МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

Кафедра интеллектуальных информационных технологий

Отчёт по лабораторной работе №7
Специальность ПО-11

Выполнил:
А. А. Билялова
студент группы ПО11

Проверил:
А. А. Крощенко,
ст. преп. кафедры ИИТ,
26.04.2025

Брест 2025

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

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

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.

Вариант 3. Изобразить четырехугольник, вращающийся в плоскости формы вокруг своего центра тяжести.

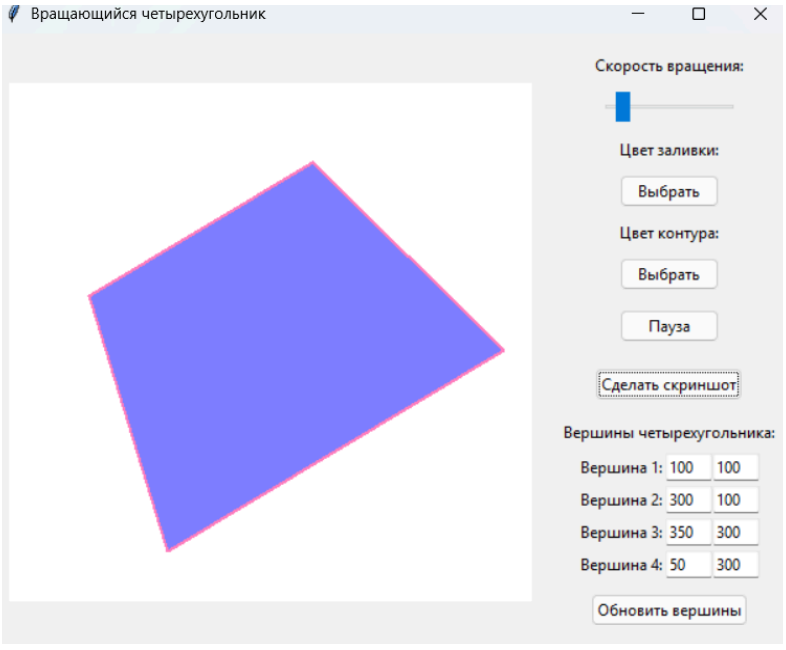
Код программы:

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

class RotatingQuadrilateralApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Вращающийся четырехугольник")

        self.width = 400
        self.height = 400
        self.center_x = self.width // 2
        self.center_y = self.height // 2
        self.vertices = [(100, 100), (300, 100), (350, 300), (50, 300)]
        self.rotation_angle = 0
        self.rotation_speed = 1
        self.fill_color = "blue"
        self.outline_color = "black"
        self.is_rotating = True

        self.canvas = tk.Canvas(root, width=self.width, height=self.height, bg="white")
        self.canvas.pack(side=tk.LEFT, padx=10, pady=10)

        self.control_frame = ttk.Frame(root)
        self.control_frame.pack(side=tk.RIGHT, padx=10, pady=10, fill=tk.Y)

        ttk.Label(self.control_frame, text="Скорость вращения:").pack(pady=5)
        self.speed_slider = ttk.Scale(self.control_frame, from_=0, to=10,
                                      command=self.update_speed)
        self.speed_slider.set(self.rotation_speed)
        self.speed_slider.pack(pady=5)

        ttk.Label(self.control_frame, text="Цвет заливки:").pack(pady=5)
        self.fill_color_btn = ttk.Button(self.control_frame, text="Выбрать",
                                         command=self.choose_fill_color)
        self.fill_color_btn.pack(pady=5)

        ttk.Label(self.control_frame, text="Цвет контура:").pack(pady=5)
```

```
        self.outline_color_btn = ttk.Button(self.control_frame, text="Выбрать",
                                            command=self.choose_outline_color)
        self.outline_color_btn.pack(pady=5)

        self.pause_btn = ttk.Button(self.control_frame, text="Пауза",
                                    command=self.toggle_rotation)
        self.pause_btn.pack(pady=10)

        self.screenshot_btn = ttk.Button(self.control_frame, text="Сделать скриншот",
                                         command=self.take_screenshot)
        self.screenshot_btn.pack(pady=10)

