

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

Отчет по лабораторной работе № 2.2

**«Условные операторы и циклы в языке Python»
по дисциплине «Основы программной инженерии»**

Выполнил студент группы

ПИЖ-б-о-21-1

Зиберов Александр

« » ноября 2022 г.

Подпись студента _____

Работа защищена

« » _____ 20__ г.

Проверил Воронкин Р.А. _____
(подпись)

Ставрополь, 2022

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.

Выполнение работы:

Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT, рисунок 1.

Ссылка: <https://github.com/afk552/lab2.2>

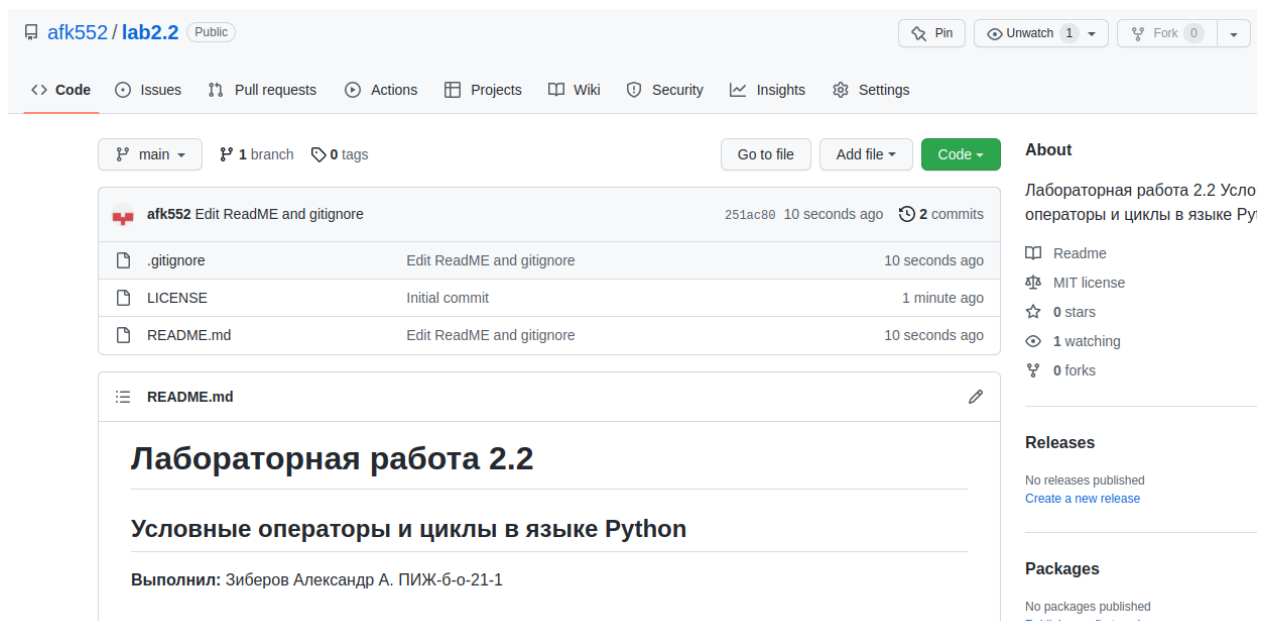
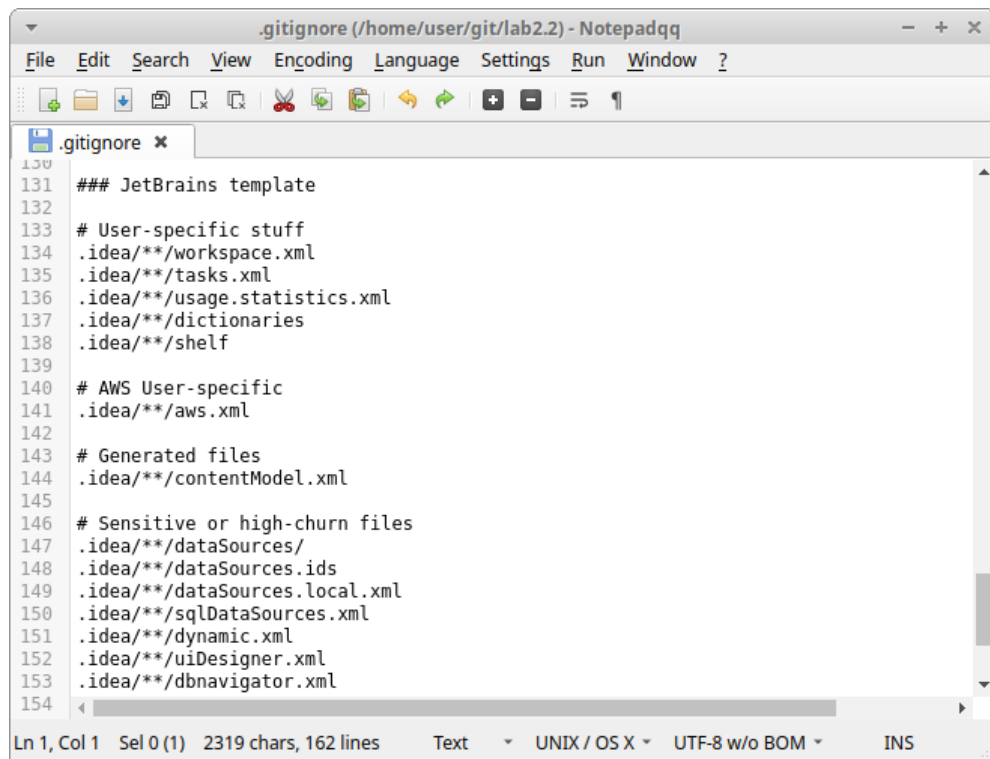


