

**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(12)
по дисциплине «Основы программной инженерии»**

Выполнил студент группы ПИЖ-б-о-21-1

Халимендик Я. Д. « » 2023г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2023

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

Ход работы:

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия IT 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.](#)

Owner * **Yana-Kh** / Repository name * **Lab-2.25-OPJ**

✔ Lab-2.25-OPJ is available.

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

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.](#)

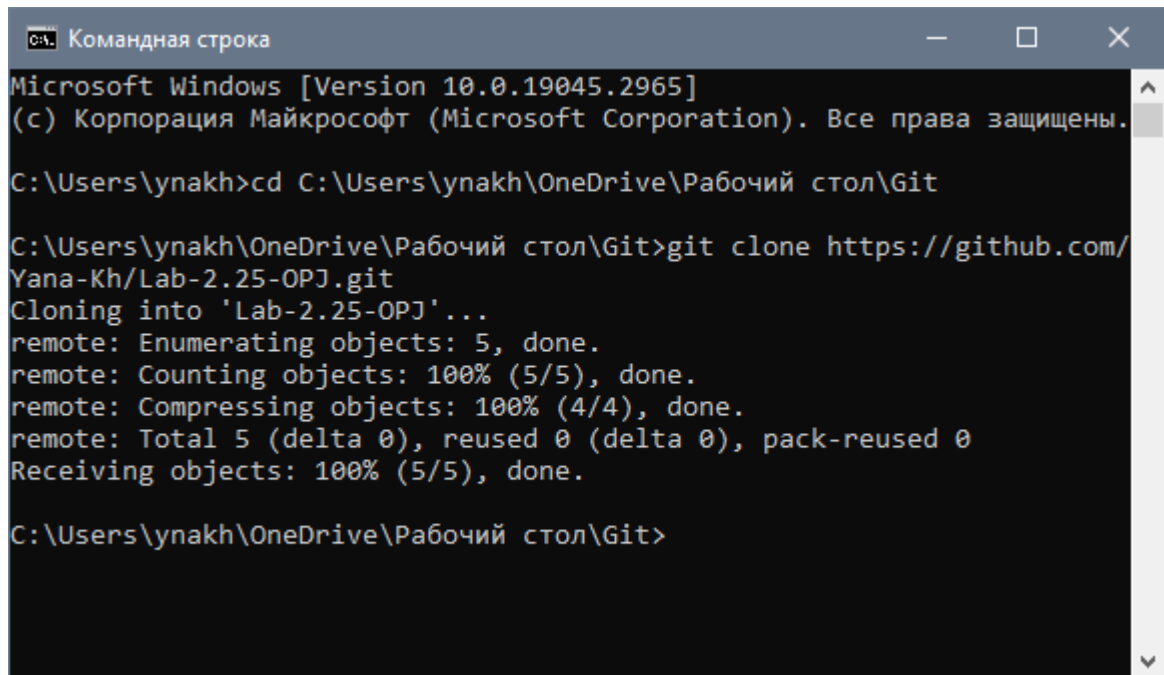
This will set **main** as the default branch. Change the default name in your [settings](#).

