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:
Рядская Мария Александровна
1 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и сопровождение программного обеспечения», очная форма обучения

_____
(подпись)

Доцент кафедры
инфокоммуникацийВоронкин 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 и continue , позволяющих реализовывать разветвляющиеся алгоритмы и алгоритмы циклической структуры.

Порядок выполнения работы

1. Создала репозиторий GitHub.

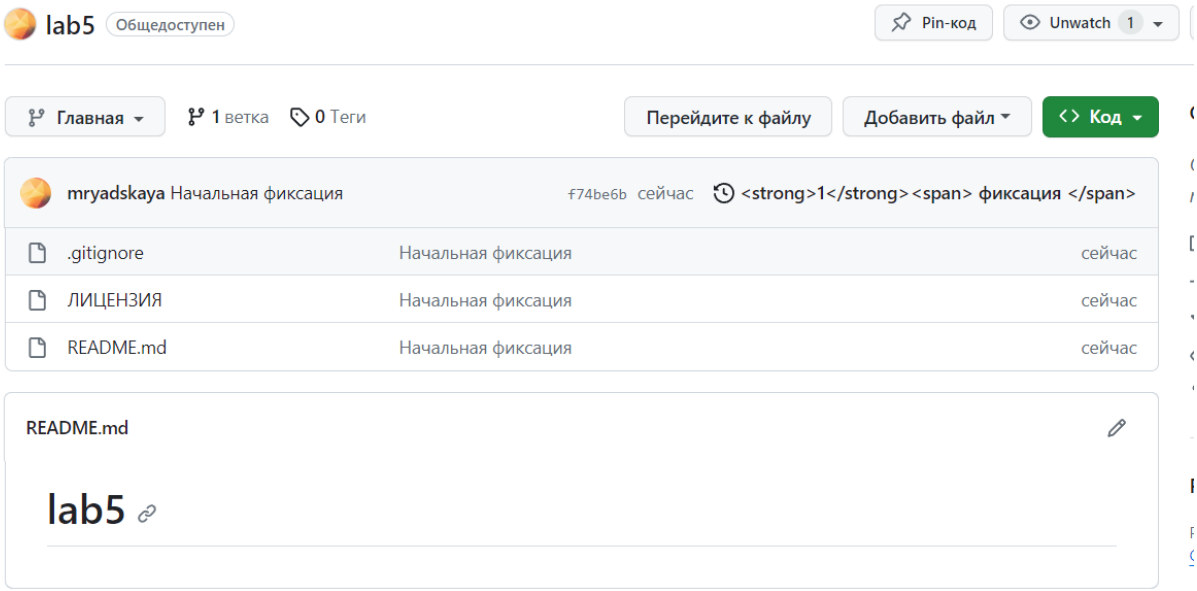

Рисунок 1 – Создание репозитория GitHub

```
C:\>cd git1

C:\git1>git clone https://github.com/mryadskaya/lab5.git
Cloning into 'lab5'...
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:\git1>cd lab5

C:\git1\lab5>
```

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

```
C:\git1\lab5>git branch develop

C:\git1\lab5>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/mryadskaya/lab5/pull/new/develop
remote:
To https://github.com/mryadskaya/lab5.git
 * [new branch]      develop -> develop
branch 'develop' set up to track 'origin/develop'.

C:\git1\lab5>git checkout develop
Switched to branch 'develop'
Your branch is up to date with 'origin/develop'.
```

Рисунок 4 – создание ветки develop

3. Проработала примеры из лабораторной работы.

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

import math

if __name__ == "__main__":
    x = float(input("Value of x? "))
    if x <= 0:
        y = 2 * x * x + math.cos(x)
    elif x < 5:
        y = x + 1
    else:
        y = math.sin(x) - x * x

    print(f"y = {y}")
```

Рисунок 2.1 – Код из примера 1

```
Value of x? -2
y = 7.583853163452858
PS C:\Users\vladi\OneDriv
Value of x? 3
y = 4.0
PS C:\Users\vladi\OneDriv
Value of x? 10
y = -100.54402111088937
```

Рисунок 2.2 – Вывод программы из примера 1

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

import sys

if __name__ == "__main__":
    n = int(input("Введите номер месяца: "))
    if n == 1 or n == 2 or n == 12:
        print("Зима")
    elif n == 3 or n == 4 or n == 5:
        print("Весна")
    elif n == 6 or n == 7 or n == 8:
        print("Лето")
    elif n == 9 or n == 10 or n == 11:
        print("Осень")
    else:
        print("Ошибка!", file=sys.stderr)
        exit(1)
```

Рисунок 2.3 – Код из примера 2

```
Введите номер месяца: 3
Весна
PS C:\Users\vladi\OneDrive\Pa
Введите номер месяца: 13
Ошибка!
```

Рисунок 2.4 – Вывод программы из примера 2

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

import math

if __name__ == "__main__":
    n = int(input("Value of n? "))
    x = float(input("Value of x? "))

    S = 0.0
    for k in range(1, n + 1):
        a = math.log(k * x) / (k * k)
        S += a

    print(f"S = {S}")
```

Рисунок 2.5 – Код и примера 3

```
Value of n? 4
Value of x? 3
S = 1.9459948857353426
```

Рисунок 2.6 – Вывод программы из примера 3

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

import math
import sys

if __name__ == "__main__":
    a = float(input("Value of a? "))
    if a < 0:
        print("Illegal value of a", file=sys.stderr)
        exit(1)

    x, eps = 1, 1e-10

    while True:
        xp = x
        x = (x + a / x) / 2
        if math.fabs(x - xp) < eps:
            break

    print(f"x = {x}\nX = {math.sqrt(a)}")
```

