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

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №3
дисциплины
«Программирование на Python»
Вариант 19

Выполнил:
Поляков Никита Александрович
2 курс, группа ИВТ-б-о-24-1,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р.А., доцент департамента цифровых, робототехнических систем и электроники института перспективной инженерии

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г.

Тема: Условные операторы и циклы в языке Python

Цель: приобретение навыков программирования разветвляющихся алгоритмов и алгоритмов циклической структуры. Освоить операторы языка Python версии 3.x if, while, for, break и continue, позволяющих реализовывать разветвляющиеся алгоритмы и алгоритмы циклической структуры.

Практическая часть:

Перед началом работы был создан новый репозиторий для лабораторной работы №3:

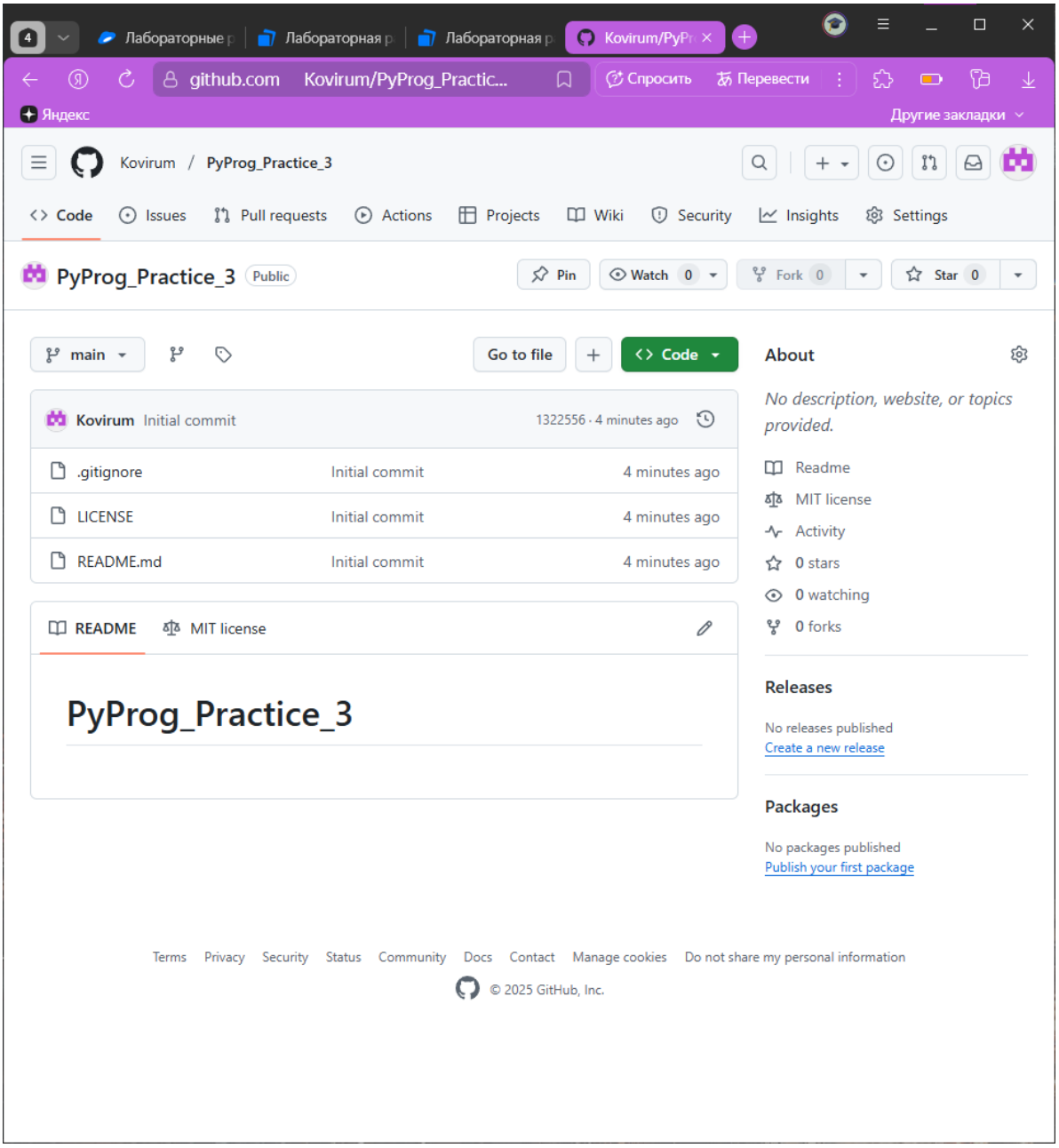

Рисунок 1. Страница созданного репозитория

Ссылка на репозиторий: https://github.com/Kovirum/PyProg_Practice_3

Далее репозиторий был клонирован на компьютер и начата работа над заданиями с соблюдением принципов модели ветвления git-flow, а также правил оформления кода PEP-8:

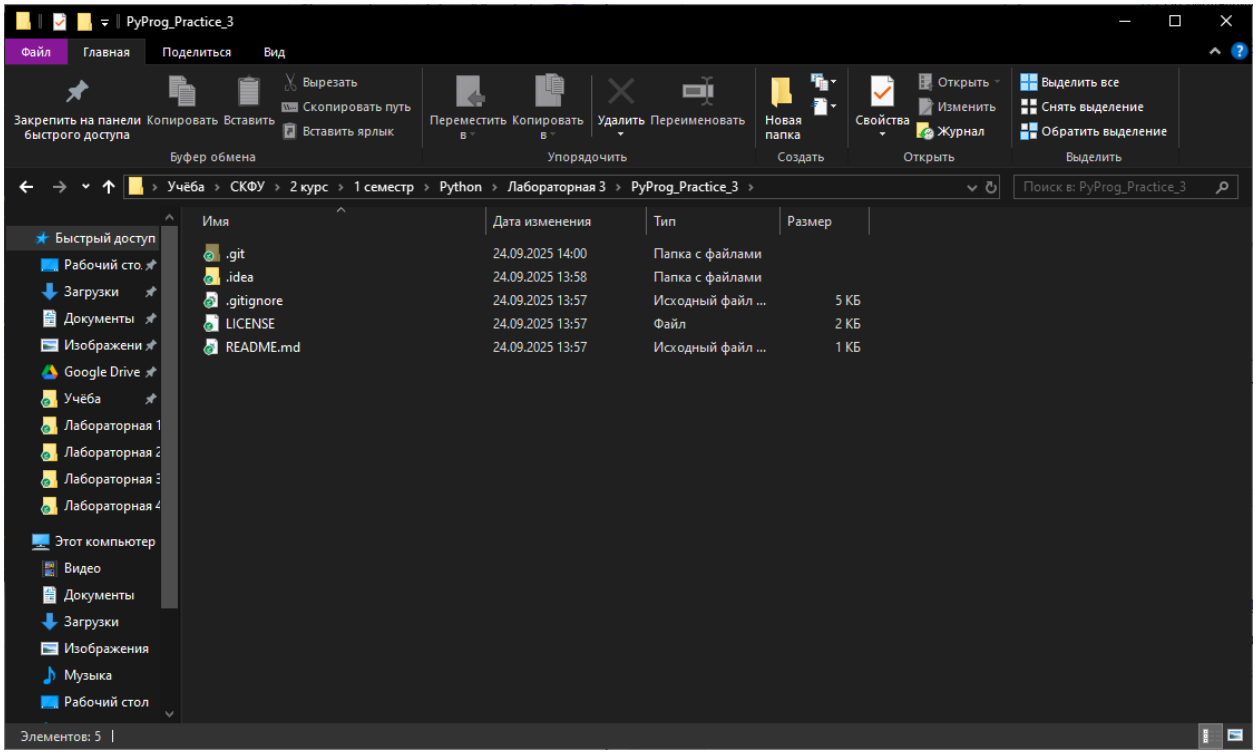

Рисунок 2. Директория локального репозитория

Для выполнения задания проработки всех примеров из методических указаний были организованы отдельные ветки с наименованием «feature/exampleN», где N – номер примера. Каждый отдельный пример реализован в собственном модуле с наименованием «exampleN.py», где N – номер примера.

