West Bengal State University

Department of Computer Science

Assignment

Subject: Basics of Gaming

Course Code: CMSPSEC01M

Name: Sayan Dutta

Registration No.: A01-1112-117-006-2021

Registration Session: 2021

Semester: II

Academic Session: 2024–2026

Submitted to:

Dr. Rajat Pandit

Department of Computer Science

West Bengal State University

Contents

1 Draw Shapes

Draw the following geometric shapes using Python:

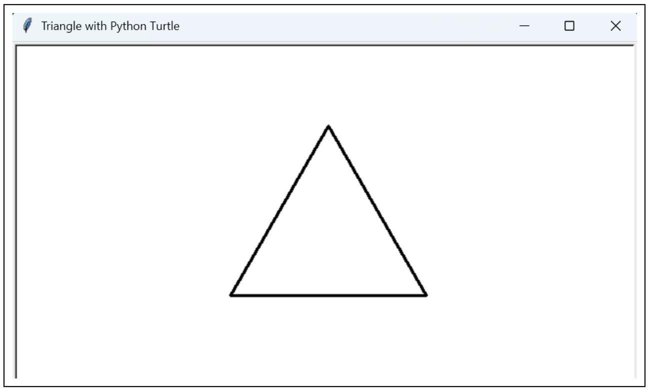
- Triangle
- Circle
- Square
- Rectangle
- Rhombus
- Trapezium
- Oval
- Line

1.1 Triangle

1.1.1 Python Implementation

```python
import turtle

screen = turtle.Screen()
screen.title("Triangle with Python Turtle")

t = turtle.Turtle()
t.pensize(3)
t.pencolor("black")
t.speed(’normal’)

t.penup()
t.left(180)
t.forward(100)
t.left(180)
t.pendown()

side_length = 200
for _ in range(3):
   t.forward(side_length)
   t.left(120)

t.hideturtle()
screen.mainloop()
```

1.1.2 Program output

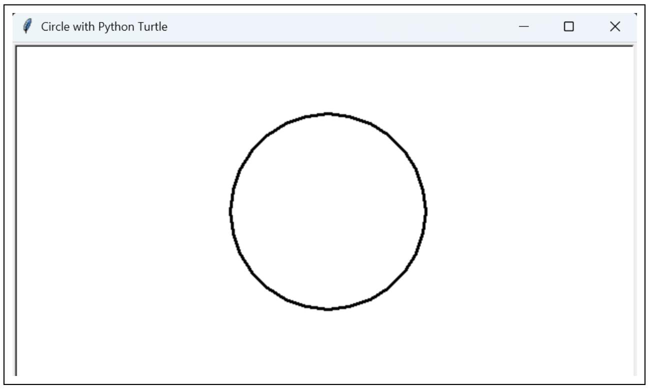


1.2 Circle

1.2.1 Python Implementation

```
import turtle

screen = turtle.Screen()
screen.title("Circle with Python Turtle")

t = turtle.Turtle()
t.pensize(3)
t.pencolor("black")
t.speed(’normal’)

radius = 100
t.circle(radius)

t.hideturtle()
screen.mainloop()
```

1.2.2 Program output

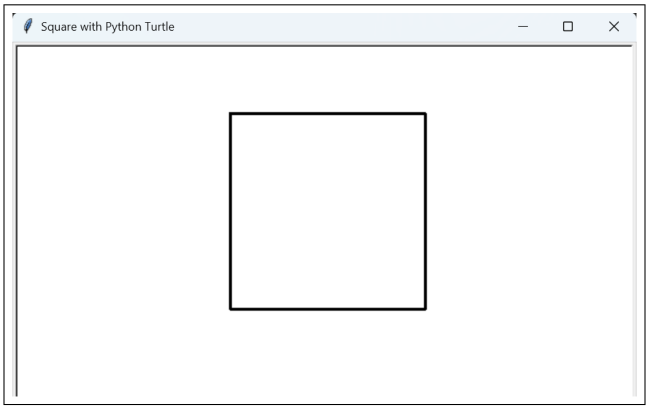


1.3 Square

1.3.1 Python Implementation

```
import turtle

screen = turtle.Screen()
screen.title("Square with Python Turtle")

t = turtle.Turtle()
t.pensize(3)
t.pencolor("black")
t.speed('normal')

t.penup()
t.left(180)
t.forward(100)
t.left(180)
t.pendown()

side_length = 200
for _ in range(4):
   t.forward(side_length)
   t.left(90)

t.hideturtle()
screen.mainloop()
```

1.3.2 Program output

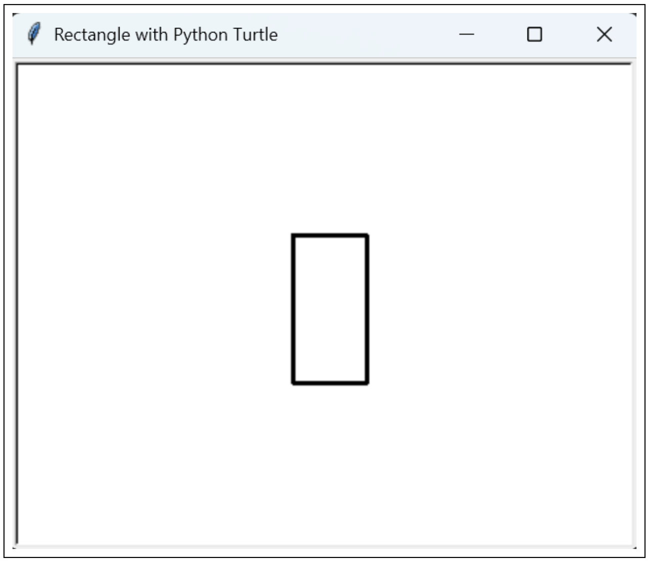


1.4 Rectangle

1.4.1 Python Implementation

```
import turtle

screen = turtle.Screen()
screen.title("Rectangle with Python Turtle")

t = turtle.Turtle()
t.pensize(3)
t.pencolor("black")
t.speed(’normal’)

width = 50
height = 100

t.penup()
t.goto(-width/2, -height/2) # center the rectangle
t.pendown()

for _ in range(2):
   t.forward(width)
   t.left(90)
   t.forward(height)
   t.left(90)

t.hideturtle()
screen.mainloop()
```

1.4.2 Program output

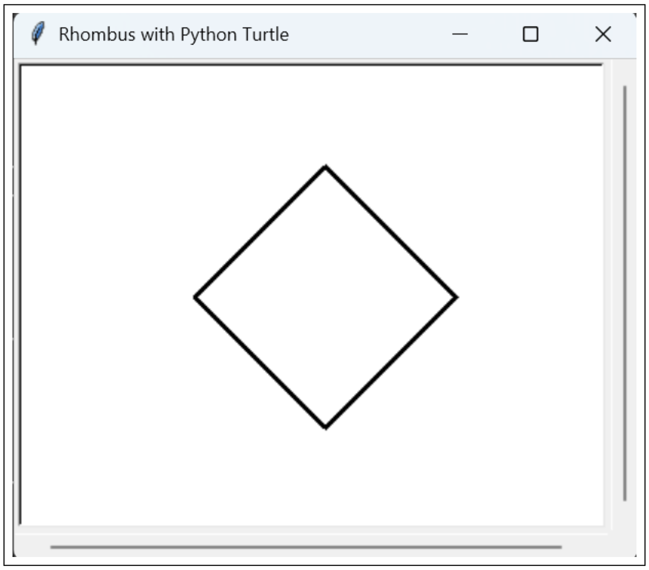


1.5 Rhombus

1.5.1 Python Implementation

```
import turtle

screen = turtle.Screen()
screen.title("Rhombus with Python Turtle")

t = turtle.Turtle()

t.penup()
t.left(180)
t.forward(80)
t.left(180)
t.pendown()

t.pensize(3)
t.pencolor("black")
t.speed('normal')

t.right(90)
t.circle(85, steps=4)

t.hideturtle()
screen.mainloop()
```

1.5.2 Program output

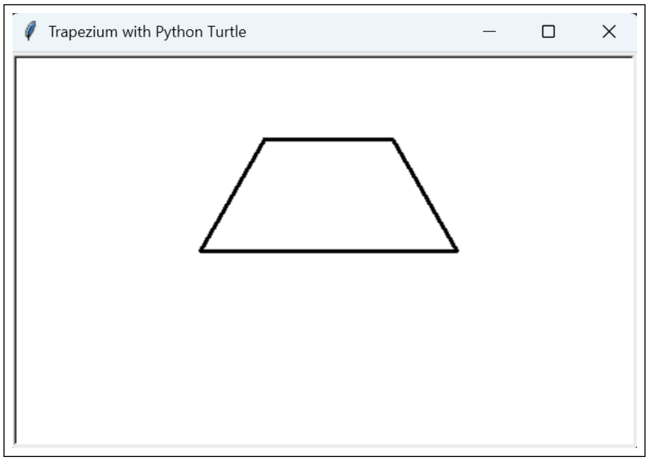


1.6 Trapezium

1.6.1 Python Implementation

```python
import turtle

screen = turtle.Screen()
screen.title("Trapezium with Python Turtle")

t = turtle.Turtle()
t.pensize(3)
t.pencolor("black")
t.speed('normal')

t.penup()
t.left(180)
t.forward(100)
t.right(180)
t.pendown()

t.forward(200)
t.left(120)
t.forward(100)
t.left(60)
t.forward(100)
t.left(60)
t.forward(100)

t.hideturtle()
screen.mainloop()
```

