

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

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №12
дисциплины «Основы программной инженерии»

Выполнил:
Звездин Алексей Сергеевич
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Руководитель практики:
Воронкин Р. А., доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

Ход работы

1. Я изучил теоретический материал работы

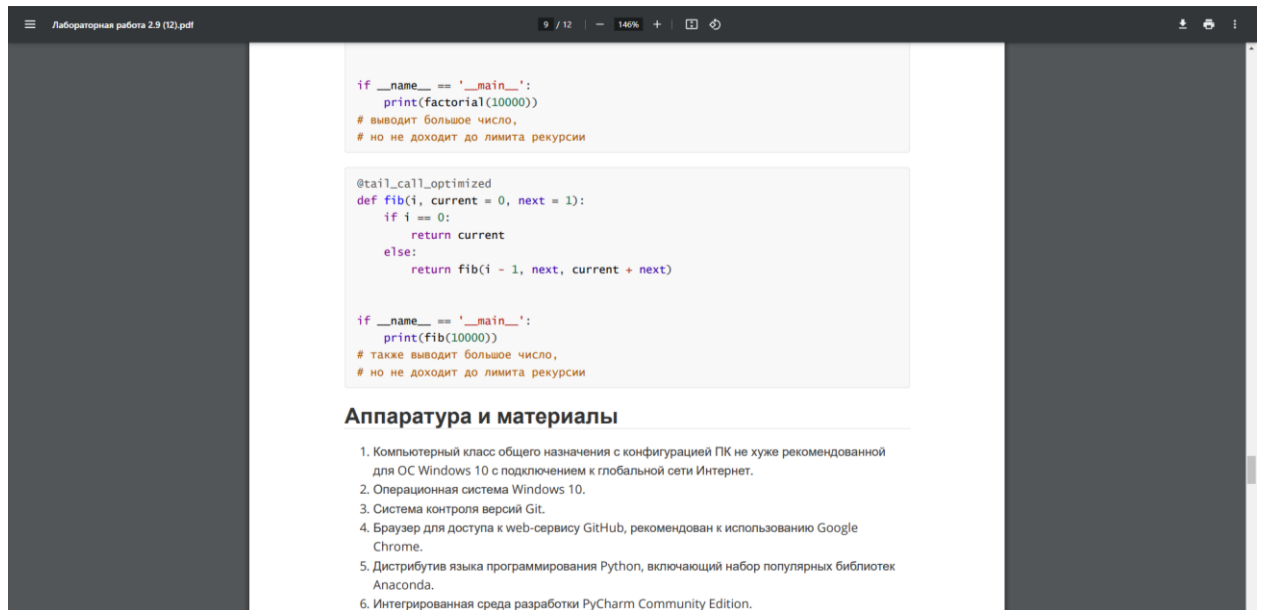


Рисунок 1.1 – Изучение материала для лабораторной работы

2. Создал общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT 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.](#)

Required fields are marked with an asterisk (*).

Owner * InternetHacker1123 / Repository name * laba_2_9(12)

ⓘ Your new repository will be created as **laba_2_9-12-**.
The repository name can only contain ASCII letters, digits, and the characters `.`, `-`, and `_`.

Great repository names are short and memorable. Need inspiration? How about [super-duper-octo-journey](#)?

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](#)

Рисунок 2.1 – Настройка репозитория

laba_2_9-12- Public

Pin Unwatch 1 Fork Star 0

main 1 Branch 0 Tags Go to file Add file Code

InternetHacker1123 Initial commit 9680d2d · now 1 Commits

.gitignore Initial commit now

LICENSE Initial commit now

README.md Initial commit now

README License

laba_2_9-12-

About

No description, website, or topics provided.

