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
import threading
from concurrent.futures import ThreadPoolExecutor, as_completed
from dataclasses import dataclass
from typing import Tuple


def series_term(n: int, x: float) -> float:
    try:
        denominator = (2 * n - 1) * (x ** (2 * n - 1))
        return 1.0 / denominator
    except OverflowError:
        return 0.0


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


@dataclass
class ChunkResult:
    start_n: int
    end_n: int
    partial_sum: float
    terms_count: int


def calculate_chunk(
    start_n: int, end_n: int, x: float, epsilon: float, stop_flag:
```

```
threading.Event
) -> ChunkResult:
    partial_sum = 0.0
    terms_count = 0

    for n in range(start_n, end_n + 1):
        if stop_flag.is_set():
            break

        term = series_term(n, x)

        if term == 0.0 or abs(term) < epsilon:
            stop_flag.set()
            break

        partial_sum += term
        terms_count += 1

    return ChunkResult(
        start_n=start_n,
        end_n=start_n + terms_count - 1,
        partial_sum=partial_sum,
        terms_count=terms_count,
    )


def calculate_series(
    x: float, epsilon: float, num_threads: int = 4
) -> Tuple[float, int]:
    stop_flag = threading.Event()

    chunk_size = 100

    with ThreadPoolExecutor(max_workers=num_threads) as executor:
        futures = []
        next_n = 1

        for i in range(num_threads * 2):
            if stop_flag.is_set():
                break

            end_n = next_n + chunk_size - 1
            future = executor.submit(
                calculate_chunk, next_n, end_n, x, epsilon, stop_flag
            )
            futures.append(future)
            next_n = end_n + 1

        total_sum = 0.0
        total_terms = 0
        completed_chunks = 0

        for future in as_completed(futures):
            if stop_flag.is_set() and completed_chunks > 0:
                break

            try:
                result = future.result()
                total_sum += result.partial_sum
                total_terms += result.terms_count
                completed_chunks += 1

                if not stop_flag.is_set() and result.terms_count <
chunk_size:
```

```
                    stop_flag.set()

            except Exception as e:
                print(f"Ошибка при вычислении блока: {e}")

    return total_sum, total_terms


def main() -> None:
    x = 3.0
    epsilon = 1e-7

    print("Вычисление суммы бесконечного ряда с использованием
многопоточности")
    print("=" * 60)

    S, terms_count = calculate_series(x, epsilon)
    print(f"Сумма бесконечного ряда S = {S:.10f}")
    print(f"Вычислено членов ряда: {terms_count}")

    y = control_value(x)
    print(f"\nКонтрольное значение y = {y:.10f}")

    print("\nСравнение полученной суммы с контрольным значением:")
    print(f"|S - y| = {abs(S - y):.2e}")

    if abs(S - y) < epsilon:
        print(f"Точность достигнута: |S - y| < ε = {epsilon}")
    else:
        print(f"Точность не достигнута: |S - y| ≥ ε = {epsilon}")


if __name__ == "__main__":
    main()
```

```
PS C:\Users\4isto\OOP_lab7> python tasks/task1.py
Вычисление суммы бесконечного ряда с использованием многопоточности
============================================================
Сумма бесконечного ряда S = 0.3465735369
Вычислено членов ряда: 6

Контрольное значение y = 0.3465735903

Сравнение полученной суммы с контрольным значением:
|S - y| = 5.34e-08
Точность достигнута: |S - y| < ε = 1e-07
```

Рисунок 1 – Результат работы программы

Тесты для написанной программы:

Листинг программы:

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


import math
import threading

import pytest
```

```python
from task1 import (
    calculate_chunk,
    calculate_series,
    control_value,
    series_term,
)


def test_series_term_basic():
    assert math.isclose(series_term(1, 3), 1 / (1 * 3**1))
    assert math.isclose(series_term(2, 3), 1 / (3 * 3**3))
    assert math.isclose(series_term(3, 3), 1 / (5 * 3**5))


def test_series_term_overflow_safe():
    assert series_term(10_000, 3) == 0.0


def test_control_value():
    x = 3.0
    expected = 0.5 * math.log((x + 1) / (x - 1))
    assert math.isclose(control_value(x), expected)


def test_calculate_chunk_basic():
    x = 3.0
    epsilon = 1e-7
    stop_flag = threading.Event()

    result = calculate_chunk(1, 50, x, epsilon, stop_flag)

    assert result.partial_sum > 0
    assert result.terms_count > 0
    assert result.end_n >= result.start_n


def test_calculate_chunk_stop_on_epsilon():
    x = 3.0
    epsilon = 1e-3
    stop_flag = threading.Event()

    result = calculate_chunk(1, 1000, x, epsilon, stop_flag)

    assert result.terms_count < 1000
    assert stop_flag.is_set()


def test_calculate_series_accuracy():
    x = 3.0
    epsilon = 1e-7

    S, terms = calculate_series(x, epsilon)
    y = 0.5 * math.log((x + 1) / (x - 1))

    assert abs(S - y) < epsilon
    assert terms > 0


def test_calculate_series_small_threads():
    x = 3.0
    epsilon = 1e-7

    S1, t1 = calculate_series(x, epsilon, num_threads=1)
    S4, t4 = calculate_series(x, epsilon, num_threads=4)
```

```python
    assert math.isclose(S1, S4, rel_tol=1e-7)


def test_calculate_series_fast_stop():
    x = 3.0
    epsilon = 1e-2

    S, terms = calculate_series(x, epsilon)

    assert terms < 10
    assert abs(S - control_value(x)) < 1e-2


def test_calculate_chunk_raises_on_invalid_args():
    with pytest.raises(TypeError):
        calculate_chunk("invalid", 10, 3.0, 1e-7, threading.Event())
```

```
PS C:\Users\4isto\OOP_lab7> pytest
cachedir: .pytest_cache
rootdir: C:\Users\4isto\OOP_lab7
configfile: pyproject.toml
testpaths: tests
plugins: anyio-4.9.0
collected 9 items

tests/test_task1.py::test_series_term_basic PASSED
tests/test_task1.py::test_series_term_overflow_safe PASSED
tests/test_task1.py::test_control_value PASSED
tests/test_task1.py::test_calculate_chunk_basic PASSED
tests/test_task1.py::test_calculate_chunk_stop_on_epsilon PASSED
tests/test_task1.py::test_calculate_series_accuracy PASSED
tests/test_task1.py::test_calculate_series_small_threads PASSED
tests/test_task1.py::test_calculate_series_fast_stop PASSED
tests/test_task1.py::test_calculate_chunk_raises_on_invalid_args PASSED

============================================================================== 9 passed in 0.07s =
```

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.