

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 №5
дисциплины «Основы программной инженерии»

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

(подпись)

Проверил Воронкин Роман Александрович

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

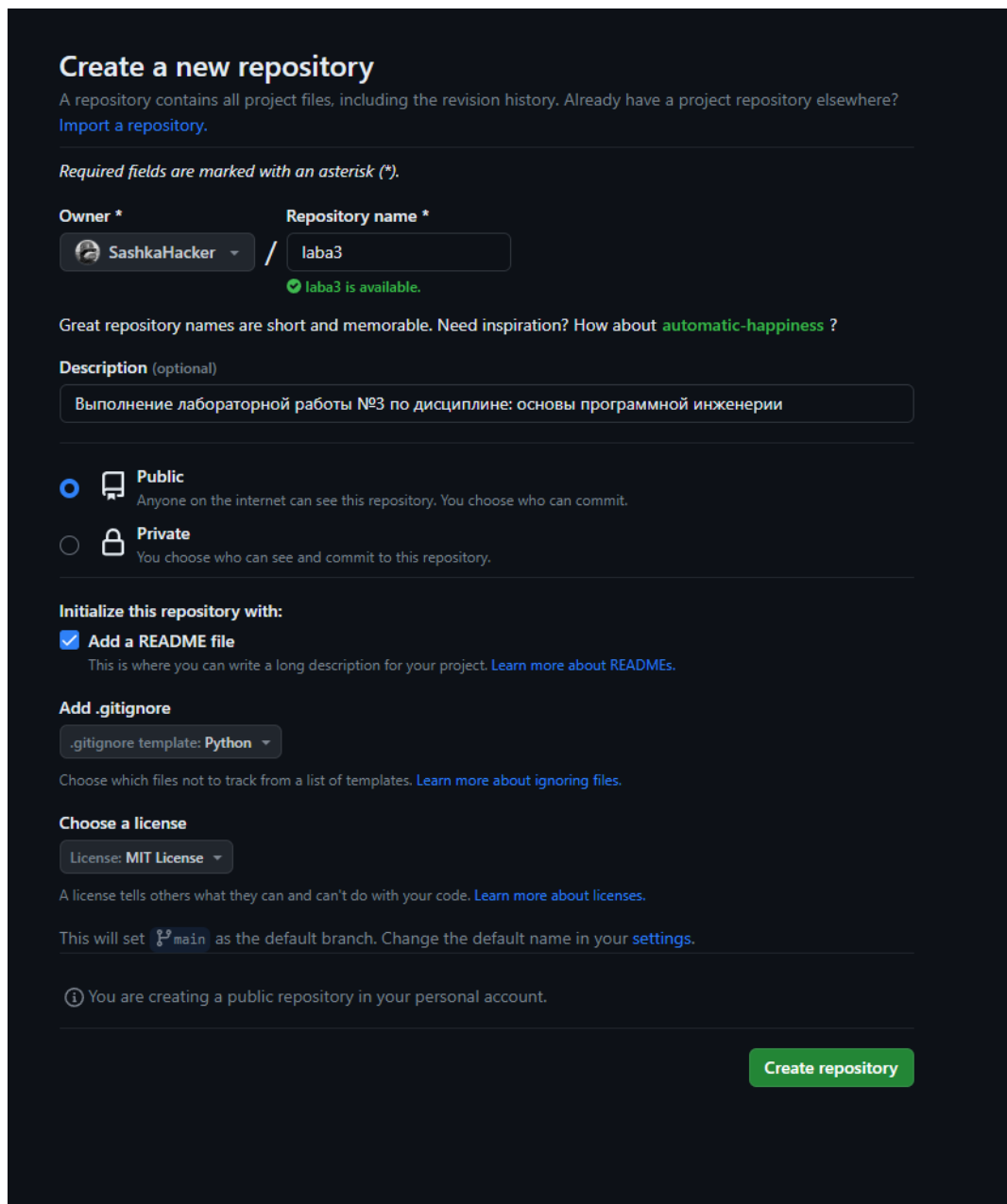
Ставрополь, 2023 г.

Тема: Условные операторы и циклы в языке Python.

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

Ход работы.

1. Создание нового репозитория с лицензией MIT.



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 * SashkaHacker / **Repository name *** laba3
✔ laba3 is available.

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

Description (optional)
Выполнение лабораторной работы №3 по дисциплине: основы программной инженерии

☒ **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

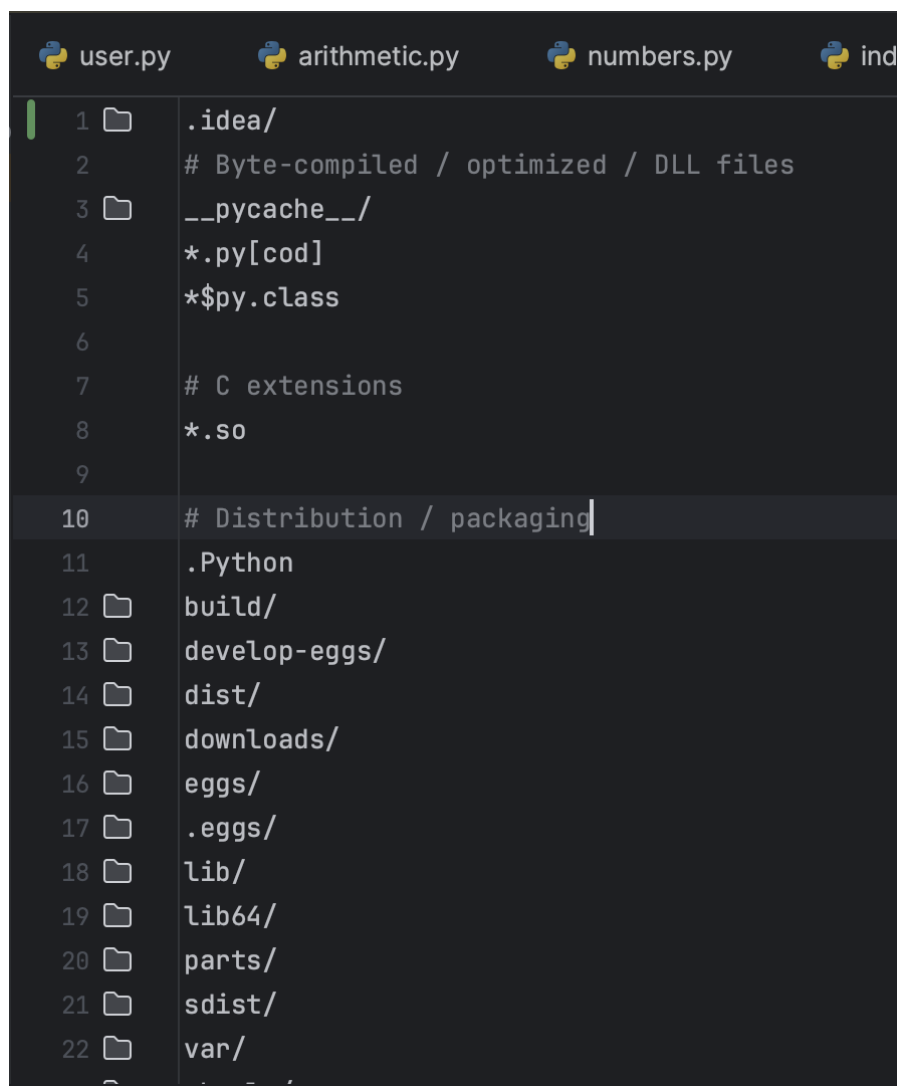
Рисунок 1 – Создание репозитория

2. Клонировал репозиторий на рабочий ПК.

```
Sashka@DESKTOP-U4RPSBI MINGW64 ~/Documents/GitHub
$ git clone https://github.com/SashkaHacker/laba5.git
Cloning into 'laba5'...
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.
```

Рисунок 2 – Клонирование репозитория

3. Дополнил файл .gitignore необходимыми инструкциями.



```
user.py arithmetic.py numbers.py ind
1  .idea/
2  # Byte-compiled / optimized / DLL files
3  __pycache__/
4  *.py[cod]
5  *$py.class
6
7  # C extensions
8  *.so
9
10 # Distribution / packaging
11 .Python
12 build/
13 develop-eggs/
14 dist/
15 downloads/
16 eggs/
17 .eggs/
18 lib/
19 lib64/
20 parts/
21 sdist/
22 var/
```

Рисунок 3 – Файл .gitignore

4. Выполнение примеров (1-3) лабораторной работы (рисунки 4 – 9).

