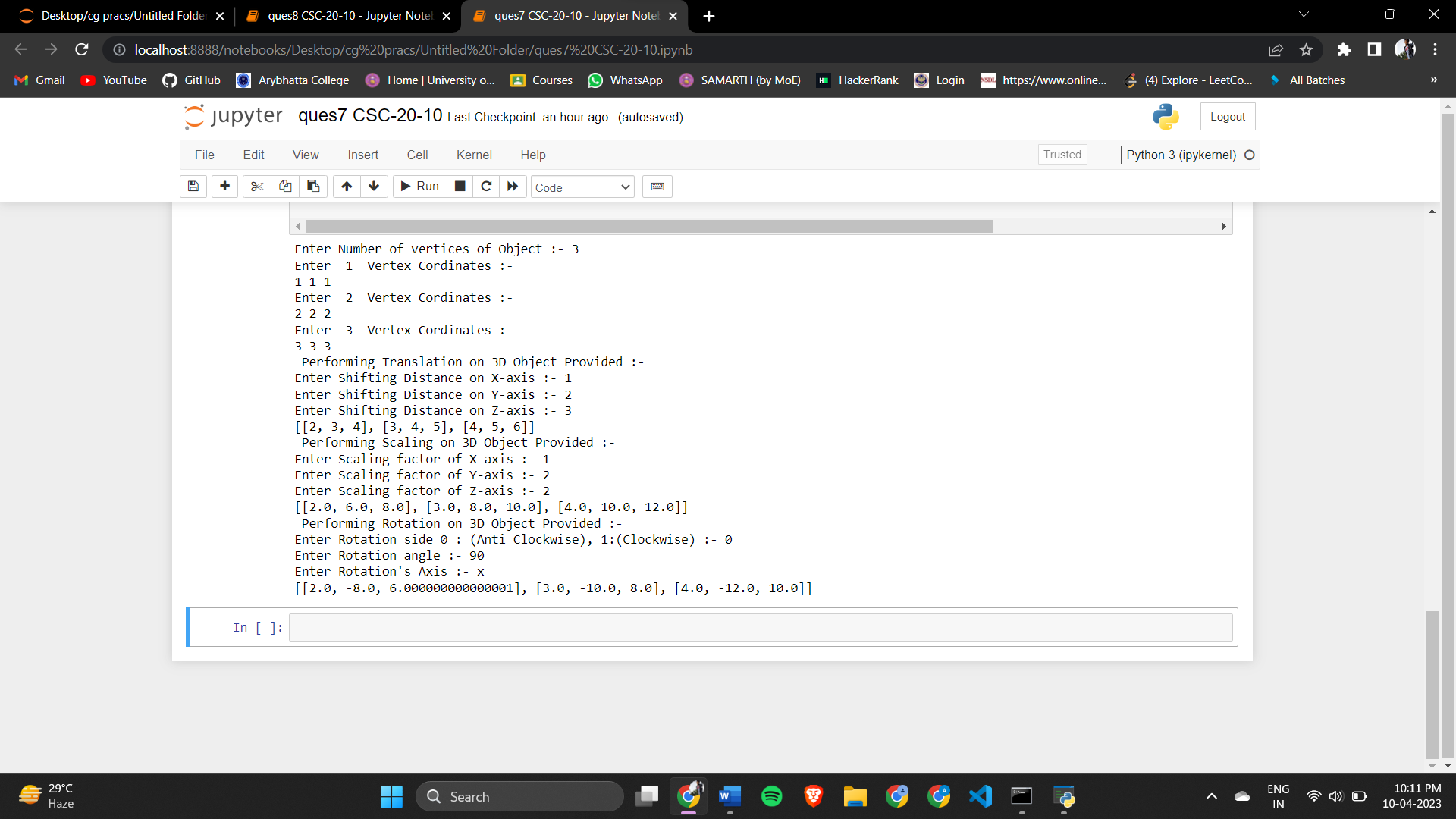


Figure 7. The output of 3D transformation on a 3D Object.

