МИНОБРНАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Восточно–Сибирский государственный университет   
технологий и управления»

(ФГБОУ ВО ВСГУТУ)

Отчет

по дисциплине «ООО Ромашка»

Тема: «г. Москва, ул. Долгоруковская, д. 0»

|  |  |  |
| --- | --- | --- |
|  | Выполнил: | студент гр. Б761-2 |
|  |  | Немеров А.П. |
|  | Проверил: | Зав. кафедры, к. т. н., доц. |
|  |  | Евдокимова И.С. |

Улан–Удэ

2024

**Цель работы:**

Разработка функции классификации треугольников, а также написание и выполнение автотестов для проверки корректности её работы.

**Задачи:**

* Разработать функцию для классификации треугольников, принимающую три стороны.
* Обработать ошибки ввода, такие как пустые строки, некорректные или отрицательные значения.
* Написать автотесты для проверки работы функции с различными типами треугольников и с некорректными данными.
* Визуализировать треугольник на основе введённых значений.

**Описание программы:**

Программа состоит из следующих основных компонентов:

*Функция classify\_triangle:*

Принимает на вход три стороны треугольника в виде строк, преобразует их в числа и проверяет корректность данных.

Осуществляет проверку существования треугольника по теореме треугольника (сумма двух сторон должна быть больше третьей).

*Классифицирует треугольник как один из следующих типов:*

* Равносторонний.
* Равнобедренный и остроугольный/тупоугольный/прямоугольный.
* Разносторонний остроугольный/тупоугольный/прямоугольный.

Обрабатывает ошибки ввода, выбрасывая соответствующие исключения при неправильных данных (некорректные или пустые значения, отрицательные числа).

*Графический интерфейс (GUI) на базе библиотеки tkinter:*

Пользователь вводит три стороны треугольника через текстовые поля.

Кнопка "Проверить" выводит результат классификации треугольника.

Кнопка "Построить" визуализирует треугольник на холсте, автоматически подстраивая его размер, если он выходит за границы или слишком мал.

*Тесты:*

Написаны с использованием библиотеки unittest.

Проверяют как корректные случаи (разные виды треугольников), так и некорректные (пустые строки, нечисловые значения, отрицательные стороны).

**Автоматические тесты:**

*В файле test.py содержатся следующие тесты:*

*Тесты на корректные данные:*

* test\_equilateral\_triangle: Проверяет равносторонний треугольник.
* test\_isosceles\_triangle: Проверяет равнобедренный остроугольный треугольник.
* test\_scalene\_triangle: Проверяет разносторонний прямоугольный треугольник.
* test\_obtuse\_triangle: Проверяет тупоугольный треугольник.
* test\_acute\_triangle: Проверяет остроугольный треугольник.

*Тесты на некорректные данные:*

* test\_invalid\_triangle: Проверяет случай, когда треугольник не существует.
* test\_invalid\_negative\_sides: Проверяет, выбрасывается ли исключение при отрицательных сторонах.
* test\_invalid\_zero\_sides: Проверяет, выбрасывается ли исключение при нулевых сторонах.
* test\_empty\_input: Проверяет, выбрасывается ли исключение при отсутствии значения.
* test\_non\_numeric\_input: Проверяет, выбрасывается ли исключение при неправильном вводе (нечисловые значения).

Эти тесты помогают убедиться, что функция корректно обрабатывает все типы данных, а также надёжно работает с ошибками ввода.

**Результаты тестирования:**

Запуск тестов с использованием библиотеки unittest дал следующие результаты:

|  |  |  |
| --- | --- | --- |
| Тест | Результат | Ожидаемый результат |
| Равносторонний треугольник  ("5", "5", "5") | Успешно | "Равносторонний" |
| Равнобедренный треугольник  ("5", "5", "3") | Успешно | "Остроугольный и равнобедренный" |
| Прямоугольный треугольник  ("3", "4", "5") | Успешно | "Прямоугольный" |
| Тупоугольный треугольник  ("7", "5", "3") | Успешно | "Тупоугольный" |
| Остроугольный треугольник  ("4", "5", "6") | Успешно | "Остроугольный" |
| Треугольник не существует  ("1", "2", "3") | Успешно | "Треугольник не существует" |
| Отрицательные стороны  ("-3", "4", "5") | Успешно | Исключение ValueError |
| Нулевые  ("0", "4", "5") | Успешно | Исключение ValueError |
| Пустой ввод  ("", "4", "5") | Успешно | Исключение ValueError |
| Нечисловой ввод  ("abc", "4", "5") | Успешно | Исключение ValueError |

Тестирование показало, что функция корректно работает и обрабатывает ошибки на всех предусмотренных этапах.

**Вывод:**

В ходе работы была успешно разработана и протестирована функция классификации треугольников. Программа корректно обрабатывает все виды треугольников, а также различные некорректные случаи ввода. Визуализация треугольников также работает с автоматическим масштабированием. Тесты подтвердили корректную работу программы и её устойчивость к некорректным данным.