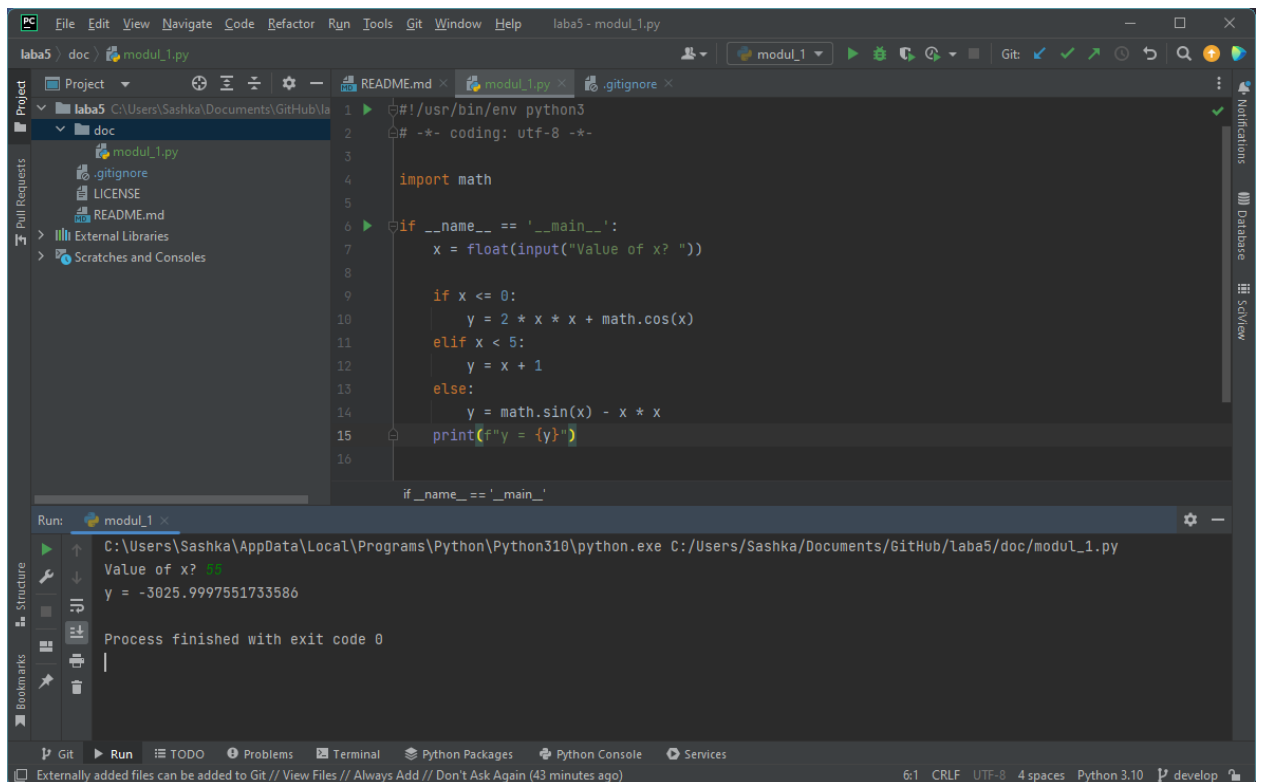


Рисунок 4 – Обработка примера №1 с различными входными данными

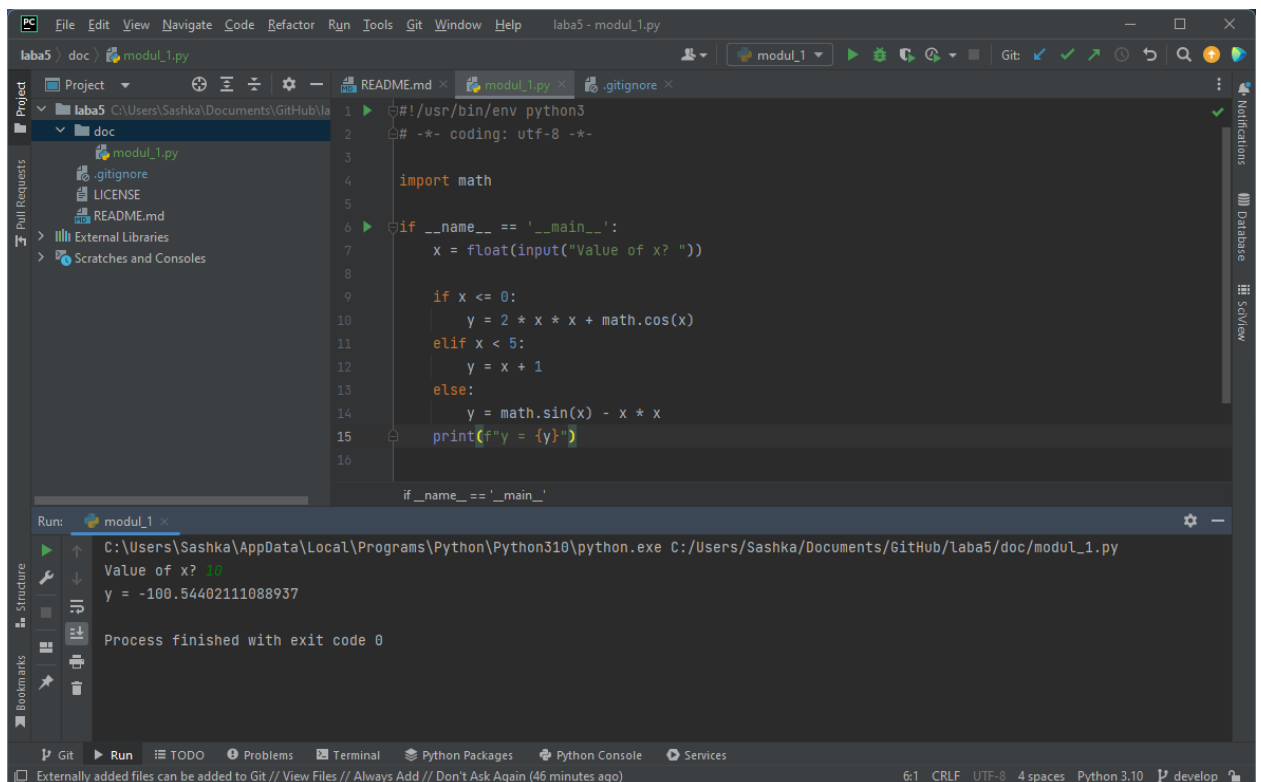


Рисунок 5 – Обработка примера №1 с различными входными данными

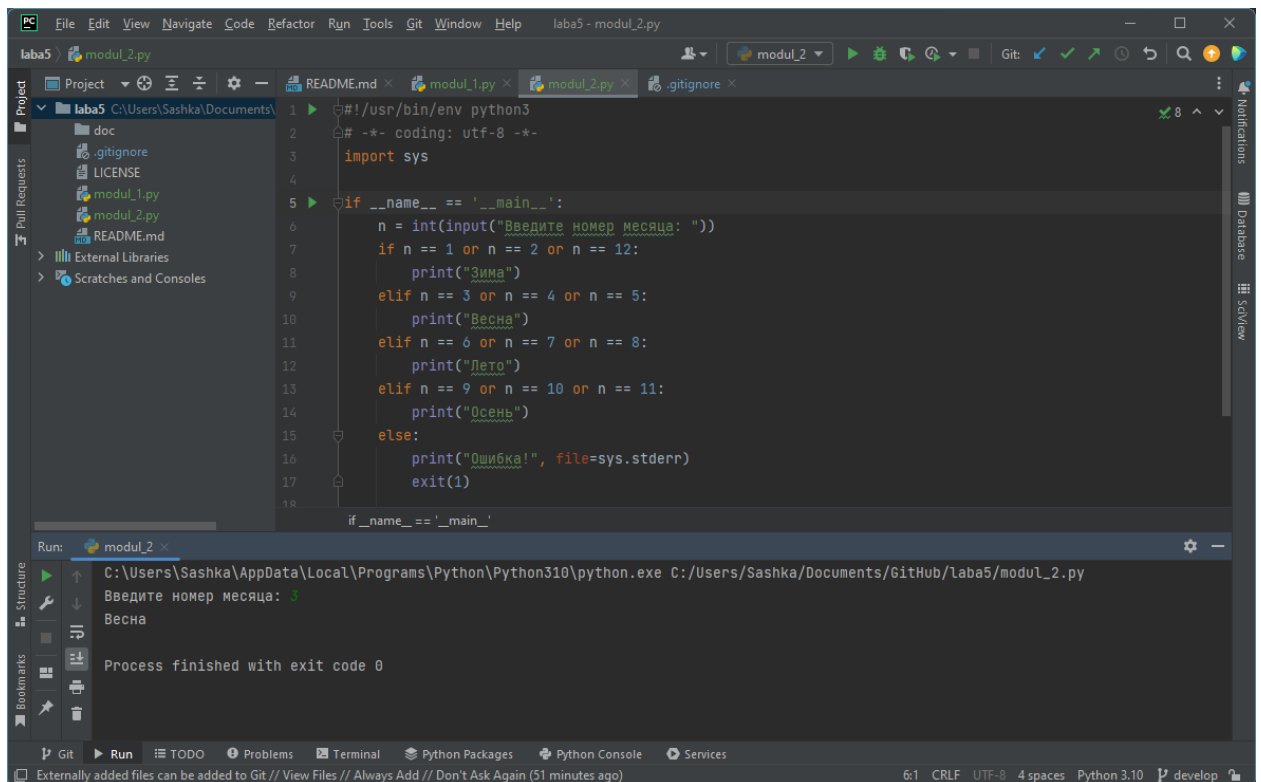


Рисунок 6 – Отработка примера №2 с различными входными данными

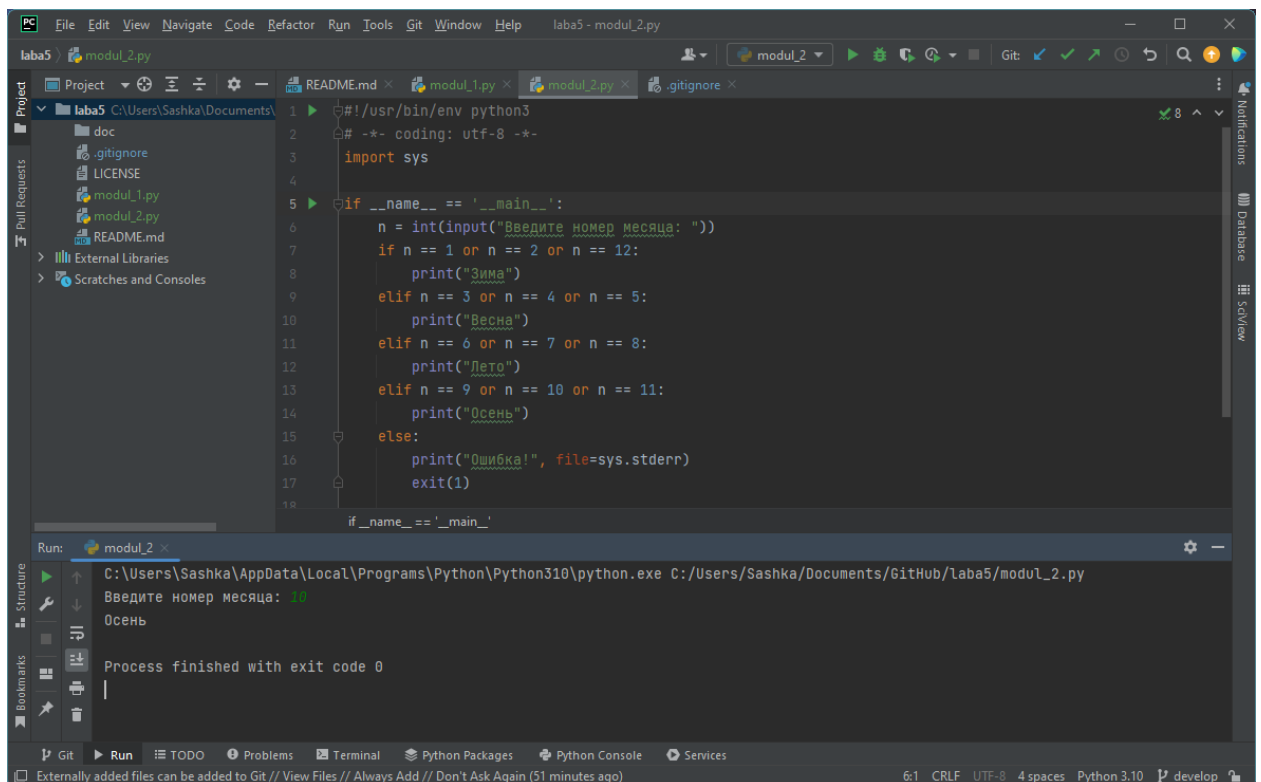


