DDA:

```
import turtle

def dda_line(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    steps = max(abs(dx), abs(dy))

    x_inc = dx / steps
    y_inc = dy / steps

    for _ in range(steps + 1):
        turtle.goto(x1, y1)
        x1 += x_inc
        y1 += y_inc

turtle.speed(0)
dda_line(0, 0, 100, 100)  # Example line from (0, 0) to (100, 100)
turtle.done()
```

Bresenham

```
import turtle

def bresenham_line(x1, y1, x2, y2):
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    sx = 1 if x1 < x2 else -1
    sy = 1 if y1 < y2 else -1
    err = dx - dy

    while True:
        turtle.goto(x1, y1)
        if x1 == x2 and y1 == y2:
            break
        err2 = err * 2
        if err2 > -dy:
            err -= dy
            x1 += sx
        if err2 < dx:
            err += dx
            y1 += sy

turtle.speed(0)
bresenham_line(0, 0, 100, 100)  # Example line from (0, 0) to (100, 100)
turtle.done()
```

Midpoint:

```
import turtle

def draw_circle(radius):
    x, y = 0, radius
    d = 1 - radius

    while x < y:
        x += 1
        if d < 0:
            d += 2 * x + 1
        else:
            y -= 1
            d += 2 * (x - y) + 1

        plot_circle_points(x, y)

def plot_circle_points(x, y):
    # Plot points in all octants
    for point in [(x, y), (y, x), (-x, y), (-y, x),
                  (x, -y), (y, -x), (-x, -y), (-y, -x)]:
        turtle.penup()
        turtle.goto(point)
        turtle.dot(5, "black")

turtle.speed(0)
radius = 100
turtle.penup()
turtle.goto(0, -radius)  # Move to the starting point
turtle.pendown()

draw_circle(radius)

turtle.done()
```

Transfrom:

```
import turtle
import math

def draw_square(vertices):
    turtle.penup()
    turtle.goto(vertices[0])
    turtle.pendown()
    for vertex in vertices[1:]:
        turtle.goto(vertex)
    turtle.goto(vertices[0])

def translate(vertices, tx, ty):
    return [(x + tx, y + ty) for (x, y) in vertices]

def rotate(vertices, angle):
    angle_rad = math.radians(angle)
    cos_angle = math.cos(angle_rad)
    sin_angle = math.sin(angle_rad)
    return [
        (x * cos_angle - y * sin_angle, x * sin_angle + y * cos_angle)
        for (x, y) in vertices
    ]

def scale(vertices, sx, sy):
    return [(x * sx, y * sy) for (x, y) in vertices]

def reflect(vertices):
    return [(x, -y) for (x, y) in vertices]

def shear(vertices, sx, sy):
    return [(x + sx * y, y + sy * x) for (x, y) in vertices]

side_length = 100
square_vertices = [
    (0, 0),
    (side_length, 0),
    (side_length, side_length),
    (0, side_length)
]

turtle.speed(0)

draw_square(square_vertices)

# Translate
tx, ty = 200,200
turtle.color("blue")
translated_square = translate(square_vertices, tx, ty)
draw_square(translated_square)

# Rotate
angle = 60
turtle.color("red")
rotated_square = rotate(square_vertices, angle)
draw_square(rotated_square)

# Scale
```

```
sx, sy = 1.5, 1.5
turtle.color("green")
scaled_square = scale(square_vertices, sx, sy)
draw_square(scaled_square)

# Reflect
turtle.color("violet")
reflected_square = reflect(square_vertices)
draw_square(reflected_square)

# Shear
shear_x, shear_y = 0.6, 0.4
turtle.color("orange")
sheared_square = shear(square_vertices, shear_x, shear_y)
draw_square(sheared_square)

turtle.done()
```

---

cohen sutherland line clipping:

```
import turtle

t = turtle.Turtle()
t.speed(0)

x_min, y_min = -100, -100
x_max, y_max = 100, 100

def draw_clipping_window():
    t.penup()
    t.goto(x_min, y_min)
    t.pendown()
    t.goto(x_min, y_max)
    t.goto(x_max, y_max)
    t.goto(x_max, y_min)
    t.goto(x_min, y_min)

def clip_line(x1, y1, x2, y2):

    # draw original line
    t.penup()
    t.goto(x1,y1)
    t.color("blue")
    t.pendown()
    t.goto(x2,y2)
    t.color("red")
    t.penup()

    # Function to check if a point is inside the clipping window
    def is_inside(x, y):
        return x_min <= x <= x_max and y_min <= y <= y_max

    # If both endpoints are inside the clipping window
    if is_inside(x1, y1) and is_inside(x2, y2):
        t.penup()
        t.goto(x1, y1)
        t.pendown()
        t.goto(x2, y2)
        return

    m = (y2-y1)/(x2-x1)
    print(m)

    if x1<x_min: # left
        y1 = y1 + m*(x_min-x1)
        x1 = x_min

    if x2>x_max: # right
        y2 = y1 + m*(x_max-x1)
        x2 = x_max

    if y2>y_max: # top
        x2 = x1 + (y_max-y1)/m
        y2 = y_max

    if y1<y_min: # bottom
        x1 = x1 + (y_min-y1)/m
        y1 = y_min
```

```
    # Draw the clipped line
    t.penup()
    t.goto(x1, y1)
    t.pendown()
    t.goto(x2, y2)

draw_clipping_window()

clip_line(50, -250, -50, 150)
turtle.done()
```

---

bezier:

```
import turtle

def bezier_curve(points, num_points=100):
    n = len(points) - 1
    for t in range(num_points + 1):
        u = t / num_points
        x = sum(points[i][0] * bernstein(n, i, u) for i in range(n + 1))
        y = sum(points[i][1] * bernstein(n, i, u) for i in range(n + 1))
        turtle.goto(x, y)

def bernstein(n, i, t):
    return factorial(n) / (factorial(i) * factorial(n - i)) * (t ** i) * ((1 - t) ** (n - i))

def factorial(num):
    if num == 0:
        return 1
    result = 1
    for i in range(2, num + 1):
        result *= i
    return result

def draw_control_points(points):
    turtle.penup()
    for point in points:
        turtle.goto(point)
        turtle.dot(5, "red")  # Draw control points as red dots
    turtle.pendown()

# Example control points for the Bezier curve
control_points = [(0, 0), (50, 100), (100, 50), (-150, 100)]

turtle.speed(0)

draw_control_points(control_points)

turtle.penup()
turtle.goto(control_points[0])
turtle.pendown()
bezier_curve(control_points)

turtle.done()
```