МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчёт по лабораторной работе №7

Специальность ПО11

Выполнил
Гулевич Е.А.
студент группы ПО11

Проверил
А. А. Крощенко
ст. преп. кафедры ИИТ,
04.05.2025 г.

Брест 2025

Цель работы: освоить возможности языка программирования Python в разработке оконных приложений

Вариант 7

Задание 1: Построение графических примитивов и надписей
Требования к выполнению
• Реализовать соответствующие классы, указанные в задании;
• Организовать ввод параметров для создания объектов (использовать экранные компоненты);
• Осуществить визуализацию графических примитивов
Важное замечание: должна быть предусмотрена возможность приостановки выполнения визуализации, изменения параметров «на лету» и снятия скриншотов с сохранением в текущую активную директорию.
Для всех динамических сцен необходимо задавать параметр скорости!
7) Изобразить в окне приложения отрезок, вращающийся в плоскости формы вокруг точки, движущейся по отрезку.
Код программы :

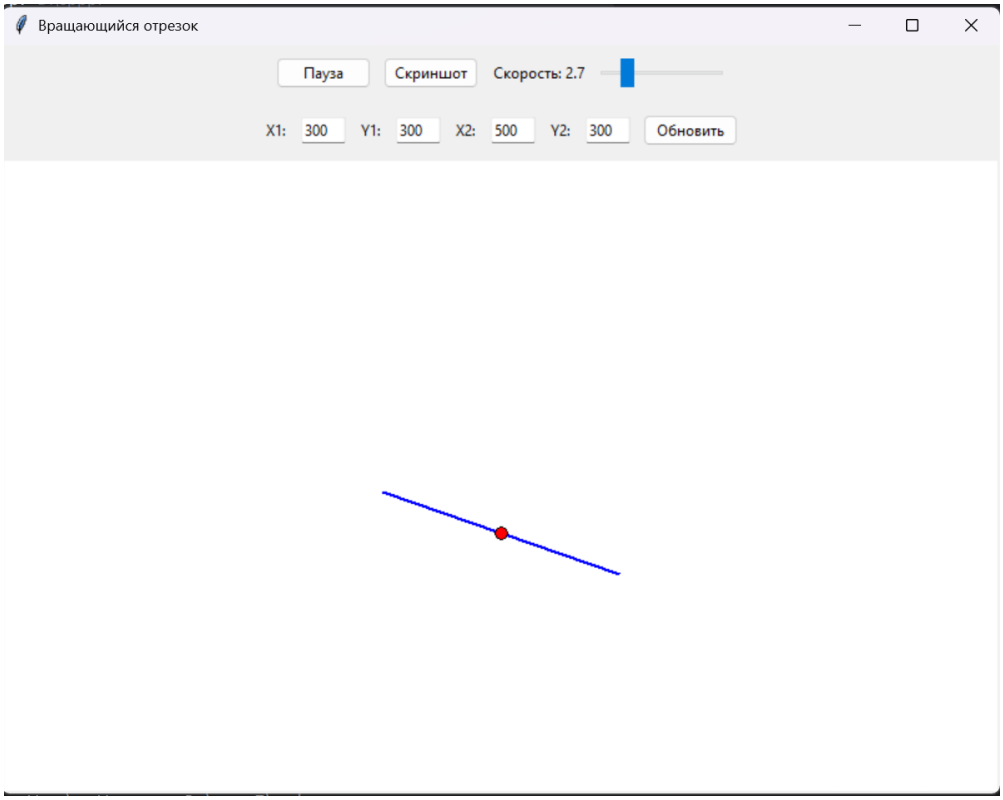
```
import tkinter as tk
from tkinter import ttk
from PIL import Image, ImageGrab
import math
import time


class RotatingLineApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Вращающийся отрезок")
        self.root.geometry("800x600")

        # Параметры анимации
        self.rotation_angle = 0
        self.point_position = 0
        self.rotation_speed = 1
        self.point_speed = 0.5
        self.is_paused = False

        # Координаты отрезка
        self.x1 = 300
        self.y1 = 300
        self.x2 = 500
        self.y2 = 300

        # Создание элементов управления
        self.create_controls()

        # Создание холста
        self.canvas = tk.Canvas(root, width=800, height=600, bg='white')
        self.canvas.pack()
```

```
        # Запуск анимации
        self.animate()

    def create_controls(self):
        control_frame = ttk.Frame(self.root)
        control_frame.pack(pady=10)

        # Кнопка паузы
        self.pause_button = ttk.Button(control_frame, text="Пауза", command=self.toggle_pause)
        self.pause_button.pack(side=tk.LEFT, padx=5)

        # Кнопка скриншота
        self.screenshot_button = ttk.Button(control_frame, text="Скриншот",
command=self.take_screenshot)
        self.screenshot_button.pack(side=tk.LEFT, padx=5)

        # Трекбар скорости
        self.speed_label = ttk.Label(control_frame, text="Скорость: 5")
        self.speed_label.pack(side=tk.LEFT, padx=5)

        self.speed_scale = ttk.Scale(control_frame, from_=1, to=10, orient=tk.HORIZONTAL,
command=self.update_speed)
        self.speed_scale.set(5)
        self.speed_scale.pack(side=tk.LEFT, padx=5)

        # Поля ввода координат
        coord_frame = ttk.Frame(self.root)
        coord_frame.pack(pady=10)

        # X1
        ttk.Label(coord_frame, text="X1:").pack(side=tk.LEFT, padx=5)
        self.x1_entry = ttk.Entry(coord_frame, width=5)
        self.x1_entry.insert(0, str(self.x1))
        self.x1_entry.pack(side=tk.LEFT, padx=5)

        # Y1
        ttk.Label(coord_frame, text="Y1:").pack(side=tk.LEFT, padx=5)
        self.y1_entry = ttk.Entry(coord_frame, width=5)
        self.y1_entry.insert(0, str(self.y1))
        self.y1_entry.pack(side=tk.LEFT, padx=5)