Рисунок 7 – Отработка примера №2 с различными входными данными

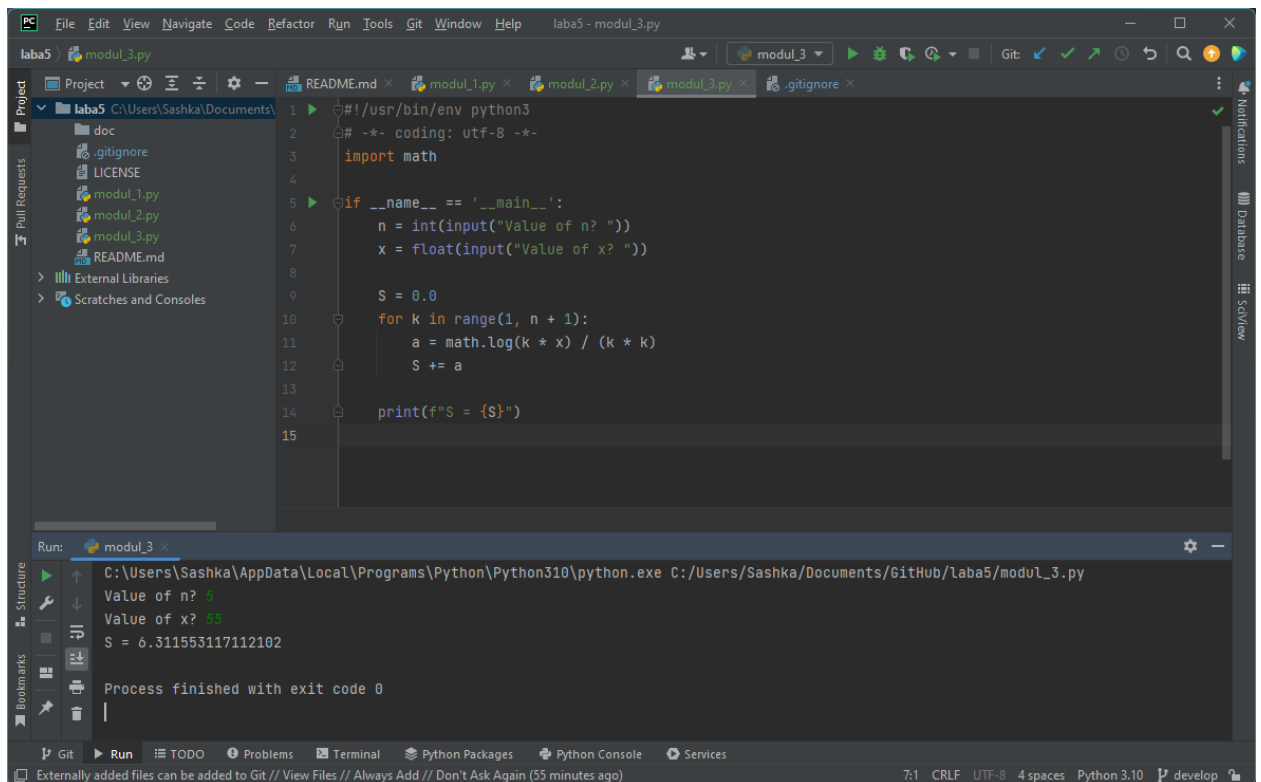


Рисунок 8 – Отработка примера №3 с различными входными данными

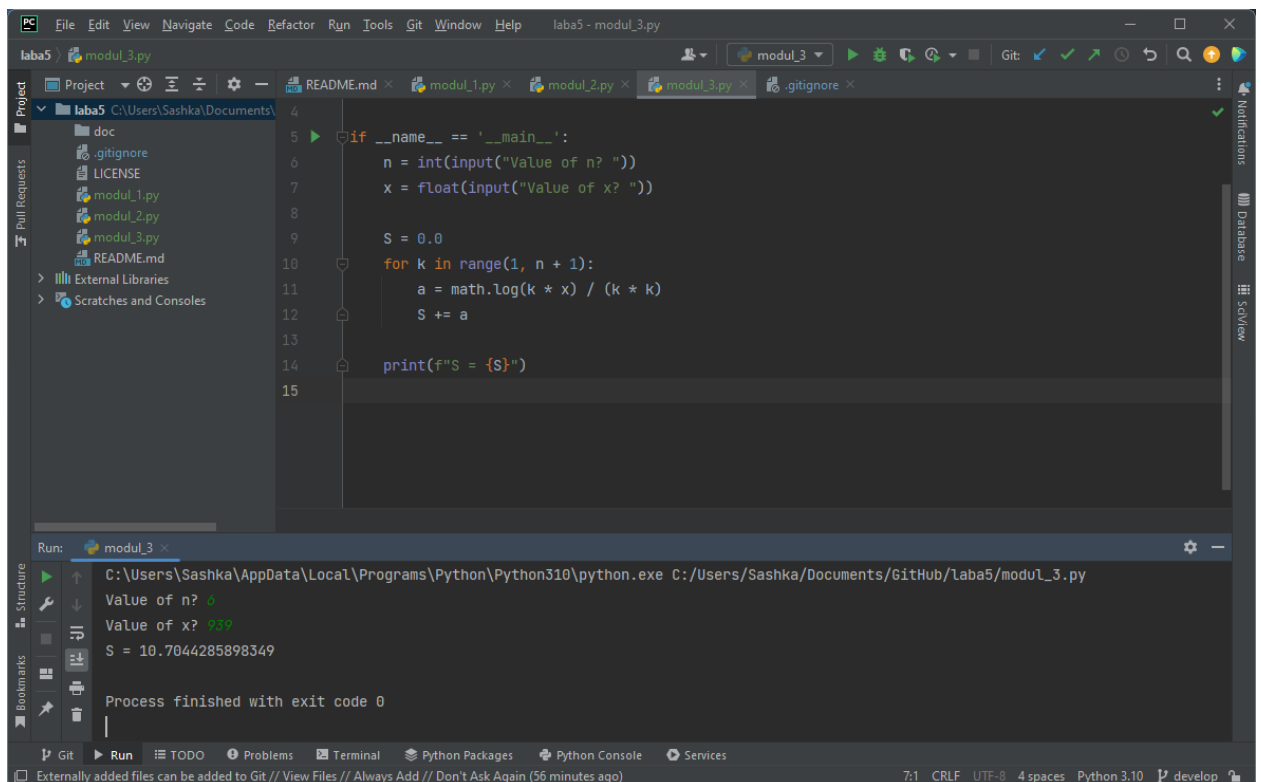


Рисунок 9 – Отработка примера №3 с различными входными данными

5. Выполнение примеров (4-5), а также построение к ним UML-диаграмм (рисунки 10 -)