Рисунок 2.7 – Код из примера 4

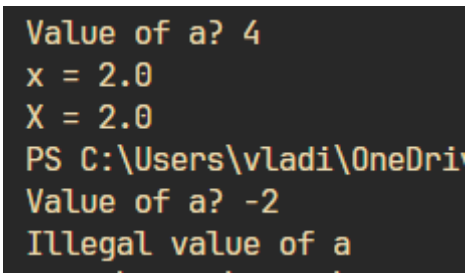

Рисунок 2.8 – Вывод программы из примера 4

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

import math
import sys

EULER = 0.5772156649015328606
EPS = 1e-10

if __name__ == "__main__":
    x = float(input("Value of x? "))
    if x == 0:
        print("Illegal value of x", file=sys.stderr)
        exit(1)

    a = x
    S, k = a, 1

    while math.fabs(a) > EPS:
        a *= x * k / (k + 1) ** 2
        S += a
        k += 1

    print(f"Ei({x}) = {EULER + math.log(math.fabs(x)) + S}")
```

Рисунок 2.9 – Код из примера 5

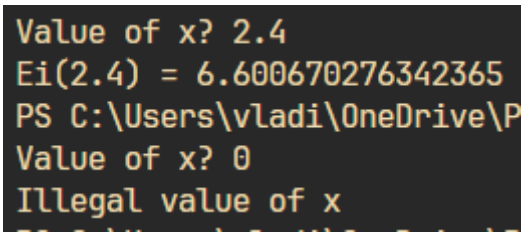