        ttk.Label(self.control_frame, text="Вершины четырехугольника:").pack(pady=5)
        self.vertex_entries = []
        for i in range(4):
            frame = ttk.Frame(self.control_frame)
            frame.pack(pady=2)
            ttk.Label(frame, text=f"Вершина {i+1}:").pack(side=tk.LEFT)
            x_entry = ttk.Entry(frame, width=5)
            x_entry.pack(side=tk.LEFT)
            y_entry = ttk.Entry(frame, width=5)
            y_entry.pack(side=tk.LEFT)
            x_entry.insert(0, str(self.vertices[i][0]))
            y_entry.insert(0, str(self.vertices[i][1]))
            self.vertex_entries.append((x_entry, y_entry))

        self.update_btn = ttk.Button(self.control_frame, text="Обновить вершины",
                                    command=self.update_vertices)
        self.update_btn.pack(pady=10)

        self.quadrilateral = None
        self.animate()

    def rotate_point(self, point, angle):
        x, y = point
        x -= self.center_x
        y -= self.center_y

        new_x = x * math.cos(angle) - y * math.sin(angle)
        new_y = x * math.sin(angle) + y * math.cos(angle)

        return (new_x + self.center_x, new_y + self.center_y)

    def draw_quadrilateral(self):
        if self.quadrilateral:
            self.canvas.delete(self.quadrilateral)

        rotated_vertices = [self.rotate_point(v, self.rotation_angle) for v in self.vertices]
        self.quadrilateral = self.canvas.create_polygon(
            rotated_vertices,
            fill=self.fill_color,
            outline=self.outline_color,
            width=2
        )

    def animate(self):
        if self.is_rotating:
            self.rotation_angle += math.radians(self.rotation_speed)
            self.draw_quadrilateral()

        self.root.after(20, self.animate)

    def update_speed(self, value):
        self.rotation_speed = float(value)

    def choose_fill_color(self):
        color = colorchooser.askcolor(title="Выберите цвет заливки")[1]
        if color:
```

```
            self.fill_color = color

    def choose_outline_color(self):
        color = colorchooser.askcolor(title="Выберите цвет контура")[1]
        if color:
            self.outline_color = color

    def toggle_rotation(self):
        self.is_rotating = not self.is_rotating
        self.pause_btn.config(text="Пауза" if self.is_rotating else "Продолжить")

    def update_vertices(self):
        try:
            new_vertices = []
            for x_entry, y_entry in self.vertex_entries:
                x = int(x_entry.get())
                y = int(y_entry.get())
                new_vertices.append((x, y))

            if len(new_vertices) == 4:
                self.vertices = new_vertices
        except ValueError:
            pass

    def take_screenshot(self):
        x = self.root.winfo_rootx() + self.canvas.winfo_x()
        y = self.root.winfo_rooty() + self.canvas.winfo_y()
        x1 = x + self.canvas.winfo_width()
        y1 = y + self.canvas.winfo_height()

        screenshot = ImageGrab.grab((x, y, x1, y1))

        timestamp = time.strftime("%Y%m%d_%H%M%S")
        filename = f"quadrilateral_{timestamp}.png"

        screenshot.save(filename)

        tk.messagebox.showinfo("Скриншот сохранен", f"Скриншот сохранен как {filename}")

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

Рисунки с результатами работы программы



Задание 2. Реализовать построение заданного типа фрактала по варианту.

Везде, где это необходимо, предусмотреть ввод параметров, влияющих на внешний вид фрактала.

Вариант 3. Треугольная салфетка Серпинского.