# Write a program to draw Hermite/Bezier Curve.

## Introduction

Bezier curves are one of the parametric curves most frequently used in computer graphics and were independently developed for computer-assisted car design by two engineers, both working for French automobile companies: Pierre Bezier who was an engineer for Renault, and Paul de Casteljau, who was an engineer for Citroën.

The concept of Bezier curves was discovered by the French engineer Pierre

Bézier.Bezier curves are basically used in computer graphics to draw shapes, for CSS animation, and in many other places.

These are the curves that are generated under the control of some other points, also called control points.

Properties of Bezier Curves:

A very important property of Bezier curves is that they always pass through the first and last control points.

The degree of the polynomial defining the curve segment is always one less than the number of defining polygon points. So, for example, if we have 4 control points, then the degree of the polynomial is 3, i.e., cubic polynomial.

In the Bezier curves, moving a control point alters the shape of the whole curve.

A Bezier curve generally follows the shape of the defining polygon.

A curve is always inside the convex hull of control points.

## Code

from wand.image import Image

from wand.drawing import Drawing

from wand.color import Color

with Drawing() as draw:

draw.stroke\_color = Color('black')

draw.stroke\_width = 1

x=int(input("Enter Number of Control Points :- "))

print("OK Plz Fill The Details :- ")

points=[]

for i in range(x):

print("Enter ",i+1,"Control Point Cordinates :- ")

x,y=input().split()

points.append((int(x),int(y)))

draw.fill\_color = Color('red')

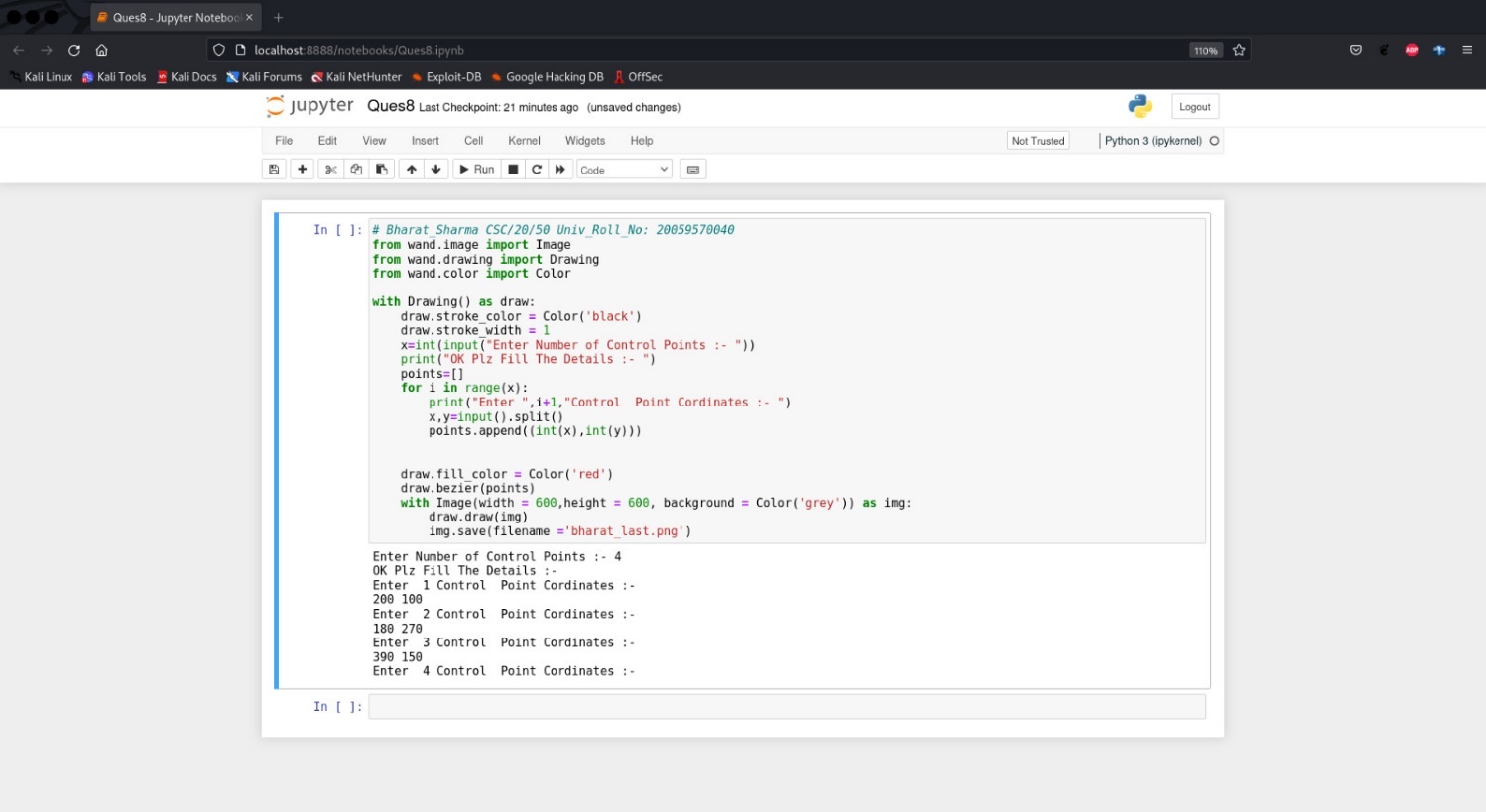
draw.bezier(points)