Readme

Activity

0 stars

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

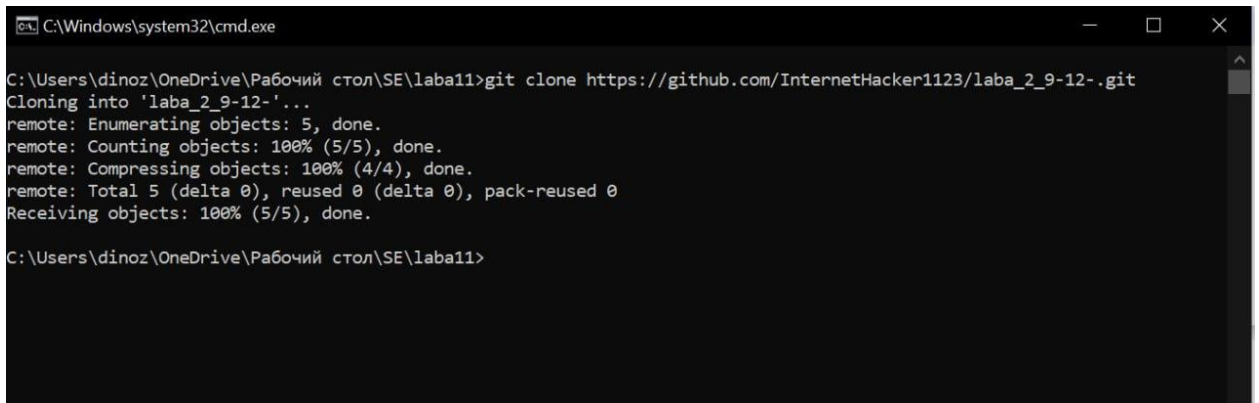
No packages published

Publish your first package

© 2023 GitHub, Inc. Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information

Рисунок 2.2 – Готовый репозиторий

3. Выполняю клонирование созданного репозитория



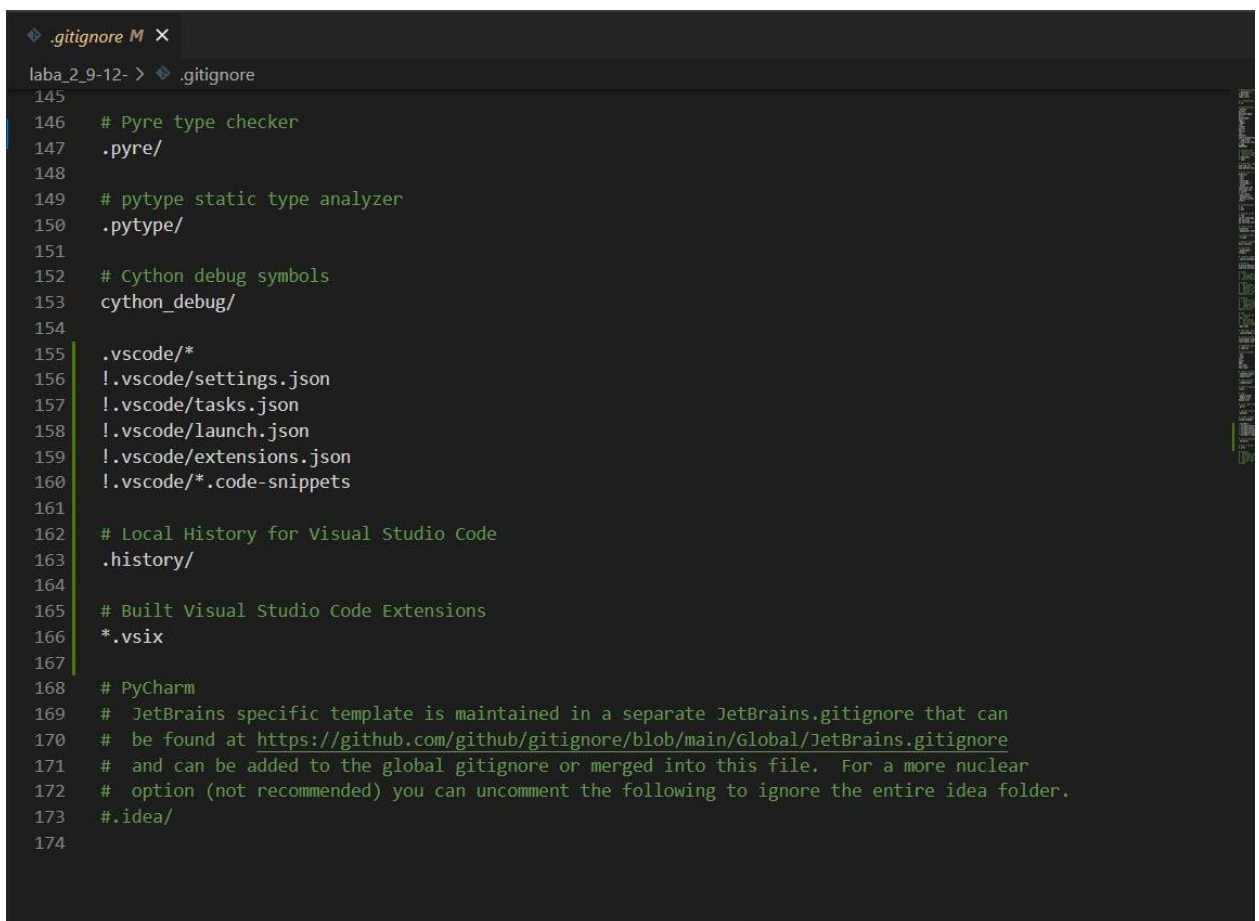
```
C:\Windows\system32\cmd.exe

C:\Users\dinoz\OneDrive\Рабочий стол\SE\laba11>git clone https://github.com/InternetHacker1123/laba_2_9-12-.git
Cloning into 'laba_2_9-12-'...
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\dinoz\OneDrive\Рабочий стол\SE\laba11>
```

