

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 №13**  
**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

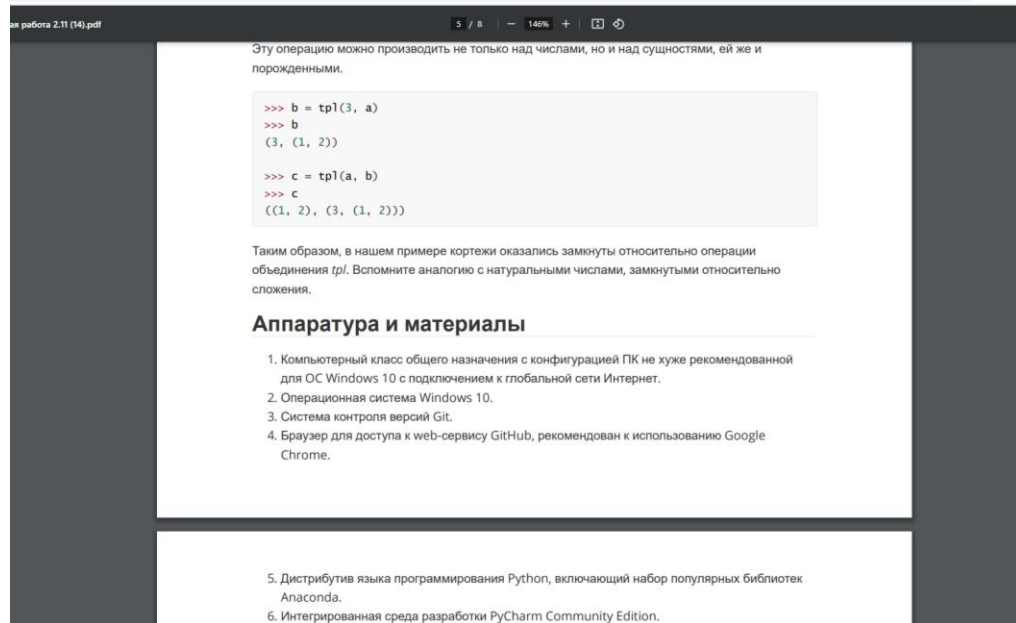


Рисунок 1.1 – Изучение материала для лабораторной работы

### 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 \*

laba14

laba14 is available.