        # X2
        ttk.Label(coord_frame, text="X2:").pack(side=tk.LEFT, padx=5)
        self.x2_entry = ttk.Entry(coord_frame, width=5)
        self.x2_entry.insert(0, str(self.x2))
        self.x2_entry.pack(side=tk.LEFT, padx=5)

        # Y2
        ttk.Label(coord_frame, text="Y2:").pack(side=tk.LEFT, padx=5)
        self.y2_entry = ttk.Entry(coord_frame, width=5)
```

```
        self.y2_entry.insert(0, str(self.y2))
        self.y2_entry.pack(side=tk.LEFT, padx=5)

        # Кнопка обновления координат
        self.update_coords_button = ttk.Button(coord_frame, text="Обновить",
command=self.update_coordinates)
        self.update_coords_button.pack(side=tk.LEFT, padx=5)

    def update_coordinates(self):
        try:
            self.x1 = int(self.x1_entry.get())
            self.y1 = int(self.y1_entry.get())
            self.x2 = int(self.x2_entry.get())
            self.y2 = int(self.y2_entry.get())
        except ValueError:
            pass

    def toggle_pause(self):
        self.is_paused = not self.is_paused
        self.pause_button.config(text="Продолжить" if self.is_paused else "Пауза")

    def update_speed(self, value):
        try:
            speed = float(value)
            self.rotation_speed = speed
            self.speed_label.config(text=f"Скорость: {speed:.1f}")
        except ValueError:
            pass

    def take_screenshot(self):
        # Делаем скриншот всего экрана
        screenshot = ImageGrab.grab()

        # Создаем имя файла с текущим временем
        filename = f"screenshot_{int(time.time())}.png"

        # Сохраняем скриншот
        screenshot.save(filename)
        print(f"Скриншот сохранен как {filename}")

    def animate(self):
        if not self.is_paused:
            self.rotation_angle += self.rotation_speed * 0.1
            self.point_position += self.point_speed * 0.1
            if self.point_position > 1:
                self.point_position = 0

            self.draw_line()

        self.root.after(16, self.animate)  # ~60 FPS
```

```
    def draw_line(self):
        self.canvas.delete("all")

        # Центр отрезка
        center_x = (self.x1 + self.x2) / 2
        center_y = (self.y1 + self.y2) / 2

        # Длина отрезка
        dx = self.x2 - self.x1
        dy = self.y2 - self.y1
        length = math.sqrt(dx*dx + dy*dy)

        # Угол наклона отрезка
        angle = math.atan2(dy, dx)

        # Новые координаты с учетом вращения
        new_angle = angle + self.rotation_angle
        half_length = length / 2

        # Вычисляем новые координаты концов отрезка
        x1_new = center_x - half_length * math.cos(new_angle)
        y1_new = center_y - half_length * math.sin(new_angle)
        x2_new = center_x + half_length * math.cos(new_angle)
        y2_new = center_y + half_length * math.sin(new_angle)

        # Рисуем отрезок
        self.canvas.create_line(x1_new, y1_new, x2_new, y2_new, width=2, fill='blue')

        # Рисуем точку вращения (центр отрезка)
        self.canvas.create_oval(center_x - 5, center_y - 5,
                                center_x + 5, center_y + 5,
                                fill='red')


if __name__ == "__main__":
    root = tk.Tk()
    app = RotatingLineApp(root)
    root.mainloop()
```