Рисунок 2.10 – Вывод программы из примера 5

4. UML-диаграммы для программ из примеров 4 и 5.

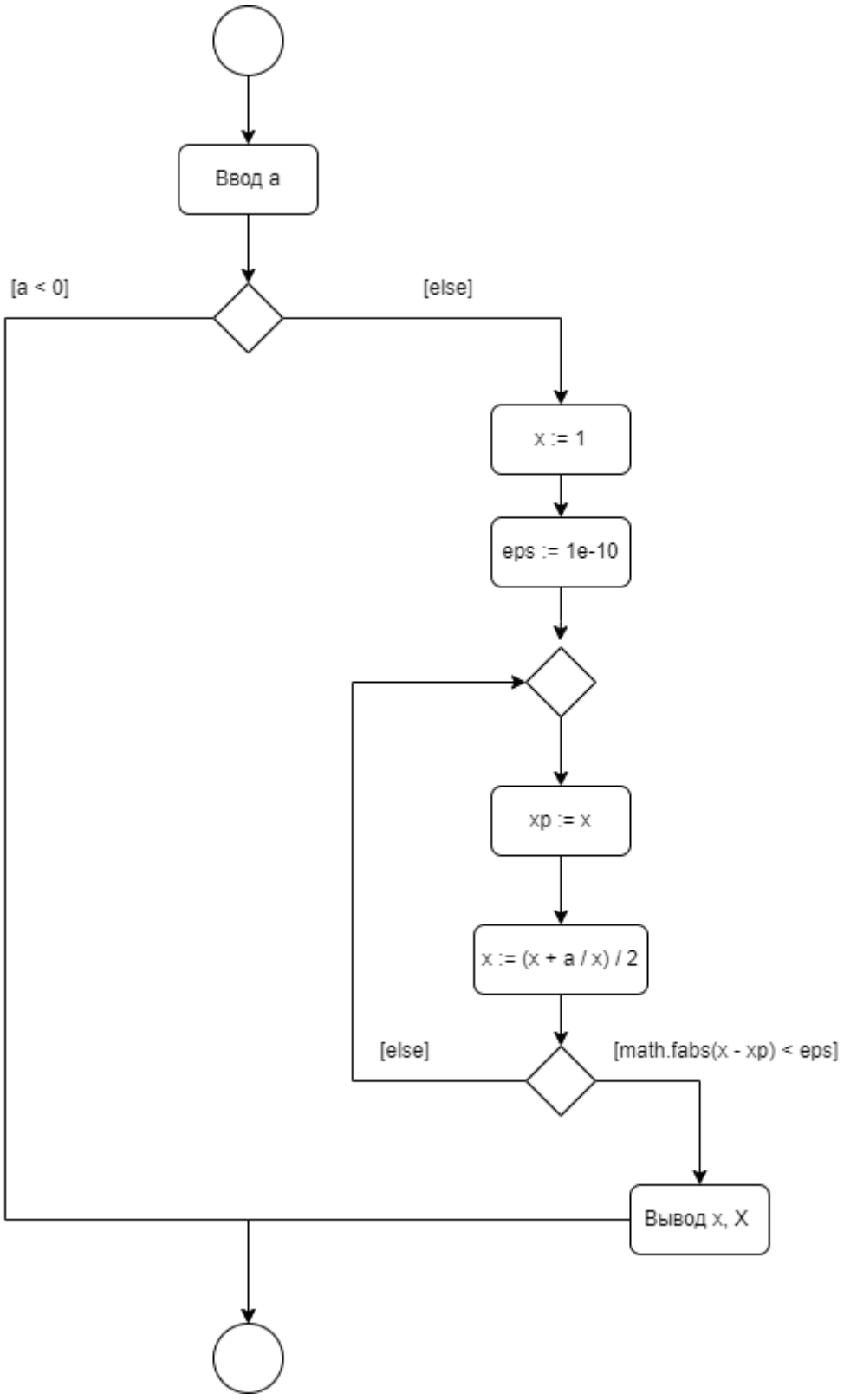

Рисунок 3.1 – UML-диаграмма для примера 4

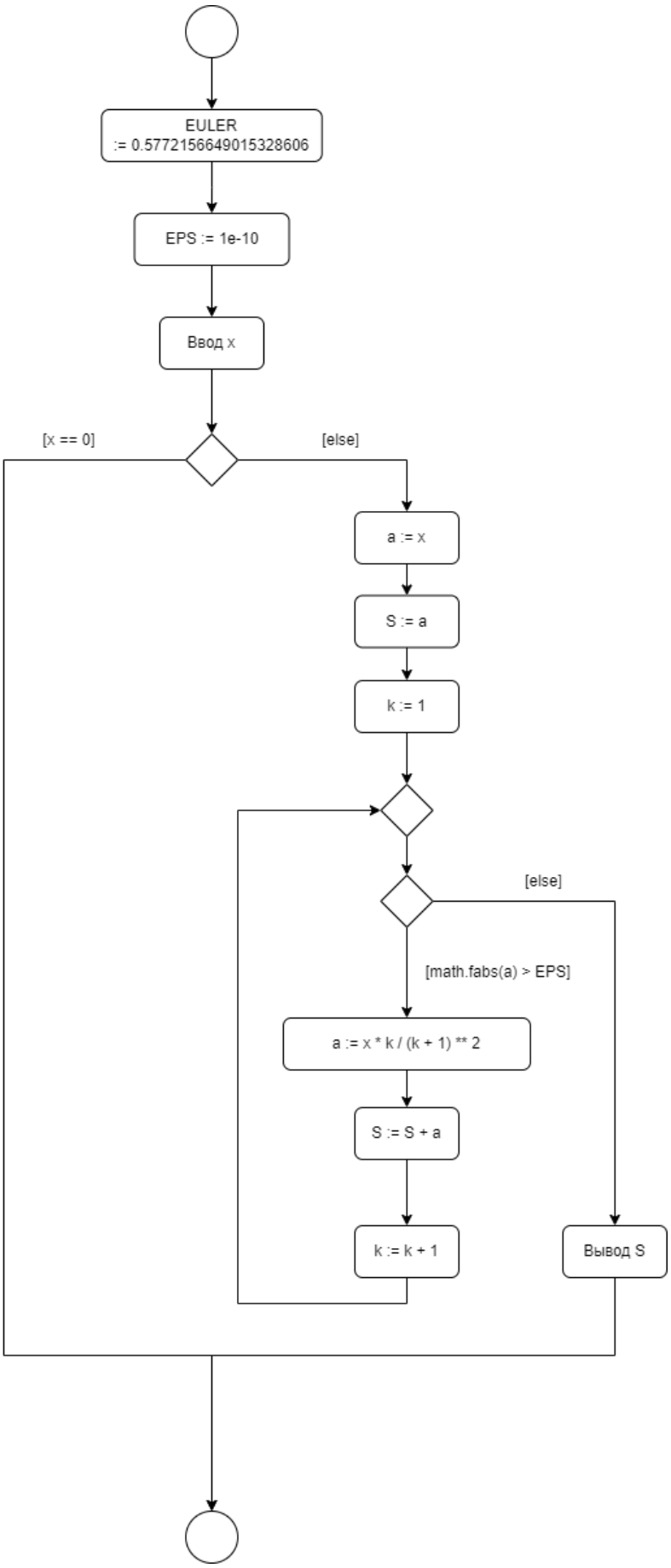

Рисунок 3.2 – UML-диаграмма для примера 5

5. Дано число m

6. (1<=m<=12).Определить, сколько дней в месяце с номером.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
if __name__ == '__main__':
    m = int(input("Введите номер месяца:"))
    if m == 2:
        print("В феврале 28 или 29 дней")
    elif m == 4 or m == 6 or m == 9 or m == 11:
        print("В этом месяце 30 дней")
    else:
        print("В этом месяце 31 день")
```

