GATE INSTITUTE OF TECHNOLOGY & MANAGEMENT SCIENCES

(Approved by AICTE , Affiliated to Sri Venkateswara University , TIRUPATI)

ICET CODE : GTMT



Master of Computer Applications

SEMESTER - 3

Hall Ticket No: _____

Name of the Student: _____

Subject: Computer Graphic's

Year: II Semester: 3 Batch 2025 - 2027

GATE EDUCATIONAL INSTITUTIONS

DEPARTMENT OF COMPUTERS

YEAR II SEMESTER 3

Reg No._____

CERTIFICATE

Certified that this is the bonafide record of practical work done in the laboratory by the candidate _____ *during the academic year* 2026-2027 *subject* computer Graphic's .

No. of Practical's Conducted 13 ***No. of Practical's Attended*** 13

LECTURER **HEAD OF THE DEPARTMENT**

Submitted for the Practical Examination held on_____

Examiners

1.

2.

INDEX

S.No.	Date	Name of the Experiment	Page No.	Marks Awarded	Remarks
01	10/09/26	To Draw a Straight line using the Slope intercept equation	01-04		
02	17/09/26	To draw a line using the Digital Diff-erential Analyzer line drawing Algorithm	05-08		
03	28/09/26	To draw a Straight line using Bresen-ham's line drawing algorithm	09-12		
04	06/10/26	To draw a circle using Polar coordinates	13-16		
05	24/10/26	To draw a circle using Bresenham's circle drawing algorithm.	17-20		
06	27/10/26	To draw a circle using the Midpoint circle Drawing Algorithm	21-24		
07	04/11/26	To Perform 2D transformations	25-30		
08	17/11/26	To implement cohen-sutherland 2D clipping & window view Port Mapping	31-36		
09	24/11/26	To implement the Liang-Barsky line clipping algorithm.	37-42		

INDEX

S.No.	Date	Name of the Experiment	Page No.	Marks Awarded	Remarks
10	28/11/26	TO Perform 3D transformations	43-48		
11	08/12/26	TO draw a house using basic geometrical shapes & line segments	49-53		
12	22/12/26	TO draw a regular decagon	54-58		
13	30/12/26	TO demonstrate the Polygon filling	59-62		

Date :	GATE Educational Institutions	Page No. : 01
	To draw a straight line using the slope intercept equation.	Practical No. : 01

Aim :-

To write and execute a python program to draw a straig-ht line using the slope intercept equation.

Description :-

The equation of a straight line is $Y = mx + bY = mx + b$

where, m = slope of the line

b = Y-intercept

x = x-coordinate

Y = corresponding Y-coordinate

In this program, the slope m is taken as 2 and the Y-intercept b is taken as 1. For a range of x-values, the corresponding Y-values are calculated using the equation $Y = mx + b$

The calculated points are then plotted using the matplotlib library.

Algorithm :-

Step 1 : Start

Step 2 : Import the matplotlib.pyplot library.

Step 3 : Define the slope m and Y-intercept b

Step 4 : Generate a range of x-values from -10 to 10

Step 5 : calculate the corresponding Y-values using $Y = mx + bY = mx + b$

Step 6 : Plot the calculated X and Y values

Step 7 : Add x-axis and Y-axis labels.

Step 8 : Add a title to the graph

Step 9 : Enable the grid

Step 10 : Display the graph

Step 11 : END

Date :

Practical No. :

1. Write a program to draw line using slope intercept equation

Source Code:

```
import matplotlib.pyplot as plt
# Given values
m = 2       # slope
b = 1       # y-intercept
# Generate x values
x = range(-10, 11)
# Calculate y using slope-intercept equation
y = [m * i + b for i in x]
# Draw the line
plt.plot(x, y)
# Add labels and title
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Line Drawing using Slope-Intercept Equation")
# Show grid
plt.grid(True)
# Display the graph
plt.show()
```

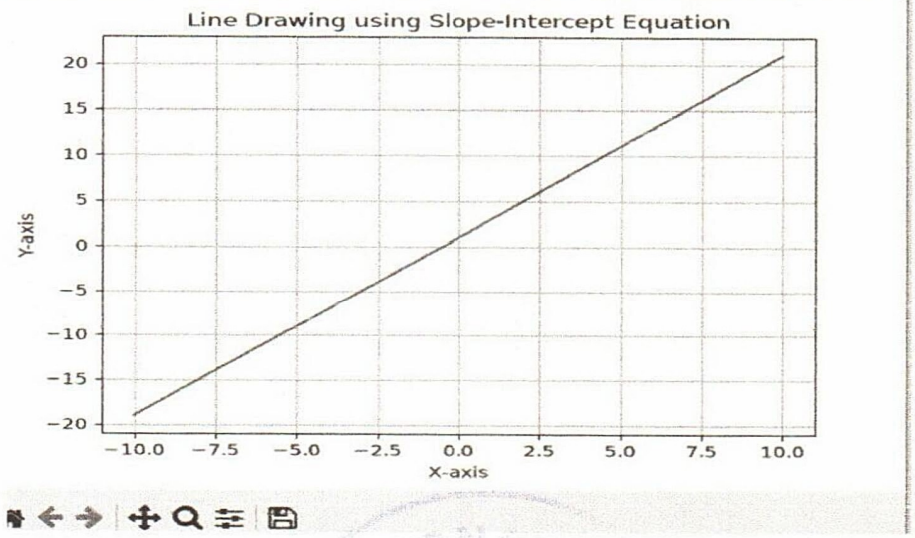
Date :

GATE Educational Institutions

Page No. : 03

OUTPUT:

Practical No. :



Result: The program was successfully executed and a straight line was drawn using the slope-intercept equation y = mx + b, with slope m = 2 and y-intercept b = 1.

Result :-

Thus, the program was successfully executed and a straight line was drawn using the slope-intercept equation.



To draw a line using the Digital Differential Analyzer line drawing Algorithm

Practical No. : 02

Aim :-

To write and execute a python program to draw a line using the Digital Differential Analyzer (DDA) line drawing algorithm

Description :-

DDA (Digital Differential Analyzer) is a line drawing algorithm used in computer graphics. It draws a straight line between two given points. It calculates the intermediate points between the starting and ending points. These points are plotted to form a straight line.

Algorithm :-

Step 1 : Import the matplotlib.pyplot library

Step 2 : Define the starting point (x1, y1) and ending point (x2, y2)

Step 3 : calculate dx and dy

Step 4 : calculate the number of steps using

steps = max (|dx|, |dy|) steps = max(|dx|, |dy|)

Step 5 : calculate x and y increments

Step 6 : Initialize the starting coordinates

Step 7 : Repeat the following steps for the calculated number of steps

* Store the rounded x-coordinate
* Store the rounded y-coordinate
* Increment x
* Increment y

Step 8 : Plot all generated points

Step 9 : Add labels, title, and grid

Step 10 : Display the graph

Date :

Page No. : 06

Practical No. :

2. Write a program to draw line using digital differential analyzer

Source Code:

```
import matplotlib.pyplot as plt
# Starting and ending points
x1 = 2
y1 = 3
x2 = 10
y2 = 8
# Calculate dx and dy
dx = x2 - x1
dy = y2 - y1
# Calculate number of steps
steps = max(abs(dx), abs(dy))
# Calculate increments
x_increment = dx / steps
y_increment = dy / steps
# Lists to store points
x_points = []
y_points = []
# Starting point
x = x1
y = y1
# DDA algorithm
for i in range(steps + 1):
    x_points.append(round(x))
    y_points.append(round(y))
    x = x + x_increment
    y = y + y_increment
# Draw the line
plt.plot(x_points, y_points, marker='o')
# Labels and title
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Line Drawing using DDA Algorithm")
# Grid
plt.grid(True)
# Display graph
plt.show()
```

GATE EDUCATIONAL INSTITUTIONS TIRUPATI

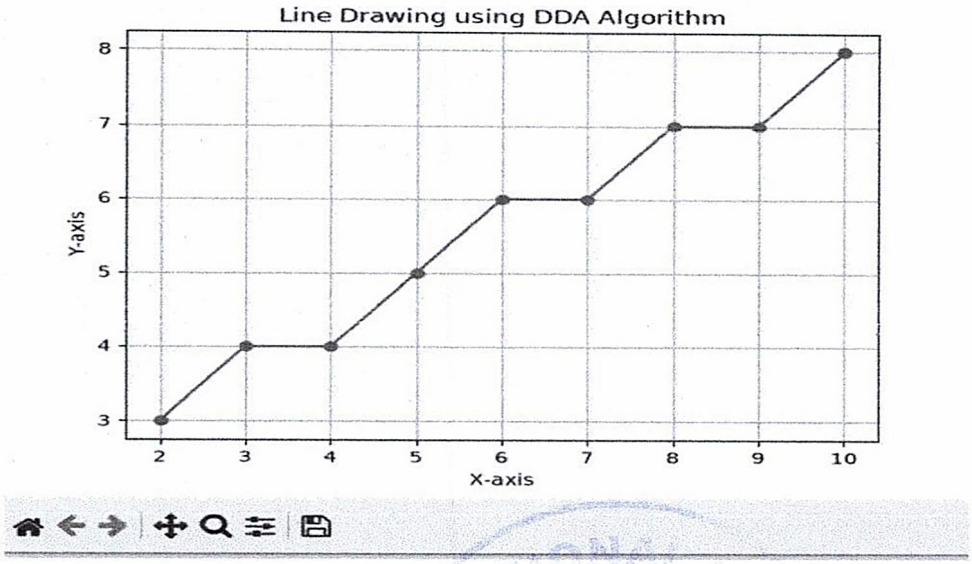
Date :

OUTPUT

GATE Educational Institutions

Page No. : 07

Practical No. :



Result: The program was successfully executed and a straight line was drawn using the Digital Differential Analyzer (DDA) line drawing algorithm between the points (2, 3) and (10, 8). Thus, the required line was successfully displayed on the graph using the DDA algorithm.

Result :-

Thus, the program was successfully executed and a line was drawn between the given points using the DDA line drawing algorithm.



To draw a straight line using Bresenhams line drawing algorithm

Practical No. : 03

Aim :-

To write and execute a Python program to draw a straight line using Bresenham's line drawing algorithm.

