

**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  
по дисциплине «Программирование на Python»**

Выполнил студент Ромащенко Данил группы ИВТ-б-о-24-1

Подпись студента \_\_\_\_\_  
Работа защищена « » \_\_\_\_\_ 20\_\_ г.  
Проверил Воронкин Р.А. \_\_\_\_\_  
(подпись)

Ставрополь 2025

**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.

Ход работы

Вариант 18

**Ссылка на репозиторий:** <https://github.com/11N322/lab-3.git>

1) Был создан общедоступный репозиторий, в котором использована лицензия MIT и язык программирования Python.

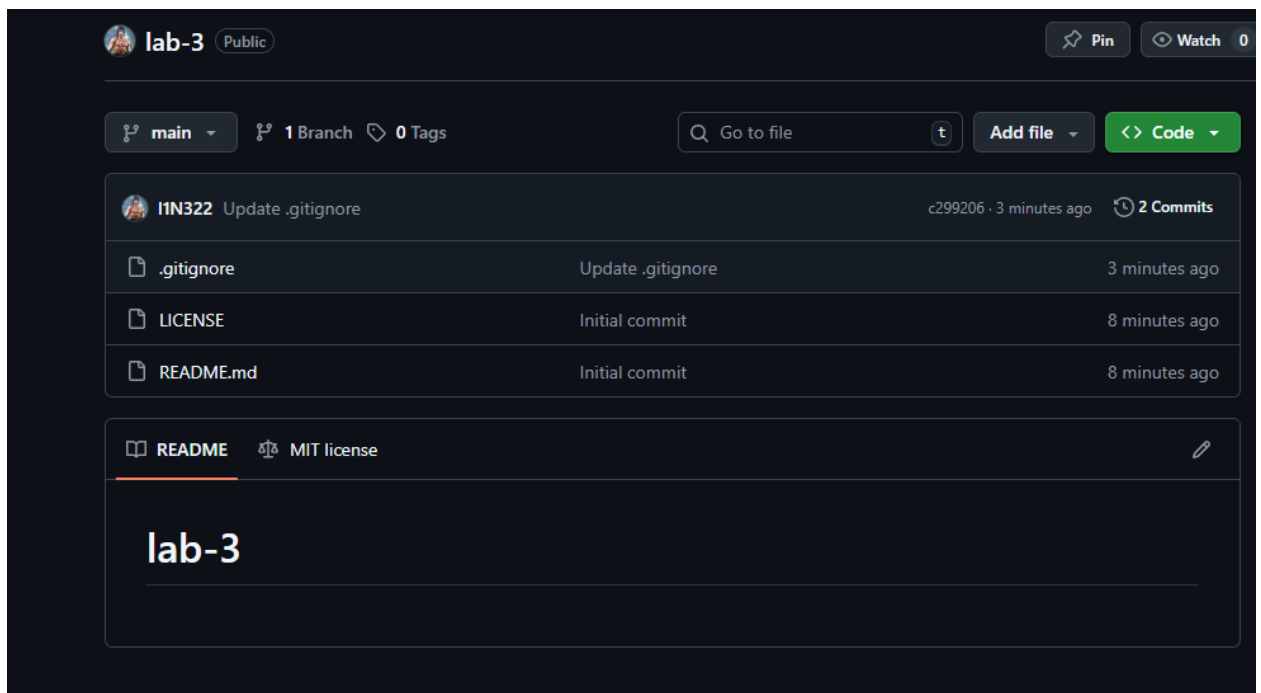
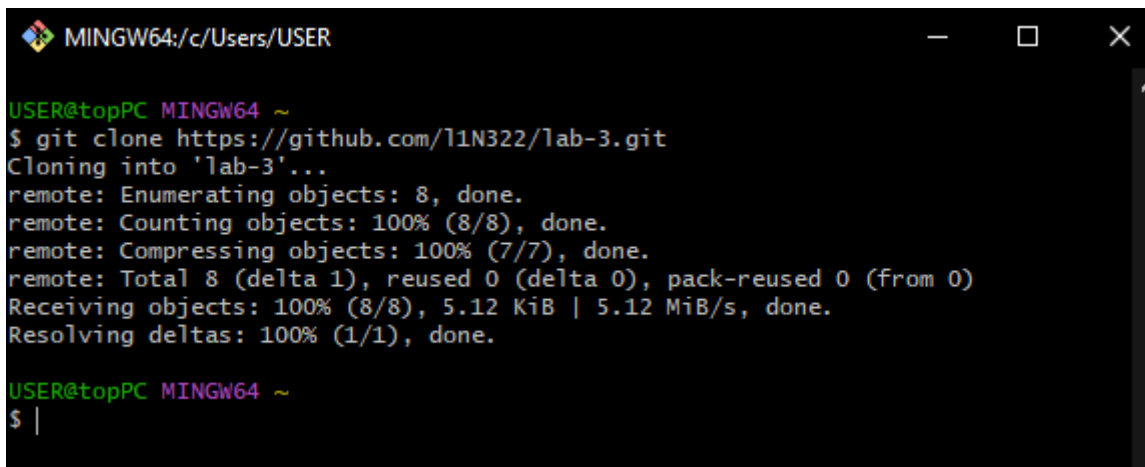


Рисунок 1. Созданный репозиторий

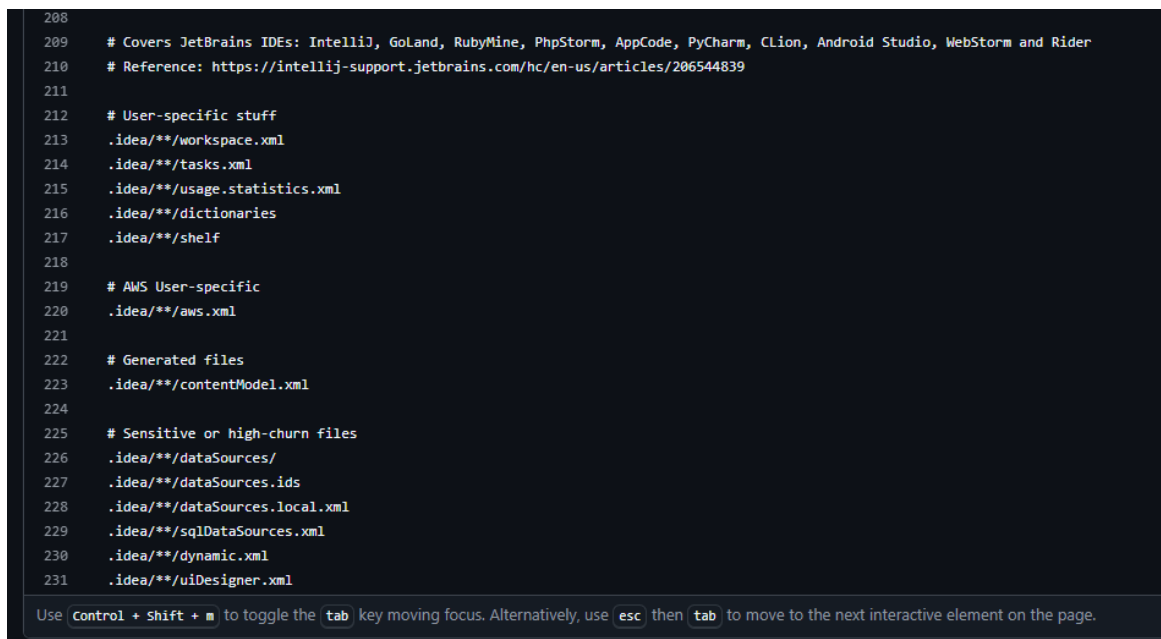
2) Было выполнено клонирование созданного репозитория на компьютер.



```
MINGW64:/c/Users/USER
USER@topPC MINGW64 ~
$ git clone https://github.com/11N322/lab-3.git
Cloning into 'lab-3'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (8/8), 5.12 KiB | 5.12 MiB/s, done.
Resolving deltas: 100% (1/1), done.
USER@topPC MINGW64 ~
$ |
```