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

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

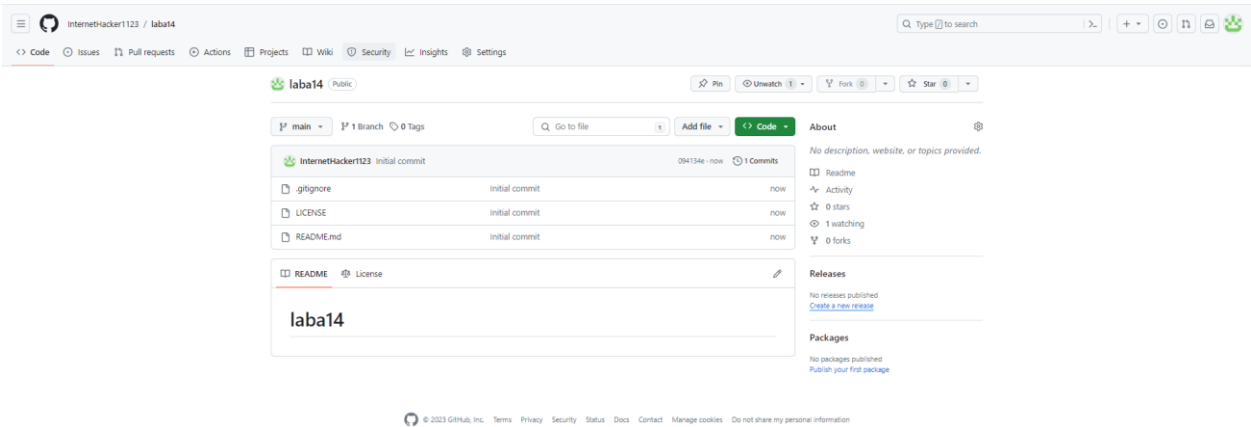
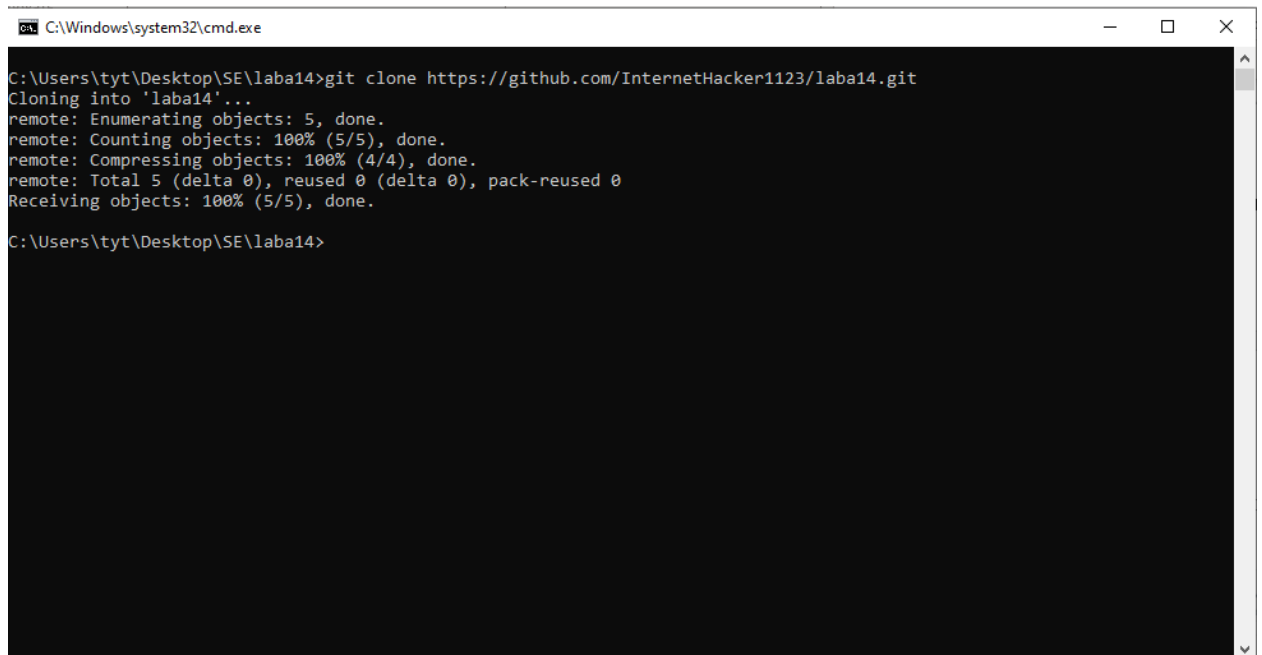


