

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ**

**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций
«Управление потоками в Python»**

**Отчет по лабораторной работе № 2.23
по дисциплине «Основы программной инженерии»**

Выполнил студент группы

ПИЖ-б-о-21-1

Трушева В. О. .« » 2023г.

Подпись студента _____

Работа защищена « _____ 20__ г.

Проверила Воронкин Р.А. _____

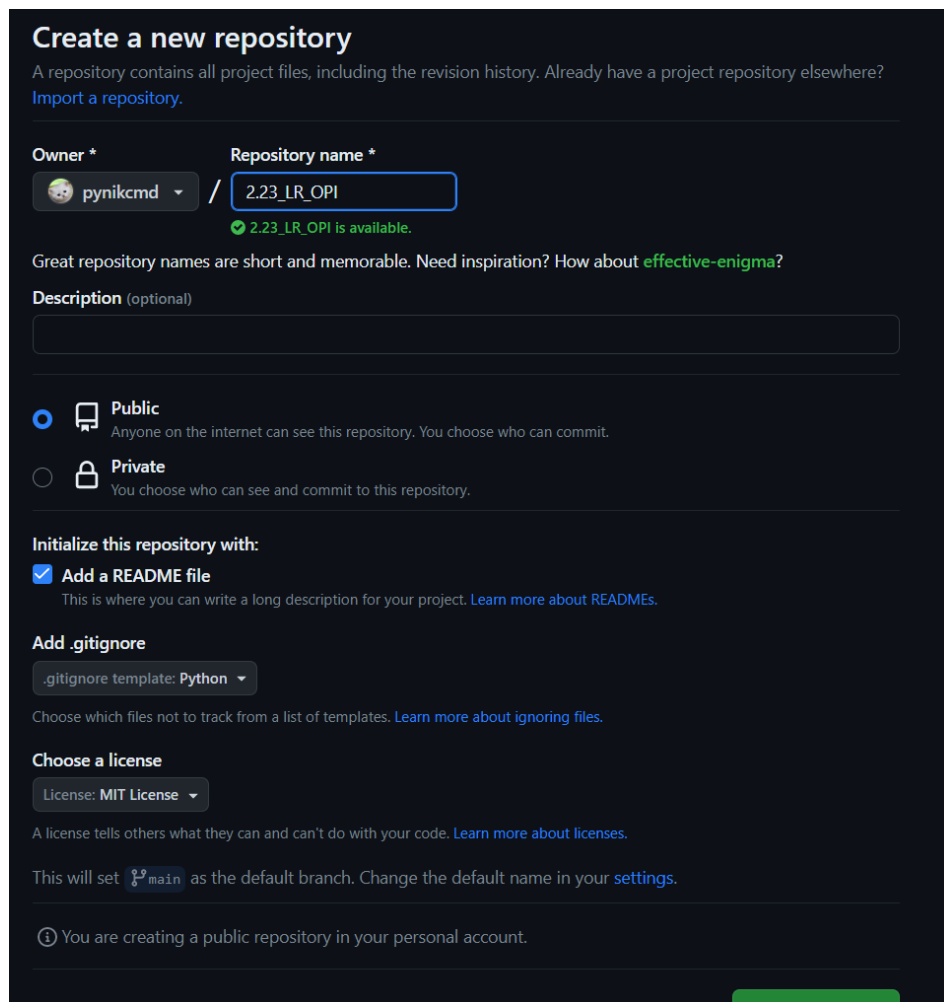
(подпись)

Ставрополь 2023

Цель работы: исследовать взаимодействие с базами данных SQLite3 с помощью языка программирования Python.

Методика и порядок выполнения работы

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




The screenshot shows the GitHub 'Create a new repository' page. At the top, it says 'Create a new repository' and provides a brief explanation of what a repository is. Below this, there are two main input fields: 'Owner *' and 'Repository name *'. The 'Owner' field is set to 'pynikcmd' and the 'Repository name' field is set to '2.23_LR_OPI'. A green checkmark indicates that the name is available. Below these fields, there is a 'Description (optional)' text area. Further down, there are two radio buttons for 'Public' and 'Private'. The 'Public' option is selected. Below this, there is a section 'Initialize this repository with:' with a checked box for 'Add a README file'. There is also a dropdown for '.gitignore template:' set to 'Python'. A 'Choose a license' section shows 'MIT License' selected. At the bottom, there is a green 'Create repository' button.

Create a new repository

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


Owner * / Repository name *


 pynikcmd / 2.23_LR_OPI

✔ 2.23_LR_OPI is available.

Great repository names are short and memorable. Need inspiration? How about [effective-enigma?](#)

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


 .gitignore template: Python


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

Choose a license

License: MIT License

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  main as the default branch. Change the default name in your [settings](#).


 You are creating a public repository in your personal account.

[Create repository](#)

3. Выполните клонирование созданного репозитория.

```
D:\fgit>git clone https://github.com/pynikcmd/2.23_LR_OPI.git
Cloning into '2.23_LR_OPI'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (8/8), 5.12 KiB | 5.12 MiB/s, done.
```

4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.