Рисунок 4.1 –Код программы

```
Введите номер месяца:7
В этом месяце 31 день
```

Рисунок 4.2 – Вывод программы

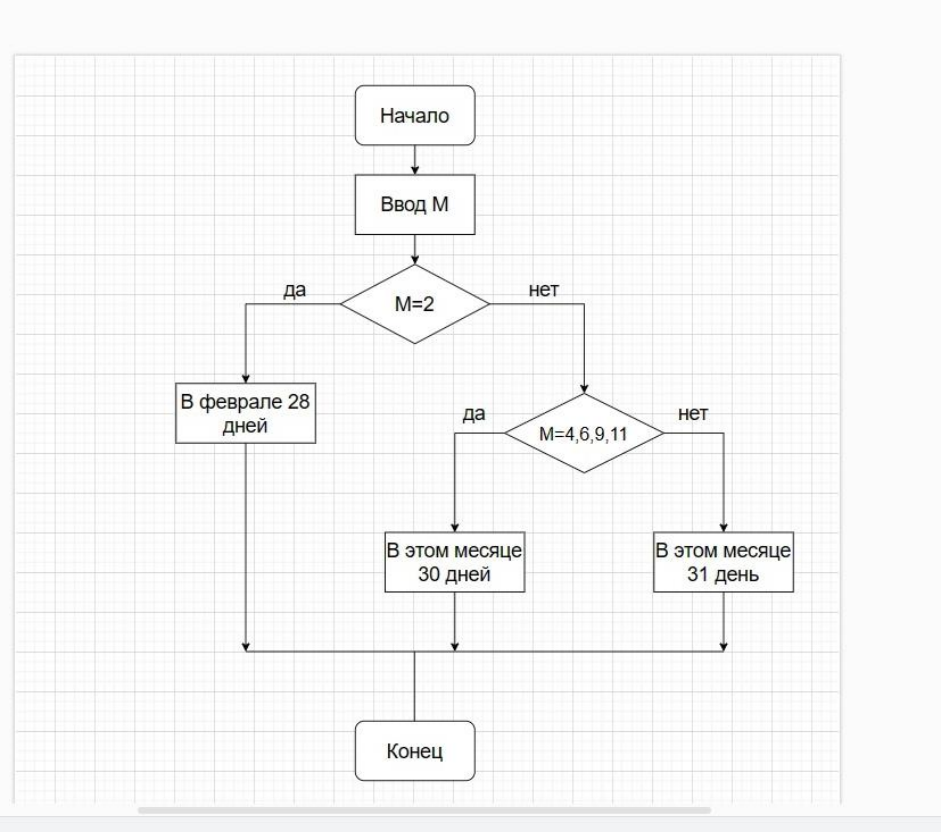

Рисунок 4.3 – UML-диаграмма

7. .Составить программу решения квадратного уравнения. Выводить также комплексныерешения.

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

if __name__ == '__main__':
    a = float(input("Введите коэффициент a: "))
    b = float(input("Введите коэффициент b: "))
    c = float(input("Введите коэффициент c: "))
    # Вычисляем дискриминант
    D = b ** 2 - 4 * a * c
    if D > 0:
        # Корни квадратного уравнения вещественные
        x1 = (-b + math.sqrt(D)) / (2 * a)
        x2 = (-b - math.sqrt(D)) / (2 * a)
        print("Корни квадратного уравнения:", x1, x2)
    elif D == 0:
        # Корни квадратного уравнения одинаковые и вещественные
        x = -b / (2 * a)
        print("Уравнение имеет один корень:", x)
    else:
        # Корни квадратного уравнения комплексные
        real_part = -b / (2 * a)
        imaginary_part = math.sqrt(-D) / (2 * a)
        x1 = complex(real_part, imaginary_part)
        x2 = complex(real_part, -imaginary_part)
```

Рисунок 5.1 – Код программы

```
C:\Users\ADMIN\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\git1\
Введите коэффициент a: 25
Введите коэффициент b: 66
Введите коэффициент c: 12
Корни квадратного уравнения: -0.19643424758494887 -2.443565752415051

Process finished with exit code 0
```

Рисунок 5.2 – Вывод программы

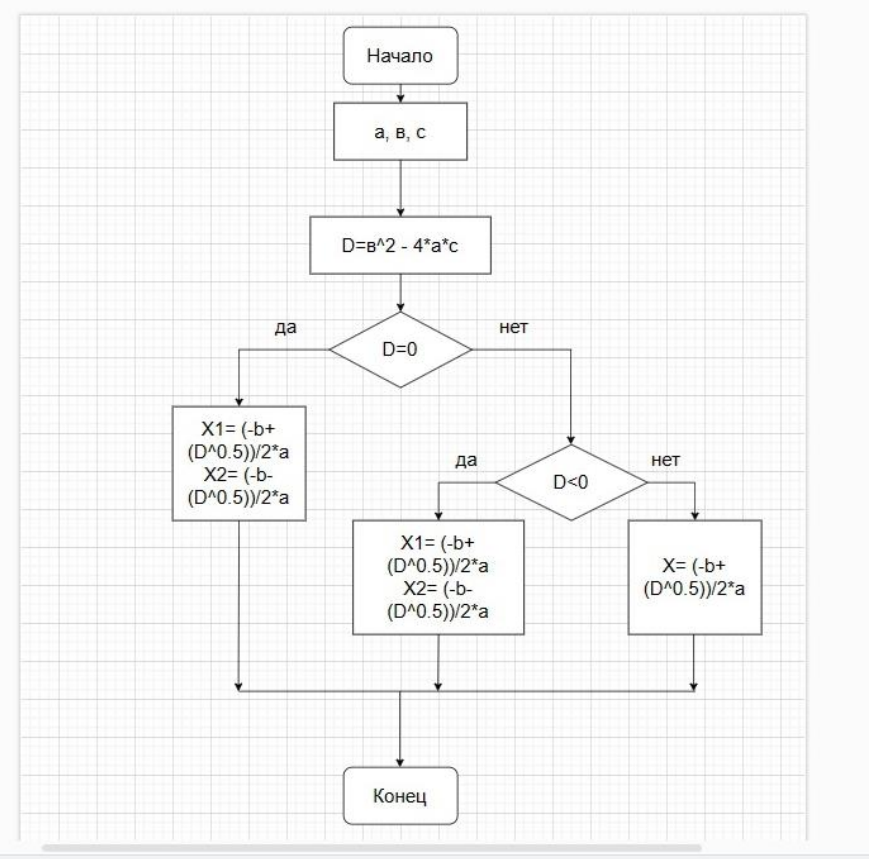

Рисунок 5.3 – UML-диаграмма

8. Вычислить сумму всех n-значных чисел, кратных k (1<=n<=4).

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


if __name__ == '__main__':
    n, k = map(int, input().split())
    a, b = (10 ** n + k - 1) // k, (10 ** (n + 1) - 1) // k
    print((b - a + 1) * k * (a + b) // 2)
```

Рисунок 6.1 – Код программы

```
C:\Users\ADMIN\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\g
2 3
165150

Process finished with exit code 0
```