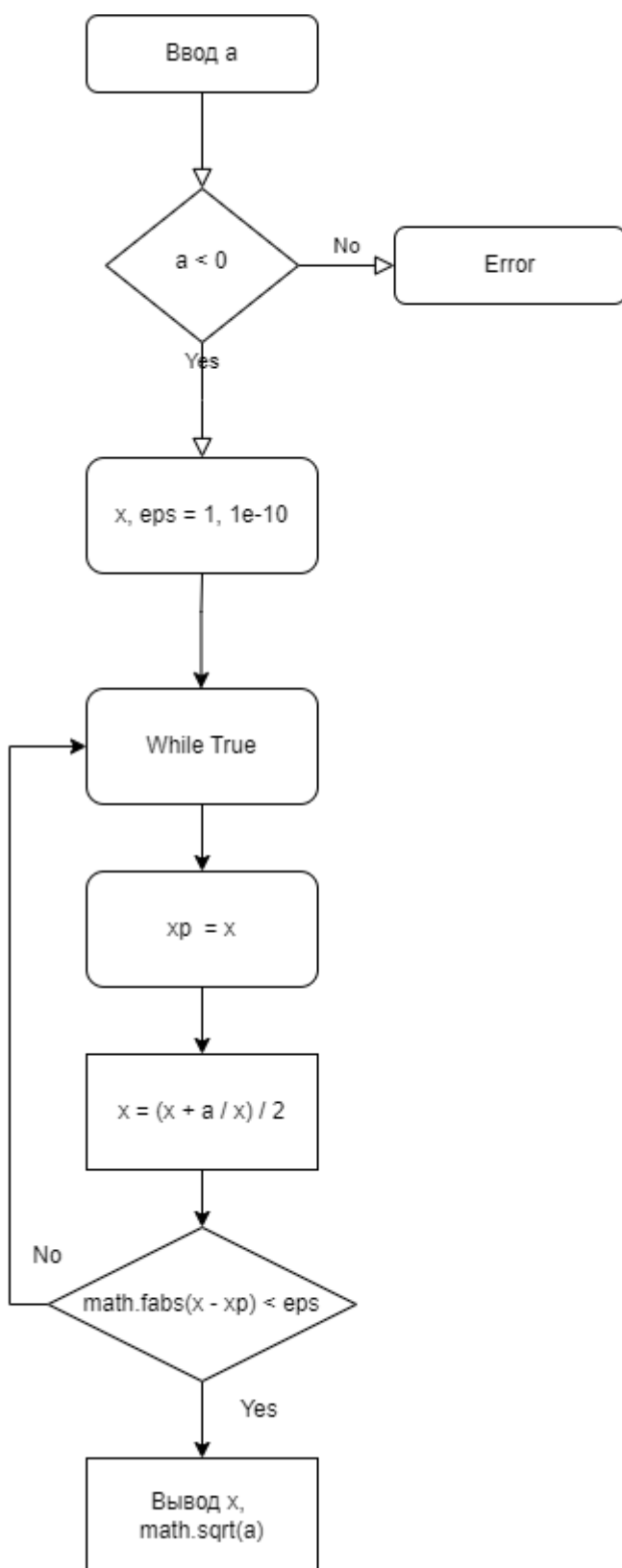


Рисунок 10 – Диаграмма для примера №4

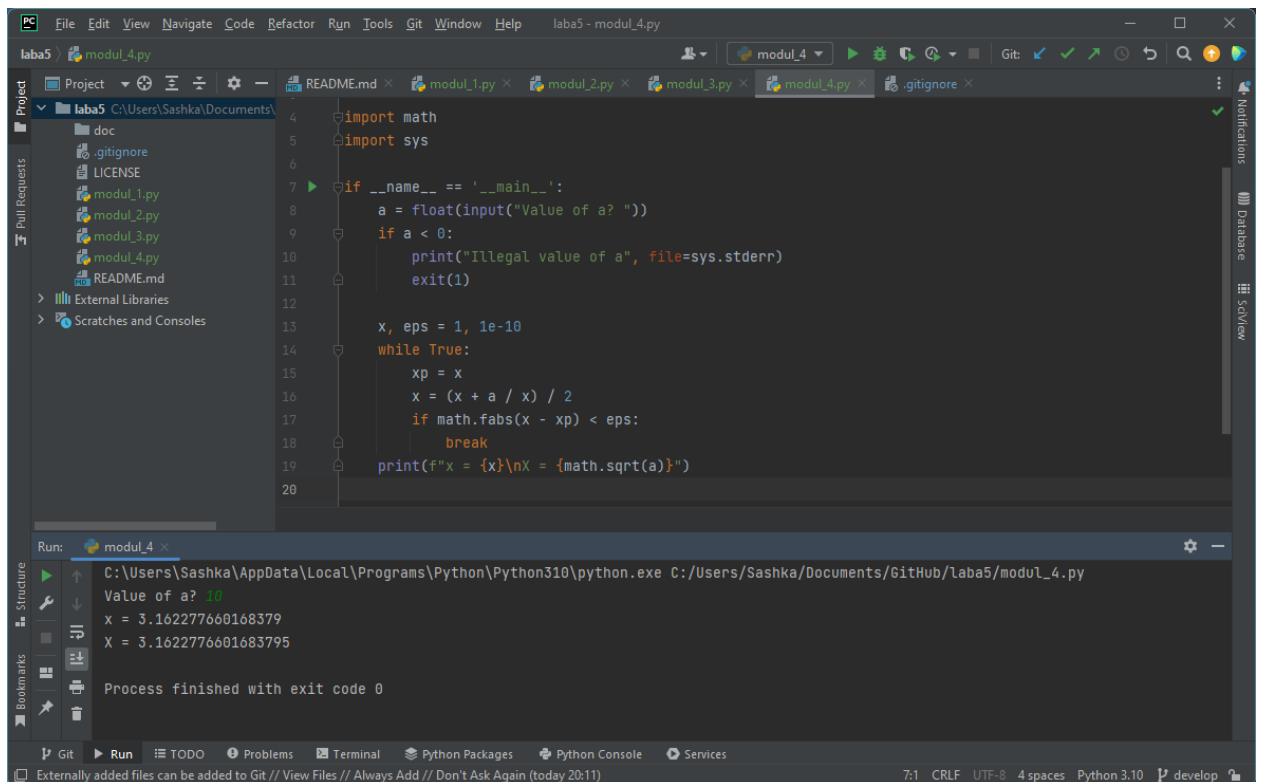


Рисунок 11 – Отработка примера №4 с различными входными данными

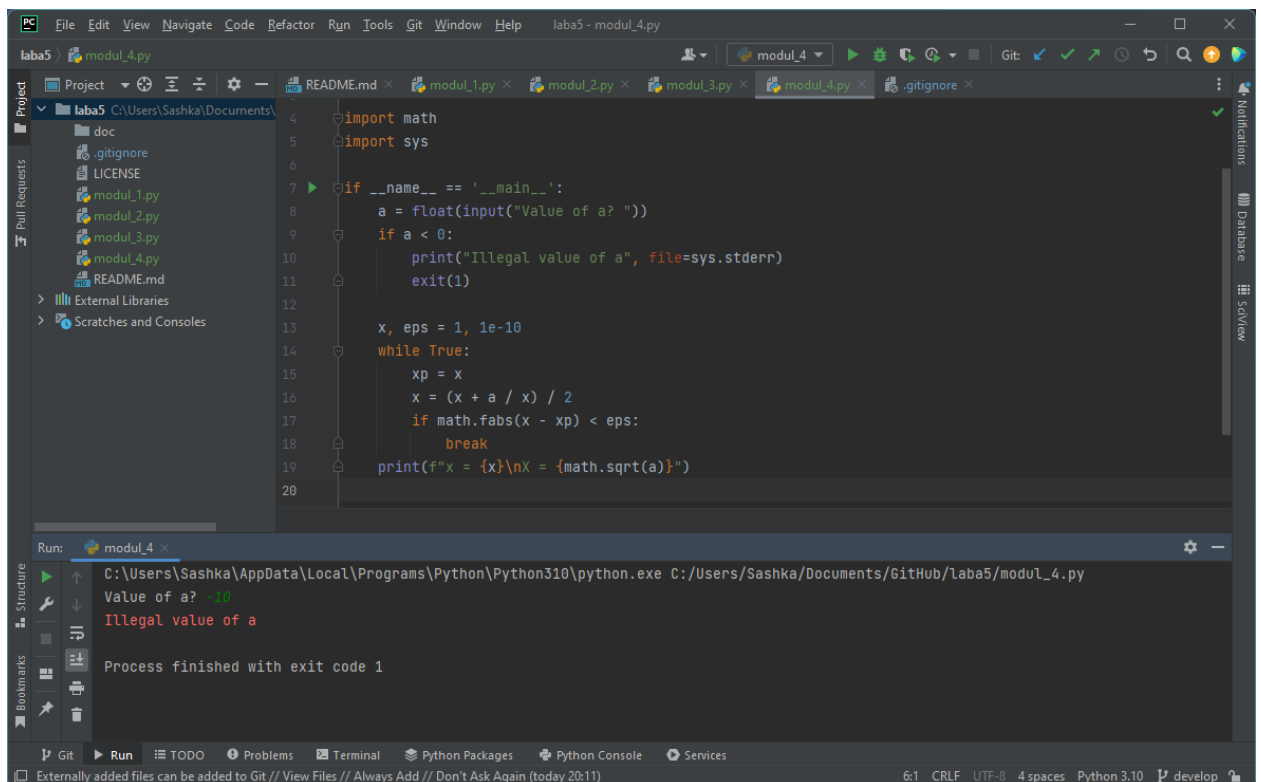


Рисунок 12 – Отработка примера №4 с различными входными данными

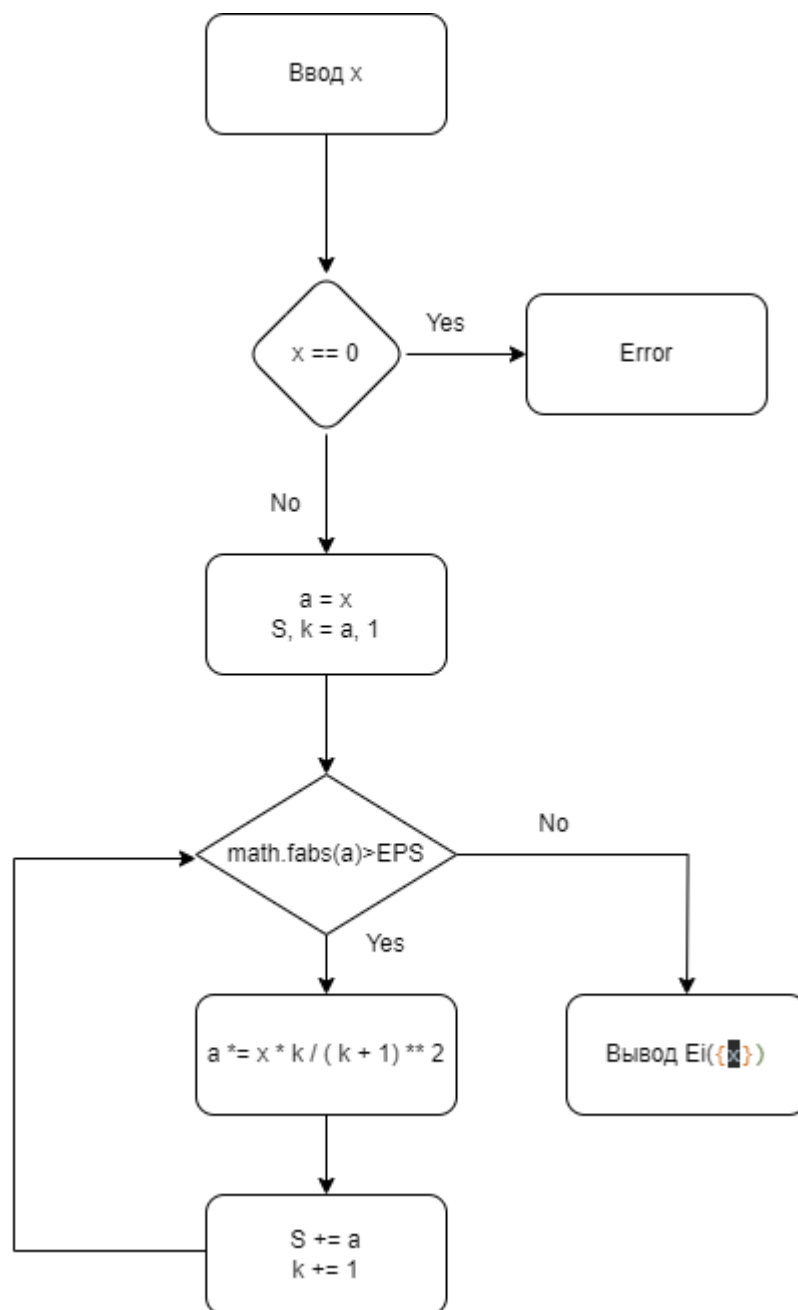


