Computer Graphics

Practical File

TANMAY JOSHI
Roll No.:- CSC/19/79
University Roll No.:- 19059570045
BSc.(H) Computer Science
Sem-VI

Submitted to: Prof. Raj K Sharma,
Department of Computer Science

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

Certified that this Practical File report of Computer is the bonafide work of "Tanmay Joshi" who carried out the project work under my supervision. This is to further certify to the best of our knowledge, that this project has not been carried out earlier in this College.

SIGNATURE

Raj K Sharma

ACKNOWLEDGEMENT

We wish to express our profound and sincere gratitude to Raj K Sharma Sir, Computer Science Department, Aryabhatta College, Benito Juarez Road who guided us into the intricacies of this practical file.

I am indebted to my family for their constant encouragement, co-operation and help. Words of gratitude are not enough to describe the accommodation and fortitude which they have shown throughout my endeavour.

DECLARATION

I hereby declare that this submission is my own work and that, to the best of our knowledge and belief, it contains neither material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any exam, except where due acknowledgment has been made in the text.

Date: 20th Apr, 2022

TANMAY JOSHI

College Roll No.:-CSC/19/79

University Roll No.:-19059570045

Index

PROGRAM-1

Write a program to implement DDA and Bresenham's Line drawing Algorithms

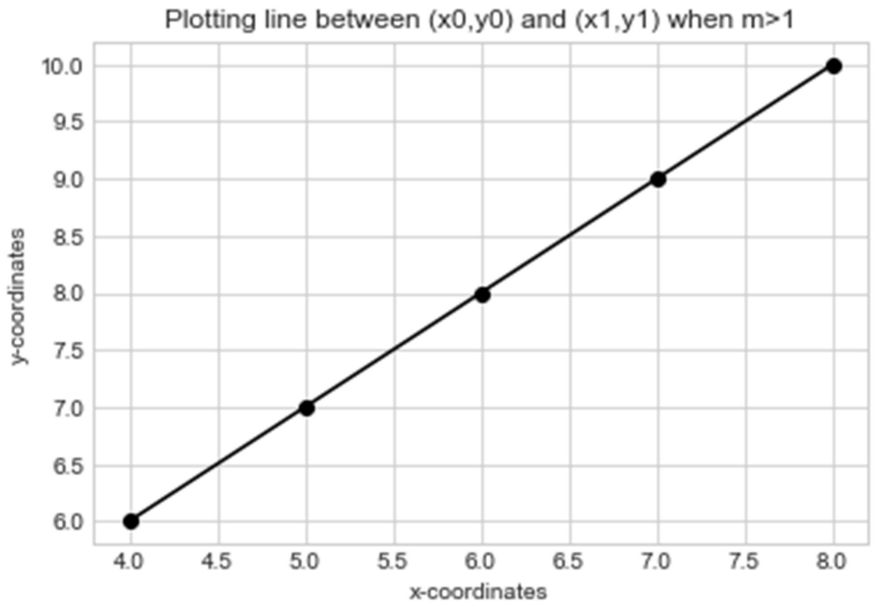
```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
x0 = int(input("x0: "))
y0 = int(input("y0: "))
x1 = int(input("x1: "))
y1 = int(input("y1: "))
m = (y1-y0)/(x1-x0)
print('m=',m)
```

```
x0: 4
y0: 6
x1: 8
y1: 10
m= 1.0
```

```
if m<=1:
    x_i = list()
    y_i = list()
    x_i.append(x0)
    y_i.append(y0)
    for i in range(x0+1, x1+1):
        x_i.append(i)
        num = y_i[-1] + m
        y_i.append(num)
    y_f = list()
    for i in y_i:
        i=round(i,2)
        y_f.append(int(i//1))
    print("Line for (x0,y0) and (x1,y1) passes through the following points: ")
    for i in range (len(x_i)):
        print('('+str(x_i[i])+','+str(y_f[i])+')')
    plt.plot(x_i, y_f, '-ok');
    plt.xlabel("x-coordinates")
    plt.ylabel("y-coordinates")
    plt.title("Plotting line between (x0,y0) and (x1,y1) when m<=1")
    plt.show()
else:
    x_i = list()
    y_i = list()
    x_i.append(x0)
    y_i.append(y0)
    for i in range(y0+1, y1+1):
        y_i.append(i)
        num = x_i[-1] + 1/m
        x_i.append(num)
    x_f = list()
    for i in x_i:
        i=round(i,2)
        x_f.append(int(i//1))
```

```
    print("Line for (x0,y0) and (x1,y1) passes through the following points: ")
    for i in range (len(y_i)):
        print('('+str(x_f[i])+','+str(y_i[i])+')')
    plt.plot(x_f, y_i, '-ok');
    plt.xlabel("x-coordinates")
    plt.ylabel("y-coordinates")
    plt.title("Plotting line between (x0,y0) and (x1,y1) when m>1")
    plt.show()
```

