

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

**Кафедра инфокоммуникаций  
Отчет по лабораторной работе № 2.23  
«Управление потоками в Python»  
по дисциплине «Основы программной инженерии»**

Выполнил студент группы

ПИЖ-б-о-21-1

Зиберов Александр

«    » мая 2023 г.

Подпись студента \_\_\_\_\_

Работа защищена

«    » \_\_\_\_\_ 20\_\_ г.

Проверил Воронкин Р.А. \_\_\_\_\_  
(подпись)

Ставрополь, 2023

## Цель работы:

Приобретение навыков написания многопоточных приложений на языке программирования Python версии 3.x.

## Выполнение работы:

Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT, рисунок 1.

Ссылка: <https://github.com/afk552/lab2.23>

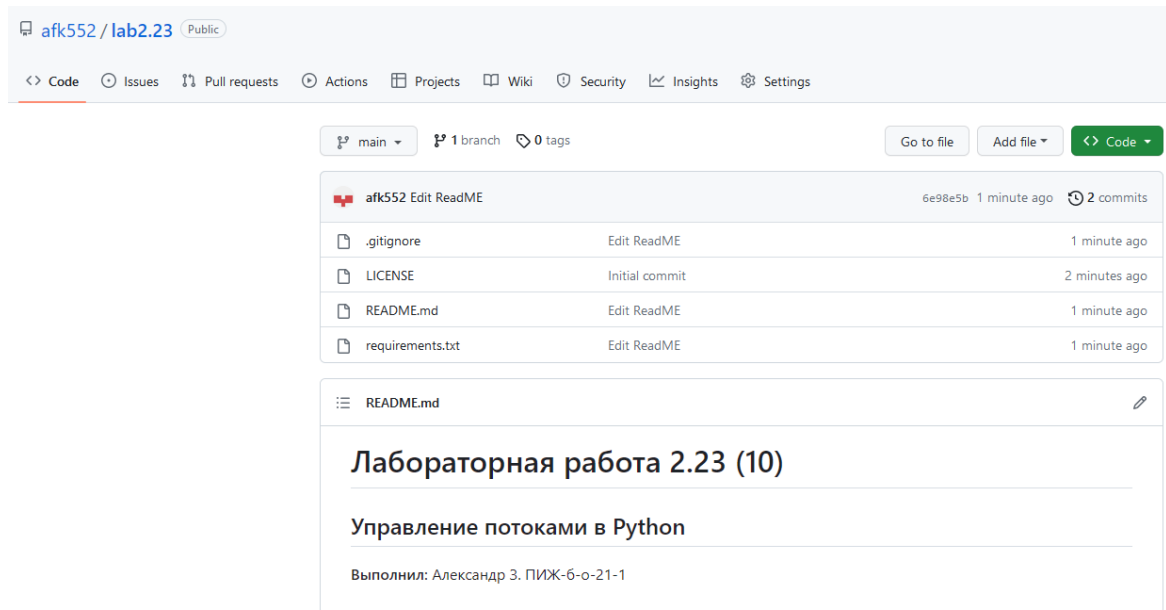


Рисунок 1 – Удаленный репозиторий на GitHub

Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm, рисунок 2.