(i) You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание репозитория

3. Выполните клонирование созданного репозитория.



```
Командная строка
Microsoft Windows [Version 10.0.19045.2965]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

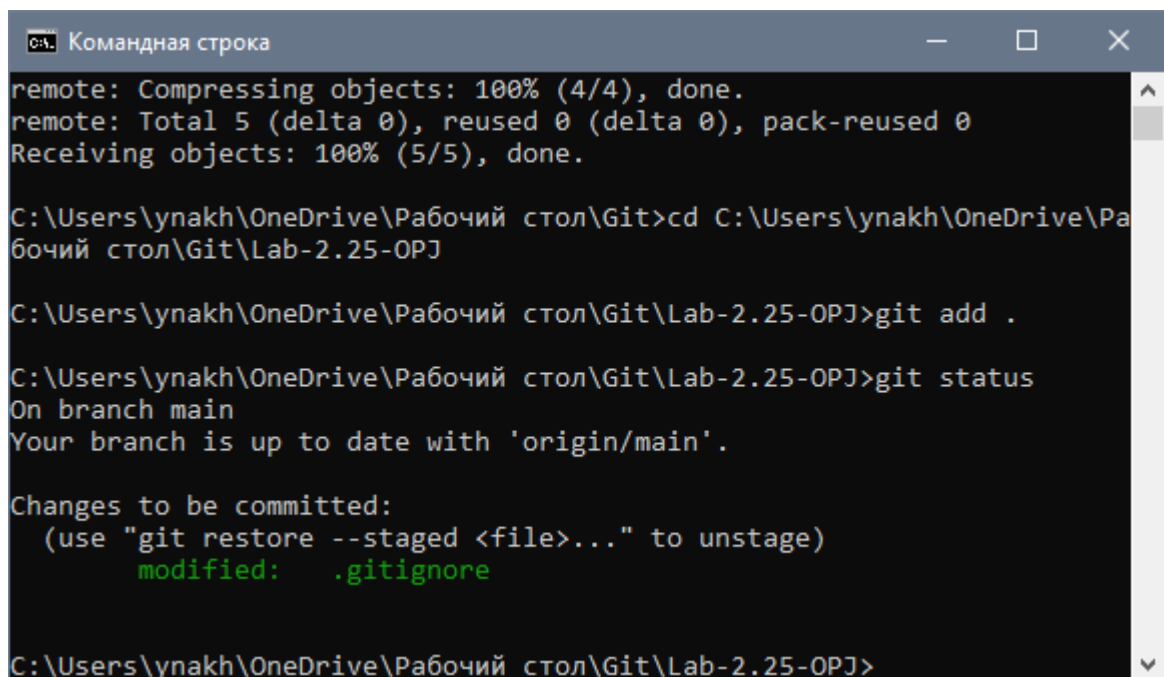
C:\Users\ynakh>cd C:\Users\ynakh\OneDrive\Рабочий стол\Git

C:\Users\ynakh\OneDrive\Рабочий стол\Git>git clone https://github.com/
Yana-Kh/Lab-2.25-OPJ.git
Cloning into 'Lab-2.25-OPJ'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.

C:\Users\ynakh\OneDrive\Рабочий стол\Git>
```

Рисунок 2 – Клонирование репозитория

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



```
Командная строка
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.

C:\Users\ynakh\OneDrive\Рабочий стол\Git>cd C:\Users\ynakh\OneDrive\Ра
бочий стол\Git\Lab-2.25-OPJ

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.25-OPJ>git add .

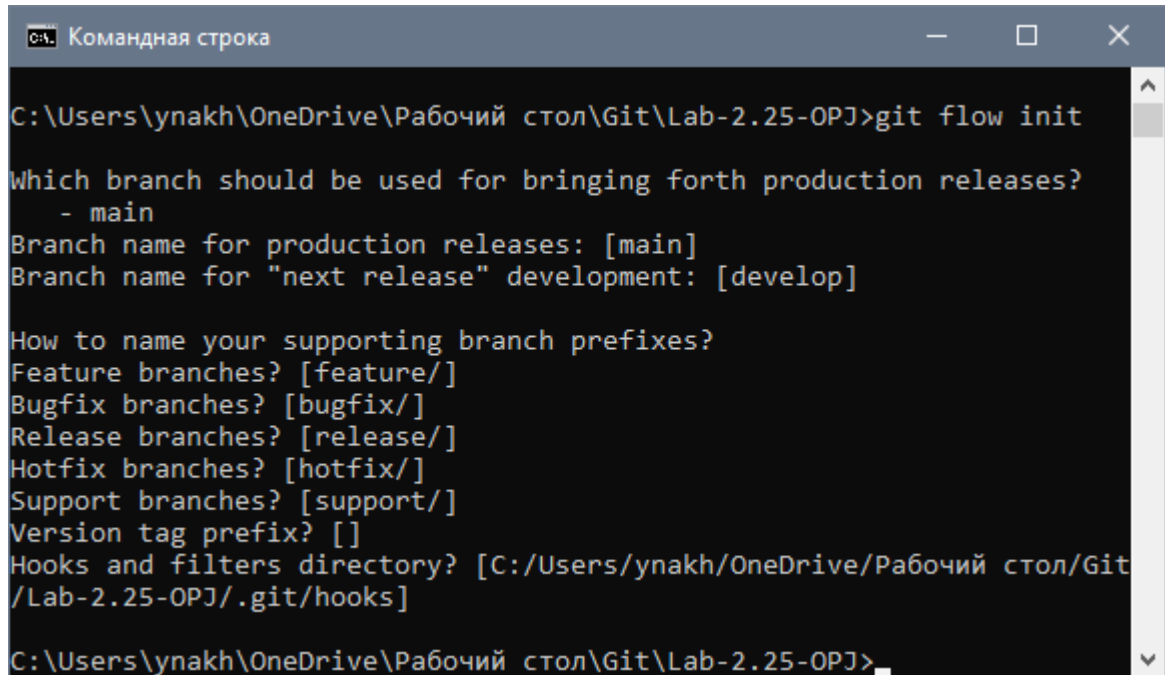
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.25-OPJ>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.25-OPJ>
```