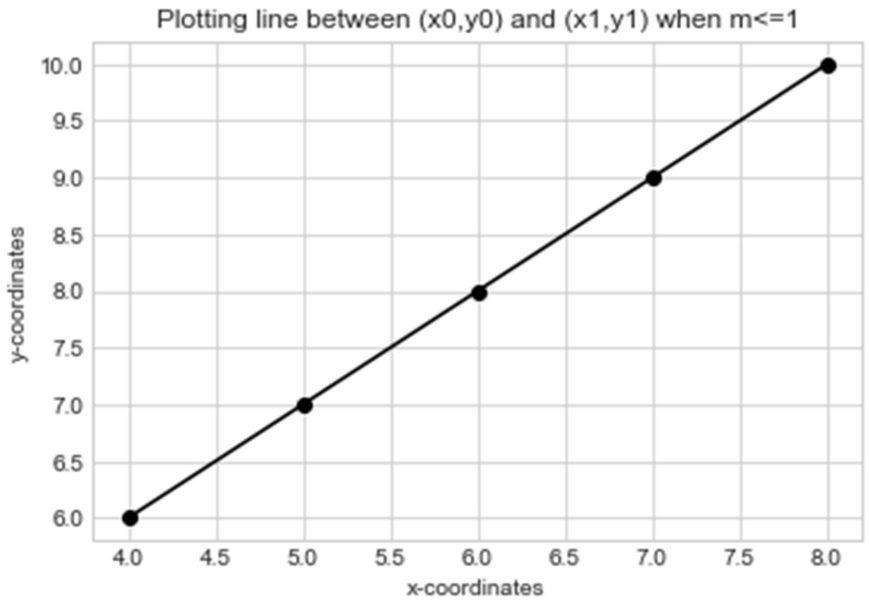
```
Line for (x0,y0) and (x1,y1) passes through the following points:
(4,6)
(5,7)
(6,8)
(7,9)
(8,10)
```




```
#Bresenham line drawing for same points
dx=x1-x0
dy=y1-y0
x=list()
y=list()
x.append(x0)
y.append(y0)
p=list()
p.append(2*dy-dx)
for i in range(x0+1, x1+1):
        x.append(i)
        if p[-1]>=0:
            y.append(y[-1]+1)
            p.append(p[-1]+2*(dy-dx))
```

```
        else:
            y.append(y[-1])
            p.append(p[-1]+2*dy)
print("Line for (x0,y0) and (x1,y1) passes through the following points: ")
for i in range (len(y_i)):
    print('('+str(x[i])+','+str(y[i])+')')
plt.plot(x, y, '-ok');
plt.xlabel("x-coordinates")
plt.ylabel("y-coordinates")
plt.title("Plotting line between (x0,y0) and (x1,y1) when m>1")
plt.show()
```

```
Line for (x0,y0) and (x1,y1) passes through the following points:
(4,6)
(5,7)
(6,8)
(7,9)
(8,10)
```



In [ ]:

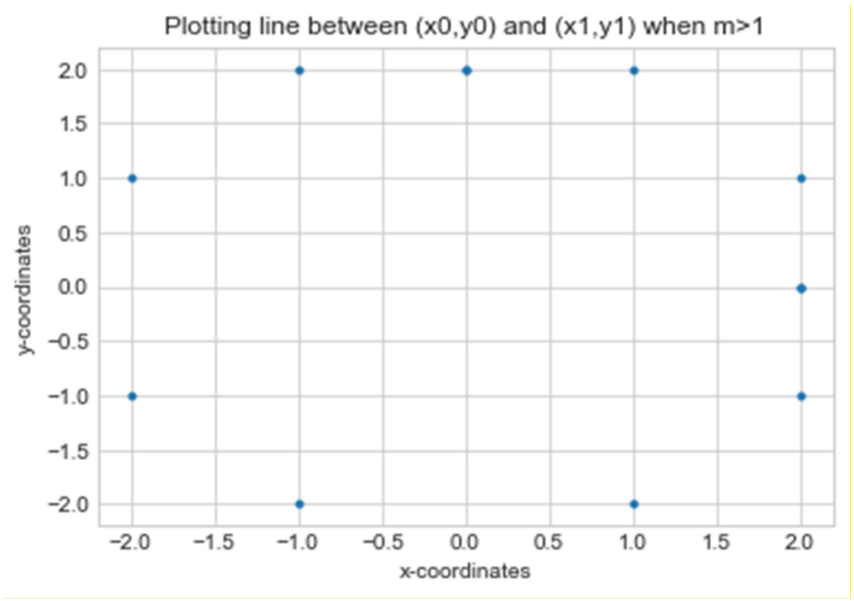
Program 2

Write a program to implement a midpoint circle drawing algorithm.

```
import matplotlib.pyplot as plt
def MidPoint(r):
    xl=list()
    yl=list()
    x = r
    y = 0
    xl.append(x)
    yl.append(y)
    print("(", x, ", ",y, ")",sep = "", end = "")
    if (r > 0) :
        xl.append(x)
        yl.append(-y)
        xl.append(y)
        yl.append(x)
        xl.append(-y)
        yl.append(x)
        print("(", x, ", ",-y, ")",sep = "", end = "")
        print("(", y, ", ",x, ")",sep = "", end = "")
        print("(", -y, ", ",x, ")", sep = "")
    P = 1 - r
    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
        xl.append(x)
        yl.append(y)
        xl.append(-x)
        yl.append(y)
        xl.append(x)
        yl.append(-y)
        xl.append(-x)
        yl.append(-y)
        print("(", x, ", ", y,")", sep = "", end = "")
        print("(", -x, ", ", y,")", sep = "", end = "")
        print("(", x, ", ", -y,")", sep = "", end = "")
        print("(", -x, ", ", -y,")", sep = "")
        if x != y:
            xl.append(y)
            yl.append(x)
            xl.append(-y)
            yl.append(x)
            xl.append(y)
            yl.append(-x)
            xl.append(-y)
            yl.append(-x)
```

```
            print("(", y, ", ", x,")", sep = "", end = "")
            print("(", -y, ", ", x,")", sep = "", end = "")
            print("(", y, ", ", -x,")", sep = "", end = "")
            print("(", -y, ", ", -x,")", sep = "")
            plt.style.use('seaborn-whitegrid')
            plt.plot(xl, yl,'.');
            plt.xlabel("x-coordinates")
            plt.ylabel("y-coordinates")
            plt.title("Plotting line between (x0,y0) and (x1,y1) when m>1")
            plt.show()
radius=int(input("Enter the Radius:-"))
MidPoint(radius)
```

```
Enter the Radius:-2
(2, 0)(2, 0)(0, 2)(0, 2)
(2, 1)(-2, 1)(2, -1)(-2, -1)
(1, 2)(-1, 2)(1, -2)(-1, -2)
```



Program 3

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

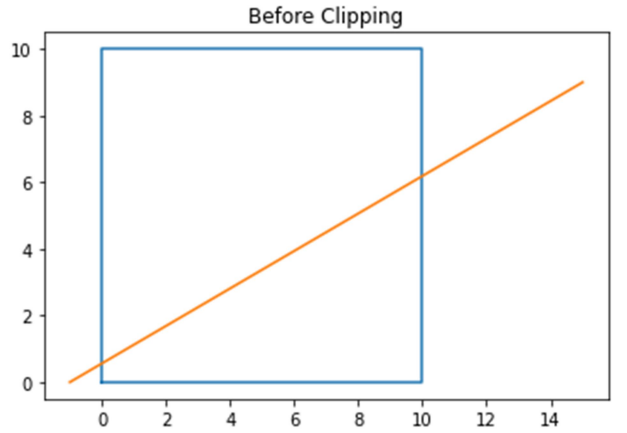
```
INSIDE = 0 #(Using ABRL)
LEFT = 1
RIGHT = 2
BOTTOM = 4
TOP = 8
def RegionCode(x, y):
   rc = INSIDE
   if x < x_min:
       rc |= LEFT
   elif x > x_max:
       rc |= RIGHT
   if y < y_min:
       rc |= BOTTOM
   elif y > y_max:
       rc |= TOP
   return rc

def cohenSutherlandClip(x1, y1, x2, y2):
    code1 = RegionCode(x1, y1)
    code2 = RegionCode(x2, y2)
    accept = False
    while True:

       if code1 == 0 and code2 == 0:
           accept = True
           break

       elif (code1 & code2) != 0:
           break

       else:

           x = 1.0
           y = 1.0
           if code1 != 0:
               code_out = code1
           else:
               code_out = code2

           if code_out & TOP:

               x = x1 + (x2 - x1) * (y_max - y1) / (y2 - y1)
               y = y_max
           elif code_out & BOTTOM:

               x = x1 + (x2 - x1) * (y_min - y1) / (y2 - y1)
               y = y_min
           elif code_out & RIGHT:
```