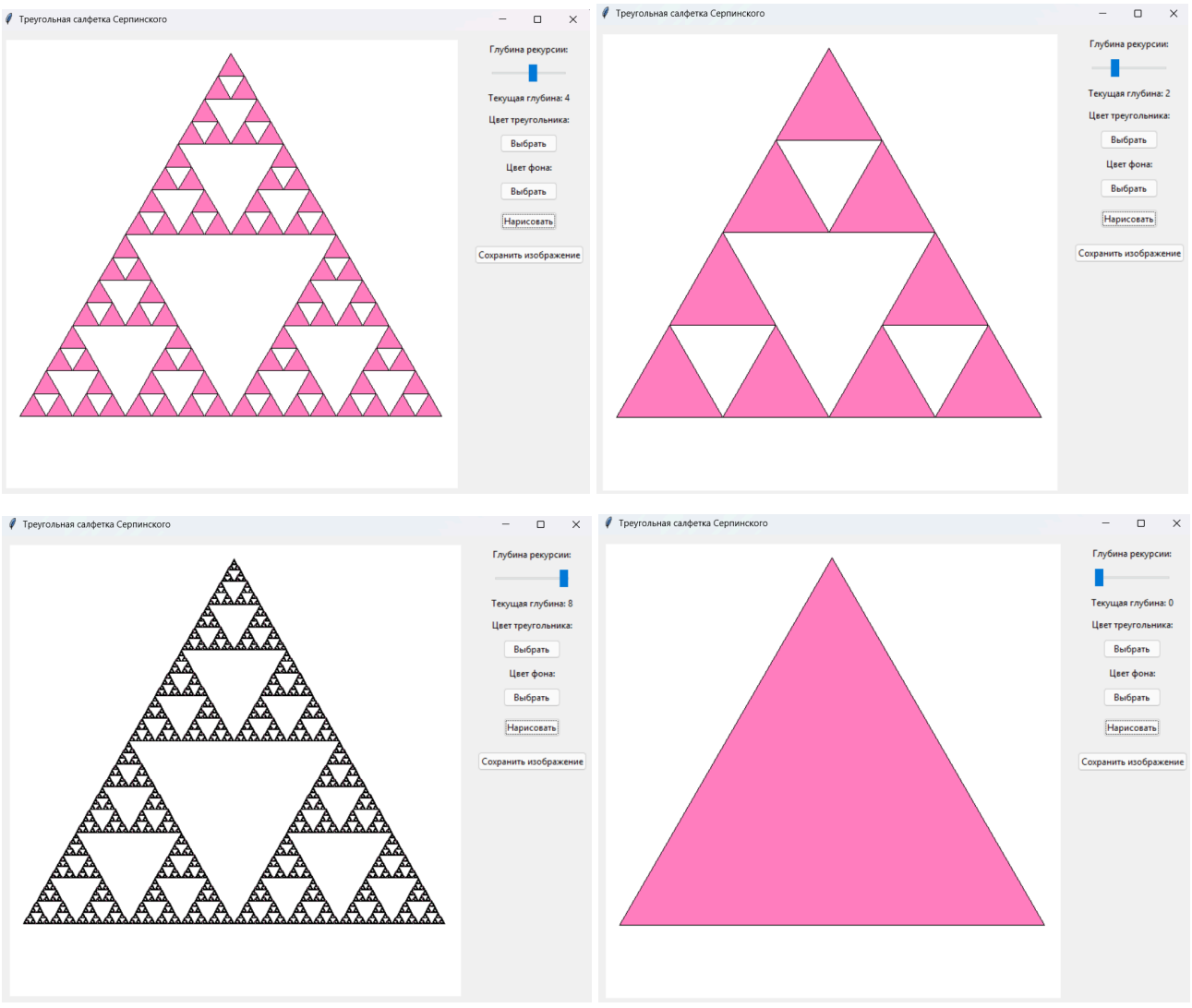
Код программы:

```
import tkinter as tk
from tkinter import ttk, colorchooser, filedialog, messagebox
from PIL import Image, ImageDraw, ImageTk
import math

class SierpinskiTriangleApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Треугольная салфетка Серпинского")

        # Параметры фрактала
        self.depth = 3
        self.width = 600
        self.height = 600
        self.bg_color = "white"
        self.triangle_color = "blue"
        self.padding = 20

        # Создаем холст для отрисовки
        self.canvas = tk.Canvas(root, width=self.width, height=self.height, bg=self.bg_color)
        self.canvas.pack(side=tk.LEFT, padx=10, pady=10)

        # Создаем панель управления
        self.control_frame = ttk.Frame(root)
        self.control_frame.pack(side=tk.RIGHT, padx=10, pady=10, fill=tk.Y)

        # Элементы управления
        ttk.Label(self.control_frame, text="Глубина рекурсии:").pack(pady=5)
        self.depth_slider = ttk.Scale(self.control_frame, from_=0, to=8,
                                      command=self.update_depth)
        self.depth_slider.set(self.depth)
        self.depth_slider.pack(pady=5)

        self.depth_label = ttk.Label(self.control_frame, text=f"Текущая глубина: {self.depth}")
        self.depth_label.pack(pady=5)

        ttk.Label(self.control_frame, text="Цвет треугольника:").pack(pady=5)
        self.color_btn = ttk.Button(self.control_frame, text="Выбрать",
                                    command=self.choose_color)
        self.color_btn.pack(pady=5)

        ttk.Label(self.control_frame, text="Цвет фона:").pack(pady=5)
        self.bg_color_btn = ttk.Button(self.control_frame, text="Выбрать",
                                       command=self.choose_bg_color)
        self.bg_color_btn.pack(pady=5)

        self.draw_btn = ttk.Button(self.control_frame, text="Нарисовать",
                                   command=self.draw_fractal)
        self.draw_btn.pack(pady=10)

        self.save_btn = ttk.Button(self.control_frame, text="Сохранить изображение",
                                   command=self.save_image)
        self.save_btn.pack(pady=10)

        self.draw_fractal()

    def draw_fractal(self):
        self.canvas.delete("all")
        self.canvas.config(bg=self.bg_color)

        # Размеры треугольника с учетом отступов
```

```
    size = min(self.width, self.height) - 2 * self.padding
    height = size * math.sqrt(3) / 2

    # Координаты вершин равностороннего треугольника
    x1, y1 = self.width // 2, self.padding
    x2, y2 = self.padding, self.padding + height
    x3, y3 = self.padding + size, self.padding + height

    # Рисуем фрактал
    self.draw_sierpinski(x1, y1, x2, y2, x3, y3, self.depth)

def draw_sierpinski(self, x1, y1, x2, y2, x3, y3, depth):
    if depth == 0:
        self.canvas.create_polygon(x1, y1, x2, y2, x3, y3,
