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

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2.25
дисциплины «Анализ данных»

Выполнила:
Мурашко Анастасия Юрьевна
2 курс, группа ИВТ-б-о-22-1,
11.03.02 «Информатика и вычислительная техника», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р.А., канд. тех. наук, доцент,
доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой_____Дата защиты_____

Ставрополь, 2024 г.

Тема: Управление потоками в Python.

Цель работы: приобретение навыков написания многопоточных приложений на языке программирования Python версии 3.x.

Порядок выполнения работы:

Задание 1.

Изучила теоретический материал работы, создала общедоступный репозиторий на GitHub, в котором использована лицензий MIT и язык программирования Python, также добавила файл .gitignore с необходимыми правилами.

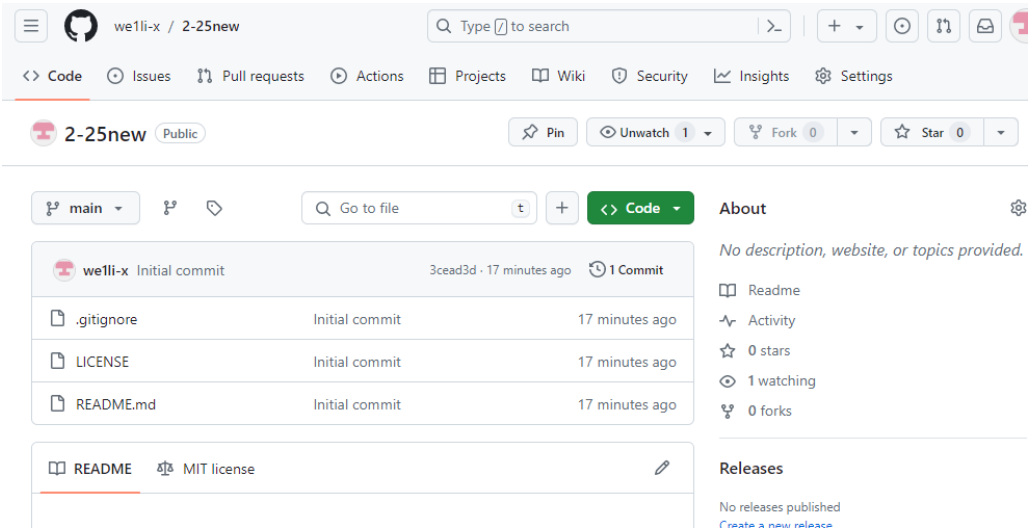

Рисунок 1. Новый репозиторий

Задание 2.

Проклонировала свой репозиторий на свой компьютер.

Организовала свой репозиторий в соответствие с моделью ветвления git-flow, появилась новая ветка develop.

Реализовывала примеры и индивидуальные задания на основе ветки develop, без создания дополнительной ветки feature/(название ветки) по указанию преподавателя.

Задание 3.

Создала виртуальное окружение (ВО) Miniconda и активировал его, также установила необходимые пакеты isort, black, flake8.

```
(base) C:\Users\nasty>cd C:\Users\nasty\PycharmProjects\pythonProject

(base) C:\Users\nasty\PycharmProjects\pythonProject>conda create -n 2.25 python=3.11
Channels:
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\nasty\anaconda3\envs\2.25

  added / updated specs:
    - python=3.11
```

Рисунок 3. Создание ВО

```
(base) C:\Users\nasty\PycharmProjects\pythonProject>conda activate 2.25

(2.25) C:\Users\nasty\PycharmProjects\pythonProject>conda install -c conda-forge black
Channels:
 - conda-forge
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done
```

Рисунок 4. Установка пакета black

```
(2.25) C:\Users\nasty\PycharmProjects\pythonProject>conda install -c conda-forge flake8
Channels:
 - conda-forge
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done
```