Пример 1:

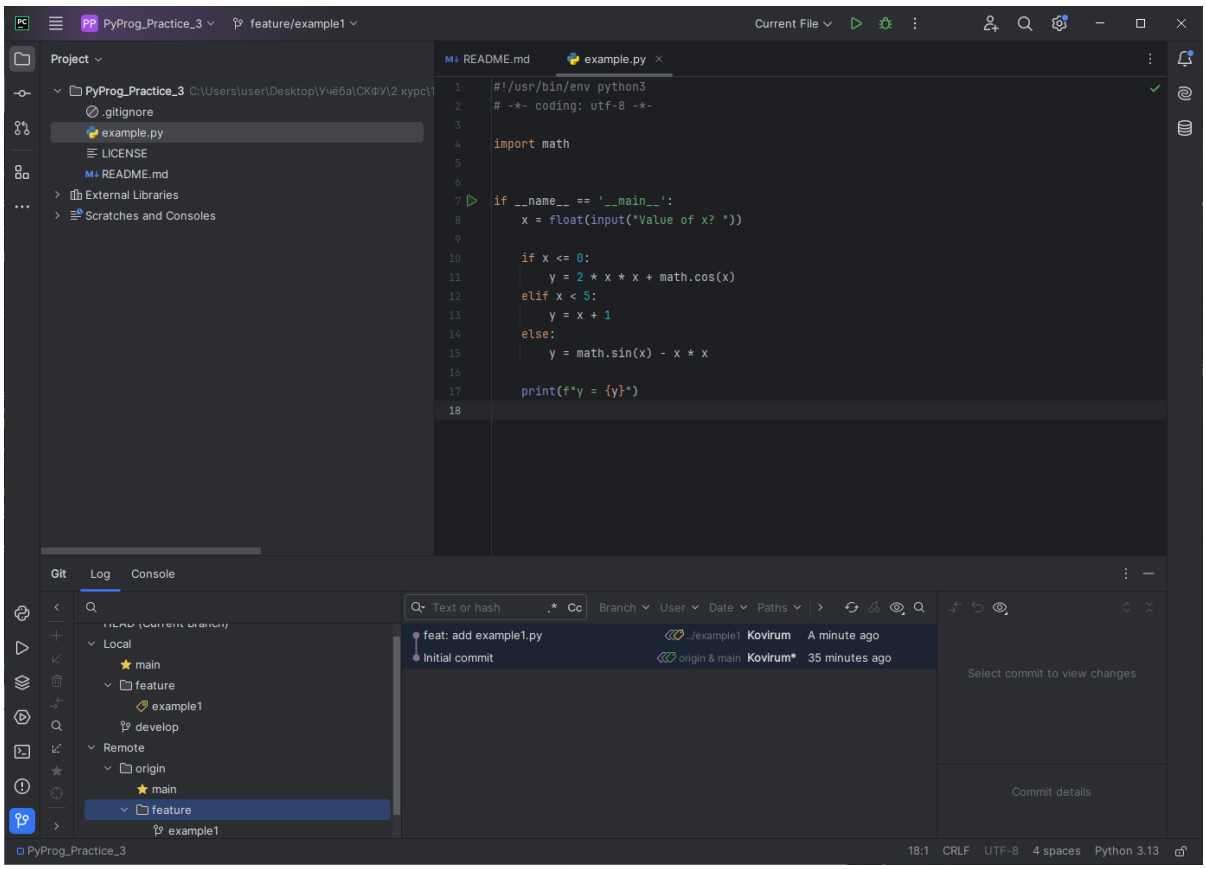

Рисунок 3. Код и git-информация примера 1

Пример 2:

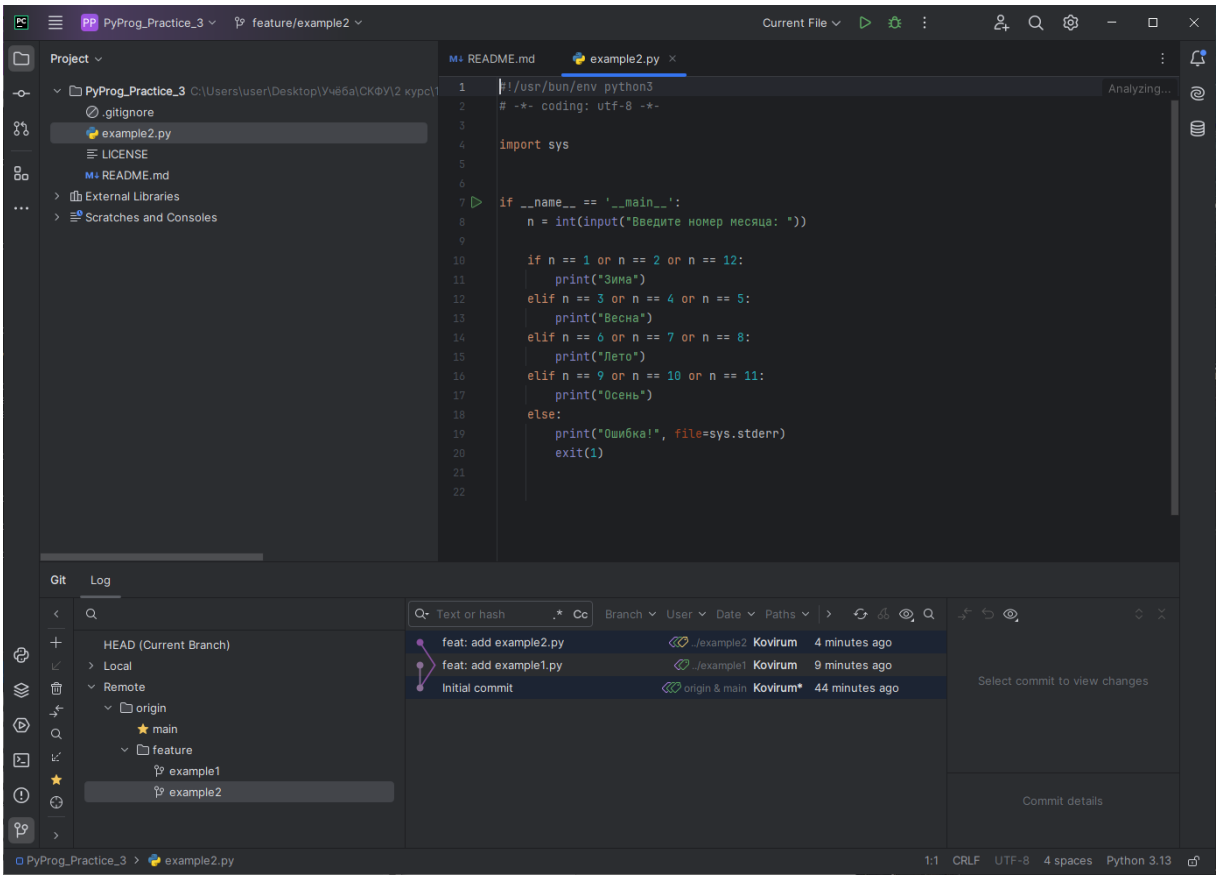

Рисунок 4. Код и git-информация примера 2

Пример 3:

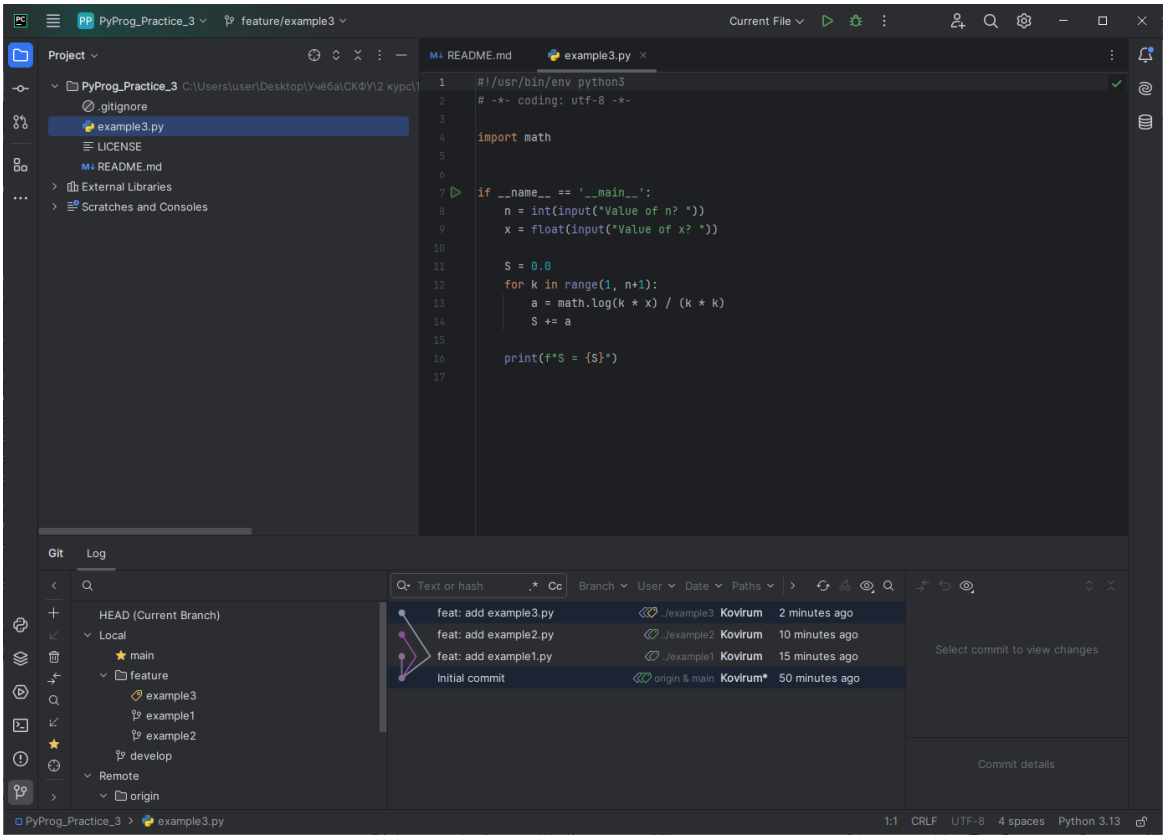

Рисунок 5. Код и git-информация примера 3

Пример 4:

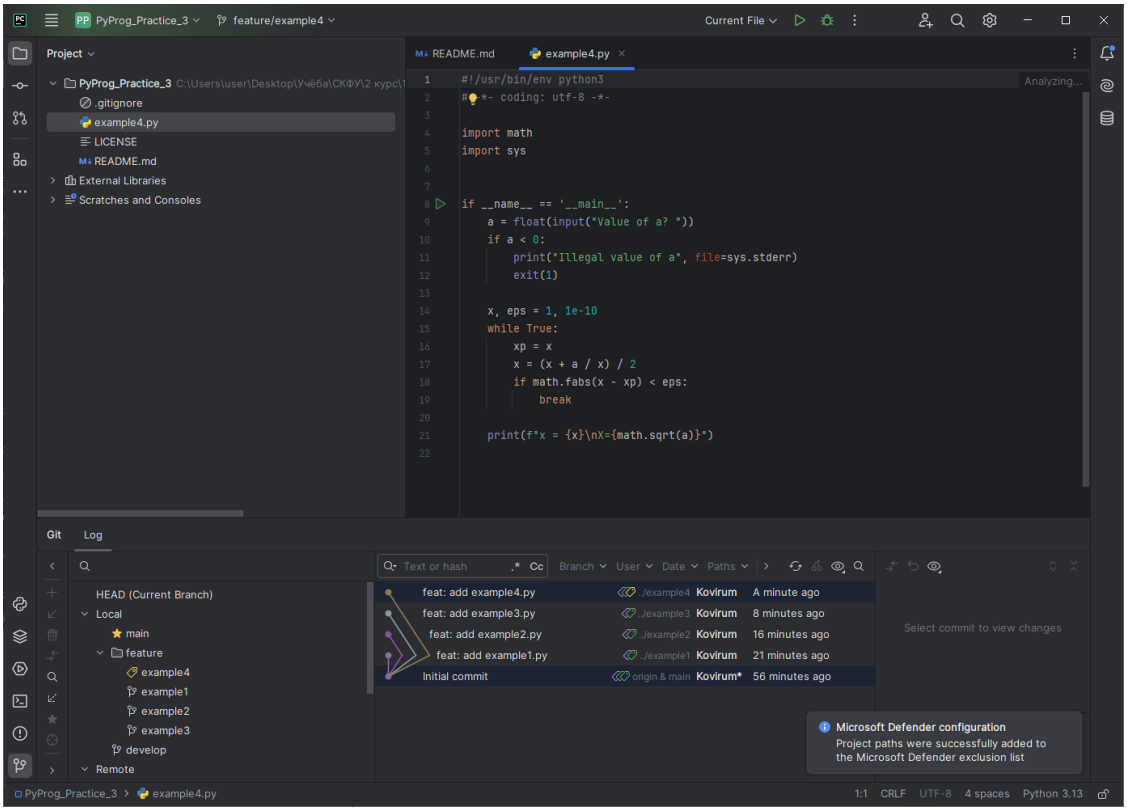

Рисунок 6. Код и git-информация примера 4

Пример 5:

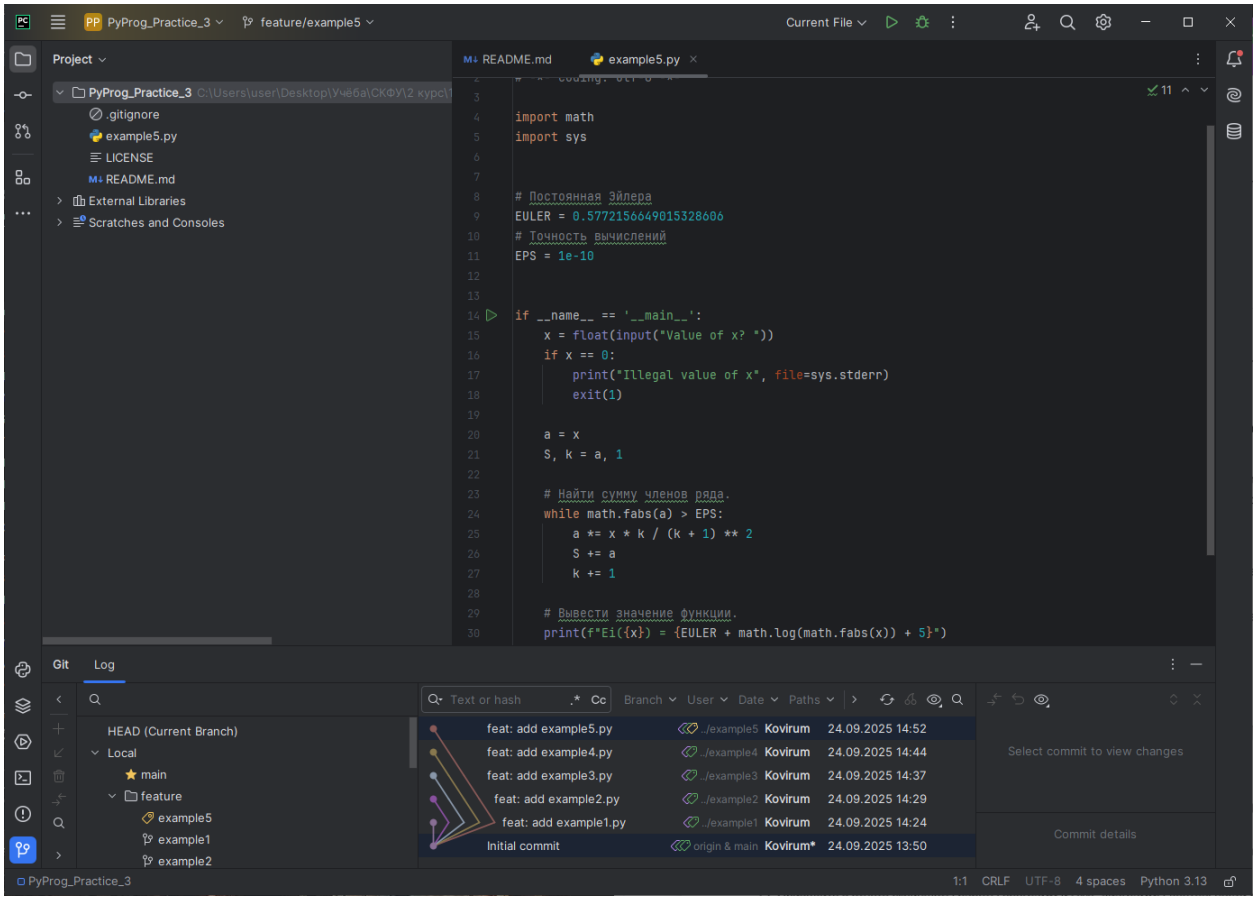

Рисунок 7. Код и git-информация примера 5

Далее были выполнены индивидуальные задания для варианта 19:

Задание 1. Дано целое число C, такое что |C| < 9. Вывести это число в словесной форме, учитывая его знак.

