

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

Институт цифрового развития
Кафедра инфокоммуникаций

«Синхронизация потоков в языке программирования Python»

ОТЧЕТ
по лабораторной работе №25
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)

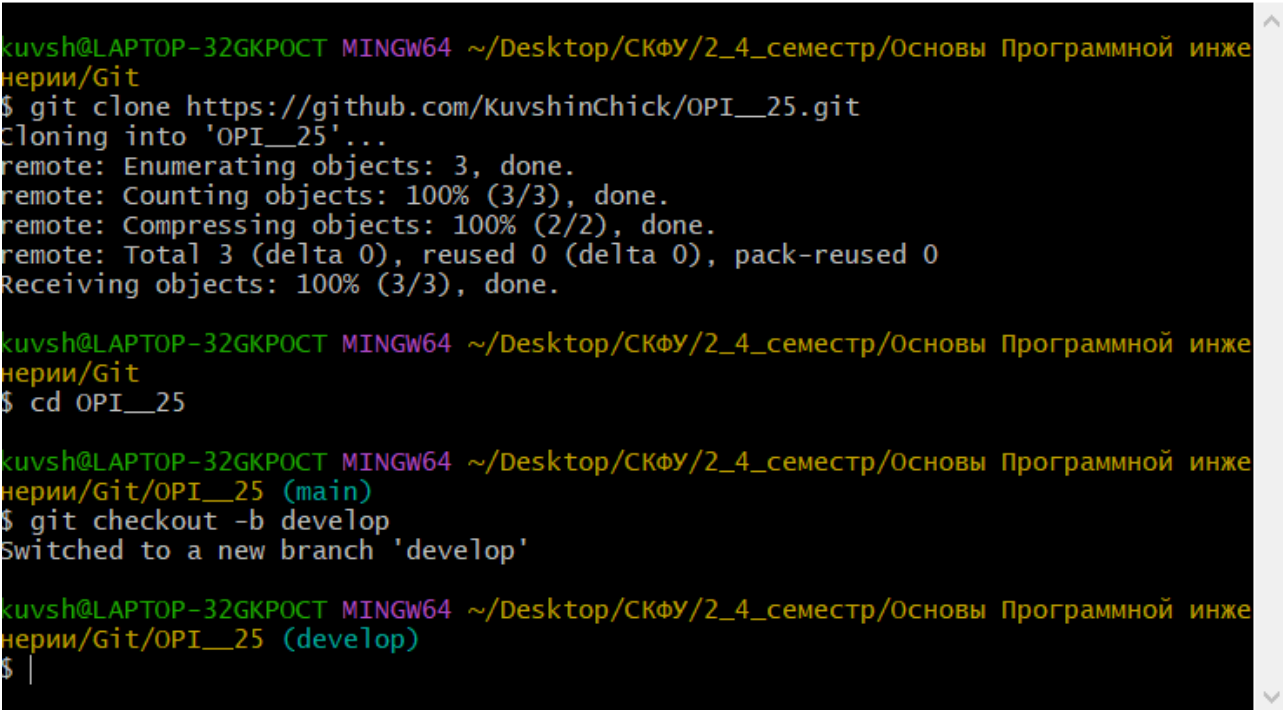
Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

Цель работы: приобретение навыков написания многозадачных 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.