Description :-

Bresenham's line algorithm is a line drawing algorithm used in computer graphics. It draws a line by selecting the best pixels between two given points. It uses integer calculations, so it is fast and efficient. It uses a decision parameter to decide the next pixel

Algorithm :-

Step 1 : Define the starting point (x1, y1) and ending point (x2, y2)

Step 2 : calculate :

$dx = x_2 - x_1$ $dX = X_2 - x_1$

$dy = y_2 - y_1$ $dY = Y_2 - y_1$

Step 3 : calculate the initial decision parameter :

$P = 2dy - dx$ $P = 2dY - dX$

Step 4 : Store the starting point

Step 5 : Increases x by one

Step 6 : Check the decision parameter

Step 7 : If $P < 0$ keep the same y-coordinate and update

$P = P + 2dy$ $P = P + 2dY$

Step 8 : Otherwise, increase y by one and update

$P = P + 2dy - 2dx$ $P = P + 2dY - 2dX$

Step 9 : Repeat until the ending point is reached

Step 10 : Plot all generated points.

Date :

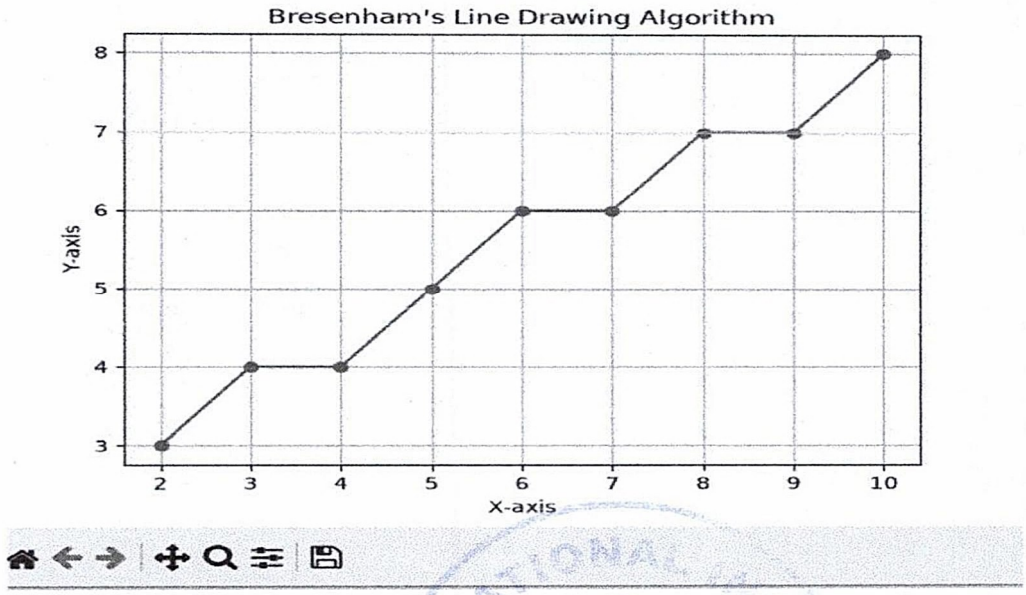
Page No. : 10

Practical No. :

3. Write a program to implement Bresenham's line drawing algorithm

Source Code:

```
import matplotlib.pyplot as plt
# Starting and ending points
x1 = 2
y1 = 3
x2 = 10
y2 = 8
# Calculate dx and dy
dx = x2 - x1
dy = y2 - y1
# Initial decision parameter
p = 2 * dy - dx
# Lists to store points
x_points = [x1]
y_points = [y1]
x = x1
y = y1
# Bresenham's algorithm
while x < x2:
    x = x + 1
    if p < 0:
        p = p + 2 * dy
    else:
        y = y + 1
        p = p + 2 * dy - 2 * dx
    x_points.append(x)
    y_points.append(y)
# Draw the line
plt.plot(x_points, y_points, marker='o')
# Labels and title
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Bresenham's Line Drawing Algorithm")
# Display grid
plt.grid(True)
# Display the graph
plt.show()
```



Result: The program was successfully executed and a **straight line was drawn using Bresenham's Line Drawing Algorithm** between the points **(2, 3)** and **(10, 8)**. Thus, the required line was successfully displayed on the graph using the **Bresenham's Line Drawing Algorithm**.

Result :-

Thus, the (a1g) program was successfully executed and a line was drawn using Bresenham's line drawing algorithm.



To draw a circle using polar coordinates

Practical No. : 04

Aim :-

To write and execute a Python program to draw a circle using polar coordinates.

Description :-

The polar circle drawing method is used to draw a circle in computer graphics. It calculates the points of a circle using the center, radius, and angle. The angle is changed from 0° to 360° to get points on the circle. These points are plotted to form the complete circle.

Algorithm :-

Step 1 : Import matplotlib and math

Step 2 : Define the center of the circle

Step 3 : Define the radius

Step 4 : Create empty lists for x and y coordinates

Step 5 : Generate angles from 0° to 360°

Step 6 : Convert each angle from degrees to radians

Step 7 : Calculate x using : $x = xc + r\cos(\theta)$; $\theta x = x_c + r*\cos(theta$

Step 8 : Calculate y using : $y = yc + r\sin(\theta)$; $\theta y = y_c + r*\sin(theta$

Step 9 : Store the generated coordinates.

Step 10 : Plot the points

Step 11 : Set equal scaling for both axes

Step 12 : Display the circle.

Date :

Page No. : 14

4. Write a program to implement scan convert using polar co-ordinates

Practical No. :

Source Code:

```
import matplotlib.pyplot as plt
import math
# Center of the circle
xc = 0
yc = 0
# Radius
r = 5
# Lists to store points
x_points = []
y_points = []
# Scan convert using polar coordinates
for angle in range(0, 361):
    theta = math.radians(angle)
    x = xc + r * math.cos(theta)
    y = yc + r * math.sin(theta)
    x_points.append(x)
    y_points.append(y)
# Draw the circle
plt.plot(x_points, y_points)
# Labels and title
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Scan Conversion using Polar Coordinates")
# Equal scale for X and Y axes
plt.axis("equal")

# Display grid
plt.grid(True)
# Display the graph
plt.show()
```

Date :

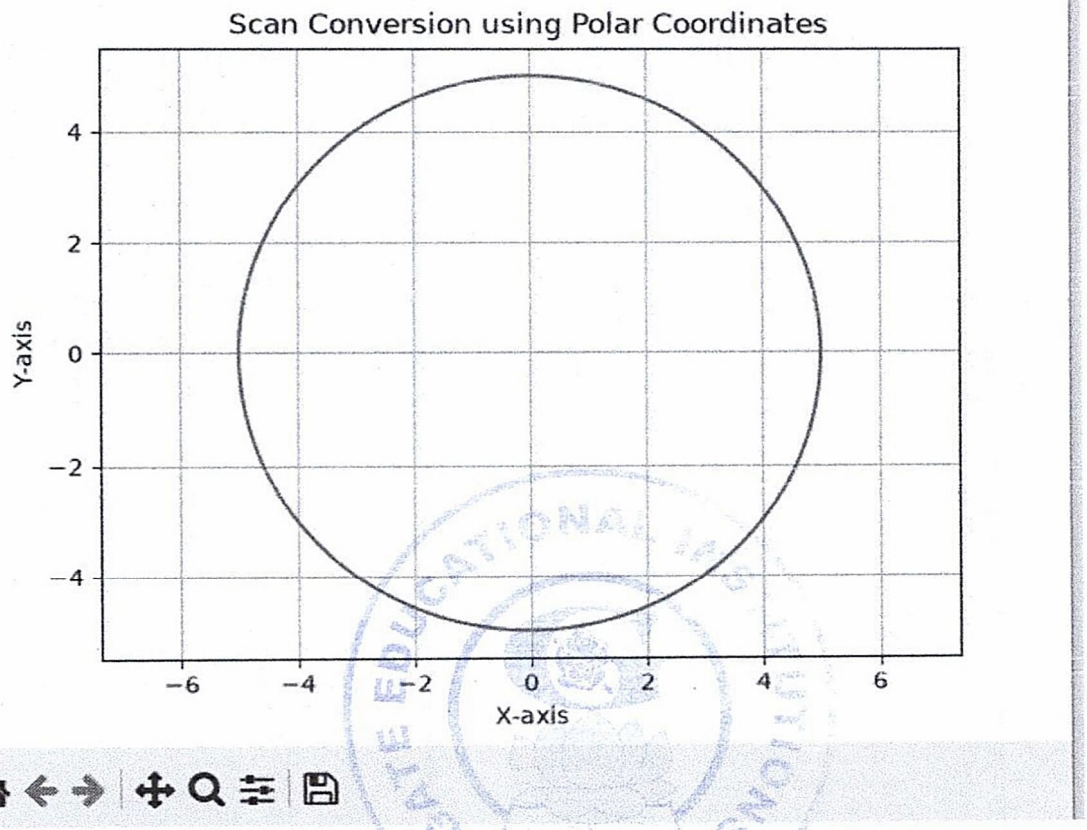
GATE Educational Institutions

Page No. : 15

OUTPUT:

Practical No. :

Figure 1



Result: The circle was successfully drawn using **scan conversion with polar coordinates**.

Result :-

Thus, the Program was successfully executed and a circle was drawn using Polar coordinate scan convers-ion.

GATE EDUCATIONAL INSTITUTIONS TIRUPATI

To draw a circle using Bresenham's circle drawing algorithm

Practical No. : 05

Aim :-

To write and execute a Python program to draw a circle using Bresenham's circle drawing algorithm.

Description :-

Bresenham's circle Algorithm is an efficient circle drawing algorithm used in computer graphics. It uses a decision parameter to select the next pixel. It uses the 8-way symmetry of a circle to draw all the points. The initial decision parameter is $P=3-2r$, and the process continues until $x > y$.

Algorithm :-

Step 1 : Define the center (xc, yc) and radius r.

Step 2 : Initialize $x=0$ $x=0$ $y=r$ $y=r$

Step 3 : Set the initial decision parameter : $P=3-2r$ $P=3-2r$

Step 4 : Calculate the eight symmetric points

Step 5 : Store these points

Step 6 : If $P<0$, update : $P=P+4x+6$ $P=P+4x+6$

Step 7 : otherwise, update : $P=P+4(x-y)+10$ $P=P+4(x-y)+10$ and decrease y.