Рисунок 3.1 – Клонирование репозитория на локальный диск

4. Дополнил файл .gitignore необходимыми правилами для работы с VS Code



```
.gitignore M x
laba_2_9-12- > .gitignore
145
146 # Pyre type checker
147 .pyre/
148
149 # pytype static type analyzer
150 .pytype/
151
152 # Cython debug symbols
153 cython_debug/
154
155 .vscode/*
156 !.vscode/settings.json
157 !.vscode/tasks.json
158 !.vscode/launch.json
159 !.vscode/extensions.json
160 !.vscode/*.code-snippets
161
162 # Local History for Visual Studio Code
163 .history/
164
165 # Built Visual Studio Code Extensions
166 *.vsix
167
168 # PyCharm
169 # JetBrains specific template is maintained in a separate JetBrains.gitignore that can
170 # be found at https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore
171 # and can be added to the global gitignore or merged into this file. For a more nuclear
172 # option (not recommended) you can uncomment the following to ignore the entire idea folder.
173 #.idea/
174
```

Рисунок 4.1 – .gitignore для VS Code

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

```
C:\Windows\system32\cmd.exe

C:\Users\dinoz\OneDrive\Рабочий стол\SE\laba11\laba_2_9-12->git branch develop
C:\Users\dinoz\OneDrive\Рабочий стол\SE\laba11\laba_2_9-12->git checkout develop
Switched to branch 'develop'

C:\Users\dinoz\OneDrive\Рабочий стол\SE\laba11\laba_2_9-12->git branch
* develop
  main

C:\Users\dinoz\OneDrive\Рабочий стол\SE\laba11\laba_2_9-12->git push -u origin develop
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:   https://github.com/InternetHacker1123/laba_2_9-12-/pull/new/develop
remote:
To https://github.com/InternetHacker1123/laba_2_9-12-.git
 * [new branch]      develop -> develop
branch 'develop' set up to track 'origin/develop'.
```

