Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО–КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №7
дисциплины
«Объектно–ориентированное программирование»
Вариант 13

Выполнил:
Рябинин Егор Алексеевич
3 курс, группа ИВТ–б–о–23–2,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Проверил:
Доцент департамента цифровых, робототехнических систем и электроники института перспективной инженерии
Воронкин Роман Александрович
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г

Тема: Управление потоками в Python.

Цель: Приобретение навыков многопоточных приложений на языке программирования Python версии 3.13.3.

Порядок выполнения работы:

Ссылка на репозиторий:

https://github.com/bohemiaaaaa/Lab7_Object-oriented-programming

Задание №1. С использованием многопоточности для заданного значения x найти сумму ряда S с точностью члена ряда по абсолютному значению $\varepsilon = 10^{-7}$ и произвести сравнение полученной суммы с контрольным значениям функции y для двух бесконечных рядов.

$$S = \sum_{n=1}^{\infty} \frac{1}{(2n-1)x^{2n-1}} = \frac{1}{x} + \frac{1}{3x^3} + \frac{1}{5x^5} + \dots;$$
$$x = 3;\ y = \frac{1}{2}\ln\frac{x+1}{x-1}.$$

Листинг программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


import math
from threading import Thread


class Series:
    def __init__(self, x: float = 3.0, eps: float = 1e-7) -> None:
        self.x = x
        self.eps = eps

    def _compute_element(self, n: int) -> float:
        try:
            return 1.0 / ((2 * n - 1) * (self.x ** (2 * n - 1)))
        except OverflowError:
            return 0.0

    def _thread_task(
        self,
        start_index: int,
        step: int,
        sums: list,
        counts: list,
        pos: int,
    ) -> None:
        partial_sum = 0.0
        elements_count = 0

        n = start_index
        term = self._compute_element(n)
```

```
        while abs(term) >= self.eps:
            partial_sum += term
            elements_count += 1

            n += step
            term = self._compute_element(n)

        sums[pos] = partial_sum
        counts[pos] = elements_count

    def evaluate(self, threads_num: int = 4) -> tuple[float, int]:
        workers = []
        partial_sums = [0.0] * threads_num
        elements_counts = [0] * threads_num

        for i in range(threads_num):
            t = Thread(
                target=self._thread_task,
                args=(i + 1, threads_num, partial_sums, elements_counts, i),
            )
            workers.append(t)
            t.start()

        for t in workers:
            t.join()

        return sum(partial_sums), sum(elements_counts)

    def analytical(self) -> float:
        return 0.5 * math.log((self.x + 1) / (self.x - 1))

    def __str__(self) -> str:
        return (
            "Ряд:\n"
            "S = Σ [ 1 / ((2n - 1) * x^(2n - 1)) ],  n = 1 .. ∞\n\n"
            f"x = {self.x}\n"
            f"epsilon = {self.eps}\n"
        )
```

```
PS C:\Users\4isto\OOP_lab7> python tasks/task1.py
Ряд:
S = Σ [ 1 / ((2n - 1) * x^(2n - 1)) ],  n = 1 .. ∞

x = 3.0
epsilon = 1e-07

Сумма ряда S = 0.3465735369
Использовано членов ряда: 6
Контрольное значение y = 0.3465735903
|S - y| = 5.34e-08
Точность достигнута
```

Рисунок 1 – Результат работы программы

Тесты для написанной программы:

Листинг программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

```python
import math

import pytest
from task1 import Series


def test_single_term_value():
    s = Series(x=3.0)
    term = s._compute_element(1)

    expected = 1.0 / ((2 * 1 - 1) * (3.0 ** (2 * 1 - 1)))
    assert math.isclose(term, expected, rel_tol=1e-12)


def test_analytical_value():
    s = Series(x=3.0)
    exact = s.analytical()

    expected = 0.5 * math.log((3.0 + 1) / (3.0 - 1))
    assert math.isclose(exact, expected, rel_tol=1e-12)


@pytest.mark.parametrize("threads", [1, 2, 4, 8])
def test_series_convergence(threads):
    eps = 1e-7
    s = Series(x=3.0, eps=eps)

    value, terms = s.evaluate(threads_num=threads)
    exact = s.analytical()

    assert abs(value - exact) < eps
    assert terms > 0


def test_result_stability():
    s = Series(x=3.0, eps=1e-7)

    value_2, _ = s.evaluate(threads_num=2)
    value_4, _ = s.evaluate(threads_num=4)

    assert abs(value_2 - value_4) < 1e-9


def test_terms_count_reasonable():
    s = Series(x=3.0, eps=1e-7)
    _, terms = s.evaluate(threads_num=4)