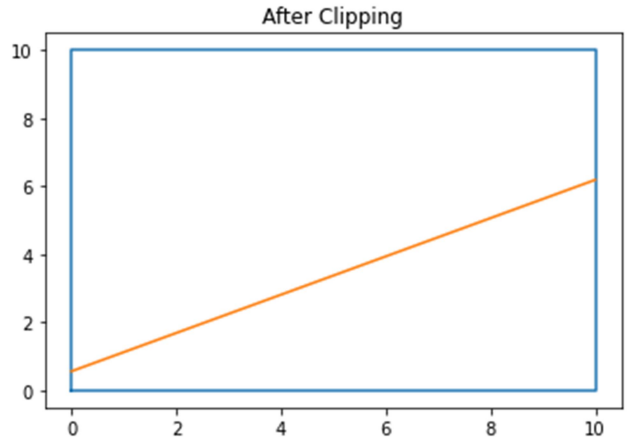
```
                y = y1 + (y2 - y1) * (x_max - x1) / (x2 - x1)
                x = x_max
            elif code_out & LEFT:

                y = y1 + (y2 - y1) * (x_min - x1) / (x2 - x1)
                x = x_min

            if code_out == code1:
                x1 = x
                y1 = y
                code1 = RegionCode(x1, y1)
            else:
                x2 = x
                y2 = y
                code2 = RegionCode(x2, y2)
    if accept:
       return [x1, y1, x2, y2]

    else:
       print("Line rejected")
```

In [11]:

```
import matplotlib.pyplot as plt
x_max = int(input("Enter the X_max for rectangular window:-"))
y_max = int(input("Enter the Y_max for rectangular window:-"))
x_min = int(input("Enter the X_min for rectangular window:-"))
y_min = int(input("Enter the Y_min for rectangular window:-"))
print("Enter the points od the Line to be Clipped:-")
x0 = int(input("x0: "))
y0 = int(input("y0: "))
x1 = int(input("x1: "))
y1 = int(input("y1: "))
x=list()
y=list()
x.append(x_min)
y.append(y_min)
x.append(x_min)
y.append(y_max)
x.append(x_max)
y.append(y_max)
x.append(x_max)
y.append(y_min)
x.append(x_min)
y.append(y_min)
lx=list()
ly=list()
lx.append(x0)
ly.append(y0)
lx.append(x1)
ly.append(y1)
plt.plot(x,y,lx,ly)
plt.title('Before Clipping')
plt.show()
```

```
cx=[cohenSutherlandClip(x0,y0,x1,y1)[0],cohenSutherlandClip(x0,y0,x1,y1)[2]]
cy=[cohenSutherlandClip(x0,y0,x1,y1)[1],cohenSutherlandClip(x0,y0,x1,y1)[3]]
plt.plot(x,y,cx,cy)
plt.title('After Clipping')
plt.show()
Enter the X_max for rectangular window:-10
Enter the Y_max for rectangular window:-10
Enter the X_min for rectangular window:-0
Enter the Y_min for rectangular window:-0
Enter the points od the Line to be Clipped:-

x0: -1
y0: 0
x1: 15
y1: 9
```