Рисунок 5.1 – Создание ветки develop от ветки main

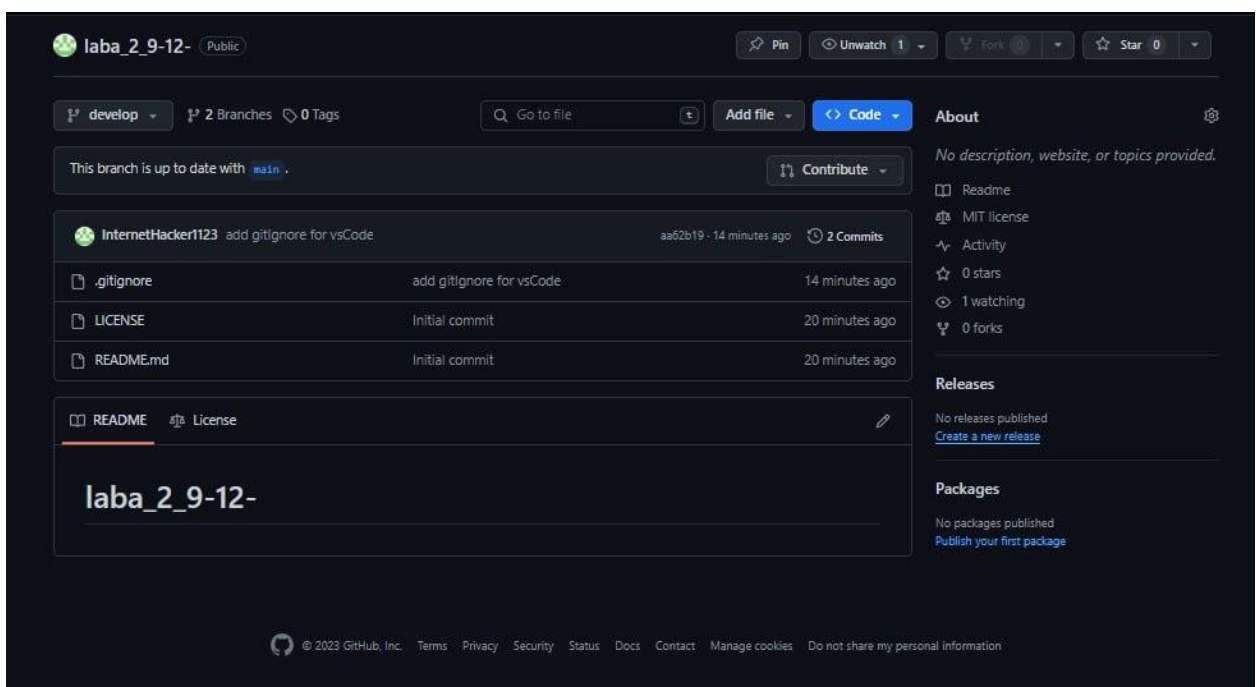


Рисунок 5.2 – Ветка develop на GitHub

6. Создал проект PyCharm в папке репозитория

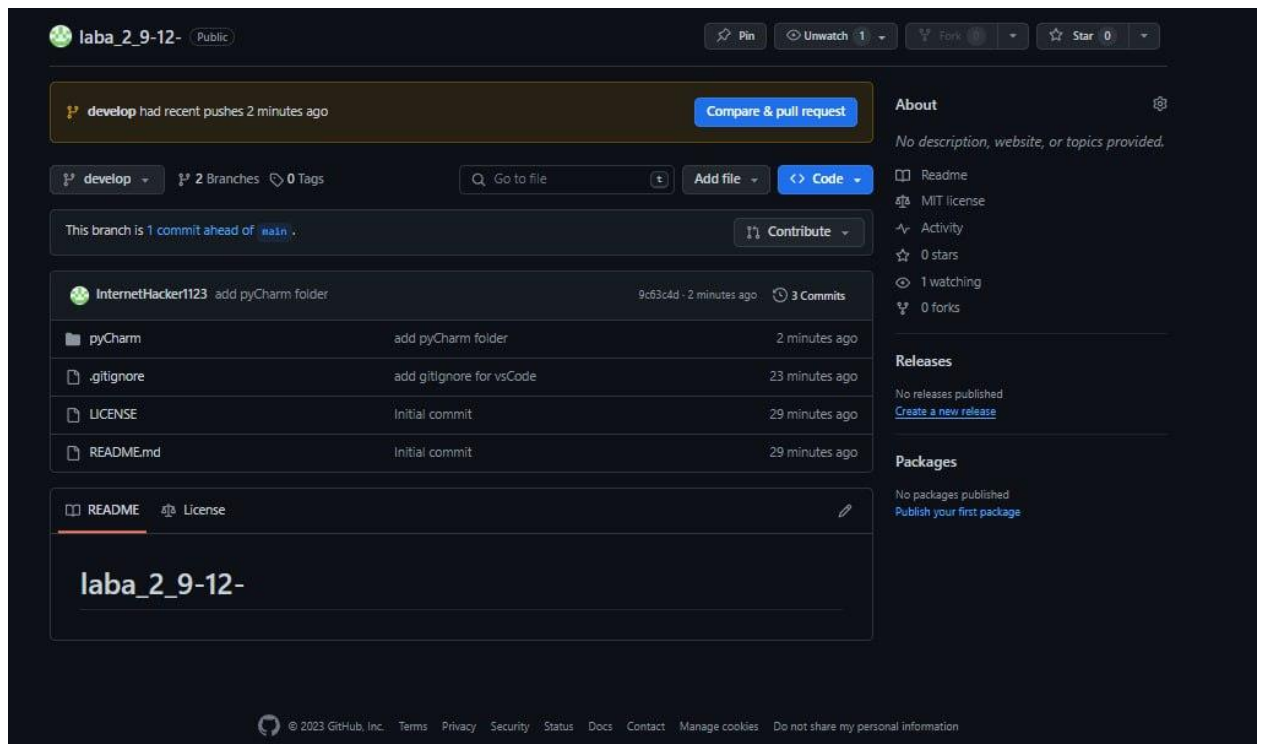


Рисунок 6.1 – Репозиторий с проектом PyCharm

7. Самостоятельно изучил работу со стандартным пакетом Python `timeit`. Оценил с помощью этого модуля скорость работы итеративной и рекурсивной версий функций `factorial` и `fib`.

```

laba_2_9-12- > pyCharm > example1.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  from functools import lru_cache
5
6  # Выполнение функций без использования декоратора.
7
8  def factorial_iterable(n):
9      # Итеративная версия функции factorial.
10     multiply = 1
11     while n > 1:
12         multiply *= n
13         n -= 1
14     return multiply
15
16
17 def fib_iterable(n):
18     # Итеративная версия функции fib.
19     a, b = 0, 1
20     while n > 0:
21         a, b = b, a + b
22         n -= 1
23     return a
24
25
26 # Выполнение функций с использованием декоратора.
27
28 @lru_cache
29 def factorial_recursion(n):
30     # Рекурсивная версия функции factorial.
31     if n == 0:
32         return 1
33     elif n == 1:
34         return 1
35     else:
36         return n * factorial_recursion(n - 1)
37
38
39 @lru_cache
40 def fib_recursion(n):
41     # Рекурсивная версия функции fib.
42     if n == 0 or n == 1:
43         return n
44     else:
45         return (fib_recursion(n - 2) +
46                 fib_recursion(n - 1))
47
48
49