    assert 1 < terms < 10_000
```

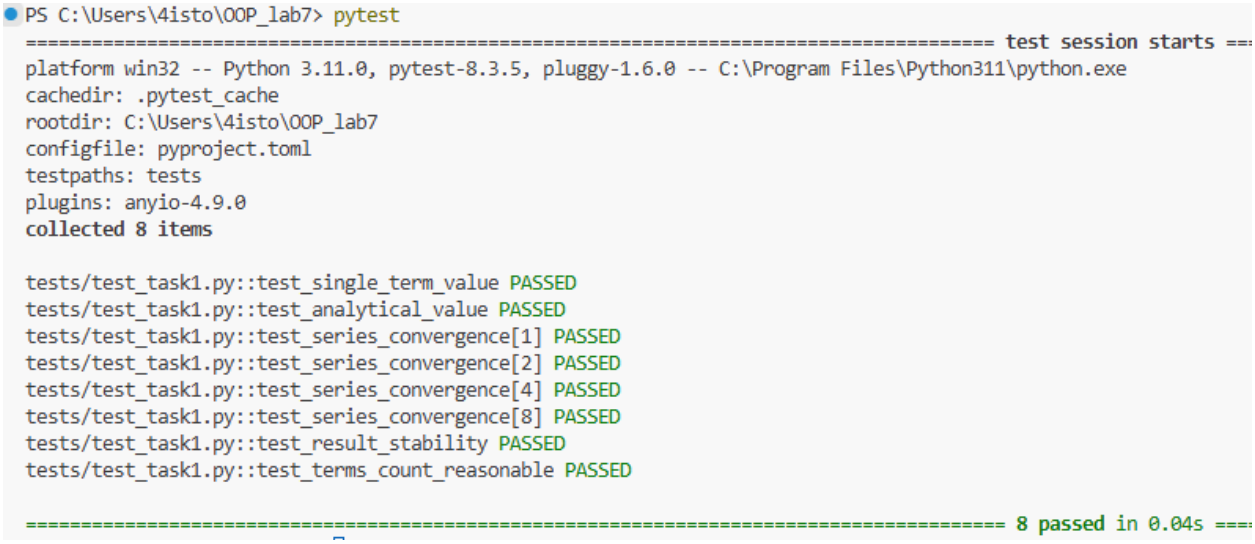

Рисунок 2 – Результат работы тестов

Контрольные вопросы:

1. Что такое синхронность и асинхронность?

Синхронность – это выполнение операций последовательно, одна за другой, где каждая следующая операция ожидает завершения предыдущей. Асинхронность – это выполнение операций независимо от основного потока, позволяющее не блокировать его, часто с использованием механизмов обратных вызовов, промисов или событий.

2. Что такое параллелизм и конкурентность?

Конкурентность – это способность системы выполнять несколько задач "одновременно" за счёт переключения контекста между ними на одном ядре процессора. Параллелизм – это реальное одновременное выполнение нескольких задач на разных ядрах процессора.

3. Что такое GIL? Какое ограничение накладывает GIL?

GIL (Global Interpreter Lock) – это глобальная блокировка интерпретатора в CPython, которая позволяет выполняться только одному потоку Python в один момент времени, даже на многопроцессорных системах. Ограничение: GIL препятствует реальной параллельной работе потоков Python на многоядерных процессорах для CPU-задач, делая их эффективными в основном для I/O-операций.

4. Каково назначение класса Thread?

Класс Thread предназначен для создания и управления потоками выполнения в программе, позволяя выполнять код конкурентно в рамках одного процесса.

5. Как реализовать в одном потоке ожидание завершения другого потока?

Вызвать метод join() у экземпляра потока, который нужно дождаться.

6. Как проверить факт выполнения потоком некоторой работы?

Проверить, жив ли поток, с помощью метода is_alive(), или использовать общие переменные-флаги, блокировки (Lock, Event) для отслеживания состояния.

7. Как реализовать приостановку выполнения потока на некоторый промежуток времени?

Использовать функцию time.sleep(секунды) из модуля time.

8. Как реализовать принудительное завершение потока?

В Python нет безопасного способа принудительно завершить поток. Рекомендуется использовать флаги для корректного завершения. Методы terminate() или kill() доступны для процессов (multiprocessing), но не для потоков напрямую.

9. Что такое потоки-демоны? Как создать поток-демон?

Поток-демон – это фоновый поток, который автоматически завершается при завершении основного потока программы. Чтобы создать поток-демон, нужно установить его свойство daemon в True перед запуском (thread.daemon = True) или передать аргумент daemon=True в конструктор Thread.

Вывод: в ходе лабораторной работы были приобретены навыки многопоточных приложений на языке программирования Python версии 3.13.3.