

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 №2.13
дисциплины «Основы программной инженерии»

Выполнил:
Соколов Михаил Романович
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:
Богданов С.С., ассистент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

Тема: Модули и пакеты.

Цель работы: приобретение навыков по работе с модулями и пакетами языка программирования Python версии 3.x.

Ход выполнения работы:

1. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и .gitignore файл для языка 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 ().*

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner * PoTtaTto ▾ / **Repository name *** fundamentals_of_software

✔ fundamentals_of_software_engineering_2_13 is available.

Great repository names are short and memorable. Need inspiration? How about **fluffy-palm-tree** ?

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

Рисунок 1 – Создание репозитория с заданными настройками

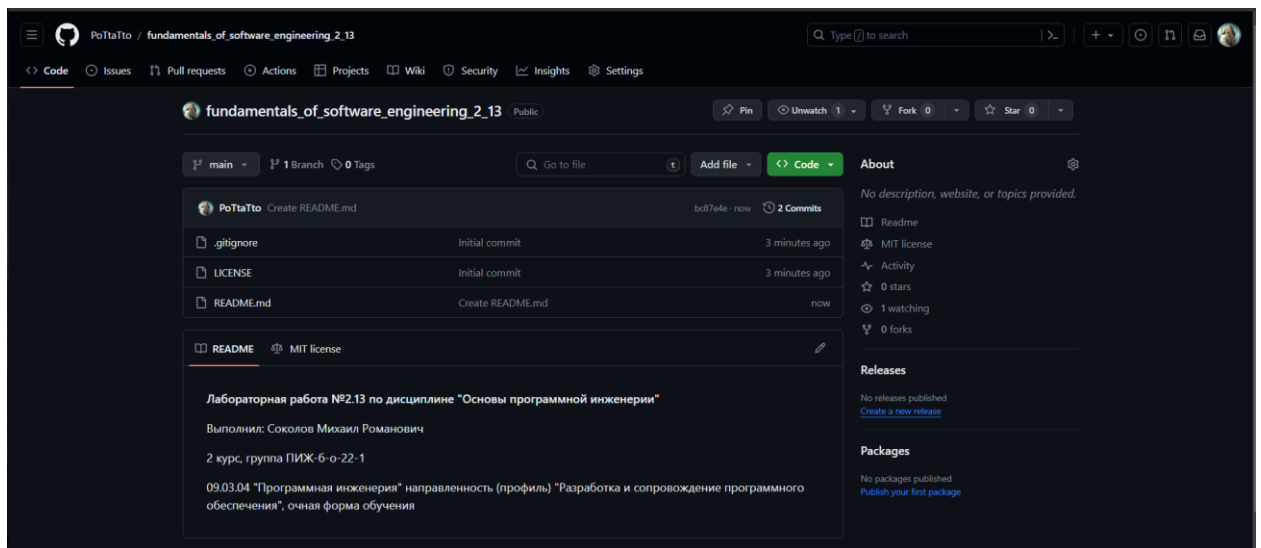


Рисунок 2 – Созданный репозиторий



Рисунок 3 – Клонирование репозитория

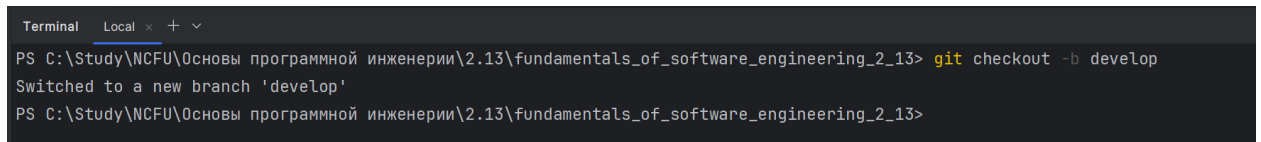


Рисунок 4 – Создание ветки develop, где будут происходить изменения проекта до его полного релиза