Step 8 : Increase x

Step 9 : Repeat until $x>y$

Step 10 : Plot all points

Date :

GATE Educational Institutions

Page No. : 18

5. Write a program to implement Bresenham's circle drawing algorithm

Practical No. :

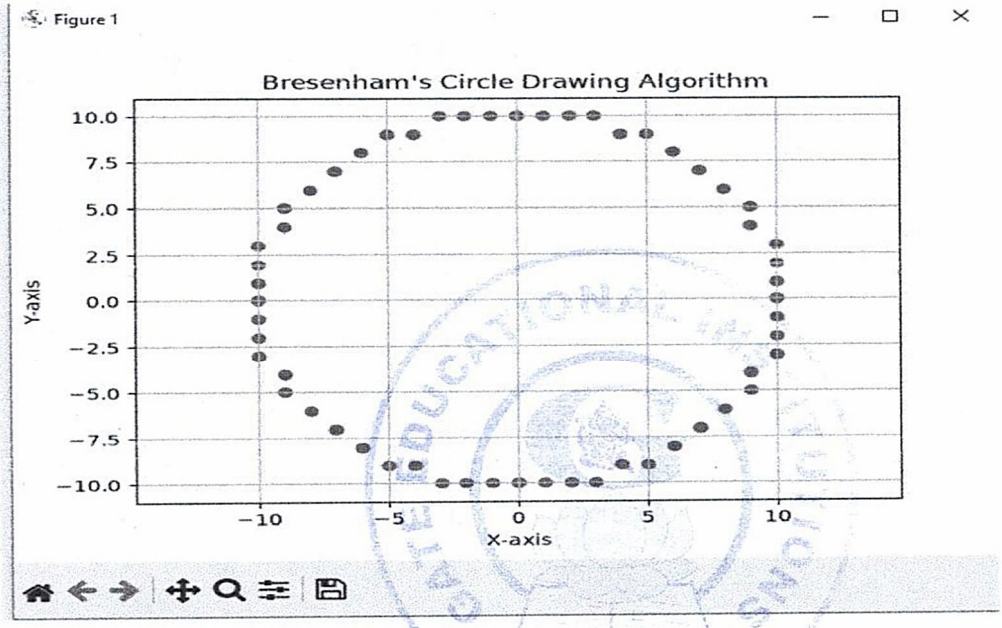
Source Code:

```
import matplotlib.pyplot as plt
# Center and radius of the circle
xc = 0
yc = 0
r = 10
# Lists to store points
x_points = []
y_points = []
# Initial values
x = 0
y = r
p = 3 - 2 * r
# Bresenham's circle algorithm
while x <= y:
    # Calculate the 8 symmetric points
    points = [
        (xc + x, yc + y),
        (xc - x, yc + y),
        (xc + x, yc - y),
        (xc - x, yc - y),
        (xc + y, yc + x),
        (xc - y, yc + x),
        (xc + y, yc - x),
        (xc - y, yc - x)
    ]
    for px, py in points:
        x_points.append(px)
        y_points.append(py)
    # Decision parameter
    if p < 0:
        p = p + 4 * x + 6
    else:
        p = p + 4 * (x - y) + 10
        y = y - 1
    x = x + 1
# Draw the circle
plt.scatter(x_points, y_points)
# Labels and title
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Bresenham's Circle Drawing Algorithm")
```

GATE EDUCATIONAL IRIS (TUTIONS) TIRUPATI

```
# Equal scale
plt.axis("equal")
# Grid
plt.grid(True)
# Display
plt.show()
```

OUTPUT:



Result:-The circle was successfully drawn using **Bresenham's Circle Drawing Algorithm**.

Result :-

Thus, the program was successfully executed and a circle was drawn using Bresenham's circle drawing algorithm.

GATE EDUCATIONAL INSTITUTIONS TIRUPATI

to draw a circle using the Midpoint circle Drawing Algorithm

Practical No. : 06

Aim :-

To write and execute a Python Program to draw a circle using the Midpoint circle Drawing Algorithm

Description :-

The Midpoint circle Algorithm is used to draw a circle in computer graphics It uses a decision Parameter to select the next Pixel. It uses the 8-way symmetry of a circle to generate the points. The initial values are $x=0$, $y=r$, and $P=1-r$

Algorithm :-

Step 1 : Define the center and radius

Step 2 : Initialize $x=0$

Step 3 : Initialize $y=r$

Step 4 : calculate the initial decision Parameter $P=1-rP=1-r$

Step 5 : Generate the eight symmetric Points

Step 6 : If $P<0$, update : $P=P+2x+3P=P+2x+3$

Step 7 : otherwise update : $P=P+2(x-y)+5P=P+2(x-y)+5$ and decrease y.

Step 8 : Increase x

Step 9 : Repeat until $x>y$

Step 10 : Plot all generated points.

Date :

GATE Educational Institutions

Page No. : ३३

6. Write a program to implement midpoint circle drawing algorithm

Practical No. :

Source Code:

```
import matplotlib.pyplot as plt
# Center and radius of the circle
xc = 0
yc = 0
r = 10
# Lists to store points
x_points = []
y_points = []
# Initial values
x = 0
y = r
p = 1 - r
# Midpoint circle algorithm
while x <= y:
    # Store 8 symmetric points
    points = [
        (xc + x, yc + y),
        (xc - x, yc + y),
        (xc + x, yc - y),
        (xc - x, yc - y),
        (xc + y, yc + x),
        (xc - y, yc + x),
        (xc + y, yc - x),
        (xc - y, yc - x)
    ]

    for px, py in points:
        x_points.append(px)
        y_points.append(py)
    # Decision parameter
    if p < 0:
        p = p + 2 * x + 3
    else:
        p = p + 2 * (x - y) + 5
        y = y - 1
    x = x + 1
# Draw the circle
plt.scatter(x_points, y_points)
# Labels and title
plt.xlabel("X-axis")
```

Date :

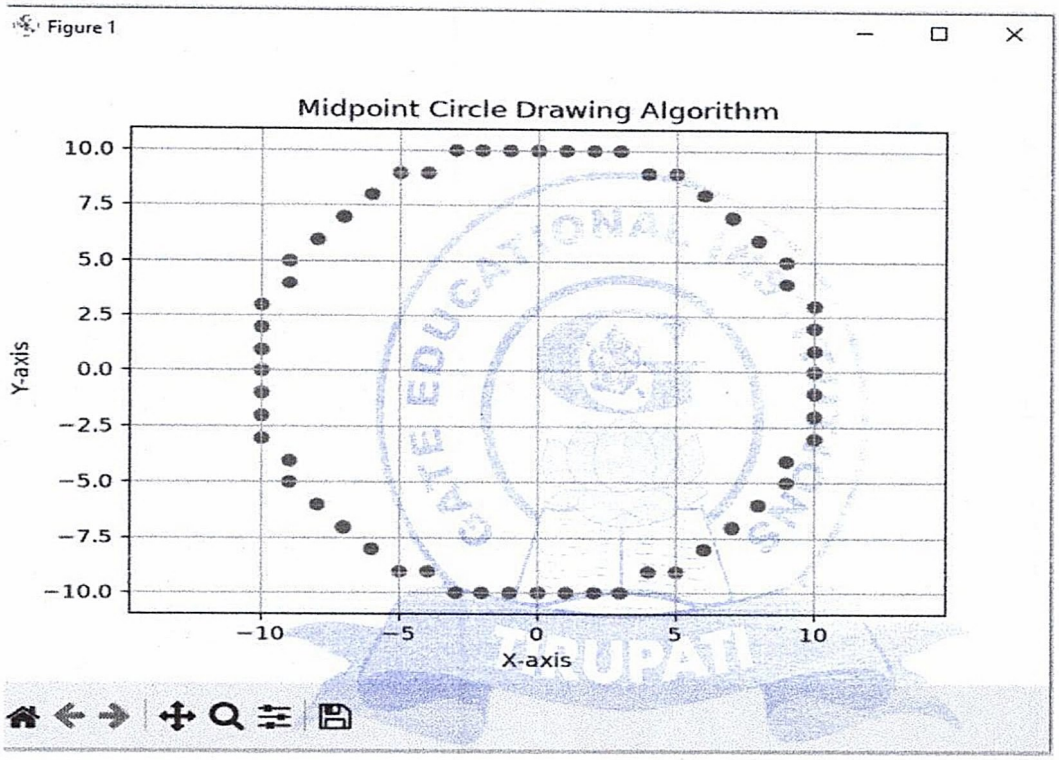
GATE Educational Institutions

Page No. : 23

Practical No. :

```
plt.ylabel("Y-axis")
plt.title("Midpoint Circle Drawing Algorithm")
# Equal scale
plt.axis("equal")
# Grid
plt.grid(True)
# Display
plt.show()
```

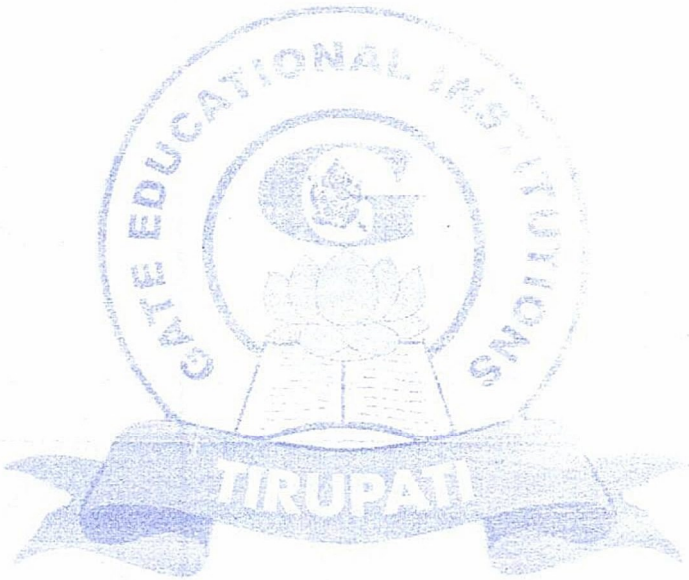
OUTPUT:



Result: The circle was successfully drawn using the **Midpoint Circle Drawing Algorithm**.

Results :-

Thus, the program was successfully executed and a circle was drawn using the Midpoint circle Drawing Algorithm.



To perform 2D Transformations

Practical No. : 07

Aim :-

To write and execute a python program to perform 2D transformations, namely :

* Translation
* Rotation
* Scaling
* Reflection
* Shearing

on a two-dimensional object.

Description :-

2D Transformations are operations used in computer graphics to change the position, size, direction, or shape of an object. The main types of 2D transformations are Translation, Rotation, Scaling, Reflection, and Shearing. Translation moves an object from one position to another, Rotation turns an object through a specified angle, Scaling changes the size of an object. Reflection produces a mirror image, and Shearing slants or changes the shape of an object.