Рисунок 6.2 – Вывод программы

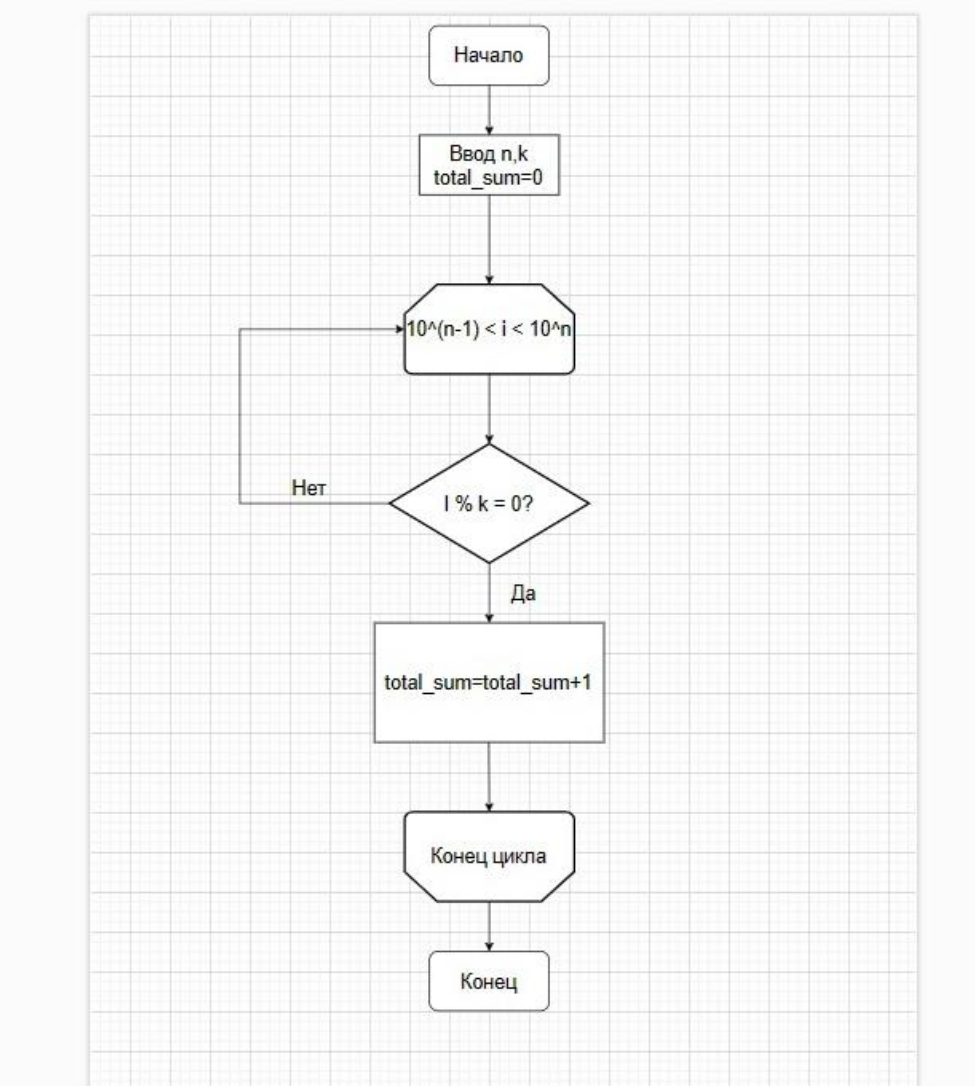

Рисунок 6.3 – UML-диаграмма

9. Задание повышенной сложности вариант 5. Вычислить данную функцию по разложению в ряд.

$$\mathrm{C}(x) = \int_0^x \cos\left(\frac{\pi}{2}t^2\right) dt = \sum_{n=0}^{\infty} \frac{(-1)^n (\pi/2)^{2n}}{(2n)!(4n+1)}.$$

$$a_n = \frac{(-1)^n x^{2n}}{(2n)!\,(4n+1)}$$

$$a_{n+1} = \frac{(-1)^{n+1} x^{2n+1}}{(2(n+1))!\,(4(n+1)+1)}$$

$$\frac{a_{n+1}}{a_n} = \frac{(-1)^{n+1} x^{2n+1}}{\big(2(n+1)\big)!\,(4(n+1)+1)} : \frac{(-1)^n x^{2n}}{(2n)!\,(4n+1)}$$

$$\frac{a_{n+1}}{a_n} = \frac{-x^2(4n+1)}{16n^3+44n^2+38n+10}$$

$$a_{n+1} = \frac{-x^2(4n+1)}{16n^3+44n^2+38n+10} * a_n$$

$$a_O = \frac{(-1)^0 x^0}{0!\,(4\cdot 0+1)} = 1$$

posllednee.py × 5primer.py

```
1
2   #!/usr/bin/env python3
3   # -*- coding: utf-8 -*-
4   import math
5   import sys
6   # Точность вычислений.
7   eps = 1e-10
8   if __name__ == '__main__':
9       x = float(input("Value of x? "))
10      if x == 0:
11          print("Illegal value of x")
12          exit(1)
13      a = 1
14      s, n = a, 0
15      # Найти сумму членов ряда.
16      while math.fabs(a) > eps:
17          a = a * (((-1) * (x ** 2) * (4 * n + 1)) / ((16 * (n ** 3) + 44 * (n ** 2) + 38 * n + 10)))
18          s += a
19          n += 1
20      # Вывести значение функции.
21      print(f"C(x) = {s}")
```

```
Value of x? 1.2
C(x) = 0.9798934003879824
```