Рисунок 1 – Репозиторий GitHub


Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm, рисунок 2.



```
.gitignore (/home/user/git/lab2.2) - Notepad++
File Edit Search View Encoding Language Settings Run Window ?
.gitignore x
130
131 ### JetBrains template
132
133 # User-specific stuff
134 .idea/**/workspace.xml
135 .idea/**/tasks.xml
136 .idea/**/usage.statistics.xml
137 .idea/**/dictionaries
138 .idea/**/shelf
139
140 # AWS User-specific
141 .idea/**/aws.xml
142
143 # Generated files
144 .idea/**/contentModel.xml
145
146 # Sensitive or high-churn files
147 .idea/**/dataSources/
148 .idea/**/dataSources.ids
149 .idea/**/dataSources.local.xml
150 .idea/**/sqlDataSources.xml
151 .idea/**/dynamic.xml
152 .idea/**/uiDesigner.xml
153 .idea/**/dbnavigator.xml
154
Ln 1, Col 1 Sel 0 (1) 2319 chars, 162 lines Text UNIX / OS X UTF-8 w/o BOM INS
```

Рисунок 2 – Окно блокнота

Организируйте свой репозиторий в соответствии с моделью ветвления git-flow, рисунок 3.



```
Terminal - user@computer: ~/git/lab2.2
File Edit View Terminal Tabs Help
user@computer:~/git/lab2.2$ git branch
* main
user@computer:~/git/lab2.2$ git branch develop
user@computer:~/git/lab2.2$ git checkout develop
Switched to branch 'develop'
user@computer:~/git/lab2.2$ git branch
* develop
  main
user@computer:~/git/lab2.2$ git push 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/afk552/lab2.2/pull/new/develop
remote:
To https://github.com/afk552/lab2.2
 * [new branch]      develop -> develop
user@computer:~/git/lab2.2$
```

Рисунок 3 – Окно командной строки

Создайте проект PyCharm в папке репозитория, рисунок 4.

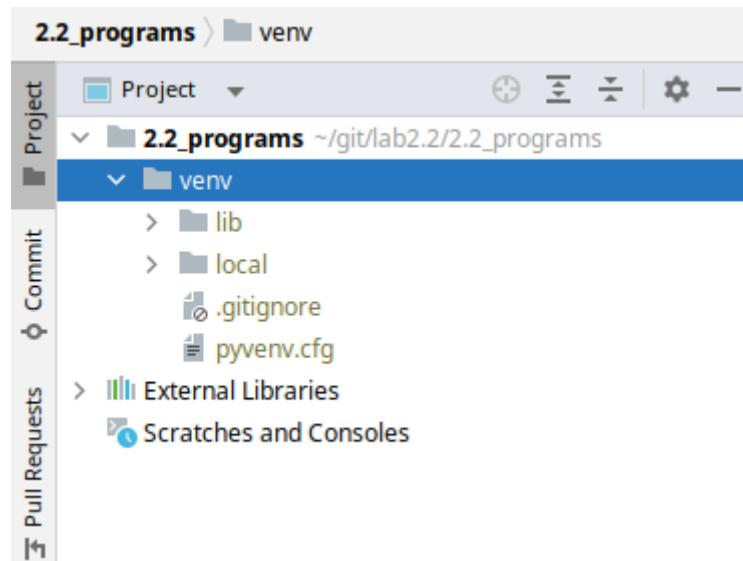


Рисунок 4 – Окно проекта в PyCharm

Выполните оформление исходного примеров лабораторной работы и индивидуальных созданий в соответствии с РЕР-8. Проработайте примеры лабораторной работы. Создайте для каждого примера отдельный модуль языка Python. Рисунок 5. Зафиксируйте изменения в репозитории, рисунок 6.

