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
Д. М. Андросюк
студент группы ПО11

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

Брест 2025

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

Ход Работы
Вариант 1
Задание 1

Построение графических примитивов и надписей. Создать класс Triangle и класс Point. Объявить список из n объектов класса Point, написать функцию, определяющую, какая из точек лежит внутри, а какая – снаружи треугольника.

Код программы:

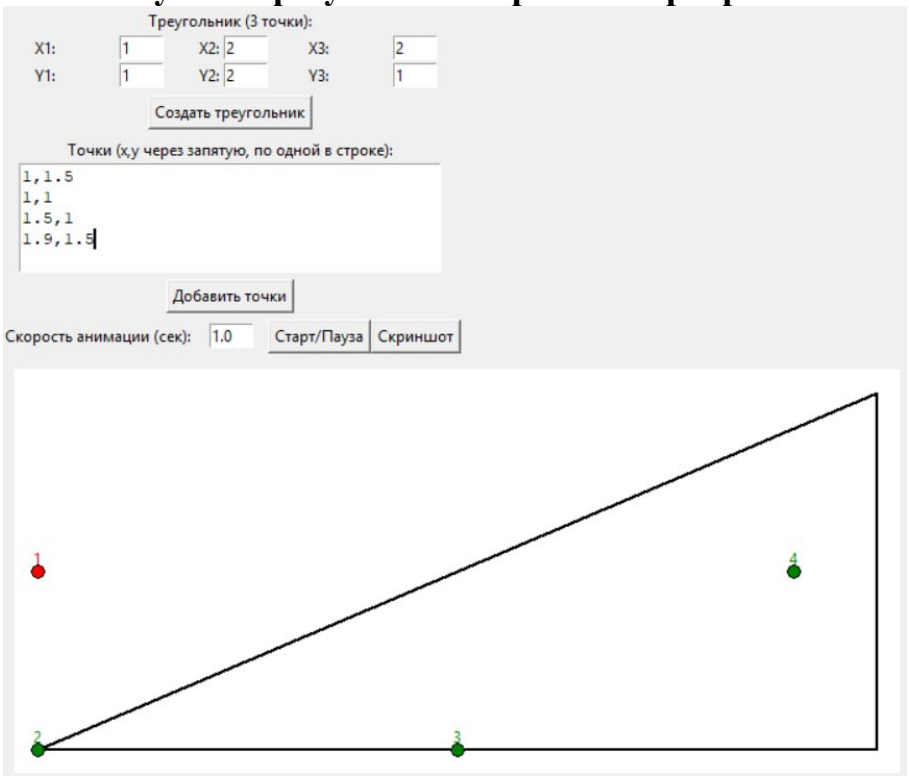
```
import tkinter as tk
from tkinter import messagebox
from PIL import ImageGrab
import time

class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

class Triangle:
    def __init__(self, p1: Point, p2: Point, p3: Point):
        self.p1 = p1
        self.p2 = p2
        self.p3 = p3

    def area(self, a: Point, b: Point, c: Point):
        return abs((a.x*(b.y - c.y) + b.x*(c.y - a.y) + c.x*(a.y - b.y)) / 2.0)

    def is_inside(self, p: Point):
        # Площадь треугольника
        A = self.area(self.p1, self.p2, self.p3)
        A1 = self.area(p, self.p2, self.p3)
        A2 = self.area(self.p1, p, self.p3)
        A3 = self.area(self.p1, self.p2, p)
        return abs(A - (A1 + A2 + A3)) < 1e-6

class Application(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Точки и треугольник")
        self.geometry("700x600")
        self.points = []
        self.triangle = None
        self.speed = 1.0  # скорость анимации
        self.is_running = False

        self.create_widgets()
        self.animation_index = 0

    def create_widgets(self):
        frame = tk.Frame(self)
        frame.pack(side=tk.TOP, fill=tk.X)

        tk.Label(frame, text="Треугольник (3 точки):").grid(row=0, column=0, columnspan=6)

        self.triangle_entries = []
        for i in range(3):
            tk.Label(frame, text=f"X{i+1}:").grid(row=1, column=2*i)
            x_entry = tk.Entry(frame, width=5)
            x_entry.grid(row=1, column=2*i+1)
```

```
            tk.Label(frame, text=f"Y{i+1}:").grid(row=2, column=2*i)
            y_entry = tk.Entry(frame, width=5)
            y_entry.grid(row=2, column=2*i+1)
            self.triangle_entries.append((x_entry, y_entry))

        tk.Button(frame, text="Создать треугольник", command=self.create_triangle).grid(row=3, column=0, columnspan=6,
pady=5)

        tk.Label(frame, text="Точки (x,y через запятую, по одной в строке):").grid(row=4, column=0, columnspan=6)
        self.points_text = tk.Text(frame, height=5, width=40)
        self.points_text.grid(row=5, column=0, columnspan=6)

        tk.Button(frame, text="Добавить точки", command=self.add_points).grid(row=6, column=0, columnspan=6, pady=5)

        tk.Label(frame, text="Скорость анимации (сек):").grid(row=7, column=0, columnspan=2)
        self.speed_entry = tk.Entry(frame, width=5)
        self.speed_entry.insert(0, "1.0")
        self.speed_entry.grid(row=7, column=2, columnspan=2)

        tk.Button(frame, text="Старт/Пауза", command=self.toggle_animation).grid(row=7, column=4)
        tk.Button(frame, text="Скриншот", command=self.take_screenshot).grid(row=7, column=5)

        self.canvas = tk.Canvas(self, bg="white", width=680, height=400)
        self.canvas.pack(pady=10)

        self.status_label = tk.Label(self, text="Статус: Ожидание треугольника и точек")
        self.status_label.pack()

    def create_triangle(self):
        try:
            coords = []
            for x_entry, y_entry in self.triangle_entries:
                x = float(x_entry.get())
                y = float(y_entry.get())
                coords.append(Point(x, y))
            self.triangle = Triangle(*coords)
            self.status_label.config(text="Треугольник создан")
            self.draw()
        except ValueError:
            messagebox.showerror("Ошибка", "Введите корректные числа для треугольника")

    def add_points(self):
        self.points.clear()
        lines = self.points_text.get("1.0", tk.END).strip().split("\n")
        for line in lines:
            if not line.strip():
                continue
            try:
                x_str, y_str = line.split(",")