1.6.2 Program output

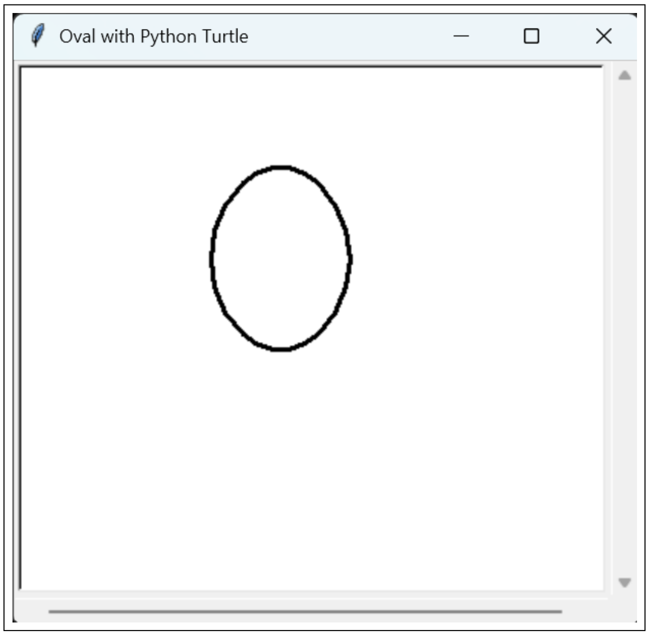


1.7 Oval

1.7.1 Python Implementation

```
import turtle

screen = turtle.Screen()
screen.title("Oval with Python Turtle")

t = turtle.Turtle()
t.pensize(3)
t.pencolor("black")
t.speed('normal')

radius = 70

t.left(45)
for loop in range(2):
   t.circle(radius,90)
   t.circle(radius/2,90)

t.hideturtle()
screen.mainloop()
```

1.7.2 Program output



1.8 Line

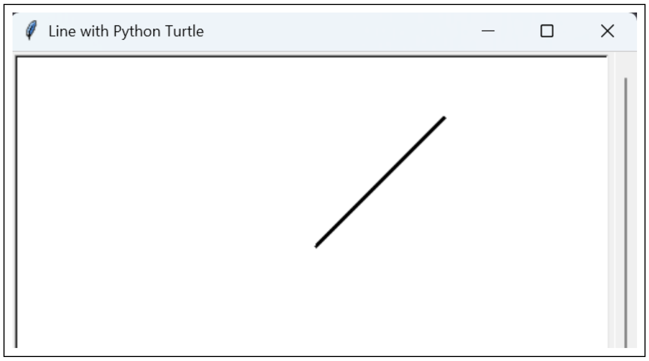
1.8.1 Python Implementation

```
import turtle

screen = turtle.Screen()
screen.title("Line with Python Turtle")

t = turtle.Turtle()
t.pensize(3)
t.pencolor("black")
t.speed('normal')

x1 = float(input("Enter starting X coordinate: "))
y1 = float(input("Enter starting Y coordinate: "))
x2 = float(input("Enter ending X coordinate: "))
y2 = float(input("Enter ending Y coordinate: "))

t.penup()
t.goto(x1, y1)
t.pendown()

t.goto(x2, y2)

t.hideturtle()
screen.mainloop()
```

1.8.2 Program intput

```
Enter starting X coordinate: 0
Enter starting Y coordinate: 0
Enter ending X coordinate: 100
Enter ending Y coordinate: 100
```

1.8.3 Program Output



2 Small Alphabets

Write a Python program to draw the small alphabets from a to z.

2.1 Python Implementation

```
import turtle as t
t.pensize(5)
t.pencolor("black")
t.speed(20)

# Move cursor
t.penup()
t.left(180)
t.fd(330)
t.right(90)
t.fd(250)
t.right(90)
t.pendown()

# a
t.left(90)
t.circle(-15,180)
t.fd(40)
t.circle(10,100)
t.penup()
t.bk(10)
t.left(85)
t.fd(35)
t.left(45)
t.pendown()
t.circle(18,255)

#space
t.penup()
t.fd(40)
t.left(55)
t.fd(60)
t.pendown()

# b
t.bk(80)
t.fd(20)
```

```
#m
t.fd(30)
t.bk(18)
t.left(180)
t.circle(-10,180)
t.fd(18)
t.bk(18)
t.left(180)
t.circle(-10,180)
t.fd(18)

#space
t.penup()
t.left(90)
t.fd(30)
t.left(90)
t.fd(30)
t.right(180)
t.pendown()

#space
t.penup()
t.left(90)
t.fd(30)
t.left(90)
t.fd(30)
t.pendown()

#o
t.left(90)
t.circle(15)

#space
t.penup()
t.right(180)
t.fd(30)
t.pendown()
```

```
t.circle(-17,325)

#space
t.penup()
t.right(35)
t.fd(30)
t.right(90)
t.fd(80)
t.pendown()

# c
t.penup()
t.left(-45)
t.fd(1)
t.pendown()
t.right(180)
t.circle(20,270)

#space
t.penup()
t.right(45)
t.fd(60)
t.left(90)
t.fd(60)
t.right(180)
t.pendown()

#d
t.fd(60)
t.bk(15)
t.circle(-15,-325)

#space
t.penup()
t.left(125)
t.fd(30)
t.left(90)
t.fd(15)
t.right(90)
t.pendown()

#e
t.fd(28)
t.left(90)
t.circle(16,310)
```

```

#p
t.right(90)
t.fd(50)
t.bk(35)
t.left(180)
t.circle(-13,330)

#space
t.penup()
t.right(120)
t.fd(70)
t.left(90)
t.fd(20)
t.right(90)
t.pendown()

#q
t.penup()
t.left(-45)
t.fd(5)
t.pendown()
t.right(180)
t.circle(13,315)
t.fd(15)
t.bk(50)
t.right(45)
t.fd(15)

#space
t.penup()
t.fd(20)
t.left(45)
t.fd(25)
t.right(90)
t.pendown()

#r
t.fd(10)
t.right(90)
t.fd(30)
t.bk(15)
t.left(140)
t.fd(15)
t.circle(-10,130)
```

```

#space
t.penup()
t.right(40)
t.fd(60)
t.left(90)
t.fd(50)
t.pendown()

#f
t.circle(15,180)
t.fd(60)
t.penup()
t.bk(40)
t.right(90)
t.fd(15)
t.left(180)
t.pendown()
t.fd(30)

#space
t.penup()
t.fd(50)
t.right(90)
t.fd(15)
t.left(90)
t.pendown()

# g
t.penup()
t.left(-45)
t.fd(5)
t.pendown()
t.right(180)
t.circle(15,315)
t.fd(15)
t.bk(50)
t.right(180)
t.circle(-15,180)

#space
t.penup()
t.fd(80)
t.right(90)
t.fd(50)
```

```

#space
t.penup()
t.left(80)
t.fd(35)
t.left(90)
t.fd(10)
t.right(90)
t.pendown()

#s
t.left(180)
t.fd(7)
t.circle(10,180)
t.fd(00)
t.circle(-10,180)
t.fd(7)

#space
t.penup()
t.right(180)
t.fd(45)
t.left(90)
t.fd(55)
t.right(90)
t.pendown()

#t
t.right(90)
t.fd(50)
t.circle(10,180)
t.penup()
t.fd(40)
t.left(90)
t.fd(30)
t.right(180)
t.pendown()
t.fd(25)

#space
t.penup()
t.fd(30)
t.right(90)
t.fd(20)
t.left(90)
```

```
t.pendown()

#h
t.right(90)
t.fd(60)
t.bk(20)
t.left(180)
t.circle(-10,180)
t.fd(20)
#space
t.penup()
t.left(90)
t.fd(30)
t.left(90)
t.fd(30)
t.right(180)
t.pendown()

#i
t.fd(30)
t.penup()
t.bk(40)
t.pendown()
t.circle(1)

#space
t.penup()
t.left(90)
t.fd(40)
t.right(90)
t.fd(20)
t.left(90)
t.pendown()

#j
t.fd(20)
t.right(90)
t.fd(40)
t.circle(-10,150)
t.penup()
t.right(30)
t.fd(55)
t.right(90)
t.fd(18)
t.pendown()
```