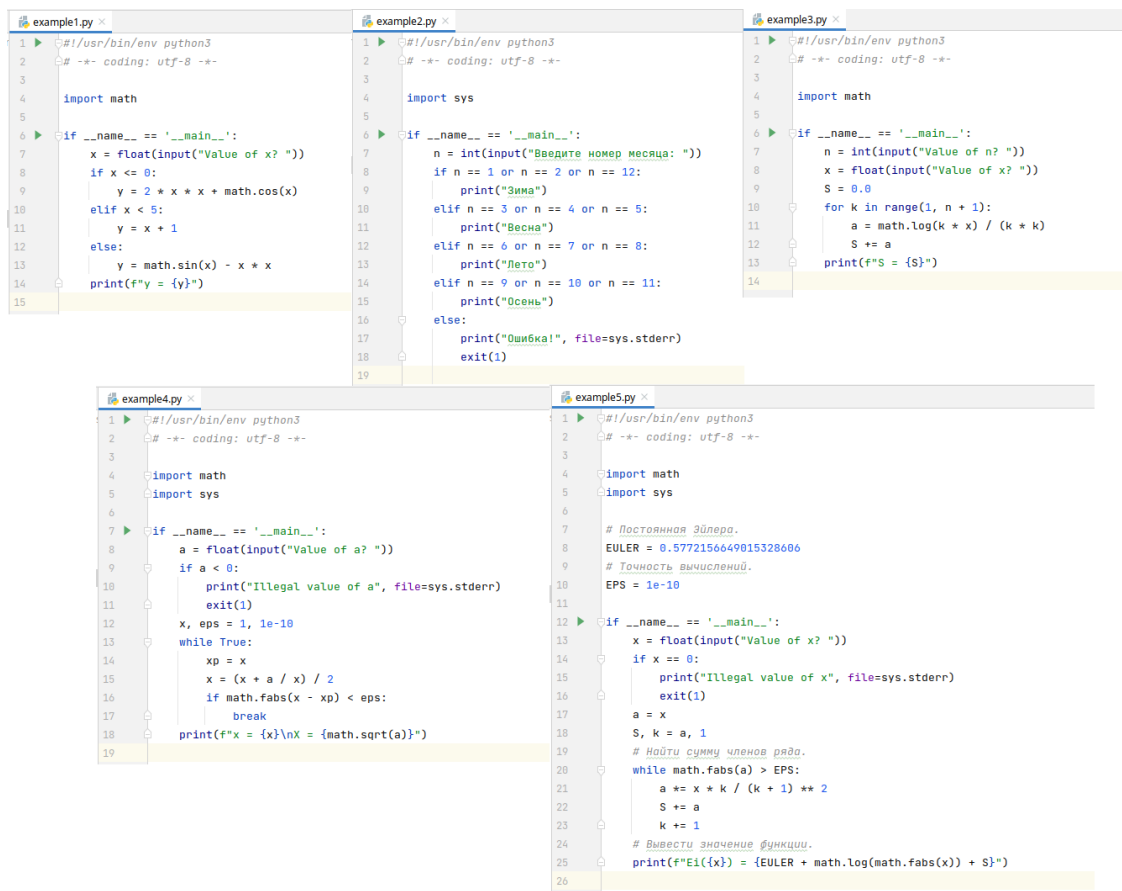


Рисунок 5 – Окно редактора кода PyCharm

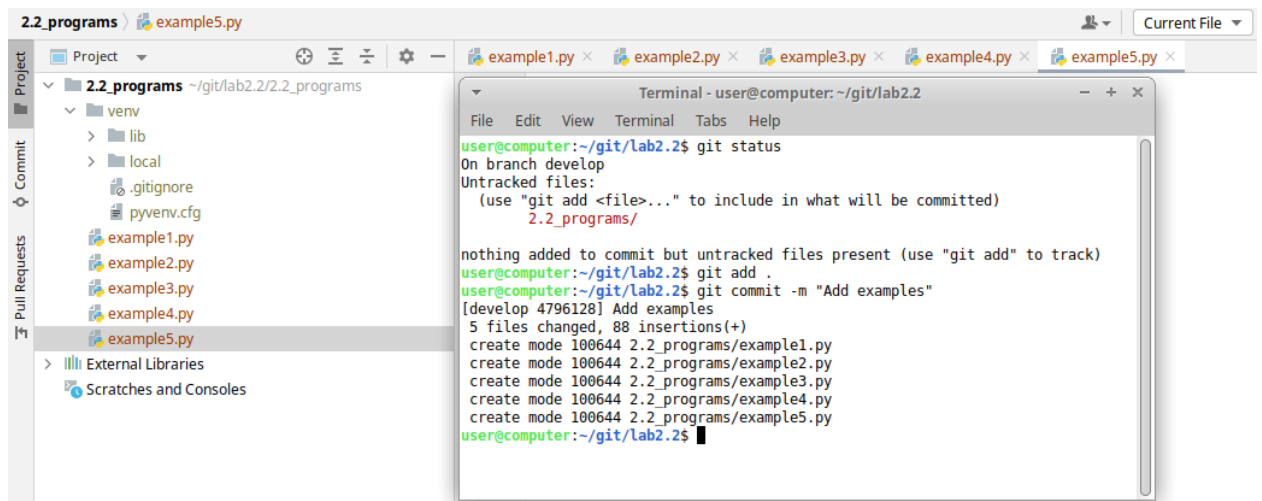


Рисунок 6 – Окно командной строки

Приведите в отчете скриншоты результатов выполнения каждой из программ примеров при различных исходных данных, вводимых с клавиатуры.

Пример 1, результаты выполнения изображены на рисунке 7.

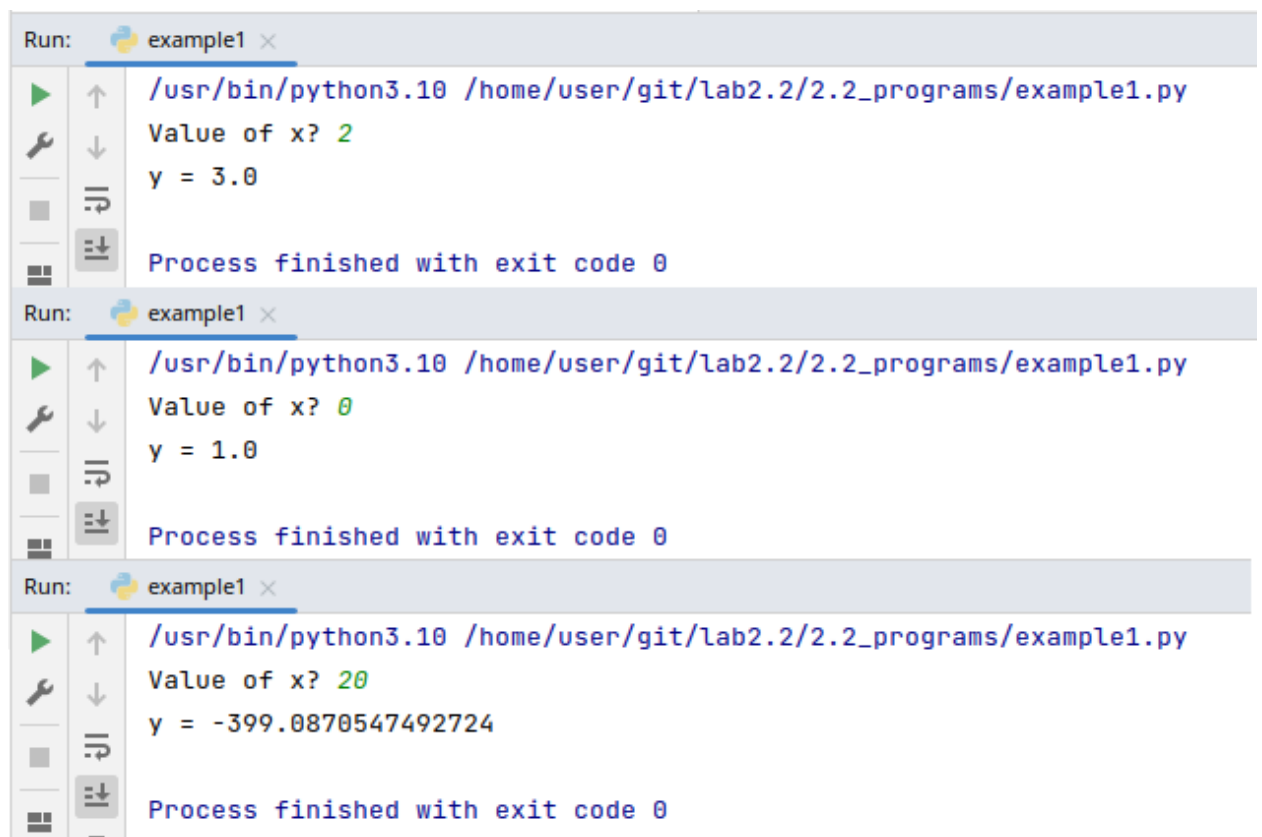
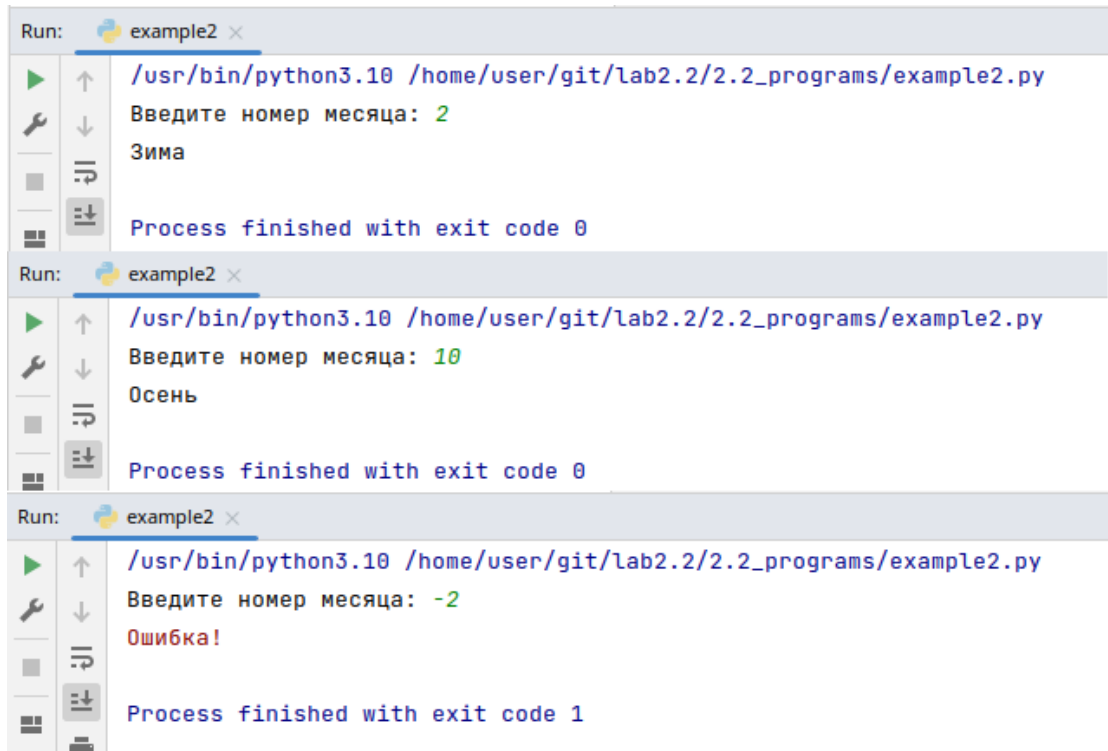


