

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

Институт цифрового развития
Кафедра инфокоммуникаций

«Управление потоками в Python»

ОТЧЕТ
по лабораторной работе №23
дисциплины
«Основы программной инженерии»

Выполнила:

Кувшин Ирина Анатольевна
2 курс, группа ПИЖ-б-о-21-1,
011.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Проверил:

(подпись)

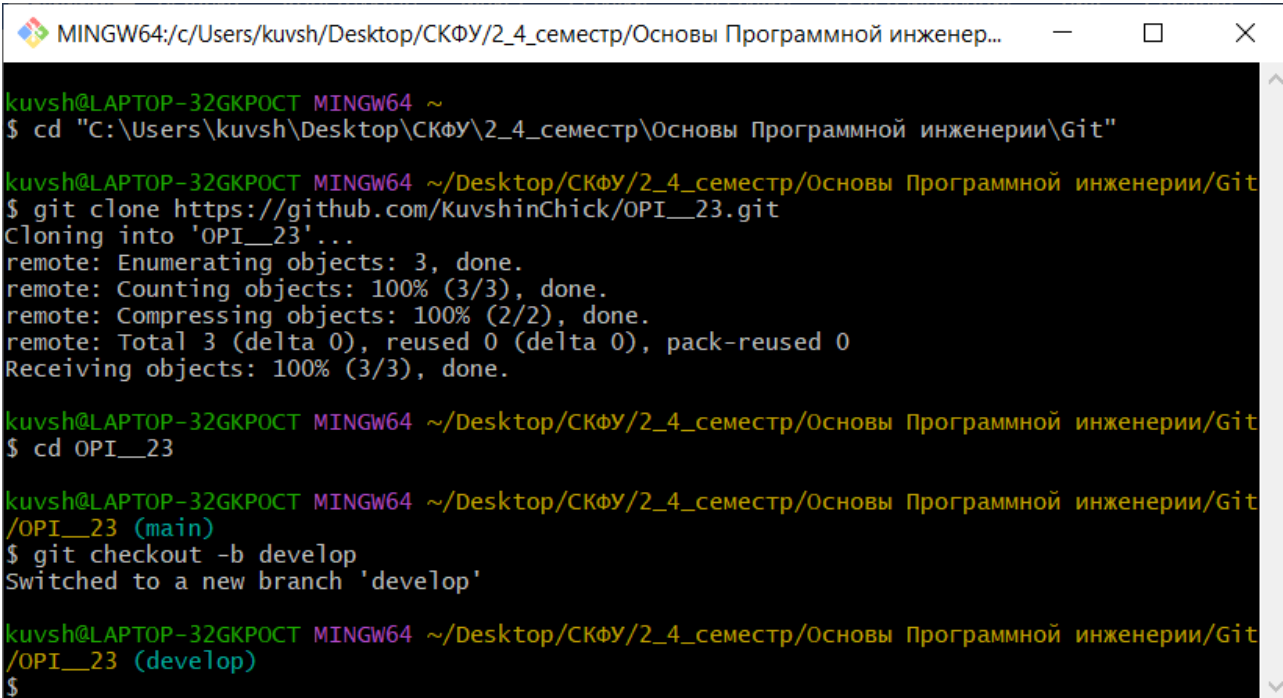
Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