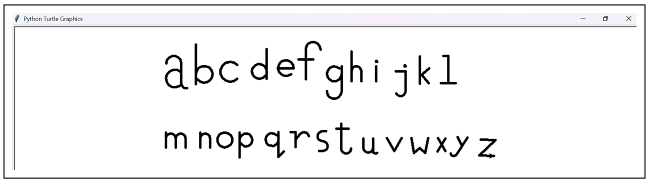
```
t.pendown()

#u
t.right(90)
t.fd(18)
t.circle(10,180)
t.fd(18)
t.bk(25)
t.right(180)
t.circle(5,130)

#space
t.penup()
t.left(50)
t.fd(30)
t.right(90)
t.fd(20)
t.pendown()

#v
t.right(60)
t.fd(30)
t.left(120)
t.fd(30)

#space
t.penup()
t.right(60)
t.fd(20)
t.pendown()

#w
t.right(80)
t.fd(30)
t.left(120)
t.fd(15)
t.right(80)
t.fd(15)
t.left(120)
t.fd(30)

#space
t.penup()
t.right(80)
t.fd(20)
```

```
t.circle(1)

#space
t.penup()
t.fd(30)
t.left(90)
t.fd(20)
t.right(180)
t.pendown()

#k
t.fd(55)
t.bk(15)
t.left(130)
t.fd(25)
t.bk(20)
t.right(90)
t.fd(22)

#space
t.penup()
t.left(50)
t.fd(30)
t.left(90)
t.fd(55)
t.pendown()

#l
t.right(90)
t.fd(10)
t.right(90)
t.fd(55)
t.right(90)
t.fd(15)
t.bk(30)

#next line
t.penup()
t.left(90)
t.fd(100)
t.right(90)
t.fd(590)
t.right(-90)
t.pendown()
```

```
t.pendown()

#x
t.right(60)
t.fd(30)
t.left(150)
t.penup()
t.fd(25)
t.pendown()
t.left(148)
t.fd(30)

#space
t.penup()
t.left(120)
t.fd(40)
t.left(90)
t.fd(30)
t.right(90)
t.pendown()

#y
t.right(60)
t.fd(20)
t.left(120)
t.fd(20)
t.bk(40)
t.right(180)
t.circle(-7,130)

#space
t.penup()
t.right(110)
t.fd(60)
t.left(90)
t.fd(30)
t.right(90)
t.pendown()

#z
t.fd(30)
t.right(135)
t.fd(45)
t.left(135)
t.fd(30)
```

```
t.done()
```

2.2 Program Output



3 Capital Alphabets

Write the program in python to draw the capital alphabets A to Z.

3.1 Python Implementation

```
import turtle as t
t.pensize(5)
t.pencolor("crimson")
t.speed(20)

#move cursor
t.penup()
t.left (180)
t.fd(330)
t.right(90)
t.fd(230)
t.pendown()

#A
t.right(30)
t.fd(50)
t.right(120)
t.fd(50)
t.bk(15)
t.right(120)
t.fd(32)

#space
t.penup()
```

```
#next line
t.penup()
t.right(90)
t.fd(50)
t.right(90)
t.fd(630)
t.right(-90)
t.pendown()

#M
t.fd(50)
t.bk(50)
t.left(180)
t.right(150)
t.fd(35)
t.right(-120)
t.fd(35)
t.right(150)
t.fd(50)

#space
t.penup()
t.left(90)
t.fd(30)
```

```
t.left(180)
t.fd(55)
t.right(90)
t.fd(20)
t.left(180)
t.pendown()

#B
t.fd(50)
t.right(90)
t.fd(5)
t.circle(-15,150)
t.right(30)
t.fd(10)
t.bk(5)
t.right(180)
t.circle(-14,149)
t.left(-30)
t.fd(15)

#space
t.penup()
t.right(180)
t.fd(55)
t.left(90)
t.fd(50)
t.right(90)
t.fd(15)
t.pendown()

#C
t.penup()
t.left(-45)
t.fd(5)
t.pendown()
t.right(180)
t.circle(25,270)

#space
t.penup()
t.right(45)
t.fd(30)
t.right(90)
t.fd(15)
t.left(180)
t.left(90)
t.pendown()

#N
t.fd(50)
t.right(150)
t.fd(60)
t.left(150)
t.fd(50)

#space
t.penup()
t.right(90)
t.fd(50)
t.pendown()

#O
t.left(180)
t.circle(25,360)

#space
t.penup()
t.right(180)
t.fd(40)
t.pendown()

#P
t.right(90)
t.fd(50)
t.bk(50)
t.right(-90)
t.fd(5)
t.circle(-15,180)
t.fd(5)

#space
t.penup()
t.right(180)
t.fd(60)
t.left(90)
t.fd(30)
t.right(90)
t.pendown()

#Q
```

```
t.pendown()

#D
t.fd(50)
t.right(90)
t.fd(5)
t.circle(-25,180)
t.fd(5)

#space
t.penup()
t.right(180)
t.fd(50)
t.left(90)
t.fd(50)
t.pendown()

#E
t.right(180)
t.fd(50)
t.bk(50)
t.left(90)
t.fd(28)
t.bk(28)
t.right(90)
t.fd(25)
t.left(90)
t.fd(15)
t.bk(15)
t.right(90)
t.fd(25)
t.left(90)
t.fd(28)

#space
t.penup()
t.fd(25)
t.left(90)
t.fd(50)
t.pendown()

#F
t.right(180)
t.fd(50)
t.bk(50)
```

```
t.left(180)
t.circle(25,210)
t.right(90)
t.fd(15)
t.bk(40)
t.fd(25)
t.left(90)
t.circle(25,150)

#space
t.penup()
t.right(180)
t.fd(50)
t.pendown()

#R
t.right(90)
t.fd(50)
t.bk(50)
t.left(90)
t.fd(5)
t.circle(-15,180)
t.fd(5)
t.bk(7)
t.left(120)
t.fd(25)

#space
t.penup()
t.left(60)
t.fd(50)
t.left(90)
t.fd(50)
t.pendown()

#S
t.left(90)
t.fd(20)
t.circle(10,150)
t.forward(20)
t.circle(
-10,150)
t.forward(20)

#space
```

```
t.left(90)
t.fd(28)
t.bk(28)
t.right(90)
t.fd(25)
t.left(90)
t.fd(15)

#space
t.penup()
t.fd(28)
t.left(90)
t.fd(30)
t.right(90)
t.fd(20)
t.pendown()

#G
t.penup()
t.right(45)
t.fd(10)
t.pendown()
t.right(150)
t.circle(30,210)
t.left(75)
t.fd(20)
t.left(90)
t.fd(10)

#space
t.penup()
t.right(180)
t.fd(30)
t.left(90)
t.fd(35)
t.right(90)
t.pendown()

#H
t.right(90)
t.forward(50)
t.backward(25)
t.right(-90)
t.forward(25)
t.left(90)
```

```
t.penup()
t.right(180)
t.fd(60)
t.left(90)
t.fd(50)
t.pendown()

#T
t.right(90)
t.fd(50)
t.bk(25)
t.right(90)
t.fd(50)

#space
t.penup()
t.left(90)
t.fd(50)
t.left(90)
t.fd(50)
t.pendown()

#U
t.right(180)
t.fd(40)
t.circle(15,180)
t.fd(40)

#space
t.penup()
t.right(90)
t.fd(30)
t.pendown()

#V
t.right(60)
t.fd(55)
t.left(120)
t.fd(55)

#next line
t.penup()
t.right(150)
t.fd(90)
t.right(90)
```

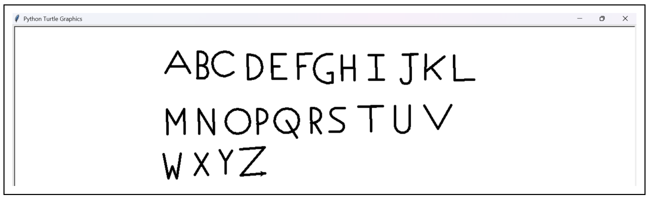
```
t.forward(25)
t.backward(50)

#space
t.penup()
t.right(90)
t.fd(30)
t.left(90)
t.fd(50)
t.right(90)
t.pendown()

#I
t.forward(30)
t.backward(15)
t.right(90)
t.forward(50)
t.left(-90)
t.forward(15)
t.backward(30)

#space
t.penup()
t.right(180)
t.fd(40)
t.left(90)
t.fd(50)
t.right(90)
t.pendown()

#J
t.fd(30)
t.bk(15)
t.right(90)
t.fd(50)
t.circle(-10, 182)

#space
t.penup()
t.right(90)
t.fd(50)
t.left(90)
t.fd(50)
t.pendown()

```

```
t.fd(590)
t.right(180)
t.pendown()

#W
t.right(80)
t.fd(50)
t.left(150)
t.fd(25)
t.right(140)
t.fd(25)
t.left(150)
t.fd(50)

#space
t.penup()
t.right(80)
t.fd(30)
t.pendown()

#X
t.right(60)
t.fd(50)
t.left(150)
t.penup()
t.fd(43)
t.pendown()
t.left(148)
t.fd(50)

#space
t.penup()
t.left(120)
t.fd(50)
t.left(90)
t.fd(50)
t.right(90)
t.pendown()

#Y
t.right(60)
t.fd(25)
t.left(120)
t.fd(25)
t.bk(25)
```

```
#K
t.right(180)
t.forward(50)
t.backward(15)
t.left(130)
t.forward(50)
t.backward(35)
t.right(90)
t.forward(30)

#space
t.penup()
t.left(46)
t.fd(25)
t.left(91)
t.fd(50)
t.pendown()

#L
t.right(175)
t.fd(50)
t.left(90)
t.fd(40)
```

```

t.right(150)
t.fd(25)

#space
t.penup()
t.left(90)
t.fd(30)
t.left(90)
t.fd(50)
t.right(85)
t.pendown()

#Z
t.fd(50)
t.right(135)
t.fd(75)
t.left(135)
t.fd(50)

t.done()
```

3.2 Program Output



4 Line algorithms

Write a Python program to draw a line using the DDA and Bresenham algorithms.