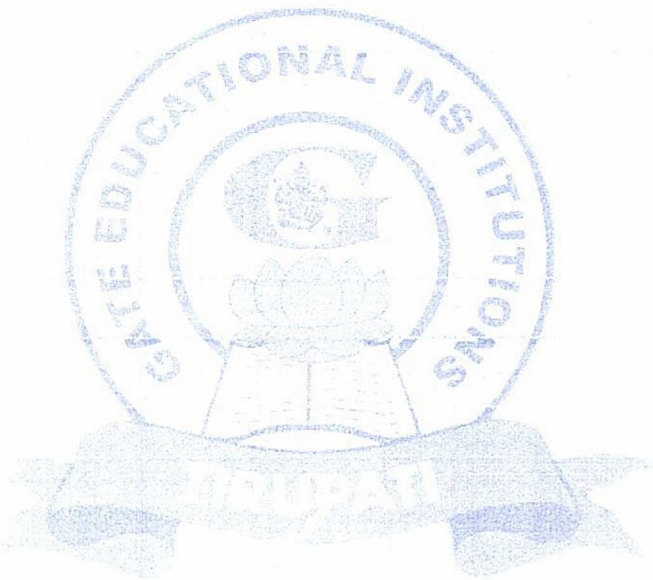
Algorithm :-

Step 1 : Import matplotlib and numpy
Step 2 : Define the coordinates of the original triangle.
Step 3 : Perform translation by adding tx and ty.
Step 4 : Perform rotation using the rotation matrix
Step 5 : Perform scaling using Sx and Sy
Step 6 : Perform reflection about the x-axis
Step 7 : Perform shearing in the x-direction

Step 8 : Create subplots for all transformations

Step 9 : Plot the original and transformed objects

Step 10 : Display all transformation Results.



Date :

GATE Educational Institutions

Page No. : 27

Practical No. :

7. To perform 2D Transformations such as Translation ,Rotation, Scaling, Reflection and Shearing.

Source Code:

```
import matplotlib.pyplot as plt
import numpy as np
# Original triangle coordinates
x = np.array([2, 5, 2, 2])
y = np.array([2, 2, 5, 2])
# ------------------------------
# 1. Translation
# ------------------------------
tx = 3
ty = 2
x_translation = x + tx
y_translation = y + ty
# ------------------------------
# 2. Rotation
# ------------------------------
angle = 45
theta = np.radians(angle)
x_rotation = x * np.cos(theta) - y * np.sin(theta)
y_rotation = x * np.sin(theta) + y * np.cos(theta)
# ------------------------------
# 3. Scaling
# ------------------------------
sx = 2
sy = 2
x_scaling = x * sx
y_scaling = y * sy
# ------------------------------
# 4. Reflection
# ------------------------------
x_reflection = x
y_reflection = -y
# ------------------------------
# 5. Shearing
# ------------------------------
shx = 1
shy = 0
x_shearing = x + shx * y
y_shearing = y + shy * x
# ------------------------------
```

Date :

GATE Educational Institutions

Page No. : 28

Practical No. :

```
# Display all transformations
# ----------------------------
plt.figure(figsize=(10, 8))
# Original
plt.subplot(2, 3, 1)
plt.plot(x, y, marker='o')
plt.title("Original")
plt.grid(True)
plt.axis("equal")
# Translation
plt.subplot(2, 3, 2)
plt.plot(x, y, marker='o', label="Original")
plt.plot(x_translation, y_translation, marker='o', label="Translated")
plt.title("Translation")
plt.grid(True)
plt.axis("equal")
plt.legend()
# Rotation
plt.subplot(2, 3, 3)
plt.plot(x, y, marker='o', label="Original")
plt.plot(x_rotation, y_rotation, marker='o', label="Rotated")
plt.title("Rotation")
plt.grid(True)
plt.axis("equal")
plt.legend()
# Scaling
plt.subplot(2, 3, 4)
plt.plot(x, y, marker='o', label="Original")
plt.plot(x_scaling, y_scaling, marker='o', label="Scaled")
plt.title("Scaling")
plt.grid(True)
plt.axis("equal")
plt.legend()
# Reflection
plt.subplot(2, 3, 5)
plt.plot(x, y, marker='o', label="Original")
plt.plot(x_reflection, y_reflection, marker='o', label="Reflected")
plt.title("Reflection")
plt.grid(True)
plt.axis("equal")
plt.legend()
# Shearing
plt.subplot(2, 3, 6)
plt.plot(x, y, marker='o', label="Original")
```

Date :

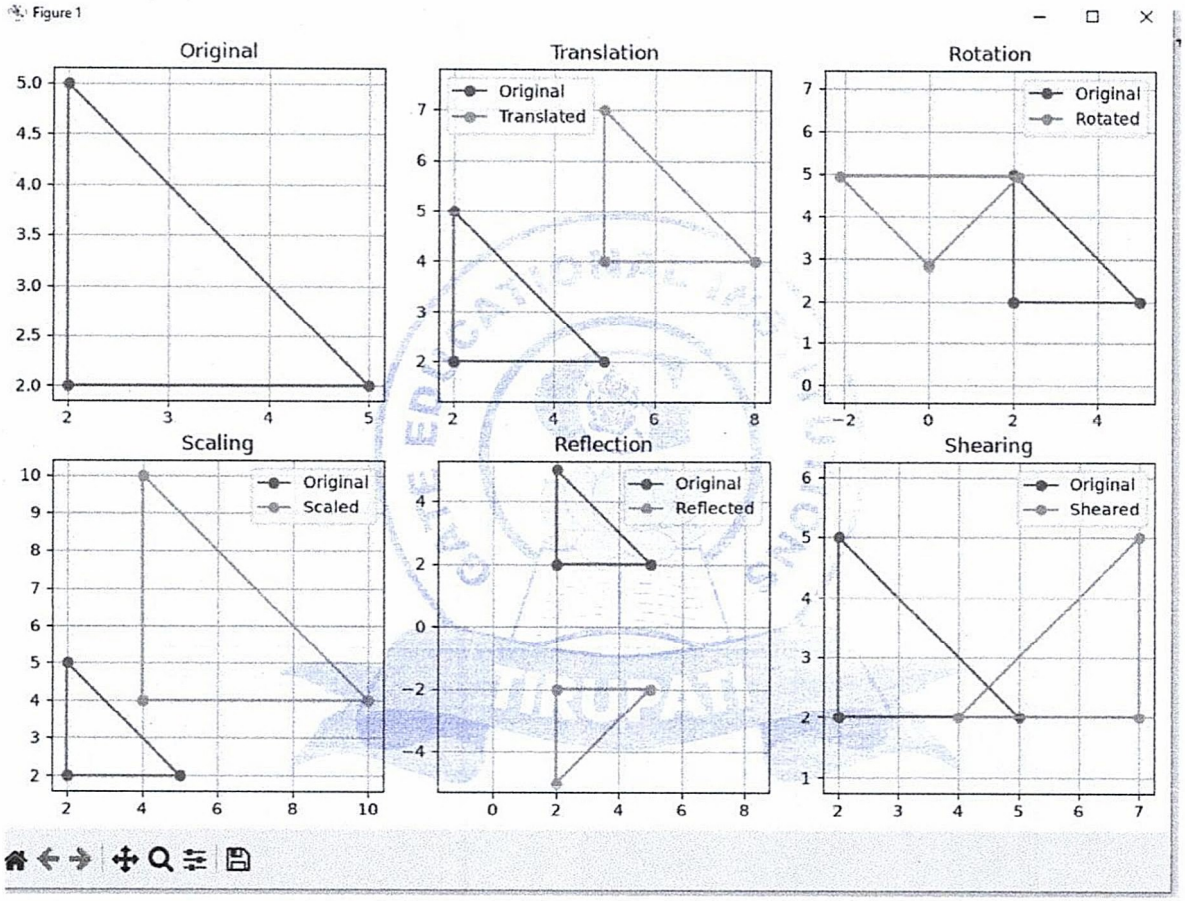
GATE Educational Institutions

Page No. : २१

Practical No. :

```
plt.plot(x_shearing, y_shearing, marker='o', label="Sheared")
plt.title("Shearing")
plt.grid(True)
plt.axis("equal")
plt.legend()
plt.tight_layout()
plt.show()
```

OUTPUT:



Result: The program was successfully executed and the **2D** transformations—Translation, Rotation, Scaling, Reflection, and Shearing—were successfully performed and displayed on the triangle.

Result :-

Thus, the program was successfully executed and the Translation, Rotation, Scaling, Reflection, and Shearing transformations were performed on a 2D triangle.

To implement cohen-Sutherland 2D clipping & window view port Mapping

Practical No. : 08

Aim :-

To write and execute a Python program to implement the cohen-sutherland line clipping algorithm and perform window-to-viewport mapping.

Description :-

The cohen-sutherland Algorithm is used in computer graphics to clip a line inside a rectangular window. each endpoint of the line is given a 4-bit region code based on its position: left, right, top or bottom. the code 0000 represents a point inside the window. Using these codes, the algorithm decides whether the line is completely inside, completely outside, or needs clipping. After clipping, the line can be displayed in the required viewport.

Algorithm :-

Step 1 : Define the clipping window

Step 2 : Define the original line end points.

Step 3 : Define region codes for LEFT, RIGHT, BOTTOM, and TOP.

Step 4 : Calculate the region code of each endpoint

Step 5 : If both codes are zero, accept the line.

Step 6 : If the bitwise AND of the two codes is not zero, reject the line.

Step 7 : Otherwise select the endpoint outside the clipping window.

Step 8 : Calculate its intersection with the appropriate boundary

Step 9 : Replace the outside endpoint with the intersection point.

Step 10 : Repeat the clipping process

Step 11 : If the line is accepted, calculate the viewport coordinates.

Step 12 : Display the original line, clipping window, clipped line, and view port.

