ANNAMALAI UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E. [COMPUTER SCIENCE AND ENGINEERING]

SEMESTER - V

CSCP508 - COMPUTER GRAPHICS AND MULTIMEDIA LAB

LABORATORY MANUAL

(July 2025 – November 2025)

LAB IN-CHARGE
Dr. G. Arulselvi, Associate Professor
Department of Computer Science and Engineering
Annamalai University

VISION

To provide a congenial ambience for individuals to develop and blossom as academically superior, socially conscious and nationally responsible citizens.

MISSION

- Impart high quality computer knowledge to the students through a dynamic scholastic environment wherein they learn to develop technical, communication and leadership skills to bloom as a versatile professional.

- Develop life-long learning ability that allows them to be adaptive and responsive to the changes in career, society, technology, and environment.

- Build student community with high ethical standards to undertake innovative research and development in thrust areas of national and international needs.

- Expose the students to the emerging technological advancements for meeting the demands of the industry.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO	PEO Statements
PEO1	To prepare the graduates with the potential to get employed in the right role and/or become entrepreneurs to contribute to the society.
PEO2	To provide the graduates with the requisite knowledge to pursue higher education and carry out research in the field of Computer Science.
PEO3	To equip the graduates with the skills required to stay motivated and adapt to the dynamically changing world so as to remain successful in their career.
PEO4	To train the graduates to communicate effectively, work collaboratively and exhibit high levels of professionalism and ethical responsibility.

LIST OF EXPERIMENTS

IMPLEMENTATION OF COMPUTER GRAPHICS USING PYTHON

1. Implementation of Line drawing Algorithm :
   - (a). Bresenham's line drawing algorithm.
   - (b). DDA (Digital Differential Analyser) Line drawing Algorithm.
2. Implementation of Circle drawing Algorithm :
   - (a). Midpoint Circle drawing Algorithm.
   - (b). Bresenham's Circle drawing Algorithm.
3. Implementation of Bresenham's Ellipse Drawing Algorithm.
4. Implementation of 2D Transformations :
   - (a). Translation, Rotation, Scaling.
   - (b). Reflection and Shearing.
5. Implementation of 2D Line Clipping Algorithm :
   - (a). Cohen-Sutherland Algorithm.
   - (b). Liang-Barsky Algorithm.
6. Polygon clipping using Sutherland-Hodgeman Algorithm.
7. Implementation of 3D Transformations - Translation,Rotation,Scaling.
8. Implementation of 2D Animation (using Timer,Loop,simple animation):
   - (a). Bouncing Ball.
   - (b). Car movement.
9. Implementation of 3D Animation - Human Facial Expressions:
   - (a) Smile. (b) Sad. (c) Surprise.
10. Drawing Three Dimensional Objects and Scenes using OpenGL.

MULTIMEDIA

I. GIMP

1. Implementation of Logo Creation
2. Implementation of Text Animation

II. AUDACITY

1. Audio Signal Processing: Silencing, Trimming, and Duplicating
2. Applying Advanced Effects to Audio Signals

III. WINDOWS MOVIE MAKER

1. Applying Visual Effects to Videos
2. Creating and Adding Titles in Video Clips

IV. SWISH

1. Implementation of Dynamic Text Effects
2. Designing a Pre-Loader Animation

V. FLASH

1. Transforming Object Shapes
2. Implementing Image Masking for Viewing Effects

VI. PHOTO IMPACT

1. Advanced Text Effects Implementation
2. Image Slicing and Segmentation Techniques

LAB INCHARGES:

A Batch – Dr. G. ARULSELVI

B Batch – Dr. A. KANTHIMATHINATHAN

Head of the Department

Course Outcomes:

At the end of this course, the students will be able to

1. Implement 2D and 3D shape drawing algorithms, transformations and its applications.
2. Develop applications on image, sound and video using editing tools such as GIMP, Audacity, Windows Movie Maker, Swish, Flash, etc.
3. Demonstrate an ability to listen and answer the viva questions related to programming skills needed for solving real-world problems in Computer Science and Engineering

Mapping of Course Outcomes with Programme Outcomes												
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
CO1	3	2	1	-	1	-	-	-	-	-	-	-
CO2	1	1	3	1	3	-	-	-	-	-	-	-
CO3	2	2	-	-	-	-	-	-	-	2	-	2

Rubric for CO3 in Laboratory Courses					
Rubric	**Distribution of 10 Marks for CIE/SEE Evaluation Out of 40/60 Marks**				
	Up To 2.5 Marks	**Up To 5 Marks**	**Up To 7.5 Marks**	**Up To 10 marks**	**Up To 2.5 Marks**
Demonstrate an ability to listen and answer the viva questions related to programming skills needed for solving real-world problems in Computer Science and Engineering.	Poor listening and communication skills. Failed to relate the programming skills needed for solving the problem.	Showed better communication skill by relating the problem with the programming skills acquired but the description showed serious errors.	Demonstrated good communication skills by relating the problem with the programming skills acquired with few errors.	Demonstrated excellent communication skills by relating the problem with the programming skills acquired and have been successful in tailoring the description.	Demonstrate an ability to listen and answer the Viva Questions related to programming skills needed for solving real-world problems in Computer Science and Engineering.

TABLE OF CONTENTS

GRAPHICS

GIMP

AUDACITY

WINDOWS MOVIE MAKER

SWISH

FLASH

PHOTO IMPACT

EX.NO.1A	IMPLEMENTATION OF LINE DRAWING ALGORITHM DDA (DIGITAL DIFFERENTIAL ANALYSER) LINE

AIM:

To draw a line between two points using DDA Line Drawing Algorithm in Python.

ALGORITHM:

1. Start
2. Accept the two endpoints of the line.
3. Calculate the number of steps using dx and dy.
4. Calculate increment values for x and y.
5. Starting from the first point, keep adding the increments and plot each point.
6. Stop

PROGRAM:

```
import tkinter as tk

print("--- LINE TYPE MENU ---")
print("1. Horizontal Line")
print("2. Vertical Line")
print("3. Forward Slanting (/)")
print("4. Backward Slanting (\\)")
choice = input("Enter your line type choice (1-4): ")

print("\nEnter starting point:")
x1 = int(input("x1: "))
y1 = int(input("y1: "))
length = int(input("Enter length of line: "))

if choice == '1':
    x2 = x1 + length
    y2 = y1
    label = "Horizontal Line"
elif choice == '2':
    x2 = x1
    y2 = y1 + length
    label = "Vertical Line"
elif choice == '3':
    x2 = x1 + length
    y2 = y1 - length
```

```
    label = "Forward Slanting (/)"
elif choice == '4':
    x2 = x1 + length
    y2 = y1 + length
    label = "Backward Slanting (\\)"
else:
    print("Invalid choice")
    exit()

print("\nColor Options: 1. Red  2. Green  3. Blue  4. Black  5. Magenta")
color_choice = input("Choose color (1-5): ")
color_map = {'1': 'red', '2': 'green', '3': 'blue', '4': 'black', '5': 'magenta'}
color = color_map.get(color_choice, 'black')

print("\nPattern Options: 1. Solid  2. Dashed  3. Dotted")
pattern_choice = input("Choose pattern (1-3): ")
pattern_map = {'1': 'solid', '2': 'dashed', '3': 'dotted'}
pattern = pattern_map.get(pattern_choice, 'solid')

root = tk.Tk()
root.title("DDA Line Drawing - No Functions")
canvas = tk.Canvas(root, width=600, height=600, bg="white")
canvas.pack()

canvas.create_text(300, 20, text=label, fill=color, font=("Helvetica", 16, "bold"))

dx = x2 - x1
dy = y2 - y1
steps = abs(dx) if abs(dx) > abs(dy) else abs(dy)
x_inc = dx / steps
y_inc = dy / steps

x = x1
y = y1

i = 0
while i <= steps:
    xi = round(x)
    yi = round(y)

    if pattern == 'solid':
        canvas.create_rectangle(xi, yi, xi+1, yi+1, outline=color)
```

```
        elif pattern == 'dashed':
            if i % 15 < 10:
                canvas.create_rectangle(xi, yi, xi+1, yi+1, outline=color)
        elif pattern == 'dotted':
            if i % 6 == 0:
                canvas.create_rectangle(xi, yi, xi+1, yi+1, outline=color)

        x += x_inc
        y += y_inc
        i += 1

    canvas.create_oval(x1 - 3, y1 - 3, x1 + 3, y1 + 3, fill=color)
    canvas.create_oval(x2 - 3, y2 - 3, x2 + 3, y2 + 3, fill=color)

    canvas.create_text(x1 - 20, y1 - 10, text=f"({x1},{y1})", fill=color, font=("Arial", 10))
    canvas.create_text(x2 + 20, y2 + 10, text=f"({x2},{y2})", fill=color, font=("Arial", 10))
    root.mainloop()
```

OUTPUT:

--- LINE TYPE MENU ---

1. Horizontal Line
2. Vertical Line
3. Forward Slanding (/)
4. Backward Slanding (\)

Enter your line type choice (1-4): 1

Enter starting point:

x1: 100

y1: 200

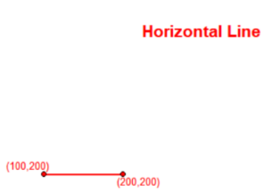
Enter length of line: 200

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta

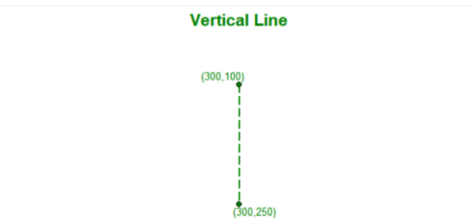
Choose color (1-5): 1

Pattern Options: 1. Solid 2. Dashed 3. Dotted

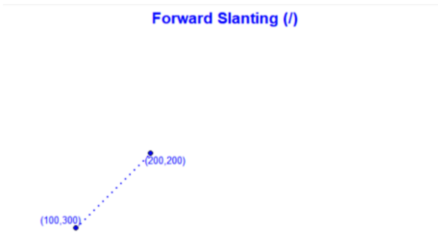
Choose pattern (1-3): 1



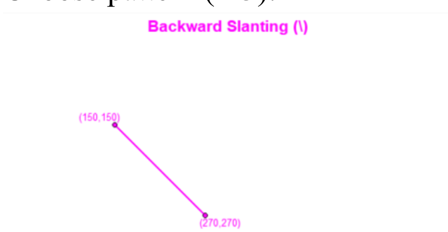
--- LINE TYPE MENU ---
1. Horizontal Line
2. Vertical Line
3. Forward Slanding (/)
4. Backward Slanding (\)
Enter your line type choice (1-4): 2
Enter starting point:
x1: 300
y1: 100
Enter length of line: 150
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 2
Pattern Options: 1. Solid 2. Dashed 3. Dotted
Choose pattern (1-3): 2



--- LINE TYPE MENU ---
1. Horizontal Line
2. Vertical Line
3. Forward Slanding (/)
4. Backward Slanding (\)
Enter your line type choice (1-4): 3
Enter starting point:
x1: 100
y1: 300
Enter length of line: 100
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 2
Pattern Options: 1. Solid 2. Dashed 3. Dotted
Choose pattern (1-3): 3



--- LINE TYPE MENU ---
1. Horizontal Line
2. Vertical Line
3. Forward Slanding (/)
4. Backward Slanding (\)
Enter your line type choice (1-4): 4
Enter starting point:
x1: 150
y1: 150
Enter length of line: 120
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 5
Pattern Options: 1. Solid 2. Dashed 3. Dotted
Choose pattern (1-3): 1



Result:

Thus the Python program to draw a line using DDA algorithm is implemented and executed successfully

EX.NO.1.B	IMPLEMENTATION OF LINE DRAWING ALGORITHM BRESENHAM'S LINE DRAWING ALGORITHM

AIM:

To draw a Line using Bresenham's Line drawing algorithm using Python program.