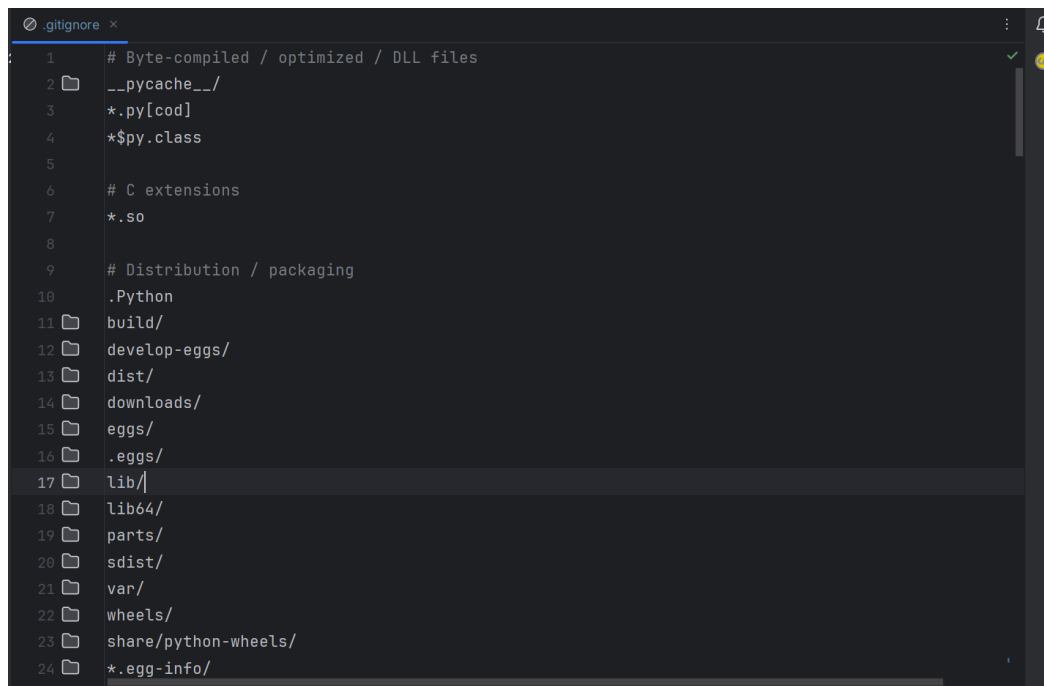


Рисунок 5 – Часть .gitignore, созданного GitHub

2. Выполним индивидуальные задания:

Задание №1: выполнить индивидуальное задание лабораторной работы 2.11, оформив все функции программы в виде отдельного модуля. Разработанный модуль должен быть подключен в основную программу с помощью одного из вариантов команды `import`:

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  2 usages
5  def area_calculator(type=0):
6      def calculate_triangle_area(base, height):
7          return 0.5 * base * height
8
9      def calculate_rectangle_area(length, width):
10         return length * width
11
12     if type == 0:
13         return calculate_triangle_area
14     else:
15         return calculate_rectangle_area
16
17  if __name__ == '__main__':
18      # Вызов внешней функции с параметром type = 0 для вычисления площади треугольника
19      triangle_area = area_calculator(0)
20      result_triangle = triangle_area(10, 8) # Вычисление площади треугольника с основанием 10 и высотой 8
21
22      print('Площадь треугольника:', result_triangle)
23
24      # Вызов внешней функции с параметром type != 0 для вычисления площади прямоугольника
25      rectangle_area = area_calculator(10)
26      result_rectangle = rectangle_area(10, 9) # Вычисление площади прямоугольника с длиной 10 и шириной 9
27
28      print('Площадь прямоугольника:', result_rectangle)
```

Рисунок 6 – Оригинальный код индивидуального задания из лабораторной работы 2.11

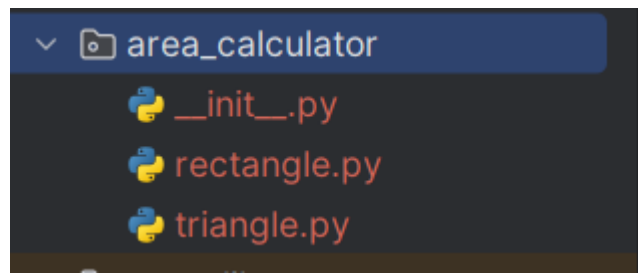


Рисунок 7 – Создадим пакет area_calculator с модулями rectangle.py и triangle.py