with Image(width = 600,height = 600, background = Color('grey')) as img:

draw.draw(img)

img.save(filename ='aadi\_last.png')

## Output



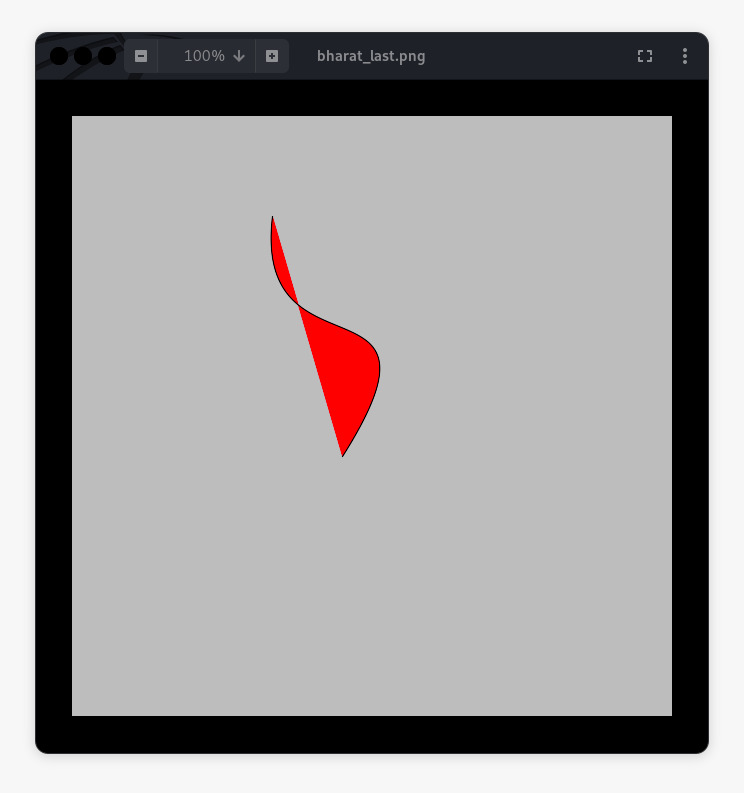


Figure 8. The output of Bezier Curve.