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

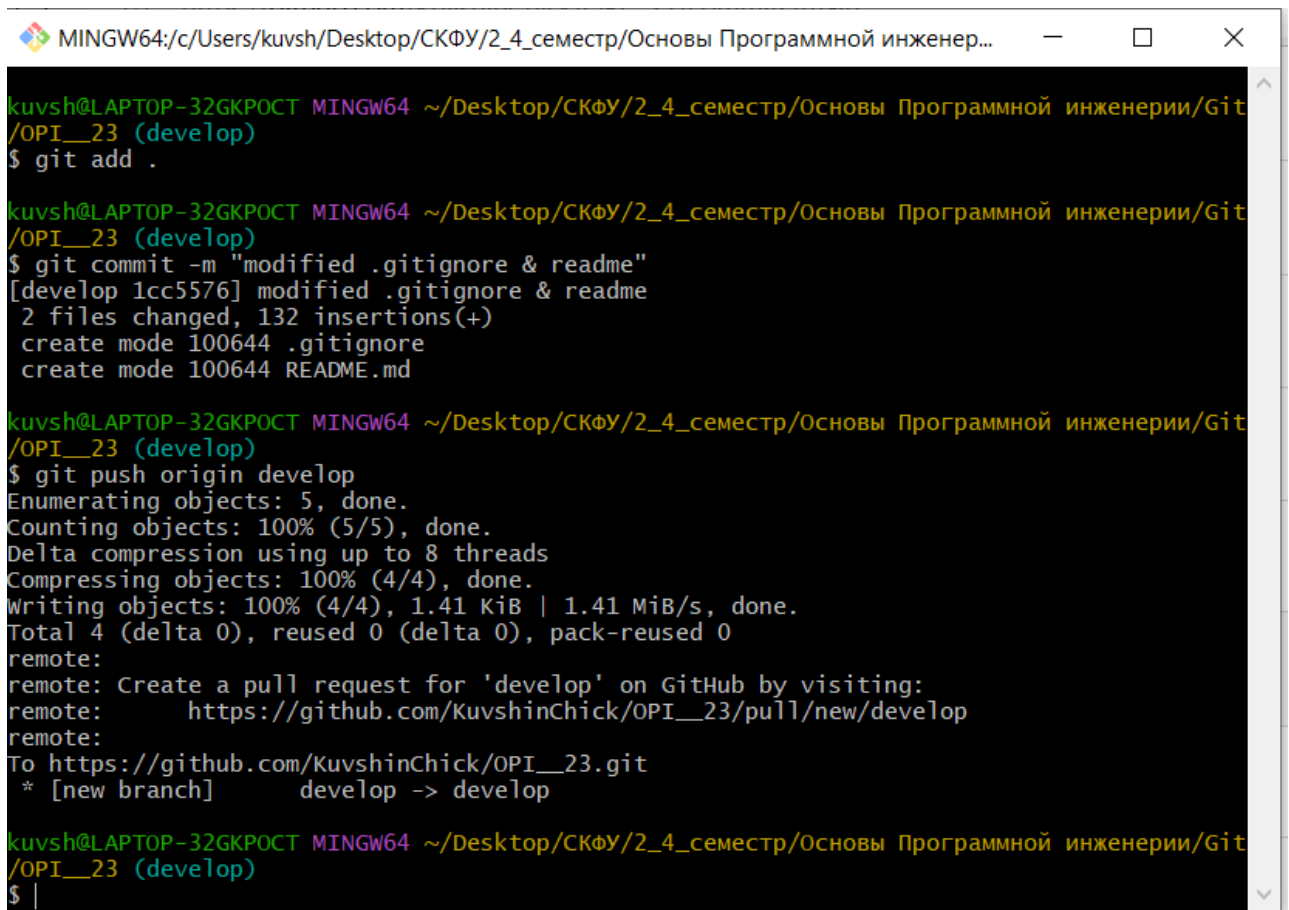
Ход работы:

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.
4. Выполните клонирование созданного репозитория.
5. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.
6. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.
7. Создайте проект PyCharm в папке репозитория.



```
MINGW64/c:/Users/kuvsh/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии...
kuvsh@LAPTOP-32GKPOCT MINGW64 ~
$ cd "C:\Users\kuvsh\Desktop\СКФУ\2_4_семестр\Основы Программной инженерии\Git"
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git
$ git clone https://github.com/KuvshinChick/OPI__23.git
Cloning into 'OPI__23'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git
$ cd OPI__23
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__23 (main)
$ git checkout -b develop
Switched to a new branch 'develop'
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__23 (develop)
$
```

Рисунок 23.1 – Клонирование репозитория и создание ветки develop



```
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__23 (develop)
$ git add .

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__23 (develop)
$ git commit -m "modified .gitignore & readme"
[develop 1cc5576] modified .gitignore & readme
2 files changed, 132 insertions(+)
create mode 100644 .gitignore
create mode 100644 README.md

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__23 (develop)
$ git push origin develop
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1.41 KiB | 1.41 MiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:   https://github.com/KuvshinChick/OPI__23/pull/new/develop
remote:
To https://github.com/KuvshinChick/OPI__23.git
 * [new branch]      develop -> develop

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__23 (develop)
$ |
```

Рисунок 23.2 – Обновление .gitignore и readme

8. Выполните индивидуальное задание. Приведите в отчете скриншоты работы программы решения индивидуального задания.
9. Зафиксируйте сделанные изменения в репозитории.
10. Выполните слияние ветки для разработки с веткой main (master).
11. Отправьте сделанные изменения на сервер GitHub.