Date :

GATE Educational Institutions

Page No. : 33

Practical No. :

8. To implement Cohen-Sutherland 2D clipping and window view port mapping.

Source Code:

```
import matplotlib.pyplot as plt
# Clipping window
xmin = 2
ymin = 2
xmax = 8
ymax = 6
# Line coordinates
x1 = 1
y1 = 1
x2 = 10
y2 = 8
# Region codes
INSIDE = 0
LEFT = 1
RIGHT = 2
BOTTOM = 4
TOP = 8
# Calculate region code
def compute_code(x, y):
    code = INSIDE
    if x < xmin:
        code |= LEFT
    elif x > xmax:
        code |= RIGHT
    if y < ymin:
        code |= BOTTOM
    elif y > ymax:
        code |= TOP
    return code
# Cohen-Sutherland Line Clipping
def cohen_sutherland(x1, y1, x2, y2):
    code1 = compute_code(x1, y1)
    code2 = compute_code(x2, y2)
    accept = False
    while True:
        # Both points are inside
        if code1 == 0 and code2 == 0:
            accept = True
            break
        # Both points are outside in the same region
        elif (code1 & code2) != 0:
            break
        else:
            # Select point outside
            if code1 != 0:
                code_out = code1
```

Date :

GATE Educational Institutions

Page No. : 34

Practical No. :

```
        else:
            code_out = code2
        # Find intersection point
        if code_out & TOP:
            x = x1 + (x2 - x1) * (ymax - y1) / (y2 - y1)
            y = ymax

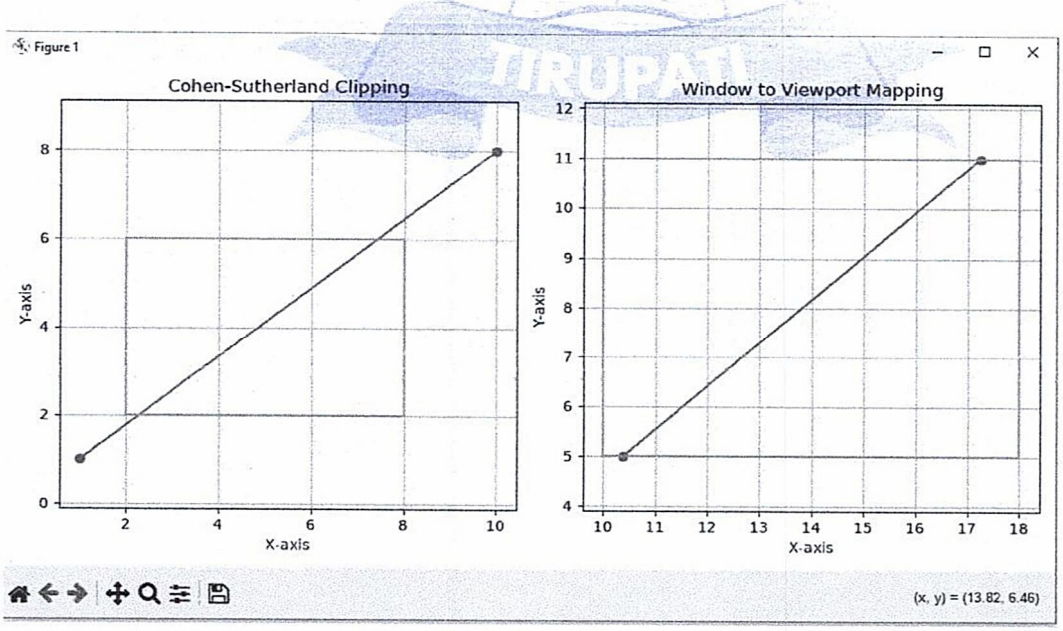
        elif code_out & BOTTOM:
            x = x1 + (x2 - x1) * (ymin - y1) / (y2 - y1)
            y = ymin
        elif code_out & RIGHT:
            y = y1 + (y2 - y1) * (xmax - x1) / (x2 - x1)
            x = xmax
        elif code_out & LEFT:
            y = y1 + (y2 - y1) * (xmin - x1) / (x2 - x1)
            x = xmin
        # Replace outside point
        if code_out == code1:
            x1 = x
            y1 = y
            code1 = compute_code(x1, y1)
        else:
            x2 = x
            y2 = y
            code2 = compute_code(x2, y2)
    if accept:
        return x1, y1, x2, y2
    else:
        return None
# Perform clipping
result = cohen_sutherland(x1, y1, x2, y2)
# Window to Viewport Mapping
if result:

    cx1, cy1, cx2, cy2 = result
    # Viewport coordinates
    vxmin = 10
    vymin = 5
    vxmax = 18
    vymax = 11
    # Map first point
    vx1 = vxmin + (cx1 - xmin) * (vxmax - vxmin) / (xmax - xmin)
    vy1 = vymin + (cy1 - ymin) * (vymax - vymin) / (ymax - ymin)
    # Map second point
    vx2 = vxmin + (cx2 - xmin) * (vxmax - vxmin) / (xmax - xmin)
    vy2 = vymin + (cy2 - ymin) * (vymax - vymin) / (ymax - ymin)
    # Create graphs
    plt.figure(figsize=(10, 5))
    # Clipping Window
    plt.subplot(1, 2, 1)
```

```
    plt.plot([x1, x2], [y1, y2], marker='o')
    plt.plot(
        [xmin, xmax, xmax, xmin, xmin],
        [ymin, ymin, ymax, ymax, ymin]
    )
    plt.title("Cohen-Sutherland Clipping")
    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    plt.grid(True)
    plt.axis("equal")

    # Viewport
    plt.subplot(1, 2, 2)
    plt.plot([vx1, vx2], [vy1, vy2], marker='o')
    plt.plot(
        [vxmin, vxmax, vxmax, vxmin, vxmin],
        [vymin, vymin, vymax, vymax, vymin]
    )
    plt.title("Window to Viewport Mapping")
    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    plt.grid(True)
    plt.axis("equal")
    plt.tight_layout()
    plt.show()
else:
    print("Line is completely outside the clipping window.")
```

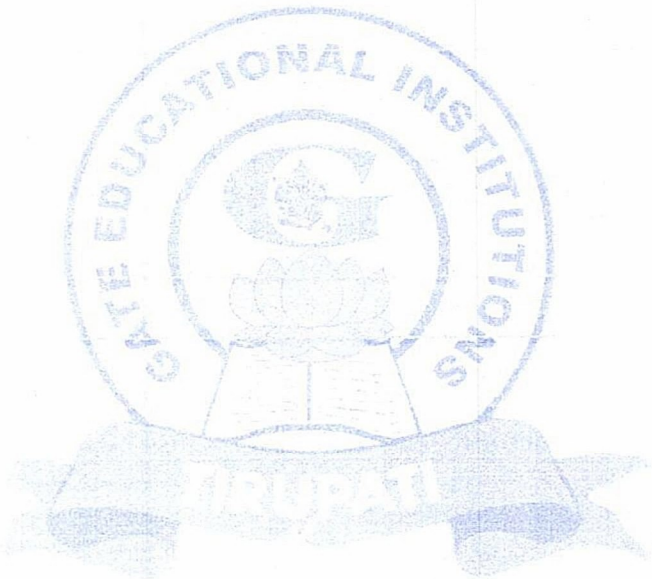
OUTPUT:



Result: The program was successfully executed and **Cohen-Sutherland 2D line clipping and window-to-viewport mapping were successfully performed and displayed.**

Result :-

Thus, the program was successfully executed and cohen-sutherland line clipping and window-to-viewport mapping were performed successfully.



To implement the Liang-Barsky line clipping algorithm

Practical No. : 09

Aim :-

To write and execute a python program to implement the Liang-Barsky line clipping algorithm.

Description :-

The Liang-Barsky Algorithm is an efficient line clipping algorithm used in computer graphics. It uses the parametric form of a line to find the portion of the line inside a clipping window. The value of u ranges from 0 to 1, representing the points between the starting and ending points of the line. The algorithm checks the four boundaries of the clipping window and selects the part of the line that lies inside the window.

Algorithm :-

Step 1 :- Define the clipping window

Step 2 : Define the starting and ending points of the line.

Step 3 : calculate

$dx = x_2 - x_1$, $dx = x_2 - x_1$

$dy = y_2 - y_1$, $dy = y_2 - y_1$

Step 4 : calculate the arrays p and q

Step 5 : initilize

$u_1 = 0$ $u_1 = 0$

$u_2 = 1$ $u-2 = 1$

Step 6: Process all four boundaries.

Step 7: If $p=0$ and $q<0$, reject the line

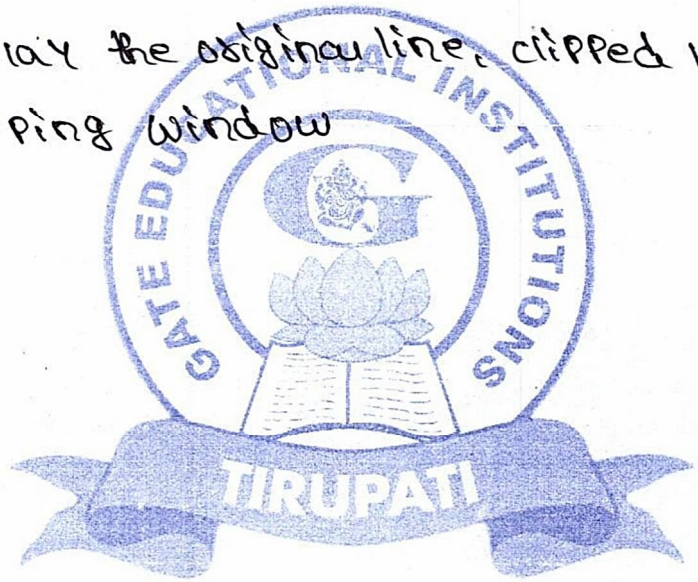
Step 8: If $p<0$, update u_1.

Step 9: If $p>0$, update u_2

Step 10: If $u_1 > u_2$, reject the line

Step 11: otherwise calculate the clipped end points

Step 12: Display the original line, clipped line, and clipping window



GATE Educational Institutions

Date :

Page No. : 39

9. To implement Liang Barsky Line clipping algorithm

Practical No. :

Source Code:

```
import matplotlib.pyplot as plt
# Clipping window
xmin = 2
ymin = 2
xmax = 8
ymax = 6
# Line coordinates
x1 = 1
y1 = 1
x2 = 10
y2 = 8
# Liang-Barsky Line Clipping
def liang_barsky(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    p = [-dx, dx, -dy, dy]
    q = [x1 - xmin, xmax - x1, y1 - ymin, ymax - y1]
    u1 = 0
    u2 = 1
    for i in range(4):
        if p[i] == 0:
            if q[i] < 0:
                return None
        else:
            u = q[i] / p[i]
            if p[i] < 0:
                u1 = max(u1, u)
            else:
                u2 = min(u2, u)
    if u1 > u2:
        return None
    # Calculate clipped line coordinates
    cx1 = x1 + u1 * dx
    cy1 = y1 + u1 * dy
    cx2 = x1 + u2 * dx
    cy2 = y1 + u2 * dy
    return cx1, cy1, cx2, cy2
# Perform clipping
result = liang_barsky(x1, y1, x2, y2)
# Display result
```

Date :

GATE Educational Institutions

Page No. : 40

Practical No. :

```
if result:
    cx1, cy1, cx2, cy2 = result
    print("Clipped Line Coordinates:")
    print("Start Point:", (cx1, cy1))
    print("End Point:", (cx2, cy2))
    # Draw graph
    plt.plot(
        [x1, x2],
        [y1, y2],
        marker='o',
        label="Original Line"
    )

    plt.plot(
        [cx1, cx2],
        [cy1, cy2],
        marker='o',
        label="Clipped Line"
    )
    # Draw clipping window
    plt.plot(
        [xmin, xmax, xmax, xmin, xmin],
        [ymin, ymin, ymax, ymax, ymin],
        label="Clipping Window"
    )
    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    plt.title("Liang-Barsky Line Clipping Algorithm")
    plt.grid(True)
    plt.axis("equal")
    plt.legend()
    plt.show()
else:
    print("Line is completely outside the clipping window.")
```