Рисунок 7 – Окно вывода PyCharm

Пример 2, результаты выполнения изображены на рисунке 8.



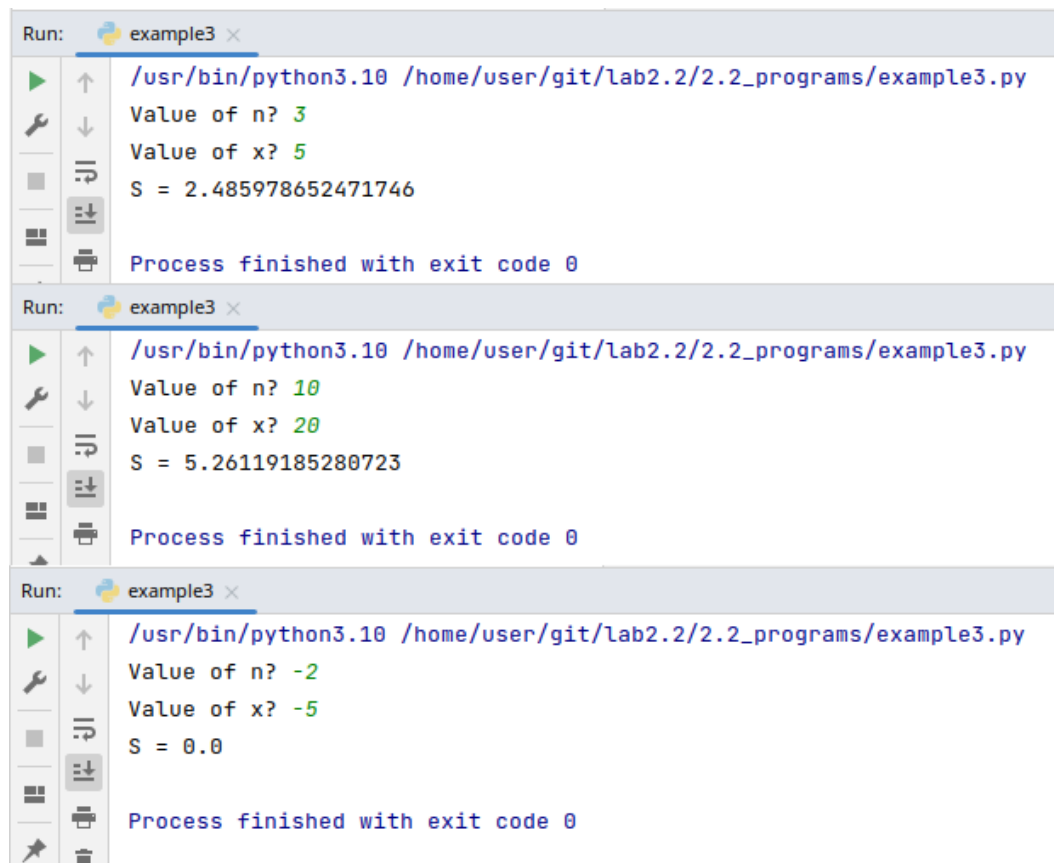
```
Run: example2 x
/usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/example2.py
Введите номер месяца: 2
Зима
Process finished with exit code 0

Run: example2 x
/usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/example2.py
Введите номер месяца: 10
Осень
Process finished with exit code 0

Run: example2 x
/usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/example2.py
Введите номер месяца: -2
Ошибка!
Process finished with exit code 1
```

Рисунок 8 – Окно вывода PyCharm

Пример 3, результаты выполнения изображены на рисунке 9.