ALGORITHM:

1. Start
2. Accept the two endpoints of the line.
3. Initialize decision parameters to choose the next pixel.
4. Use only integer addition to move in the closest pixel direction.
5. Plot each point from start to end based on the decision value.
6. Stop

PROGRAM:

```
import tkinter as tk


print("--- LINE TYPE MENU ---")
print("1. Horizontal Line")
print("2. Vertical Line")
print("3. Forward Slanting (/)")
print("4. Backward Slanting (\\)")

choice = input("Enter your line type choice (1-4): ")

print("\nEnter Starting Point:")
x0 = int(input("x0: "))
y0 = int(input("y0: "))
length = int(input("Enter length of line: "))

print("\nColor Options: 1. Red  2. Green  3. Blue  4. Black  5. Magenta")
color_choice = input("Choose color (1-5): ")
color_map = {'1': 'red', '2': 'green', '3': 'blue', '4': 'black', '5': 'magenta'}
color = color_map.get(color_choice, 'black')
```

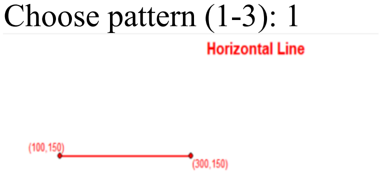
```
print("\nPattern Options: 1. Solid  2. Dashed  3. Dotted")
pattern_choice = input("Choose pattern (1-3): ")
pattern_map = {'1': 'solid', '2': 'dashed', '3': 'dotted'}
pattern = pattern_map.get(pattern_choice, 'solid')

if choice == '1':
    x1 = x0 + length
    y1 = y0
    label = "Horizontal Line"
elif choice == '2':
    x1 = x0
    y1 = y0 + length
    label = "Vertical Line"
elif choice == '3':
    x1 = x0 + length
    y1 = y0 - length
    label = "Forward Slanting (/)"
elif choice == '4':
    x1 = x0 + length
    y1 = y0 + length
    label = "Backward Slanting (\\)"
else:
    print("Invalid line type choice!")
    exit(

root = tk.Tk()
root.title("Bresenham Line Drawing - With Color & Pattern")
canvas = tk.Canvas(root, width=800, height=800, bg="white")
canvas.pack()

canvas.create_text(400, 20, text=label, fill=color, font=("Helvetica", 16, "bold"))
```

```
dx = abs(x1 - x0)
dy = abs(y1 - y0)
sx = 1 if x0 < x1 else -1
sy = 1 if y0 < y1 else -1
err = dx - dy

x = x0
y = y0
step_count = 0

while True:
    # Apply pattern
    if pattern == 'solid':
        canvas.create_rectangle(x, y, x + 1, y + 1, outline=color)
    elif pattern == 'dashed':
        if step_count % 15 < 10:
            canvas.create_rectangle(x, y, x + 1, y + 1, outline=color)
    elif pattern == 'dotted':
        if step_count % 6 == 0:
            canvas.create_rectangle(x, y, x + 1, y + 1, outline=color)

    if x == x1 and y == y1:
        break

    e2 = 2 * err
    if e2 > -dy:
        err -= dy
        x += sx
    if e2 < dx:
        err += dx
        y += sy
```

```
    step_count += 1

canvas.create_oval(x0 - 3, y0 - 3, x0 + 3, y0 + 3, fill=color)
canvas.create_oval(x1 - 3, y1 - 3, x1 + 3, y1 + 3, fill=color)

canvas.create_text(x0 - 20, y0 - 10, text=f"({x0},{y0})", fill=color, font=("Arial", 10))
canvas.create_text(x1 + 30, y1 + 10, text=f"({x1},{y1})", fill=color, font=("Arial", 10))
root.mainloop()
```

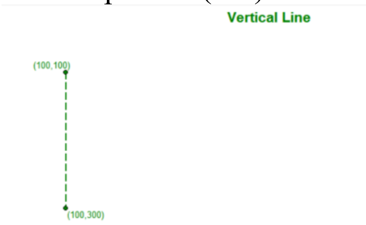
SAMPLE INPUT/OUTPUT:

```
--- LINE TYPE MENU ---
1. Horizontal Line
2. Vertical Line
3. Forward Slanding (/)
4. Backward Slanding (\)
Enter your line type choice (1-4): 1
Enter Starting Point:
x1: 100
y1: 150
Enter length of line: 200
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 1
Pattern Options: 1. Solid 2. Dashed 3. Dotted
Choose pattern (1-3): 1
```

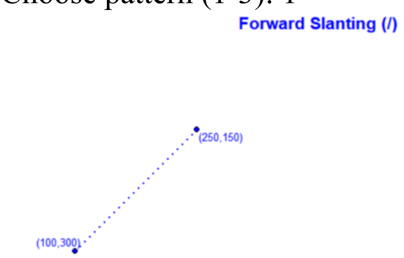



```
--- LINE TYPE MENU ---
1. Horizontal Line
2. Vertical Line
3. Forward Slanding (/)
4. Backward Slanding (\)
Enter your line type choice (1-4): 2
Enter Starting Point:
x1: 100
y1: 100
Enter length of line: 200
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 2
Pattern Options: 1. Solid 2. Dashed 3. Dotted
```

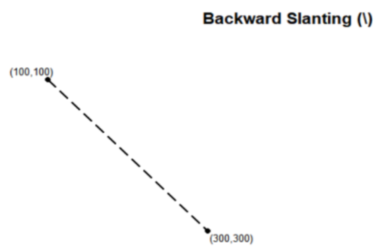
Choose pattern (1-3): 3



--- LINE TYPE MENU ---
1. Horizontal Line
2. Vertical Line
3. Forward Slanding (/)
4. Backward Slanding (\)
Enter your line type choice (1-4): 3
Enter Starting Point:
x1: 100
y1: 300
Enter length of line: 150
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 4
Pattern Options: 1. Solid 2. Dashed 3. Dotted
Choose pattern (1-3): 1



--- LINE TYPE MENU ---
1. Horizontal Line
2. Vertical Line
3. Forward Slanding (/)
4. Backward Slanding (\)
Enter your line type choice (1-4): 4
Enter Starting Point:
x1: 100
y1: 100
Enter length of line: 200
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 4
Pattern Options: 1. Solid 2. Dashed 3. Dotted
Choose pattern (1-3): 2



RESULT:

Thus the Python program to draw a line using Bresenham's algorithm is implemented and executed successfully.

EX.NO.2(a)	Implementation of Midpoint Circle drawing Algorithm.

AIM:

To draw an Circle using Midpoint drawing algorithm using Python progam.

ALGORITHM:

1. Start
2. Accept radius and center point.
3. Start at the top of the circle (0, r).
4. Use a decision parameter to find the next pixel.
5. Use symmetry to draw eight points at once.
6. Continue until x > y.
7. Stop

PROGRAM:

```
import tkinter as tk

import math


def draw_point(canvas, xc, yc, x, y, color):

    for dx, dy in [(x, y), (-x, y), (x, -y), (-x, -y),

                   (y, x), (-y, x), (y, -x), (-y, -x)]:

        canvas.create_oval(xc+dx, yc+dy, xc+dx+1, yc+dy+1, fill=color, outline=color)


def midpoint_circle(canvas, xc, yc, r, color):

    x = 0

    y = r

    p = 1 - r

    draw_point(canvas, xc, yc, x, y, color)


    while x < y:

        x += 1
```

```
        if p < 0:
            p += 2 * x + 1
        else:
            y -= 1
            p += 2 * (x - y) + 1
        draw_point(canvas, xc, yc, x, y, color)


def draw_internal_pattern(canvas, xc, yc, r, pattern, color):
    if pattern == 'radial':
        for angle in range(0, 360, 15):
            rad = math.radians(angle)
            x = xc + int(r * math.cos(rad))
            y = yc + int(r * math.sin(rad))
            canvas.create_line(xc, yc, x, y, fill=color)

    elif pattern == 'horizontal':
        for dy in range(-r, r+1, 10):
            y = yc + dy
            span = int((r**2 - dy**2) ** 0.5)
            canvas.create_line(xc - span, y, xc + span, y, fill=color)

    elif pattern == 'vertical':
        for dx in range(-r, r+1, 10):
            x = xc + dx
            span = int((r**2 - dx**2) ** 0.5)
            canvas.create_line(x, yc - span, x, yc + span, fill=color)

    elif pattern == 'cross':
```