Date :

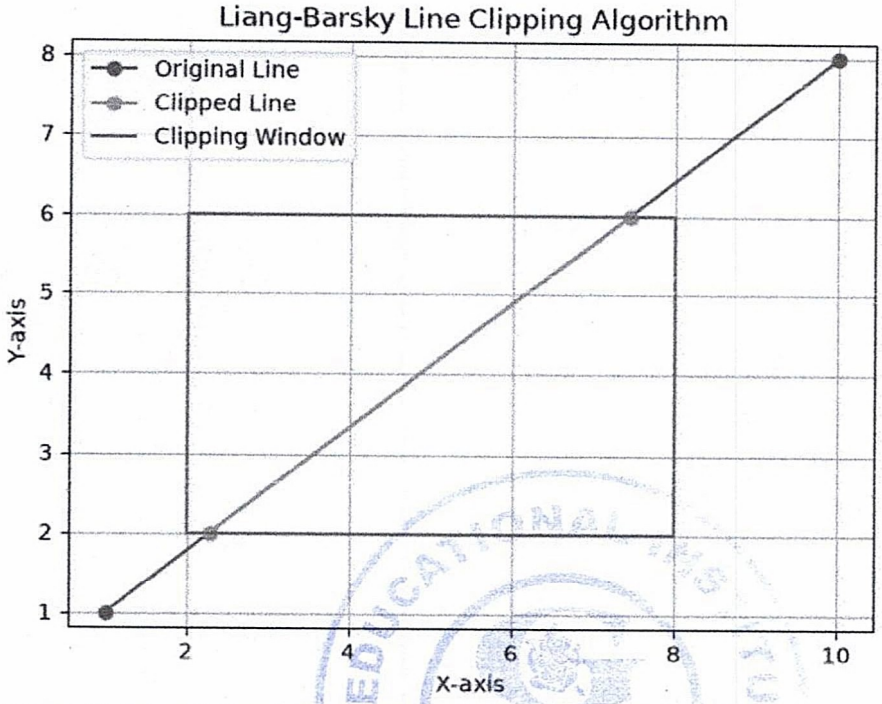
OUTPUT:

Figure 1

GATE Educational Institutions

Page No. : 41

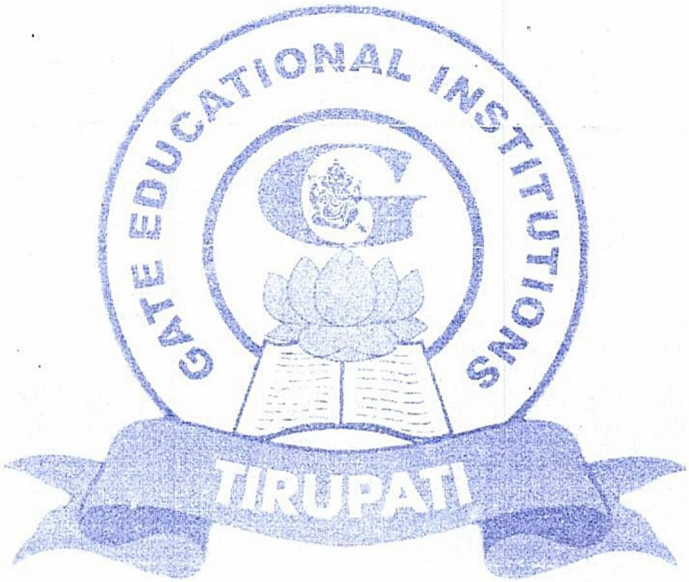
Practical No. :



Result: The program was successfully executed and the **Liang-Barsky Line Clipping Algorithm** was successfully implemented to clip the given line against the specified clipping window.

Result :-

Thus, the program was successfully executed and the given line was clipped using the Liang-Barsky line clipping algorithm



Aim :-

To write and execute a Python program to perform 3D transformations, namely Translation, Rotation and scaling on a three-dimensional object.

Description :-

3D Transformations are used in computer graphics to change the position, size or direction of an object in 3D space. The main transformations are Translation, Rotation and scaling. translation moves the object, Rotation turn the object and scaling changes its size. A cube is used as the orginal 3D objects

Algorithm :-

Step 1 : Import matplotlib and numpy.

Step 2 : Define the eight vertices of the cube.

Step 3 : Define the twelve edges of the cube.

Step 4 : create a function to draw the cube.

Step 5 : Perform translation by adding translation values to all vertices.

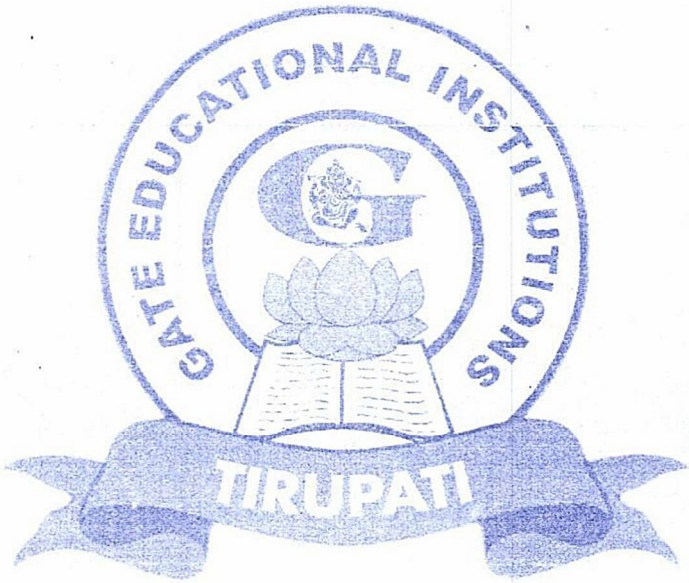
Step 6 : create the rotation matrix for rotation around the z-axis

Step 7 : APPLY the rotation matrix to the vertices

Step 8 : Perform scaling by multiplying the coordinates by scaling factors.

Step 9 : create separate 3D subplots.

Step 10 : Display the original, translated, rotated and scaled cubes



Date :

GATE Educational Institutions

Page No. : 45

10. To perform 3D Transformations such as Translation, Rotation and Scaling.

Practical No. :

Source Code:

```
import matplotlib.pyplot as plt
import numpy as np
# Original 3D cube vertices
vertices = np.array([
    [1, 1, 1],
    [4, 1, 1],
    [4, 4, 1],
    [1, 4, 1],
    [1, 1, 4],
    [4, 1, 4],
    [4, 4, 4],
    [1, 4, 4]
])
# Cube edges
edges = [
    (0, 1), (1, 2), (2, 3), (3, 0),
    (4, 5), (5, 6), (6, 7), (7, 4),
    (0, 4), (1, 5), (2, 6), (3, 7)
]
# Function to draw cube
def draw_cube(ax, points, title):
    for edge in edges:
        p1 = points[edge[0]]
        p2 = points[edge[1]]
        ax.plot(
            [p1[0], p2[0]],
            [p1[1], p2[1]],
            [p1[2], p2[2]]
        )
    ax.set_title(title)
    ax.set_xlabel("X-axis")
    ax.set_ylabel("Y-axis")
    ax.set_zlabel("Z-axis")
# --------------------------------
# 1. Translation
# --------------------------------
tx = 3
ty = 2
tz = 2
translation = np.array([tx, ty, tz])
translated = vertices + translation
```

Date :

GATE Educational Institutions

Page No. : 46

Practical No. :

```
# ---------------------------------
# 2. Rotation about Z-axis
# ---------------------------------
angle = 45
theta = np.radians(angle)
rotation_matrix = np.array([
   [np.cos(theta), -np.sin(theta), 0],
   [np.sin(theta),  np.cos(theta), 0],
   [0,             0,              1]
])
rotated = vertices @ rotation_matrix.T

# ---------------------------------
# 3. Scaling
# ---------------------------------
sx = 2
sy = 2
sz = 2
scaling = np.array([sx, sy, sz])
scaled = vertices * scaling
# ---------------------------------
# Display transformations
# ---------------------------------
fig = plt.figure(figsize=(12, 8))
# Original
ax1 = fig.add_subplot(2, 2, 1, projection='3d')
draw_cube(ax1, vertices, "Original 3D Object")
# Translation
ax2 = fig.add_subplot(2, 2, 2, projection='3d')
draw_cube(ax2, translated, "Translation")
# Rotation
ax3 = fig.add_subplot(2, 2, 3, projection='3d')
draw_cube(ax3, rotated, "Rotation about Z-axis")
# Scaling
ax4 = fig.add_subplot(2, 2, 4, projection='3d')
draw_cube(ax4, scaled, "Scaling")
plt.tight_layout()
plt.show()
```

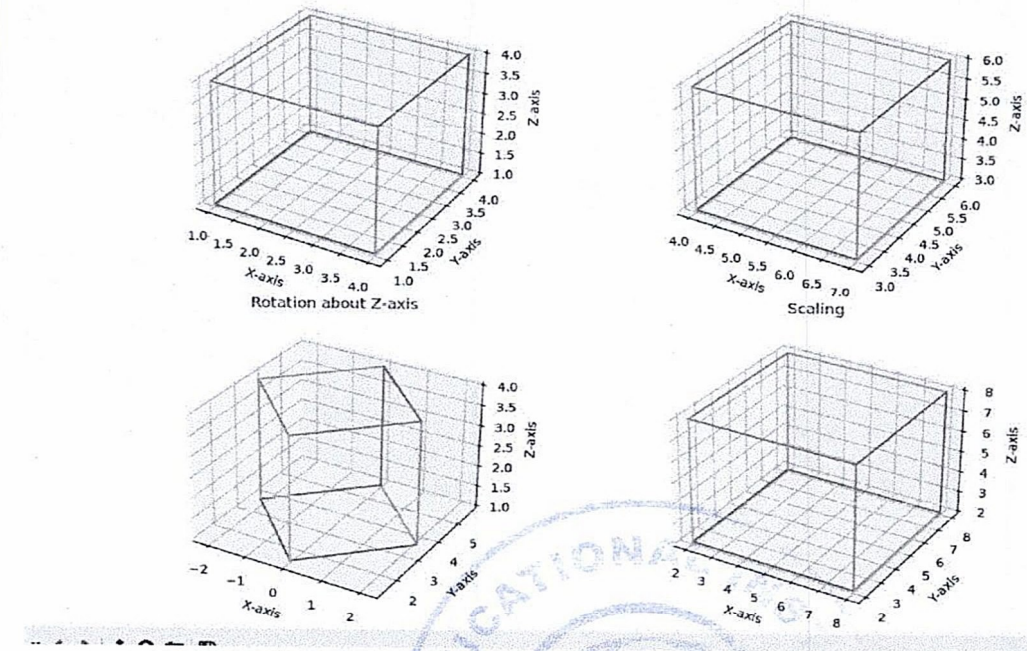
Date :