Рисунок 2.2 – Готовый репозиторий

### 3. Выполняю клонирование созданного репозитория



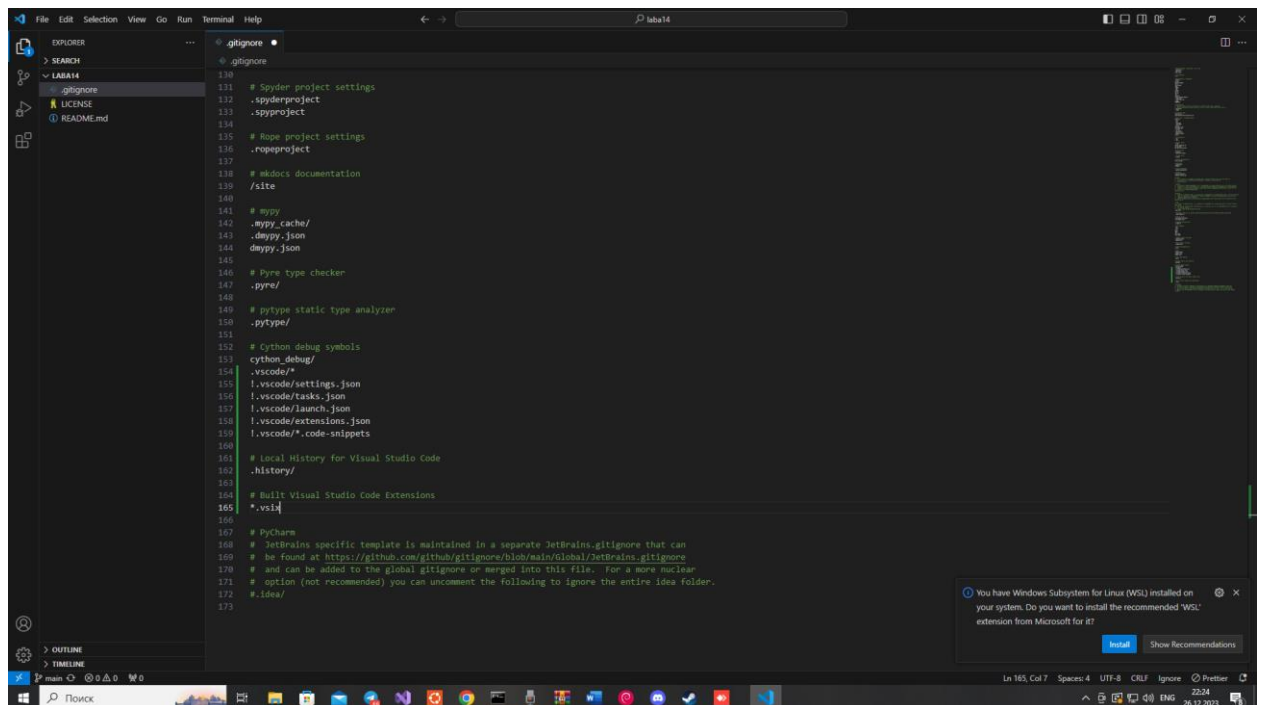
```
C:\Windows\system32\cmd.exe

C:\Users\tyt\Desktop\SE\laba14>git clone https://github.com/InternetHacker1123/laba14.git
Cloning into 'laba14'...
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\tyt\Desktop\SE\laba14>
```