Рисунок 2. Клонирование репозитория

3) Был дополнен файл .gitignore необходимыми правилами для работы с IDE PyCharm.



```
208
209 # Covers JetBrains IDEs: IntelliJ, GoLand, RubyMine, PhpStorm, AppCode, PyCharm, CLion, Android Studio, WebStorm and Rider
210 # Reference: https://intellij-support.jetbrains.com/hc/en-us/articles/206544839
211
212 # User-specific stuff
213 .idea/**/workspace.xml
214 .idea/**/tasks.xml
215 .idea/**/usage.statistics.xml
216 .idea/**/dictionaries
217 .idea/**/shelf
218
219 # AWS User-specific
220 .idea/**/aws.xml
221
222 # Generated files
223 .idea/**/contentModel.xml
224
225 # Sensitive or high-churn files
226 .idea/**/dataSources/
227 .idea/**/dataSources.ids
228 .idea/**/dataSources.local.xml
229 .idea/**/sqlDataSources.xml
230 .idea/**/dynamic.xml
231 .idea/**/uiDesigner.xml
```

Рисунок 3. Добавленные правила в файл .gitignore

4) Был создан проект PyCharm в папке репозитория.

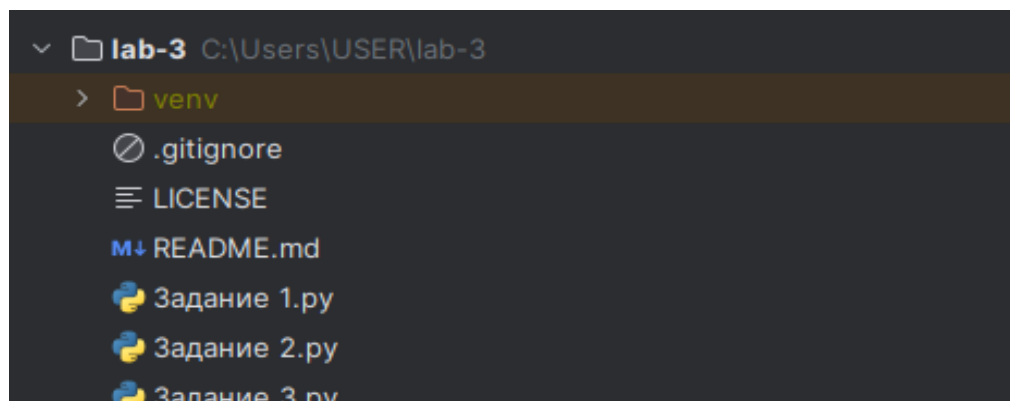


Рисунок 4. Созданный проект

5) Были проработаны примеры лабораторной работы и зафиксированы результаты выполнения каждой из программ при разных исходных данных, вводимых с клавиатуры.

```
1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 import math
5
6
7 ▶ if __name__ == "__main__":
8     x = float(input("Value of x? "))
9
10    if x <= 0:
11        y = 2 * x * x + math.cos(x)
12    elif x < 5:
13        y = x + 1
14    else:
15        y = math.sin(x) - x * x
16
17    print(f"y = {y}")
```

Рисунок 5. Пример 1

```
Пример 1 ×
D:\git\lab-3\venv\Scripts\python.exe "D:/git/lab-3/Пример 1.py"
Value of x? 0
y = 1.0

Process finished with exit code 0
```

Рисунок 6. Результат выполнения примера 1

```
Пример 1 ×
D:\git\lab-3\venv\Scripts\python.exe "D:/git/lab-3/Пример 1.py"
Value of x? 4
y = 5.0

Process finished with exit code 0
```

Рисунок 7. Результат выполнения примера 1 с другими данными

```

1 ▶ #!/usr/bin/env python3
2   # -*- coding: utf-8 -*-
3
4   import sys
5
6
7 ▶ if __name__ == "__main__":
8     n = int(input("Введите номер месяца: "))
9
10    if n == 1 or n == 2 or n == 12:
11        print("Зима")
12    elif n == 3 or n == 4 or n == 5:
13        print("Весна")
14    elif n == 6 or n == 7 or n == 8:
15        print("Лето")
16    elif n == 9 or n == 10 or n == 11:
17        print("Осень")
18    else:
19        print("Ошибка!", file=sys.stderr)
20    exit(1)

```

Рисунок 8. Пример 2

```

Пример 2 x
D:\git\lab-3\venv\Scripts\python.exe "D:/git/lab-3/Пример 2.py"
Введите номер месяца: 1
Зима

Process finished with exit code 0

```

Рисунок 9. Результат выполнения примера 2

```

Пример 2 x
D:\git\lab-3\venv\Scripts\python.exe "D:/git/lab-3/Пример 2.py"
Введите номер месяца: 8
Лето

Process finished with exit code 0

```

Рисунок 10. Результат выполнения примера 2 с другими данными

```

1 ▶ #!/usr/bin/env python3
2   # -*- coding: utf-8 -*-
3
4   import math
5
6
7 ▶   if __name__ == "__main__":
8       n = int(input("Value of n? "))
9       x = float(input("Value of x? "))
10
11       S = 0.0
12       for k in range(1, n + 1):
13           a = math.log(k * x) / (k * k)
14           S += a
15
16       print(f"S = {S}")

```

Рисунок 11. Пример 3

```

Пример 3 ×
D:\git\lab-3\venv\Scripts\python.exe "D:/git/lab-3/Пример 3.py"
Value of n? 2
Value of x? 4
S = 1.9061547465398494

Process finished with exit code 0