Код программы:

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

import sys

if __name__ == '__main__':
    c = int(input("Введите число C такое, что |C| < 9: "))

    if not abs(c) < 9:
        print("Число C должно соответствовать условию: |C| < 9",
file=sys.stderr)
        exit(1)

    result_str = ""

    if c < 0:
        result_str += "минус "
```

```
    match abs(c):
        case 0:
            result_str += "ноль"
        case 1:
            result_str += "один"
        case 2:
            result_str += "два"
        case 3:
            result_str += "три"
        case 4:
            result_str += "четыре"
        case 5:
            result_str += "пять"
        case 6:
            result_str += "шесть"
        case 7:
            result_str += "семь"
        case 8:
            result_str += "восемь"
        case 9:
            result_str += "девять"

    print(result_str)
)
```

Демонстрация работы программы:

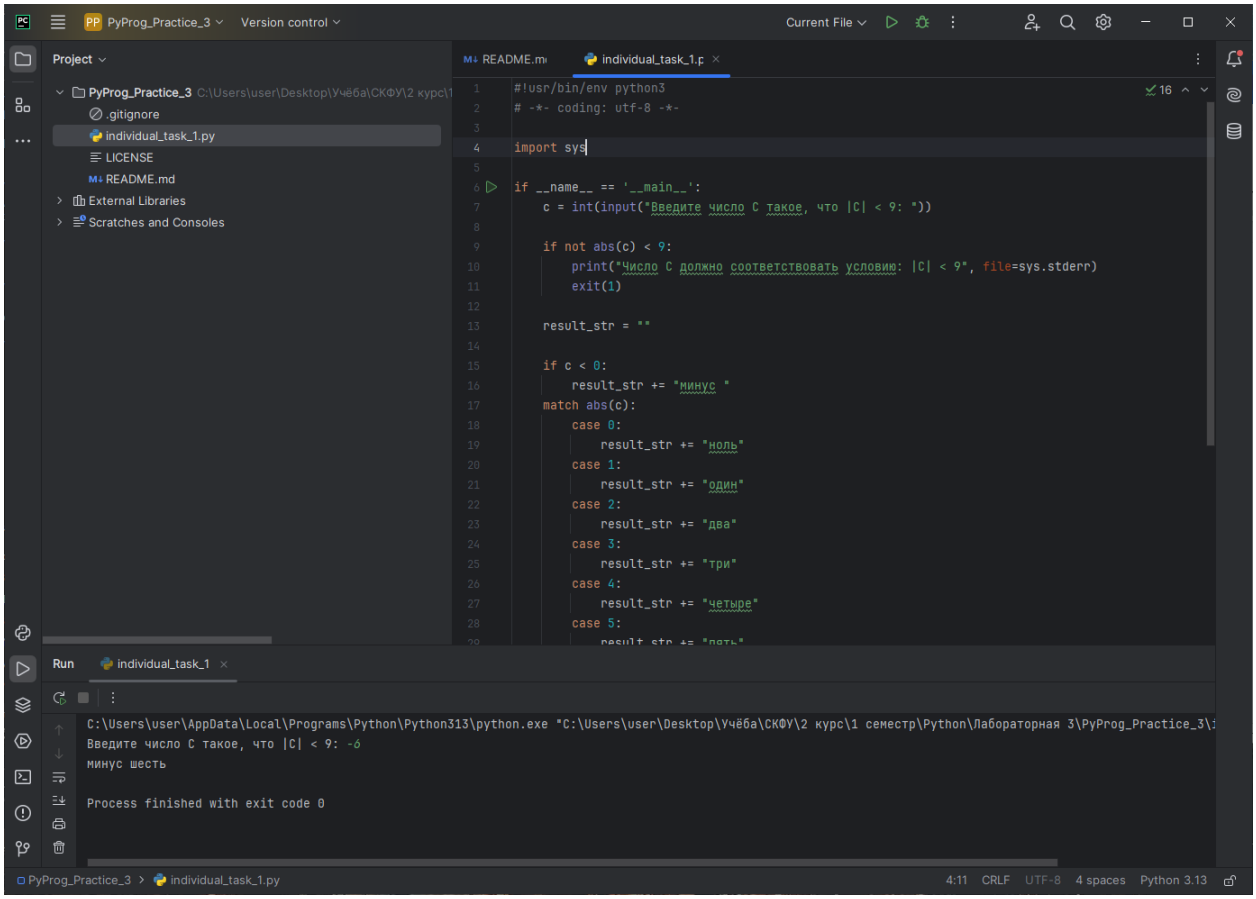

Рисунок 8. Демонстрация работы программы индивидуального задания 1

UML-диаграмма деятельности:

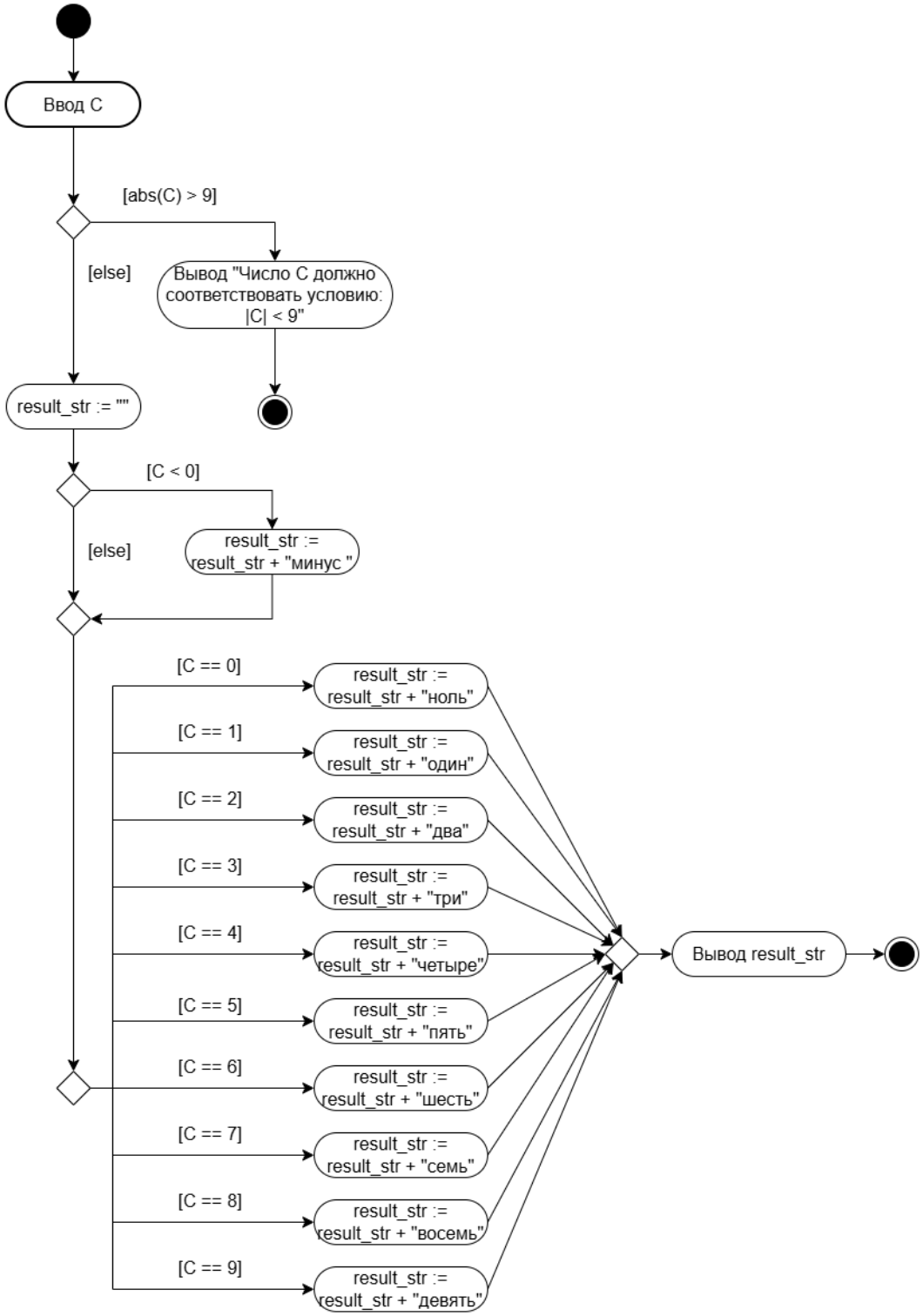

Рисунок 9. UML-диаграмма для индивидуального задания 1

Задание 2. Какая из точек A(a1, a2) или B(b1, b2) находится дальше от центра координат?

Код программы:

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