Рисунок 3.1 – Клонирование репозитория на локальный диск

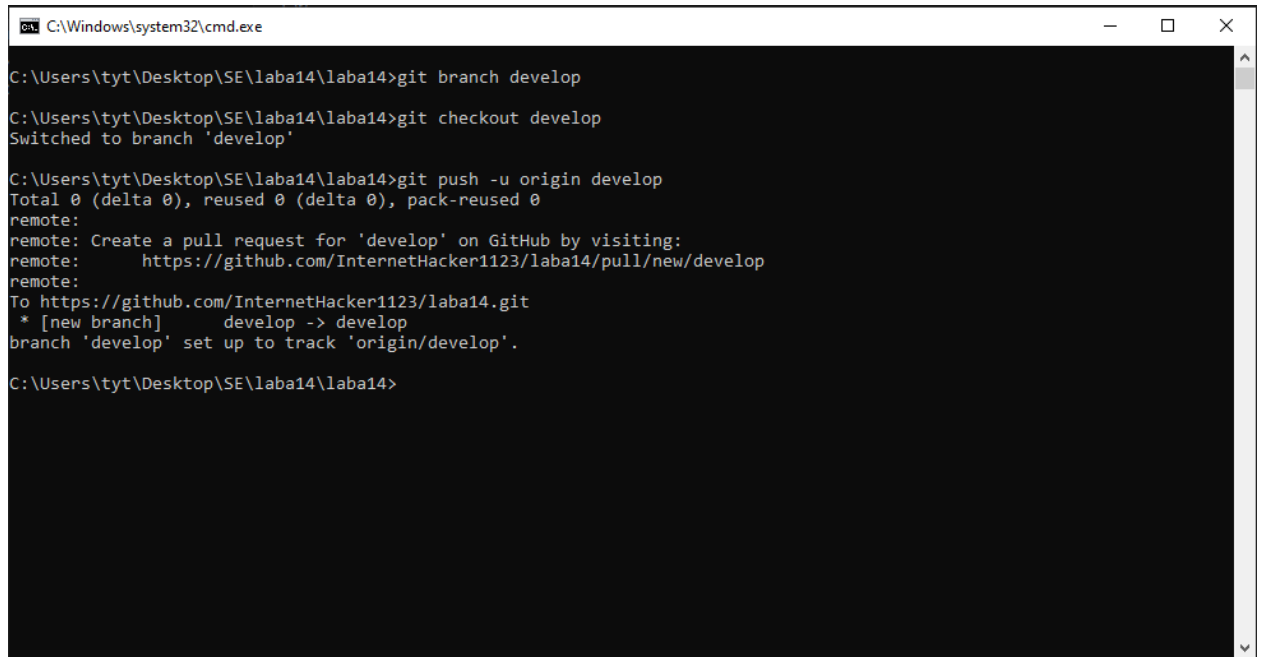
### 4. Дополнил файл .gitignore необходимыми правилами для работы с VS Code



```
.gitignore
110
111 # Spyder project settings
112 .spyderproject
113 .spyproject
114
115 # Rope project settings
116 .ropeproject
117
118 # mkdocs documentation
119 /site
120
121 # mypy
122 .mypy_cache/
123 .dmypy.json
124 dmypy.json
125
126 # Pyre type checker
127 .pyre/
128
129 # pytype static type analyzer
130 .pytype/
131
132 # cython debug symbols
133 cython_debug/
134
135 .vscode/*
136 !.vscode/settings.json
137 !.vscode/tasks.json
138 !.vscode/launch.json
139 !.vscode/extensions.json
140 !.vscode/*.code-snippets
141
142 # Local History for Visual Studio Code
143 .history/
144
145 # Built Visual Studio Code Extensions
146 *.vsix
147
148 # PyCharm
149 # JetBrains specific template is maintained in a separate JetBrains.gitignore that can
150 # be found at https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore
151 # and can be added to the global gitignore or merged into this file. For a more nuclear
152 # option (not recommended) you can uncomment the following to ignore the entire idea folder.
153 *.idea/
154
```

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\tyt\Desktop\SE\laba14\laba14>git branch develop
C:\Users\tyt\Desktop\SE\laba14\laba14>git checkout develop
Switched to branch 'develop'