```
rectangle.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  1 usage
5  def calculate_area(length: float, width: float) -> float:
6      return length * width
```

Рисунок 8 – Код модуля rectangle.py

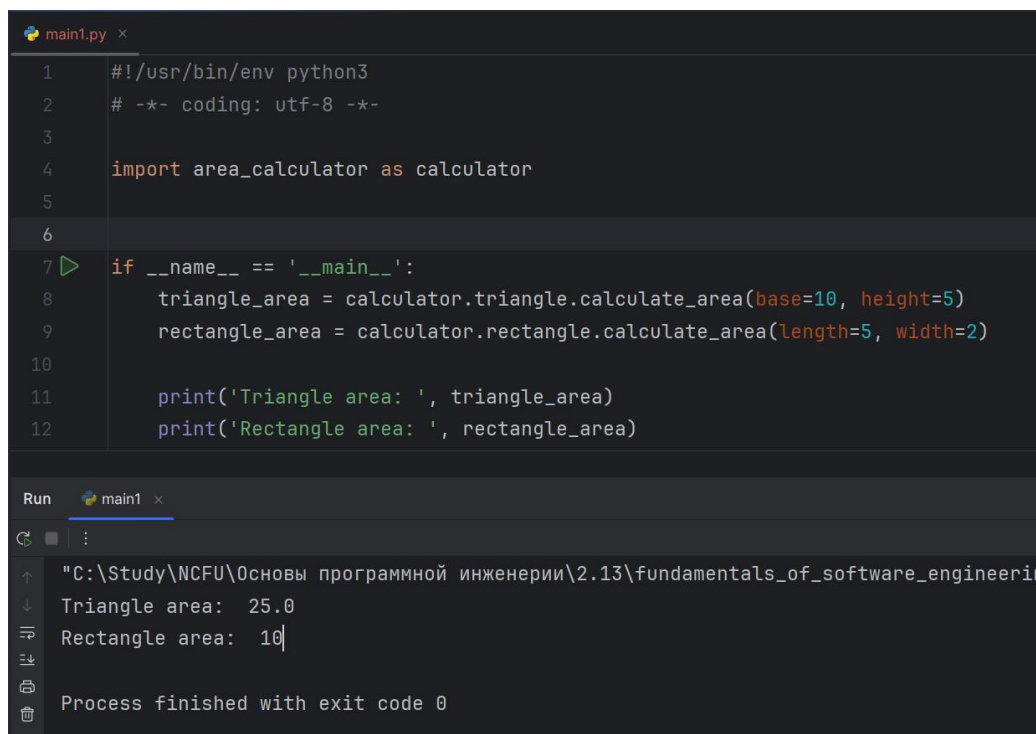
```
triangle.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  1 usage
5  def calculate_area(base: float, height: float) -> float:
6      return 0.5 * base * height
```

Рисунок 9 – Код модуля triangle.py



```
__init__.py x
1 from . import rectangle, triangle
2
3 __all__ = ['rectangle', 'triangle']
```

Рисунок 10 – Сохранение списка модулей пакета в файле `__init__.py`