4.1 Digital Differential Analyzer (DDA) algorithm

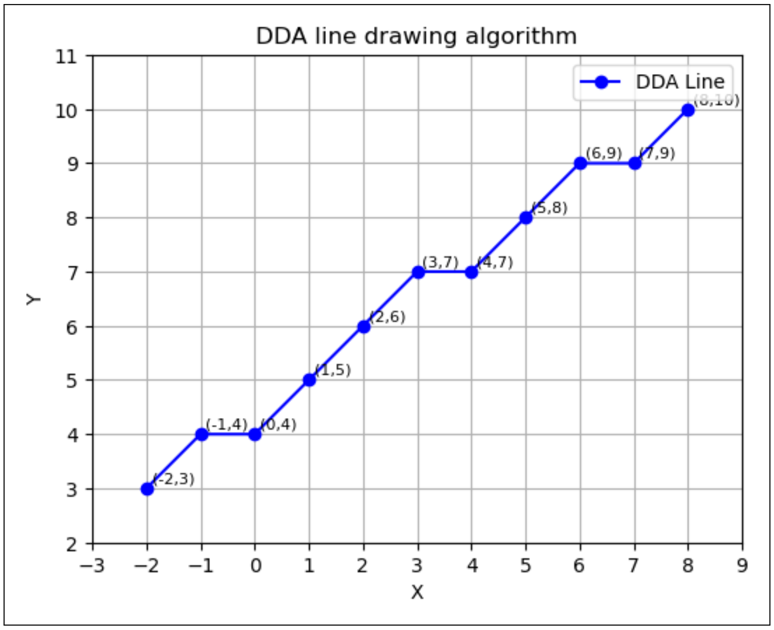
4.1.1 Python Implementation

```
import matplotlib.pyplot as plt

def dda(x0, y0, x1, y1):
   dx = x1 - x0
   dy = y1 - y0
   steps = max(abs(dx), abs(dy))

   x_increment = dx / steps
   y_increment = dy / steps

   x, y = x0, y0
   points = []

   for _ in range(int(steps) + 1):
      points.append((round(x), round(y)))
      x += x_increment
      y += y_increment

   return points


x_start = float(input("Enter starting x: "))
y_start = float(input("Enter starting y: "))
x_end = float(input("Enter ending x: "))
y_end = float(input("Enter ending y: "))

points = dda(x_start, y_start, x_end, y_end)

x_vals = [pt[0] for pt in points]
y_vals = [pt[1] for pt in points]

plt.figure(figsize=(6, 6))
plt.plot(x_vals, y_vals, ’bo-’, label=’DDA Line’)
plt.grid(True)
plt.title(’DDA Line Drawing Algorithm’)
plt.xlabel(’X’)
plt.ylabel(’Y’)
plt.xticks(range(int(min(x_vals) - 1), int(max(x_vals) + 2)))
plt.yticks(range(int(min(y_vals) - 1), int(max(y_vals) + 2)))
plt.legend()
plt.gca().set_aspect('equal', adjustable='box')

for (x, y) in points:
   plt.text(x + 0.1, y + 0.1, f'({x},{y})', fontsize=8)

plt.show()
```

4.1.2 Program Input

```
Enter starting x: -2
Enter starting y: 3
Enter ending x: 8
Enter ending y: 10
```

4.1.3 Program Output



4.2 Bresenham Line drawing algorithm

4.2.1 Python Implementation

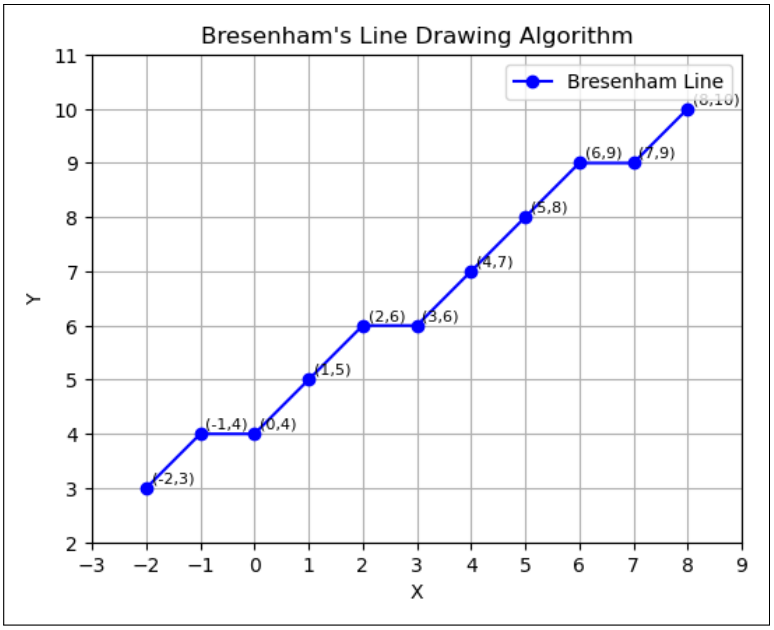
```
import matplotlib.pyplot as plt

def bresenham(x0, y0, x1, y1):
   points = []

   dx = abs(x1 - x0)
   dy = abs(y1 - y0)
   x, y = x0, y0

   sx = 1 if x1 > x0 else -1
   sy = 1 if y1 > y0 else -1

   if dx > dy:
      err = dx // 2
      while x != x1:
         points.append((x, y))
         err -= dy
         if err < 0:
            y += sy
            err += dx
         x += sx
   else:
      err = dy // 2
      while y != y1:
         points.append((x, y))
         err -= dx
         if err < 0:
            x += sx
            err += dy
         y += sy

   points.append((x1, y1))
   return points

x_start = int(input("Enter starting x: "))
y_start = int(input("Enter starting y: "))
x_end = int(input("Enter ending x: "))
y_end = int(input("Enter ending y: "))

points = bresenham(x_start, y_start, x_end, y_end)

x_vals = [pt[0] for pt in points]
y_vals = [pt[1] for pt in points]
```

```
plt.figure(figsize=(6, 6))
plt.plot(x_vals, y_vals, 'bo-', label="Bresenham Line")
plt.grid(True)
plt.title("Bresenham's Line Drawing Algorithm")
plt.xlabel("X")
plt.ylabel("Y")
plt.xticks(range(min(x_vals)-1, max(x_vals)+2))
plt.yticks(range(min(y_vals)-1, max(y_vals)+2))
plt.legend()
plt.gca().set_aspect('equal', adjustable='box')

for (x, y) in points:
   plt.text(x + 0.1, y + 0.1, f'({x},{y})', fontsize=8)

plt.show()
```

4.2.2 Program Input

```
Enter starting x: -2
Enter starting y: 3
Enter ending x: 8
Enter ending y: 10
```

4.2.3 Program Output



5 Circle algorithms

Write a Python program to draw a circle using the midpoint circle algorithm and the Bresenham algorithm.