Рисунок 3 – Дополнение файла .gitignore

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



```
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.25-OPJ>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? [C:/Users/ynakh/OneDrive/Рабочий стол/Git/Lab-2.25-OPJ/.git/hooks]

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.25-OPJ>
```

Рисунок 4 – Организация репозитория в соответствии с моделью git-flow

6. Создайте проект PyCharm в папке репозитория.

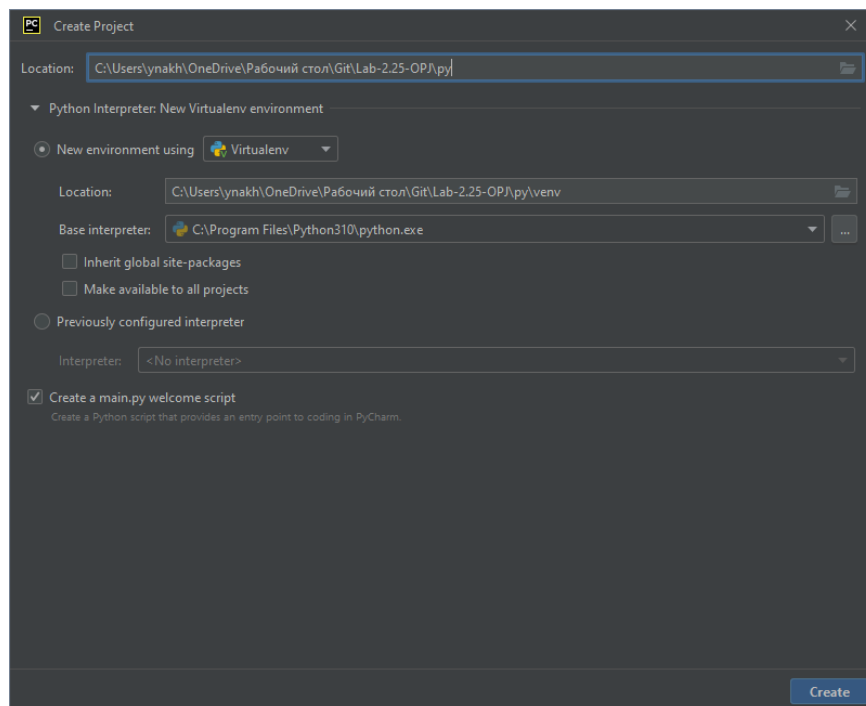


Рисунок 5 – Создание проекта PyCharm

7. Проработать примеры лабораторной работы. Создайте для них отдельные модули языка. Зафиксируйте изменения в репозитории.

Пример 1.

