实验一 ——椭圆的扫描转换算法

输入：椭圆的长短半径
输出：以原点为中心的椭圆图

核心源程序:

```
import turtle

def drawpixel(x, y):
    turtle.penup()
    turtle.goto(x, y)
    turtle.dot(3)

def ellipse(a, b):
    x, y = 0, b
    d1 = b * b + a * a * (-b + 0.25)
    drawpixel(x, y)
    while b * b * (x + 1) < a * a * (y - 0.5):
        if d1 < 0:
            d1 += b * b * (2 * x + 3)
            x += 1
        else:
            d1 += b * b * (2 * x + 3) + a * a * (-2 * y + 2)
            y -= 1
            x += 1
        drawpixel(x, y)
        drawpixel(x, -y)
        drawpixel(-x, y)
        drawpixel(-x, -y)

    d2 = b * b * (x + 0.5) * (x + 0.5) + a * a * (y - 1) * (y - 1) - a * a * b *
    while y > 0:
        if d2 < 0:
            d2 += b * b * (2 * x + 2) + a * a * (-y + 3)
            x += 1
            y -= 1
        else:
            d2 += a * a * (-2 * y + 3)
            y -= 1
        drawpixel(x, y)
        drawpixel(x, -y)
        drawpixel(-x, y)
        drawpixel(-x, -y)
```

```
def main():
    a = int(input("Please input the long radius:"))
    b = int(input("Please input the short radius:"))
    turtle.pencolor('red')
    turtle.hideturtle()
    turtle.speed(50)
    turtle.screensize(800, 600, 'white')
    ellipse(a, b)
    turtle.done()

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

运行结果截图:

实验二——凸六边形的扫描线填充算法

输入：凸六边形的六个顶点
输出：填充后的六边形图

核心源程序:

```
import numpy as np
import matplotlib.pyplot as plt
from math import *
import turtle

class Node(object):
```

```
    def __init__(self, x = None, y = None, increment = None, ymax = None):
        self.x = x
        self.y = y
        self.increment = increment
        self.ymax = ymax
        self.next = None


class LinkList(object):
    def __init__(self, node=None):
        self.head = node


def drawpixel(x, y, color='black'):
    turtle.pencolor(color)
    turtle.up()
    turtle.goto(x, y)
    turtle.dot(3)


def main():
    polygon = [20, 20, 20, 70, 50, 50, 110, 80, 110, 30, 50, 10]
    points_x, points_y = [], []
    for i in range(0, 11, 2):                    # 横纵坐标分离
        points_x.append(polygon[i])
        points_y.append(polygon[i+1])
    print(points_x)
    print(points_y)
    ymin, ymax = min(points_y), max(points_y)

    # 初始化新边表
    net = {}
    for i in range(ymin, ymax + 1):
        net[i] = []

    # 初始化各条边
    for i in range(len(points_x)):
        index1, index2 = i, (i+1) % len(points_x)
        y_min = min(points_y[index1], points_y[index2])
        y_max = max(points_y[index1], points_y[index2])
        if y_max == points_y[index1]:
            x2 = points_x[index1]
            x1 = points_x[index2]
        else:
            x2 = points_x[index2]
            x1 = points_x[index1]
        # 增量计算都是： 坐标点y值大的对应的x2 减去坐标点y值小的对应的x1
```

```
        increment = (x2 - x1) / (y_max - y_min)
        # 根据底顶点的y值来建立net表的索引
        net[y_min].append(Node(x1, y_min, increment, y_max))    # y值小的对应点添加
        # 输出每条扫描线的新边表
        for j in range(len(net[y_min])):
            print("edge:",net[y_min][j].x, net[y_min][j].increment, net[y_min][j

    # 建立活性边表
    aet = LinkList(Node())
    for y in range(ymin, ymax + 1):
        print("扫描线:   " , y)
        for i in range(len(net[y])):    # 遍历每一条扫描线经过的活性边
            data = net[y][i]
            head_node = aet.head
            if head_node.next == None:  # 为空，就直接插入
                head_node.next = data
            else:       # 按x递增的顺序插入到表中
                while data.x > head_node.next.x:     # 遍历链表找到可以插入的位置(插
                    head_node = head_node.next
                data.next = head_node.next           # 插入节点 data
                head_node.next = data