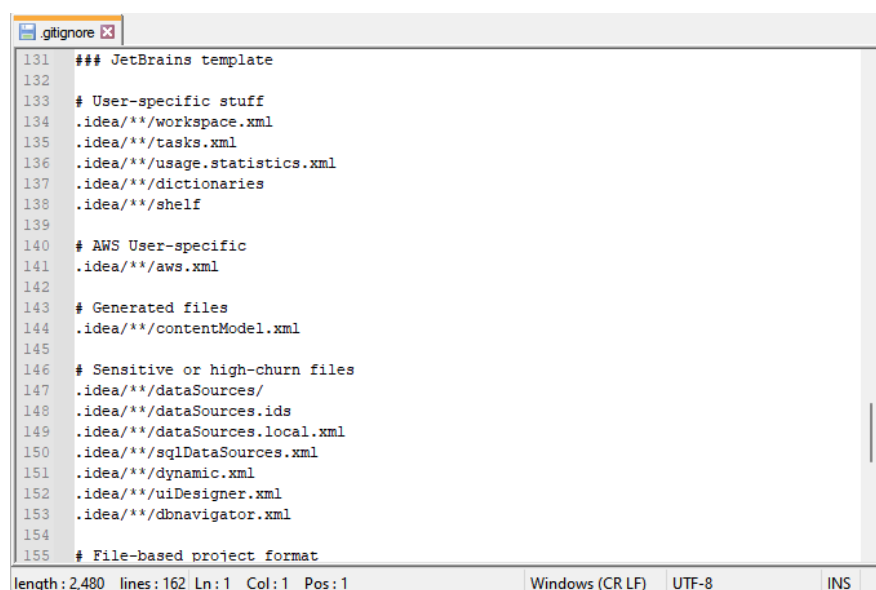


Рисунок 2 – Окно блокнота

Организируйте свой репозиторий в соответствии с моделью ветвления git-flow, рисунок 3.

```
C:\git\lab2.23>git checkout -b develop
Switched to a new branch 'develop'

C:\git\lab2.23>git branch
* develop
  main
```

Рисунок 3 – Окно командной строки

## Примеры

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  from threading import Thread
5  from time import sleep
6
7
8  def func():
9      for i in range(5):
10         print(f"from child thread: {i}")
11         sleep(0.5)
12
13
14  if __name__ == '__main__':
15      th = Thread(target=func)
16      th.start()
17
18      for i in range(5):
19         print(f"from main thread: {i}")
20         sleep(1)
```

```
Run [icon] [icon] [icon]
C:\git\lab2.23\venv\Scripts\python.exe C:\git\lab2.23\examples\ex1.py
from child thread: 0
from main thread: 0
from child thread: 1
from child thread: 2
from main thread: 1
from child thread: 3
from main thread: 2
from child thread: 4
from main thread: 3
from main thread: 4
Process finished with exit code 0
```

Рисунок 4 – Пример 1

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  from threading import Thread
6  from time import sleep
7
8
9  def func():
10     for i in range(5):
11         print(f"from child thread: {i}")
12         sleep(0.5)
13
14
15  ▶ if __name__ == '__main__':
16     th = Thread(target=func)
17     print(f"thread status: {th.is_alive()}")
18     th.start()
19     print(f"thread status: {th.is_alive()}")
20     sleep(5)
21     print(f"thread status: {th.is_alive()}")
22

```

Run

```

C:\git\lab2.23\venv\Scripts\python.exe C:\git\lab2.23\examples\ex2.py
thread status: False
from child thread: 0
thread status: True
from child thread: 1
from child thread: 2
from child thread: 3
from child thread: 4
thread status: False

Process finished with exit code 0

```

Рисунок 5 – Пример 2

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  from threading import Thread
6  from time import sleep
7
8
9  class CustomThread(Thread):
10     def __init__(self, limit):
11         Thread.__init__(self)
12         self.limit_ = limit
13
14     def run(self):
15         for i in range(self.limit_):
16             print(f"from CustomThread: {i}")
17             sleep(0.5)
18
19 if __name__ == '__main__':
20     cth = CustomThread(3)
21     cth.start()
22
23

```

Run

```

C:\git\lab2.23\venv\Scripts\python.exe C:\git\lab2.23\examples\ex3.py
from CustomThread: 0
from CustomThread: 1
from CustomThread: 2

Process finished with exit code 0

```

Рисунок 6 – Пример 3

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  from threading import Thread, Lock
6  from time import sleep
7
8  lock = Lock()
9  stop_thread = False
10
11 def infinit_worker():
12     print("Start infinit_worker()")
13     while True:
14         print("--> thread work")
15         lock.acquire()
16         if stop_thread is True:
17             break
18         lock.release()
19         sleep(0.1)
20     print("Stop infinit_worker()")
21
22
23

```

Run

```

C:\git\lab2.23\venv\Scripts\python.exe C:\git\lab2.23\examples\ex3.py
from CustomThread: 0
from CustomThread: 1
from CustomThread: 2

Process finished with exit code 0

```

Рисунок 7 – Пример 4

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  from threading import Thread
6  from time import sleep
7
8
9  def func():
10     for i in range(5):
11         print(f"from child thread: {i}")
12         sleep(0.5)
13
14
15  if __name__ == '__main__':
16     th = Thread(target=func, daemon=True)
17     th.start()
18     print("\nApp stop")
19
```

Run