Рисунок 13 – Диаграмма для примера №5

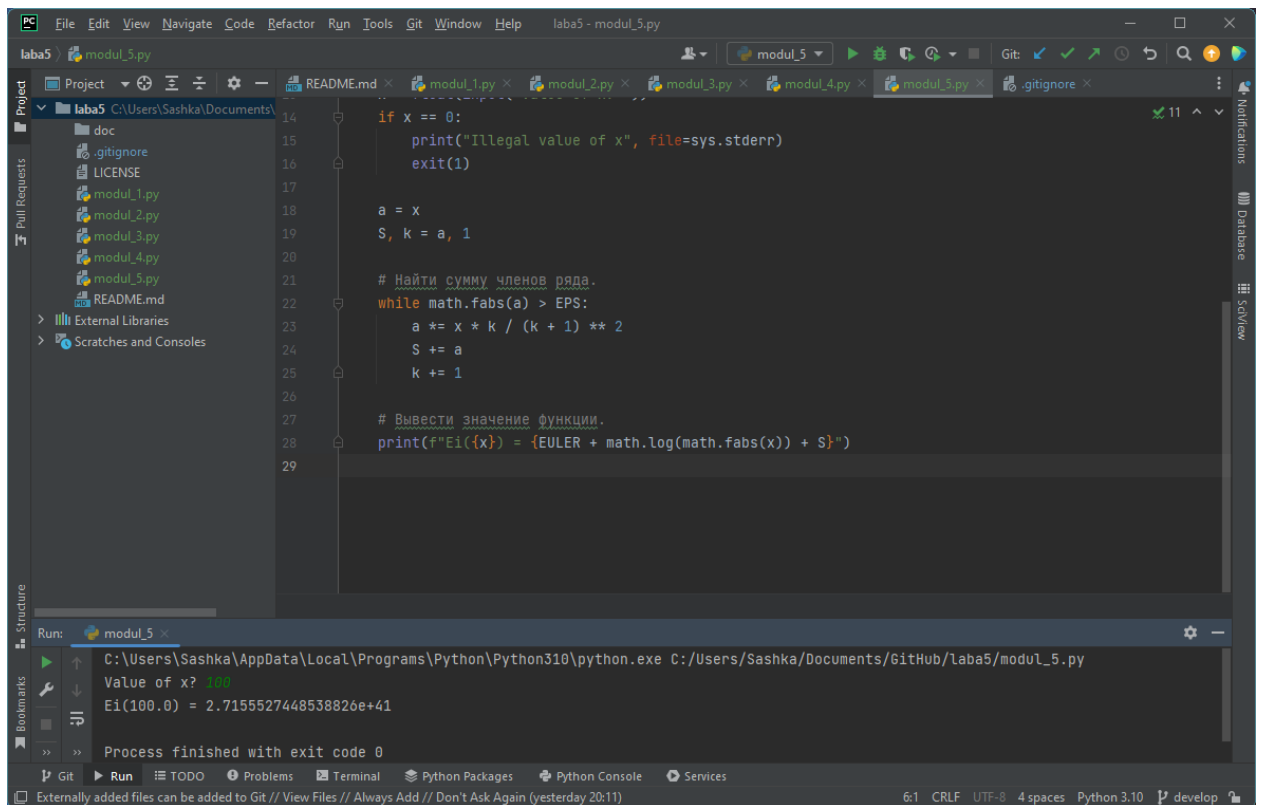


Рисунок 14 – Отработка примера №5 с различными входными данными

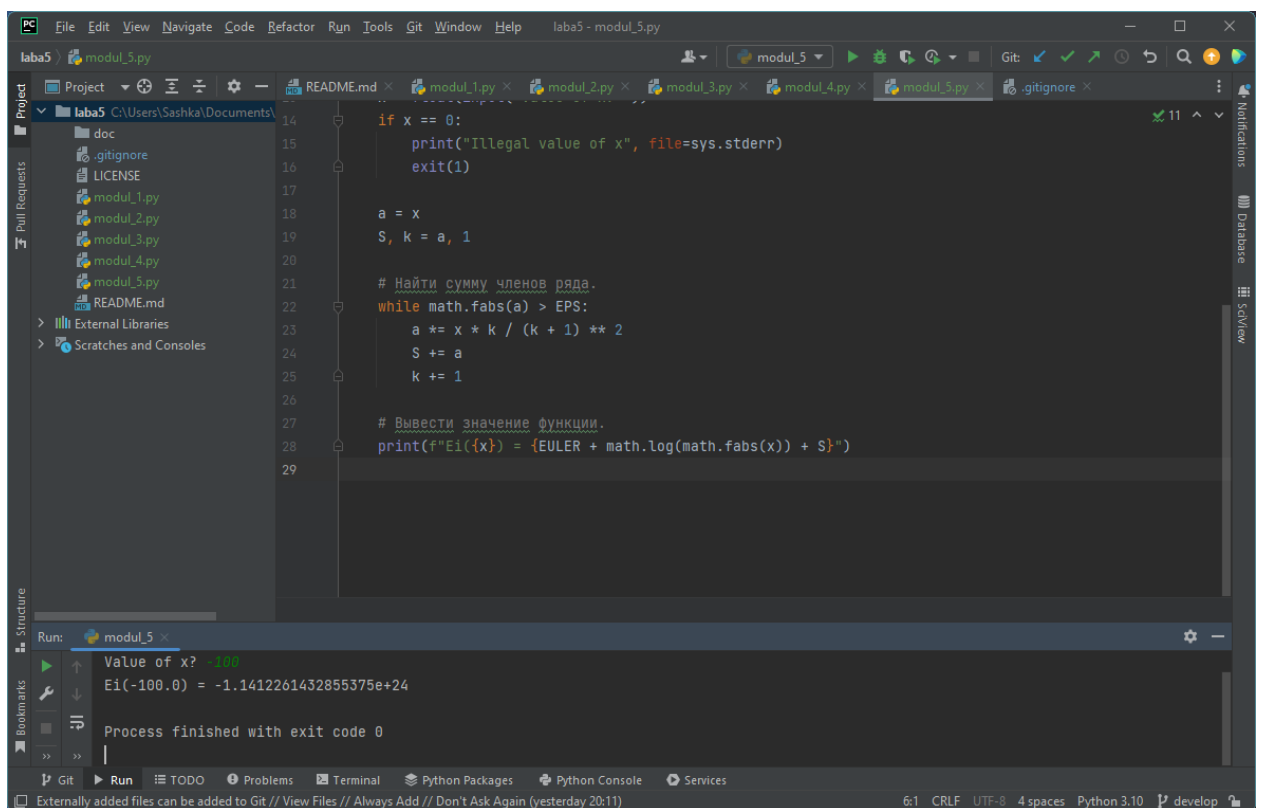


Рисунок 15 – Отработка примера №5 с различными входными данными

6. Составим UML-диаграмму и программу для решения задания №1
(вариант – 11).