        # 遍历AET表，寻找填充区间
        fill_points = []
        head_node = aet.head
        while head_node.next != None:
            ymin = head_node.next.y
            ymax = head_node.next.ymax
            if y > ymin and y <= ymax:
                fill_points.append(head_node.next.x)
            elif y == ymin and y < ymax:
                fill_points.append(head_node.next.x)
                fill_points.append(head_node.next.x)

            head_node = head_node.next
        fill_points = sorted(fill_points)

        # 进行填充
        for i in range(0, len(fill_points), 2):
            # print("length: ", len(fill_points))
            x1, x2 = fill_points[i], fill_points[i+1]
            for x in range(int(x1), int(x2)):
                drawpixel(x, y)
        # 遍历aet进行测试
        head_node = aet.head
        while head_node != None:
            head_node = head_node.next
```

```
        # 遍历aet，把ymax = y的节点从aet删除，并把ymax > i结点的x值递增increment
        head_node = aet.head
        while head_node.next != None:
            if head_node.next.ymax == y:
                head_node.next = head_node.next.next
            else:
                head_node.next.x = head_node.next.x + head_node.next.increment
                head_node = head_node.next

if __name__ == "__main__":
    turtle.hideturtle()
    turtle.speed(10)
    turtle.screensize(600, 400, 'red')
    main()
    turtle.mainloop()
```

运行结果截图：

实验三——凸六边形绕点旋转

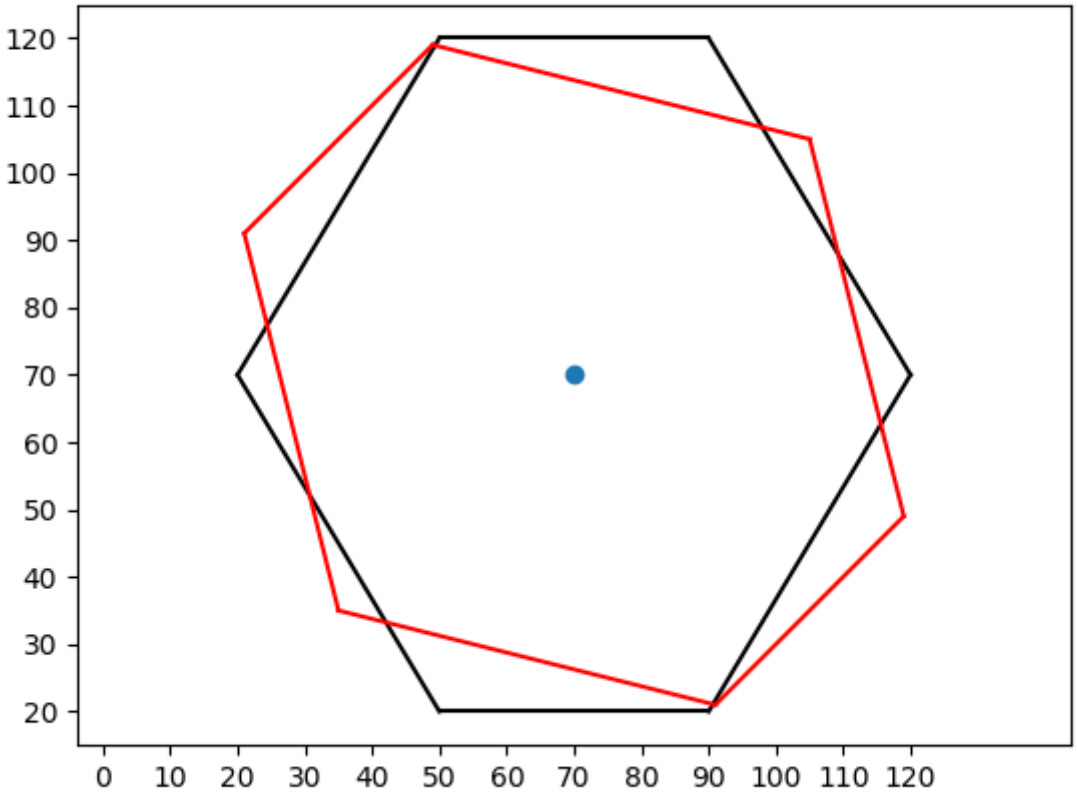
输入：凸六边形的六个顶点，旋转的角度
输出：原图六边形，旋转后的凸六边形

核心源程序：