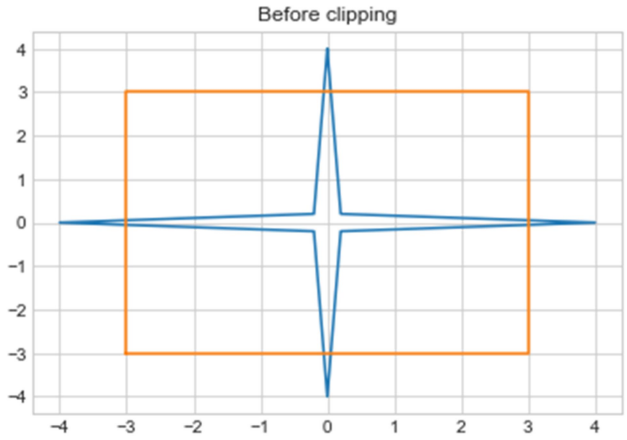
Program 4

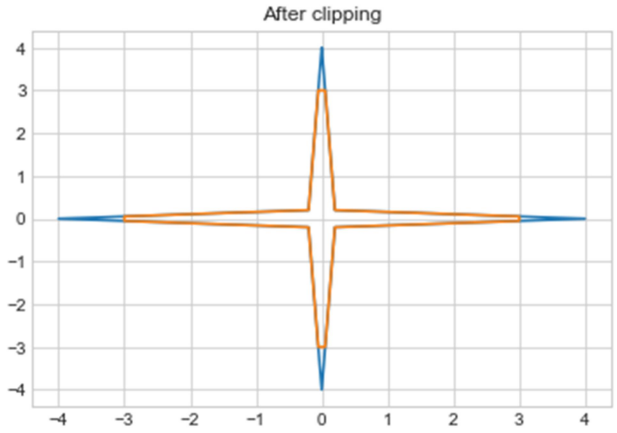
Write a program to clip a polygon using Sutherland Hodgeman algorithm.

```
import numpy as np
import matplotlib.pyplot as plt
import warnings
plt.style.use('seaborn-whitegrid')

class PolygonClipper:

    def __init__(self,warn_if_empty=True):
        self.warn_if_empty = warn_if_empty

    def is_inside(self,p1,p2,q):
        R = (p2[0] - p1[0]) * (q[1] - p1[1]) - (p2[1] - p1[1]) * (q[0] - p1[0])
        if R <= 0:
            return True
        else:
            return False

    def compute_intersection(self,p1,p2,p3,p4):
        if p2[0] - p1[0] == 0:
            x = p1[0]
            m2 = (p4[1] - p3[1]) / (p4[0] - p3[0])
            b2 = p3[1] - m2 * p3[0]
            y = m2 * x + b2

        elif p4[0] - p3[0] == 0:
            x = p3[0]

            m1 = (p2[1] - p1[1]) / (p2[0] - p1[0])
            b1 = p1[1] - m1 * p1[0]

            y = m1 * x + b1

        else:
            m1 = (p2[1] - p1[1]) / (p2[0] - p1[0])
            b1 = p1[1] - m1 * p1[0]

            m2 = (p4[1] - p3[1]) / (p4[0] - p3[0])
            b2 = p3[1] - m2 * p3[0]

            # x-coordinate of intersection
            x = (b2 - b1) / (m1 - m2)

            # y-coordinate of intersection
            y = m1 * x + b1

        intersection = (x,y)

        return intersection
```

```
    def clip(self,subject_polygon,clipping_polygon):

        final_polygon = subject_polygon.copy()

        for i in range(len(clipping_polygon)):

            next_polygon = final_polygon.copy()

            final_polygon = []
            c_edge_start = clipping_polygon[i - 1]
            c_edge_end = clipping_polygon[i]

            for j in range(len(next_polygon)):

                s_edge_start = next_polygon[j - 1]
                s_edge_end = next_polygon[j]

                if self.is_inside(c_edge_start,c_edge_end,s_edge_end):
                    if not self.is_inside(c_edge_start,c_edge_end,s_edge_start):
                        intersection =
self.compute_intersection(s_edge_start,s_edge_end,c_edge_start,c_edge_end)
                        final_polygon.append(intersection)
                    final_polygon.append(tuple(s_edge_end))
                elif self.is_inside(c_edge_start,c_edge_end,s_edge_start):
                    intersection =
self.compute_intersection(s_edge_start,s_edge_end,c_edge_start,c_edge_end)
                    final_polygon.append(intersection)

        return np.asarray(final_polygon)

    def __call__(self,A,B):
        clipped_polygon = self.clip(A,B)
        if len(clipped_polygon) == 0 and self.warn_if_empty:
            warnings.warn("No intersections found. Are you sure your \
                           polygon coordinates are in clockwise order?")

        return clipped_polygon

if __name__ == '__main__':

    # some test polygons

    clip = PolygonClipper()


    subject_polygon = [(0,4),(0.2,0.2),(4,0),(0.2,-0.2),(0,-4),(-0.2,-0.2),(-
4,0),(-0.2,0.2)]
    clipping_polygon = [(-3,-3),(-3,3),(3,3),(3,-3)]

    subject_polygon = np.array(subject_polygon)
    clipping_polygon = np.array(clipping_polygon)
    clipped_polygon = clip(subject_polygon,clipping_polygon)
```

```
    #plotting subject polygon
    x_subject_polygon = list()
    y_subject_polygon = list()
    for i in subject_polygon:
      x_subject_polygon.append(i[0])
      y_subject_polygon.append(i[1])
    x_subject_polygon.append(subject_polygon[0][0])
    y_subject_polygon.append(subject_polygon[0][1])

    #plotting clipping polygon
    x_clipping_polygon = list()
    y_clipping_polygon = list()
    for i in clipping_polygon:
      x_clipping_polygon.append(i[0])
      y_clipping_polygon.append(i[1])
    x_clipping_polygon.append(clipping_polygon[0][0])
    y_clipping_polygon.append(clipping_polygon[0][1])
    plt.plot(x_subject_polygon, y_subject_polygon)
    plt.plot(x_clipping_polygon, y_clipping_polygon)

    print("Sutherland Hodgeman Polygon Clipping")
    plt.title("Before clipping")
    plt.show()

    plt.title("After clipping")
    #plotting subject polygon
    x_subject_polygon = list()
    y_subject_polygon = list()
    for i in subject_polygon:
      x_subject_polygon.append(i[0])
      y_subject_polygon.append(i[1])
    x_subject_polygon.append(subject_polygon[0][0])
    y_subject_polygon.append(subject_polygon[0][1])

    #plotting clipping polygon
    x_clipped_polygon = list()
    y_clipped_polygon = list()
    for i in clipped_polygon:
      x_clipped_polygon.append(i[0])
      y_clipped_polygon.append(i[1])
    x_clipped_polygon.append(clipped_polygon[0][0])
    y_clipped_polygon.append(clipped_polygon[0][1])
    plt.plot(x_subject_polygon, y_subject_polygon)
    plt.plot(x_clipped_polygon, y_clipped_polygon)
    plt.show()
Sutherland Hodgeman Polygon Clipping
```