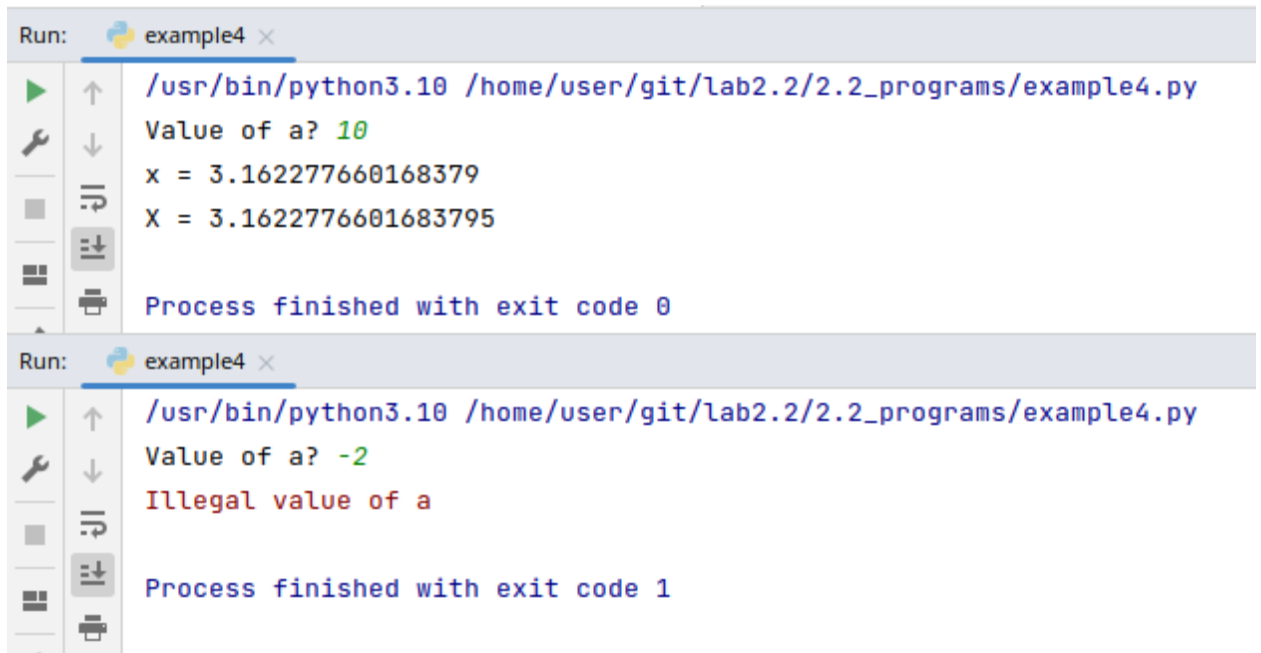
```
Run: example3 x
/usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/example3.py
Value of n? 3
Value of x? 5
S = 2.485978652471746
Process finished with exit code 0

Run: example3 x
/usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/example3.py
Value of n? 10
Value of x? 20
S = 5.26119185280723
Process finished with exit code 0

Run: example3 x
/usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/example3.py
Value of n? -2
Value of x? -5
S = 0.0
Process finished with exit code 0
```

Рисунок 9 – Окно вывода PyCharm

Пример 4, результаты выполнения изображены на рисунке 10.



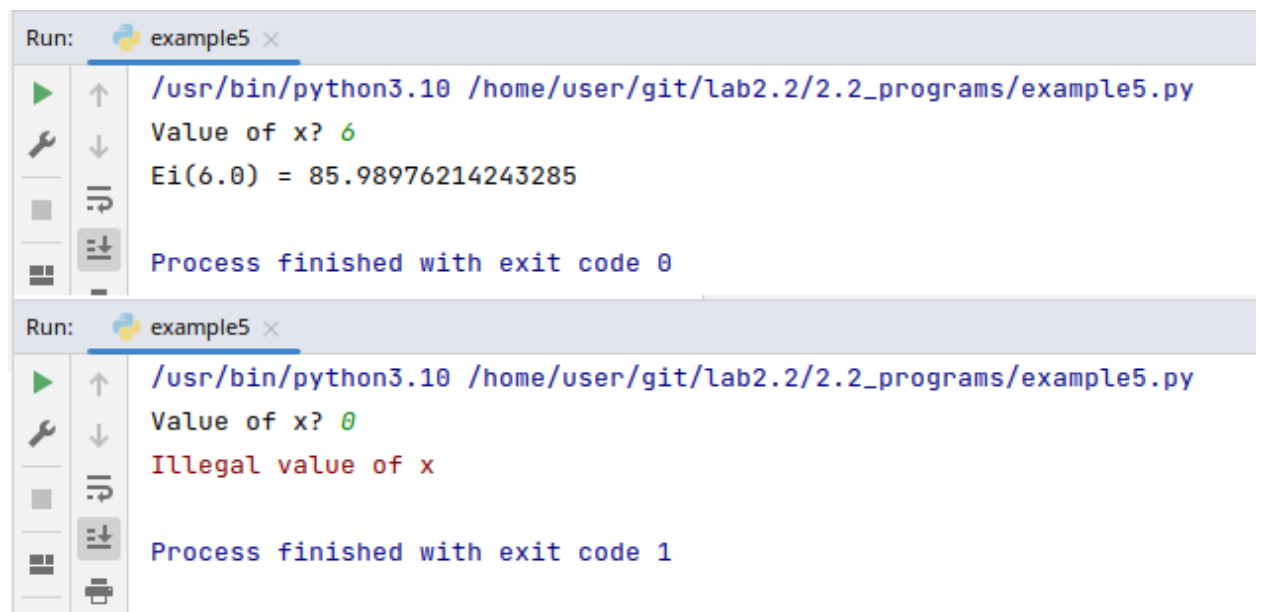
The screenshot shows the PyCharm Run window for 'example4'. It contains two successful runs and one failed run. The first run shows the program executing successfully with exit code 0. The second run shows the program executing successfully with exit code 0. The third run shows the program failing with exit code 1 due to an illegal value of 'a'.

```
Run: example4 x
/usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/example4.py
Value of a? 10
x = 3.162277660168379
X = 3.1622776601683795
Process finished with exit code 0

Run: example4 x
/usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/example4.py
Value of a? -2
Illegal value of a
Process finished with exit code 1
```

Рисунок 10 – Окно вывода PyCharm

Пример 5, результаты выполнения изображены на рисунке 11.