5.1 Midpoint Circle drawing Algorithm

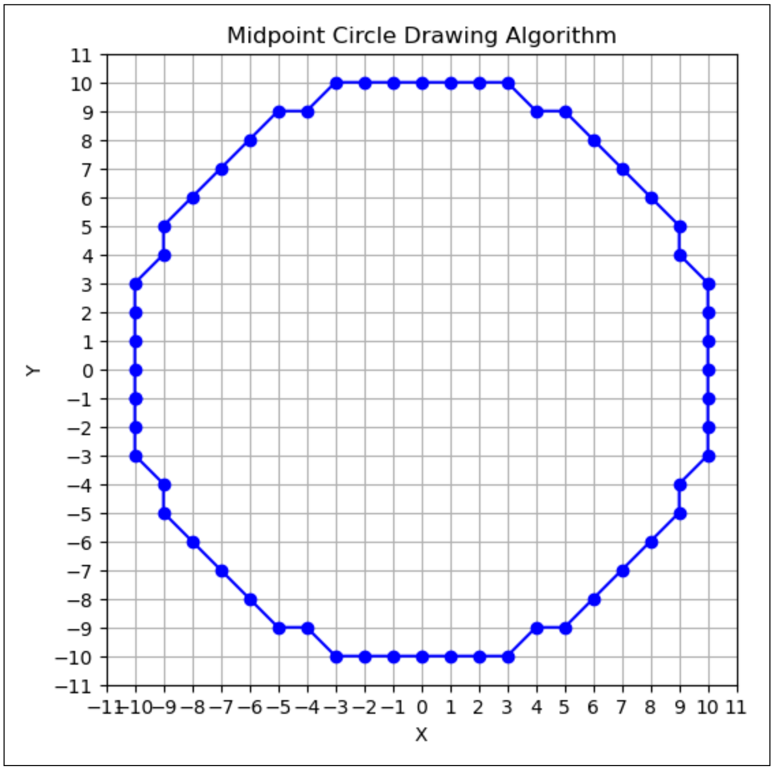
5.1.1 Python Implementation

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

def midpoint_circle(cx, cy, r):
   x = 0
   y = r
   d = 1 - r
   points = []

   def plot_circle_points(cx, cy, x, y):
      return [
         (cx + x, cy + y), (cx - x, cy + y),
         (cx + x, cy - y), (cx - x, cy - y),
         (cx + y, cy + x), (cx - y, cy + x),
         (cx + y, cy - x), (cx - y, cy - x),
      ]

   while x <= y:
      points.extend(plot_circle_points(cx, cy, x, y))
      x += 1
      if d < 0:
         d += 2 * x + 1
      else:
         y -= 1
         d += 2 * (x - y) + 1

   unique_points = sorted(set(points), key=lambda p: np.arctan2(p[1] - cy,
      p[0] - cx))
   return unique_points

cx = int(input("Enter circle center x (cx): "))
cy = int(input("Enter circle center y (cy): "))
radius = int(input("Enter radius: "))

circle_points = midpoint_circle(cx, cy, radius)
x_vals, y_vals = zip(*circle_points)

plt.figure(figsize=(6, 6))
plt.plot(x_vals + (x_vals[0],), y_vals + (y_vals[0],), color=’blue’,
    marker=’o’)
plt.title("Midpoint Circle Drawing Algorithm")
plt.xlabel("X")
plt.ylabel("Y")
plt.grid(True)
plt.gca().set_aspect(’equal’, adjustable=’box’)
plt.xticks(range(min(x_vals) - 1, max(x_vals) + 2))
plt.yticks(range(min(y_vals) - 1, max(y_vals) + 2))
plt.show()
```

5.1.2 Program Input

```
Enter circle center x (cx): 0
Enter circle center y (cy): 0
Enter radius: 10
```

5.1.3 Program Output



5.2 Bresenham Circle Drawing Algorithm

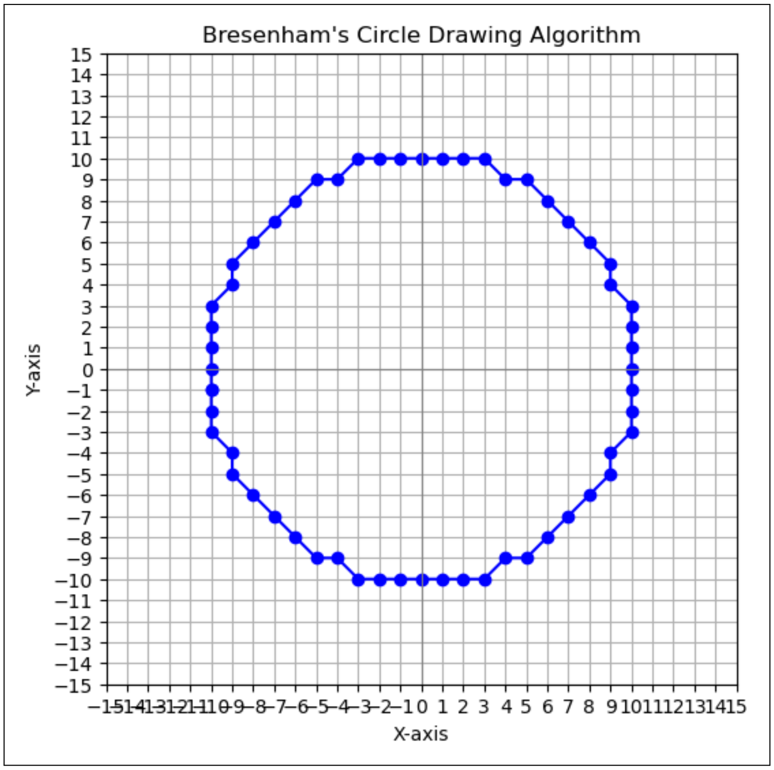
5.2.1 Python Implementation

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

def bresenham_circle(cx, cy, r):
   x = 0
   y = r
   d = 3 - 2 * r
   points = []

   def plot_circle_points(cx, cy, x, y):
      return [
         (cx + x, cy + y), (cx - x, cy + y),
         (cx + x, cy - y), (cx - x, cy - y),
         (cx + y, cy + x), (cx - y, cy + x),
         (cx + y, cy - x), (cx - y, cy - x)
      ]

   while x <= y:
      points.extend(plot_circle_points(cx, cy, x, y))
      if d < 0:
         d += 4 * x + 6
      else:
         d += 4 * (x - y) + 10
         y -= 1
      x += 1

   unique_points = sorted(set(points), key=lambda p: np.arctan2(p[1] - cy,
       p[0] - cx))
   return unique_points

cx = int(input("Enter circle center x (cx): "))
cy = int(input("Enter circle center y (cy): "))
radius = int(input("Enter radius: "))
padding = 5 # padding around circle in the plot

circle_points = bresenham_circle(cx, cy, radius)
x_vals, y_vals = zip(*circle_points)

plt.figure(figsize=(6, 6))
plt.plot(x_vals + (x_vals[0],), y_vals + (y_vals[0],), color=’blue’,
    marker=’o’)
plt.title("Bresenham’s Circle Drawing Algorithm")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(True)
plt.gca().set_aspect('equal', adjustable='box')

plt.axhline(0, color='gray', linewidth=0.8)
plt.axvline(0, color='gray', linewidth=0.8)

plt.xlim(cx - radius - padding, cx + radius + padding)
plt.ylim(cy - radius - padding, cy + radius + padding)
plt.xticks(range(cx - radius - padding, cx + radius + padding + 1))
plt.yticks(range(cy - radius - padding, cy + radius + padding + 1))

plt.show()
```

5.2.2 Program Input

```
Enter circle center x (cx): 0
Enter circle center y (cy): 0
Enter radius: 10
```

5.2.3 Program Output



6 Ellipse Drawing Algorithm

Write a Python program to draw an ellipse using the midpoint ellipse drawing algorithm.

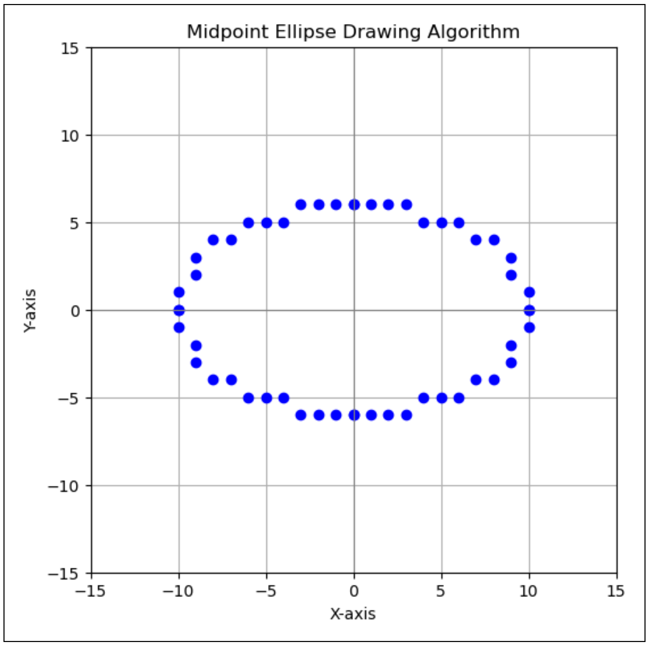
6.1 Python Implementation

```
import matplotlib.pyplot as plt

def midpoint_ellipse(cx, cy, rx, ry):
   points = []

   x = 0
   y = ry

   # Initial decision parameter of region 1
   ry2 = ry * ry
   rx2 = rx * rx
   d1 = ry2 - (rx2 * ry) + (0.25 * rx2)
   dx = 2 * ry2 * x
   dy = 2 * rx2 * y

   # Region 1
   while dx < dy:
      # 4-way symmetry
      points.extend([
         (cx + x, cy + y), (cx - x, cy + y),
         (cx + x, cy - y), (cx - x, cy - y)
      ])
      if d1 < 0:
         x += 1
         dx = dx + (2 * ry2)
         d1 = d1 + dx + ry2
      else:
         x += 1
         y -= 1
         dx = dx + (2 * ry2)
         dy = dy - (2 * rx2)
         d1 = d1 + dx - dy + ry2

   # Region 2
   d2 = (ry2) * ((x + 0.5) ** 2) + (rx2) * ((y - 1) ** 2) - (rx2 * ry2)

   while y >= 0:
      points.extend([
         (cx + x, cy + y), (cx - x, cy + y),
         (cx + x, cy - y), (cx - x, cy - y)
      ])
      if d2 > 0:
         y -= 1
         dy = dy - (2 * rx2)
         d2 = d2 + rx2 - dy
      else:
         y -= 1
         x += 1
         dx = dx + (2 * ry2)
         dy = dy - (2 * rx2)
         d2 = d2 + dx - dy + rx2

   return points

cx = int(input("Enter center x (cx): "))
cy = int(input("Enter center y (cy): "))
rx = int(input("Enter horizontal radius (rx): "))
ry = int(input("Enter vertical radius (ry): "))


points = midpoint_ellipse(cx, cy, rx, ry)
x_vals, y_vals = zip(*points)


plt.figure(figsize=(6, 6))
plt.plot(x_vals, y_vals, ’bo’) # plotted as discrete blue points
plt.title("Midpoint Ellipse Drawing Algorithm")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(True)
plt.gca().set_aspect(’equal’, adjustable=’box’)

plt.axhline(0, color=’gray’, linewidth=0.8)
plt.axvline(0, color=’gray’, linewidth=0.8)

padding = max(rx, ry) + 5
plt.xlim(cx - padding, cx + padding)
plt.ylim(cy - padding, cy + padding)
plt.show()
```