OUTPUT:

GATE Educational Institutions

Page No. : 47

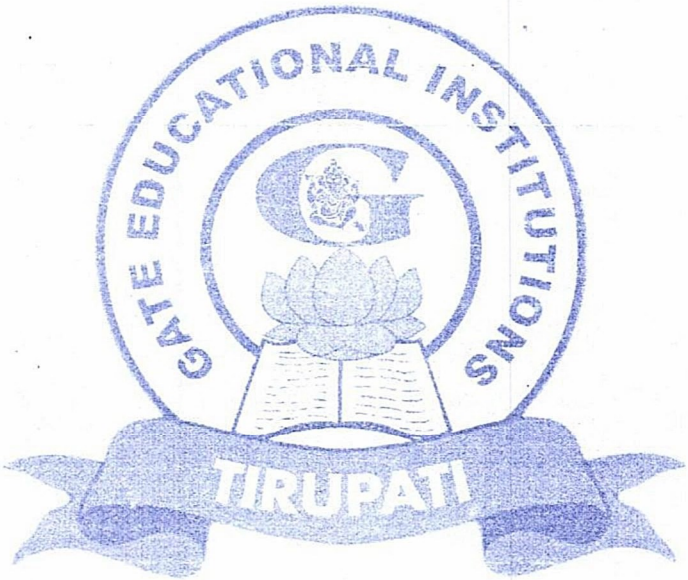
Practical No. :



Result: The program was successfully executed and the 3D transformations—Translation, Rotation about the Z-axis, and Scaling—were successfully performed and displayed on the 3D cube.

Result :-

Thus, the program was successfully executed and Translation, Rotation and scaling transformation were performed on a 3D cube.



TO draw a house using basic geometric -al shapes & line segments.

Practical No. : 11

Aim :-

To write and execute a Python program to draw a house (HUT) using basic geometrical shapes and line segments.

Description :-

A geometrical figure can be constructed by combining basic lines, rectangles and polygons

In this program, a house is created using :

* Rectangular body
* Triangular roof
* Rectangular door
* two windows
* window cross-lines

The matplotlib library is used to draw the geometrical figure.

Algorithm :-

Step 1 : Import matplotlib.pyplot

Step 2 : create a figure window

Step 3 : Define the coordinates of the house body.

Step 4 : Plot the house body

Step 5 : Define and plot the roof

Step 6 : Define and plot the door

Step 7 : Define and plot the left window

Step 8 : Define and Plot the Right window

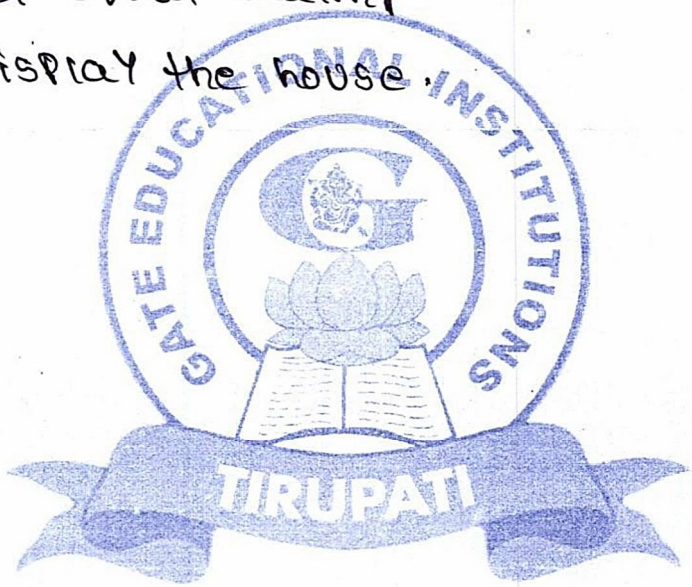
Step 9 : Draw horizantal and vertical lines inside the windows.

Step 10 : Add title and axis labels

Step 11 : Enable the grid

Step 12 : Set equal scaling

Step 13 : Display the house.



Date :	GATE Educational Institutions	Page No. : 51
11. Write a program to draw HUT or another Geometrical Figure		Practical No. :
Source Code:		

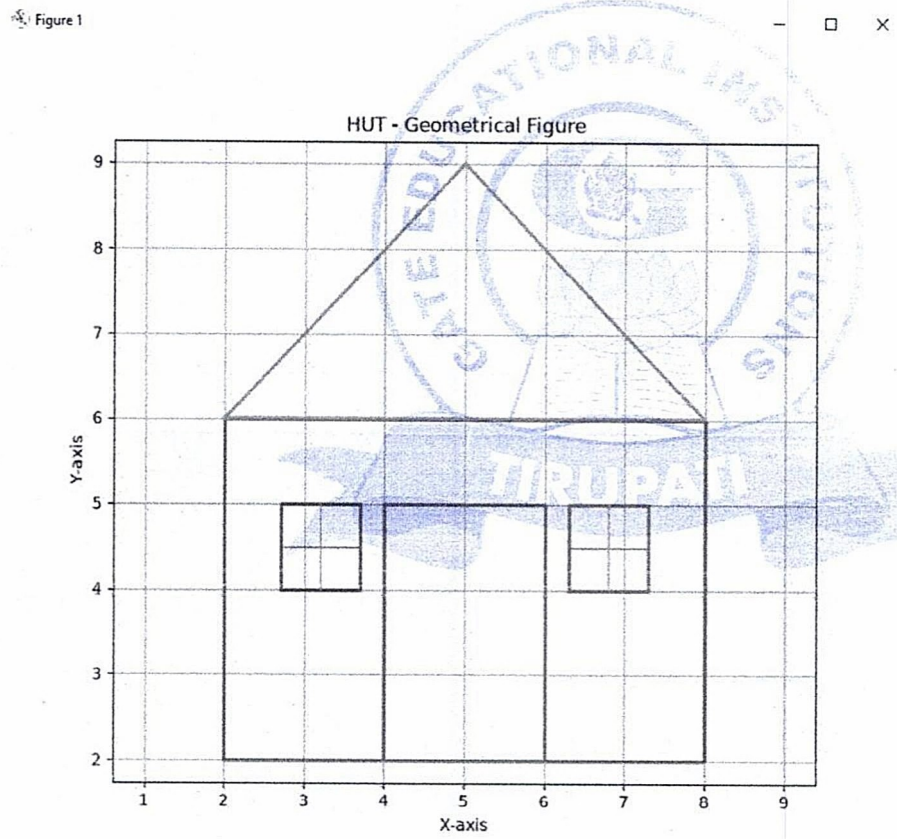
```
import matplotlib.pyplot as plt
# Create figure
plt.figure(figsize=(8, 8))
# ------------------------------
# House Body
# ------------------------------
x_body = [2, 8, 8, 2, 2]
y_body = [2, 2, 6, 6, 2]
plt.plot(x_body, y_body, linewidth=2)
# ------------------------------
# Roof
# ------------------------------
x_roof = [2, 5, 8, 2]
y_roof = [6, 9, 6, 6]
plt.plot(x_roof, y_roof, linewidth=2)
# ------------------------------
# Door
# ------------------------------
x_door = [4, 6, 6, 4, 4]
y_door = [2, 2, 5, 5, 2]
plt.plot(x_door, y_door, linewidth=2)
# ------------------------------
# Left Window
# ------------------------------
x_window1 = [2.7, 3.7, 3.7, 2.7, 2.7]
y_window1 = [4, 4, 5, 5, 4]

plt.plot(x_window1, y_window1, linewidth=2)
# ------------------------------
# Right Window
# ------------------------------
x_window2 = [6.3, 7.3, 7.3, 6.3, 6.3]
y_window2 = [4, 4, 5, 5, 4]
plt.plot(x_window2, y_window2, linewidth=2)
# ------------------------------
# Window Cross Lines
# ------------------------------
plt.plot([2.7, 3.7], [4.5, 4.5], linewidth=1)
plt.plot([3.2, 3.2], [4, 5], linewidth=1)
plt.plot([6.3, 7.3], [4.5, 4.5], linewidth=1)
```

GATE Educational Institutions

```
plt.plot([6.8, 6.8], [4, 5], linewidth=1)
# ------------------------------
# Labels
# ------------------------------
plt.title("HUT - Geometrical Figure")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
# Grid
plt.grid(True)
# Equal scale
plt.axis("equal")
# Display
plt.show()
```

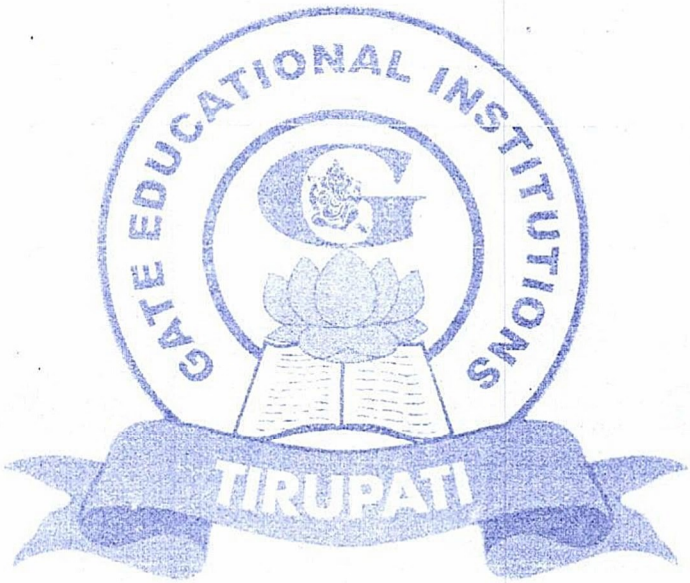
OUTPUT:



Result: The program was successfully executed and the **Hut (House) geometrical figure was successfully drawn and displayed using Python.**

Result :

Thus, the program was successfully executed and a house / Hut geometrical figure was drawn using Python.



To draw a regular decagon

Aim :-

To write and explain a Python program to draw a regular decagon

Description :-

A decagon is a polygon with 10 side and 10 vertices in rectangular regular decagon, the vertices are equally spaced around a circle. The angle between two consecutive vertices is 36°. The coordinates of the vertices are calculated using the radius and angle of the circle.