```
C:\git\lab2.23\venv\Scripts\python.exe C:\git\lab2.23\examples\ex5.py
from child thread: 0
App stop

Process finished with exit code 0
```

Рисунок 8 – Пример 5

### Индивидуальное задание.

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

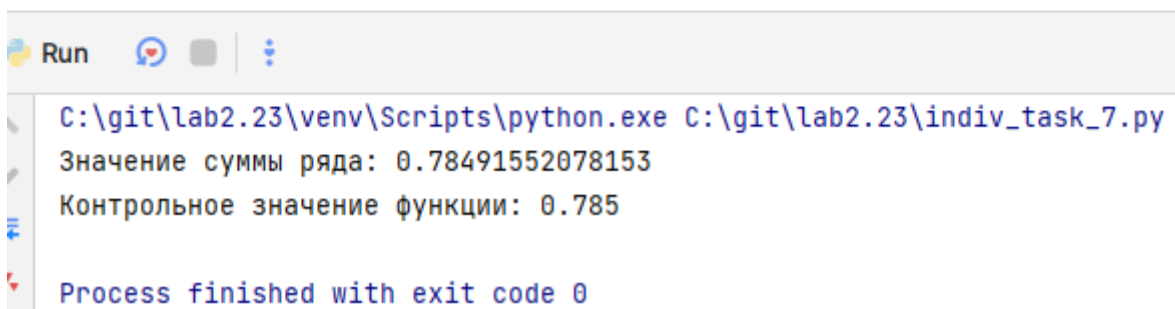
7.

$$S = \sum_{n=1}^{\infty} \frac{(-1)^{n+1} \sin nx}{n} = \sin x - \frac{\sin 2x}{2} + \dots; \quad x = -\frac{\pi}{2}; y = \frac{x}{2}.$$

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  from threading import Thread
5  from math import sin
6
7  EPS = 10**-7
8  PI = 3.14
9
10
11 def check_result(x):
12     y = x / 2
13     print(f"Контрольное значение функции: {y}")
14
15
16 def sum_inf(x):
17     n = 1
18     epsilon = 1e-7
19     s = (-1) ** (n + 1) * sin(n * x) / n
20     summ = s
21
22     while abs(s) > epsilon:
23         n += 1
24         s = (-1) ** (n + 1) * sin(n * x) / n
25         summ += s
26
27     print(f"Значение суммы ряда: {summ}")
28
29
30 def main():
31     x = PI / 2
32     th1 = Thread(target=sum_inf(x))
33     th2 = Thread(target=check_result(x))
34     th1.start()
35     th2.start()
36
37
38 if __name__ == "__main__":
39     main()
40

```



```

Run
C:\git\lab2.23\venv\Scripts\python.exe C:\git\lab2.23\indiv_task_7.py
Значение суммы ряда: 0.78491552078153
Контрольное значение функции: 0.785
Process finished with exit code 0

```

Рисунок 9 – Решение индивидуального задания

**Вывод:** Были приобретены и применены навыки написания многопоточных приложений на языке Python.

## **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?**

Параллельность предполагает параллельное выполнение задач разными исполнителями, например: один человек занимается готовкой, другой приборкой.

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.

### **3. Что такое GIL? Какое ограничение накладывает GIL?**

GIL — это аббревиатура от Global Interpreter Lock – глобальная блокировка интерпретатора. Он является элементом эталонной реализации языка Python, которая носит название CPython. Суть GIL заключается в том, что выполнять байт код может только один поток. Это нужно для того, чтобы упростить работу с памятью (на уровне интерпретатора) и сделать комфортной разработку модулей на языке C.

Пока выполняется одна задача, остальные простаивают (из-за GIL), переключение происходит через определенные промежутки времени. Таким образом, в каждый конкретный момент времени, будет выполняться только один поток, несмотря на то, что у вас может быть многоядерный процессор (или многопроцессорный сервер), плюс ко всему, будет тратиться время на переключение между задачами.



#### **4. Каково назначение класса Thread?**

Он отвечает за создание, управление и мониторинг потоков.

#### **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()`.

#### **6. Как проверить факт выполнения потоком некоторой работы?**

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

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

Для приостановки выполнения потока на некоторый промежуток времени в языке Python вы можете использовать функцию `time.sleep()`. Эта функция приостанавливает выполнение потока на указанное количество секунд.

#### **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.

#### **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.