```

Рисунок 12. Результат выполнения примера 3

```

Пример 3 ×
D:\git\lab-3\venv\Scripts\python.exe "D:/git/lab-3/Пример 3.py"
Value of n? 6
Value of x? 5
S = 2.89644465760977

Process finished with exit code 0

```

Рисунок 13. Результат выполнения примера 3 с другими данными

```

1 ▶ #!/usr/bin/env python3
2   # -*- coding: utf-8 -*-
3
4   import math
5   import sys
6
7
8 ▶   if __name__ == "__main__":
9       a = float(input("Value of a? "))
10      if a < 0:
11          print("Illegal value of a", file=sys.stderr)
12          exit(1)
13
14      x, eps = 1, 1e-10
15      while True:
16          xp = x
17          x = (x + a / x) / 2
18          if math.fabs(x - xp) < eps:
19              break
20
21      print(f"x = {x}\nX = {math.sqrt(a)}")

```

Рисунок 14. Пример 4

```

Пример 4 ×
D:\git\lab-3\venv\Scripts\python.exe "D:/git/lab-3/Пример 4.py"
Value of a? 4
x = 2.0
X = 2.0

Process finished with exit code 0

```

Рисунок 15. Результат выполнения примера 4

```

Пример 4 ×
D:\git\lab-3\venv\Scripts\python.exe "D:/git/lab-3/Пример 4.py"
Value of a? 22
x = 4.69041575982343
X = 4.69041575982343

Process finished with exit code 0

```

Рисунок 16. Результат выполнения примера 4 с другими данными

```

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

```

Рисунок 17. Пример 5

```

Пример 5 ×
D:\git\lab-3\venv\Scripts\python.exe "D:/git/lab-3/Пример 5.py"
Value of x? 7
Ei(7.0) = 191.50474333549477

Process finished with exit code 0

```

Рисунок 18. Результат выполнения примера 5



```
Пример 5 ×
D:\git\lab-3\venv\Scripts\python.exe "D:/git/lab-3/Пример 5.py"
Value of x? 3
Ei(3.0) = 9.93383257061422

Process finished with exit code 0
```

Рисунок 19. Результат выполнения примера 5 с другими данными

б) Для примеров 4 и 5 были построены UML-диаграммы деятельности.