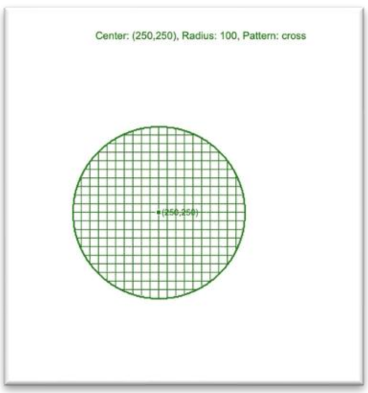
```
        draw_internal_pattern(canvas, xc, yc, r, 'horizontal', color)
        draw_internal_pattern(canvas, xc, yc, r, 'vertical', color)

def get_user_input():
    print("\n--- Midpoint Circle with Internal Patterns ---")
    xc = int(input("Enter center X: "))
    yc = int(input("Enter center Y: "))
    r = int(input("Enter radius: "))

    print("\nColor Options: 1. Red  2. Green  3. Blue  4. Black  5. Magenta")
    color_choice = input("Choose color (1-5): ")
    color_map = {'1': 'red', '2': 'green', '3': 'blue', '4': 'black', '5': 'magenta'}
    color = color_map.get(color_choice, 'black')

    print("\nPattern Inside Circle: 1. None  2. Radial  3. Horizontal  4. Vertical  5. Cross")
    pattern_choice = input("Choose internal pattern (1-5): ")
    pattern_map = {'1': 'none', '2': 'radial', '3': 'horizontal', '4': 'vertical', '5': 'cross'}
    pattern = pattern_map.get(pattern_choice, 'none')

    return xc, yc, r, color, pattern

def draw_circle_with_pattern(xc, yc, r, color, pattern):
    root = tk.Tk()
    root.title("Circle with Patterns Inside")
    canvas = tk.Canvas(root, width=600, height=600, bg="white")
    canvas.pack()

    canvas.create_text(300, 20, text="Midpoint Circle with Patterns", font=("Helvetica", 16, "bold"))
```

```
    canvas.create_text(300, 45, text=f"Center: ({xc},{yc}), Radius: {r}, Pattern: {pattern}", font=("Arial",
12), fill=color)

    midpoint_circle(canvas, xc, yc, r, color)

    if pattern != 'none':
        draw_internal_pattern(canvas, xc, yc, r, pattern, color)

    canvas.create_oval(xc - 3, yc - 3, xc + 3, yc + 3, fill=color)
    canvas.create_text(xc + 25, yc, text=f"({xc},{yc})", font=("Arial", 10), fill=color)

    root.mainloop()

# Main loop
while True:
    user_input = get_user_input()
    draw_circle_with_pattern(*user_input)
    again = input("\nDraw another circle? (y/n): ")
    if again.lower() != 'y':
        break
```

OUTPUT:
--- MidPoint Circle with Patterns ---
Enter center X: 250
Enter center Y: 250
Enter radius: 100

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 2

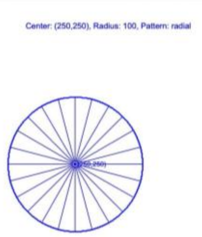
Pattern Inside Circle: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 5



--- MidPoint Circle with Patterns ---
Enter center X: 250
Enter center Y: 250
Enter radius: 100

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 3

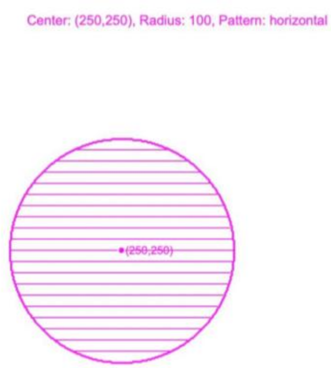
Pattern Inside Circle: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 2



--- MidPoint Circle with Patterns ---
Enter center X: 250
Enter center Y: 250
Enter radius: 100

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 5

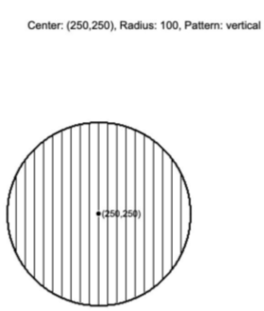
Pattern Inside Circle: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 3



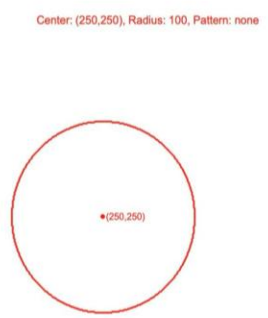
--- MidPoint Circle with Patterns ---
Enter center X: 250
Enter center Y: 250
Enter radius: 100

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 4

Pattern Inside Circle: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 4



```
--- MidPoint Circle with Patterns ---
 Enter center X: 250
 Enter center Y: 250
 Enter radius: 100

 Color Options: 1. Red  2. Green  3. Blue  4. Black  5. Magenta
 Choose color (1-5): 1

 Pattern Inside Circle: 1. None  2. Radial  3. Horizontal  4. Vertical  5. Cross
Choose internal pattern (1-5): 1
```



RESULT:

Thus the program to draw an Circle using MidPoint Drawing algorithm is implemented and executed successfully.

EX.NO.2(b)	Implementation of Bresenham’s Circle drawing Algorithm.

AIM:

To draw an Circle using Bresenham’s drawing algorithm using Python progam.

ALGORITHM:

1. Start
2. Accept radius and center coordinates.
3. Initialize variables to track the circle's edge.
4. Use integer decision parameters to determine next pixel.
5. Plot all 8 symmetric points.
6. Loop until x > y.
7. Stop

PROGRAM:

```
import tkinter as tk
import math
print("\n--- Bresenham Circle with Patterns ---")
xc = int(input("Enter center X: "))
yc = int(input("Enter center Y: "))
r = int(input("Enter radius: "))

print("\nColor Options: 1. Red  2. Green  3. Blue  4. Black  5. Magenta")
color_choice = input("Choose color (1-5): ")
color_map = {'1': 'red', '2': 'green', '3': 'blue', '4': 'black', '5': 'magenta'}
color = color_map.get(color_choice, 'black')

print("\nPattern Inside Circle: 1. None  2. Radial  3. Horizontal  4. Vertical  5. Cross")
pattern_choice = input("Choose internal pattern (1-5): ")
pattern_map = {'1': 'none', '2': 'radial', '3': 'horizontal', '4': 'vertical', '5': 'cross'}
pattern = pattern_map.get(pattern_choice, 'none')
```

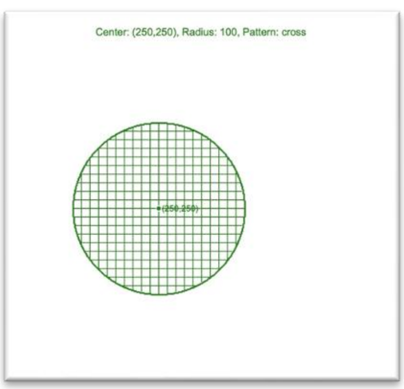
```
root = tk.Tk()
root.title("Bresenham Circle with Patterns")
canvas = tk.Canvas(root, width=600, height=600, bg="white")
canvas.pack()

x = 0
y = r
d = 3 - 2 * r

points = [(x, y), (-x, y), (x, -y), (-x, -y),
          (y, x), (-y, x), (y, -x), (-y, -x)]
for dx, dy in points:
    canvas.create_oval(xc + dx, yc + dy, xc + dx + 1, yc + dy + 1, fill=color, outline=color)

while x <= y:
    x += 1
    if d <= 0:
        d = d + 4 * x + 6
    else:
        y -= 1
        d = d + 4 * (x - y) + 10

    points = [(x, y), (-x, y), (x, -y), (-x, -y),
              (y, x), (-y, x), (y, -x), (-y, -x)]
    for dx, dy in points:
        canvas.create_oval(xc + dx, yc + dy, xc + dx + 1, yc + dy + 1, fill=color, outline=color)
```

```
if pattern == 'radial':
    for angle in range(0, 360, 15):
        rad = math.radians(angle)
        x1 = xc + int(r * math.cos(rad))
        y1 = yc + int(r * math.sin(rad))
        canvas.create_line(xc, yc, x1, y1, fill=color)

elif pattern == 'horizontal':
    for dy in range(-r, r+1, 10):
        y = yc + dy
        span = int((r**2 - dy**2) ** 0.5)
        canvas.create_line(xc - span, y, xc + span, y, fill=color)

elif pattern == 'vertical':
    for dx in range(-r, r+1, 10):
        x = xc + dx
        span = int((r**2 - dx**2) ** 0.5)
        canvas.create_line(x, yc - span, x, yc + span, fill=color)

elif pattern == 'cross':

    for dy in range(-r, r+1, 10):
        y = yc + dy
        span = int((r**2 - dy**2) ** 0.5)
        canvas.create_line(xc - span, y, xc + span, y, fill=color)

    for dx in range(-r, r+1, 10):
        x = xc + dx
```

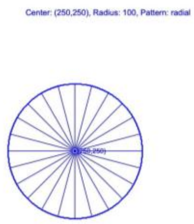
```
        span = int((r**2 - dx**2) ** 0.5)
        canvas.create_line(x, yc - span, x, yc + span, fill=color)


canvas.create_oval(xc - 3, yc - 3, xc + 3, yc + 3, fill=color)
canvas.create_text(xc + 25, yc, text=f"({xc},{yc})", font=("Arial", 10), fill=color)


root.mainloop()
```

OUTPUT:

```
--- Bresenham's Circle with Patterns ---
Enter center X: 250
Enter center Y: 250
Enter radius: 100

Color Options: 1. Red  2. Green  3. Blue  4. Black  5. Magenta
Choose color (1-5): 2

Pattern Inside Circle: 1. None  2. Radial  3. Horizontal  4. Vertical  5. Cross
Choose internal pattern (1-5): 5
```



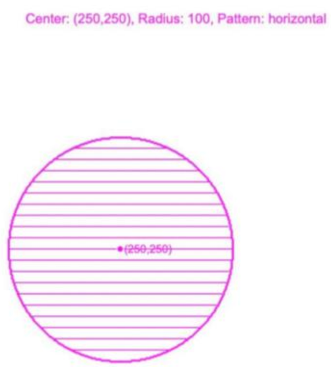

```
--- Bresenham's Circle with Patterns ---
Enter center X: 250
Enter center Y: 250
Enter radius: 100

Color Options: 1. Red  2. Green  3. Blue  4. Black  5. Magenta
Choose color (1-5): 3

Pattern Inside Circle: 1. None  2. Radial  3. Horizontal  4. Vertical  5. Cross
Choose internal pattern (1-5): 2
```



---Bresenham’s Circle with Patterns ---
Enter center X: 250
Enter center Y: 250
Enter radius: 100

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 5

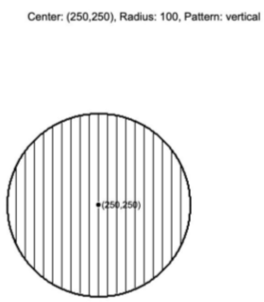
Pattern Inside Circle: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 3



--- Bresenham’s Circle with Patterns ---
Enter center X: 250
Enter center Y: 250
Enter radius: 100

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 4

Pattern Inside Circle: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 4



--- Bresenham’s Circle with Patterns ---
Enter center X: 250
Enter center Y: 250
Enter radius: 100

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 1

Pattern Inside Circle: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 1

RESULT:

Thus the program to draw an Circle using Bresenham’s Drawing algorithm is implemented and executed successfully.

