

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 №11
дисциплины «Анализ данных»

Выполнил:
Кожуховский Виктор Андреевич
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной
техники и автоматизированных систем
», очная форма обучения

(подпись)

Руководитель практики:
Воронкин Роман Александрович

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

Тема: Управление процессами в Python

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

Порядок выполнения работы:

1. Изучил теоретический материал работы.
2. Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT и язык программирования Python.
3. Выполнил клонирование созданного репозитория.
4. Дополнил файл .gitignore необходимыми правилами для работы с IDE PyCharm.
5. Организовал свой репозиторий в соответствие с моделью ветвления git-flow.
6. Создал проект в папке репозитория.
8. Выполнил индивидуальное задание.

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

Код:

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

import math
from multiprocessing import Process, Array

epsilon = 1e-7

def func(x, result):
    sum = 0
    n = 0
    term = 1
    while abs(term) > epsilon:
        sum += term
        n += 1
        term = (-1)**n * x**(2 * n) / math.factorial(n)
    result[0] = sum

def func2(x, result):
    sum = 0
    n = 1
    while True:
        term = 1 / (2 * n - 1) * ((x - 1) / (x + 1))**(2 * n - 1)
        if abs(term) < epsilon:
            break
        else:
            sum += term
            n += 1
    result[1] = sum
```

```

def main():
    result = Array('d', [0.0, 0.0])

    process1 = Process(target=func, args=(-0.7, result))
    process2 = Process(target=func2, args=(0.6, result))

    process1.start()
    process2.start()

    process1.join()
    process2.join()

    sum_func = result[0]
    sum_func2 = result[1]

    test1 = math.exp(-(-0.7)**2)
    test2 = 1/2 * math.log(0.6)

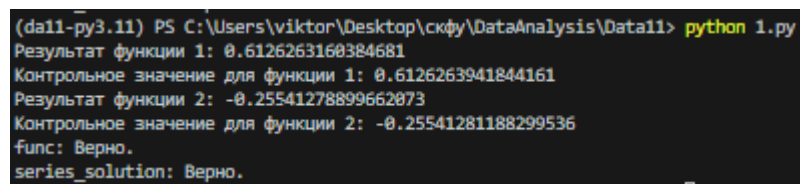
    print(f"Результат функции 1: {sum_func}")
    print(f"Контрольное значение для функции 1: {test1}")
    print(f"Результат функции 2: {sum_func2}")
    print(f"Контрольное значение для функции 2: {test2}")

    if abs(sum_func - test1) < epsilon:
        print("func: Верно.")
    else:
        print("func: Неверно.")

    if abs(sum_func2 - test2) < epsilon:
        print("series_solution: Верно.")
    else:
        print("series_solution: Неверно.")

if __name__ == "__main__":
    main()

```



```

(dall1-py3.11) PS C:\Users\viktor\Desktop\скфу\DataAnalysis\Data11> python 1.py
Результат функции 1: 0.6126263160384681
Контрольное значение для функции 1: 0.6126263941844161
Результат функции 2: -0.25541278899662073
Контрольное значение для функции 2: -0.25541281188299536
func: Верно.
series_solution: Верно.

```

Рисунок 1. Решение индивидуального задания

9. Зафиксировал сделанные изменения в репозитории.
10. Добавил отчет по лабораторной работе в формате PDF в папку doc репозитория. Зафиксировал изменения.
11. Выполнил слияние ветки для разработки с веткой master/main.
12. Отправил сделанные изменения на сервер GitHub.

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 многопоточкового приложения к многопроцессному. Помимо

одноименных с Thread методов, класс Process дополнительно предоставляет ряд своих.

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. Ниже представлен пример с реализацией этого подхода.

3. Как выполнить принудительное завершение процесса?

В отличие 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.