

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ**

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

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

**Отчет по лабораторной работе № 2.25
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. _____

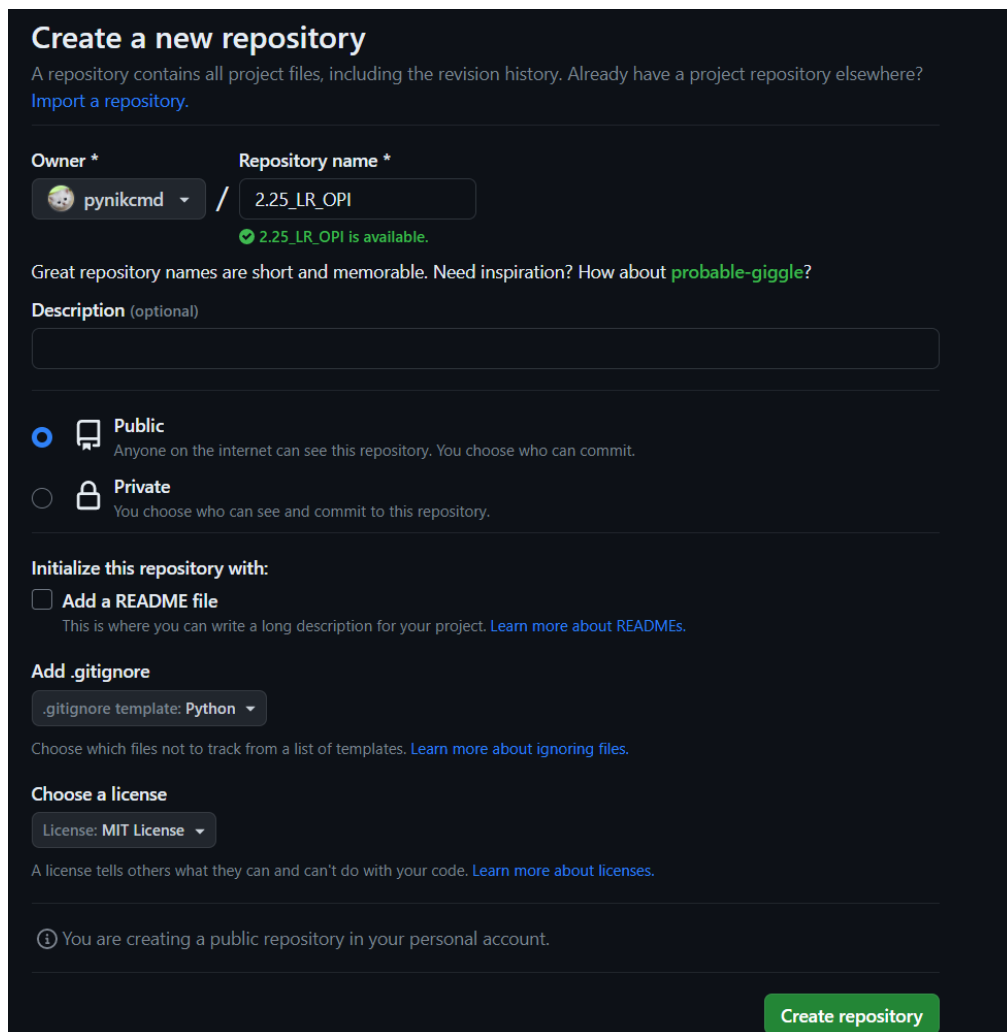
(подпись)

Ставрополь 2023

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

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' interface. 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 sections: 'Owner' and 'Repository name'. The 'Owner' is set to 'pynikcmd' and the 'Repository name' is '2.25_LR_OPI'. A green checkmark indicates that the name is available. Below these fields, there is a 'Description' field (optional) and a section for 'Initialize this repository with:'. In this section, the 'Add a README file' checkbox is checked, and the '.gitignore' template is set to 'Python'. The 'Choose a license' section shows 'MIT License' selected. At the bottom right, there is a green 'Create repository' button. A small information icon at the bottom left indicates that the user is creating a public repository in their personal account.

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 / 2.25_LR_OPI

✓ 2.25_LR_OPI is available.

Great repository names are short and memorable. Need inspiration? How about **probable-giggle?**

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.](#)

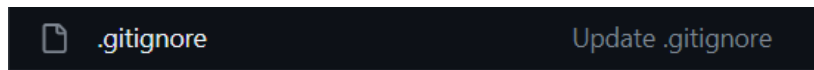
 You are creating a public repository in your personal account.