EX.NO.3	Implementation of Bresenham's Ellipse Drawing Algorithm.

AIM:

To draw a Ellipse Using a Bresenham’s ellipse drawing algorithm.

ALGORITHM:

1. Start
2. Accept center and radii along X and Y axes.
3. Divide the drawing into two regions (slope < 1 and > 1).
4. Use decision parameters to move across pixels in both regions.
5. Plot symmetric points in all 4 quadrants.
6. Continue until full ellipse is drawn.
7. Stop

PROGRAM:

```
import tkinter as tk
import math

def draw_ellipse_points(canvas, xc, yc, x, y, color):
    points = [(x, y), (-x, y), (x, -y), (-x, -y)]
    for dx, dy in points:
        canvas.create_oval(xc + dx, yc + dy, xc + dx + 1, yc + dy + 1, fill=color, outline=color)

def bresenham_ellipse(canvas, xc, yc, rx, ry, color):
    x = 0
    y = ry

    rx2 = rx * rx
    ry2 = ry * ry
    two_rx2 = 2 * rx2
    two_ry2 = 2 * ry2

    px = 0
    py = two_rx2 * y
    p1 = ry2 - (rx2 * ry) + (0.25 * rx2)

    while px < py:
        draw_ellipse_points(canvas, xc, yc, x, y, color)
```

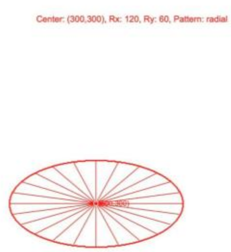
```
        x += 1
        px += two_ry2
        if p1 < 0:
            p1 += ry2 + px
        else:
            y -= 1
            py -= two_rx2
            p1 += ry2 + px - py

    p2 = ry2 * (x + 0.5)**2 + rx2 * (y - 1)**2 - rx2 * ry2

    while y >= 0:
        draw_ellipse_points(canvas, xc, yc, x, y, color)
        y -= 1
        py -= two_rx2
        if p2 > 0:
            p2 += rx2 - py
        else:
            x += 1
            px += two_ry2
            p2 += rx2 - py + px

def draw_internal_pattern(canvas, xc, yc, rx, ry, pattern, color):
    if pattern == 'radial':
        for angle in range(0, 360, 15):
            rad = math.radians(angle)
            x = xc + int(rx * math.cos(rad))
            y = yc + int(ry * math.sin(rad))
            canvas.create_line(xc, yc, x, y, fill=color)
    elif pattern == 'horizontal':
        for dy in range(-ry, ry + 1, 10):
            y = yc + dy
            span = int(rx * ((1 - (dy / ry) ** 2) ** 0.5))
            canvas.create_line(xc - span, y, xc + span, y, fill=color)
    elif pattern == 'vertical':
        for dx in range(-rx, rx + 1, 10):
            x = xc + dx
            span = int(ry * ((1 - (dx / rx) ** 2) ** 0.5))
            canvas.create_line(x, yc - span, x, yc + span, fill=color)
    elif pattern == 'cross':
        draw_internal_pattern(canvas, xc, yc, rx, ry, 'horizontal', color)
        draw_internal_pattern(canvas, xc, yc, rx, ry, 'vertical', color)
```

```
def get_user_input():
    xc = int(input("Enter center X: "))
    yc = int(input("Enter center Y: "))
    rx = int(input("Enter X radius (horizontal): "))
    ry = int(input("Enter Y radius (vertical): "))

    print("Color Options: 1. Red  2. Green  3. Blue  4. Black  5. Magenta")
    color_choice = input("Choose color (1-5): ")
    color_map = {'1': 'red', '2': 'green', '3': 'blue', '4': 'black', '5': 'magenta'}
    color = color_map.get(color_choice, 'black')

    print("Pattern Inside Ellipse: 1. None  2. Radial  3. Horizontal  4. Vertical  5. Cross")
    pattern_choice = input("Choose internal pattern (1-5): ")
    pattern_map = {'1': 'none', '2': 'radial', '3': 'horizontal', '4': 'vertical', '5': 'cross'}
    pattern = pattern_map.get(pattern_choice, 'none')

    return xc, yc, rx, ry, color, pattern

def draw_bresenham_ellipse_with_pattern(xc, yc, rx, ry, color, pattern):
    root = tk.Tk()
    root.title("Bresenham Ellipse with Patterns")
    canvas = tk.Canvas(root, width=700, height=700, bg="white")
    canvas.pack()

    bresenham_ellipse(canvas, xc, yc, rx, ry, color)

    if pattern != 'none':
        draw_internal_pattern(canvas, xc, yc, rx, ry, pattern, color)

    canvas.create_oval(xc - 3, yc - 3, xc + 3, yc + 3, fill=color)
    canvas.create_text(xc + 25, yc, text=f"({xc},{yc})", font=("Arial", 10), fill=color)

    root.mainloop()

while True:
    details = get_user_input()
    draw_bresenham_ellipse_with_pattern(*details)
    again = input("Draw another ellipse? (y/n): ")
    if again.lower() != 'y':
        break
```

OUTPUT:
--- Bresenham Ellipse Drawing with Patterns ---
Enter center X: 300
Enter center Y: 300
Enter X radius (horizontal): 120
Enter Y radius (vertical): 60

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 1

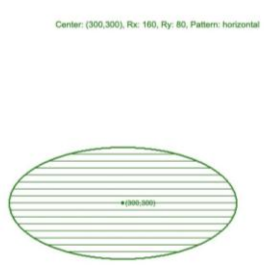
Pattern Inside Ellipse: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 2



--- Bresenham Ellipse Drawing with Patterns ---
Enter center X: 300
Enter center Y: 300
Enter X radius (horizontal): 120
Enter Y radius (vertical): 80

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 2

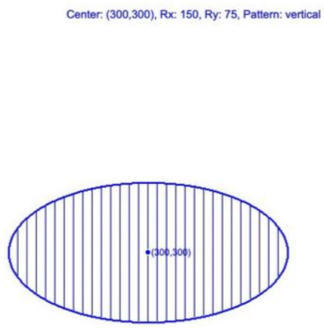
Pattern Inside Ellipse: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 3



--- Bresenham Ellipse Drawing with Patterns ---
Enter center X: 300
Enter center Y: 300
Enter X radius (horizontal): 150
Enter Y radius (vertical): 75

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 3

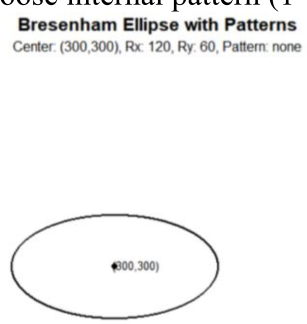
Pattern Inside Ellipse: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 4



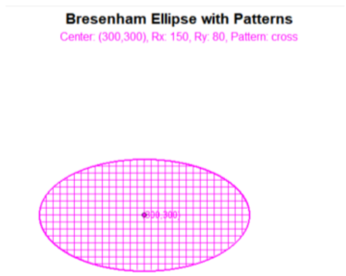
-- Bresenham Ellipse Drawing with Patterns ---
Enter center X: 300
Enter center Y: 300
Enter X radius (horizontal): 120
Enter Y radius (vertical): 60

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 4

Pattern Inside Ellipse: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 1



--- Bresenham Ellipse Drawing with Patterns ---
Enter center X: 300
Enter center Y: 300
Enter X radius (horizontal): 150
Enter Y radius (vertical): 80
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 5
Pattern Inside Ellipse: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 5



RESULT:

Thus the program to draw line, circle and ellipse attributes is implemented and executed successfully.

EX.NO.4	TWO DIMENSIONAL TRANSFORMATIONS - TRANSLATION, ROTATION, SCALING, REFLECTION AND SHEAR

AIM:

To implement 2D transformations like Translation, Rotation, Scaling, Reflection, Shear using Python program.

ALGORITHM:

1. Start
2. Input the object's original coordinates.
3. Apply translation by adding Tx and Ty.
4. Apply scaling by multiplying with Sx and Sy.
5. Apply rotation using angle θ.
6. Reflect across X, Y, or origin by changing signs.
7. Apply shearing using shearing factors.
8. Display the transformed object.
9. Stop

PROGRAM:

```
import tkinter as tk
import math

root = tk.Tk()
root.title(" ")
canvas = tk.Canvas(root, width=600, height=600, bg="white")
canvas.pack()

x_offset = 300
y_offset = 300

canvas.create_text(300, 10, text="2D TRANSFORMATIONS", fill="black", font=("Arial", 14, "bold"))
canvas.create_line(0, y_offset, 600, y_offset, fill="gray")  # X-axis
canvas.create_line(x_offset, 0, x_offset, 600, fill="gray")  # Y-axis

original_points = [[100, 100], [200, 100], [150, 200]]

print("--- 2D Transformation Menu ---")
print("1. Translation")
print("2. Rotation")
print("3. Scaling")
print("4. Shearing")
print("5. Reflection")
```

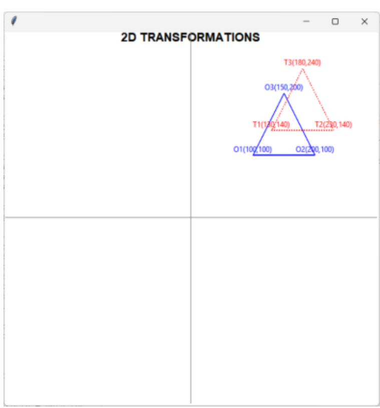
```python
choice = int(input("Enter your choice (1-5): "))

if choice == 1:
    print("--- TRANSLATION ---")
    tx = int(input("Enter translation in X (tx): "))
    ty = int(input("Enter translation in Y (ty): "))
elif choice == 2:
    print("--- ROTATION ---")
    angle = float(input("Enter angle (in degrees): "))
    theta = math.radians(angle)
elif choice == 3:
    print("--- SCALING ---")
    sx = float(input("Enter scaling in X (sx): "))
    sy = float(input("Enter scaling in Y (sy): "))
elif choice == 4:
    print("--- SHEARING ---")
    shx = float(input("Enter shearing in X (shx): "))
    shy = float(input("Enter shearing in Y (shy): "))
elif choice == 5:
    print("--- REFLECTION ---")
    axis = input("Reflect about X or Y axis? (x/y): ").lower()

transformed_points = []
for x, y in original_points:
    if choice == 1:  # Translation
        xt = x + tx
        yt = y + ty
    elif choice == 2:  # Rotation (about origin)
        xt = round(x * math.cos(theta) - y * math.sin(theta))
        yt = round(x * math.sin(theta) + y * math.cos(theta))
    elif choice == 3:  # Scaling
        xt = x * sx
        yt = y * sy
    elif choice == 4:  # Shearing
        xt = x + y * shx
        yt = y + x * shy
    elif choice == 5:  # Reflection
        if axis == 'x':
            xt = x
            yt = -y
        elif axis == 'y':
            xt = -x
            yt = y
        else:
            xt, yt = x, y  # Invalid axis input fallback
    transformed_points.append([xt, yt])
```

```
def to_canvas_coords(x, y):
    return x + x_offset, y_offset - y

for i in range(3):
    x1, y1 = to_canvas_coords(*original_points[i])
    x2, y2 = to_canvas_coords(*original_points[(i+1)%3])
    canvas.create_line(x1, y1, x2, y2, fill="blue", width=2)
    canvas.create_text(x1, y1 - 10, text=f"O{i+1}({original_points[i][0]},{original_points[i][1]})", fill="blue")

for i in range(3):
    x1, y1 = to_canvas_coords(*transformed_points[i])
    x2, y2 = to_canvas_coords(*transformed_points[(i+1)%3])
    canvas.create_line(x1, y1, x2, y2, fill="red", dash=(4, 2), width=2)
    canvas.create_text(x1, y1 - 10, text=f"T{i+1}({transformed_points[i][0]},{transformed_points[i][1]})",
fill="red")

root.mainloop()
```