Индивидуальное задание

С использованием многопоточности для заданного значения найти сумму ряда с точностью члена ряда по абсолютному значению и произвести сравнение полученной суммы с контрольным значением функции для двух бесконечных рядов. Номера вариантов необходимо уточнить у преподавателя:

8.
$$S = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!} = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \dots; \quad x = 2; \quad y = \frac{e^x - e^{-x}}{2}. \quad (14)$$

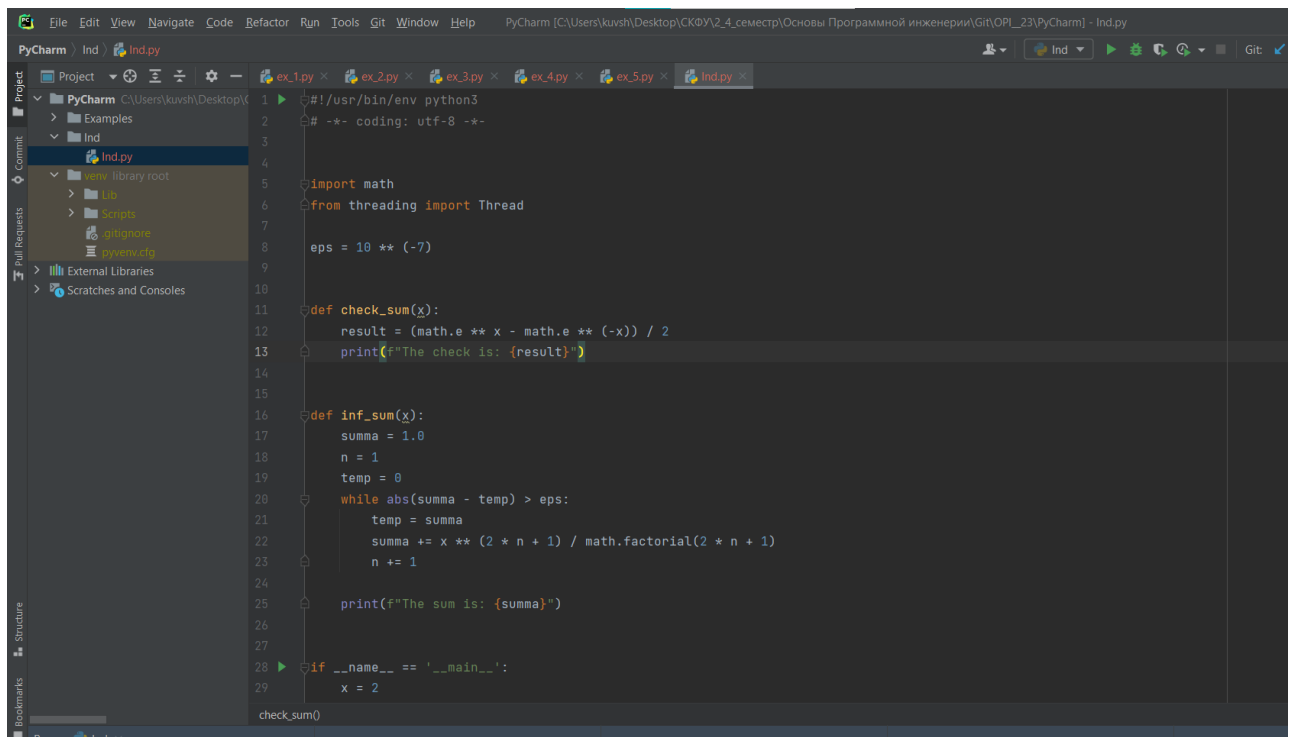


Рисунок 23.3 – Проработка программы

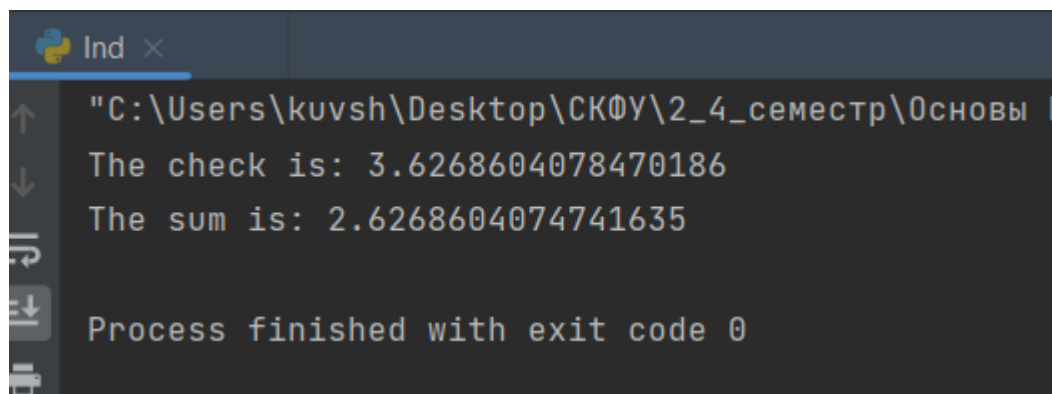


Рисунок 23.4 – Результат работы программы

Контрольные вопросы

1. Что такое синхронность и асинхронность?

Синхронное выполнение программы подразумевает последовательное выполнение операций. Асинхронное – предполагает возможность независимого выполнения задач.

