

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 №2.23
дисциплины «Объектно-ориентированное программирование»

Выполнил:
Магомедов Имран Борисович
3 курс, группа «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Руководитель практики:
Воронкин Р.А., доцент департамента
цифровых, робототехнических систем и
электроники

(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.

Методика выполнения работы

1. Изучить теоретический материал работы.
2. Проработайте примеры лабораторной работы.
3. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*

Owner * m4g0med0v / **Repository name ***

✔ OOP_LR_10 is available.

Great repository names are short and memorable. Need inspiration? How about **crispy-winner** ?

Description (optional)

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

4. Выполните клонирование созданного репозитория.
5. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.

6. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.

7. Выполните индивидуальное задание. Приведите в отчете скриншоты работы программы решения индивидуального задания.

С использованием многопоточности для заданного значения найти сумму ряда с точностью члена ряда S по абсолютному значению $\varepsilon = 10^{-7}$ и произвести сравнение полученной суммы с контрольным значением функции для двух бесконечных рядов. Номера вариантов необходимо уточнить у преподавателя:

$$S = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \dots; \quad x = 2; \quad y = \frac{e^x - e^{-x}}{2}.$$

Решение:

```
9 import math
10 import threading
11 from queue import Queue
12
13
14 # Функция для вычисления одного члена ряда
15 def calculate_term(x, n):
16     return (x ** (2 * n + 1)) / math.factorial(2 * n + 1)
17
18
19 # Функция для вычисления суммы ряда с заданной точностью
20 def calculate_series(x, epsilon, queue):
21     n = 0
22     term = calculate_term(x, n)
23     total_sum = 0
24     while abs(term) > epsilon:
25         total_sum += term
26         n += 1
27         term = calculate_term(x, n)
28     queue.put(total_sum)
29     queue.task_done()
```

```

32 def main():
33     queue = Queue()
34
35     # Значения x, epsilon, и расчет y
36     x = 2
37     epsilon = 1e-7
38     y = (math.exp(x) - math.exp(-x)) / 2 # Контрольное значение
39
40     # Создаем и запускаем поток
41     thread = threading.Thread(
42         target=calculate_series, args=(x, epsilon, queue)
43     )
44     thread.start()
45
46     # Ожидаем завершения потоков
47     thread.join()
48     queue.join()
49
50     # Получаем результат
51     s = queue.get()
52
53     # Вывод результатов
54     print(f"Рассчитанная сумма ряда S: {s}")
55     print(f"Контрольное значение функции y: {y}")
56     print(f"Разница между S и y: {abs(s - y)}")
57
58
59 if __name__ == "__main__":
60     main()

```

8. Зафиксируйте сделанные изменения в репозитории.
9. Выполните слияние ветки для разработки с веткой main (master).
10. Отправьте сделанные изменения на сервер GitHub.

Контрольные вопросы

1. Что такое синхронность и асинхронность?

Синхронность: Это модель выполнения, где операции выполняются последовательно, то есть каждая операция должна завершиться до начала следующей. В случае блокирующих операций, выполнение программы приостанавливается, пока не завершится текущая задача.

Асинхронность: Это модель выполнения, при которой операции могут выполняться независимо друг от друга, не блокируя основной поток исполнения. Ожидание результатов может происходить в фоновом режиме, и выполнение программы не приостанавливается, пока не завершится выполнение этих операций.

2. Что такое параллелизм и конкурентность?

Параллелизм: Это ситуация, когда несколько операций выполняются одновременно (параллельно). Это может происходить, если у системы есть несколько процессоров или ядер, и они могут одновременно обрабатывать различные задачи.

Конкурентность: Это концепция, при которой несколько задач могут быть выполнены в промежутках друг друга (не обязательно одновременно). Например, в многозадачной операционной системе различные процессы или потоки могут выполнять операции в разные моменты времени, но не обязательно одновременно. Это включает в себя параллельное выполнение, но не обязательно требует многозадачности на уровне аппаратного обеспечения.

3. Что такое GIL? Какое ограничение накладывает GIL?

GIL (Global Interpreter Lock) — это механизм синхронизации в интерпретаторе CPython (официальная реализация Python), который ограничивает выполнение байт-кода Python в одно время только в одном потоке. Это означает, что даже на многопроцессорных системах только один поток может исполнять Python-код одновременно.

Ограничения: GIL не позволяет эффективно использовать многозадачность на многопроцессорных системах для вычислительных задач, так как он блокирует выполнение Python-кода в разных потоках. Однако для задач ввода-вывода (например, работа с сетью, файловой системой) GIL обычно не мешает, так как в эти моменты потоки могут переключаться.

4. Каково назначение класса Thread ?

Класс Thread в Python используется для создания и управления потоками выполнения. Он предоставляет методы для запуска потоков, их остановки и синхронизации. Потоки, созданные с помощью этого класса, выполняют функции параллельно с основным потоком программы.

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

Для ожидания завершения другого потока в Python используется метод `join()`:

```
1 import threading
2
3 def task():
4     print("Задача выполнена")
5
6 thread = threading.Thread(target=task)
7 thread.start()
8 thread.join() # Ожидаем завершения потока
9 print("Главный поток продолжает выполнение")
```

Метод `join()` блокирует выполнение текущего потока до тех пор, пока указанный поток не завершит свою работу.

6. Как проверить факт выполнения потоком некоторой работы?

Можно использование метода `is_alive()`.

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

Механизм **блокировки (lock)** используется для предотвращения доступа нескольких потоков к общим данным одновременно. Если одному потоку предоставляется блокировка (с помощью `lock.acquire()`), другие потоки должны ожидать, пока этот поток не освободит блокировку с помощью `lock.release()`.

Если вы хотите "приостановить" поток с помощью блокировки, можно использовать метод `acquire()` с параметром `timeout`, чтобы поток ждал освобождения блокировки на определённое время, а затем продолжил выполнение.

8. Как реализовать принудительное завершение потока?

В Python нет прямого способа принудительно завершить поток, так как потоки должны завершать работу корректно. Однако можно использовать флаг, чтобы указать потоку, что ему нужно завершиться:

```

1 import threading
2 import time
3
4 def task(stop_flag):
5     while not stop_flag.is_set():
6         print("Поток работает")
7         time.sleep(1)
8
9 stop_flag = threading.Event()
10 thread = threading.Thread(target=task, args=(stop_flag,))
11 thread.start()
12
13 time.sleep(5) # Работаем 5 секунд
14 stop_flag.set() # Устанавливаем флаг для завершения потока
15 thread.join()

```

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

Потоки-демоны — это потоки, которые работают в фоновом режиме и не блокируют завершение программы. Когда все остальные потоки завершат свою работу, программа завершится, даже если потоки-демоны ещё продолжают работать.

Для создания потока-демона нужно установить атрибут `daemon` в `True` перед запуском потока:

```

1 import threading
2 import time
3
4 def task():
5     print("Поток-демон работает")
6     time.sleep(10)
7     print("Поток-демон завершён")
8
9 thread = threading.Thread(target=task)
10 thread.daemon = True # Устанавливаем поток как демона
11 thread.start()
12
13 time.sleep(2) # Главный поток завершится до завершения потока-демона

```