МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Отчет по лабораторной работе №2.25

Управление процессами в Python

По дисциплине «Теории программирования и алгоритмизации»

Выполнил студент группы ИВТ-б-о-20-1

Плотников Д. В. « »_____20__г.

Подпись студента_____

Работа защищена « »_____20__г.

Проверил Воронкин Р. А. _____

(подпись)

Ставрополь 2022

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

Ход работы

1. Создал новый собственный репозиторий. Ссылка на репозиторий: https://github.com/Dmitry-15/2.25_laba.

2. С помощью команды git clone клонировал удаленный репозиторий на свой ПК. Дополнил файл .gitignore необходимыми правилами для работы с IDE PyCharm.

3. После ознакомления с теоретическим материалом приступил к выполнению задания.

Индивидуальное задание

1. Условие задания: для своего индивидуального задания лабораторной работы 2.23 необходимо реализовать вычисление значений в двух функций в отдельных процессах.

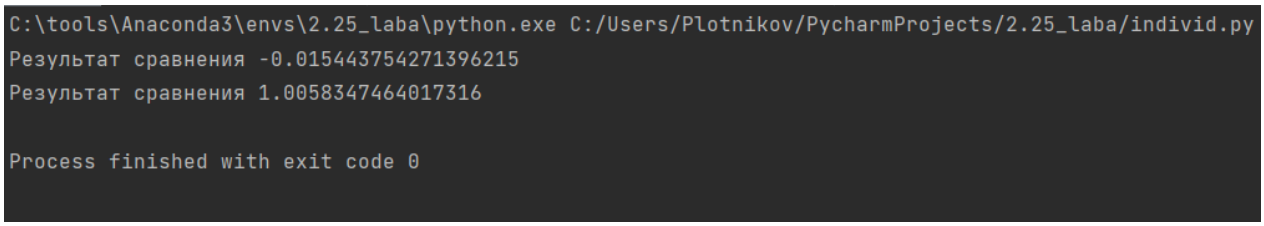

Рисунок 1 – Результат выполнения индивидуального задания

Контрольные вопросы:

1. Как создаются и завершаются процессы в Python?

Классом, который отвечает за создание и управление процессами является *Process* из пакета *multiprocessing*. Он совместим по сигнатурам методов и конструктора с *threading.Thread*, это сделано для более простого перехода от многопотокового приложения к многопроцессному.

За ожидание завершения работы процесса(ов) отвечает метод *join*, со следующей сигнатурой: *join([timeout])*.

При вывозе метода *join()* выполнение программы будет остановлено до тех пор пока соответствующий процесс не завершит работу. Параметр *timeout* отвечает за время ожидания завершения работы процесса, если указанное

время прошло, а процесс еще не завершился, то ожидание будет прервано и выполнение программы продолжится дальше. В случае, если метод *join()* завершился по таймауту или в результате того, что процесс был завершен аварийно (терминирован), то он вернет *None*.

2. В чем особенность создания классов-наследников от Process?

В классе наследнике от *Process* необходимо переопределить метод *run()* для того, чтобы он (класс) соответствовал протоколу работы с процессами.

3. Как выполнить принудительное завершение процесса?

В отличии от потоков, работу процессов можно принудительно завершить, для этого класс *Process* предоставляет набор методов:

- terminate() - принудительно завершает работу процесса. В *Unix* отправляется команда *SIGTERM*, в *Windows* используется функция *TerminateProcess()*.
- kill() - метод аналогичный *terminate()* по функционалу, только вместо *SIGTERM* в *Unix* будет отправлена команда *SIGKILL*.

4. Что такое процессы-демоны? Как запустить процесс-демон?

Процессы демоны по своим свойствам похожи на потоки-демоны, их суть заключается в том, что они завершают свою работу, если завершился родительский процесс.

Указание на то, что процесс является демоном должно быть сделано до его запуска (до вызова метода *start()*). Для демонического процесса запрещено самостоятельно создавать дочерние процессы. Эти процессы не являются демонами (сервисами) в понимании *Unix*, единственное их свойство – это завершение работы вместе с родительским процессом.

Указать на то, что процесс является демоном можно при создании экземпляра класса через аргумент *daemon*, либо после создания через свойство *daemon*.

Вывод: в ходе выполнения лабораторной работы успешно приобретены навыки по написания многозадачных приложений на языке программирования Python версии 3.x.