```

Рисунок 7.1 – Код программы example1.py

```
C:\Windows\system32\cmd.exe

C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>python -m timeit -r 5 -n 100 -u sec -s "from example1 import factorial_iterable" "factorial_iterable(1000)"
100 loops, best of 5: 0.00027 sec per loop

C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>
```

Рисунок 7.2 – Результат замера времени выполнения

```
C:\Windows\system32\cmd.exe

C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>python -m timeit -r 5 -n 100 -u sec -s "from example1 import fib_iterable" "fib_iterable(1000)"
100 loops, best of 5: 6.52e-05 sec per loop

C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>
```

Рисунок 7.3 – Результат замера времени выполнения

```
C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>python -m timeit -r 5 -n 100 -u sec -s "from example1 import factorial_recursion" "factorial_recursion(100)"
100 loops, best of 5: 6.7e-08 sec per loop
:0: UserWarning: The test results are likely unreliable. The worst time (1.29e-06 sec) was more than four times slower than the best time (6.7e-08 sec).

C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>
```

Рисунок 7.4 – Результат замера времени выполнения

```
C:\Windows\system32\cmd.exe

C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>python -m timeit -r 5 -n 100 -u sec -s "from example1 import fib_recursion" "fib_recursion(100)"
100 loops, best of 5: 6.7e-08 sec per loop
:0: UserWarning: The test results are likely unreliable. The worst time (8.3e-07 sec) was more than four times slower than the best time (6.7e-08 sec).

C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>
```

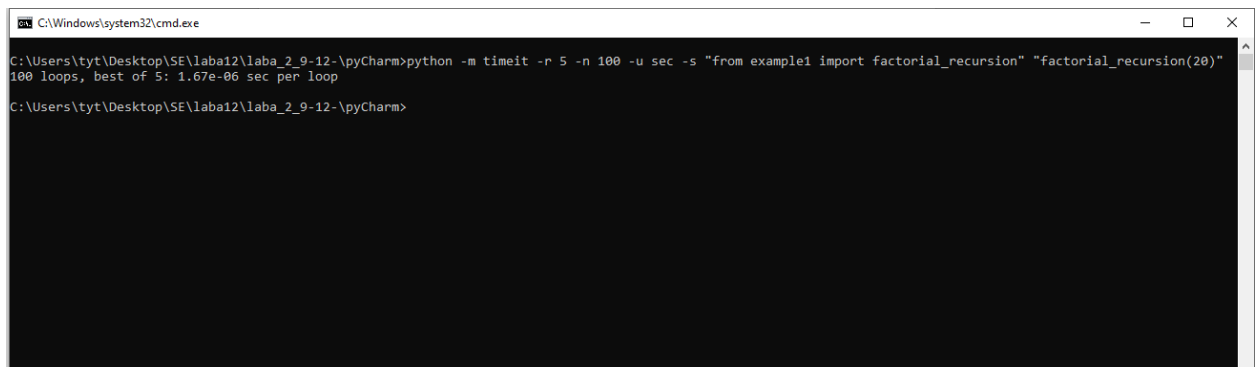
Рисунок 7.5 – Результат замера времени выполнения

```
C:\Windows\system32\cmd.exe

C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>python -m timeit -r 5 -n 100 -u sec -s "from example1 import fib_recursion" "fib_recursion(20)"
100 loops, best of 5: 0.00158 sec per loop

C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>
```

Рисунок 7.6 – Результат замера времени выполнения



```
C:\Windows\system32\cmd.exe
C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>python -m timeit -r 5 -n 100 -u sec -s "from example1 import factorial_recursion" "factorial_recursion(20)"
100 loops, best of 5: 1.67e-06 sec per loop
C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>
```