OUTPUT:

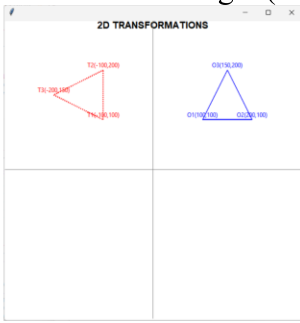
--- 2D Transformation Menu ---
1. Translation
2. Rotation
3. Scaling
4. Shearing
5. Reflection

Enter your choice (1-5): 1
Enter translation in X (tx): 30
Enter translation in Y (ty): 40

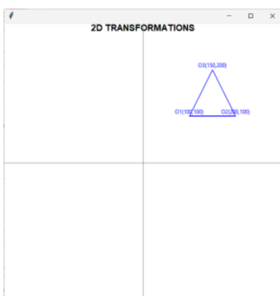


--- 2D Transformation Menu ---
1. Translation
2. Rotation
3. Scaling
4. Shearing
5. Reflection

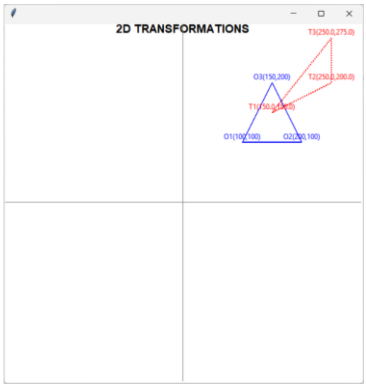
Enter your choice (1-5): 2
Enter rotation angle (in degrees): 90



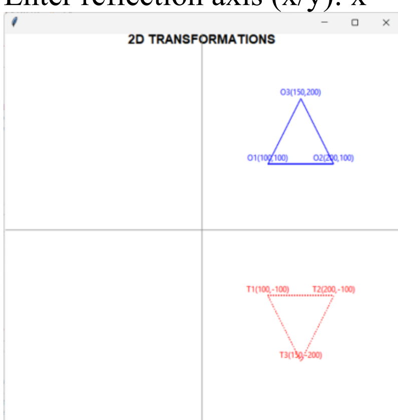
--- 2D Transformation Menu ---
1. Translation
2. Rotation
3. Scaling
4. Shearing
5. Reflection
Enter your choice (1-5): 3
Enter scaling factor in X (sx): 2
Enter scaling factor in Y (sy): 3



--- 2D Transformation Menu ---
1. Translation
2. Rotation
3. Scaling
4. Shearing
5. Reflection
Enter your choice (1-5): 4
Enter shearing factor in X (shx): 0.5
Enter shearing factor in Y (shy): 0.5



--- 2D Transformation Menu ---
1. Translation
2. Rotation
3. Scaling
4. Shearing
5. Reflection
Enter your choice (1-5): 5
Enter reflection axis (x/y): x



RESULT:

Thus the program to implement 2D transformations is executed successfully.

EX.NO.5 A	IMPLEMENTATION OF 2D LINE CLIPPING ALGORITHM - COHEN-SUTHERLAND 2D LINE CLIPPING ALGORITHM

AIM:

To implement Cohen-Sutherland 2D line clipping algorithm using Python program.

ALGORITHM:

1. Start
2. Define clipping window and input line endpoints.
3. Assign region codes to both endpoints.
4. If both codes are 0000 → accept.
5. If logical AND of codes ≠ 0000 → reject.
6. Else, calculate intersection and repeat checks.
7. Stop

PROGRAM:

```
import tkinter as tk

INSIDE, LEFT, RIGHT, BOTTOM, TOP = 0, 1, 2, 4, 8

def compute_code(x, y, x_min, y_min, x_max, y_max):
    code = INSIDE
    if x < x_min: code |= LEFT
    elif x > x_max: code |= RIGHT
    if y < y_min: code |= BOTTOM
    elif y > y_max: code |= TOP
    return code

def cohen_sutherland_clip(x1, y1, x2, y2, x_min, y_min, x_max, y_max):
    code1 = compute_code(x1, y1, x_min, y_min, x_max, y_max)
    code2 = compute_code(x2, y2, x_min, y_min, x_max, y_max)
    accept = False

    while True:
        if code1 == 0 and code2 == 0:
            accept = True
            break
        elif (code1 & code2) != 0:
```

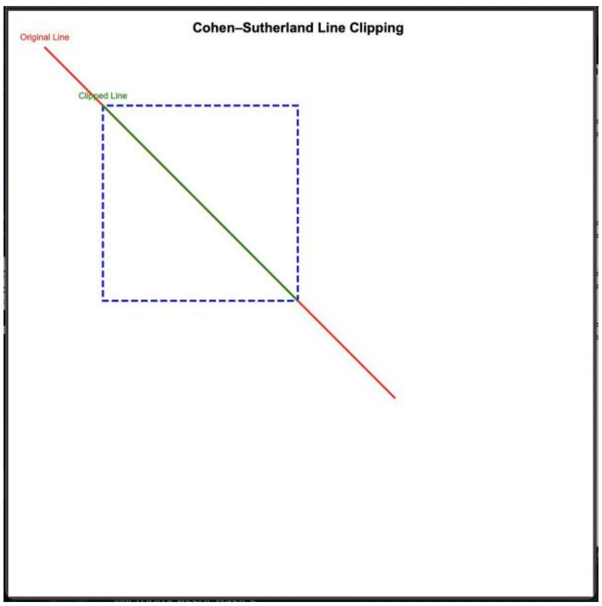
```
            break
        else:
            code_out = code1 if code1 != 0 else code2
            if code_out & TOP:
                x = x1 + (x2 - x1) * (y_max - y1) / (y2 - y1)
                y = y_max
            elif code_out & BOTTOM:
                x = x1 + (x2 - x1) * (y_min - y1) / (y2 - 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, y1 = x, y
                code1 = compute_code(x1, y1, x_min, y_min, x_max, y_max)
            else:
                x2, y2 = x, y
                code2 = compute_code(x2, y2, x_min, y_min, x_max, y_max)

    return accept, x1, y1, x2, y2

def draw_result(x1, y1, x2, y2, clipped, x_min, y_min, x_max, y_max):
    root = tk.Tk()
    root.title("Cohen-Sutherland Line Clipping")
    canvas = tk.Canvas(root, width=600, height=600, bg="white")
    canvas.pack()

    canvas.create_rectangle(x_min, y_min, x_max, y_max, outline="blue", dash=(5, 5), width=2)
    canvas.create_text(300, 20, text="Cohen–Sutherland Line Clipping", fill="black", font=("Arial", 14,
"bold"))

    canvas.create_line(orig_x1, orig_y1, orig_x2, orig_y2, fill="red", width=2)
    canvas.create_text(orig_x1, orig_y1 - 10, text="Original Line", fill="red", font=("Arial", 9))
```

```
    if clipped:
        # Clipped line in green
        canvas.create_line(x1, y1, x2, y2, fill="green", width=2)
        canvas.create_text(x1, y1 - 10, text="Clipped Line", fill="green", font=("Arial", 9))
    else:
        canvas.create_text(300, 560, text="Line Rejected (Outside Window)", fill="red", font=("Arial", 12,
"bold"))

    root.mainloop()

print("Enter Line Coordinates:")
orig_x1 = int(input("x1: "))
orig_y1 = int(input("y1: "))
orig_x2 = int(input("x2: "))
orig_y2 = int(input("y2: "))

print("\nEnter Clipping Window:")
x_min = int(input("x_min: "))
y_min = int(input("y_min: "))
x_max = int(input("x_max: "))
y_max = int(input("y_max: "))

accept, cx1, cy1, cx2, cy2 = cohen_sutherland_clip(orig_x1, orig_y1, orig_x2, orig_y2, x_min, y_min,
x_max, y_max)

draw_result(cx1, cy1, cx2, cy2, accept, x_min, y_min, x_max, y_max)
```

OUTPUT:

Enter Line Coordinates:
x1: 40
y1: 40
x2: 400
y2: 400

Enter Clipping Window:
x_min: 100
y_min: 100
x_max: 300
y_max: 300



RESULT:

Thus the program to implement Cohen-Sutherland 2D line clipping is executed successfully.

E X.NO. 5 B IMPLEMENTATION OF 2D LINE CLIPPING ALGORITHM: LIANG-BARSKY ALGORITHM.

AIM:

To implement the 2D Line using a liang-barsky algorithm.