Рисунок 5. Установка пакета flake8

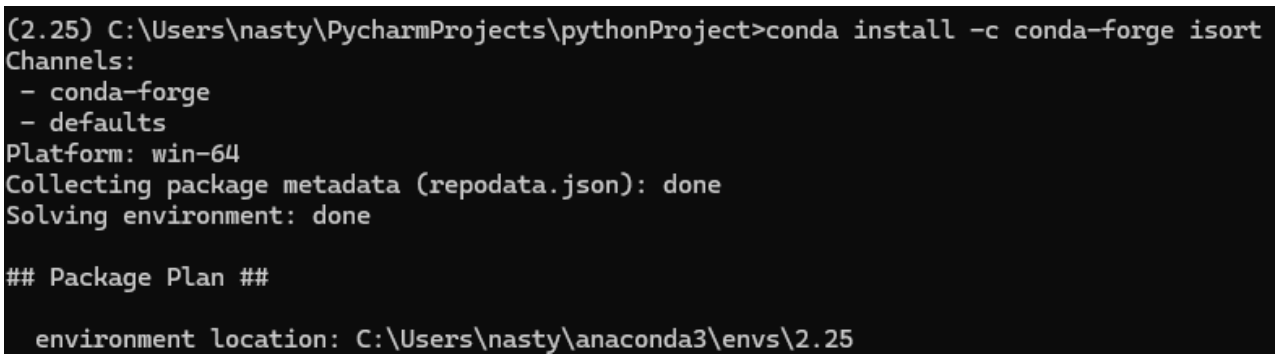

Рисунок 6. Установка пакета isort

Пакет isort (isrot) является инструментом для автоматической сортировки импортов в Python-кодах. Он используется для удобства чтения и поддержания порядка в коде.

Пакет black представляет инструмент автоматического форматирования кода для языка Python. Он помогает обеспечить единообразие стиля кодирования в проекте и улучшает читаемость кода.

Пакет flake8 отвечает за статический анализ и проверку Python-кода. Он проводит проверку на соответствие стилю кодирования PEP 8, а также наличие потенциальных ошибок и проблемных паттернов в коде.

Задание 3.

Выполнение индивидуального задания.

Для своего индивидуального задания лабораторной работы 2.23 необходимо реализовать вычисление значений в двух функций в отдельных процессах.

Условие задания 2.23: с использованием многопоточности для заданного значения **x** найти сумму ряда **S** с точностью члена ряда по абсолютному значению e = 10^-7 и произвести сравнение полученной суммы с контрольным значением функции для двух бесконечных рядов.

$$S = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots; \; x = 0,3; \; y = \cos x.$$

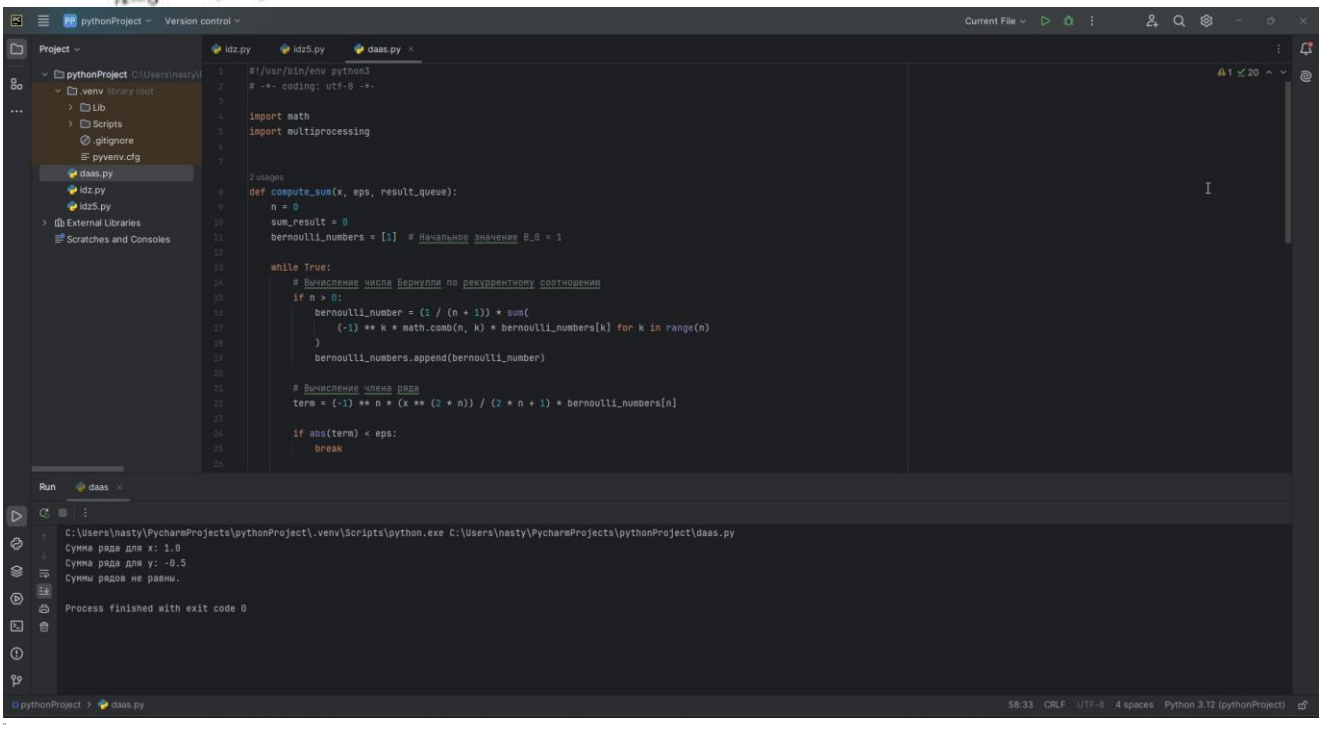

Рисунок 7. Результат индивидуального задания

Код начинается с импорта необходимых модулей - `math` для математических операций и `multiprocessing` для работы с многопоточностью.