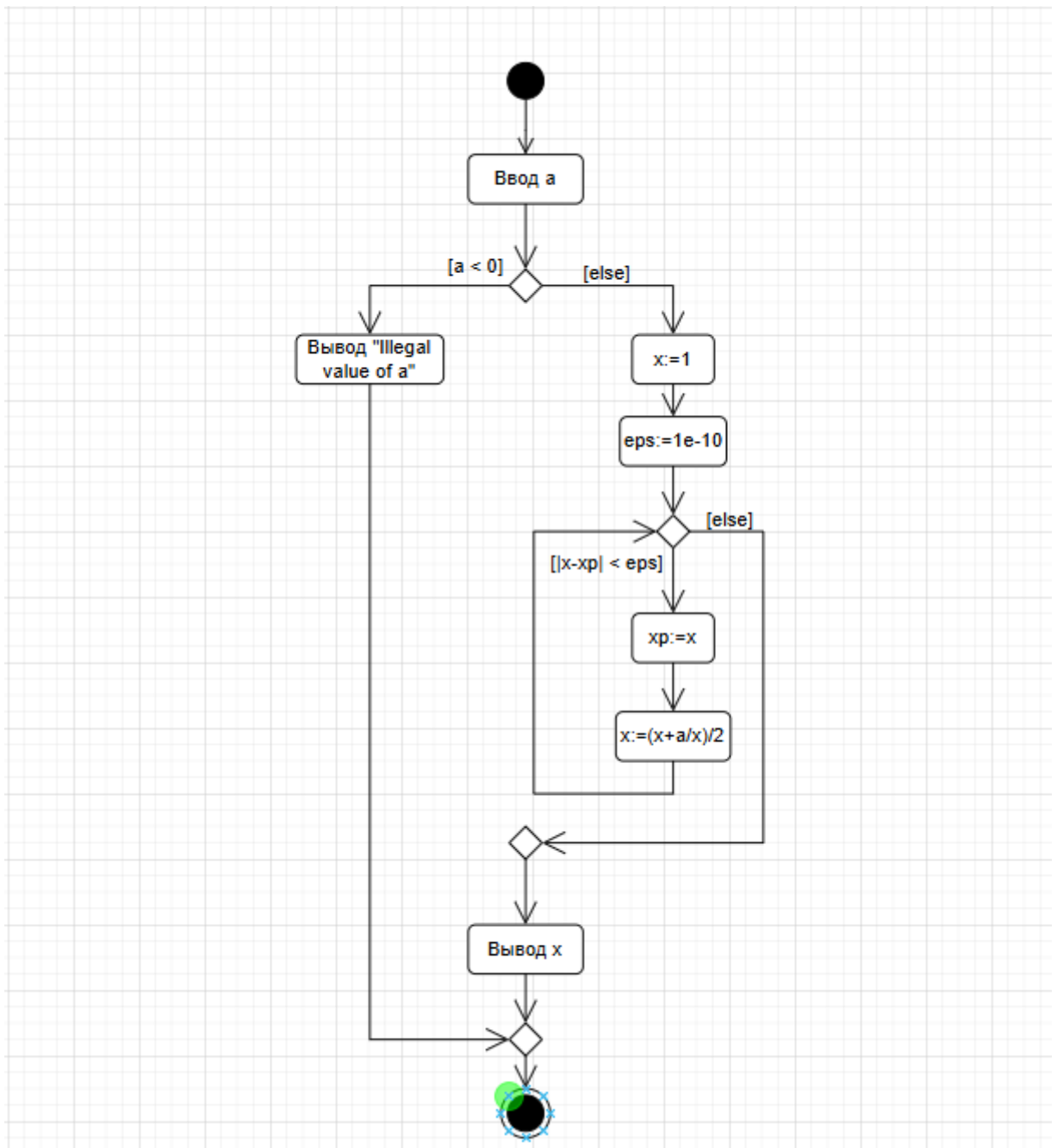


Рисунок 20. UML-диаграмма деятельности для примера 4

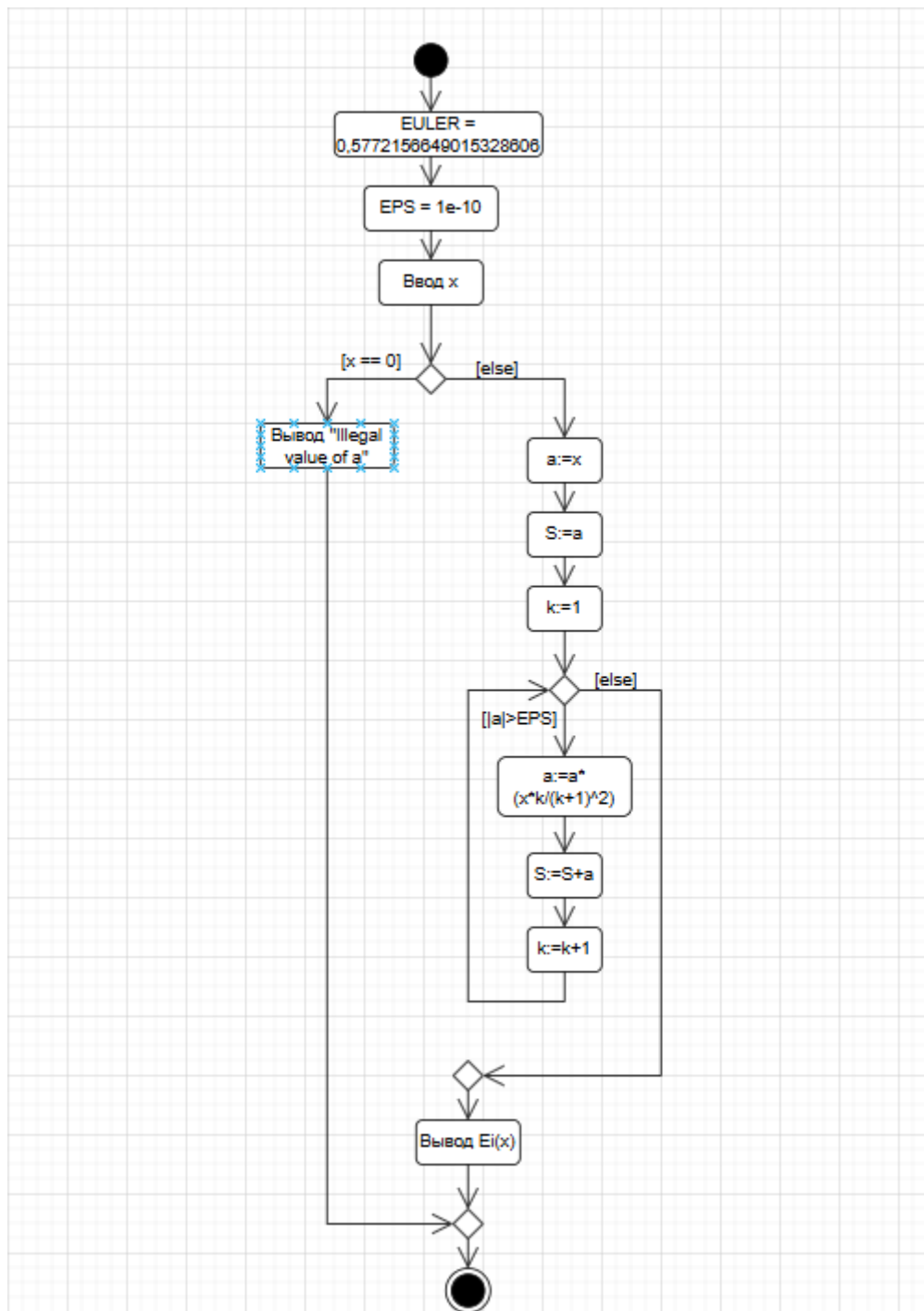


Рисунок 21. UML-диаграмма деятельности для примера 5

7) Были выполнены индивидуальные задания и построены UML-диаграммы деятельности.

8) Задание 1: с клавиатуры вводится цифра  $m$  (от 1 до 4). Вывести на экран названия месяцев, соответствующих времени года с номером  $m$  (считать зиму временем года № 1).

```

1 ▶ #!/usr/bin/env python3
2   # -*- coding: utf-8 -*-
3
4   import sys
5
6
7 ▶   if __name__ == "__main__":
8       m = int(input("Введите цифру от 1 до 4: "))
9
10      if m == 1:
11          print("Декабрь, январь, февраль")
12      elif m == 2:
13          print("Март, апрель, май")
14      elif m == 3:
15          print("Июнь, июль, август")
16      elif m == 4:
17          print("Сентябрь, октябрь, ноябрь")
18      else:
19          print("Ошибка!", file=sys.stderr)
20      exit(1)

```

Рисунок 22. Задание 1

```

Задание 1 x
D:\git\lab-3\venv\Scripts\python.exe "D:/git/lab-3/Задание 1.py"
Введите цифру от 1 до 4: 1
Декабрь, январь, февраль

Process finished with exit code 0

```

Рисунок 23. Результат выполнения задания 1

```

D:\git\lab-3\venv\Scripts\python.exe "C:/Users/USER/lab-3/Задание 1.py"
Введите цифру от 1 до 4: 3
Июнь, июль, август

Process finished with exit code 0