6.2 Program Input

```
Enter center x (cx): 0
Enter center y (cy): 0
Enter horizontal radius (rx): 10
Enter vertical radius (ry): 6
```

6.3 Program Output



7 Window and Viewport

Write a Python programme for the window to the viewport.

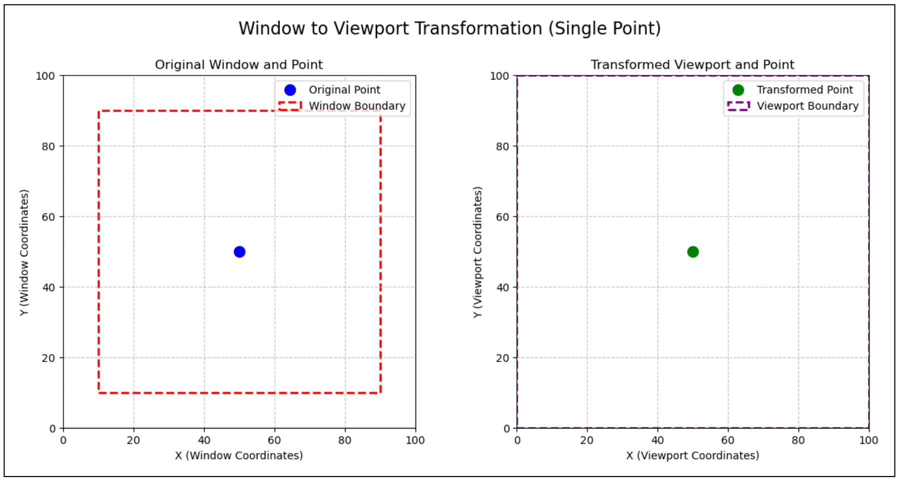
7.1 Python Implementation

```
import matplotlib.pyplot as plt

def window_to_viewport(wx, wy, W_xmin, W_ymin, W_xmax, W_ymax, V_xmin, V_ymin,
    V_xmax, V_ymax):

   # Calculate scaling factors for X and Y axes
   sx = (V_xmax - V_xmin) / (W_xmax - W_xmin)
   sy = (V_ymax - V_ymin) / (W_ymax - W_ymin)

   # Calculate transformed X and Y coordinates
   vx = V_xmin + (wx - W_xmin) * sx
   vy = V_ymin + (wy - W_ymin) * sy

   return vx, vy

def main():
   # 1. Define Window and Viewport Coordinates
   # Window (World Coordinates)
   W_xmin, W_ymin = 10, 10
```

```
W_xmax, W_ymax = 90, 90

# Viewport (Device/Screen Coordinates - relative to subplot)
V_xmin, V_ymin = 0, 0
V_xmax, V_ymax = 100, 100

# 2. Define a single point in window coordinates
point_window_x, point_window_y = 50, 50 # Center of the window

# 3. Apply the transformation to the point
point_viewport_x, point_viewport_y = window_to_viewport(
   point_window_x, point_window_y,
   W_xmin, W_ymin, W_xmax, W_ymax,
   V_xmin, V_ymin, V_xmax, V_ymax
)

# 4. Plotting with Matplotlib
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
fig.suptitle('Window to Viewport Transformation (Single Point)',
    fontsize=16)

# Plot 1: Original Window and Point
ax1.plot(point_window_x, point_window_y, 'o', color='blue', markersize=10,
    label='Original Point')
ax1.set_title('Original Window and Point')
ax1.set_xlabel('X (Window Coordinates)')
ax1.set_ylabel('Y (Window Coordinates)')
ax1.set_xlim(0, 100)
ax1.set_ylim(0, 100)
ax1.grid(True, linestyle='--', alpha=0.7)
# Draw the window boundary
ax1.add_patch(plt.Rectangle((W_xmin, W_ymin), W_xmax - W_xmin, W_ymax -
    W_ymin,
                     fill=False, edgecolor='red', linewidth=2,
                        linestyle='--', label='Window Boundary'))
ax1.set_aspect('equal', adjustable='box')
ax1.legend()

# Plot 2: Transformed Viewport and Point
ax2.plot(point_viewport_x, point_viewport_y, 'o', color='green',
    markersize=10, label='Transformed Point')
ax2.set_title('Transformed Viewport and Point')
ax2.set_xlabel('X (Viewport Coordinates)')
ax2.set_ylabel('Y (Viewport Coordinates)')
ax2.set_xlim(0, 100)
ax2.set_ylim(0, 100)
ax2.grid(True, linestyle='--', alpha=0.7)
```

```
    # Draw the viewport boundary
    ax2.add_patch(plt.Rectangle((V_xmin, V_ymin), V_xmax - V_xmin, V_ymax -
        V_ymin,
                         fill=False, edgecolor='purple', linewidth=2,
                            linestyle='--', label='Viewport Boundary'))
    ax2.set_aspect('equal', adjustable='box')
    ax2.legend()

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()

if __name__ == "__main__":
    main()
```

7.2 Program Output



8 Line Clipping

Write a Python program to perform line clipping using the Cohen–Sutherland algorithm.

8.1 Python Implementation

```
import matplotlib.pyplot as plt

# Region codes
INSIDE = 0
LEFT  = 1
RIGHT = 2
BOTTOM = 4
TOP   = 8
```

```python
# Fixed clipping window
x_min, y_min = 2, 2
x_max, y_max = 8, 6

def compute_code(x, y):
   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(x0, y0, x1, y1):
   code0 = compute_code(x0, y0)
   code1 = compute_code(x1, y1)
   accept = False

   while True:
      if code0 == 0 and code1 == 0:
         accept = True
         break
      elif (code0 & code1) != 0:
         break
      else:
         code_out = code0 if code0 != 0 else code1
         if code_out & TOP:
            x = x0 + (x1 - x0) * (y_max - y0) / (y1 - y0)
            y = y_max
         elif code_out & BOTTOM:
            x = x0 + (x1 - x0) * (y_min - y0) / (y1 - y0)
            y = y_min
         elif code_out & RIGHT:
            y = y0 + (y1 - y0) * (x_max - x0) / (x1 - x0)
            x = x_max
         elif code_out & LEFT:
            y = y0 + (y1 - y0) * (x_min - x0) / (x1 - x0)
            x = x_min

         if code_out == code0:
            x0, y0 = x, y
            code0 = compute_code(x0, y0)
         else:
            x1, y1 = x, y
            code1 = compute_code(x1, y1)

   return (x0, y0, x1, y1) if accept else None
```