```
main1.py x
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 import area_calculator as calculator
5
6
7 if __name__ == '__main__':
8     triangle_area = calculator.triangle.calculate_area(base=10, height=5)
9     rectangle_area = calculator.rectangle.calculate_area(length=5, width=2)
10
11     print('Triangle area: ', triangle_area)
12     print('Rectangle area: ', rectangle_area)
```

Run main1 x

"C:\Study\NCFU\Основы программной инженерии\2.13\fundamentals_of_software_engineering\main1.py"

Triangle area: 25.0
Rectangle area: 10

Process finished with exit code 0

Рисунок 11 – Запуск программы, использующая созданный пакет и его модули

Задание №2: выполнить индивидуальное задание лабораторной работы 2.8, оформив все классы программы в виде отдельного пакета. Разработанный пакет должен быть подключен в основную программу с помощью одного из вариантов команды `import`. Настроить соответствующим образом переменную `__all__` в файле `__init__.py` пакета:

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

import sys
from random import randint
```

```

def add_train(trains):
    """
    Добавляет информацию о поезде в список trains.

    Args:
    - trains (list): Список поездов.

    """
    train_num = int(input('Введите номер поезда: '))
    destination = input('Введите пункт назначения: ')
    start_time = input('Введите время выезда: ')
    trains.append({'num': train_num, 'destination': destination,
'start_time': start_time})
    if len(trains) > 1:
        trains.sort(key=lambda item: item['start_time'])

def list_trains(trains):
    """
    Выводит список поездов на экран.

    Args:
    - trains (list): Список поездов.

    """
    line = f'+-{"-" * 15}+-{"-" * 30}+-{"-" * 25}+-'
    print(line)
    header = f"| {'№ поезда':^15} | {'Пункт назначения':^30} | {'Время
отъезда':^25} |"
    print(header)
    print(line)
    for train in trains:
        num = train.get('num', randint(1000, 10000))
        destination = train.get('destination', 'None')
        start_time = train.get('start_time', 'None')
        recording = f"| {num:^15} | {destination:^30} | {start_time:^25}
|"
        print(recording)
    print(line)

def select_train(trains, cmd_parts):
    """
    Выводит информацию о поездах, направляющихся в указанный пункт.

    Args:
    - trains (list): Список поездов.
    - cmd_parts (list): Список команды и параметра.

    """
    cmd_destination = cmd_parts[1]
    select_trains = [train for train in trains if
train['destination'].strip() == cmd_destination]
    if len(select_trains) >= 1:
        for train in select_trains:
            print(f'{train["num"]:^15}: {train["start_time"]:^25}')
    else:
        print('Нет поездов едущих в данное место!', file=sys.stderr)

```

```

def show_help():
    """
    Выводит список доступных команд на экран.

    """
    print("Список команд:\n")
    print("add - добавить поезд;")
    print("list - вывести список поездов;")
    print("select <пункт назначения> - запросить поезда с пунктом
назначения;")
    print("help - отобразить справку;")
    print("exit - завершить работу с программой.")

if __name__ == '__main__':
    trains = []
    while True:
        cmd = input('>>> ')
        cmd_parts = cmd.split(maxsplit=1)
        match cmd_parts[0]:
            case 'add':
                add_train(trains)
            case 'list':
                list_trains(trains)
            case 'select':
                select_train(trains, cmd_parts)
            case 'help':
                show_help()
            case 'exit':
                break
            case _:
                print(f'Неизвестная команда {cmd}', file=sys.stderr)

```

Листинг 1 – Оригинальный код индивидуального задания лабораторной работы 2.8

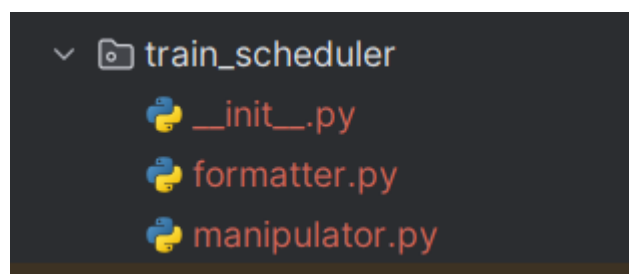


Рисунок 12 – Создадим пакет train_scheduler с модулями formatter.py и manipulator.py


```
formatter.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  # Standard
6  import sys
7
8
9  1 usage
10 def show_help():
11     """
12     Выводит список доступных команд на экран.
13
14     """
15     print("Список команд:\n")
16     print("add - добавить поезд;")
17     print("list - вывести список поездов;")
18     print("select <пункт назначения> - запросить поезда с пунктом назначения;")
19     print("help - отобразить справку;")
20     print("exit - завершить работу с программой.")
21
22  1 usage
23 def show_error(command: str):
24     """
25     Выводит ошибку о недопустимой команде на экран.
26
27     """
28     print(f'Неизвестная команда {command}', file=sys.stderr)
```

Рисунок 13 – Код модуля formatter.py

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

# Standard
import sys
from random import randint

def add_train(trains):
    """
    Добавляет информацию о поезде в список trains.

    Args:
    - trains (list): Список поездов.

    """
    train_num = int(input('Введите номер поезда: '))
    destination = input('Введите пункт назначения: ')
    start_time = input('Введите время выезда: ')
    trains.append({'num': train_num, 'destination': destination,
    'start_time': start_time})
    if len(trains) > 1:
```

```

        trains.sort(key=lambda item: item['start_time'])

def list_trains(trains):
    """
    Выводит список поездов на экран.