Create repository

3. Выполните клонирование созданного репозитория.

```
D:\fgit>git clone https://github.com/pynikcmd/2.25_LR_OPI.git
Cloning into '2.25_LR_OPI'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 10 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (10/10), 5.67 KiB | 830.00 KiB/s, done.
Resolving deltas: 100% (1/1), done.
```

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



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

```
D:\fgit\2.25_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/2.25_LR_OPI/.git/hooks]
```

6. Создайте проект PyCharm в папке репозитория.

7. Проработайте примеры лабораторной работы.

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

from multiprocessing import Process

def func():
    print("Hello from child Process")

if __name__ == "__main__":
    print("Hello from main Process")
    proc = Process(target=func)
    proc.start()

e_ == "__main__"
```

Primer_1 x

D:\fgit\2.25_LR_OPI\Tasks\venv\Scripts\python3
Hello from main Process
Hello from child Process

Пример 1

```
4
5     from multiprocessing import Process
6
7
8     def func():
9         print("Hello from child Process")
10
11
12     if __name__ == "__main__":
13         print("Hello from main Process")
14         proc = Process(target=func)
15         proc.start()
16         print(f"Proc is_alive status: {proc.is_alive()}")
17         proc.join()
18         print("Goodbye")
19         print(f"Proc is_alive status: {proc.is_alive()}")
20
21 if __name__ == "__main__":
```

Run: Primer_2 x

D:\fgit\2.25_LR_OPI\Tasks\venv\Scripts\python.exe D:\f
Hello from main Process
Proc is_alive status: True
Hello from child Process
Goodbye
Proc is_alive status: False

Пример 2

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

CustomProcess › run() › for i in range(self._limit)

Run: Primer_3 x

D:\fgit\2.25_LR_OPI\Tasks\venv\Scripts\python.
From CustomProcess: 0
From CustomProcess: 1
From CustomProcess: 2

Пример 3

```
from multiprocessing import Process
from time import sleep

def func():
    counter = 0
    while True:
        print(f"counter = {counter}")
        counter += 1
        sleep(0.1)

if __name__ == "__main__":
    proc = Process(target=func)
    proc.start()
    sleep(0.7)
    proc.terminate()
```

while True

Primer_4 x

D:\fgit\2.25_LR_OPI\Tasks\venv\Scripts\python.exe

counter = 0
counter = 1
counter = 2
counter = 3
counter = 4
counter = 5
counter = 6

Пример 4

```
from multiprocessing import Process
from time import sleep

def func(name):
    counter = 0
    while True:
        print(f"proc {name}, counter = {counter}")
        counter += 1
        sleep(0.1)

if __name__ == "__main__":
    proc1 = Process(target=func, args=("proc1",), daemon=True)
    proc2 = Process(target=func, args=("proc2",))
    proc2.daemon = True
    proc1.start()
    proc2.start()
    sleep(0.3)

__name__ == "__main__"
```

Primer_5 x

D:\fgit\2.25_LR_OPI\Tasks\venv\Scripts\python.exe D:\fgit\2.25_L

proc proc1, counter = 0

proc proc2, counter = 0

proc proc1, counter = 1

proc proc2, counter = 1

proc proc1, counter = 2

proc proc2, counter = 2

Пример 5

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

Вариант – 2 (27). Условие

2.

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


```

def sum(x):
    n = 1
    sum = 1
    temp = x
    while abs(temp) >= EPSILON:
        sum += temp
        n += 1
        temp *= x

    print(f"Значение суммы для x={x}: {sum}")
    print(f"Проверка: 1/(1 - {x}) = {1 / (1 - x)}")

def main():
    x = 0.7

    process1 = Process(target=sum, args=(x,))
    process1.start()
    process2 = Process(target=sum, args=(x,))
    process2.start()

```

e_ == "__main__"

Ind x

```

D:\fgit\2.25_LR_OPI\Tasks\venv\Scripts\python.exe D:\f
Значение суммы для x=0.7: 3.333333083650555
Проверка: 1/(1 - 0.7) = 3.333333333333333
Значение суммы для x=0.7: 3.333333083650555
Проверка: 1/(1 - 0.7) = 3.333333333333333

```

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. Как создаются и завершаются процессы в Python?

```
proc = Process(target=func)
```

```
proc.start()
```

`join()` для того, чтобы программа ожидала завершения процесса.

Процессы завершаются при завершении функции, указанной в `target`, либо принудительно с помощью `kill()`, `terminate()`

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.

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.

```
proc1 = Process(target=func, args=("proc1",), daemon=True)
proc2.daemon = True
proc1.start()
proc2.start()
```