Для реализации вычисления значений двух функций в отдельных процессах, мы будем использовать модуль `multiprocessing` вместо `threading`.

Многопроцессорность более подходит для задач, требующих интенсивных вычислений, так как каждый процесс выполняется в отдельном адресном пространстве и использует отдельный набор ядер процессора.

Следующим определяется класс `SumThread`, который наследуется от `threading.Thread`. Этот класс переопределяет метод `_init_`, чтобы принимать два аргумента: `x` и `eps`. В методе `_init_` также инициализируется переменная `self.sum`, которая будет хранить вычисленную сумму.

Переменная k используется в списковом включении (генераторе) при вычислении числа Бернулли. Это переменная цикла, которая перебирает значения от 0 до n+1

Список bernoulli_numbers инициализируется значением B_0 = 1. Рекуррентное соотношение используется для вычисления значений чисел Бернулли B_n для n > 0. Вместо факториалов в формуле члена ряда используются числа Бернулли.

Метод `run` вычисляет члены ряда и добавляет их к `self.sum`, пока абсолютное значение члена ряда больше `eps`.

В этом коде мы создаем процессы вместо потоков. Каждый процесс вычисляет сумму ряда для заданного значения x и y с помощью функции `compute_sum`. Результаты вычислений передаются через очереди `multiprocessing.Queue()`, чтобы можно было получить их в основном процессе для сравнения.

Запуск функции `compare_sums` происходит внутри блока `if __name__ == "__main__":`, что гарантирует корректный запуск процессов при импорте этого скрипта в другие модули или при запуске его как основного скрипта.

Следующим определяется функция `compare_sums`, которая создает два экземпляра класса `SumThread`, каждый из которых вычисляет сумму ряда для

своего аргумента. Затем она запускает оба потока с помощью метода `start`, а затем ожидает их завершения с помощью метода `join`.

После завершения работы потоков, функция `compare_sums` извлекает вычисленные суммы из потоков и выводит их на экран. Затем она сравнивает эти суммы и выводит сообщение о том, равны ли они.

Наконец, вызывается функция `compare_sums` с аргументами `0`, `3`и `10**-7`.

Ответы на контрольные вопросы:

1. Как создаются и завершаются процессы в Python?

Создание процессов:

Модуль os:

- Используйте функцию os.fork(), которая создает копию текущего процесса.
- Возвращает 0 в дочернем процессе и PID (идентификатор процесса) дочернего процесса в родительском процессе.

Модуль subprocess:

- Предоставляет функции более высокого уровня для работы с процессами, такие как Popen(), check_output() и call().
- Упрощает запуск процессов, захват их вывода и управление ими.

Модуль multiprocessing:

- Предназначен для более сложного управления несколькими процессами.
- Предоставляет классы Process и Queue для создания и взаимодействия между процессами.

Завершение процессов:

Метод join():

- Блокирует родительский процесс до завершения дочернего.

Метод terminate():

- Отправляет процессу сигнал SIGTERM, который **обычно** приводит к его **корректной остановке**.

Метод kill():

- Отправляет процессу сигнал SIGKILL, который **немедленно**

завершает его работу **без очистки ресурсов**.

2. Особенности создания классов-наследников от Process: Взомжность создать собственные классы-наследники от Process, чтобы лучше структурировать код и данные, связанные с процессом. Основное отличие заключается в том, что вы можете переопределить метод run, который будет выполняться при запуске процесса. Это позволяет более гибко управлять поведением процесса. Совместимость с threading.Thread: Поддерживает сигнатуры методов и конструктора, упрощая переход от многопоточного к многопроцессному приложению.
3. Принудительное завершение процесса: Принудительное завершение процесса осуществляется методом terminate. После вызова этого метода процесс завершится немедленно. Стоит отметить, что terminate может оставить ресурсы в некорректном состоянии, поэтому его следует использовать с осторожностью.
4. Процессы-демоны: Процессы-демоны (daemon processes) работают в фоновом режиме и автоматически завершаются, когда завершится основной процесс. Запуск процесса-демона: Чтобы запустить процесс в режиме демона, установите его атрибут daemon в True перед вызовом start.

Вывод: в ходе выполнения лабораторной работы были приобретены навыки написания многопоточных приложений на языке программирования Python версии 3.x.