 .gitignore Update .gitignore

5. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.

```
D:\fgit\2.23_LR_OPI>git flow init
Which branch should be used for bringing forth production releases?
- main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [D:/fgit/2.23_LR_OPI/.git/hooks]
```

6. Создайте проект PyCharm в папке репозитория.

```
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     th.start()
18
19     for i in range(5):
20         print(f"from main thread: {i}")
21         sleep(1)
22
```

Run: Primer_1 x

D:\fgit\2.23_LR_0PI\Tasks\venv\Scripts\python.exe D:\fgit\2.23_LR_0PI\Tasks\venv\Scripts\python.exe D:\fgit\2.23_LR_0PI\Tasks\venv\Scripts\python.exe

from child thread: 0
from main thread: 0
from main thread: 1
from main thread: 2
from main thread: 3
from main thread: 4
from child thread: 1
from child thread: 2
from child thread: 3

Пример 1

```
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: Primer_2 x

D:\fgit\2.23_LR_0PI\Tasks\venv\Scripts\python.exe D:\fgit\2.23_LR_0PI\Tasks\venv\Scripts\python.exe

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

Пример 2

```
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
20 if __name__ == '__main__':
21     cth = CustomThread(3)
22     cth.start()
23
```

Run: Primer_3 x

D:\fgit\2.23_LR_OPI\Tasks\venv\Scripts\python.exe
from CustomThread: 0
from CustomThread: 1
from CustomThread: 2

Пример 3

```
13 def infinit_worker():
14     print("Start infinit_worker()")
15     while True:
16         print("--> thread work")
17         lock.acquire()
18         if stop_thread is True:
19             break
20         lock.release()
21         sleep(0.1)
22     print("Stop infinit_worker()")
23
24
25 if __name__ == '__main__':
26     # Create and start thread
27     th = Thread(target=infinit_worker)
28     th.start()
29     sleep(2)
30     # Stop thread
31     lock.acquire()
32     stop_thread = True
33     lock.release()
```

infinit_worker() > while True > if stop_thread is True

Run: Primer_4 x

D:\fgit\2.23_LR_OPI\Tasks\venv\Scripts\python.exe
Start infinit_worker()
--> thread work
--> thread work
--> thread work
--> thread work
--> thread work
--> thread work

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("App stop")
19
```

Run: Primer_5 x

from child thread: 0

App stop

Пример 5

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

Вариант – 2 (27). Условие

$$S = \sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + \dots; x = 0,7; y = \frac{1}{1-x}.$$

2.

```
6 def sum(x):
7     n = 1
8     sum = 1
9     temp = x
10    while abs(temp) >= EPSILON:
11        sum += temp
12        n += 1
13        temp *= x
14
15    print(f"Значение суммы для x={x}: {sum}")
16    print(f"Проверка: 1/(1 - {x}) = {1 / (1 - x)}")
17
18
19 def main():
20     x = 0.7
21
22     # Создаем потоки
23     thread1 = Thread(target=sum, args=(x,))
24     thread2 = Thread(target=sum, args=(-x,))
25
Run: Ind (1) x
D:\fgit\2.23_LR_OPI\Tasks\venv\Scripts\python.exe D:\fgit\2.
Значение суммы для x=0.7: 3.333333083650555
Проверка: 1/(1 - 0.7) = 3.333333333333333
Значение суммы для x=-0.7: 0.5882352500559803
Проверка: 1/(1 - -0.7) = 0.5882352941176471
```

Результат работы программы

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

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 4^2 и $2*4$ могут выполнять два разных процессора.

3. Что такое GIL? Какое ограничение накладывает GIL?

GIL (Global Interpreter Lock) – это механизм, используемый в некоторых интерпретаторах языков программирования, таких как Python, для синхронизации доступа к объектам в памяти между потоками исполнения.

GIL ограничивает возможность одновременного исполнения нескольких потоков в одном процессе Python. Это означает, что в любой момент времени только один поток может исполняться, а другие потоки будут ожидать, пока GIL не будет освобожден.

Ограничение, наложенное GIL, означает, что многопоточные приложения на Python не могут использовать полностью вычислительные мощности многоядерных процессоров. Несмотря на то, что GIL не является проблемой для однопоточных приложений или приложений,

которые не зависят от высокой производительности, это может быть существенным ограничением для приложений, которые требуют высокой степени параллелизма.

4. Каково назначение класса Thread?

Класс Thread - это класс в языке программирования Python, который предоставляет средства для создания и управления потоками исполнения в многопоточных приложениях. Назначение класса Thread заключается в том, чтобы предоставить программистам возможность создавать и запускать несколько потоков внутри одного процесса и управлять ими. Каждый поток исполнения выполняется в своей собственной последовательности инструкций и может обращаться к различным частям памяти, что позволяет реализовать параллельное выполнение задач.

Класс Thread обеспечивает управление потоками путем запуска их в фоновом режиме, ожидания их завершения, остановки, приостановки и возобновления их выполнения. Он также предоставляет методы для синхронизации выполнения потоков и обеспечения безопасности при работе с общими ресурсами, такими как глобальные переменные или файлы.

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

Для ожидания завершения другого потока в Python можно использовать метод `join()` класса Thread. Метод `join()` блокирует выполнение текущего потока до тех пор, пока поток, на который вызывается метод `join()`, не завершится.

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?

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

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?

Поток демона – это тип потока, который может работать независимо от фонового режиме. Эти типы потоков выполняются независимо от основного потока. Поэтому они называются неблокирующими потоками. Чтобы создать такой поток необходимо при создании объекта Thread аргументу daemon присвоить значение True, либо после создания потока, перед его запуском присвоить свойству daemon значение True.

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

Запустим ее, получим следующий результат:

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