                                   fill=self.triangle_color,
                                   outline="black")
    else:
        x12 = (x1 + x2) / 2
        y12 = (y1 + y2) / 2
        x13 = (x1 + x3) / 2
        y13 = (y1 + y3) / 2
        x23 = (x2 + x3) / 2
        y23 = (y2 + y3) / 2

        self.draw_sierpinski(x1, y1, x12, y12, x13, y13, depth-1)
        self.draw_sierpinski(x12, y12, x2, y2, x23, y23, depth-1)
        self.draw_sierpinski(x13, y13, x23, y23, x3, y3, depth-1)

def update_depth(self, value):
    self.depth = int(float(value))
    self.depth_label.config(text=f"Текущая глубина: {self.depth}")

def choose_color(self):
    color = colorchooser.askcolor(title="Выберите цвет треугольника")[1]
    if color:
        self.triangle_color = color

def choose_bg_color(self):
    color = colorchooser.askcolor(title="Выберите цвет фона")[1]
    if color:
        self.bg_color = color
        self.canvas.config(bg=self.bg_color)
def save_image(self):
    image = Image.new("RGB", (self.width, self.height), self.bg_color)
    draw = ImageDraw.Draw(image)

    # Рисуем фрактал на изображении
    self.draw_sierpinski_on_image(draw, self.depth)

    filetypes = [("PNG файлы", "*.png"), ("JPEG файлы", "*.jpg"), ("Все файлы", "*.*")]
    filename = filedialog.asksaveasfilename(defaultextension=".png",
                                            filetypes=filetypes)
    if filename:
        try:
            image.save(filename)
            messagebox.showinfo("Успешно", "Изображение сохранено")
        except Exception as e:
            messagebox.showerror("Ошибка", f"Не удалось сохранить файл: {e}")

def draw_sierpinski_on_image(self, draw, depth):
    size = min(self.width, self.height) - 2 * self.padding
    height = size * math.sqrt(3) / 2

    x1, y1 = self.width // 2, self.padding
    x2, y2 = self.padding, self.padding + height
    x3, y3 = self.padding + size, self.padding + height

    self._draw_sierpinski_recursive(draw, x1, y1, x2, y2, x3, y3, depth)
```

```
    def _draw_sierpinski_recursive(self, draw, x1, y1, x2, y2, x3, y3, depth):
        if depth == 0:
            draw.polygon([(x1, y1), (x2, y2), (x3, y3)],
                         fill=self.triangle_color,
                         outline="black")
        else:
            x12 = (x1 + x2) / 2
            y12 = (y1 + y2) / 2
            x13 = (x1 + x3) / 2
            y13 = (y1 + y3) / 2
            x23 = (x2 + x3) / 2
            y23 = (y2 + y3) / 2

            self._draw_sierpinski_recursive(draw, x1, y1, x12, y12, x13, y13, depth-1)
            self._draw_sierpinski_recursive(draw, x12, y12, x2, y2, x23, y23, depth-1)
            self._draw_sierpinski_recursive(draw, x13, y13, x23, y23, x3, y3, depth-1)

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

Рисунки с результатами работы программы



Вывод: освоила возможности языка программирования Python в разработке оконных приложений.