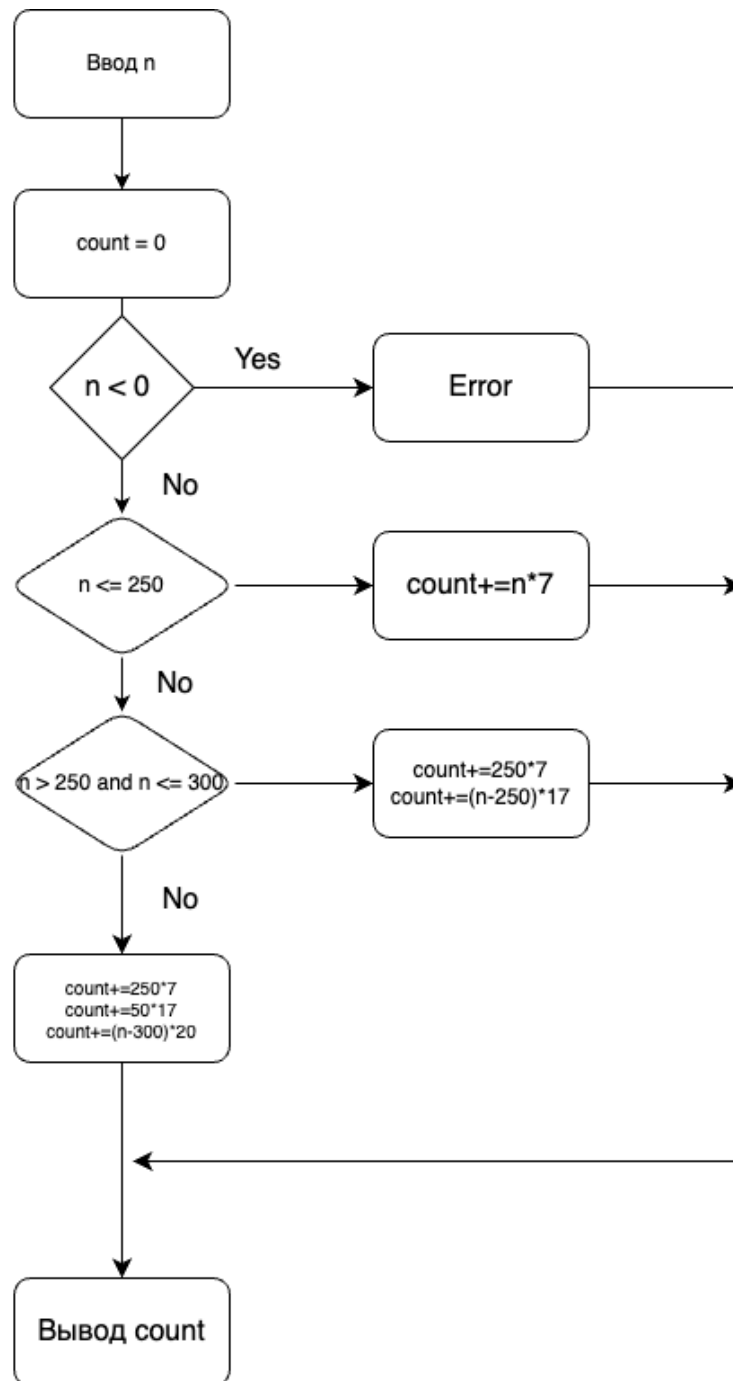
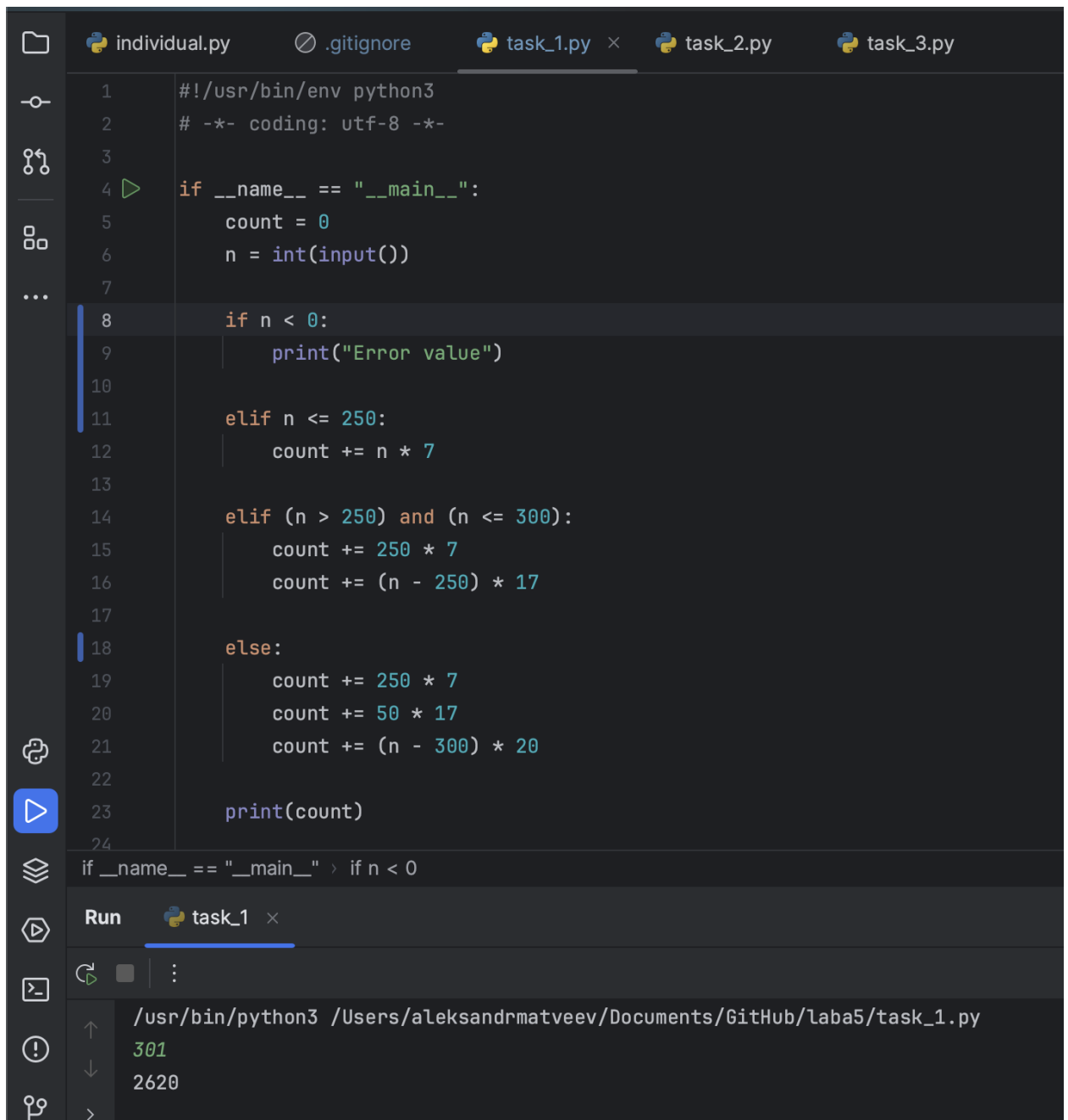


Рисунок 16 – UML-диаграмма для решения задания №1



```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  if __name__ == "__main__":
5      count = 0
6      n = int(input())
7
8      if n < 0:
9          print("Error value")
10
11      elif n <= 250:
12          count += n * 7
13
14      elif (n > 250) and (n <= 300):
15          count += 250 * 7
16          count += (n - 250) * 17
17
18      else:
19          count += 250 * 7
20          count += 50 * 17
21          count += (n - 300) * 20
22
23      print(count)
24
```

if __name__ == "__main__" > if n < 0

Run task_1 ×

/usr/bin/python3 /Users/aleksandrmatveev/Documents/GitHub/laba5/task_1.py

301

2620

Рисунок 17 – Решение задания №1

7. Составим UML-диаграмму и программу для решения задания №2 (вариант – 11).