    Args:
    - trains (list): Список поездов.

    """
    line = f'+-{"-" * 15}-+-{"-" * 30}-+-{"-" * 25}-+'
    print(line)
    header = f"| {'№ поезда':^15} | {'Пункт назначения':^30} | {'Время отъезда':^25} |"
    print(header)
    print(line)
    for train in trains:
        num = train.get('num', randint(1000, 10000))
        destination = train.get('destination', 'None')
        start_time = train.get('start_time', 'None')
        recording = f"| {num:^15} | {destination:^30} | {start_time:^25} |"
        print(recording)
    print(line)

def select_train(trains, cmd_parts):
    """
    Выводит информацию о поездах, направляющихся в указанный пункт.

    Args:
    - trains (list): Список поездов.
    - cmd_parts (list): Список команды и параметра.

    """
    cmd_destination = cmd_parts[1]
    select_trains = [train for train in trains if train['destination'].strip() == cmd_destination]
    if len(select_trains) >= 1:
        for train in select_trains:
            print(f'{train["num"]:^15}: {train["start_time"]:^25}')
    else:
        print('Нет поездов едущих в данное место!', file=sys.stderr)

```

Листинг 2 – Код модуля manipulator.py



```

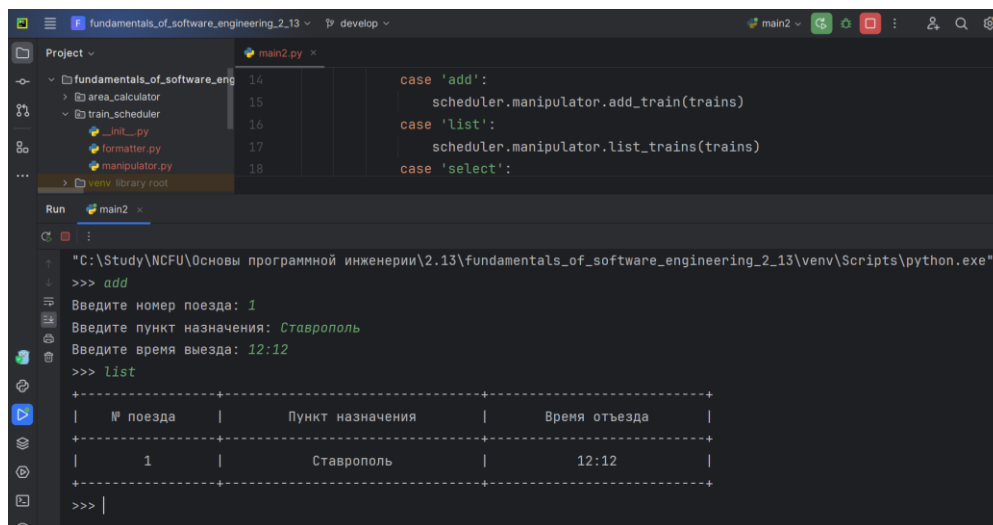
__init__.py x
1 from . import manipulator, formatter
2
3 __all__ = ['manipulator', 'formatter']

```

Рисунок 14 – Сохранение списка модулей пакета в файле __init__.py

```
main2.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  # Project
5  import train_scheduler as scheduler
6
7
8  1 usage
9  def start_program():
10     trains = []
11     while True:
12         cmd = input('>>> ')
13         cmd_parts = cmd.split(maxsplit=1)
14         match cmd_parts[0]:
15             case 'add':
16                 scheduler.manipulator.add_train(trains)
17             case 'list':
18                 scheduler.manipulator.list_trains(trains)
19             case 'select':
20                 scheduler.manipulator.select_train(trains, cmd_parts)
21             case 'help':
22                 scheduler.formatter.show_help()
23             case 'exit':
24                 break
25             case _:
26                 scheduler.formatter.show_error(command=cmd)
27
28  if __name__ == '__main__':
29     start_program()
30
```

Рисунок 15 – Использование созданного пакета и его модулей



```
main2.py x
14     case 'add':
15         scheduler.manipulator.add_train(trains)
16     case 'list':
17         scheduler.manipulator.list_trains(trains)
18     case 'select':
```

Run main2

"C:\Study\NCFU\Основы программной инженерии\2.13\fundamentals_of_software_engineering_2.13\venv\Scripts\python.exe"