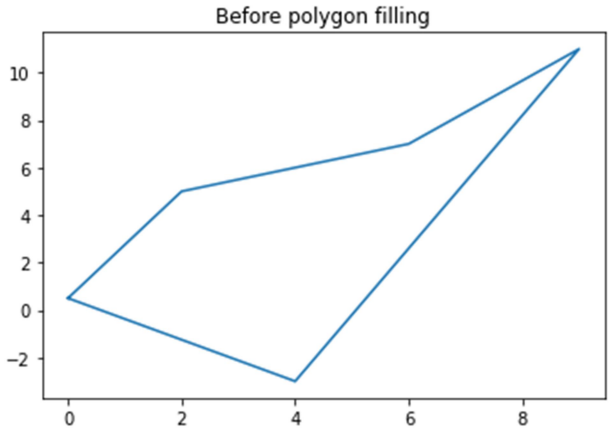


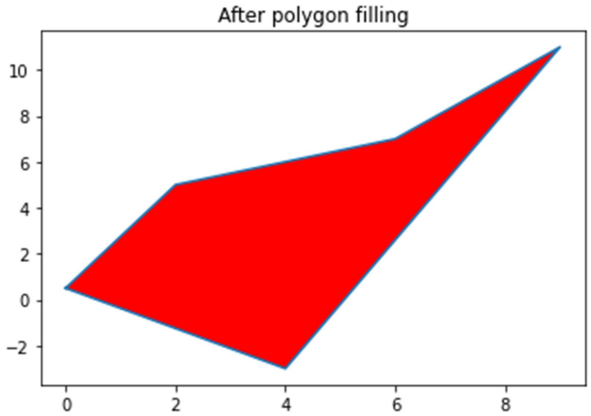
Program 5

Write a program to fill a polygon using Scan line fill algorithm.

```
import numpy as np
import matplotlib.pyplot as plt

def scanLineFill(x_subject_polygon, y_subject_polygon):
        plt.plot(x_subject_polygon, y_subject_polygon)
        print("Scan line filling")
        plt.title("Before polygon filling")
        plt.show()

        plt.title("After polygon filling")
        plt.plot(x_subject_polygon, y_subject_polygon)
        plt.fill(x_subject_polygon, y_subject_polygon, 'r')
        plt.show()
```

```
def definingPolygon():


        subject_polygon = [(0,0.5),(2,5),(6,7),(9,11),(4,-3),(0,0.5)]
        x_subject_polygon = list()
        y_subject_polygon = list()
        for i in subject_polygon:
                x_subject_polygon.append(i[0])
                y_subject_polygon.append(i[1])
        x_subject_polygon.append(subject_polygon[0][0])
        y_subject_polygon.append(subject_polygon[0][1])
        return x_subject_polygon, y_subject_polygon

# main function
x_subject_polygon, y_subject_polygon = definingPolygon()
scanLineFill(x_subject_polygon, y_subject_polygon)
Scan line filling
```