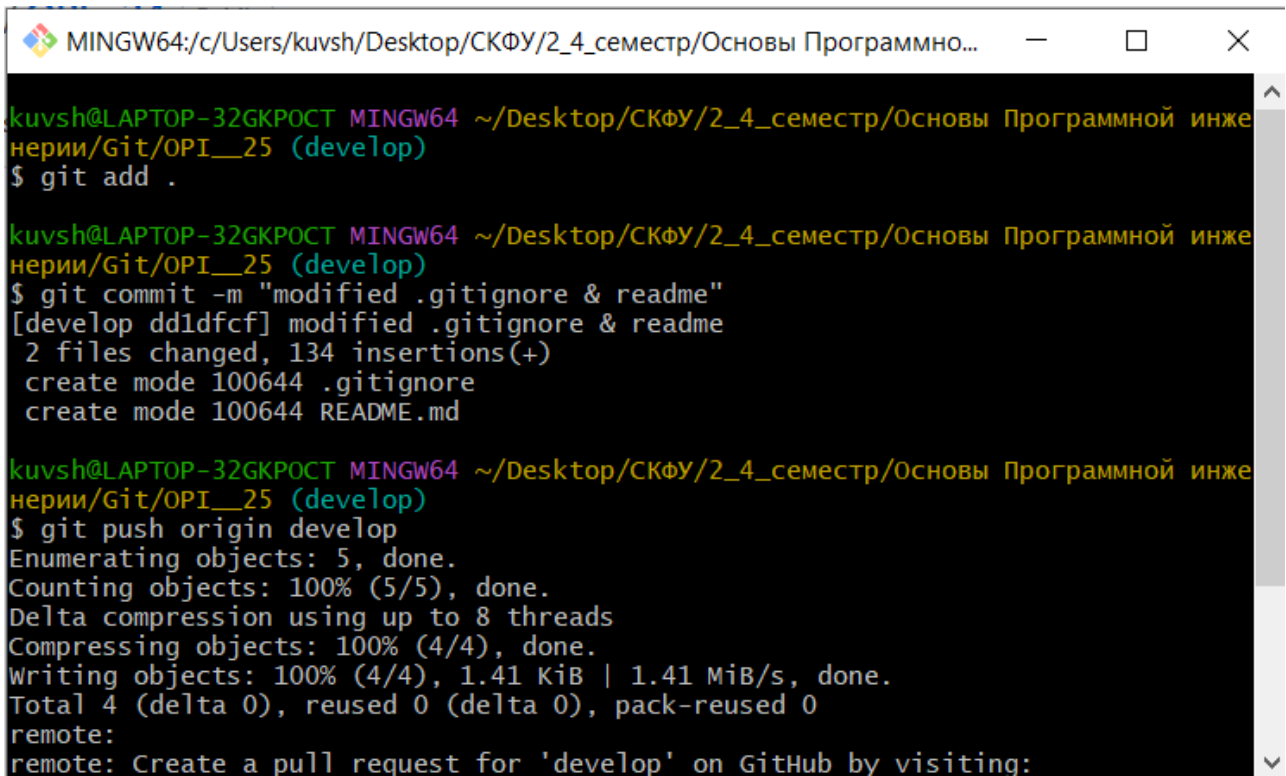


```
MINGW64:/c:/Users/kuvsh/Desktop/СКФУ/2_4_семестр/Основы Программно...
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программно...
$ git clone https://github.com/KuvshinChick/OPI__25.git
Cloning into 'OPI__25'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (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 Программно...
$ cd OPI__25

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программно...
$ git checkout -b develop
Switched to a new branch 'develop'
```