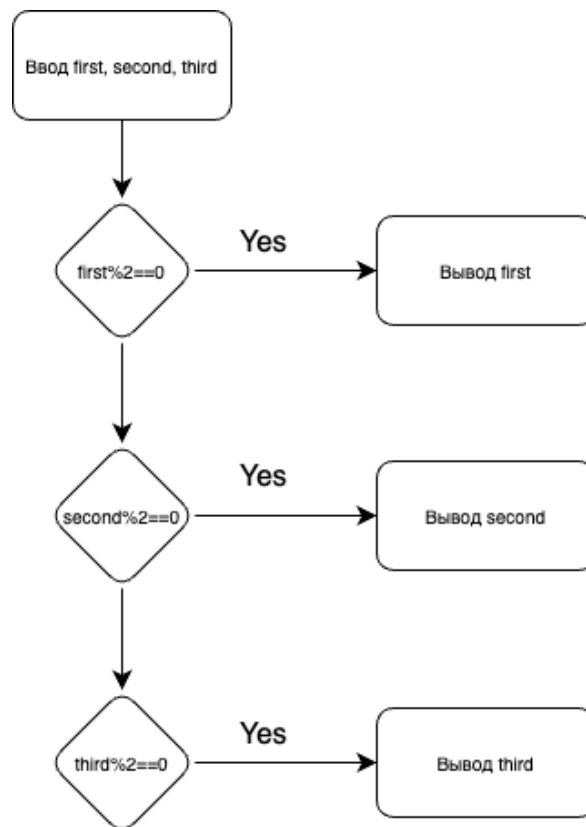


Рисунок 18 – UML-диаграмма для решения задания №2

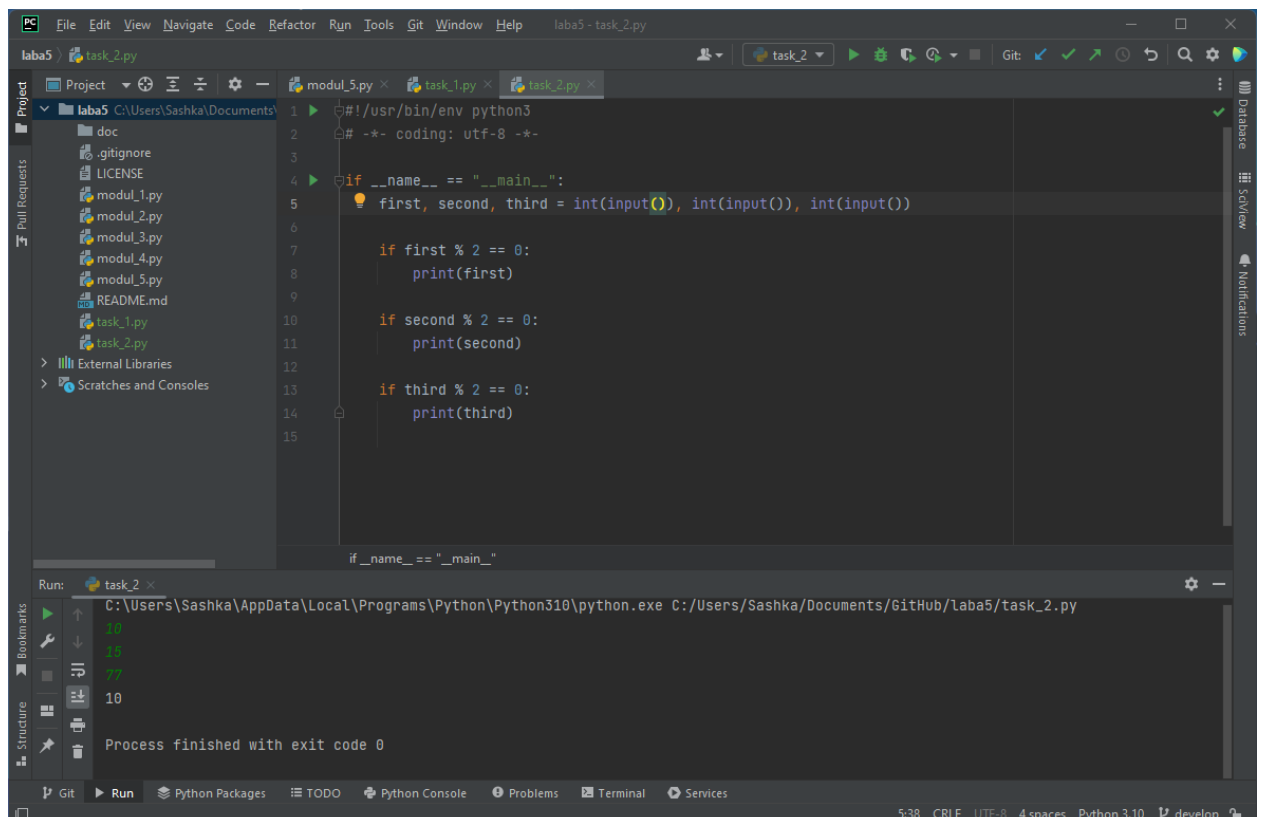


Рисунок 19 – Решение задания №2

8. Составим UML-диаграмму и программу для решения задания №3 (вариант – 11).

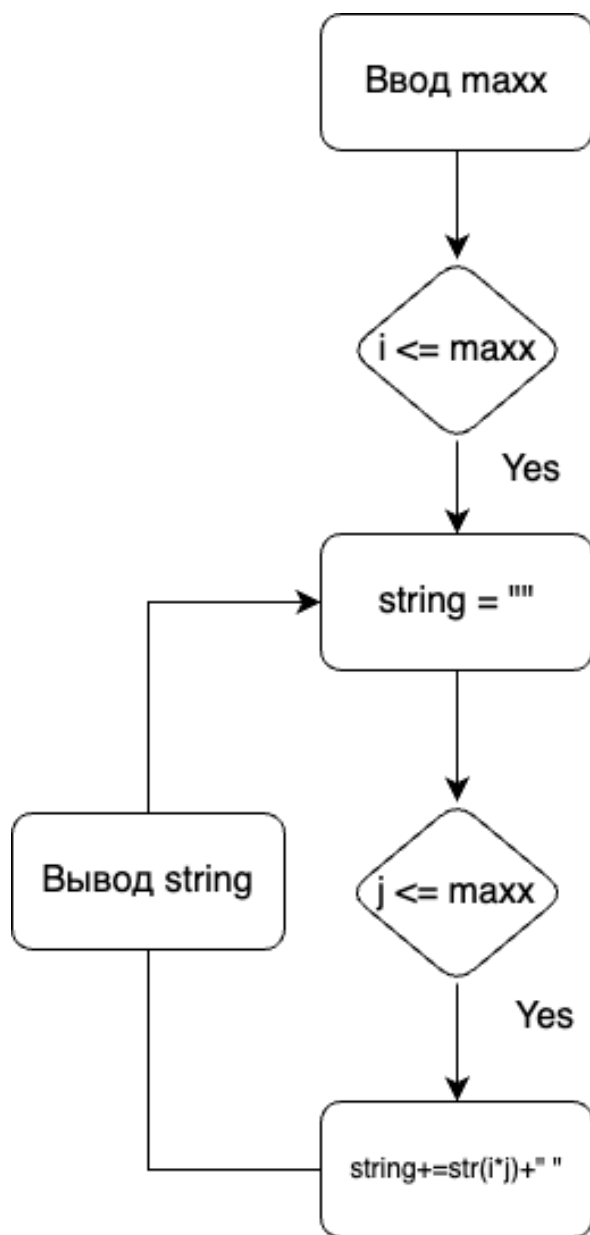


Рисунок 20 – UML-диаграмма для решения задания №3

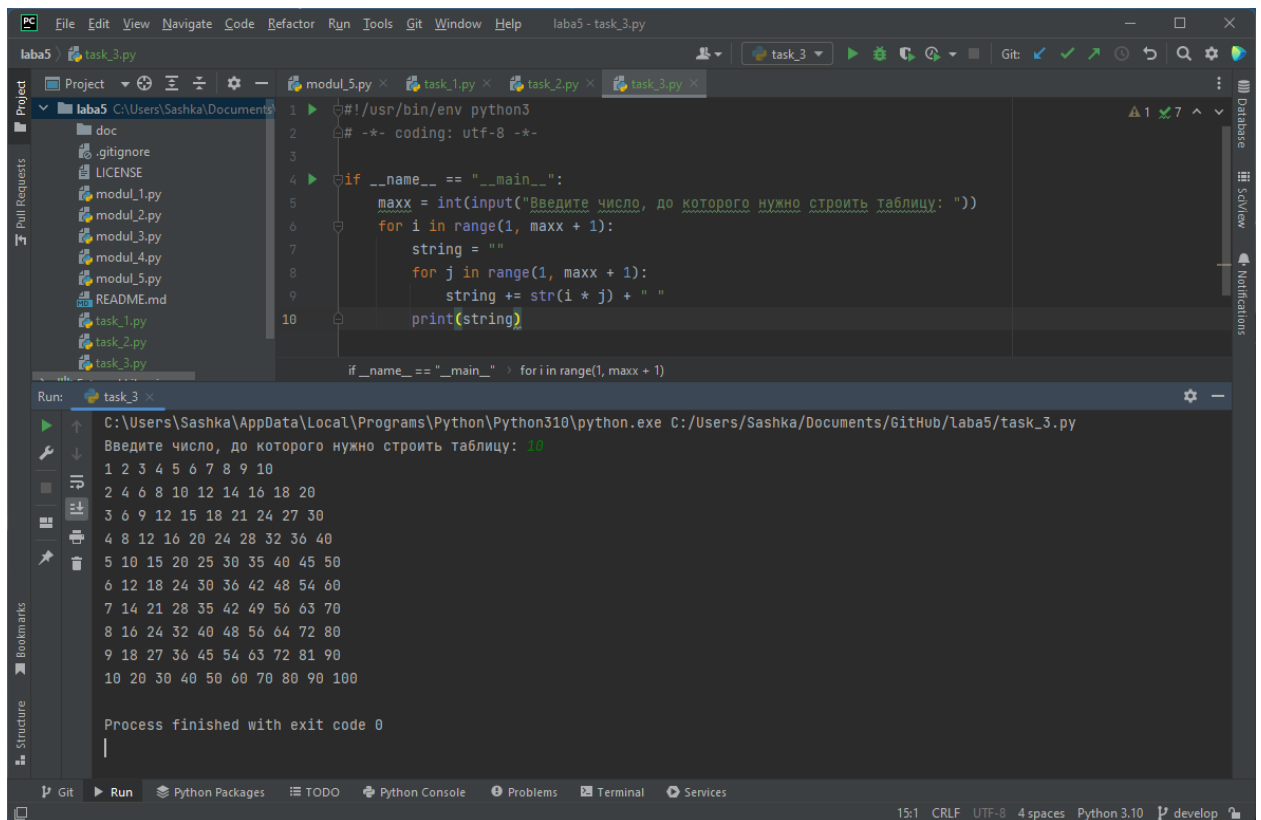


Рисунок 21 – Решение задания №3

9. Составим UML-диаграмму и программу для решения задания повышенной сложности (вариант - 2).