```

Рисунок 24. Результат выполнения задания 1 с другими данными

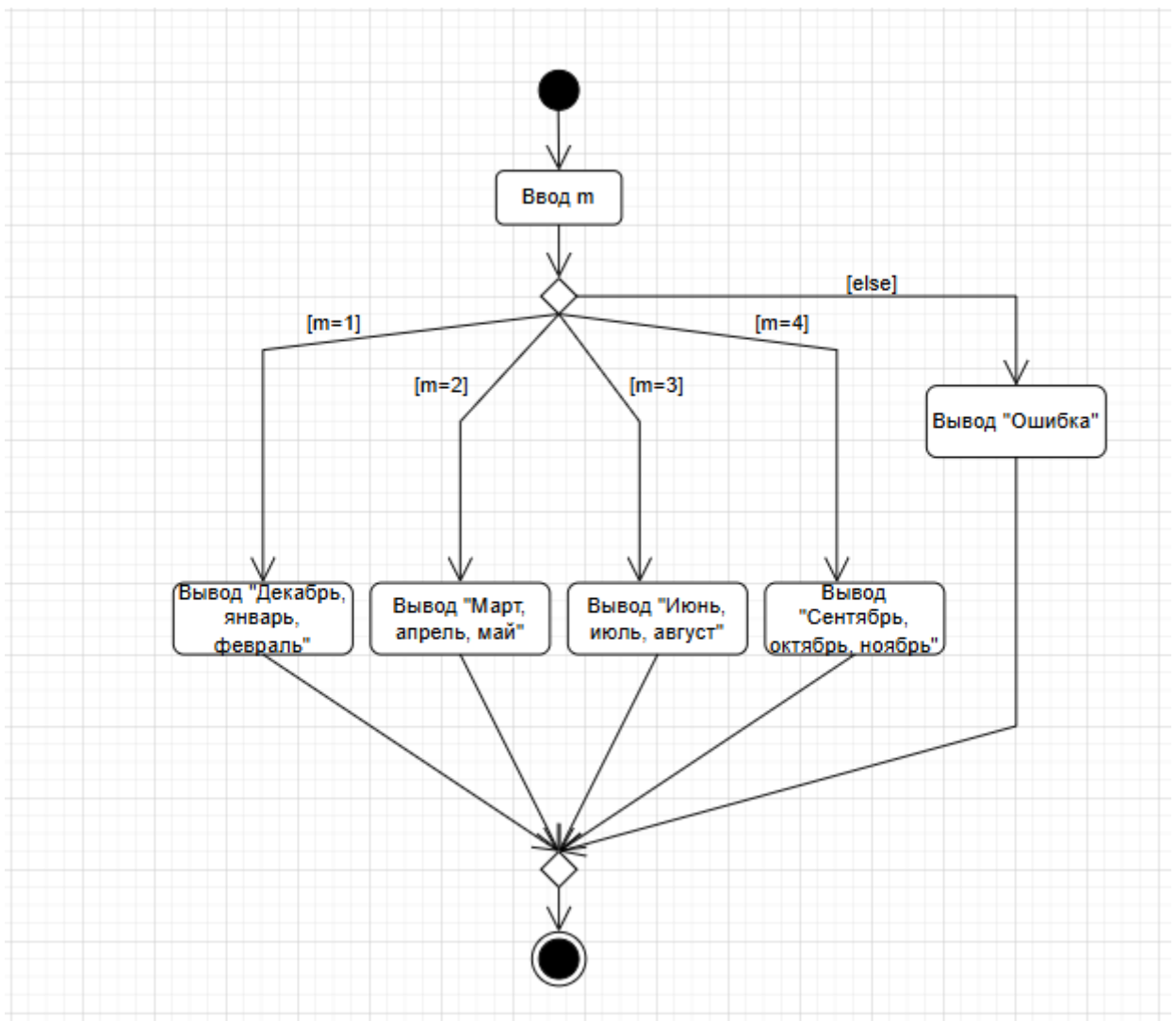


Рисунок 25. UML-диаграмма деятельности задания 1

9) Задание 2: даны произвольные действительные числа  $a$ ,  $b$  и  $c$ . Вывести на экран сообщения: треугольник с данными длинами сторон построить можно (указать равнобедренный, равносторонний или разносторонний получится треугольник), либо треугольник с данными длинами сторон построить нельзя.

```

1 ▶ #!/usr/bin/env python3
2   # -*- coding: utf-8 -*-
3
4 ▶ if __name__ == "__main__":
5     a = int(input("Введите число a: "))
6     b = int(input("Введите число b: "))
7     c = int(input("Введите число c: "))
8
9     if a + b > c and a + c > b and b + c > a:
10        if a == b and a == c:
11            print("С данными длинами сторон можно построить равносторонний треугольник")
12        elif a == b or a == c or b == c:
13            print("С данными длинами сторон можно построить равнобедренный треугольник")
14        else:
15            print("С данными длинами сторон можно построить разносторонний треугольник")
16    else:
17        print("С данными длинами сторон нельзя построить треугольник")

```

Рисунок 26. Задание 2

```

Задание 2 ×
D:\git\lab-3\venv\Scripts\python.exe "D:/git/lab-3/Задание 2.py"
Введите число a: 3
Введите число b: 3
Введите число c: 5
С данными длинами сторон можно построить равнобедренный треугольник

Process finished with exit code 0

```

Рисунок 27. Результат выполнения задания 2

```

D:\git\lab-3\venv\Scripts\python.exe "C:/Users/USER/lab-3/Задание 2.py"
Введите число a: 1
Введите число b: 0
Введите число c: 2
С данными длинами сторон нельзя построить треугольник

Process finished with exit code 0