Рисунок 25.1 – Клонирование репозитория и создание ветки develop



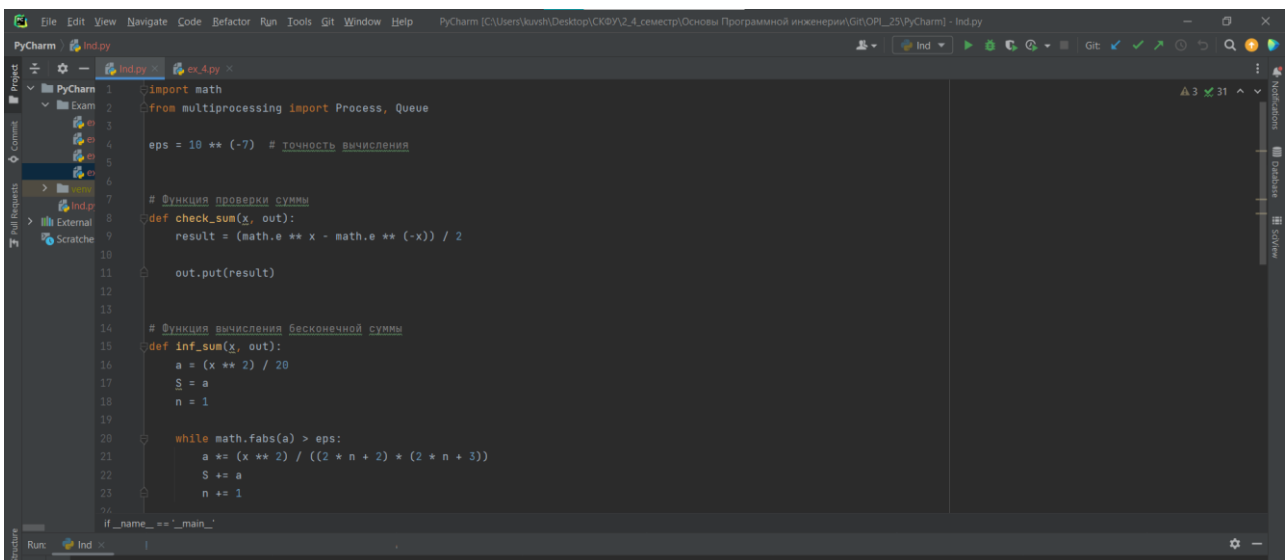
```
MINGW64:/c/Users/kuvsh/Desktop/CKФУ/2_4_семестр/Основы Программно...
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/CKФУ/2_4_семестр/Основы Программно...
нерии/Git/OPI__25 (develop)
$ git add .

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/CKФУ/2_4_семестр/Основы Программно...
нерии/Git/OPI__25 (develop)
$ git commit -m "modified .gitignore & readme"
[develop dd1dfcf] modified .gitignore & readme
2 files changed, 134 insertions(+)
create mode 100644 .gitignore
create mode 100644 README.md

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/CKФУ/2_4_семестр/Основы Программно...
нерии/Git/OPI__25 (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:
```

Рисунок 25.2 – Обновление .gitignore и readme

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



```
PyCharm [C:/Users/kuvsh/Desktop/CKФУ/2_4_семестр/Основы Программно.../Git/OPI__25/PyCharm] - Ind.py
File Edit View Navigate Code Refactor Run Tools Git Window Help
Project: PyCharm
- Ind.py x
- ex.4.py x
1 import math
2 from multiprocessing import Process, Queue
3
4 eps = 10 ** (-7) # ТОЧНОСТЬ ВЫЧИСЛЕНИЯ
5
6
7 # Функция проверки суммы
8 def check_sum(x, out):
9     result = (math.e ** x - math.e ** (-x)) / 2
10
11     out.put(result)
12
13
14 # Функция вычисления бесконечной суммы
15 def inf_sum(x, out):
16     a = (x ** 2) / 20
17     S = a
18     n = 1
19
20     while math.fabs(a) > eps:
21         a = (x ** 2) / ((2 * n + 2) * (2 * n + 3))
22         S += a
23         n += 1
24
25 if __name__ == '__main__':
26     pass
```