Изображение с результатами работы программы:



Задание 2. Реализовать построение заданного типа фрактала по варианту. 7) Снежинка Коха

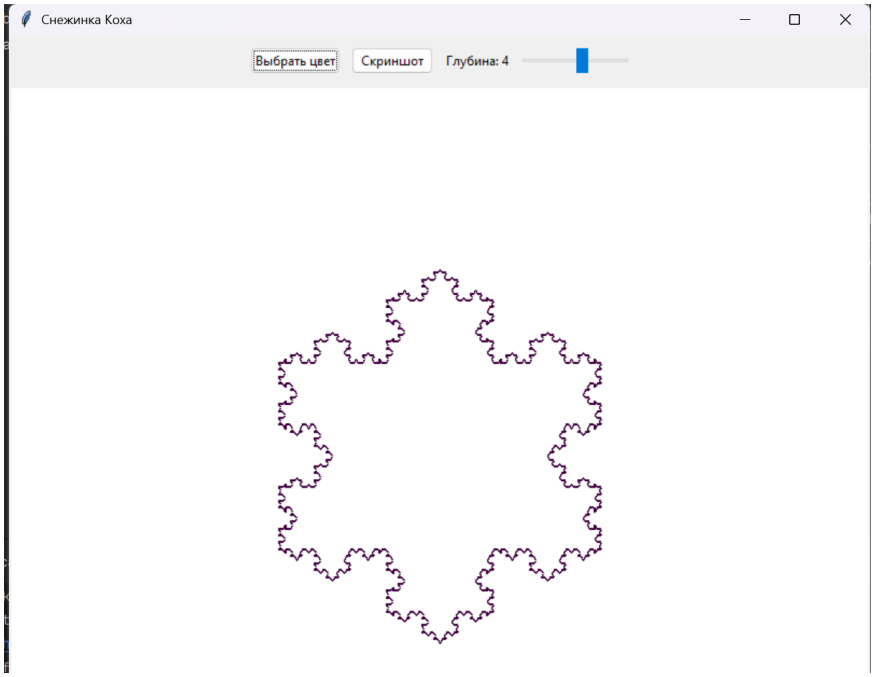
Код программы:

```
import tkinter as tk
from tkinter import ttk, colorchooser
from PIL import Image, ImageGrab
import math
import time

class KochSnowflake:
    def __init__(self, root):
        self.root = root
        self.root.title("Снежинка Коха")
        self.root.geometry("800x600")

        # Параметры снежинки
        self.depth = 0
        self.size = 300
        self.center_x = 400
        self.center_y = 300
        self.color = 'blue'  # Цвет по умолчанию

        # Создание элементов управления
        self.create_controls()

        # Создание холста
        self.canvas = tk.Canvas(root, width=800, height=600, bg='white')
        self.canvas.pack()

        # Отрисовка снежинки
```

```
        self.draw_snowflake()

    def create_controls(self):
        control_frame = ttk.Frame(self.root)
        control_frame.pack(pady=10)

        # Кнопка выбора цвета
        self.color_button = ttk.Button(control_frame, text="Выбрать цвет", command=self.choose_color)
        self.color_button.pack(side=tk.LEFT, padx=5)

        # Кнопка скриншота
        self.screenshot_button = ttk.Button(control_frame, text="Скриншот", command=self.take_screenshot)
        self.screenshot_button.pack(side=tk.LEFT, padx=5)

        # Слайдер глубины рекурсии
        self.depth_label = ttk.Label(control_frame, text="Глубина: 0")
        self.depth_label.pack(side=tk.LEFT, padx=5)

        self.depth_scale = ttk.Scale(control_frame, from_=0, to=7, orient=tk.HORIZONTAL,
command=self.update_depth)
        self.depth_scale.set(0)
        self.depth_scale.pack(side=tk.LEFT, padx=5)

    def choose_color(self):
        color = colorchooser.askcolor(title="Выберите цвет снежинки")
        if color[1]:  # Если цвет выбран (не нажата кнопка отмены)
            self.color = color[1]
            self.draw_snowflake()

    def update_depth(self, value):
        try:
            self.depth = int(float(value))
            self.depth_label.config(text=f"Глубина: {self.depth}")
            self.draw_snowflake()
        except ValueError:
            pass

    def take_screenshot(self):
        # Делаем скриншот всего экрана