                x = float(x_str.strip())
                y = float(y_str.strip())
                self.points.append(Point(x, y))
            except Exception:
                messagebox.showerror("Ошибка", f"Неверный формат точки: {line}")
                return
        self.status_label.config(text=f"Добавлено {len(self.points)} точек")
        self.draw()

    def draw(self):
        self.canvas.delete("all")
        if not self.triangle:
            return


        margin = 20
        all_x = [self.triangle.p1.x, self.triangle.p2.x, self.triangle.p3.x] + [p.x for p in self.points]
```

```
    all_y = [self.triangle.p1.y, self.triangle.p2.y, self.triangle.p3.y] + [p.y for p in self.points]

    if not all_x or not all_y:
        return

    min_x, max_x = min(all_x), max(all_x)
    min_y, max_y = min(all_y), max(all_y)

    width = self.canvas.winfo_width() - 2 * margin
    height = self.canvas.winfo_height() - 2 * margin

    def scale_x(x):
        return margin + (x - min_x) / (max_x - min_x) * width if max_x != min_x else margin + width/2

    def scale_y(y):
        return margin + height - (y - min_y) / (max_y - min_y) * height if max_y != min_y else margin + height/2

    #  треугольник
    self.canvas.create_polygon(
        scale_x(self.triangle.p1.x), scale_y(self.triangle.p1.y),
        scale_x(self.triangle.p2.x), scale_y(self.triangle.p2.y),
        scale_x(self.triangle.p3.x), scale_y(self.triangle.p3.y),
        outline="black", fill="", width=2
    )

    #  точки
    for i, p in enumerate(self.points):
        color = "green" if self.triangle.is_inside(p) else "red"
        r = 5
        cx = scale_x(p.x)
        cy = scale_y(p.y)
        self.canvas.create_oval(cx-r, cy-r, cx+r, cy+r, fill=color)
        self.canvas.create_text(cx, cy-10, text=str(i+1), fill=color)

def toggle_animation(self):
    if not self.triangle or not self.points:
        messagebox.showwarning("Внимание", "Сначала создайте треугольник и добавьте точки")
        return

    if self.is_running:
        self.is_running = False
        self.status_label.config(text="Анимация приостановлена")
    else:
        try:
            self.speed = float(self.speed_entry.get())
            if self.speed <= 0:
                raise ValueError
        except ValueError:
            messagebox.showerror("Ошибка", "Скорость должна быть положительным числом")
            return
        self.is_running = True
        self.status_label.config(text="Анимация запущена")
        self.animate_points()

def animate_points(self):
    if not self.is_running:
        return



    self.canvas.delete("highlight")
    if self.animation_index >= len(self.points):
        self.animation_index = 0

    p = self.points[self.animation_index]
    margin = 20
    all_x = [self.triangle.p1.x, self.triangle.p2.x, self.triangle.p3.x] + [pt.x for pt in self.points]
```

```
        all_y = [self.triangle.p1.y, self.triangle.p2.y, self.triangle.p3.y] + [pt.y for pt in self.points]
        min_x, max_x = min(all_x), max(all_x)
        min_y, max_y = min(all_y), max(all_y)

        width = self.canvas.winfo_width() - 2 * margin
        height = self.canvas.winfo_height() - 2 * margin

        def scale_x(x):
            return margin + (x - min_x) / (max_x - min_x) * width if max_x != min_x else margin + width/2

        def scale_y(y):
            return margin + height - (y - min_y) / (max_y - min_y) * height if max_y != min_y else margin + height/2

        cx = scale_x(p.x)
        cy = scale_y(p.y)
        self.canvas.create_oval(cx-10, cy-10, cx+10, cy+10, outline="blue", width=3, tags="highlight")

        self.animation_index += 1
        self.after(int(self.speed * 1000), self.animate_points)

    def take_screenshot(self):
        x = self.winfo_rootx()
        y = self.winfo_rooty()
        w = self.winfo_width()
        h = self.winfo_height()
        filename = f"screenshot_{int(time.time())}.png"
        ImageGrab.grab(bbox=(x, y, x + w, y + h)).save(filename)
        messagebox.showinfo("Скриншот", f"Скриншот сохранён как {filename}")

if __name__ == "__main__":
    app = Application()
    app.mainloop()
```