$$\text{Ci}(x) = \gamma + \ln x + \int_0^x \frac{\cos t - 1}{t} dt = \gamma + \ln x + \sum_{n=1}^{\infty} \frac{(-1)^n x^{2n}}{(2n)(2n)!}.$$

Рисунок 22 – Условие задачи

Чтобы вычислить сумму ряда найдем рекуррентное соотношение, позволяющее определить следующий член ряда исходя из значения текущего. Для этого разделим следующий член ряда на текущий. Текущий член ряда задается выражением:

$$a_n = \frac{(-1)^n x^{2n}}{(2n)(2n)!}$$

Тогда как следующий член ряда может быть определен следующим образом:

$$a_{n+1} = \frac{(-1)^{n+1} x^{2(n+1)}}{(2(n+1))(2(n+1))!}$$

Найдем отношение следующего и текущего членов ряда:

$$\frac{a_{n+1}}{a_n} = - \frac{x^2 * n}{4n^3 + 10n^2 + 8n + 2}$$

Следовательно:

$$a_{n+1} = - \frac{x^2 * n}{4n^3 + 10n^2 + 8n + 2} * a_n$$

Помимо выражения, связывающего a_n и a_{n+1} , для вычисления значения рекуррентного соотношения необходимо найти значение первого члена ряда. В данном случае:

$$a_1 = - \frac{x^4}{4}$$

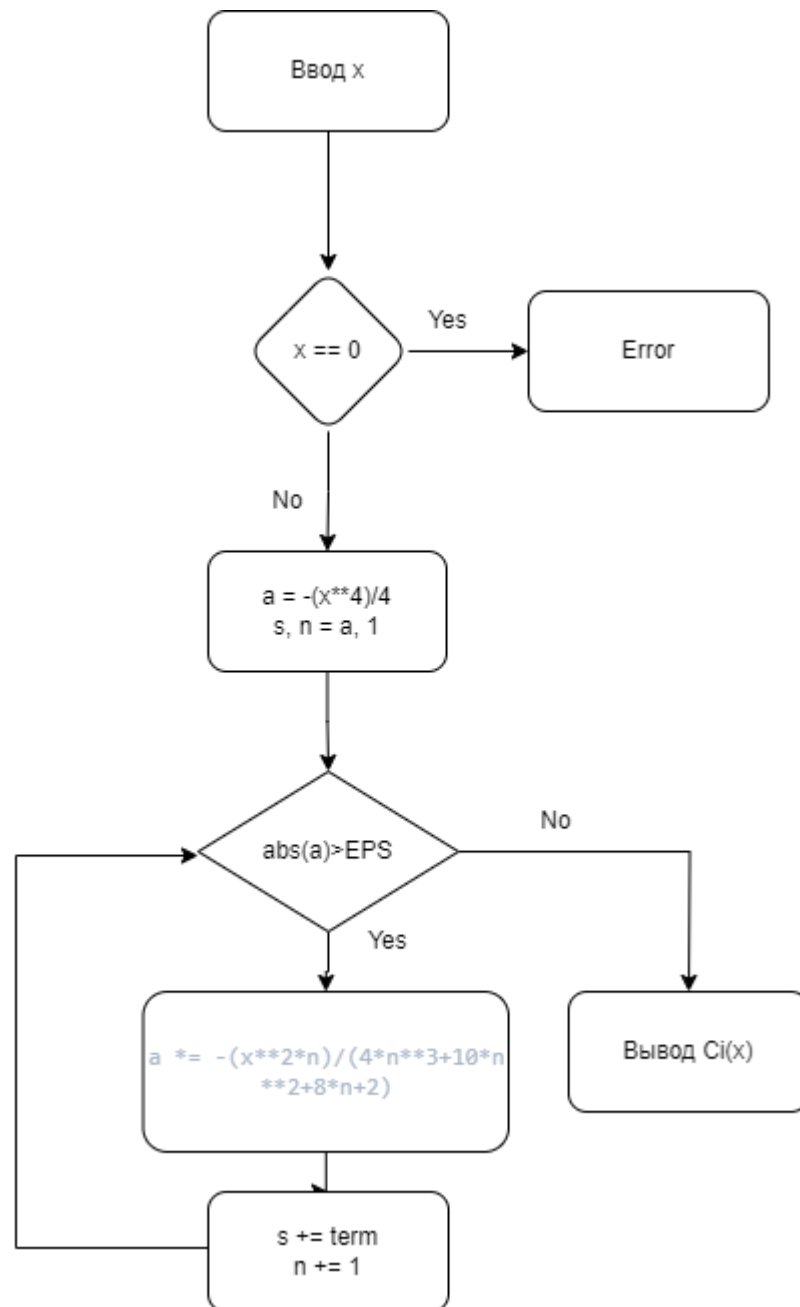


Рисунок 23 – UML-диаграмма для задания повышенной сложности

```

1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4      # вариант - 2
5
6      import ...
7
8
9      # Постоянная Эйлера.
10     EULER = 0.5772156649015328606
11     # Точность вычислений.
12     EPS = 1e-10
13
14     if __name__ == '__main__':
15         x = float(input("Value of x? "))
16         if x == 0:
17             print("Illegal value of x", file=sys.stderr)
18             exit(1)
19
20         a = -(x ** 4) / 4
21         S, n = a, 1
22
23         while math.fabs(a) > EPS:
24             a *= -(x ** 2 * n) / (4 * n ** 3 + 10 * n ** 2 + 8 * n + 2)
25             S += a
26             n += 1
27
28         print(f"Ci({x}) = {EULER + math.log(math.fabs(x)) + S}")
29
30

```

Рисунок 24 – Решение задания повышенной сложности

Ответы на контрольные вопросы:

1. Диаграммы деятельности (UML) используются для моделирования последовательностей действий.
2. Состояние действия обозначает текущий этап выполнения определенного действия. Состояние деятельности относится к более общему состоянию, в котором находится процесс или система в целом.
3. В UML диаграммах для обозначения переходов и ветвлений существуют обозначения: линия или линия со стрелкой – обозначает переход; ромб обозначает ветвление.
4. Алгоритм разветвляющейся структуры — это алгоритм, в котором вычислительный процесс осуществляется либо по одной, либо по другой ветви, в зависимости от выполнения некоторого условия.
5. Разветвляющийся алгоритм отличается от линейного тем, что он позволяет выбрать один из нескольких путей выполнения действий в

зависимости от условия или значения переменной, в то время как линейный алгоритм выполняет действия последовательно по порядку.

6. Условный оператор - конструкция, которая позволяет выполнять различные последовательности действий в зависимости от выполнения или не выполнения условия.

7. В Python используются следующие операторы сравнения: `==`; `!=`; `>`; `<`, `>=`, `<=`.

8. Простое условие, это условие, в результате которого будет получено `true` или `false`. Примеры: `(y == "Привет", x < 199)`.

9. Составное условие, это условие, в котором используются логические операторы для комбинации простых условий

10. `And`, `or`, `not`.

11. Да, может. Называется вложенным ветвлением.

12. Алгоритм, при котором какой-либо блок кода выполняется n-ое количество раз, пока истинно условие.

13. Существуют два типа циклов в Python (`for`, `while`).

14. Функция `range(start, stop, step)` в Python создает последовательность и имеет следующие параметры: `start` – начало последовательности; `stop` – до какого числа будет создана последовательность; `step` – шаг последовательности. Можно использовать в циклах `for` или создавать `list` с нужной последовательностью.

15. `range(15, -1, -2)`

16. Да, циклы могут быть вложенными.

17. Чтобы получился бесконечный цикл, нужно, чтобы его условие всегда было истинно. Для выхода из цикла используется оператор `break`.

18. Оператор `break` необходим для досрочного выхода из цикла.

19. Оператор `continue` используется для досрочного прекращения текущей итерации цикла и перехода к следующей.

20. `Stdout` – стандартный поток вывода, `stderr` – стандартный поток вывода ошибок и предупреждений.

21. Для вывода в стандартный поток `stderr` необходимо в именованный аргумент `file` функции `print(file=sys.stderr)` указать `sys.stderr`.

22. `Exit` используется для досрочного выхода из программы.