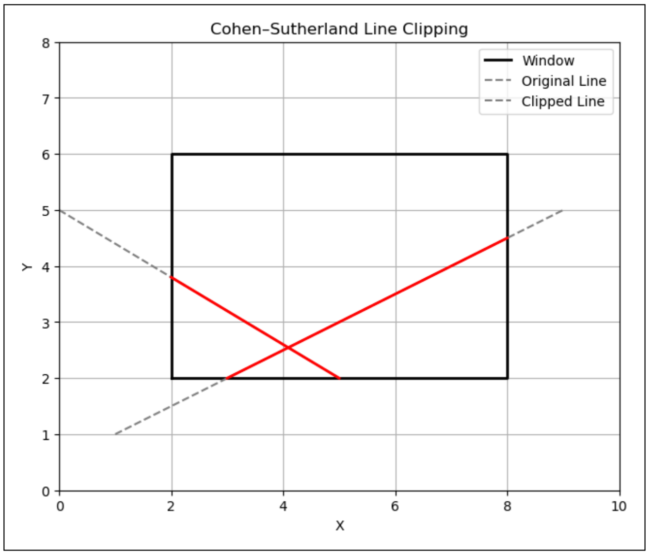
```
lines = []
for i in range(2):
   print(f"Enter Line {i+1} coordinates:")
   x0 = float(input(" x0: "))
   y0 = float(input(" y0: "))
   x1 = float(input(" x1: "))
   y1 = float(input(" y1: "))
   lines.append((x0, y0, x1, y1))

# Plot
plt.figure(figsize=(8, 6))
plt.plot([x_min, x_max, x_max, x_min, x_min],
       [y_min, y_min, y_max, y_max, y_min], ’k-’, linewidth=2, label=’Window’)

# Plot original lines (gray dashed)
for x0, y0, x1, y1 in lines:
   plt.plot([x0, x1], [y0, y1], ’--’, color=’gray’, label=’Original Line’)

# Plot clipped lines (red solid)
for x0, y0, x1, y1 in lines:
   clipped = cohen_sutherland_clip(x0, y0, x1, y1)
   if clipped:
      cx0, cy0, cx1, cy1 = clipped
      plt.plot([cx0, cx1], [cy0, cy1], ’-’, color=’red’, linewidth=2,
         label=’Clipped Line’)

# Display
plt.title("Cohen-Sutherland Line Clipping")
plt.xlabel("X")
plt.ylabel("Y")
plt.grid(True)
plt.legend(["Window", "Original Line", "Clipped Line"])
plt.xlim(0, 10)
plt.ylim(0, 8)
plt.gca().set_aspect(’equal’, adjustable=’box’)
plt.show()
```

8.2 Program Input

```
Enter Line 1 coordinates:
  x0: 1
  y0: 1
  x1: 9
  y1: 5

Enter Line 2 coordinates:
  x0: 0
  y0: 5
```

```
   x1: 5
   y1: 2
```

8.3 Program Output



9 Point Clipping

Write a Python program to perform point clipping.

9.1 Python Implementation

```
import matplotlib.pyplot as plt

# Get clipping window coordinates from user
xmin = float(input("Enter xmin of window: "))
ymin = float(input("Enter ymin of window: "))
xmax = float(input("Enter xmax of window: "))
ymax = float(input("Enter ymax of window: "))

# Get number of points from user
n = int(input("Enter number of points: "))

points = []
for i in range(n):
   print(f"Point {i+1}:")
   x = float(input(" x: "))
   y = float(input(" y: "))
   points.append((x, y))
```

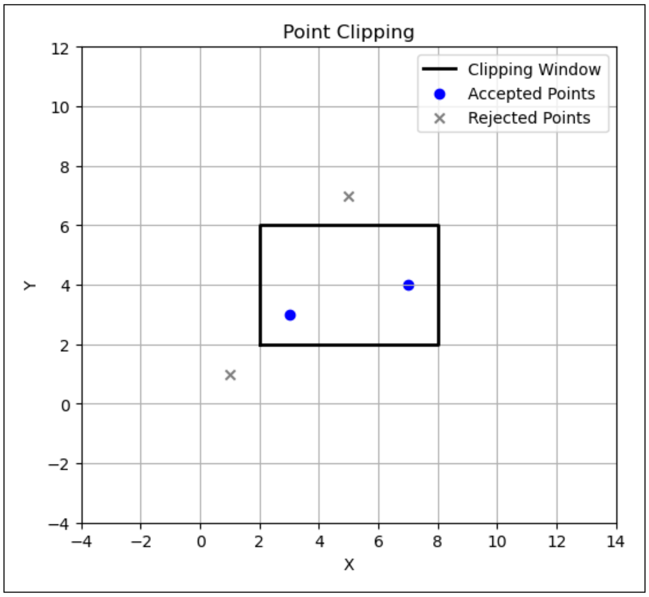
```
# Perform point clipping
clipped_points = []
outside_points = []

for x, y in points:
   if xmin <= x <= xmax and ymin <= y <= ymax:
      clipped_points.append((x, y))
   else:
      outside_points.append((x, y))

plt.figure(figsize=(6, 6))

# Draw clipping window
plt.plot([xmin, xmax, xmax, xmin, xmin],
       [ymin, ymin, ymax, ymax, ymin], ’k-’, linewidth=2, label="Clipping
          Window")

# Plot inside points (blue)
if clipped_points:
   x_in, y_in = zip(*clipped_points)
   plt.scatter(x_in, y_in, c=’blue’, label=’Accepted Points’)

# Plot outside points (gray)
if outside_points:
   x_out, y_out = zip(*outside_points)
   plt.scatter(x_out, y_out, c=’gray’, marker=’x’, label=’Rejected Points’)


plt.title("Point Clipping")
plt.xlabel("X")
plt.ylabel("Y")
plt.grid(True)
plt.legend()
plt.gca().set_aspect(’equal’, adjustable=’box’)
plt.xlim(min(xmin - 5, *[x for x, _ in points]) - 1,
       max(xmax + 5, *[x for x, _ in points]) + 1)
plt.ylim(min(ymin - 5, *[y for _, y in points]) - 1,
       max(ymax + 5, *[y for _, y in points]) + 1)
plt.show()
```

9.2 Program Input

```
Enter xmin of window: 2
Enter ymin of window: 2
Enter xmax of window: 8
Enter ymax of window: 6
Enter number of points: 4
```

```
Point 1:
  x: 3
  y: 3
Point 2:
  x: 1
  y: 1
Point 3:
  x: 5
  y: 7
Point 4:
  x: 7
  y: 4
```

9.3 Program Output



10 2D transformations

Write a Python program to perform basic geometric transformations applied to a shape in a 2D plane.

- Translation
- Scaling
- Rotation
- Shearing
- Reflection

Here, I will use a square to perform the basic geometric transformations.

10.1 Common Setup (shared by all Transformation)

10.1.1 Python implementation

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

# Define the square using homogeneous coordinates
square = np.array([
   [0, 0, 1],
   [1, 0, 1],
   [1, 1, 1],
   [0, 1, 1],
   [0, 0, 1] # to close the square
]).T # shape (3, 5)

# Function to plot original and transformed shape
def plot_transformation(title, transformed):
   plt.figure()
   plt.plot(square[0, :], square[1, :], ’ro--’, label=’Original’)
   plt.plot(transformed[0, :], transformed[1, :], ’bo-’, label=’Transformed’)
   plt.title(title)
   plt.grid(True)
   plt.axis(’equal’)
   plt.legend()
   plt.axhline(0, color=’gray’)
   plt.axvline(0, color=’gray’)
   plt.show()
```

10.2 Translation

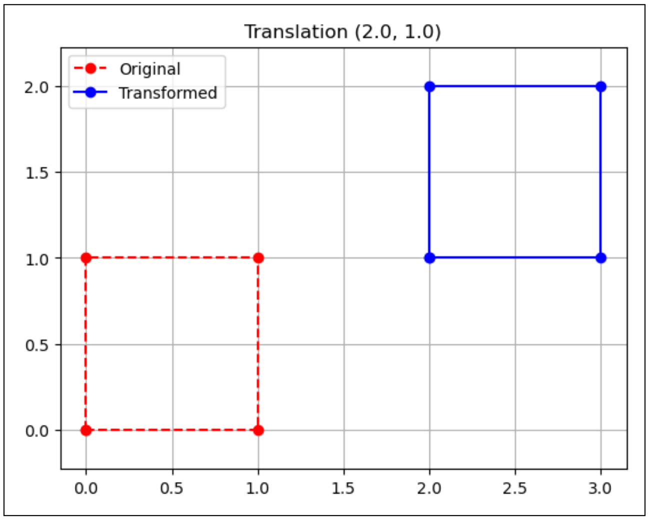
10.2.1 Python implementation

```
# Translation matrix
def translation(tx, ty):
    return np.array([[1, 0, tx],
                     [0, 1, ty],
                     [0, 0, 1]])

tx = float(input("Enter translation in X (tx): "))
ty = float(input("Enter translation in Y (ty): "))

translated = translation(tx, ty) @ square
plot_transformation(f"Translation ({tx}, {ty})", translated)
```

10.2.2 Program Input

```
Enter translation in X (tx): 2
Enter translation in Y (ty): 1
```

10.2.3 Program output



10.3 Scaling

10.3.1 Python implementation

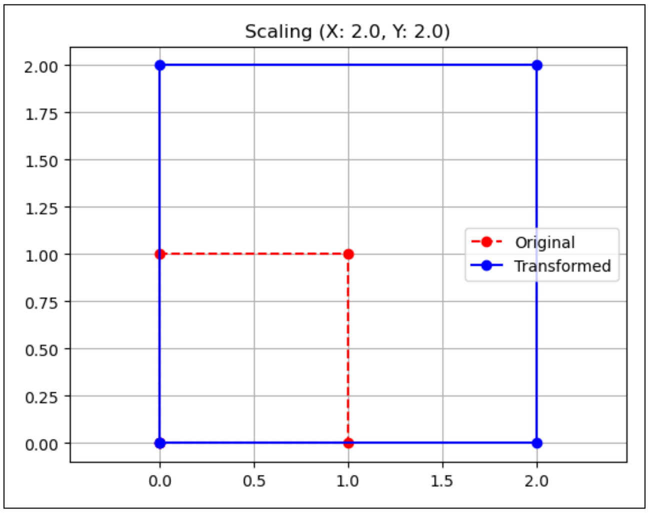
```
def scaling(sx, sy):
    return np.array([[sx, 0, 0],
                     [0, sy, 0],
                     [0, 0, 1]])
```