ALGORITHM:

1. Start
2. Accept endpoints and clipping window.
3. Convert line to parametric form.
4. Calculate p and q values for edges.
5. Determine entry and exit points using t values.
6. If valid t values, clip and draw the line.
7. Stop

PROGRAM:

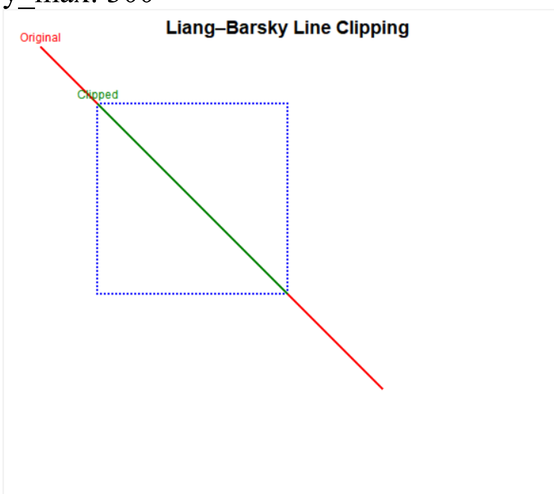
```
import tkinter as tk

def liang_barsky(x1, y1, x2, y2, x_min, y_min, x_max, y_max):
    dx = x2 - x1
    dy = y2 - y1
    p = [-dx, dx, -dy, dy]
    q = [x1 - x_min, x_max - x1, y1 - y_min, y_max - y1]

    u1 = 0.0
    u2 = 1.0

    for i in range(4):
        if p[i] == 0:
            if q[i] < 0:
                return False, x1, y1, x2, y2
        else:
            u = q[i] / p[i]
            if p[i] < 0:
                if u > u1:
                    u1 = u
            else:
                if u < u2:
                    u2 = u

    if u1 > u2:
        return False, x1, y1, x2, y2

    x1_clip = x1 + u1 * dx
```

```
    y1_clip = y1 + u1 * dy
    x2_clip = x1 + u2 * dx
    y2_clip = y1 + u2 * dy

    return True, x1_clip, y1_clip, x2_clip, y2_clip

def draw_result(x1, y1, x2, y2, clipped, x_min, y_min, x_max, y_max):
    root = tk.Tk()
    root.title("Liang–Barsky Line Clipping Algorithm")
    canvas = tk.Canvas(root, width=600, height=600, bg="white")
    canvas.pack()

    canvas.create_rectangle(x_min, y_min, x_max, y_max, outline="blue", dash=(5, 5), width=2)
    canvas.create_text(300, 20, text="Liang–Barsky Line Clipping", fill="black", font=("Arial", 14,
"bold"))

    canvas.create_line(orig_x1, orig_y1, orig_x2, orig_y2, fill="red", width=2)
    canvas.create_text(orig_x1, orig_y1 - 10, text="Original", fill="red", font=("Arial", 9))

    if clipped:
        # Draw clipped line in green
        canvas.create_line(x1, y1, x2, y2, fill="green", width=2)
        canvas.create_text(x1, y1 - 10, text="Clipped", fill="green", font=("Arial", 9))
    else:
        canvas.create_text(300, 560, text="Line Rejected (Outside)", fill="red", font=("Arial", 12, "bold"))

    root.mainloop()

print("Enter Line Coordinates:")
orig_x1 = int(input("x1: "))
orig_y1 = int(input("y1: "))
orig_x2 = int(input("x2: "))
orig_y2 = int(input("y2: "))

print("\nEnter Clipping Window:")
x_min = int(input("x_min: "))
y_min = int(input("y_min: "))
x_max = int(input("x_max: "))
y_max = int(input("y_max: "))

accept, cx1, cy1, cx2, cy2 = liang_barsky(orig_x1, orig_y1, orig_x2, orig_y2, x_min, y_min, x_max,
y_max)


draw_result(cx1, cy1, cx2, cy2, accept, x_min, y_min, x_max, y_max)
```

OUTPUT:

Enter Line Coordinates:
x1: 40
y1: 40
x2: 400
y2: 400

Enter Clipping Window:
x_min: 100
y_min: 100
x_max: 300
y_max: 300



RESULT:

Thus the python program to implement the 2D Line using a liang-barsky algorithm was successfully executed.

EX.NO.6	POLYGON CLIPPING USING SUTHERLAND-HODGEMAN ALGORITHM

AIM:

To implement Sutherland-Hodgeman polygon clipping algorithm using Python program.

ALGORITHM:

1. Start
2. Input polygon vertices and clipping window.
3. Clip the polygon edge-by-edge for all 4 window edges.
4. Check each edge: inside/outside combinations.
5. Add resulting points to output list.
6. After final edge, draw the clipped polygon.
7. Stop

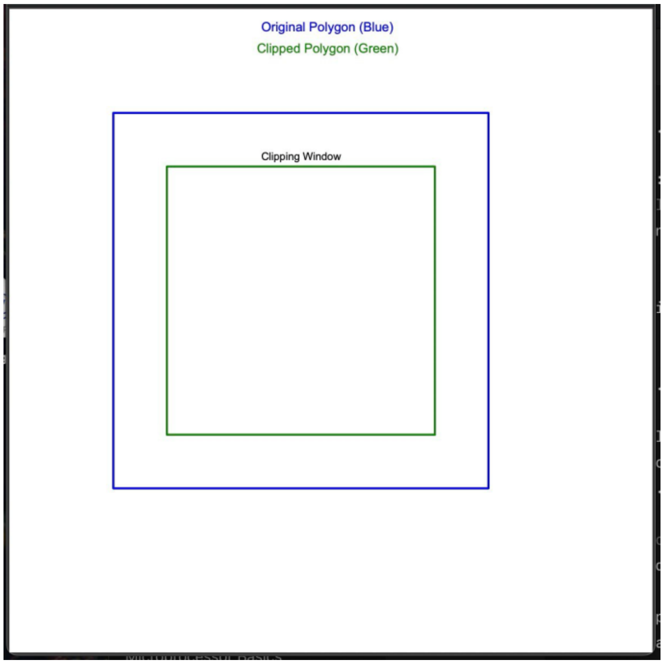
PROGRAM:

```
import tkinter as tk

clip_window = {'xmin': 150, 'ymin': 150, 'xmax': 400, 'ymax': 400}

def inside(p, edge):
    x, y = p
    if edge == 'LEFT':
        return x >= clip_window['xmin']
    elif edge == 'RIGHT':
        return x <= clip_window['xmax']
    elif edge == 'BOTTOM':
        return y >= clip_window['ymin']
    elif edge == 'TOP':
        return y <= clip_window['ymax']

def intersect(p1, p2, edge):
    x1, y1 = p1
    x2, y2 = p2

    if edge == 'LEFT':
        x = clip_window['xmin']
        y = y1 + (y2 - y1) * (x - x1) / (x2 - x1)
    elif edge == 'RIGHT':
        x = clip_window['xmax']
        y = y1 + (y2 - y1) * (x - x1) / (x2 - x1)
    elif edge == 'BOTTOM':
        y = clip_window['ymin']
```

```
        x = x1 + (x2 - x1) * (y - y1) / (y2 - y1)
    elif edge == 'TOP':
        y = clip_window['ymax']
        x = x1 + (x2 - x1) * (y - y1) / (y2 - y1)
    return (x, y)

def suth_hodg_clip(polygon, edges=['LEFT', 'RIGHT', 'BOTTOM', 'TOP']):
    output_list = polygon
    for edge in edges:
        input_list = output_list
        output_list = []
        if not input_list:
            break
        s = input_list[-1]

        for e in input_list:
            if inside(e, edge):
                if inside(s, edge):
                    output_list.append(e)
                else:
                    output_list.append(intersect(s, e, edge))
                    output_list.append(e)
            elif inside(s, edge):
                output_list.append(intersect(s, e, edge))
            s = e
    return output_list

def draw_polygon(canvas, points, outline_color, fill_color="", width=2):
    if len(points) >= 2:
        canvas.create_polygon(points, outline=outline_color, fill=fill_color, width=width)

def draw_clipping_window(canvas):
    canvas.create_rectangle(
        clip_window['xmin'], clip_window['ymin'],
        clip_window['xmax'], clip_window['ymax'],
        outline='black', dash=(4, 2)
    )
    canvas.create_text(275, 140, text="Clipping Window", fill="black", font=('Arial', 10))

def main():

    polygon_points = [(100, 100), (450, 100), (450, 450), (100, 450)]

    root = tk.Tk()
    root.title("Sutherland-Hodgman Polygon Clipping")

    canvas = tk.Canvas(root, width=600, height=600, bg='white')
    canvas.pack()
```

```
    draw_clipping_window(canvas)
    draw_polygon(canvas, polygon_points, outline_color="blue", fill_color="", width=2)
    canvas.create_text(300, 20, text="Original Polygon (Blue)", fill="blue", font=('Arial', 12))

    clipped_polygon = suth_hodg_clip(polygon_points)

    if clipped_polygon:
        draw_polygon(canvas, clipped_polygon, outline_color="green", fill_color="", width=2)
        canvas.create_text(300, 40, text="Clipped Polygon (Green)", fill="green", font=('Arial', 12))

    root.mainloop()

main()
```

OUTPUT:



RESULT:

Thus Polygon Clipping using Sutherland Hodgeman algorithm is implemented and output is verified.

EX.NO.7	THREE DIMENSIONAL TRANSFORMATIONS – TRANSLATION, ROTATION AND SCALING

AIM:

To implement 3D transformations using Python program.

ALGORITHM:

1. Start
2. Accept coordinates of the 3D object.
3. Apply translation by adding Tx, Ty, Tz.
4. Apply scaling using Sx, Sy, Sz.
5. Rotate around X, Y, or Z using rotation matrices.
6. Display transformed object using 3D plotting.
7. Stop

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

cube = np.array([
    [0, 0, 0],
    [0, 0, 1],
    [0, 1, 0],
    [0, 1, 1],
    [1, 0, 0],
    [1, 0, 1],
    [1, 1, 0],
    [1, 1, 1]
])

edges = [
    (0,1), (0,2), (0,4), (1,3), (1,5),
    (2,3), (2,6), (3,7), (4,5), (4,6),
    (5,7), (6,7)
]

def plot_3d(original, transformed, title):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.set_title(title)
```

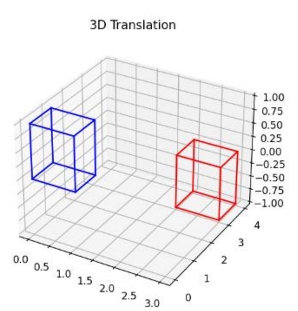
```
    for edge in edges:
        p1 = original[edge[0]]
        p2 = original[edge[1]]
        ax.plot([p1[0], p2[0]], [p1[1], p2[1]], [p1[2], p2[2]], color='blue')

    for edge in edges:
        p1 = transformed[edge[0]]
        p2 = transformed[edge[1]]
        ax.plot([p1[0], p2[0]], [p1[1], p2[1]], [p1[2], p2[2]], color='red')

    plt.show()

def translate(cube, tx, ty, tz):
    matrix = np.array([
        [1, 0, 0, tx],
        [0, 1, 0, ty],
        [0, 0, 1, tz],
        [0, 0, 0, 1]
    ])
    points = np.hstack((cube, np.ones((cube.shape[0], 1))))
    return (matrix @ points.T).T[:, :3]

def scale(cube, sx, sy, sz):
    matrix = np.array([
        [sx, 0,  0,  0],
        [0,  sy, 0,  0],
        [0,  0,  sz, 0],
        [0,  0,  0,  1]
    ])
    points = np.hstack((cube, np.ones((cube.shape[0], 1))))
    return (matrix @ points.T).T[:, :3]

def rotate_x(cube, angle_deg):
    angle = np.radians(angle_deg)
    matrix = np.array([
        [1, 0,             0,              0],
        [0, np.cos(angle), -np.sin(angle), 0],
        [0, np.sin(angle),  np.cos(angle), 0],
        [0, 0,             0,              1]
    ])
    points = np.hstack((cube, np.ones((cube.shape[0], 1))))
    return (matrix @ points.T).T[:, :3]

def rotate_y(cube, angle_deg):
    angle = np.radians(angle_deg)
    matrix = np.array([
        [ np.cos(angle), 0, np.sin(angle), 0],
        [ 0,             1, 0,             0],
```