Рисунок 7.3 – Вывод программы

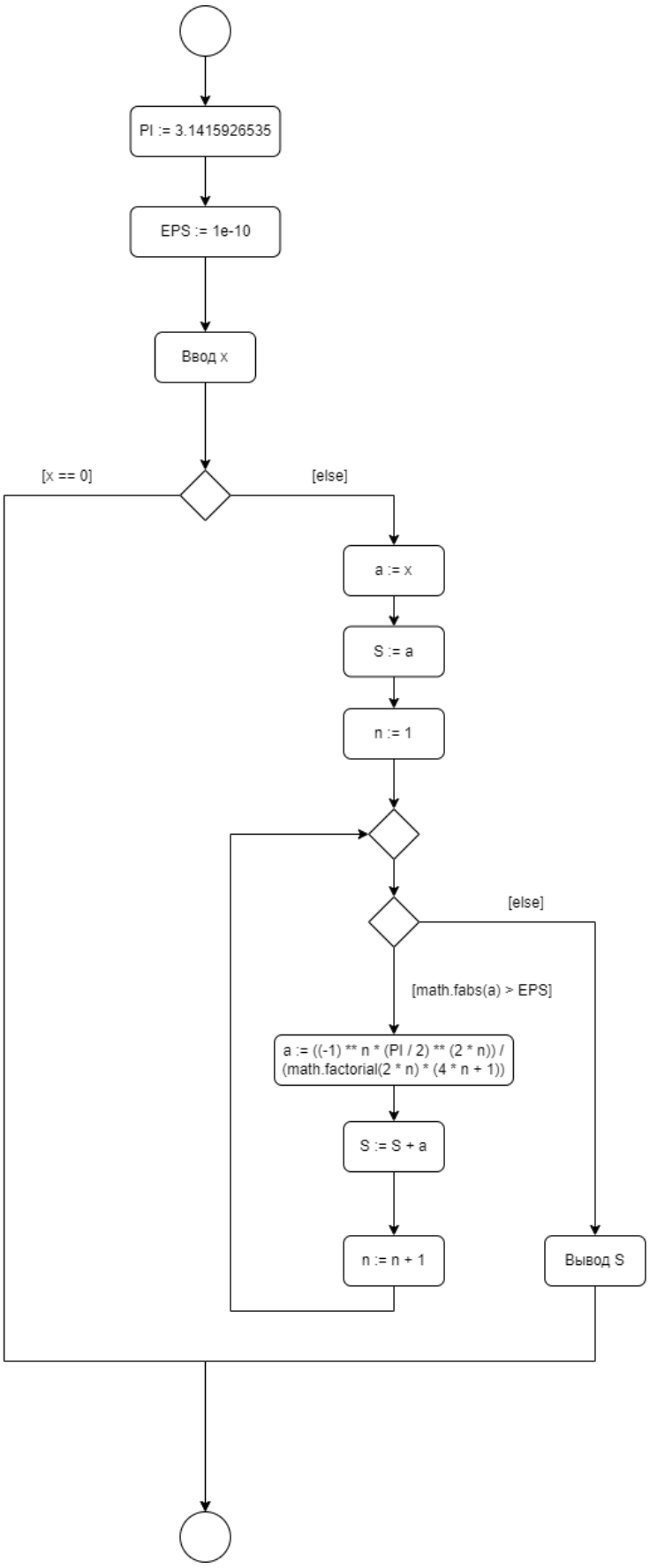

Рисунок 7.4 – UML-диаграмма

9. Зафиксировал все изменения в github в ветке develop.