Рисунки с результатами работы программы:



Задание 2

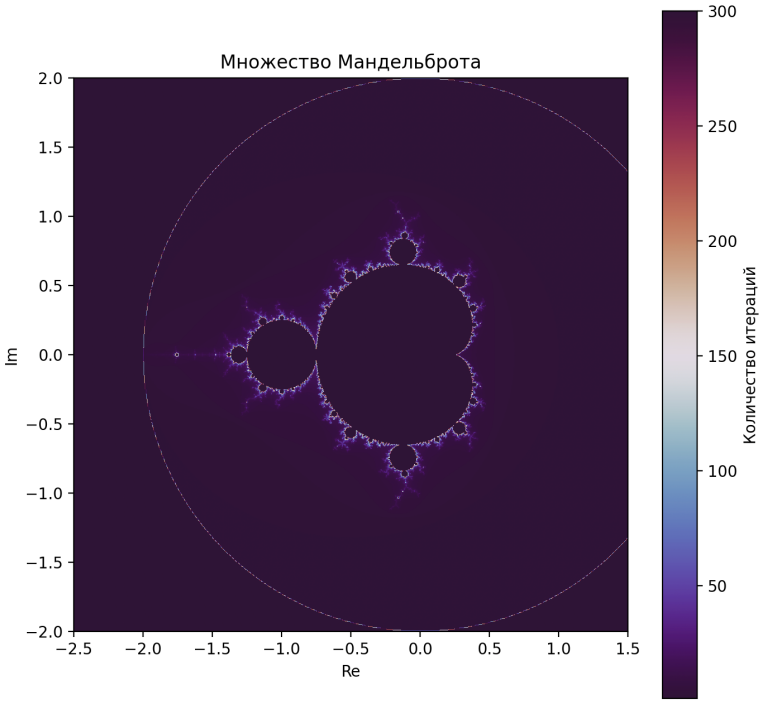
Реализовать построение заданного типа фрактала по варианту. Множество Мальдеброта.

Код программы:

```
import numpy as np
```

```
import matplotlib.pyplot as plt

def mandelbrot(xmin, xmax, ymin, ymax, width, height, max_iter):
    real = np.linspace(xmin, xmax, width)
    imag = np.linspace(ymin, ymax, height)
    c = real[:, np.newaxis] + 1j * imag[np.newaxis, :]
    z = np.zeros_like(c)
    div_time = np.zeros(c.shape, dtype=int)

    mask = np.full(c.shape, True, dtype=bool)

    for i in range(max_iter):
        z[mask] = z[mask]**2 + c[mask]
        mask_new = np.abs(z) <= 2
        div_now = mask & (~mask_new)
        div_time[div_now] = i
        mask = mask_new

    div_time[div_time == 0] = max_iter
    return div_time.T

if __name__ == "__main__":
    xmin, xmax = -2.5, 1.5
    ymin, ymax = -2.0, 2.0
    width, height = 800, 800
    max_iter = 300

    image = mandelbrot(xmin, xmax, ymin, ymax, width, height, max_iter)

    plt.figure(figsize=(8, 8))
    plt.imshow(image, cmap='twilight_shifted', extent=[xmin, xmax, ymin, ymax])
    plt.colorbar(label='Количество итераций')
    plt.title("Множество Мандельброта")
    plt.xlabel("Re")
    plt.ylabel("Im")
    plt.show()
```

Рисунки с результатами работы программы:



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