Рисунок 7.7 – Результат замера времени выполнения

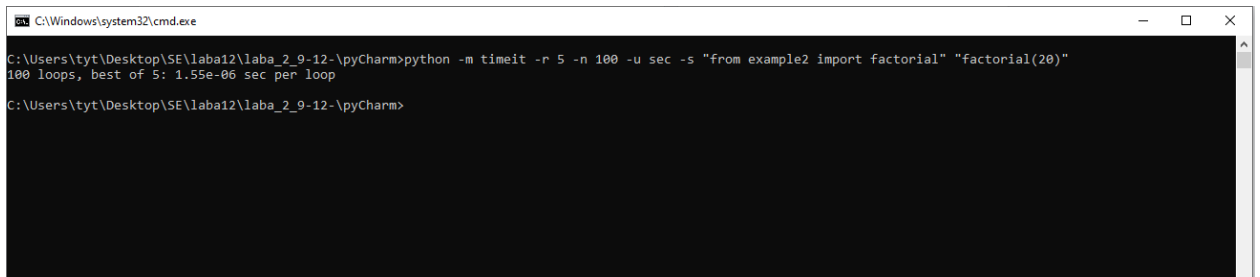
8. Самостоятельно проработал пример с оптимизацией хвостовых вызовов в Python. С помощью пакета timeit оценил скорость работы функций factorial и fib с использованием интроспекции стека и без использования интроспекции стека. Привел полученные результаты в отчет

```

laba_2_9-12- > pyCharm > example2.py > fib
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  from timeit import timeit
5
6  # Выполнение функций без использования интроспекции стека.
7
8
9  def factorial(n):
10     # Функция для вычисления факториала.
11     if n == 0:
12         return 1
13     else:
14         return n * factorial(n - 1)
15
16
17  def fib(n):
18     # Функция для чисел фибоначчи.
19     if n <= 1:
20         return n
21     else:
22         return fib(n - 1) + fib(n - 2)
23
24  # Выполнение функций с использованием интроспекции стека.
25
26
27  def factorial_tail(n, acc=1):
28     if n == 0:
29         return acc
30     else:
31         return factorial_tail(n - 1, acc * n)
32
33
34  def fib_tail(n, a=0, b=1):
35     if n == 0:
36         return a
37     else:
38         return fib_tail(n - 1, b, a + b)
39

```

Рисунок 8.1 – Код программы example2.py в IDE PyCharm

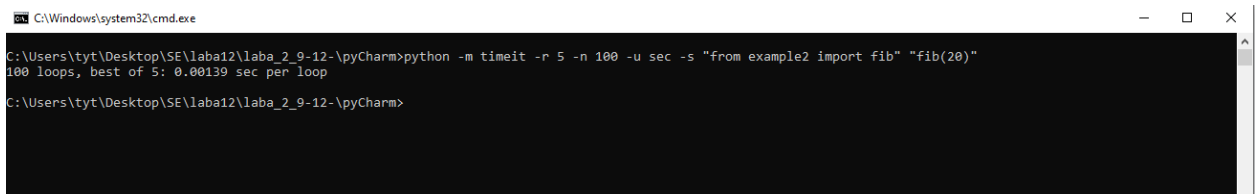


```
C:\Windows\system32\cmd.exe

C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>python -m timeit -r 5 -n 100 -u sec -s "from example2 import factorial" "factorial(20)"
100 loops, best of 5: 1.55e-06 sec per loop

C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>
```

Рисунок 8.2 – Результат замера времени выполнения

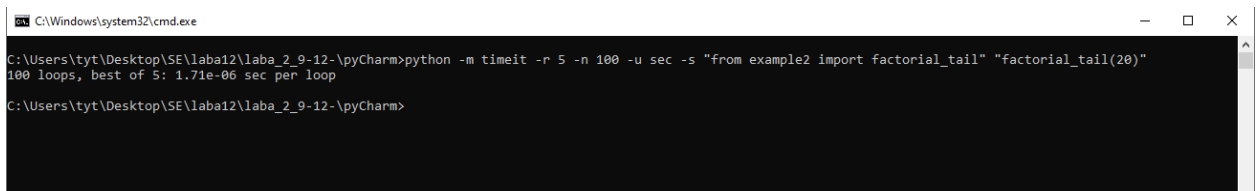


```
C:\Windows\system32\cmd.exe

C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>python -m timeit -r 5 -n 100 -u sec -s "from example2 import fib" "fib(20)"
100 loops, best of 5: 0.00139 sec per loop

C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>
```

Рисунок 8.3 – Результат замера времени выполнения

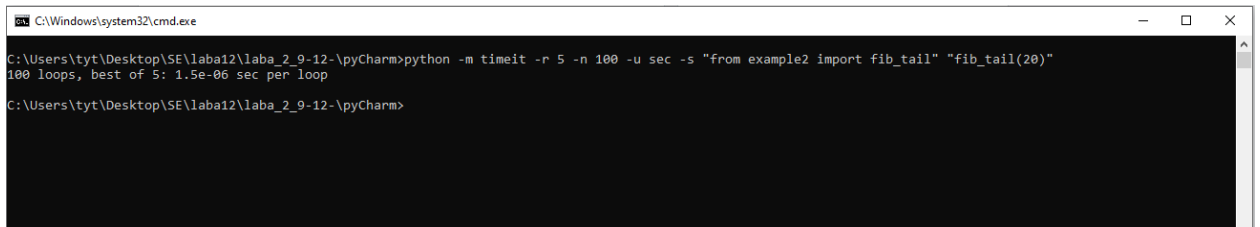


```
C:\Windows\system32\cmd.exe

C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>python -m timeit -r 5 -n 100 -u sec -s "from example2 import factorial_tail" "factorial_tail(20)"
100 loops, best of 5: 1.71e-06 sec per loop

C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>
```