```

Рисунок 28. Результат выполнения задания 2 с другими данными

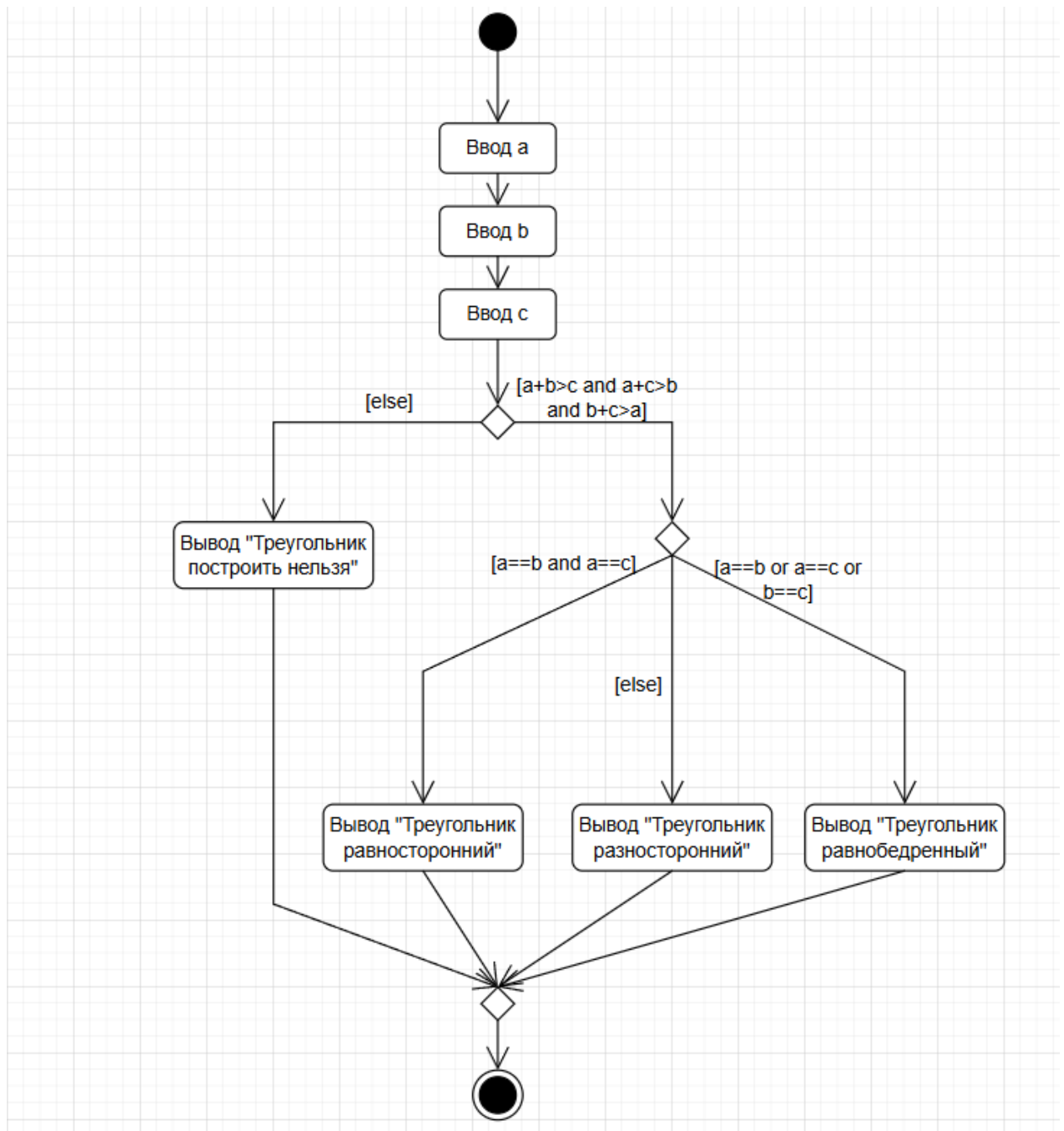


Рисунок 29. UML-диаграмма деятельности задания 2

10) Задание 3: составить программу, выдающую 1, если заданное число - простое и 0 - в противном случае. Число называется простым, если он делится только на 1 и на само себя. Делители числа лежат в интервале от 2 до корня из k, где k - заданное число.

```

1 ▶ #!/usr/bin/env python3
2   # -*- coding: utf-8 -*-
3
4   import math
5
6
7 ▶   if __name__ == "__main__":
8       k = int(input("Введите число: "))
9
10      if k < 2:
11          print(0)
12      else:
13          simple = 1
14          for i in range(2, int(math.sqrt(k)) + 1):
15              if k % i == 0:
16                  simple = 0
17                  break
18      print(simple)

```

Рисунок 30. Задание 3

```

D:\git\lab-3\venv\Scripts\python.exe "D:/git/lab-3/Задание 3.py"
Введите число: 17
1
Process finished with exit code 0

```

Рисунок 31. Результат выполнения задания 3

```

D:\git\lab-3\venv\Scripts\python.exe "D:/git/lab-3/Задание 3.py"
Введите число: 15
0
Process finished with exit code 0

```

Рисунок 32. Результат выполнения задания 3 с другими данными

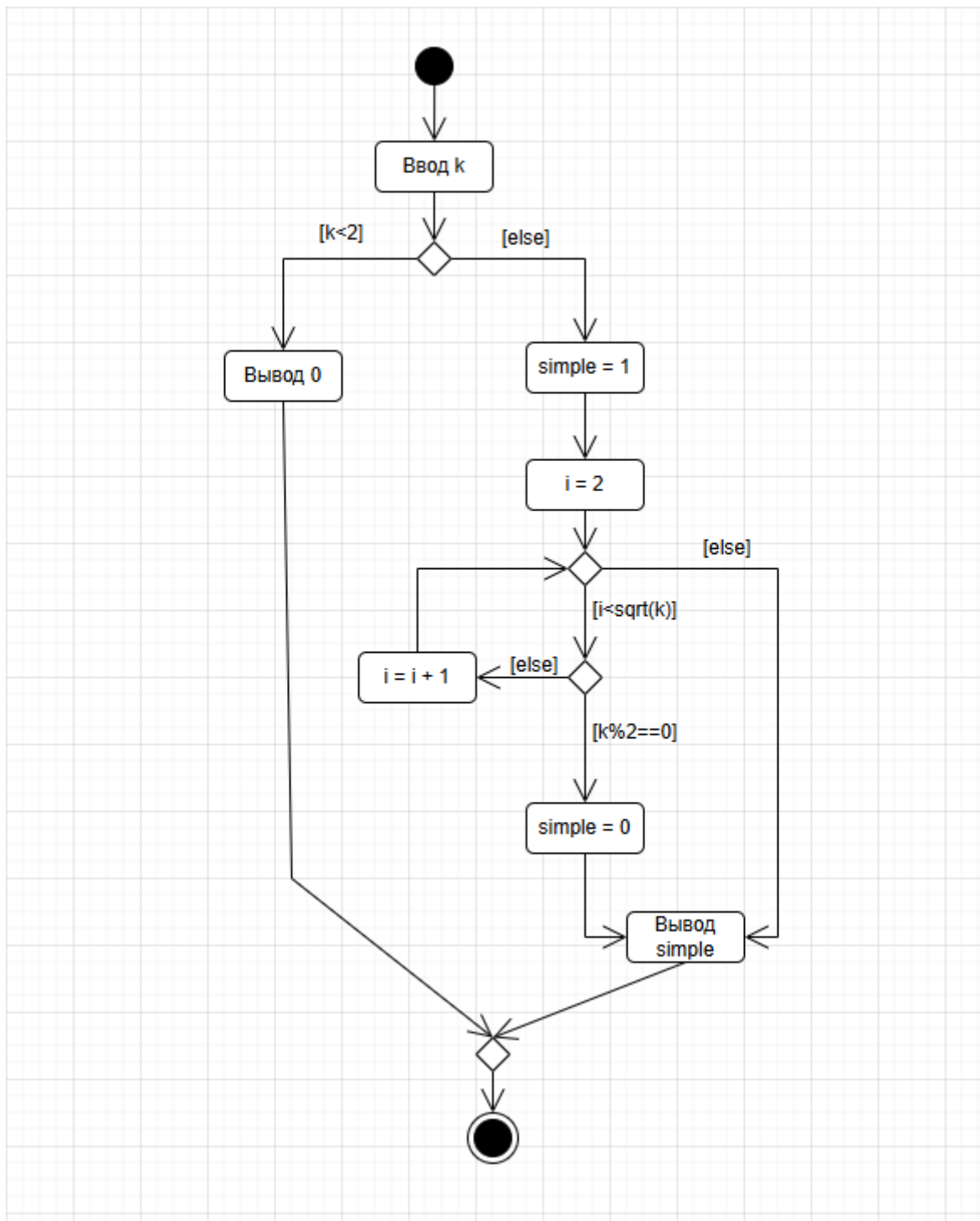


Рисунок 33. UML-диаграмма деятельности задания 3

11) Было выполнено задание повышенной сложности и построена UML-диаграмма деятельности.



```

1 ▶ #!/usr/bin/env python3
2   # -*- coding: utf-8 -*-
3
4   import math
5   import sys
6
7   EPS = 1e-10
8
9
10  ▶ if __name__ == "__main__":
11      x = float(input("Введите x (0 <= x <= 2): "))
12
13      # Если x = 1, то все члены ряда равны 0.
14      if x == 1:
15          print("f(1) = 0")
16      elif x < 0 or x > 2:
17          print("Некорректное значение x", file=sys.stderr)
18          exit(1)
19
20      else:
21          total = 0
22          k = 1
23
24          while True:
25              value = (-1)**k * (x - 1)**k / (k * k)
26              total += value
27              if math.fabs(value) < EPS:
28                  break
29              k += 1
30
31      print(f"f({x}) = {total}")