The screenshot shows the PyCharm Run window for 'example5'. It contains two successful runs and one failed run. The first run shows the program executing successfully with exit code 0. The second run shows the program executing successfully with exit code 0. The third run shows the program failing with exit code 1 due to an illegal value of 'x'.

```
Run: example5 x
/usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/example5.py
Value of x? 6
Ei(6.0) = 85.98976214243285
Process finished with exit code 0

Run: example5 x
/usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/example5.py
Value of x? 0
Illegal value of x
Process finished with exit code 1
```

Рисунок 11 – Окно вывода PyCharm

Зафиксируйте изменения в репозитории, рисунок 12.

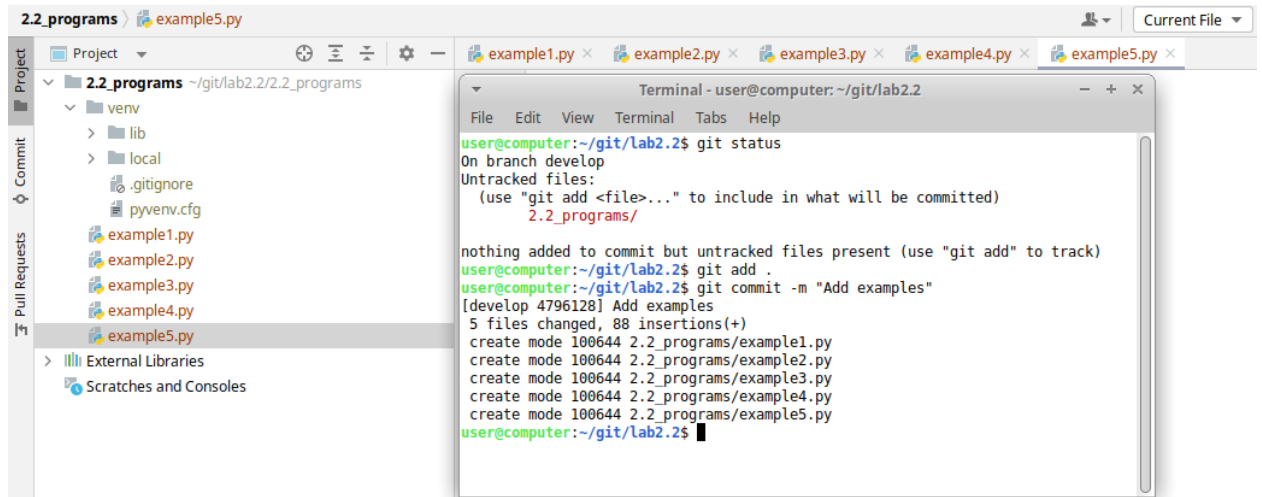


Рисунок 12 – Окно командной строки

Для примеров 4 и 5 постройте UML-диаграмму деятельности. Для построения диаграмм деятельности использовать веб-сервис [Google diagrams.net](https://diagrams.net).