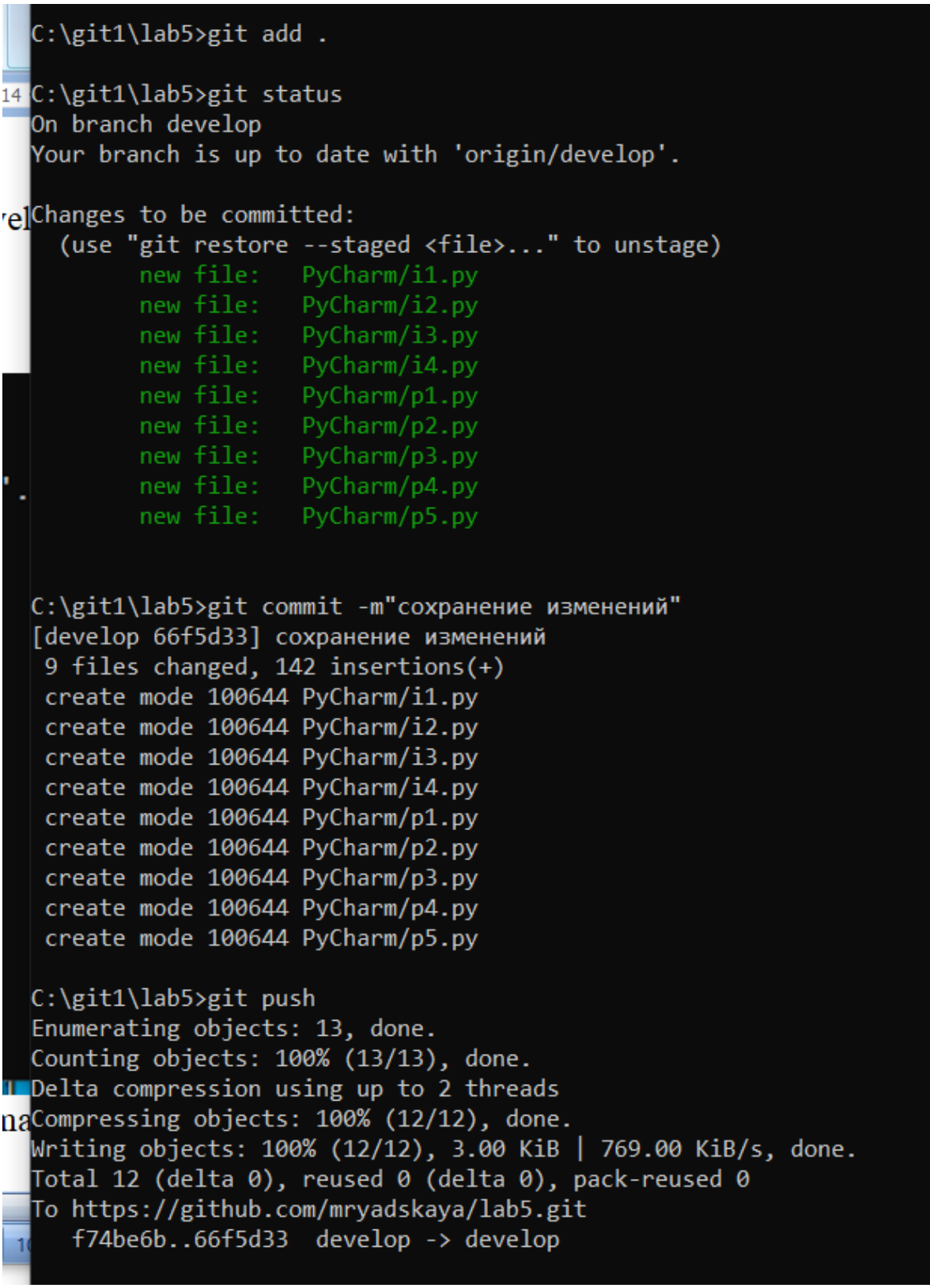

Рисунок 19 – фиксация изменений в ветку develop

10. Слила ветки.

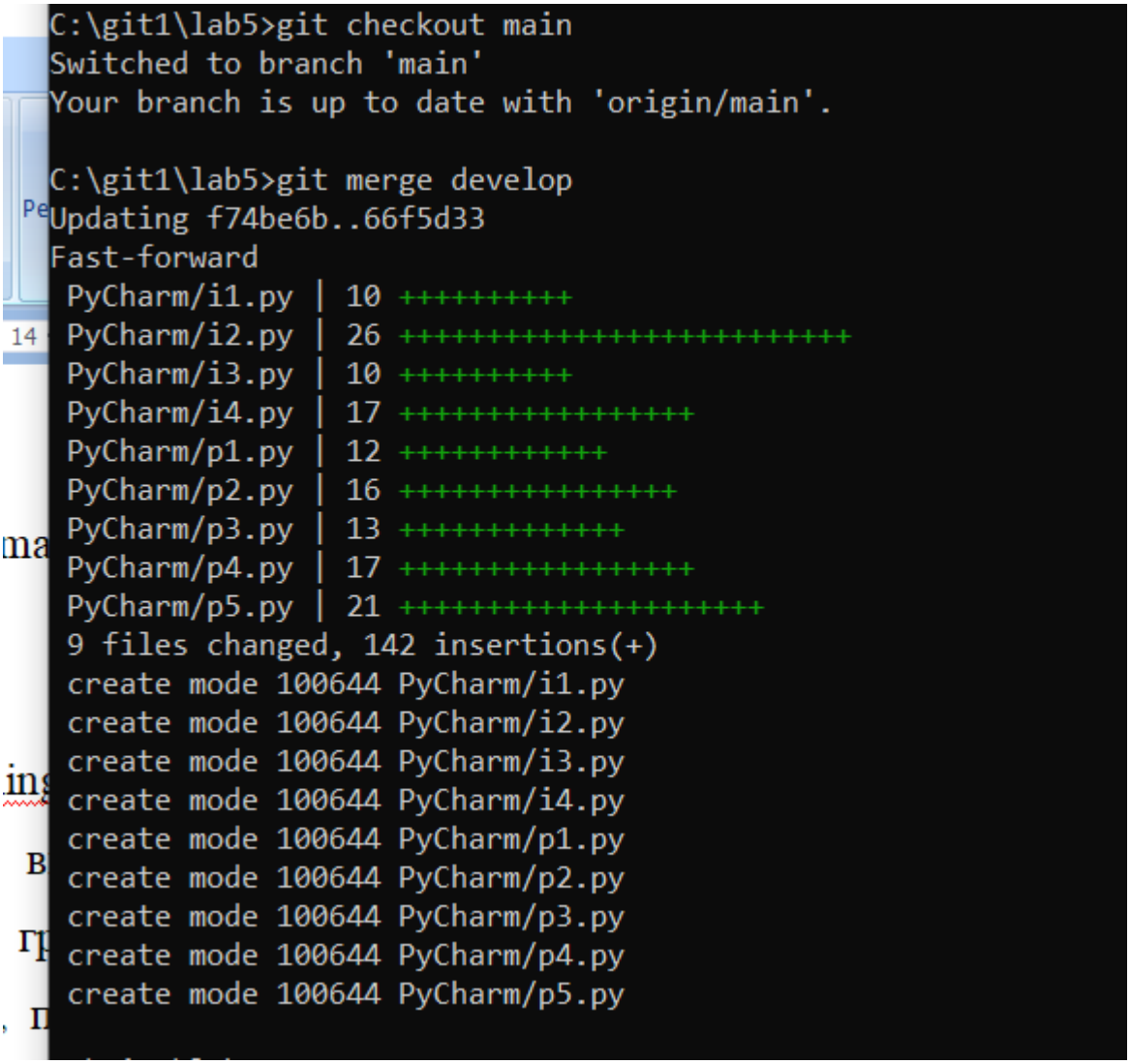

Рисунок 20 – сливание ветки developв ветку main

Ответы на контрольные вопросы

1. Диаграммы деятельности в UML (UnifiedModelingLanguage) используются для моделирования и визуализации процесса выполнения действий или задач в системе. Они представляют собой графическое представление последовательности действий, включая условия, переходы и ветвления, и помогают разработчикам и заинтересованным сторонам лучше понять и описать работу системы.