Рисунок 8.4 – Результат замера времени выполнения



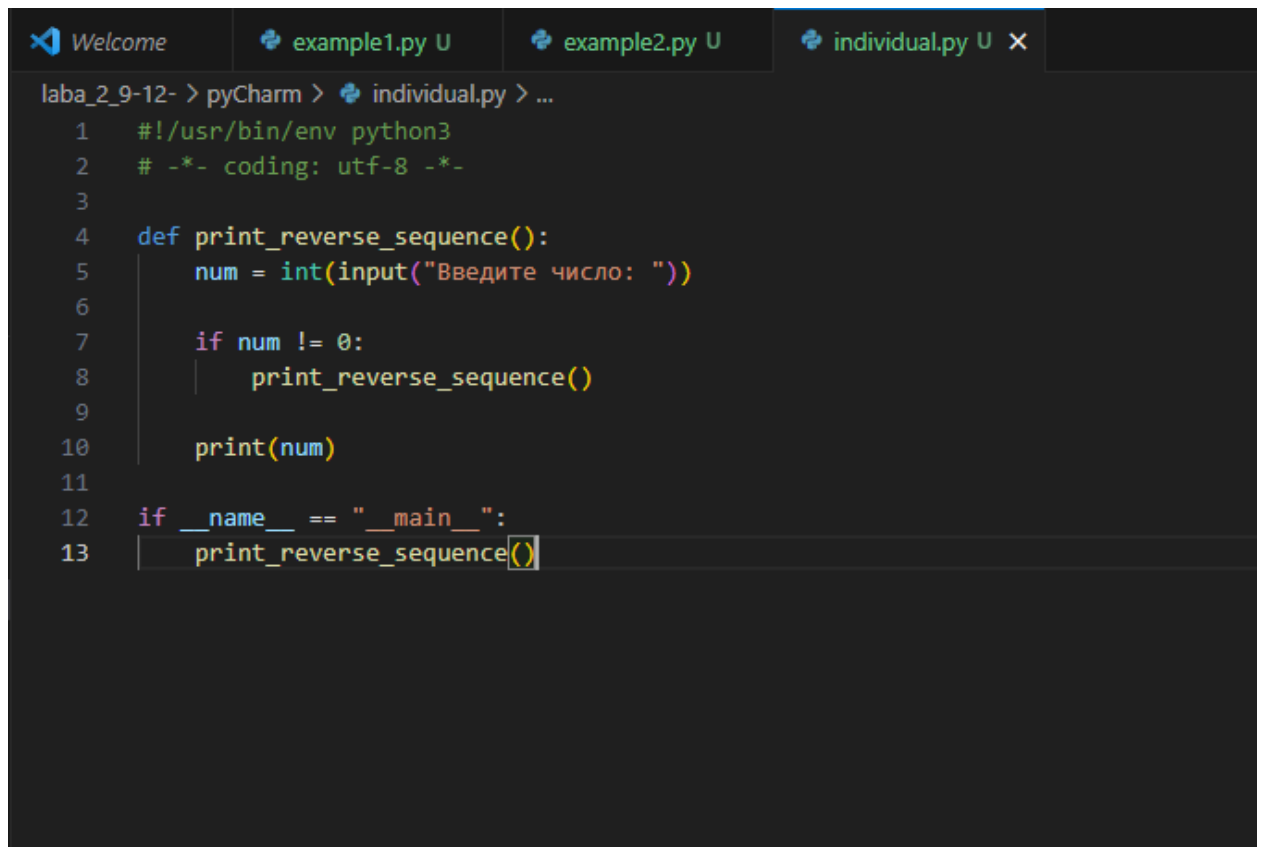
```
C:\Windows\system32\cmd.exe

C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>python -m timeit -r 5 -n 100 -u sec -s "from example2 import fib_tail" "fib_tail(20)"
100 loops, best of 5: 1.5e-06 sec per loop

C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12-\pyCharm>
```