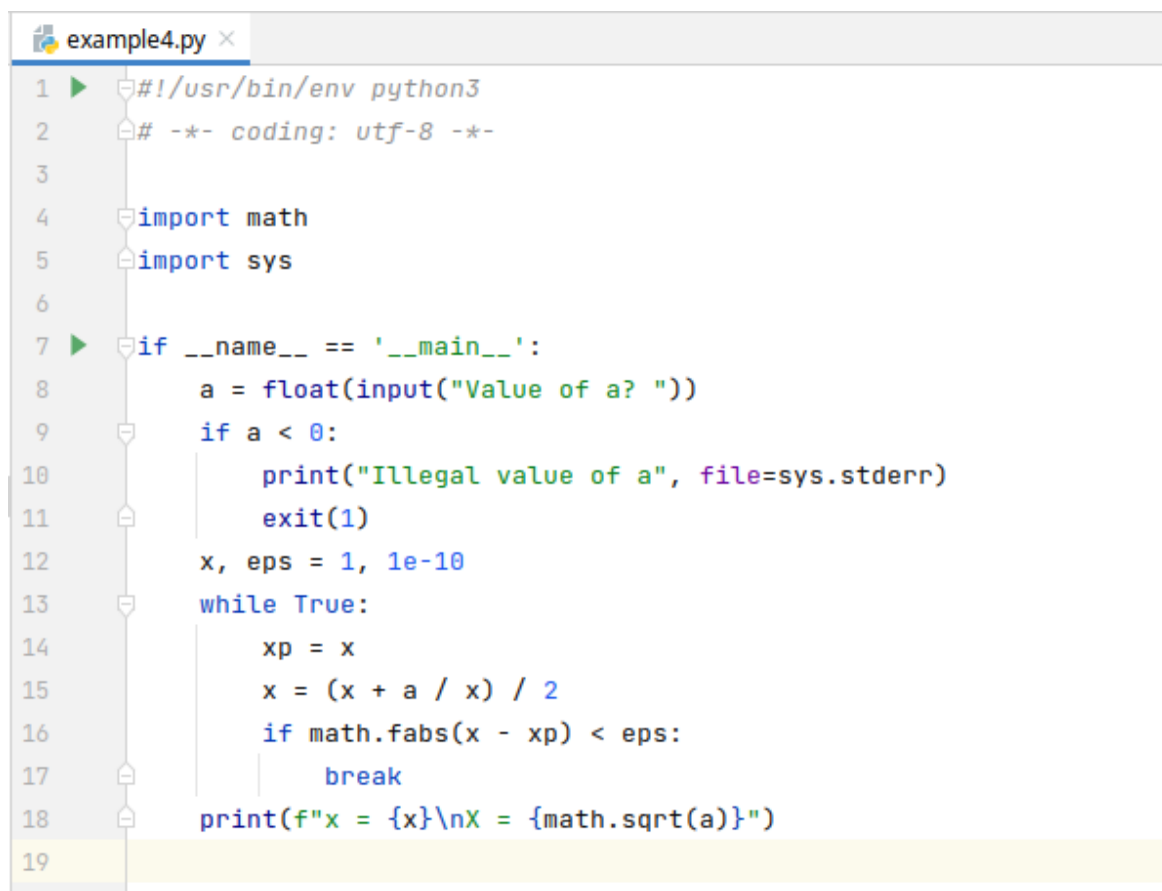


Рисунок 13 – Листинг примера 4

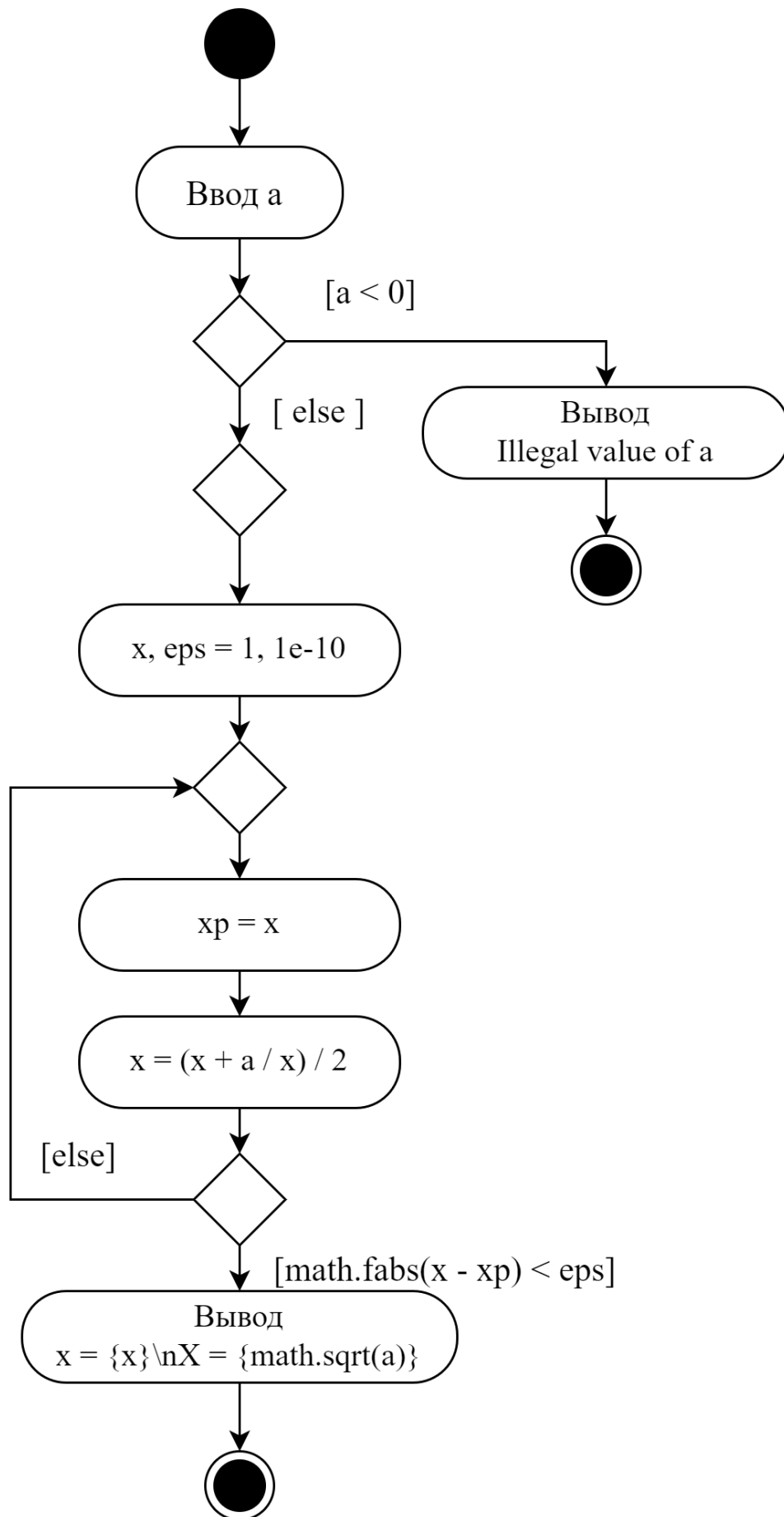


Рисунок 14 – UML-диаграмма деятельности программы из примера 4

```

example5.py x
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  # Точность вычислений.
10 EPS = 1e-10
11
12 ▶ if __name__ == '__main__':
13   x = float(input("Value of x? "))
14   if x == 0:
15     print("Illegal value of x", file=sys.stderr)
16     exit(1)
17   a = x
18   S, k = a, 1
19   # Найти сумму членов ряда.
20   while math.fabs(a) > EPS:
21     a *= x * k / (k + 1) ** 2
22     S += a
23     k += 1
24   # Вывести значение функции.
25   print(f"Ei({x}) = {EULER + math.log(math.fabs(x)) + S}")
26