```python
sx = float(input("Enter scaling factor along X (sx): "))
sy = float(input("Enter scaling factor along Y (sy): "))

scaled = scaling(sx, sy) @ square
plot_transformation(f"Scaling (X: {sx}, Y: {sy})", scaled)
```

10.3.2 Program Input

```
Enter scaling factor along X (sx): 2
Enter scaling factor along Y (sy): 2
```

10.3.3 Program output



10.4 Rotation

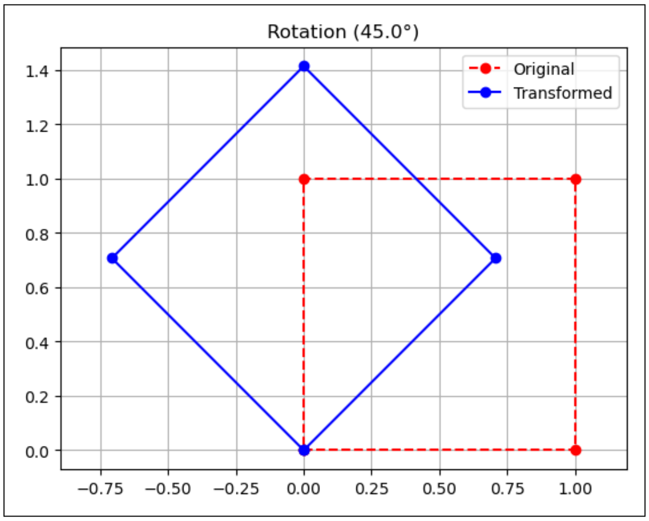
10.4.1 Python implementation

```python
def rotation(theta_degrees):
   theta = np.radians(theta_degrees)
   c, s = np.cos(theta), np.sin(theta)
   return np.array([[c, -s, 0],
                [s, c, 0],
                [0, 0, 1]])

angle = float(input("Enter rotation angle (in degrees): "))
rotated = rotation(angle) @ square
plot_transformation(f"Rotation ({angle}°)", rotated)
```

10.4.2 Program Input

```
Enter rotation angle (in degrees): 45
```

10.4.3 Program output



10.5 Shearing

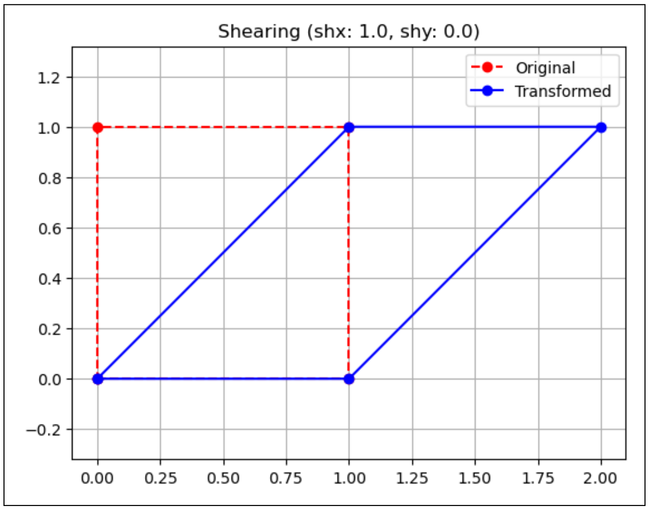
10.5.1 Python implementation

```
def shearing(shx=0, shy=0):
    return np.array([[1, shx, 0],
                     [shy, 1, 0],
                     [0,  0, 1]])

shx = float(input("Enter shearing in X (shx): "))
shy = float(input("Enter shearing in Y (shy): "))

sheared = shearing(shx, shy) @ square
plot_transformation(f"Shearing (shx: {shx}, shy: {shy})", sheared)
```

10.5.2 Program Input

```
Enter shearing in X (shx): 1
Enter shearing in Y (shy): 0
```

10.5.3 Program output


Shearing (shx: 1.0, shy: 0.0)

10.6 Reflection

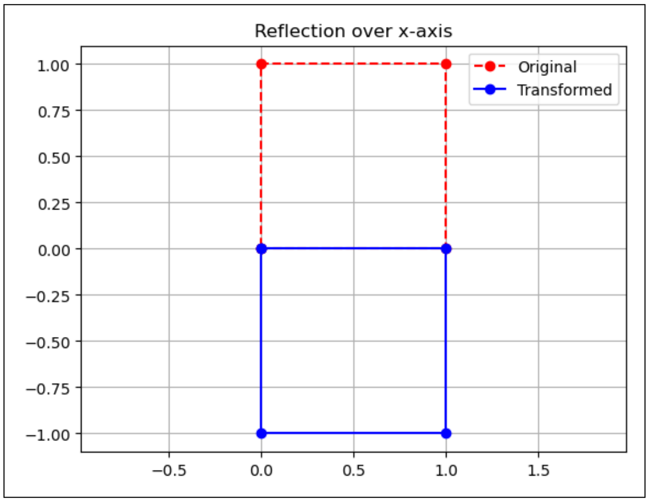
10.6.1 Python implementation

```
def reflection(axis=’x’):
   if axis == ’x’:
      return np.array([[1, 0, 0],
                     [0, -1, 0],
                     [0, 0, 1]])
   elif axis == ’y’:
      return np.array([[-1, 0, 0],
                     [0, 1, 0],
                     [0, 0, 1]])
   else:
      raise ValueError("Axis must be ’x’ or ’y’")

axis = input("Enter axis of reflection (x or y): ").lower()
reflected = reflection(axis) @ square
plot_transformation(f"Reflection over {axis}-axis", reflected)
```

10.6.2 Program Input

```
Enter axis of reflection (x or y): x
```

10.6.3 Program output



11 Bezier Curve

Write a Python program to generate and plot a Bezier curve from given control points.

11.1 Python Implementation

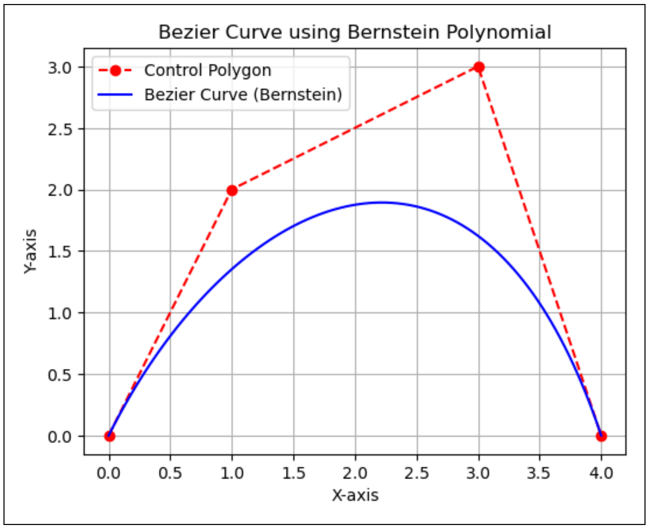
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import comb # for binomial coefficient

# Function to compute Bezier curve using Bernstein polynomials
def bernstein_bezier(control_points, num_points=100):
   n = len(control_points) - 1
   t = np.linspace(0, 1, num_points)
   curve = np.zeros((num_points, 2))

   for i in range(n + 1):
      binomial = comb(n, i)
      term = (binomial * ((1 - t) ** (n - i)) * (t ** i))[:, None] # shape: (num_points, 1)
      curve += term * control_points[i]

   return curve

n = int(input("Enter number of control points (at least 2): "))
control_points = []
for i in range(n):
   x = float(input(f"Enter x{i+1}: "))
   y = float(input(f"Enter y{i+1}: "))
```

```
    control_points.append([x, y])

control_points = np.array(control_points)
bezier_curve = bernstein_bezier(control_points)

plt.figure(figsize=(6, 6))
plt.plot(control_points[:, 0], control_points[:, 1], ’ro--’, label=’Control Polygon’)
plt.plot(bezier_curve[:, 0], bezier_curve[:, 1], ’b-’, label=’Bezier Curve (Bernstein)’)
plt.title("Bezier Curve using Bernstein Polynomial")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(True)
plt.gca().set_aspect(’equal’, adjustable=’box’)
plt.legend()
plt.show()
```

11.2 Program Input

```
Enter number of control points (at least 2): 4
Enter x1: 0
Enter y1: 0
Enter x2: 1
Enter y2: 2
Enter x3: 3
Enter y3: 3
Enter x4: 4
Enter y4: 0
```

11.3 Program Output


Bezier Curve using Bernstein Polynomial
Control Polygon
Bezier Curve (Bernstein)
Y-axis
X-axis