```
from math import *
import numpy as np
import matplotlib.pyplot as plt

# 绘图函数
def draw(points, color):
```

```
        points_x, points_y = [], []
        for i in range(0, 11, 2):                          # 横纵坐标分离
                points_x.append(points[i])
                points_y.append(points[i+1])
        x, y = [-1, -1], [-1, -1]
        for i in range(0, 5):
                x[0] = points_x[i]
                x[1] = points_x[i+1]
                y[0] = points_y[i]
                y[1] = points_y[i+1]
                plt.plot(x, y, color)
        plt.plot([points_x[5], points_x[0]], [points_y[5], points_y[0]], color)

# 旋转函数
def rotate(points, angle, rotate_x, rotate_y):
    for i in range(0, 11, 2):  # 遍历 0 ~ 10，只取偶数位置即 x 值
        temp = points[i]
        points[i] = round((points[i] - rotate_x) * cos(radians(angle))
                          - (points[i + 1] - rotate_y) * sin(radians(angle))
                          + rotate_x)
        points[i + 1] = round((temp - rotate_x) * sin(radians(angle))
                          + (points[i + 1] - rotate_x) * cos(radians(angle))
                          + rotate_y)
    return points

# 六边形六个顶点
polygon = [50, 20, 90, 20, 120, 70, 90, 120, 50, 120, 20, 70]

x_ticks = np.arange(0, 130, 10)
y_ticks = np.arange(0, 130, 10)
plt.xlim(0, 130)                # 设置坐标轴范围
plt.ylim(0, 130)
plt.xticks(x_ticks)             # 设置坐标轴刻度
plt.yticks(y_ticks)
plt.axis('equal')               # 社会横纵坐标轴刻度间隔相等

draw(polygon, 'black')  # 画原凸六边形

rotate_polygon = rotate(polygon, 45, 70, 70)    # 绕点旋转45度

draw(rotate_polygon, 'red')             # 画旋转后的凸六边形
plt.scatter([70], [70])         # 画出旋转点
plt.show()
```

运行结果截图：



实验四——给定特征多边形，生成Bezier曲线

输入：特征六边形的六个顶点
输出：特征六边形以及对应的Bezier曲线

核心源程序：

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

# 绘图函数
def draw(points, color):
        points_x, points_y = [], []
        for i in range(0, 11, 2):                       # 横纵坐标分离
                points_x.append(points[i])
                points_y.append(points[i+1])
        x, y = [-1, -1], [-1, -1]
        for i in range(0, 5):
                x[0] = points_x[i]
                x[1] = points_x[i+1]
                y[0] = points_y[i]
                y[1] = points_y[i+1]
                plt.plot(x, y, color)
```

```
        plt.plot([points_x[5], points_x[0]], [points_y[5], points_y[0]], color)

# 线性bezier
def one_bezier(x, y, t):
    return (1 - t) * x + t * y

# n 次bezier曲线
# data--数据，n--阶数，k--索引
def n_bezier(data, n, k, t):
    if n == 1:
        return one_bezier(data[k], data[k+1], t)
    else:
        return (1 - t) * n_bezier(data, n-1, k, t) + t * n_bezier(data, n-1, k+1


def bezier(xs, ys, num, b_xs, b_ys):
    n = len(xs) - 1
    t_step = 1.0 / (num - 1)
    t_list = np.arange(0.0, 1 + t_step, t_step)
    for t in t_list:
        b_xs.append(n_bezier(xs, n, 0, t))
        b_ys.append(n_bezier(ys, n, 0, t))


def main():
    # 六边形六个顶点
    polygon = [50, 20, 90, 20, 120, 70, 90, 120, 50, 120, 20, 70]

    x_ticks = np.arange(0, 130, 10)
    y_ticks = np.arange(0, 130, 10)
    plt.xlim(0, 130)            # 设置坐标轴范围
    plt.ylim(0, 130)
    plt.xticks(x_ticks)         # 设置坐标轴刻度
    plt.yticks(y_ticks)
    plt.axis('equal')           # 社会横纵坐标轴刻度间隔相等

    draw(polygon, 'black')  # 画原凸六边形

    points_x, points_y = [], []
    for i in range(0, 11, 2):                   # 横纵坐标分离
        points_x.append(polygon[i])
        points_y.append(polygon[i+1])
    # 除了六个顶点，还有要把第一个顶点添加到最后
    points_x.append(50)
    points_y.append(20)

    num = 100
```

```
    b_xs, b_ys = [], []
    bezier(points_x, points_y, num, b_xs, b_ys)
    plt.plot(b_xs, b_ys)
    plt.show()

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

运行结果截图：