```

Рисунок 34. Задание повышенной сложности

```

Задание 4 ×
D:\git\lab-3\venv\Scripts\python.exe "D:/git/lab-3/Задание 4.py"
Введите x (0 <= x <= 2): 0
f(0.0) = 1.6449240669982403

Process finished with exit code 0

```

Рисунок 35. Результат выполнения задания повышенной сложности

```

D:\git\lab-3\venv\Scripts\python.exe "C:/Users/USER/lab-3/Задание 4.py"
Введите x (0 <= x <= 2): 5
Некорректное значение x

Process finished with exit code 1

```

Рисунок 36. Результат выполнения задания повышенной сложности с другими данными

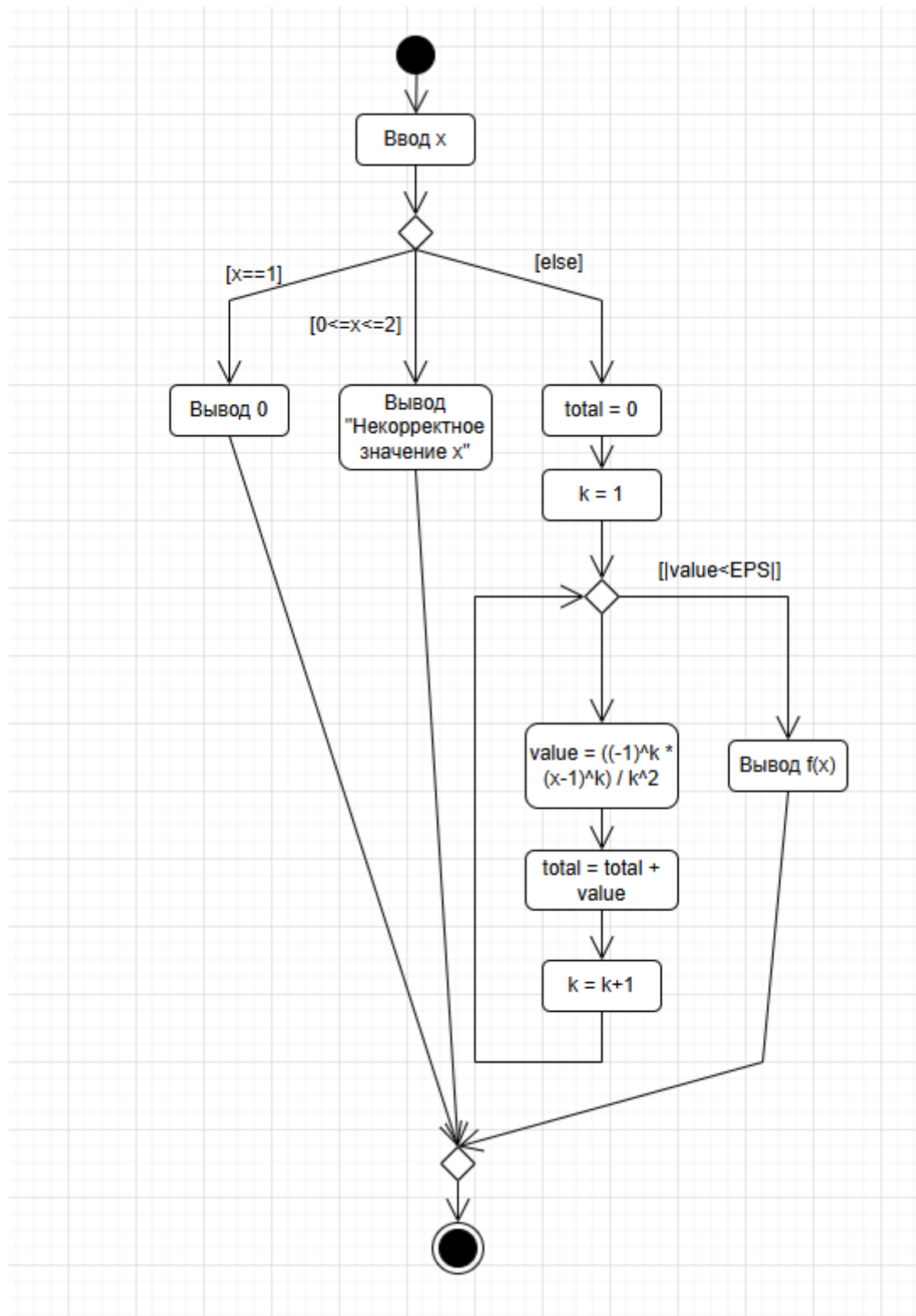


Рисунок 37. UML-диаграмма деятельности задания повышенной сложности

12) Были зафиксированы отправлены сделанные изменения на сервер GitHub.

```

USER@topPC MINGW64 ~/lab-3 (main)
$ git commit -m "Задание повышенной сложности"
[main dc38c39] Задание повышенной сложности
16 files changed, 231 insertions(+)
create mode 100644 .idea/.gitignore
create mode 100644 .idea/inspectionProfiles/Project_Default.xml
create mode 100644 .idea/inspectionProfiles/profiles_settings.xml
create mode 100644 .idea/lab-3.iml
create mode 100644 .idea/misc.xml
create mode 100644 .idea/modules.xml
create mode 100644 .idea/vcs.xml

USER@topPC MINGW64 ~/lab-3 (main)
$ git push origin main
Enumerating objects: 30, done.
Counting objects: 100% (22/22), done.
Delta compression using up to 12 threads
Compressing objects: 100% (19/19), done.
Writing objects: 100% (20/20), 4.59 KiB | 1.15 MiB/s, done.
Total 20 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/l1N322/lab-3.git
 a77fc16..dc38c39 main -> main

```

Рисунок 38. Отправка изменений

Ответы на контрольные вопросы:

1) Диаграммы деятельности. Унифицированный язык моделирования (UML) является стандартным инструментом для создания «чертежей» программного обеспечения. С помощью UML можно визуализировать, специфицировать, конструировать и документировать артефакты программных систем. UML пригоден для моделирования любых систем: от информационных систем масштаба предприятия до распределенных Web-приложений и даже встроенных систем реального времени. Это очень выразительный язык, позволяющий рассмотреть систему со всех точек зрения, имеющих отношение к ее разработке и последующему развертыванию. Несмотря на обилие выразительных возможностей, этот язык прост для понимания и использования.