```
>>> add
Введите номер поезда: 1
Введите пункт назначения: Ставрополь
Введите время выезда: 12:12
>>> list
```

№ поезда	Пункт назначения	Время отъезда
1	Ставрополь	12:12

```
>>> |
```

Рисунок 16 – Пример запущенной программы

3. Сольем ветки develop и main/master и отправим изменения на удаленный репозиторий:

```
PS C:\Study\NCFU\Основы программной инженерии\2.13\fundamentals_of_software_engineering_2_13> git log --oneline
4f7f58b (HEAD -> develop) individual task2 completed
96c21ad individual task1 completed
bc87e4e (origin/main, origin/HEAD, main) Create README.md
a81eec7 Initial commit
PS C:\Study\NCFU\Основы программной инженерии\2.13\fundamentals_of_software_engineering_2_13>
```

Рисунок 17 – История коммитов

```
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\Study\NCFU\Основы программной инженерии\2.13\fundamentals_of_software_engineering_2_13> git merge develop
Updating bc87e4e..4f7f58b
Fast-forward
 .gitignore | 2 +-
 area_calculator/__init__.py | 3 ++
 area_calculator/rectangle.py | 5 ++++
 area_calculator/triangle.py | 5 ++++
 main1.py | 12 +++++++
 main2.py | 29 +++++++++++++++++++++
 train_scheduler/__init__.py | 3 ++
 train_scheduler/formatter.py | 27 +++++++++++++++++++++
 train_scheduler/manipulator.py | 62 +++++++++++++++++++++
 9 files changed, 147 insertions(+), 1 deletion(-)
```

Рисунок 18 – Слияние веток

```
PS C:\Study\NCFU\Основы программной инженерии\2.13\fundamentals_of_software_engineering_2_13> git push origin main
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 12 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (15/15), 2.99 KiB | 2.99 MiB/s, done.
Total 15 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/PoItaIto/fundamentals_of_software_engineering_2_13
 bc87e4e..4f7f58b main -> main
PS C:\Study\NCFU\Основы программной инженерии\2.13\fundamentals_of_software_engineering_2_13>
```

Рисунок 19 – Отправка изменений на удаленный репозиторий

Ответы на контрольные вопросы:

1. Что является модулем языка Python?

Модуль языка Python — это файл с расширением `.py`, содержащий определения классов, функций, переменных и исполняемый код. Модули используются для организации кода в логические блоки, которые можно переиспользовать в различных частях программы.

2. Какие существуют способы подключения модулей в языке Python?

- Используя ключевое слово `import`, можно подключить модуль целиком. Например, `import math` подключит модуль `math`.

- С помощью конструкции `from ... import ...` можно импортировать определенные объекты из модуля. Например, `from datetime import datetime` импортирует класс `datetime` из модуля `datetime`.

- Использование `import ... as ...` позволяет импортировать модуль или его части под другим именем, что может быть полезно для сокращения названий или избежания конфликтов имен. Например, `import numpy as np`.

3. Что является пакетом языка Python?

Пакет языка Python — это способ структурирования пространства имен модулей путем использования "точечной" нотации. В простейшем случае это каталог с файлами `.py`, который также должен содержать файл `__init__.py`. Пакеты позволяют организовать модули в иерархию и использовать их как единое целое.

4. Каково назначение файла `__init__.py` ?

Назначение файла `__init__.py` — это указание для интерпретатора Python о том, что каталог является пакетом Python и может содержать модули и подпакеты. Файл `__init__.py` может быть пустым или содержать исполняемый код, который инициализирует пакет, например, устанавливает необходимые переменные, выполняет конфигурацию и т.д.

5. Каково назначение переменной `__all__` файла `__init__.py` ?

Переменная `__all__` в файле `__init__.py` определяет список имен модулей и пакетов, которые будут импортированы, когда из пакета выполняется

импорт с использованием конструкции `from ... import *`. Если `__all__` не определена, то импортируются все имена, не начинающиеся с символа подчеркивания (`_`). Определение `__all__` позволяет контролировать, какие части пакета будут доступны для пользователя после такого импорта.