2. Что такое параллелизм и конкурентность?

Конкурентность предполагает выполнение нескольких задач одним исполнителем. Из примера с готовкой: один человек варит картошку и

прибирается, при этом, в процессе, он может переключаться: немного прибрался, пошел помешал-посмотрел на картошку, и делает он это до тех пор, пока все не будет готово. Параллельность предполагает параллельное выполнение задач разными исполнителями: один человек занимается готовкой, другой приборкой. В примере с математикой операции и могут выполнять два разных процессора.

3. Что такое GIL? Какое ограничение накладывает GIL?

Конкурентность предполагает выполнение нескольких задач одним исполнителем. Из примера с готовкой: один человек варит картошку и прибирается, при этом, в процессе, он может переключаться: немного прибрался, пошел помешал-посмотрел на картошку, и делает он это до тех пор, пока все не будет готово. Параллельность предполагает параллельное выполнение задач разными исполнителями: один человек занимается готовкой, другой приборкой. В примере с математикой операции и могут выполнять два разных процессора.

4. Каково назначение класса Thread ?

За создание, управление и мониторинг потоков отвечает класс Thread из модуля threading. Поток можно создать на базе функции, либо реализовать свой класс – наследник Thread и переопределить в нем метод run().

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

Если необходимо дождаться завершения работы потока(ов) перед тем как начать выполнять какую-то другую работу, то воспользуйтесь методом join():

```
th1 = Thread(target=func)
th2 = Thread(target=func)

th1.start()
th2.start()

th1.join()
th2.join()

print("--> stop")
```

У `join()` есть параметр `timeout`, через который задается время ожидания завершения работы потоков.

6. Как проверить факт выполнения потоком некоторой работы?

Для того, чтобы определить выполняет ли поток какую-то работу или завершился используется метод `is_alive()`.

```
th = Thread(target=func)
print(f"thread status: {th.is_alive()}")

th.start()
print(f"thread status: {th.is_alive()}")

sleep(5)
print(f"thread status: {th.is_alive()}")
```

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

В Python можно использовать функцию `time.sleep()` для приостановки выполнения потока на некоторый промежуток времени.

```

from threading import Thread
from time import sleep

def func():
    for i in range(5):
        print(f"from child thread: {i}")
        sleep(0.5)

th = Thread(target=func)
th.start()

for i in range(5):
    print(f"from main thread: {i}")
    sleep(1)

```

В приведенном выше примере мы импортировали нужные модули. После этого объявили функцию *func()*, которая выводит пять раз сообщение с числовым маркером с задержкой в 500 мс. Далее создали объект класса *Thread*, в нем, через параметр *target*, указали, какую функцию запускать как поток и запустили его. В главном потоке добавили код вывода сообщений с интервалом в 1000 мс.

8. Как реализовать принудительное завершение потока?

В Python у объектов класса *Thread* нет методов для принудительного завершения работы потока. Один из вариантов решения этой задачи – это создать специальный флаг, через который потоку будет передаваться сигнал остановки. Доступ к такому флагу должен управляться объектом синхронизации.

```

from threading import Thread, Lock
from time import sleep

lock = Lock()

stop_thread = False

def infinit_worker():
    print("Start infinit_worker()")
    while True:
        print("--> thread work")
        lock.acquire()

        if stop_thread is True:
            break

        lock.release()
        sleep(0.1)

    print("Stop infinit_worker()")

# Create and start thread
th = Thread(target=infinit_worker)
th.start()

sleep(2)

# Stop thread
lock.acquire()
stop_thread = True
lock.release()

```

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

Есть такая разновидность потоков, которые называются демоны (терминология взята из мира Unix-подобных систем). Python-приложение не будет закрыто до тех пор, пока в нем работает хотя бы один недемонический поток.

```
def func():
    for i in range(5):
        print(f"from child thread: {i}")
        sleep(0.5)

th = Thread(target=func)
th.start()

print("App stop")
```

Вывод программы:

```
from child thread: 0
App stop
from child thread: 1
from child thread: 2
from child thread: 3
from child thread: 4
```

Как вы можете видеть, приложение продолжает работать, даже после того, как главный поток завершился (сообщение: "App stop").

Для того, чтобы потоки не мешали остановке приложения (т.е. чтобы они останавливались вместе с завершением работы программы) необходимо при создании объекта *Thread* аргументу *daemon* присвоить значение *True*, либо после создания потока, перед его запуском присвоить свойству *daemon* значение *True*. Изменим процесс создания потока в приведенной выше программе:

```
th = Thread(target=func, daemon=True)
```

Запустим ее, получим следующий результат:

```
from child thread: 0
App stop
```

Поток остановился вместе с остановкой приложения.