Рисунок 8.5 – Результат замера времени выполнения

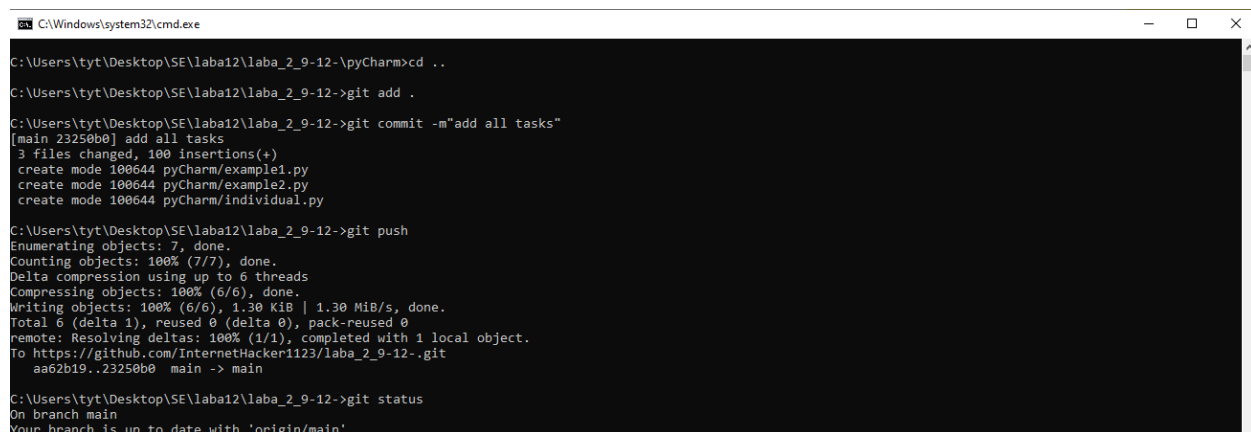
9. Выполнил индивидуальные задания. Привел в отчете скриншоты работы программ решения индивидуального задания.



```
laba_2_9-12 > pyCharm > individual.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  def print_reverse_sequence():
5      num = int(input("Введите число: "))
6
7      if num != 0:
8          print_reverse_sequence()
9
10     print(num)
11
12 if __name__ == "__main__":
13     print_reverse_sequence()
```

Рисунок 9.1 – Код программы individual.py в IDE PyCharm

10. Зафиксировал сделанные изменения в репозитории.



```
C:\Windows\system32\cmd.exe
C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12\pyCharm>cd ..
C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12>git add .
C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12>git commit -m"add all tasks"
[main 23250b0] add all tasks
3 files changed, 100 insertions(+)
create mode 100644 pyCharm/example1.py
create mode 100644 pyCharm/example2.py
create mode 100644 pyCharm/individual.py
C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12>git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 6 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1.30 KiB | 1.30 MiB/s, done.
Total 6 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/InternetHacker1123/laba_2_9-12-.git
aa62b19..23250b0 main -> main
C:\Users\tyt\Desktop\SE\laba12\laba_2_9-12>git status
On branch main
Your branch is up to date with 'origin/main'.
```

Рисунок 10.1 – Коммит файлов в репозитории git

Контрольные вопросы

1. Для чего нужна рекурсия?

Рекурсия функции нужна, когда требуется выполнить последовательность из одинаковых действий.

2. Что называется базой рекурсии?

База рекурсии – это такие аргументы функции, которые делают задачу настолько простой, что решение не требует дальнейших вложенных вызовов.

3. Самостоятельно изучите что является стеком программы. Как используется стек программы при вызове функций?

Стек — это особая область памяти, которая используется для временного хранения данных во время выполнения программы. Стек работает по принципу LIFO (last in, first out). Стек вызовов работает так: при вызове вложенной функции, основная функция, откуда был вызов останавливается и создается блок памяти под новый вызов. В ячейку памяти записываются значения переменных и адрес возврата.

4. Как получить текущее значение максимальной глубины рекурсии в языке Python?

Чтобы проверить текущие параметры лимита, нужно запустить: `sys.getrecursionlimit()`

5. Что произойдет если число рекурсивных вызовов превысит максимальную глубину рекурсии в языке Python?

Существует предел глубины возможной рекурсии, который зависит от реализации Python. Когда предел достигнут, возникает исключение `RuntimeError`

6. Как изменить максимальную глубину рекурсии в языке Python?

Можно изменить предел глубины рекурсии с помощью вызова: `sys.setrecursionlimit(limit)`

7. Каково назначение декоратора `lru_cache`?

Декоратор `@lru_cache()` модуля `functools` оборачивает функцию с переданными в нее аргументами и запоминает возвращаемый результат соответствующий этим аргументам. Такое поведение может сэкономить время

и ресурсы, когда дорогая или связанная с вводом/выводом функция периодически вызывается с одинаковыми аргументами.

8. Что такое хвостовая рекурсия? Как проводится оптимизация хвостовых вызовов?

Хвостовая рекурсия — частный случай рекурсии, при котором любой рекурсивный вызов является последней операцией перед возвратом из функции. Чтобы оптимизировать рекурсивные функции, мы можем использовать декоратор `@tail_call_optimized` для вызова нашей функции