2. Состояние действия (actionstate) в диаграммах деятельности описывает мгновенное выполнение операции или действия. Это простое действие, которое выполняется без переходов в другие состояния.

Состояние деятельности (activitystate) представляет собой группу действий, которые выполняются вместе. В диаграммах деятельности оно может быть использовано для описания более сложных действий, которые состоят из нескольких этапов.

3. Для обозначения переходов и ветвлений в диаграммах деятельности существуют различные нотации:

- Переходы (Transitions) обычно обозначаются стрелками и могут иметь условное выражение или событие, при котором происходит переход.
- Ветвления (Decisions) обычно обозначаются ромбовидными формами и представляют точки принятия решений. Они могут иметь один или несколько исходящих переходов, каждый из которых имеет условное выражение.
- Слияния (Merge) обычно обозначаются стрелками, сходящимися к одной точке, и представляют объединение потоков выполнения в один.

4. Алгоритм разветвляющейся структуры (conditionalstructure) представляет собой алгоритм, в котором происходит разделение потока выполнения на два или более возможных направления в зависимости от условия. Примером такого алгоритма может быть условный оператор if-else в языке программирования.

5. Разветвляющийся алгоритм отличается от линейного алгоритма тем, что он позволяет выбирать одну из нескольких возможных ветвей выполнения в зависимости от условия. Такой алгоритм может иметь

различные пути выполнения, в то время как линейный алгоритм имеет только один последовательный путь выполнения без разветвлений.

6. Условный оператор (conditionalstatement) — это конструкция в программировании, которая позволяет выполнить определенные действия в зависимости от условия, которое может быть истинным или ложным. В Python условный оператор обычно представлен конструкцией if-elif-else. Есть также форма условного оператора без else, называемая простым условным оператором.

7. В Python используются следующие операторы сравнения:

- == (равно)
- != (не равно)
- > (больше)
- < (меньше)
- >= (больше или равно)
- <= (меньше или равно)

8. Простое условие (simplecondition) в условном операторе представляет собой одно условие, которое может быть истинным или ложным.

9. Составное условие (compoundcondition) в условном операторе представляет собой комбинацию нескольких простых условий с помощью логических операторов.

10. При составлении сложных условий в условном операторе в Python допускаются следующие логические операторы:

- and (логическое И)
- or (логическое ИЛИ)

- not (логическое НЕ)

11. Да, оператор ветвления в Python может содержать внутри себя другие ветвления. Это позволяет создавать более сложные иерархии условий и выполнять различные действия в зависимости от набора условий.

12. Алгоритм циклической структуры (loopstructure) представляет собой алгоритм, в котором определенный блок кода выполняется несколько раз в зависимости от условия. Примером такого алгоритма может быть цикл while или for в языке программирования.

13. В языке Python существуют следующие типы циклов:

- Цикл for используется для итерации по последовательности или коллекции элементов.
- Цикл while выполняется, пока условие истинно.

14. Функция range в Python используется для создания последовательности чисел. Она возвращает итерируемый объект, который может быть использован в циклах for. Функция range может принимать от одного до трех аргументов: range(stop), range(start, stop), range(start, stop, step).

15. Для организации перебора значений от 15 до 0 с шагом 2 с помощью функции range можно использовать следующий код:

```
for i in range(15, -1, -2):
  print(i)
```

16. Да, в языке программирования Python циклы могут быть вложенными, то есть один цикл может находиться внутри другого цикла. Это позволяет создавать более сложные алгоритмы и выполнять итерации в нескольких уровнях.

17. Бесконечный цикл (infiniteloop) образуется, когда условие цикла всегда истинно, или когда цикл не имеет условия выхода. Для выхода из бесконечного цикла можно использовать операторы break или return, которые прерывают выполнение цикла и переносят управление за его пределы.

18. Оператор break используется в циклах для немедленного прекращения выполнения цикла и выхода из него. Когда break достигается, программа продолжает выполнение со следующего после цикла оператора. Оператор break особенно полезен для прерывания циклов в зависимости от определенных условий.

19. Оператор continue используется в циклах для пропуска текущей итерации и перехода к следующей итерации цикла. Когда continue достигается, оставшаяся часть текущего цикла не выполняется, и управление передается обратно к началу цикла для следующей итерации. Оператор continue позволяет пропустить определенные шаги цикла в зависимости от условий.

20. Стандартные потоки stdout (стандартный поток вывода) и stderr (стандартный поток ошибок) используются для обработки вывода программы.

stdout используется для вывода обычной информации, результатов работы программы и других сообщений.

stderr используется для вывода сообщений об ошибках и предупреждений.

Обычно stdout отображается в консоли или сохраняется в файл, в то время как stderr также может быть перенаправлен в отдельный файл или игнорироваться.

21. Чтобы организовать вывод в стандартный поток stderr в Python, можно использовать модуль sys и его атрибут stder:

22. Функция exit в Python используется для немедленного выхода из программы. Когда вызывается эта функция, выполнение программы прекращается, и управление возвращается операционной системе или среде выполнения. Функция exit принимает необязательный аргумент, который может использоваться для указания кода завершения программы.