```
        [-np.sin(angle), 0, np.cos(angle), 0],
        [ 0,             0, 0,             1]
    ])
    points = np.hstack((cube, np.ones((cube.shape[0], 1))))
    return (matrix @ points.T).T[:, :3]

def rotate_z(cube, angle_deg):
    angle = np.radians(angle_deg)
    matrix = np.array([
        [np.cos(angle), -np.sin(angle), 0, 0],
        [np.sin(angle),  np.cos(angle), 0, 0],
        [0,             0,             1, 0],
        [0,             0,             0, 1]
    ])
    points = np.hstack((cube, np.ones((cube.shape[0], 1))))
    return (matrix @ points.T).T[:, :3]

while True:
    print("\n3D Transformation Menu:")
    print("1. Translation")
    print("2. Scaling")
    print("3. Rotation about X-axis")
    print("4. Rotation about Y-axis")
    print("5. Rotation about Z-axis")
    print("6. Exit")

choice = int(input("Enter your choice: "))
    if choice == 1:
        tx = float(input("Translate X by: "))
        ty = float(input("Translate Y by: "))
        tz = float(input("Translate Z by: "))
        new_cube = translate(cube, tx, ty, tz)
        plot_3d(cube, new_cube, "3D Translation")
    elif choice == 2:
        sx = float(input("Scale X by: "))
        sy = float(input("Scale Y by: "))
        sz = float(input("Scale Z by: "))
        new_cube = scale(cube, sx, sy, sz)
        plot_3d(cube, new_cube, "3D Scaling")
    elif choice == 3:
        angle = float(input("Enter rotation angle (X-axis): "))
        new_cube = rotate_x(cube, angle)
        plot_3d(cube, new_cube, "Rotation about X-axis")
    elif choice == 4:
        angle = float(input("Enter rotation angle (Y-axis): "))
        new_cube = rotate_y(cube, angle)
        plot_3d(cube, new_cube, "Rotation about Y-axis")
    elif choice == 5:
```

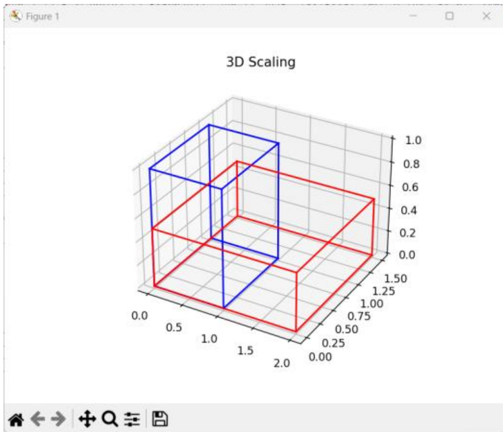
```
        angle = float(input("Enter rotation angle (Z-axis): "))
        new_cube = rotate_z(cube, angle)
        plot_3d(cube, new_cube, "Rotation about Z-axis")
    elif choice == 6:
        break
    else:
        print("Invalid choice. Try again.")
```

OUTPUT:

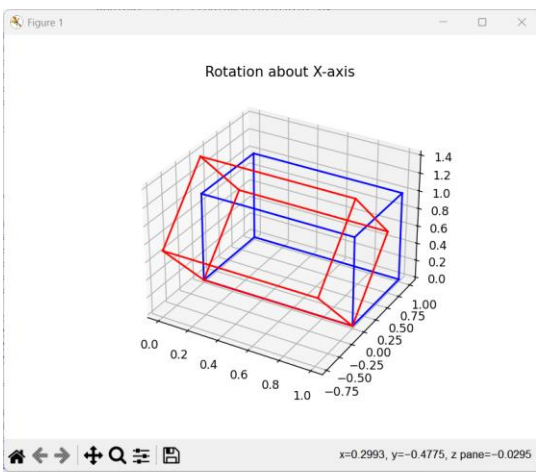
```
--- 3D Transformation Menu ---
1. Translation
2. Scaling
3. Rotation about X-axis
4. Rotation about Y-axis
5. Rotation about Z-axis
6. Exit
Enter your choice: 1
Translate X by: 2
Translate Y by: 3
Translate Z by: -1
```



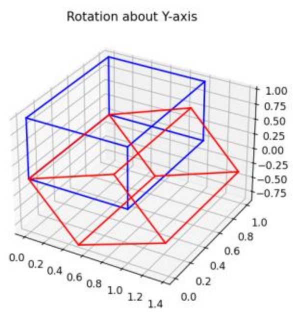

```
--- 3D Transformation Menu ---
1. Translation
2. Scaling
3. Rotation about X-axis
4. Rotation about Y-axis
5. Rotation about Z-axis
6. Exit
Enter your choice: 2
Scale X by: 2
Scale Y by: 1.5
Scale Z by: 0.5
```



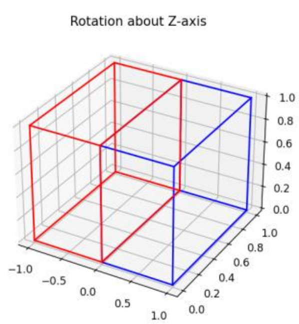
--- 3D Transformation Menu ---
1. Translation
2. Scaling
3. Rotation about X-axis
4. Rotation about Y-axis
5. Rotation about Z-axis
6. Exit
Enter your choice: 3
Enter rotation angle (X-axis): 45



--- 3D Transformation Menu ---
1. Translation
2. Scaling
3. Rotation about X-axis
4. Rotation about Y-axis
5. Rotation about Z-axis
6. Exit
Enter your choice: 4
Enter rotation angle (Y-axis): 60



--- 3D Transformation Menu ---
1. Translation
2. Scaling
3. Rotation about X-axis
4. Rotation about Y-axis
5. Rotation about Z-axis
6. Exit
Enter your choice: 5
Enter rotation angle (Z-axis): 90



RESULT:

Thus, the 3D Transformations on a cube are implemented and the output is verified.

EX. NO. 8A IMPLEMENTATION OF 2D ANIMATION (USING TIMER, LOOP, SIMPLE ANIMATION) – BOUNCING BALL)

AIM:

To implement the 2D Animation (using timer, Loop, Simple animation) Bouncing Ball

ALGORITHM:

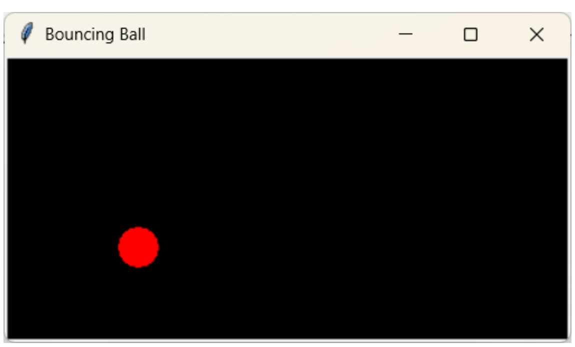
1. Start
2. Draw the ball at initial position.
3. Use loop or timer to move it.
4. Detect boundaries and reverse direction if needed.
5. Redraw ball in new position continuously.
6. Stop

PROGRAM:

```
import tkinter as tk

width = 400

height = 200

ball_radius = 15

x = 50

y = 50

dx = 3

dy = 2

root = tk.Tk()

root.title("Bouncing Ball")


canvas = tk.Canvas(root, width=width, height=height, bg="black")

canvas.pack()


ball = canvas.create_oval(x - ball_radius, y - ball_radius,

                          x + ball_radius, y + ball_radius,
```

```
                    fill="red")


def move_ball():
    global x, y, dx, dy
    x += dx
    y += dy
    if x - ball_radius <= 0 or x + ball_radius >= width:
        dx = -dx
    if y - ball_radius <= 0 or y + ball_radius >= height:
        dy = -dy
    canvas.coords(ball, x - ball_radius, y - ball_radius, x + ball_radius, y + ball_radius)
    root.after(20, move_ball)


move_ball()
root.mainloop()
```

OUTPUT:



RESULT:

Thus the python program for 2d animation (using timer, loop, simple animation)-Bouncing ball has successfully created.

EX. NO. 8B IMPLEMENTATION OF 2D ANIMATION (USING TIMER, LOOP, SIMPLE ANIMATION)- CAR MOVEMENT

AIM:

To implement the 2D Animation (using timer, Loop, Simple animation) Car Movement

ALGORITHM:

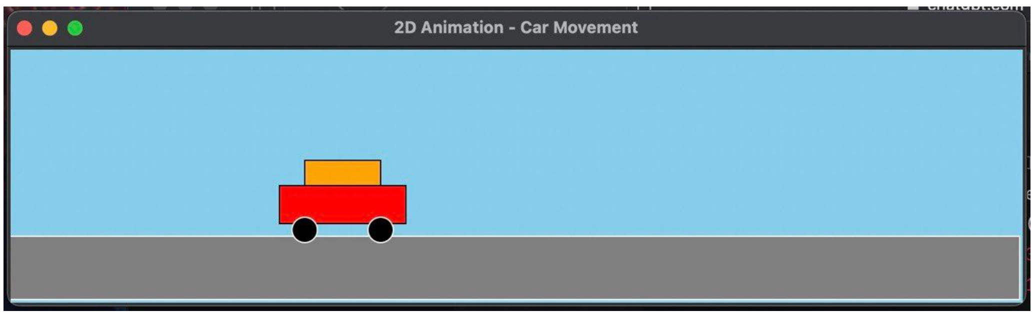
1. Start
2. Draw the car at starting position.
3. Use loop or timer to update its position forward.
4. Redraw at each step to simulate motion.
5. Repeat or stop on key event.
6. Stop

PROGRAM:

```
import tkinter as tk

root = tk.Tk()

root.title("2D Animation - Car Movement")

width = 800

height = 200

canvas = tk.Canvas(root, width=width, height=height, bg="skyblue")

canvas.pack()

canvas.create_rectangle(0, 150, width, height, fill="gray")

car_body = canvas.create_rectangle(50, 110, 150, 140, fill="red", outline="black")

car_top = canvas.create_rectangle(70, 90, 130, 110, fill="orange", outline="black")

wheel1 = canvas.create_oval(60, 135, 80, 155, fill="black")

wheel2 = canvas.create_oval(120, 135, 140, 155, fill="black")

car_parts = [car_body, car_top, wheel1, wheel2]

def move_car():

    for part in car_parts:

        canvas.move(part, 5, 0)  # Move right by 5 pixels
```

```
    if canvas.coords(car_body)[2] < width:
        root.after(50, move_car)


move_car()
root.mainloop()
```

OUTPUT:



RESULT:

Thus the python program for 2d animation (using timer, loop, simple animation)-Bouncing ball has successfully created.

EX. NO. 9	IMPLEMENTATION OF 3D ANIMATION - HUMAN FACIAL EXPRESSIONS: SMILE, SAD, SURPRISE.

AIM:

To generate the 3D Animation Human Facial Expresions like Smile, Sad, Surprise

ALGORITHM:

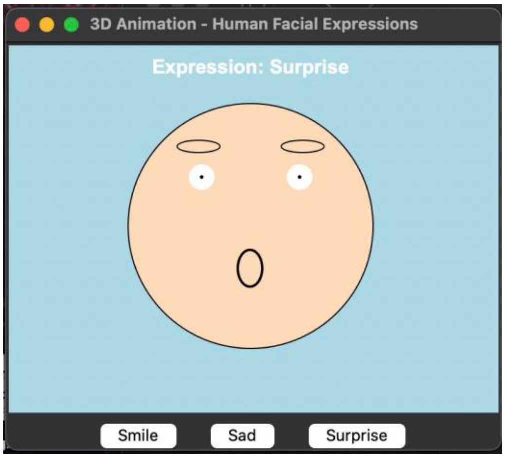
1. Start
2. Draw basic 3D face model.
3. Use different mouth, eye, and eyebrow positions for each emotion.
4. Change facial parts using transformations or predefined shapes.
5. Update frames to animate transitions.
6. Stop

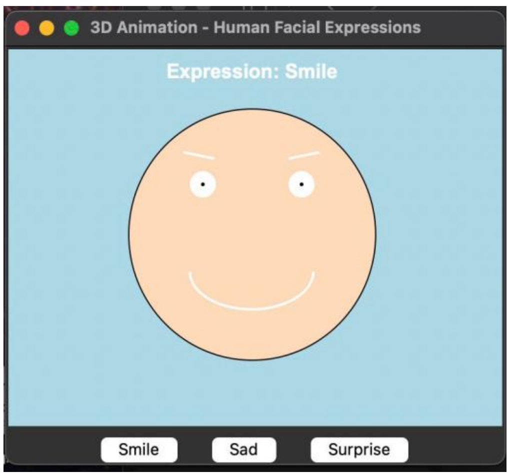
PROCEDURE:

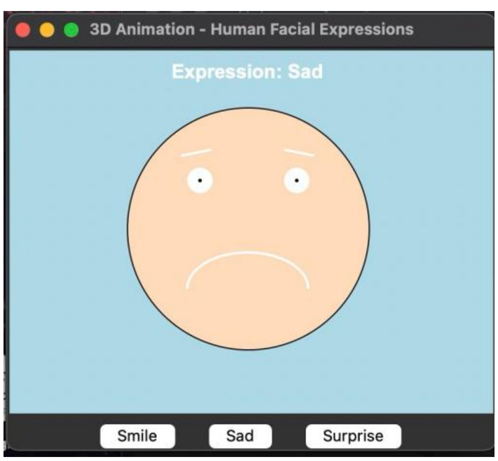
```
import tkinter as tk

def draw_face(expression):
    canvas.delete("all")

    canvas.create_oval(100, 50, 300, 250, fill="peachpuff", outline="black")

    canvas.create_oval(150, 100, 170, 120, fill="white")
    canvas.create_oval(230, 100, 250, 120, fill="white")
    canvas.create_oval(158, 108, 162, 112, fill="black")  # Pupils
    canvas.create_oval(238, 108, 242, 112, fill="black")

    if expression == "sad":
        canvas.create_line(145, 90, 170, 85, width=2)
        canvas.create_line(230, 85, 255, 90, width=2)
    elif expression == "surprise":
        canvas.create_oval(140, 80, 175, 90, outline="black")
        canvas.create_oval(225, 80, 260, 90, outline="black")
    else:
        canvas.create_line(145, 85, 170, 90, width=2)
        canvas.create_line(230, 90, 255, 85, width=2)

    if expression == "smile":
        canvas.create_arc(150, 150, 250, 210, start=0, extent=-180, style="arc", width=2)
    elif expression == "sad":
        canvas.create_arc(150, 170, 250, 230, start=0, extent=180, style="arc", width=2)
    elif expression == "surprise":
        canvas.create_oval(190, 170, 210, 200, outline="black", width=2)
```

```
    canvas.create_text(200, 20, text=f"Expression: {expression.capitalize()}", font=("Arial", 16, "bold"))

root = tk.Tk()
root.title("3D Animation - Human Facial Expressions")
canvas = tk.Canvas(root, width=400, height=300, bg="lightblue")
canvas.pack()

frame = tk.Frame(root)
frame.pack()

tk.Button(frame, text="Smile", command=lambda: draw_face("smile")).pack(side=tk.LEFT, padx=10)
tk.Button(frame, text="Sad", command=lambda: draw_face("sad")).pack(side=tk.LEFT, padx=10)
tk.Button(frame, text="Surprise", command=lambda: draw_face("surprise")).pack(side=tk.LEFT,
padx=10)

draw_face("smile")

root.mainloop()
```

OUTPUT:







RESULT:

Thus the 3D Animation – Human Facial Expressions like Smile, Sad, Surprise are successfully generated.

Ex. No. 10	**Drawing 3D Objects and Scenes using OpenGL**

AIM:

To draw a 3d objects and scenes using opengl

ALGORITHM:

1. Start
2. Set up OpenGL environment and viewport.
3. Define camera position and 3D perspective.
4. Use OpenGL functions to create 3D shapes (cube, cone, etc.).
5. Add colors, lights, and textures to the scene.
6. Render the full 3D scene.
7. Stop

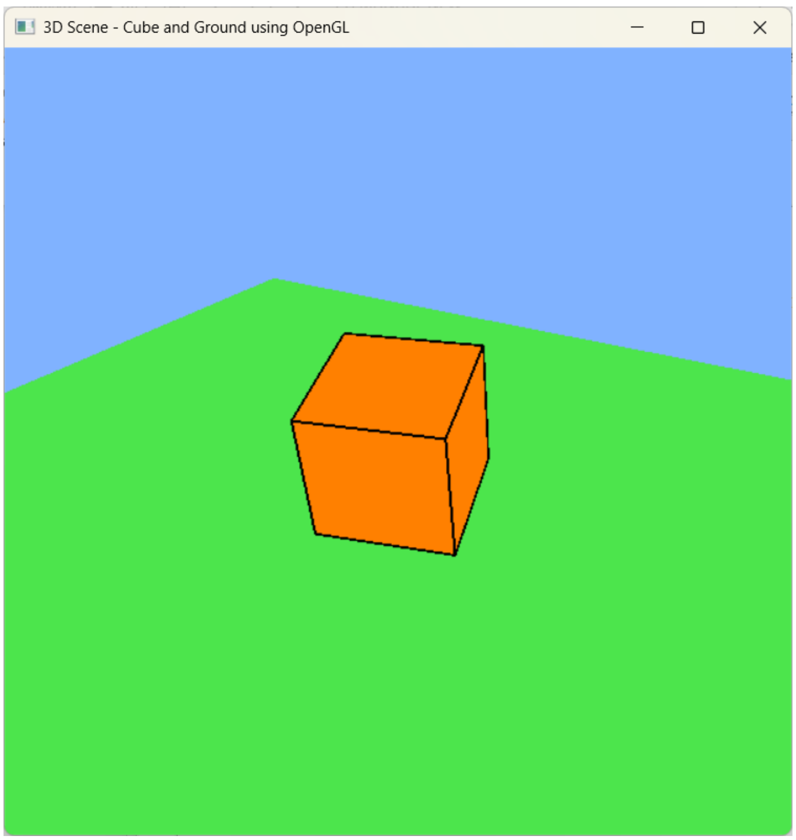
PROGRAM:

```
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *

angle = 0

def init():
   glClearColor(0.5, 0.7, 1.0, 1.0)
   glEnable(GL_DEPTH_TEST)

def draw_cube_with_outline():

   glColor3f(1.0, 0.5, 0.0)
   glutSolidCube(1.0)

   glColor3f(0.0, 0.0, 0.0)
   glLineWidth(2)
   glutWireCube(1.01)

def draw_ground():
   glColor3f(0.3, 0.9, 0.3)
   glBegin(GL_QUADS)
   glVertex3f(-5, -1, -5)
   glVertex3f(-5, -1, 5)
```

```
    glVertex3f(5, -1, 5)
    glVertex3f(5, -1, -5)
    glEnd()

def display():
    global angle
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()

    gluLookAt(3, 3, 5, 0, 0, 0, 0, 1, 0)

    draw_ground()

    glPushMatrix()
    glRotatef(angle, 1, 1, 0)
    draw_cube_with_outline()
    glPopMatrix()

    glutSwapBuffers()

def update(value):
    global angle
    angle += 1
    if angle > 360:
        angle -= 360
    glutPostRedisplay()
    glutTimerFunc(16, update, 0)

def reshape(w, h):
    glViewport(0, 0, w, h)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(45, w / h, 1, 50)
    glMatrixMode(GL_MODELVIEW)

def main():
    glutInit()
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH)
    glutInitWindowSize(600, 600)
    glutCreateWindow(b"3D Scene - Cube and Ground using OpenGL")
```

```
    init()
    glutDisplayFunc(display)
    glutReshapeFunc(reshape)
    glutTimerFunc(0, update, 0)
    glutMainLoop()

main()
```

OUTPUT:



RESULT:

Thus the program to create and draw a 3d objects and scenes using opengl has successfully created.