Практическая работа продемонстрировала важность тестирования на всех этапах разработки, что позволяет вовремя обнаруживать ошибки и повышать надёжность программного обеспечения.

Приложение

Файл main.py — основной код программы.

import tkinter as tk  
from tkinter import messagebox  
from math import sqrt, isclose  
  
  
def classify\_triangle(a\_str, b\_str, c\_str):  
 # Проверка на пустые поля  
 if not a\_str or not b\_str or not c\_str:  
 raise ValueError("Все три стороны должны быть введены.")  
  
 try:  
 # Преобразование строк в числа  
 a = float(a\_str)  
 b = float(b\_str)  
 c = float(c\_str)  
 except ValueError as ve:  
 raise ValueError("Введите корректное число")  
  
 # Проверка на отрицательные или нулевые значения  
 if a <= 0 or b <= 0 or c <= 0:  
 raise ValueError("Стороны треугольника должны быть положительными числами.")  
  
 # Сортируем стороны по возрастанию  
 sides = sorted([a, b, c])  
  
 # Проверка существования треугольника  
 if sides[0] + sides[1] <= sides[2]:  
 return "Треугольник не существует"  
  
 # Проверка на равностороний треугольник  
 if isclose(sides[0], sides[1]) and isclose(sides[1], sides[2]):  
 return "Равносторонний"  
  
 # Проверка на равнобедренный треугольник  
 elif isclose(sides[0], sides[1]) or isclose(sides[1], sides[2]) or isclose(sides[0], sides[2]):  
 is\_isosceles = True  
 else:  
 is\_isosceles = False  
  
 # Проверка на прямоугольный, тупоугольный и остроугольный треугольник (используем теорему Пифагора)  
 a2, b2, c2 = sides[0] \*\* 2, sides[1] \*\* 2, sides[2] \*\* 2  
 if isclose(c2, a2 + b2):  
 triangle\_type = "Прямоугольный"  
 elif c2 > a2 + b2:  
 triangle\_type = "Тупоугольный"  
 else:  
 triangle\_type = "Остроугольный"  
  
 # Если равнобедреный треугольник, добавляем это в описание  
 if is\_isosceles:  
 triangle\_type += " и равнобедренный"  
  
 return triangle\_type  
  
  
def on\_check():  
 try:  
 # Считывание значений с полей ввода  
 a\_str = entry\_a.get().strip()  
 b\_str = entry\_b.get().strip()  
 c\_str = entry\_c.get().strip()  
  
 # Проверка на существование треугольника и классификация  
 triangle\_type = classify\_triangle(a\_str, b\_str, c\_str)  
  
 # Вывод результата в UI  
 result\_label.config(text=f"Треугольник: {triangle\_type}")  
  
 except ValueError as ve:  
 # Вывод ошибки пользователю  
 messagebox.showerror("Ошибка", f"Ошибка ввода: {ve}")  
 except Exception as e:  
 # Обработка любых других ошибок  
 messagebox.showerror("Ошибка", f"Произошла ошибка: {e}")  
  
  
def on\_draw():  
 try:  
 # Считывание значений с полей ввода  
 a\_str = entry\_a.get().strip()  
 b\_str = entry\_b.get().strip()  
 c\_str = entry\_c.get().strip()  
  
 # Классификация треугольника  
 triangle\_type = classify\_triangle(a\_str, b\_str, c\_str)  
 result\_label.config(text=f"Треугольник: {triangle\_type}")  
  
 # Преобразование строк в числа  
 a = float(a\_str)  
 b = float(b\_str)  
 c = float(c\_str)  
  
 # Расчет координат для треугольника на холсте  
 # Первая точка - нижняя левая (фиксированная)  
 p1 = (10, 10)  
  
 # Вторая точка находится ровно ниже первой  
 p2 = (p1[0], p1[1] + b)  
  
 # Высота треугольника (с использованием формулы Герона для площади)  
 s = (a + b + c) / 2  
 area = (s \* (s - a) \* (s - b) \* (s - c)) \*\* 0.5  
 height = (2 \* area) / b  
  
 # Определение положения третьей точки, расположенной справа  
 p3\_x = p1[0] + a  
 p3\_y = p1[1] + height  
 p3 = (p3\_x, p3\_y)  
  
 # Очистка холста перед рисованием  
 canvas.delete("all")  
  
 # Проверка размеров треугольника и масштабирование, если он превышает 500x500  
 max\_dim = max(10, abs(p1[0] - p2[0]), abs(p1[1] - p2[1]), abs(p1[0] - p3[0]), abs(p1[1] - p3[1]))  
 # Установим границы для минимального и максимального размера треугольника  
 min\_size = 100 # минимальный размер сторон  
 max\_size = 500 # максимальный размер поля вывода  
  
 scale = 1  
 if max\_dim > max\_size: # если треугольник слишком большой  
 scale = max\_size / max\_dim  
 elif max\_dim < min\_size: # если треугольник слишком маленький  
 scale = min\_size / max\_dim  
  