C:\Users\tyt\Desktop\SE\laba14\laba14>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/laba14/pull/new/develop
remote:
To https://github.com/InternetHacker1123/laba14.git
 * [new branch]      develop -> develop
branch 'develop' set up to track 'origin/develop'.
```

Рисунок 5.1 – Создание ветки develop от ветки main

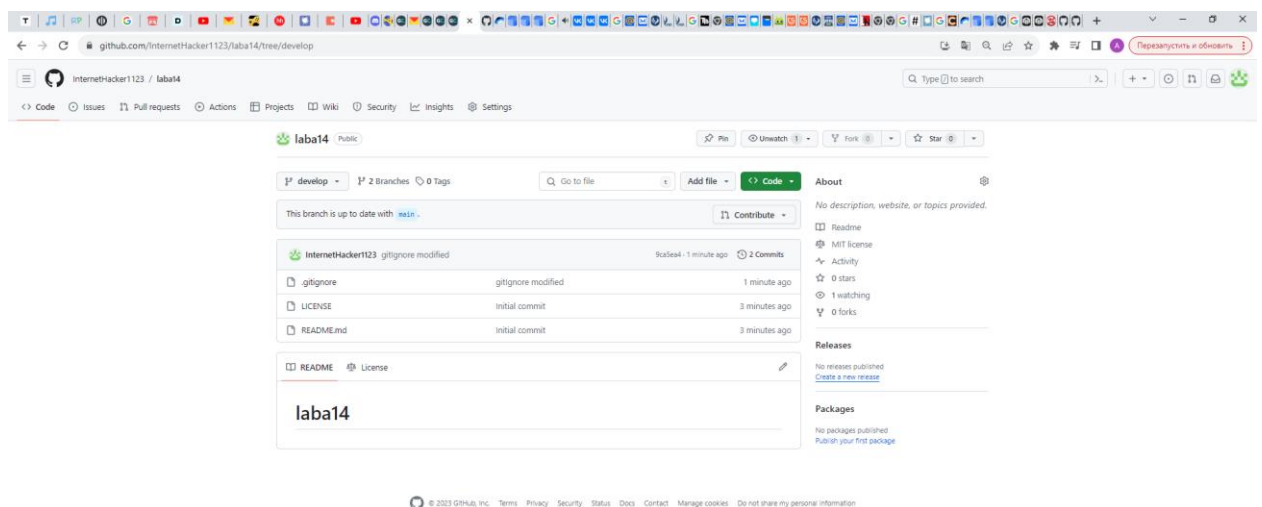


Рисунок 5.2 – Ветка develop на GitHub

## 6. Создал проект PyCharm в папке репозитория

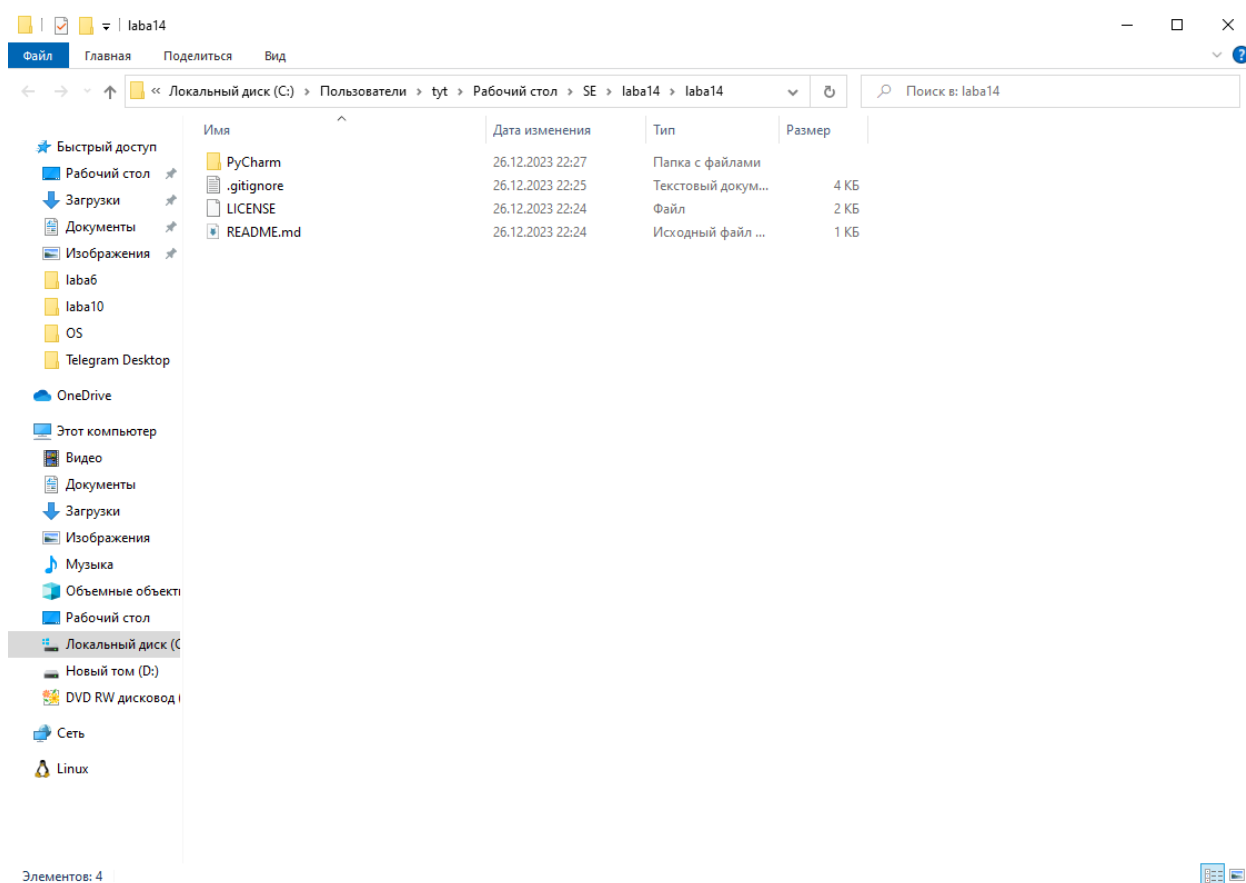


Рисунок 6.1 – Репозиторий с проектом PyCharm

7. Проработал примеры лабораторной работы.

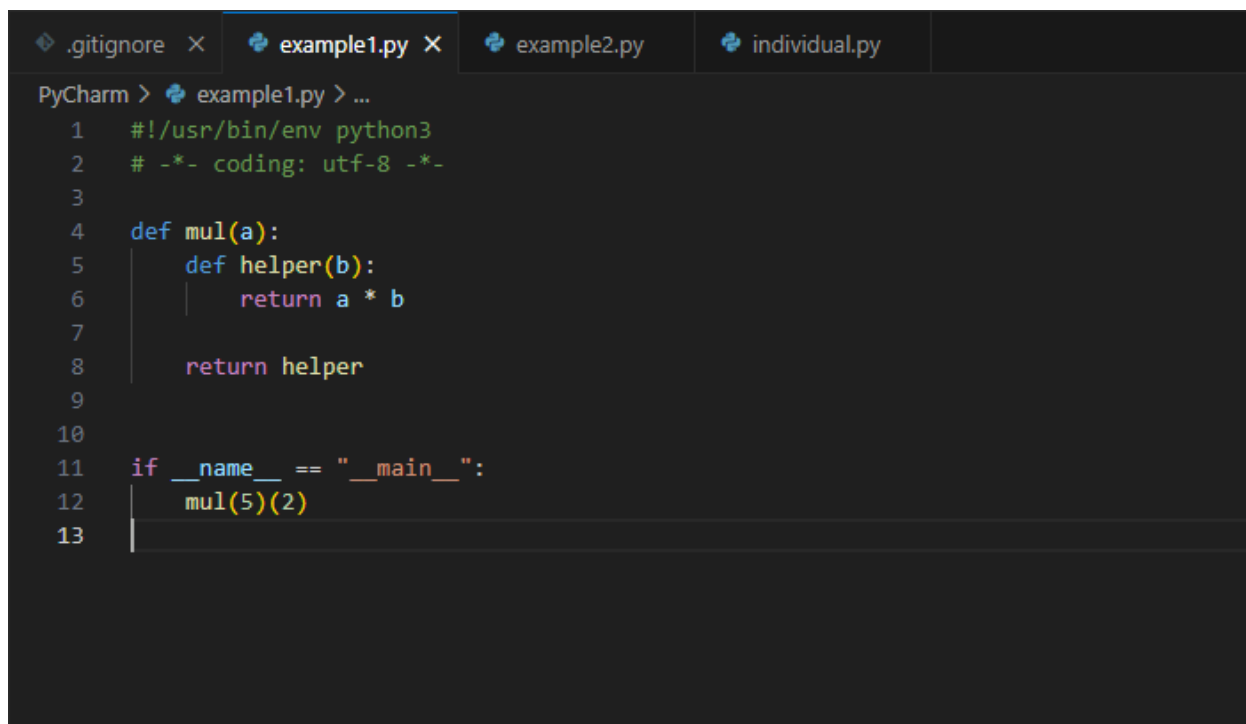


Рисунок 7.1 – Код программы example1.py