        screenshot = ImageGrab.grab()

        # Создаем имя файла с текущим временем
        filename = f"screenshot_{int(time.time())}.png"

        # Сохраняем скриншот
        screenshot.save(filename)
        print(f"Скриншот сохранен как {filename}")

    def draw_snowflake(self):
        self.canvas.delete("all")
```

```
        # Вычисляем координаты вершин равностороннего треугольника
        height = self.size * math.sqrt(3) / 2
        x1 = self.center_x - self.size/2
        y1 = self.center_y + height/2
        x2 = self.center_x + self.size/2
        y2 = self.center_y + height/2
        x3 = self.center_x
        y3 = self.center_y - height/2

        # Рисуем три стороны снежинки
        self.draw_koch_line(x1, y1, x2, y2, self.depth)
        self.draw_koch_line(x2, y2, x3, y3, self.depth)
        self.draw_koch_line(x3, y3, x1, y1, self.depth)

    def draw_koch_line(self, x1, y1, x2, y2, depth):
        if depth == 0:
            self.canvas.create_line(x1, y1, x2, y2, width=2, fill=self.color)
        else:
            # Вычисляем точки деления отрезка
            dx = x2 - x1
            dy = y2 - y1

            # Точки деления на три части
            x1_3 = x1 + dx/3
            y1_3 = y1 + dy/3
            x2_3 = x1 + 2*dx/3
            y2_3 = y1 + 2*dy/3

            # Вычисляем вершину треугольника
            # Поворачиваем вектор на 60 градусов
            angle = math.radians(60)
            vx = (x2_3 - x1_3) * math.cos(angle) - (y2_3 - y1_3) * math.sin(angle)
            vy = (x2_3 - x1_3) * math.sin(angle) + (y2_3 - y1_3) * math.cos(angle)

            # Координаты вершины треугольника
            x_triangle = x1_3 + vx
            y_triangle = y1_3 + vy

            # Рекурсивно рисуем четыре отрезка
            self.draw_koch_line(x1, y1, x1_3, y1_3, depth-1)
            self.draw_koch_line(x1_3, y1_3, x_triangle, y_triangle, depth-1)
            self.draw_koch_line(x_triangle, y_triangle, x2_3, y2_3, depth-1)
            self.draw_koch_line(x2_3, y2_3, x2, y2, depth-1)

if __name__ == "__main__":
    root = tk.Tk()
    app = KochSnowflake(root)
    root.mainloop()
```

Изображение с результатами работы программы:



Вывод: освоил возможности языка программирования Python в разработке оконных приложений