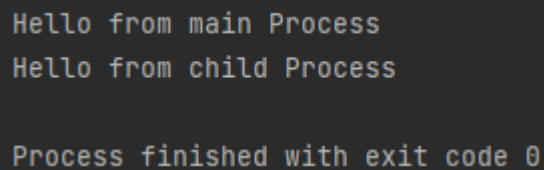
Код:

```
#!/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()
```



```
Hello from main Process
Hello from child Process

Process finished with exit code 0
```

Рисунок 6 – Результат работы программы

Пример 2.

Код:

```
#!/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()
    print(f"Proc is alive status: {proc.is_alive()}")
    proc.join()
    print("Goodbye")
    print(f"Proc is alive status: {proc.is_alive()}")
```

```
Hello from main Process
Proc is_alive status: True
Hello from child Process
Goodbye
Proc is_alive status: False

Process finished with exit code 0
```

Рисунок 7 – Результат работы программы

Пример 3.

Код:

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

from multiprocessing import Process
from time import sleep

class CustomProcess(Process):
    def __init__(self, limit):
        Process.__init__(self)
        self._limit = limit

    def run(self):
        for i in range(self._limit):
            print(f"From CustomProcess: {i}")
            sleep(0.5)

if __name__ == "__main__":
    cpr = CustomProcess(3)
    cpr.start()
```

```
From CustomProcess: 0
From CustomProcess: 1
From CustomProcess: 2

Process finished with exit code 0
```

Рисунок 8 – Результат работы программы

Пример 4.

Код:

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

```

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()

```

```

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

Process finished with exit code 0

```

Рисунок 9 – Результат работы программы

Пример 5.

Код:

```

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

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)

```

```

proc proc1, counter = 0
proc proc2, counter = 0
proc proc1, counter = 1proc proc2, counter = 1

proc proc1, counter = 2proc proc2, counter = 2
|

Process finished with exit code 0

```

Рисунок 10 – Результат работы программы

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

Код:

```

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

from multiprocessing import Process
from time import sleep

def check_sum(x):
    y = (5-2*x) / (6 - 5*x + x**2)
    print(f"The result check-function is: {y}")

def s_sum(x):
    n = 1
    term = (1 / (2 ** n) + 1 / (3 ** n)) * (x ** (n - 1))
    epsilon = 1e-7
    ssum = term

    while abs(term) >= epsilon:
        n += 1
        term = (1 / (2 ** n) + 1 / (3 ** n)) * (x ** (n - 1))
        ssum += term

    print(f"The sum S is: {ssum}")

def main():
    x = -0.8

    proc1 = Process(target=check_sum, args=(x,))
    proc2 = Process(target=s_sum, args=(x,))

    proc1.start()
    proc2.start()

    sleep(0.1)

    proc1.terminate()

```



```
proc2.terminate()

if __name__ == "__main__":
    main()
```

```
The result check-function is: 0.6203007518796991
The sum S is: 0.6203007273247924

Process finished with exit code 0
```

Рисунок 11 – Результат работы программы

9. Зафиксируйте сделанные изменения в репозитории.







 ex1.py	ex
 ex2.py	ex
 ex3.py	ex
 ex4.py	ex
 ex5.py	ex
 id.py	id

Рисунок 12 – Фиксирование изменений в репозитории

Вопросы для защиты работы:

1. Как создаются и завершаются процессы в Python?

Классом, который отвечает за создание и управление процессами является Process из пакета multiprocessing. Для запуска процесса используется метод start(), а за ожидание завершения работы процесса(ов) отвечает метод join(). При выводе метода join() выполнение программы будет остановлено до тех пор пока соответствующий процесс не завершит работу. Параметр timeout отвечает за время ожидания завершения работы процесса, если указанное

время прошло, а процесс еще не завершился, то ожидание будет прервано и выполнение программы продолжится дальше.

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. Указать на то, что процесс является демоном можно при создании экземпляра класса через аргумент `daemon`, либо после создания через свойство `daemon`