Program 6

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

```
import numpy as np
from math import sin,cos,radians

#translation
def translation(x,y):
  tx=int(input("enter translation in x direction: "))
  ty=int(input("enter translation in y direction: "))
  P=np.array([x,y,1])
  T=np.array([[1,0,tx],[0,1,ty],[0,0,1]])
  P_=np.matmul(T,np.transpose(P))
  x_=P_[0]
  y_=P_[1]
  return x_,y_

#rotation
def rotation(x,y):
  theta=int(input("enter rotating angle(anti-clockwise): "))
  P=np.array([x,y,1])
  R=np.array([[cos(radians(theta)),-(sin(radians(theta))),0],
              [sin(radians(theta)),cos(radians(theta)),0],
              [0,0,1]])
  P_=np.matmul(R,np.transpose(P))
  x_=round(P_[0],3)
  y_=round(P_[1],3)
  return x_,y_

#scaling
def scaling(x,y):
  sx=int(input("enter scaling factor in x direction: "))
  sy=int(input("enter scaling factor in y direction: "))
  P=np.array([x,y,1])
  S=np.array([[sx,0,0],[0,sy,0],[0,0,1]])
  P_=np.matmul(S,np.transpose(P))
  x_=P_[0]
  y_=P_[1]
  return x_,y_

#reflection
def reflection(x,y):
  P=np.array([x,y,1])
  print("1. about x-axis")
  print("2. about y-axis")
  print("3. about origin")
  print("4. about y=x")
  print("5. about y=-x")
  choice=int(input("enter choice: "))
  Re=np.empty([3,3])
  if choice==1:
```

```
    Re=np.array([[1,0,0],[0,-1,0],[0,0,1]])
  elif choice==2:
    Re=np.array([[-1,0,0],[0,1,0],[0,0,1]])
  elif choice==3:
    Re=np.array([[-1,0,0],[0,-1,0],[0,0,1]])
  elif choice==4:
    Re=np.array([[0,1,0],[1,0,0],[0,0,1]])
  elif choice==5:
    Re=np.array([[0,-1,0],[-1,0,0],[0,0,1]])
  else:
    print("wrong choice!!")
  P_=np.matmul(Re,np.transpose(P))
  x_=P_[0]
  y_=P_[1]
  return x_,y_

#shearing
def shearing(x,y):
  shx=int(input("enter shearing factor in x direction: "))
  shy=int(input("enter shearing factor in y direction: "))
  P=np.array([x,y,1])
  Sh=np.array([[1,shx,0],[shy,1,0],[0,0,1]])
  P_=np.matmul(Sh,np.transpose(P))
  x_=P_[0]
  y_=P_[1]
  return x_,y_

print("1. TRANSLATION")
print("2. ROTATION")
print("3. SCALING")
print("4. REFLECTION")
print("5. SHEARING")
choice=int(input("Enter your choice: "))
x,y=0,0
if choice<6 and choice>0:
  x=int(input("enter x-coordinate: "))
  y=int(input("enter y-coordinate: "))
if choice==1:
  print("Coordinates after transformation: ",translation(x,y))
elif choice==2:
  print("Coordinates after transformation: ",rotation(x,y))
elif choice==3:
  print("Coordinates after transformation: ",scaling(x,y))
elif choice==4:
  print("Coordinates after transformation: ",reflection(x,y))
elif choice==5:
  print("Coordinates after transformation: ",shearing(x,y))
else:
  print("wrong choice!!")
```

```
1. TRANSLATION
2. ROTATION
3. SCALING
```

```
4. REFLECTION
5. SHEARING
Enter your choice: 5
enter x-coordinate: 5
enter y-coordinate: 4
enter shearing factor in x direction: 6
enter shearing factor in y direction: 7
Coordinates after transformation:  (29, 39)
1. TRANSLATION
2. ROTATION
3. SCALING
4. REFLECTION
5. SHEARING

Enter your choice: 1
enter x-coordinate: 2
enter y-coordinate: 3
enter translation in x direction: 4
enter translation in y direction: 4
Coordinates after transformation:  (6, 7)
```

Program 7

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
from math import sin,cos,radians

#translation
def translation(x,y,z):
  tx=int(input("enter translation in x direction: "))
  ty=int(input("enter translation in y direction: "))
  tz=int(input("enter translation in z direction: "))
  P=np.array([x,y,z,1])
  T=np.array([[1,0,0,tx],[0,1,0,ty],[0,0,1,tz],[0,0,0,1]])
  P_=np.matmul(T,np.transpose(P))
  x_=P_[0]
  y_=P_[1]
  z_=P_[2]
  return x_,y_,z_

#rotation
def rotation(x,y,z):
  print("1. about x-axis")
  print("2. about y-axis")
  print("3. about z-axis")
  choice=int(input("enter choice: "))
  theta=int(input("enter rotating angle(anti-clockwise): "))
  P=np.array([x,y,z,1])
  R=np.empty([4,4])
  if choice==1:
    R=np.array([[1,0,0,0],
                [0,cos(radians(theta)),-(sin(radians(theta))),0],
                [0,sin(radians(theta)),cos(radians(theta)),0],
                [0,0,0,1]])
  elif choice==2:
    R=np.array([[cos(radians(theta)),0,sin(radians(theta)),0],
                [0,1,0,0],
                [-(sin(radians(theta))),0,cos(radians(theta)),0],
                [0,0,0,1]])
  elif choice==3:
    R=np.array([[cos(radians(theta)),-(sin(radians(theta))),0,0],
                [sin(radians(theta)),cos(radians(theta)),0,0],
                [0,0,1,0],
                [0,0,0,1]])
  else:
    print("wrong choice!!")
  P_=np.matmul(R,np.transpose(P))
  x_=round(P_[0],3)
  y_=round(P_[1],3)
  z_=round(P_[2],3)
  return x_,y_,z_
```

```
#scaling
def scaling(x,y,z):
  sx=int(input("enter scaling factor in x direction: "))
  sy=int(input("enter scaling factor in y direction: "))
  sz=int(input("enter scaling factor in z direction: "))
  P=np.array([x,y,z,1])
  S=np.array([[sx,0,0,0],[0,sy,0,0],[0,0,sz,0],[0,0,0,1]])
  P_=np.matmul(S,np.transpose(P))
  x_=P_[0]
  y_=P_[1]
  z_=P_[2]
  return x_,y_,z_

#reflection
def reflection(x,y,z):
  P=np.array([x,y,z,1])
  print("1. about x-y plane")
  print("2. about y-z plane")
  print("3. about x-z plane")
  choice=int(input("enter choice: "))
  Re=np.empty([4,4])
  if choice==1:
    Re=np.array([[1,0,0,0],[0,1,0,0],[0,0,-1,0],[0,0,0,1]])
  elif choice==2:
    Re=np.array([[-1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]])
  elif choice==3:
    Re=np.array([[1,0,0,0],[0,-1,0,0],[0,0,1,0],[0,0,0,1]])
  else:
    print("wrong choice!!")
  P_=np.matmul(Re,np.transpose(P))
  x_=P_[0]
  y_=P_[1]
  z_=P_[2]
  return x_,y_,z_