```
.gitignore × example1.py example2.py × individual.py
PyCharm > example2.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  def fun1(a):
5      x = a * 3
6
7      def fun2(b):
8          nonlocal x
9          return b + x
10
11     return fun2
12
13
14 if __name__ == "__main__":
15     test_fun = fun1(4)
16     test_fun(7)
17     print(test_fun)
18
```

Рисунок 7.2 – Код программы example2.py

8.     Выполнил индивидуальное задание.

```
.gitignore example1.py M example2.py U individual.py U X
PyCharm > individual.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  def outer_function(f):
5      def inner_function(a, b):
6          result = f(a, b)
7          return f"Для значений {a}, {b} функция f({a}, {b}) = {result}"
8      return inner_function
9
10
11 # Пример функции f, которую мы передадим во внешнюю функцию
12 def my_function(a, b):
13     return a * b + a - b
14
15
16 # Создаем замыкание
17 closure = outer_function(my_function)
18
19
20 if __name__ == "__main__":
21     a = int(input("Введите первое число: "))
22     b = int(input("Введите второе число: "))
23     print(closure(a, b))
24
```

Рисунок 8.1 – Код программы individual.py

## 9. Зафиксировал изменения в репозитории.

```
C:\Windows\system32\cmd.exe
C:\Users\tyt\Desktop\SE\laba14\laba14>explorer .
C:\Users\tyt\Desktop\SE\laba14\laba14>git add .
C:\Users\tyt\Desktop\SE\laba14\laba14>git commit -m"add examples and task"
[develop b9a07ae] add examples and task
3 files changed, 52 insertions(+)
create mode 100644 PyCharm/example2.py
create mode 100644 PyCharm/individual.py
C:\Users\tyt\Desktop\SE\laba14\laba14>git push
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 6 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1.02 KiB | 1.02 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/laba14.git
   bd5fc33..b9a07ae  develop -> develop
C:\Users\tyt\Desktop\SE\laba14\laba14>
```



### Контрольные вопросы

1. Что такое замыкание?

Замыкание (closure) в программировании — это функция, в теле которой присутствуют ссылки на переменные, объявленные вне тела этой функции в окружающем коде и не являющиеся ее параметрами.

2. Как реализованы замыкания в языке программирования Python?

Замыкание (closure) в Python — это функция, которая запоминает значения в окружающей (внешней) области видимости, даже если эта область видимости больше не существует. Замыкание возникает, когда внутри функции определены вложенные функции, и вложенная функция ссылается на переменные из внешней функции.

3. Что подразумевает под собой область видимости Local?

Эту область видимости имеют переменные, которые создаются и используются внутри функций

4. Что подразумевает под собой область видимости Enclosing?

Суть данной области видимости в том, что внутри функции могут быть вложенные функции и локальные переменные, так вот локальная переменная функции для ее вложенной функции находится в enclosing области видимости.

5. Что подразумевает под собой область видимости Global?

Переменные области видимости global – это глобальные переменные уровня модуля (модуль – это файл с расширением .py).

6. Что подразумевает под собой область видимости Build-in?

Уровень Python интерпретатора. В рамках этой области видимости находятся функции open, len и т. п., также туда входят исключения. Эти сущности доступны в любом модуле Python и не требуют предварительного импорта. Builtin – это максимально широкая область видимости.

7. Как использовать замыкания в языке программирования Python?

Использование замыканий в Python обычно связано с созданием функций внутри других функций и возвратом этих вложенных функций. Замыкания полезны, когда вам нужно передать часть контекста (переменные) в функцию, чтобы она могла использовать их даже после того, как внешняя функция завершила свою работу.

8. Как замыкания могут быть использованы для построения иерархических данных?

Замыкания в Python можно использовать для построения иерархических данных, таких как деревья или вложенные структуры. Замыкания позволяют сохранять состояние внешней функции внутри вложенной функции, что обеспечивает уровень иерархии.