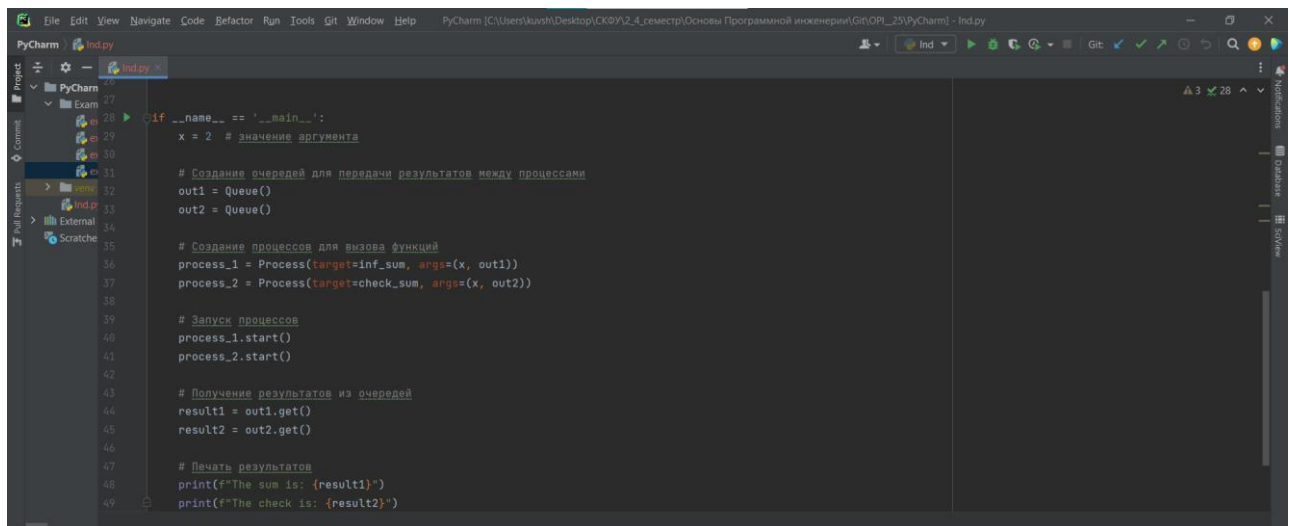


Рисунок 25.3 – Проработка программы

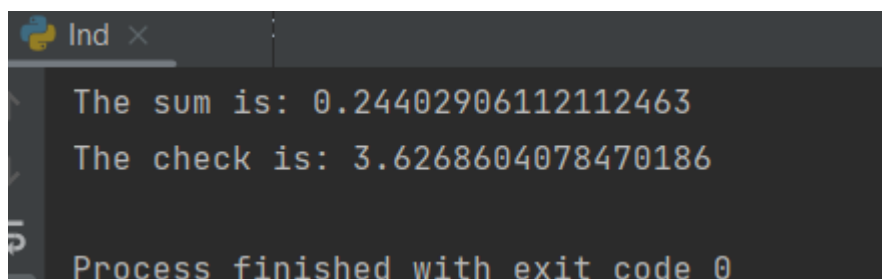


Рисунок 25.4 – Результат работы программы

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. Как создаются и завершаются процессы в Python?

Процессы в Python создаются с помощью модуля multiprocessing. Для создания процесса необходимо создать объект класса Process, передав в конструктор функцию или метод класса, который будет выполняться в процессе. Для запуска процесса используется метод start(). Завершение процесса происходит автоматически при завершении выполнения функции или метода, переданного в конструктор Process.

2. В чем особенность создания классов-наследников от Process?

Особенность создания классов-наследников от `Process` заключается в том, что в методе `run()` необходимо указать код, который будет выполняться в процессе. Этот метод вызывается при запуске процесса. Также класс-наследник должен иметь свой конструктор, в котором нужно вызвать конструктор родительского класса и передать в него аргументы.

3. Как выполнить принудительное завершение процесса?

Для принудительного завершения процесса в Python можно использовать метод `terminate()`, который отправляет процессу сигнал `SIGTERM`. Обработка этого сигнала внутри процесса приводит к его завершению. Однако, не рекомендуется использовать `terminate()` без необходимости, так как это может привести к утечкам ресурсов.

4. Что такое процессы-демоны? Как запустить процесс-демон?

Процессы-демоны - это процессы, которые работают в фоновом режиме, не взаимодействуя с пользователем. Они не имеют своего терминала и не могут взаимодействовать с консолью. Запустить процесс-демон можно, используя методы из модуля `daemonize`. Кроме того, при создании процесса можно указать параметр `daemon=True`, что превращает процесс в демон.