#shearing
def shearing(x,y,z):
  shx=int(input("enter shearing factor in x direction: "))
  shy=int(input("enter shearing factor in y direction: "))
  shz=int(input("enter shearing factor in z direction: "))
  P=np.array([x,y,z,1])
  print("1. about x axis")
  print("2. about y axis")
  print("3. about z axis")
  choice=int(input("enter choice: "))
  Sh=np.empty([4,4])
  if choice==1:
    Sh=np.array([[1,shy,shz,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]])
  elif choice==2:
    Sh=np.array([[1,0,0,0],[shx,1,shy,0],[0,0,1,0],[0,0,0,1]])
  elif choice==3:
    Sh=np.array([[1,0,0,0],[0,1,0,0],[shx,shy,1,0],[0,0,0,1]])
  else:
    print("wrong choice!!")
```

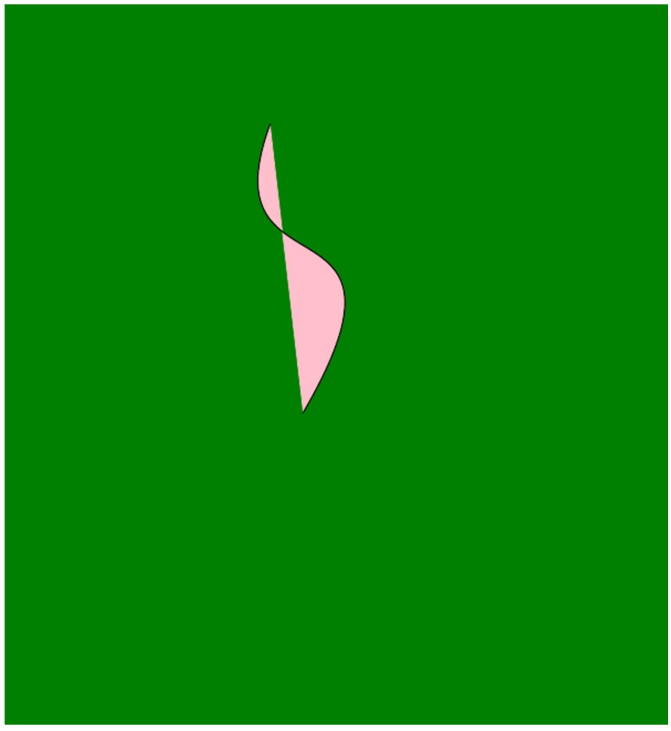
```
  P_=np.matmul(np.transpose(Sh),np.transpose(P))
  x_=P_[0]
  y_=P_[1]
  z_=P_[2]
  return x_,y_,z_
print("1. TRANSLATION")
print("2. ROTATION")
print("3. SCALING")
print("4. REFLECTION")
print("5. SHEARING")
choice=int(input("enter choice: "))
x,y=0,0
if choice<6 and choice>0:
  x=int(input("enter x-coordinate: "))
  y=int(input("enter y-coordinate: "))
  z=int(input("enter z-coordinate: "))
if choice==1:
  print("Coordinates after transformation: ",translation(x,y,z))
elif choice==2:
  print("Coordinates after transformation: ",rotation(x,y,z))
elif choice==3:
  print("Coordinates after transformation: ",scaling(x,y,z))
elif choice==4:
  print("Coordinates after transformation: ",reflection(x,y,z))
elif choice==5:
  print("Coordinates after transformation: ",shearing(x,y,z))
else:
  print("wrong choice!!")
```

```
1. TRANSLATION
2. ROTATION
3. SCALING
4. REFLECTION
5. SHEARING
enter choice: 2
enter x-coordinate: 3
enter y-coordinate: 5
enter z-coordinate: 7
1. about x-axis
2. about y-axis
3. about z-axis
enter choice: 2
enter rotating angle(anti-clockwise): 45
Coordinates after transformation:  (7.071, 5.0, 2.828
```

Program 8

Write a program to draw Hermite /Bezier curves.

```
from wand.image import Image
from wand.drawing import Drawing
from wand.color import Color
with Drawing() as draw:

        # set stroke color
        draw.stroke_color = Color('black')

        # set width for stroke
        draw.stroke_width = 1

        # points list to determine curve
        points = [(240, 100),
                        (180, 250),
                        (390, 150),
                        (270, 340)]
        draw.fill_color = Color('pink')

        draw.bezier(points)
        with Image(width = 600,
                        height = 600,
                        background = Color('green')) as img:

                draw.draw(img)
                img.save(filename ='bezier.png')
```



THANK YO