Algorithm :-

Step 1 : Import matplotlib and math

Step 2 : Set the number of sides to 10

Step 3 : Define the radius.

Step 4 : Create empty lists for x and y coordinates

Step 5 : Calculate the angle for each vertix

Step 6 : Calculate x - coordinate using

$$x = r \cos(\theta) \quad \theta x = r \cos \theta$$

Step 7 : Calculate y - coordinate using

$$y = r \sin(\theta) \quad \theta y = r \sin \theta$$

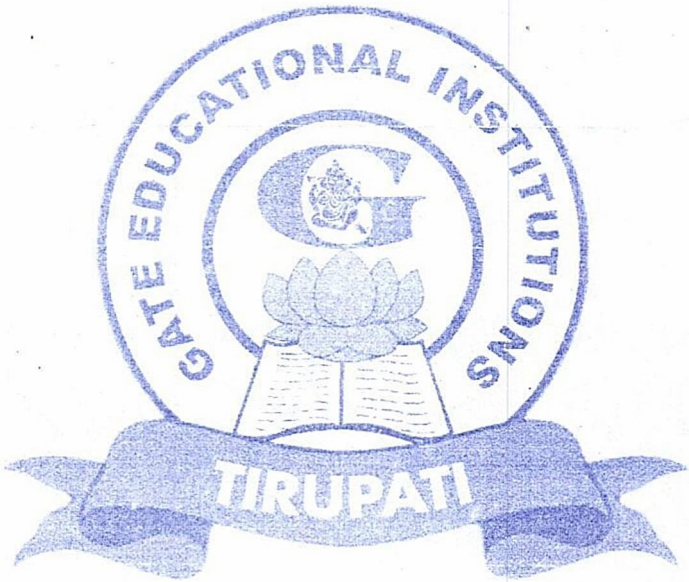
Step 8 : Store all coordinates

Step 9 : Plot the points and connect them

Step 10: Add title and axis labels.

Step 11: Enable grid and equal scaling.

Step 12: Display the decagon.



Date :

GATE Educational Institutions

Page No. : 56

12. Write a program to draw Decagon

Practical No. :

Source Code:

```
import matplotlib.pyplot as plt
import math
# Number of sides
n = 10
# Radius
r = 5
# Lists to store coordinates
x_points = []
y_points = []
# Calculate the vertices of the decagon
for i in range(n + 1):
    angle = 2 * math.pi * i / n
    x = r * math.cos(angle)
    y = r * math.sin(angle)
    x_points.append(x)
    y_points.append(y)
# Draw the decagon
plt.plot(x_points, y_points, marker='o')
# Title and labels
plt.title("Decagon")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
# Grid
plt.grid(True)
# Equal scale
plt.axis("equal")

# Display
plt.show()
```

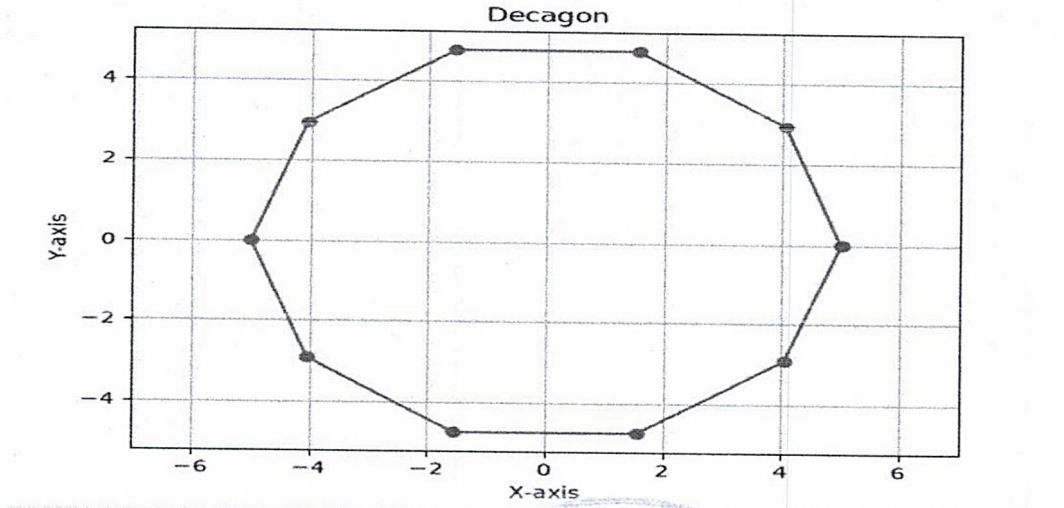
Date :

OUTPUT:

GATE Educational Institutions

Page No. : 57

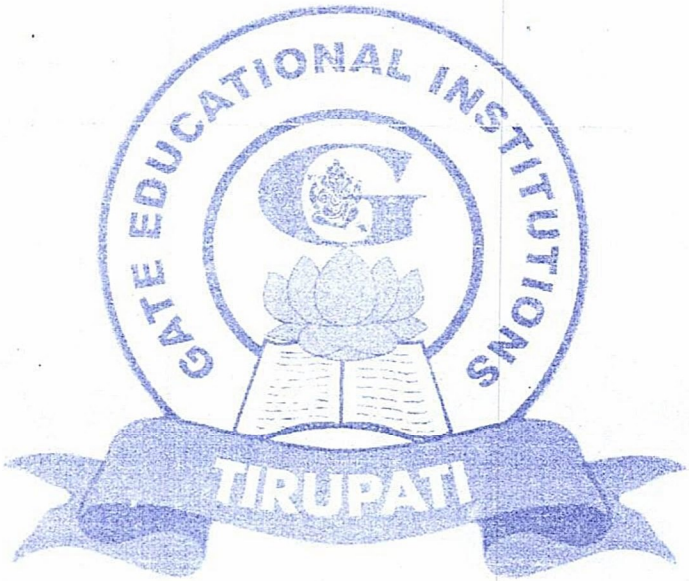
Practical No. :



Result: The program was successfully executed and a **Decagon with 10 sides** was successfully drawn and displayed using Python.

Result :-

Thus, the program was successfully executed and a regular decagon with 10 sides was drawn.



Aim :-

To write and execute a Python Program to demonstrate Polygon filling using Python's Matplotlib library.

Description :-

A Polygon is a closed 2D shape made up of straight lines. Polygon filling fills inside of a polygon with a color or pattern. In their Program, PH:fill() fills the Polygon and PH.plot() draws its boundary.

Algorithm :-

Step 1 : Import matplotlib.Pyplot.
Step 2 : Define the x-coordinates of the Polygon vertices
Step 3 : Define the y-coordinates of the Polygon vertices
Step 4 : Use the fill() function to fill the Polygon
Step 5 : Add the first vertex at the end of the coordinate lists to close the boundary.
Step 6 : Draw the boundary of the Polygon
Step 7 : Add the title and axis labels
Step 8 : Enable the grid
Step 9 : Set equal scaling for both axes
Step 10 : Display the filled Polygon.

Date :

GATE Educational Institutions

Page No. : 60

13. Write a program to demonstrate the polygon filling

Practical No. :

Source Code:

```
import matplotlib.pyplot as plt
# Polygon vertices
x = [2, 6, 8, 7, 4, 2]
y = [2, 2, 5, 8, 6, 5]
# Fill the polygon
plt.fill(x, y)
# Draw the boundary
plt.plot(x + [x[0]], y + [y[0]], marker='o')
# Title and labels
plt.title("Polygon Filling")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
# Grid
plt.grid(True)
# Equal scale
plt.axis("equal")
# Display
plt.show()
```

GATE EDUCATIONAL INSTITUTIONS TIRUPATI

Date :

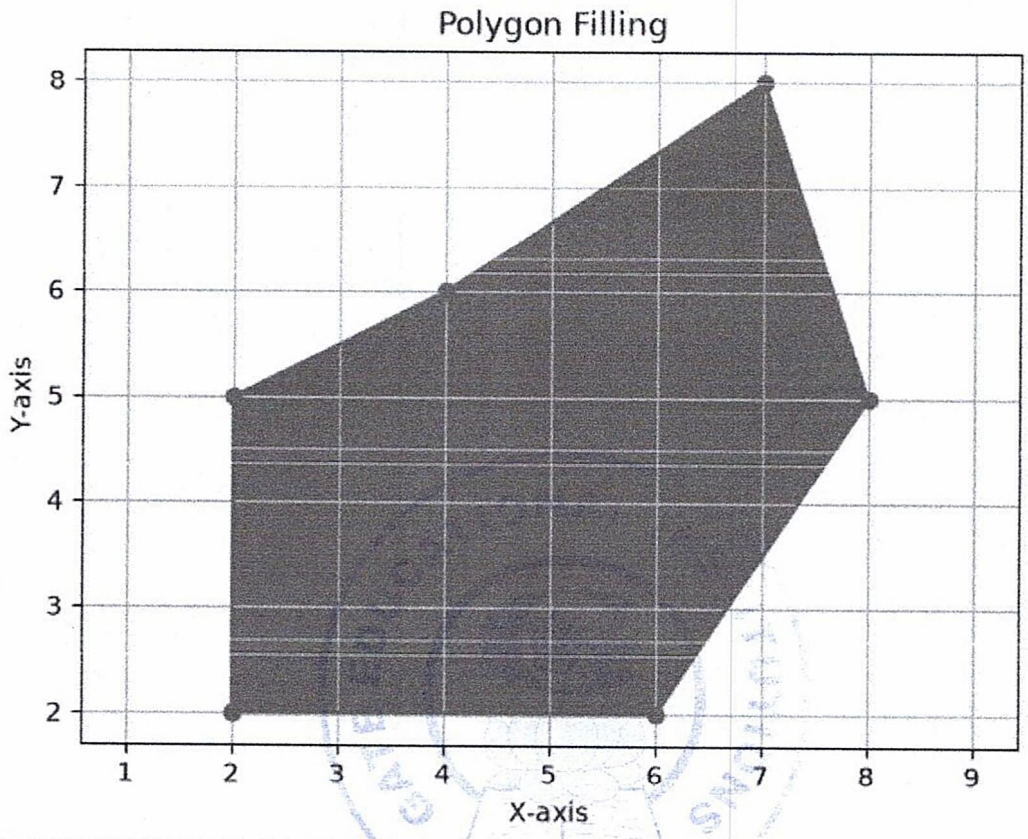
OUTPUT:

Figure 1

GATE Educational Institutions

Page No. : 61

Practical No. :



Result: The program was successfully executed and the **polygon was filled and displayed with its boundary and vertices using Python.**

Result :-

Thus, the program was successfully executed and the given polygon was filled successfully using Python.