2) Состояние действия и состояние деятельности. В потоке управления, моделируемом диаграммой деятельности, происходят различные события. Вы можете вычислить выражение, в результате чего изменяется значение некоторого атрибута или возвращается некоторое значение. Также, например, можно выполнить операцию над объектом, послать ему сигнал или даже создать его или уничтожить. Все эти выполняемые атомарные вычисления называются состояниями действия, поскольку каждое из них есть состояние

системы, представляющее собой выполнение некоторого действия. Состояния действия изображаются прямоугольниками с закругленными краями. Внутри такого символа можно записывать произвольное выражение. Состояния действия не могут быть подвергнуты декомпозиции. Кроме того, они атомарны. Это значит, что внутри них могут происходить различные события, но выполняемая в состоянии действия работа не может быть прервана. Обычно предполагается, что длительность одного состояния действия занимает неощутимо малое время. В противоположность этому состояния деятельности могут быть подвергнуты дальнейшей декомпозиции, вследствие чего выполняемую деятельность можно представить с помощью других диаграмм деятельности. Состояния деятельности не являются атомарными, то есть могут быть прерваны. Предполагается, что для их завершения требуется заметное время. Можно считать, что состояние действия - это частный вид состояния деятельности, а конкретнее - такое состояние, которое не может быть подвергнуто дальнейшей декомпозиции. А состояние деятельности можно представлять себе как составное состояние, поток управления которого включает только другие состояния деятельности и действий.

3) Переходы. Когда действие или деятельность в некотором состоянии завершается, поток управления сразу переходит в следующее состояние действия или деятельности. Для описания этого потока используются переходы, показывающие путь из одного состояния действия или деятельности в другое. В UML переход представляется простой линией со стрелкой.

Ветвление. Простые последовательные переходы встречаются наиболее часто, но их одних недостаточно для моделирования любого потока управления. Как и в блок-схеме, вы можете включить в модель ветвление, которое описывает различные пути выполнения в зависимости от значения некоторого булевского выражения. Как видно из рис. 4.3, точка ветвления представляется ромбом. В точку ветвления может входить ровно один переход, а выходить - два или более. Для каждого исходящего перехода

задается булевское выражение, которое вычисляется только один раз при входе в точку ветвления. Ни для каких двух исходящих переходов эти сторожевые условия не должны одновременно принимать значение «истина», иначе поток управления окажется неоднозначным. Но эти условия должны покрывать все возможные варианты, иначе поток остановится.

4) Алгоритм разветвляющейся структуры - это алгоритм, в котором вычислительный процесс осуществляется либо по одной, либо по другой ветви, в зависимости от выполнения некоторого условия. Программа разветвляющейся структуры реализует такой алгоритм. В программе разветвляющейся структуры имеется один или несколько условных операторов. Для программной реализации условия используется логическое выражение. В сложных структурах с большим числом ветвей применяют оператор выбора.

5) Разветвляющиеся алгоритмы отличаются от линейных тем, что в них последовательность выполнения команд зависит от выполнения условия.

6) Условный оператор в программировании — это конструкция, которая проверяет истинность некоторого логического выражения (условия) и в зависимости от результата выполняет тот или иной блок кода. Существуют следующие его формы: конструкция if, конструкция if – else, конструкция if – elif – else

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

8) Простое условие — это одиночное логическое выражение, которое проверяет одно утверждение. Оно возвращает True или False и состоит из одного оператора сравнения или логического оператора.

Пример:

```
x = 5
```

```
if x == 5:
```

```
    print("x равен 5")
```

9) Составное условие в Python — это конструкция, которая проверяет несколько условий и выполняет только подходящий код. Для этого используются операторы if, elif (сокращение от «else if») и else:

Пример:

```
x = 2
```

```
y = 5
```

```
if x > 3:
```

```
    print(y+x)
```

```
elif x < 3:
```

```
    print(y-x)
```

```
else:
```

```
    print(y*x)
```

10) При составлении сложных условий широко используются два оператора - так называемые логические И (and) и ИЛИ (or). Чтобы получить True при использовании оператора and, необходимо, чтобы результаты обоих простых выражений, которые связывает данный оператор, были истинными. Если хотя бы в одном случае результатом будет False, то и все сложное выражение будет ложным. Чтобы получить True при использовании оператора or, необходимо, чтобы результат хотя бы одного простого выражения, входящего в состав сложного, был истинным. В случае оператора or сложное выражение становится ложным лишь тогда, когда ложны оба составляющие его простые выражения.

11) Да, оператор ветвления может содержать внутри себя другие ветвления (вложенные ветвления). В этом случае один оператор ветвления можно расположить внутри другого, что позволяет производить выбор более чем из двух вариантов.

12) Алгоритм циклической структуры это тот, который позволяет многократно выполнять одни и те же действия в зависимости от заранее определённых условий.

13) В Python есть два основных типа циклов: `for` и `while`. Оператор цикла `while` выполняет указанный набор инструкций до тех пор, пока условие цикла истинно. Истинность условия определяется также как и в операторе `if`. Оператор `for` выполняет указанный набор инструкций заданное количество раз, которое определяется количеством элементов в наборе.

14) Функция `range` возвращает неизменяемую последовательность чисел в виде объекта `range`. Она хранит только информацию о значениях `start`, `stop` и `step` и вычисляет значения по мере необходимости. Это значит, что независимо от размера диапазона, который описывает функция `range`, она всегда будет занимать фиксированный объем памяти.

15) Чтобы организовать перебор значений от 15 до 0 с шагом 2 нужно выполнить следующую программу:

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

16) Да, циклы могут быть вложенными.

17) Бесконечный цикл образуется, если в условии цикла `while` не предусмотрено условие выхода. Для выхода из бесконечного цикла в Python можно использовать оператор `break`.

18) Оператор `break` предназначен для досрочного прерывания работы цикла `while`.

19) Оператор `continue` запускает цикл заново, при этом код, расположенный после данного оператора, не выполняется.

20) В операционной системе по умолчанию присутствуют стандартных потока вывода на консоль: буферизованный поток `stdout` для вывода данных и информационных сообщений, а также небуферизованный поток `stderr` для вывода сообщений об ошибках. По умолчанию функция `print` использует поток `stdout`. Для того, чтобы использовать поток `stderr` необходимо передать его в параметре `file` функции `print`. Само же определение потоков `stdout` и `stderr` находится в стандартном пакете Python `sys`. Хорошим стилем программирования является наличие вывода ошибок в стандартный поток

stderr поскольку вывод в потоки stdout и stderr может обрабатываться как операционной системой, так и сценариями пользователя по-разному.

21) Для того, чтобы использовать поток stderr необходимо передать его в параметре file функции print: `print("Error!", file=sys.stderr)`

22) В Python завершить программу и передать операционной системе заданный код возврата можно посредством функции `exit`. Вызов `exit(1)` (0 стоит по умолчанию) приводит к немедленному завершению программы и операционной системе передается 1 в качестве кода возврата, что говорит о том, что в процессе выполнения программы произошли ошибки.

**Вывод:** в ходе работы были приобретены навыки программирования разветвляющихся алгоритмов и алгоритмов циклической структуры. Были освоены операторы языка Python версии 3.x `if`, `while`, `for`, `break` и `continue`, позволяющих реализовывать разветвляющиеся алгоритмы и алгоритмы циклической структуры.