 # Применение масштаба  
 p1 = (p1[0], p1[1])  
 p2 = (p2[0] \* scale, p2[1] \* scale)  
 p3 = (p3[0] \* scale, p3[1] \* scale)  
  
 # Рисование треугольника  
 canvas.create\_line(p1, p2, fill="red")  
 canvas.create\_line(p2, p3, fill="green")  
 canvas.create\_line(p3, p1, fill="blue")  
  
 except ValueError as ve:  
 # Вывод ошибки пользователю  
 messagebox.showerror("Ошибка", f"Ошибка ввода: {ve}")  
 except Exception as e:  
 # Обработка любых других ошибок  
 messagebox.showerror("Ошибка", f"Произошла ошибка: {e}")  
 pass  
  
  
root = tk.Tk()  
root.title("Классификация треугольника")  
  
frame = tk.Frame(root)  
frame.pack(pady=10)  
  
tk.Label(frame, text="Сторона A:").grid(row=0, column=0, padx=5, pady=5)  
entry\_a = tk.Entry(frame)  
entry\_a.grid(row=0, column=1, padx=5, pady=5)  
  
tk.Label(frame, text="Сторона B:").grid(row=1, column=0, padx=5, pady=5)  
entry\_b = tk.Entry(frame)  
entry\_b.grid(row=1, column=1, padx=5, pady=5)  
  
tk.Label(frame, text="Сторона C:").grid(row=2, column=0, padx=5, pady=5)  
entry\_c = tk.Entry(frame)  
entry\_c.grid(row=2, column=1, padx=5, pady=5)  
  
check\_button = tk.Button(root, text="Проверить", command=on\_check)  
check\_button.pack(pady=10)  
  
draw\_button = tk.Button(root, text="Построить", command=on\_draw)  
draw\_button.pack(pady=10)  
  
canvas = tk.Canvas(root, width=500, height=500, bg="white")  
canvas.pack()  
  
result\_label = tk.Label(root, text="Тип треугольника:")  
result\_label.pack(pady=10)  
  
root.mainloop()

Файл test.py — файл с автотестами.

import unittest  
from main import classify\_triangle  
  
'''  
test\_equilateral\_triangle: Проверяет равносторонний треугольник.  
test\_isosceles\_triangle: Проверяет равнобедренный остроугольный треугольник.  
test\_scalene\_triangle: Проверяет разносторонний прямоугольный треугольник.  
test\_invalid\_triangle: Проверяет случай, когда треугольник не существует.  
test\_obtuse\_triangle: Проверяет тупоугольный треугольник.  
test\_acute\_triangle: Проверяет остроугольный треугольник.  
test\_invalid\_negative\_sides: Проверяет, выбрасывается ли исключение при отрицательных сторонах.  
test\_invalid\_zero\_sides: Проверяет, выбрасывается ли исключение при нулевых сторонах.  
test\_empty\_input: Проверяет, выбрасывается ли исключение при отсутствии значения.  
test\_non\_numeric\_input: Проверяет, выбрасывается ли исключение при неправильном вводе   
'''  
  
class TestTriangleClassification(unittest.TestCase):  
 def test\_equilateral\_triangle(self):  
 self.assertEqual(classify\_triangle("5", "5", "5"), "Равносторонний")  
  
 def test\_isosceles\_triangle(self):  
 self.assertEqual(classify\_triangle("5", "5", "3"), "Остроугольный и равнобедренный")  
  
 def test\_scalene\_triangle(self):  
 self.assertEqual(classify\_triangle("3", "4", "5"), "Прямоугольный")  
  
 def test\_invalid\_triangle(self):  
 self.assertEqual(classify\_triangle("1", "2", "3"), "Треугольник не существует")  
  
 def test\_obtuse\_triangle(self):  
 self.assertEqual(classify\_triangle("7", "5", "3"), "Тупоугольный")  
  
 def test\_acute\_triangle(self):  
 self.assertEqual(classify\_triangle("4", "5", "6"), "Остроугольный")  
  
 def test\_invalid\_negative\_sides(self):  
 with self.assertRaises(ValueError):  
 classify\_triangle("-3", "4", "5") # отрицательное значение  
  
 def test\_invalid\_zero\_sides(self):  
 with self.assertRaises(ValueError):  
 classify\_triangle("0", "4", "5") # нулевое значение  
  
 def test\_empty\_input(self):  
 with self.assertRaises(ValueError):  
 classify\_triangle("", "4", "5") # пустое значение  
  
 def test\_non\_numeric\_input(self):  
 with self.assertRaises(ValueError):  
 classify\_triangle("abc", "4", "5") # неправильный ввод  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