```
import math

if __name__ == '__main__':
    a1, a2 = map(int, input("Укажите координаты точки A (a1,
a2): ").split())
    b1, b2 = map(int, input("Укажите координаты точки B (b1,
b2): ").split())

    OA_distance = math.sqrt(math.pow(a1 - 0, 2) + math.pow(a2 -
0, 2))
    OB_distance = math.sqrt(math.pow(b1 - 0, 2) + math.pow(b2 -
0, 2))

    if OA_distance > OB_distance:
        print("Точка A находится дальше от начала координат")
    elif OA_distance == OB_distance:
        print("Точки A и B находятся на одинаковом удалении от
начала координат")
    else:
        print("Точка B находится дальше от начала координат")
```

Демонстрация работы программы:

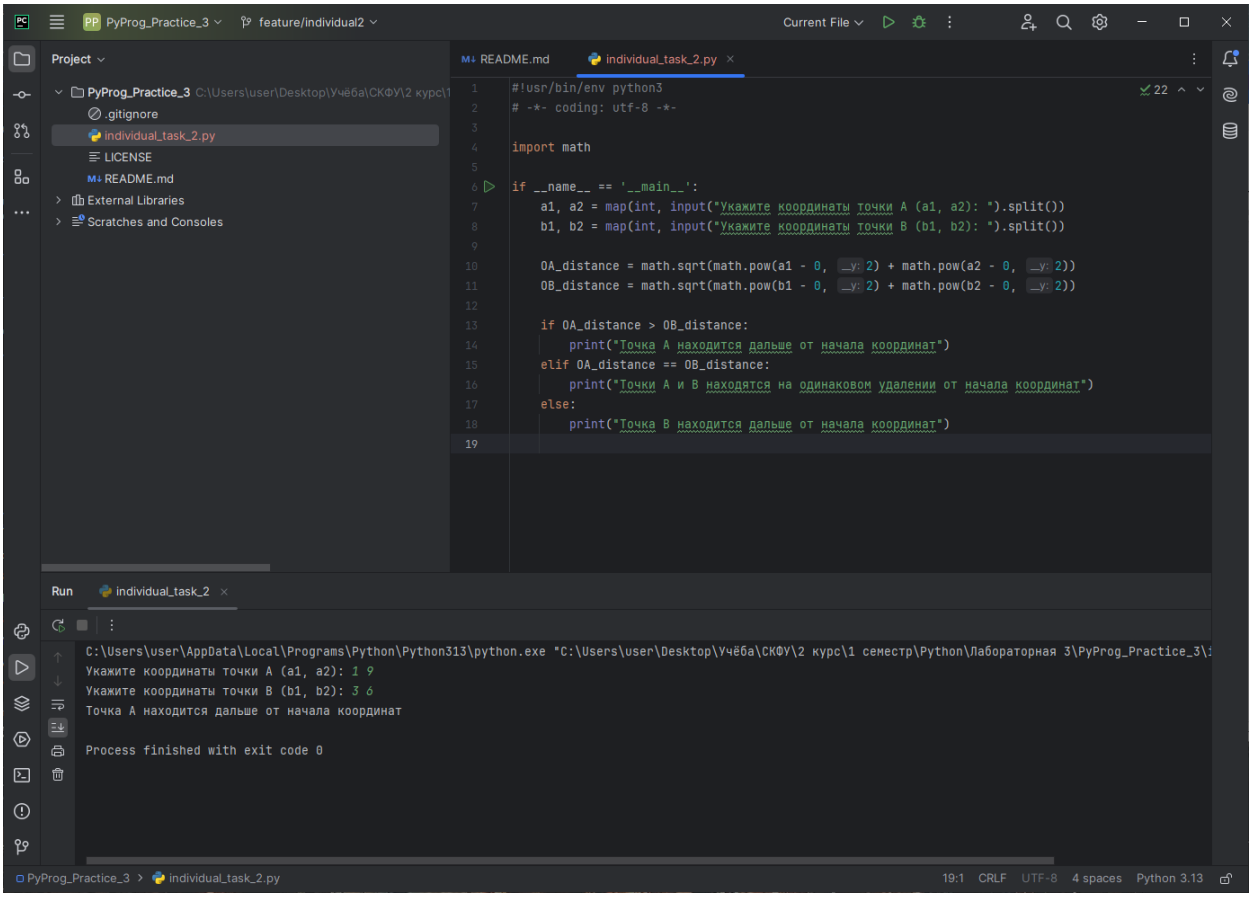

Рисунок 10. Демонстрация работы программы индивидуального задания 2

UML-диаграмма:

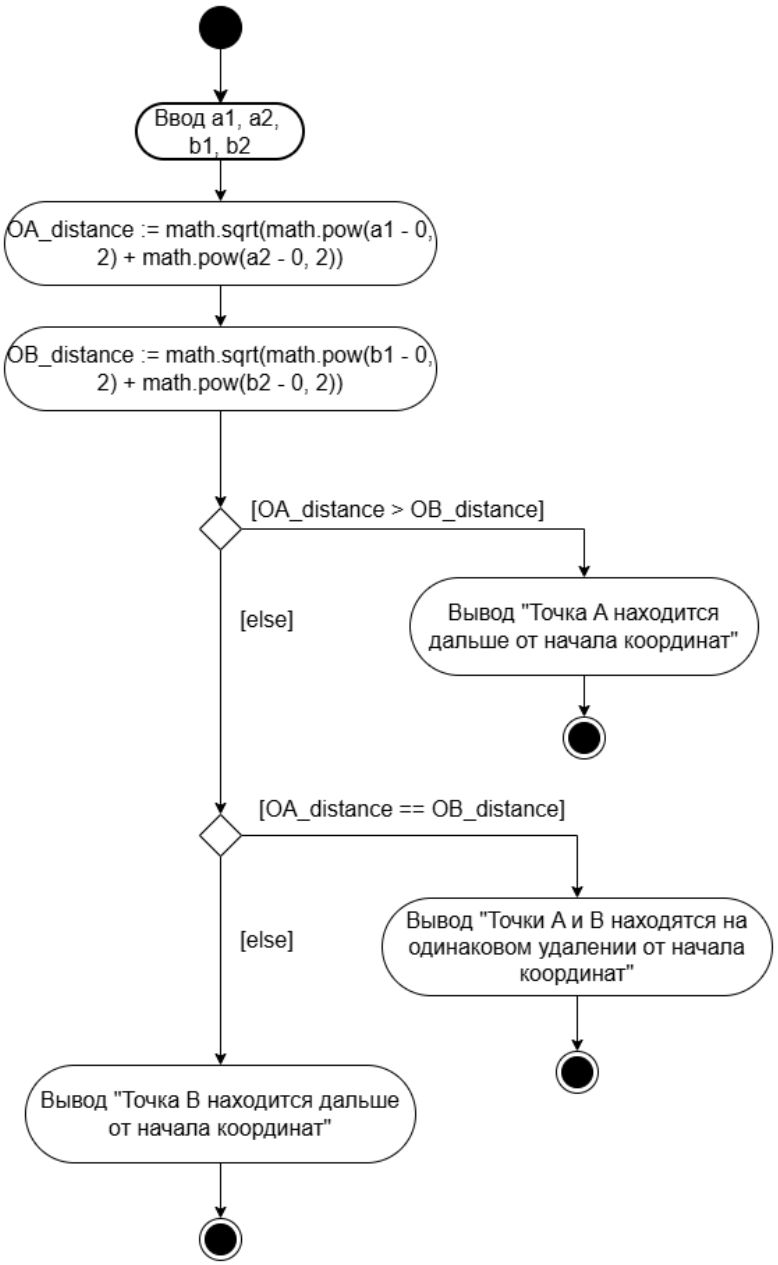

Рисунок 11. UML-диаграмма для индивидуального задания 2

Задание 3. У гусей и кроликов вместе 64 лапы. Сколько могло быть кроликов и гусей (указать все сочетания, которые возможны)

Код программы:

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

TOTAL_PAWS = 64

if __name__ == '__main__':
    for c, i in enumerate(range(0, TOTAL_PAWS + 1, 4), 1):
        goose_count = (TOTAL_PAWS - i) // 2
        rabbit_count = (TOTAL_PAWS - goose_count * 2) // 4
```

```
        print(f"[#{c}] Гусей - {goose_count}, Кроликов -
{rabbit_count}. "
              f"Лап: {goose_count * 2} + {rabbit_count * 4} =
{goose_count * 2 + rabbit_count * 4}")
```

Демонстрация работы программы:

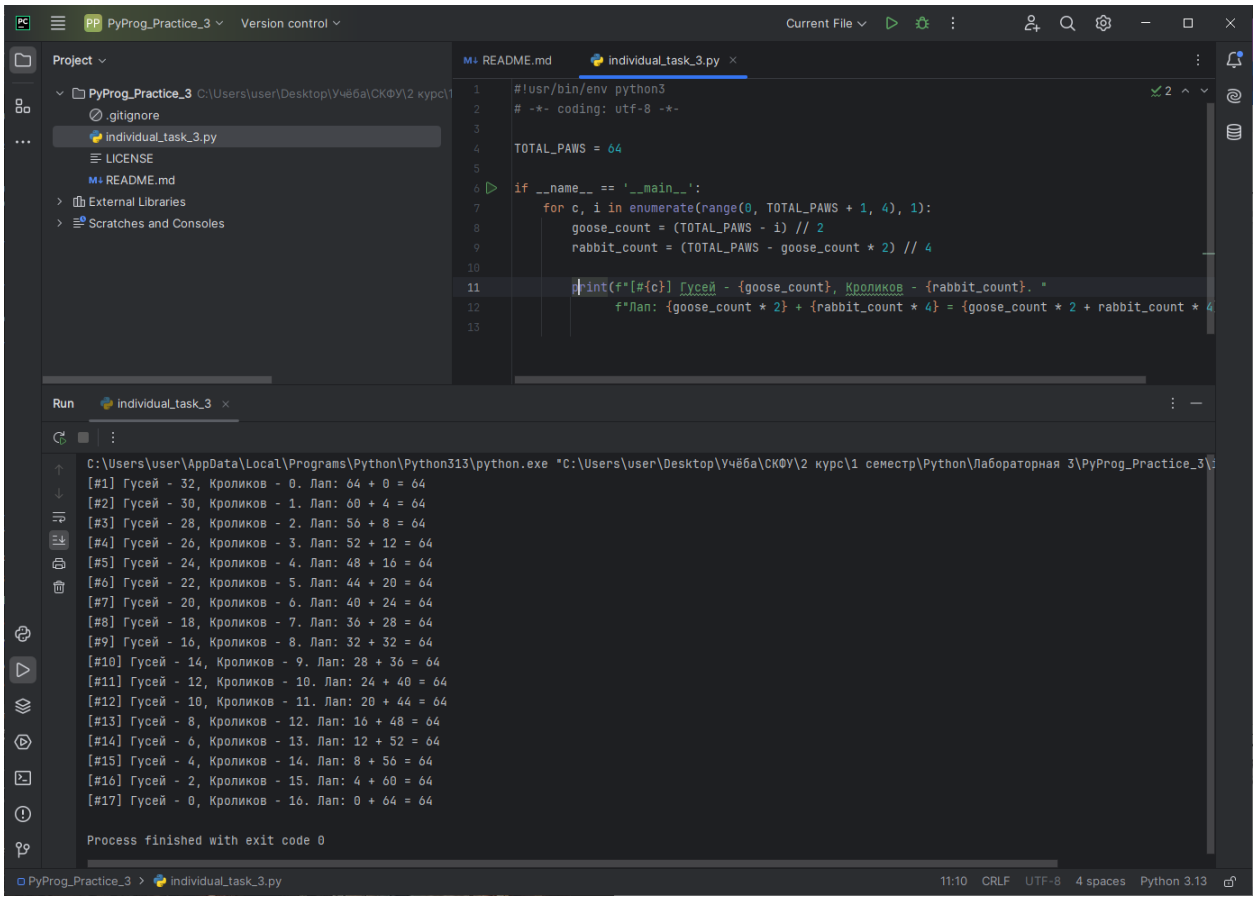

Рисунок 12. Демонстрация работы программы индивидуального задания 3

UML-диаграмма:

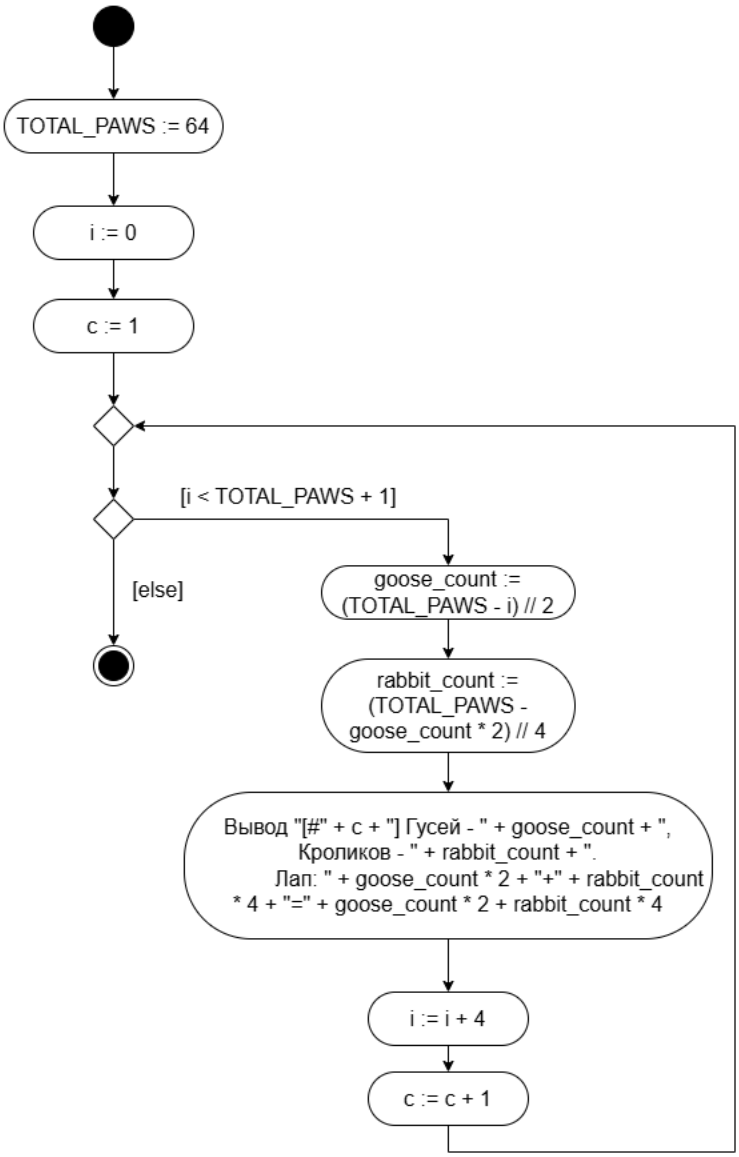

Рисунок 13. UML-диаграмма для индивидуального задания 3

Далее, после выполнения всех примеров и индивидуальных заданий, необходимо внести все изменения в основную ветку согласно модели ветвления git-flow.

Для этого сначала все feature-ветки были слиты в develop:

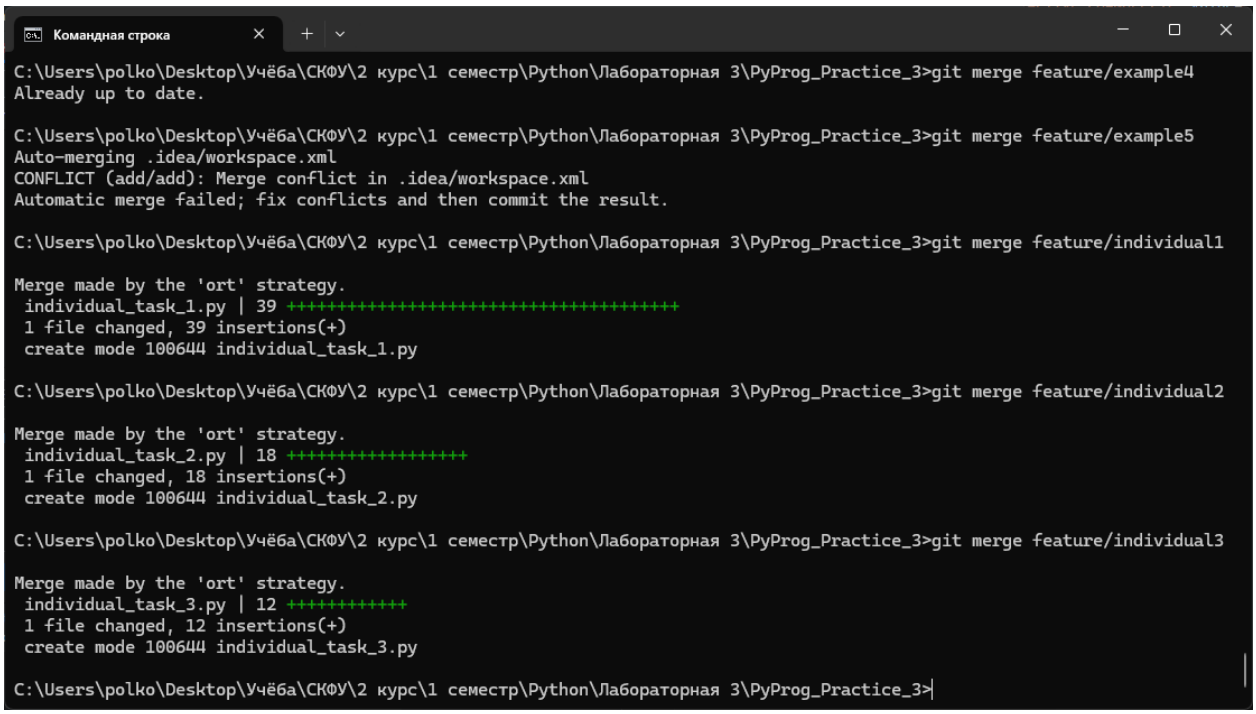

Рисунок 14. Процесс сливания feature-веток в develop

Далее была подготовлена release-ветка:

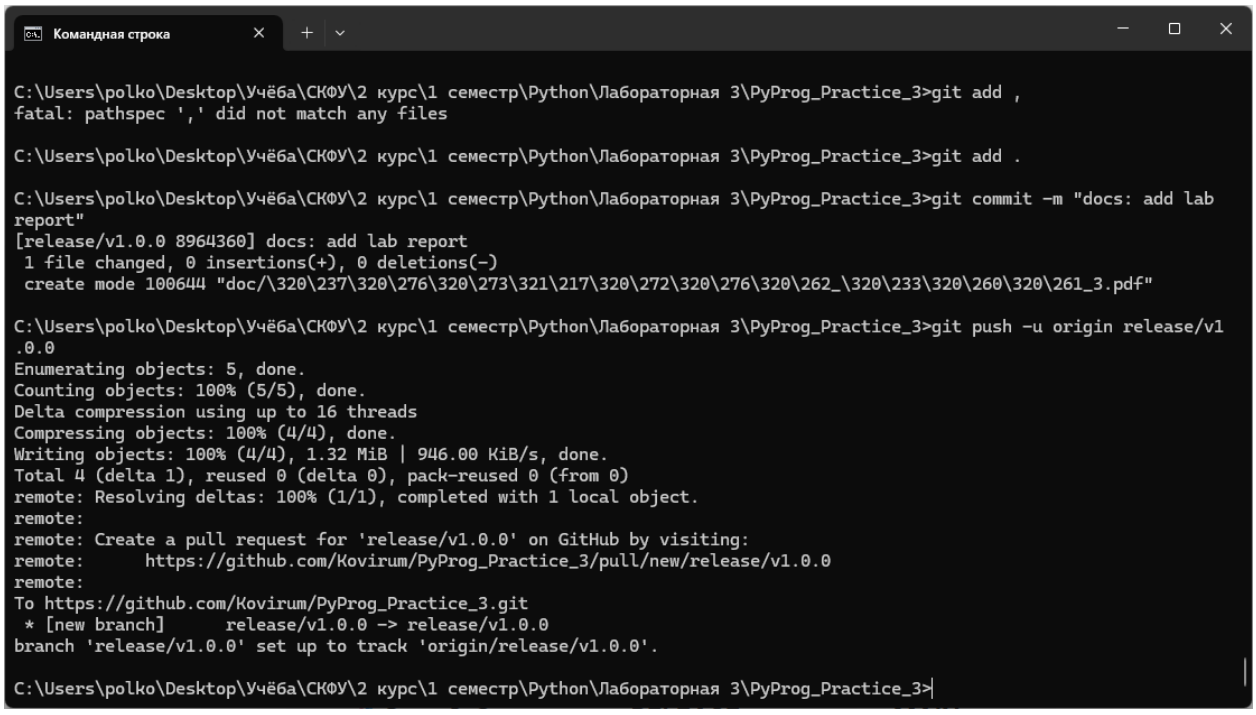

Рисунок 15. Подготовка release-ветки

Далее release-ветка была слита в main-ветку с добавлением соответствующего тега:

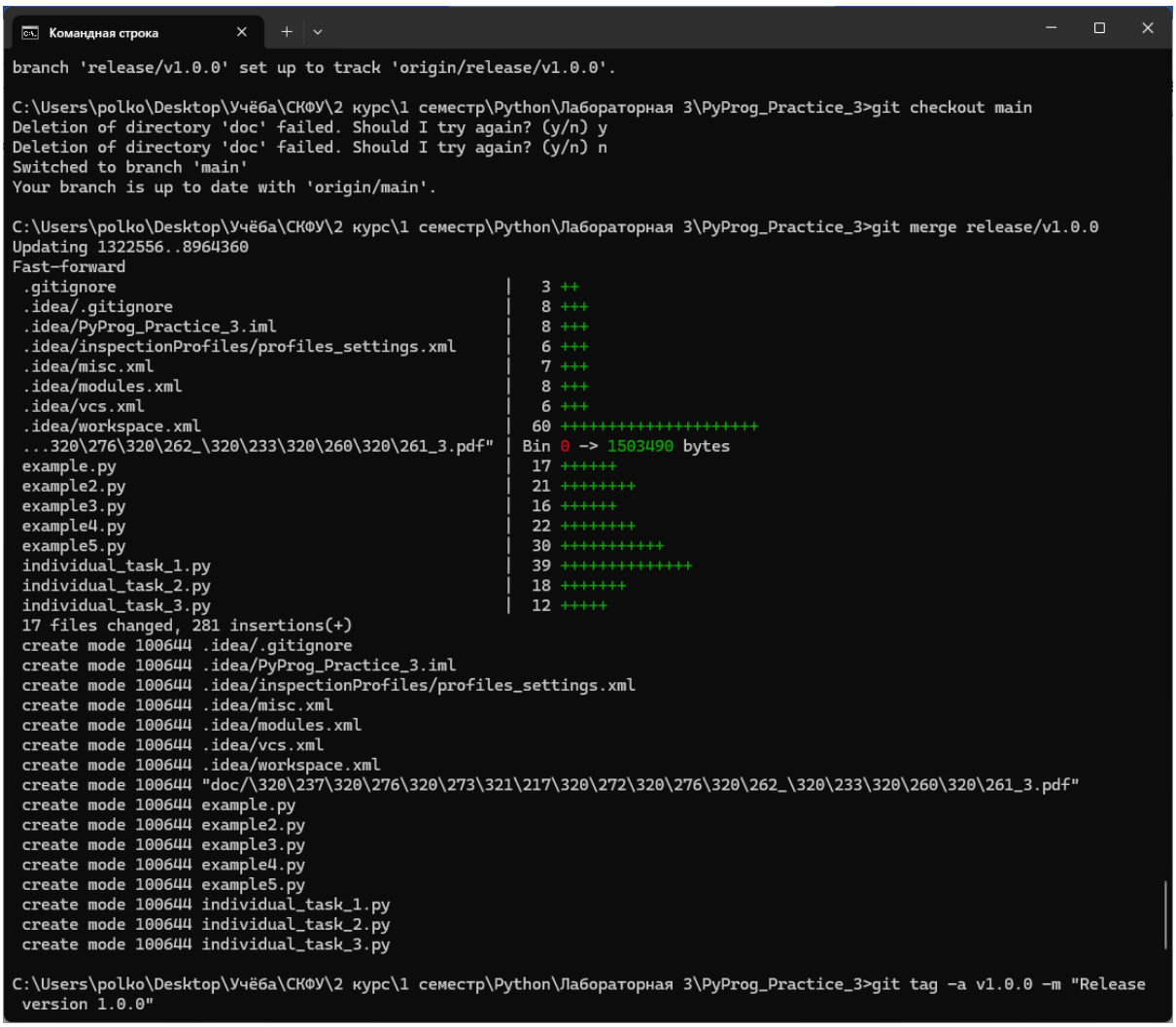

Рисунок 16. Слияние release-ветки с main-веткой и добавление тега

В результате в main ветку были добавлены все работы с соблюдением модели ветвления git-flow:

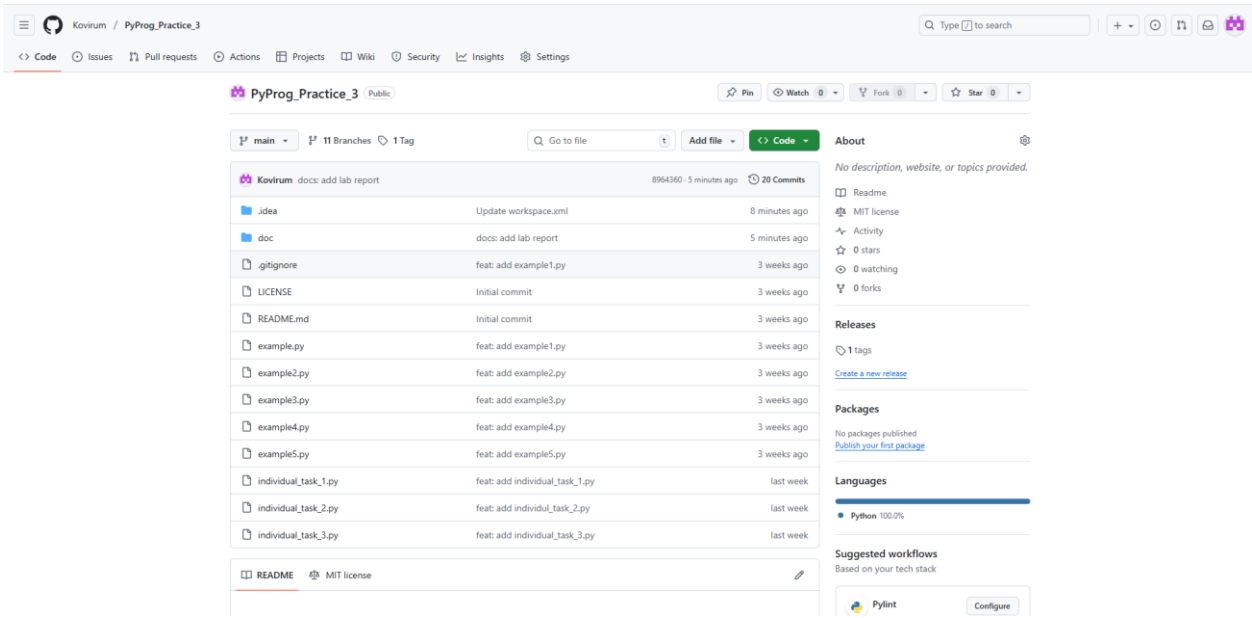

Рисунок 17. Результат работы с ветками

Контрольные вопросы

1. Для чего нужны диаграммы деятельности UML?

Диаграммы деятельности UML нужны для визуального моделирования бизнес-процессов и алгоритмов работы системы. Они отображают последовательность действий и поток управления от одного действия к другому, помогая понять логику поведения системы на разных этапах.

2. Что такое состояние действия и состояние деятельности?

Состояние действия (action state) представляет собой выполнение атомарного, неделимого действия, которое обычно завершается за один шаг. Состояние деятельности (activity state) может быть составным и включать в себя последовательность действий, имеющую внутреннюю структуру и длительность во времени.

3. Какие нотации существуют для обозначения переходов и ветвлений в диаграммах деятельности?

Для переходов в диаграммах деятельности используется стрелка. Для ветвлений используется ромб (символ решения), который имеет один входящий поток управления и несколько исходящих, охраняемых условиями. Условия пишутся в квадратных скобках рядом с исходящей стрелкой.

4. Какой алгоритм является алгоритмом разветвляющейся структуры?

Алгоритм разветвляющейся структуры — это алгоритм, в котором в зависимости от результата проверки условия выполняется одна или другая последовательность действий. Таким образом, ход выполнения программы зависит от истинности некоторого условия.

5. Чем отличается разветвляющийся алгоритм от линейного?

Разветвляющийся алгоритм в отличие от линейного содержит проверку условия, что приводит к выбору одного из нескольких возможных путей выполнения. Линейный алгоритм выполняет действия строго последовательно, без всяких ветвлений.

6. Что такое условный оператор? Какие существуют его формы?

Условный оператор — это конструкция языка программирования, которая позволяет выполнять различные участки кода в зависимости от истинности заданного условия. В Python существуют формы: простая (if), альтернативная (if-else) и множественное ветвление (if-elif-else).

7. Какие операторы сравнения используются в Python?

В Python используются следующие операторы сравнения: == (равно), != (не равно), > (больше), < (меньше), >= (больше или равно), <= (меньше или равно).

8. Что называется простым условием? Приведите примеры.

Простое условие — это одиночное логическое выражение, которое может быть истинным или ложным. Оно образуется с помощью одного оператора сравнения. Примеры: age >= 18, name == "Alice".

9. Что такое составное условие? Приведите примеры.

Составное условие образуется путем объединения нескольких простых условий с помощью логических операторов. Это позволяет проверять более сложную логику. Примеры: age > 18 and age < 65, x < 0 or y < 0.

10. Какие логические операторы допускаются при составлении сложных условий?

При составлении сложных условий в Python допускаются логические операторы: and (логическое И), or (логическое ИЛИ), not (логическое НЕ).

11. Может ли оператор ветвления содержать внутри себя другие ветвления?

Да, оператор ветвления может содержать внутри себя другие ветвления. Это называется вложенными условными операторами. Внутри блока if, else или elif можно разместить еще один условный оператор.

12. Какой алгоритм является алгоритмом циклической структуры?

Алгоритм циклической структуры — это алгоритм, в котором некоторая последовательность действий выполняется многократно (в цикле) до тех пор, пока выполняется заданное условие.

13. Типы циклов в языке Python.

В Python существуют два основных типа циклов: цикл while (с предусловием) и цикл for (для перебора элементов итерируемого объекта). Цикл for часто используется с функцией range() для генерации последовательности чисел.

14. Назовите назначение и способы применения функции range.

Функция range используется для генерации последовательности целых чисел. Она часто применяется в цикле for для выполнения блока кода заданное количество раз. Способы применения: range(stop), range(start, stop), range(start, stop, step). Она создает арифметическую прогрессию.

15. Как с помощью функции range организовать перебор значений от 15 до 0 с шагом 2?

Чтобы организовать перебор значений от 15 до 0 с шагом -2 с помощью функции range, нужно использовать запись: range(15, -1, -2). Start = 15, stop = -1 (так как stop не включается в диапазон, чтобы дойти до 0, нужно указать -1), step = -2.

16. Могут ли быть циклы вложенными?

Да, циклы могут быть вложенными. Это означает, что внутри тела одного цикла (внешнего) может располагаться другой цикл (внутренний). Каждый раз, когда внешний цикл выполняет одну итерацию, внутренний цикл выполняется полностью.

17. Как образуется бесконечный цикл и как выйти из него?

Бесконечный цикл образуется, когда условие выхода из цикла никогда не становится ложным. Например, в цикле while True. Выйти из него можно с помощью оператора break или принудительного прерывания выполнения программы.

18. Для чего нужен оператор break?

Оператор break используется для немедленного выхода из цикла (for или while), прерывая его выполнение, даже если условие завершения цикла еще не стало ложным. Управление передается следующему за циклом оператору.

19. Где употребляется оператор continue и для чего он используется?

Оператор continue употребляется внутри цикла (for или while). Он используется для пропуска оставшейся части кода в текущей итерации цикла и немедленного перехода к следующей итерации (к проверке условия или взятию следующего элемента).

20. Для чего нужны стандартные потоки stdout и stderr?

Стандартные потоки stdout (стандартный вывод) и stderr (стандартный вывод ошибок) используются для вывода информации из программы. stdout обычно применяется для обычного вывода данных, а stderr — для вывода

сообщений об ошибках, что позволяет разделять их при перенаправлении вывода.

21. Как в Python организовать вывод в стандартный поток stderr?

В Python для вывода в стандартный поток ошибок stderr можно использовать модуль sys и его атрибут stderr. Например: import sys и затем sys.stderr.write("Сообщение об ошибке\n") или print("Ошибка", file=sys.stderr).

22. Каково назначение функции exit?

Назначение функции exit (из модуля sys) — завершить выполнение программы немедленно. При вызове она может принять необязательный аргумент — код возврата (ноль обычно означает успешное завершение, а ненулевое значение — ошибку), который может быть проверен вызывающей средой или операционной системой.

Вывод

В результате выполнения данной лабораторной работы были приобретены навыки программирования разветвляющихся алгоритмов и алгоритмов циклической структуры. Освоены операторы языка Python версии 3.x if, while, for, break и continue, позволяющие реализовывать разветвляющиеся алгоритмы и алгоритмы циклической структуры.