```

Рисунок 15 – Листинг примера 5

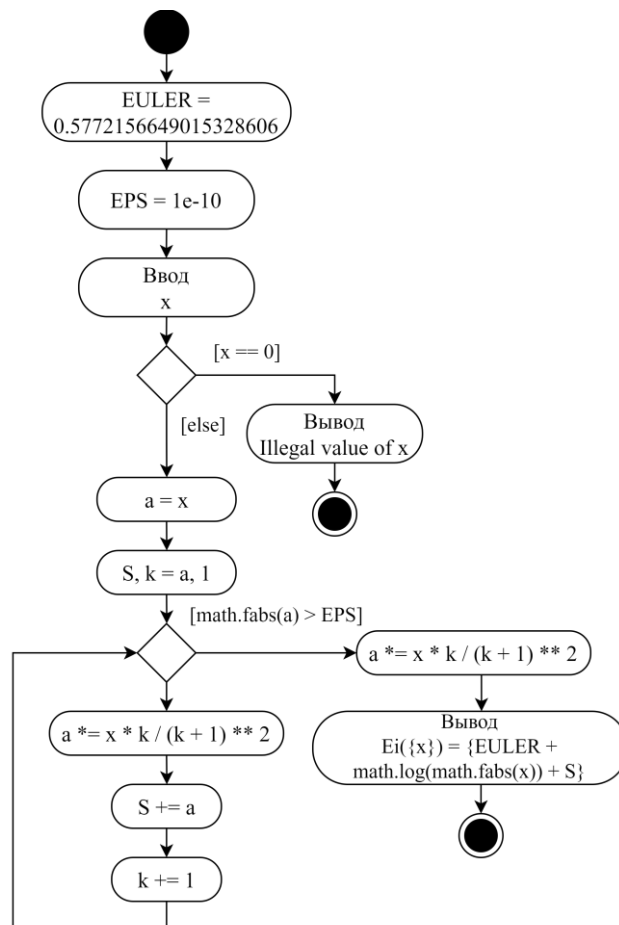


Рисунок 16 – UML-диаграмма деятельности программы из примера 5

Выполните индивидуальные задания, согласно своего варианта. Приведите в отчете скриншоты работы программ и UML-диаграммы деятельности решения индивидуальных заданий.

Вариант 11.

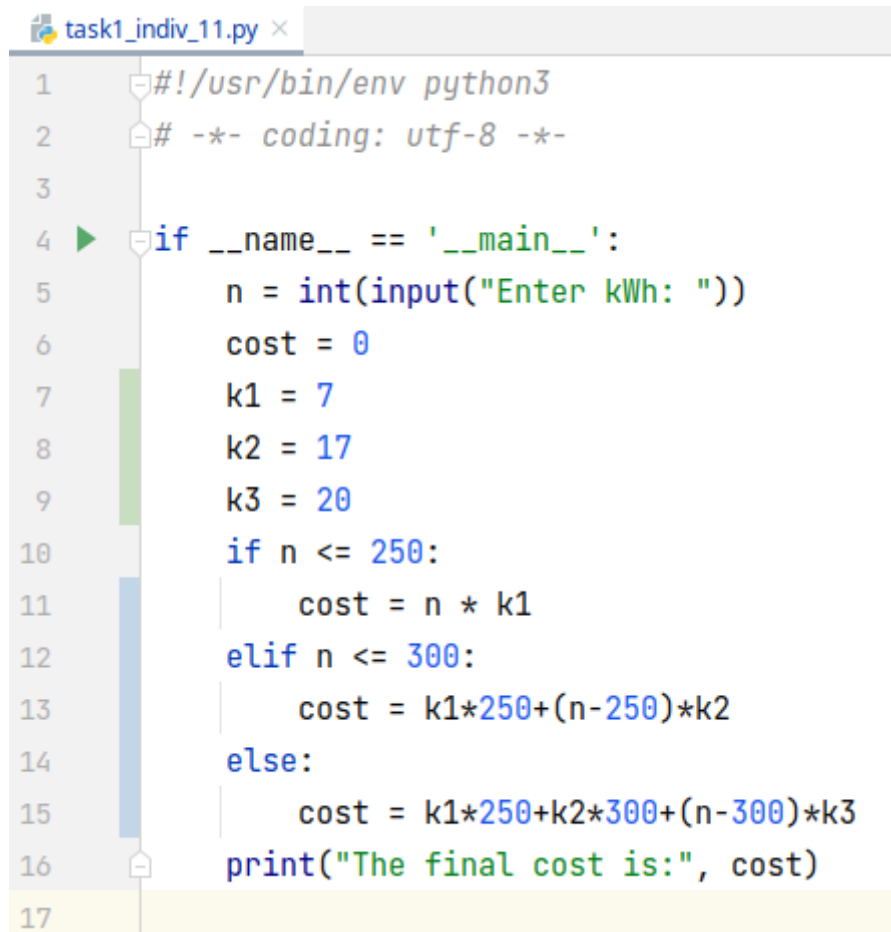
Задание 1. Компания по снабжению электроэнергией взимает плату с клиентов по тарифу:

7 р. за 1 кВт/ч за первые 250 кВт/ч;

17 р. за кВт/ч, если потребление свыше 250, но не превышает 300 кВт/ч;

20 р. за кВт/ч, если потребление свыше 300 кВт/ч.

Потребитель израсходовал кВт/ч. Подсчитать плату. Рисунки 17, 18, 19.



```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  if __name__ == '__main__':
5      n = int(input("Enter kWh: "))
6      cost = 0
7      k1 = 7
8      k2 = 17
9      k3 = 20
10     if n <= 250:
11         cost = n * k1
12     elif n <= 300:
13         cost = k1*250+(n-250)*k2
14     else:
15         cost = k1*250+k2*300+(n-300)*k3
16     print("The final cost is:", cost)
17
```

Рисунок 17 – Листинг программы индивидуального задания 1

```

Run: task1_indiv_11 x
  /usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/task1_indiv_11.py
  Enter kWh: 43
  The final cost is: 301

Run: task1_indiv_11 x
  /usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/task1_indiv_11.py
  Enter kWh: 253
  The final cost is: 2583

Run: task1_indiv_11 x
  /usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/task1_indiv_11.py
  Enter kWh: 299
  The final cost is: 2583

Run: task1_indiv_11 x
  /usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/task1_indiv_11.py
  Enter kWh: 301
  The final cost is: 6870

```

Рисунок 18 – Результат выполнения программы индивидуального задания 1

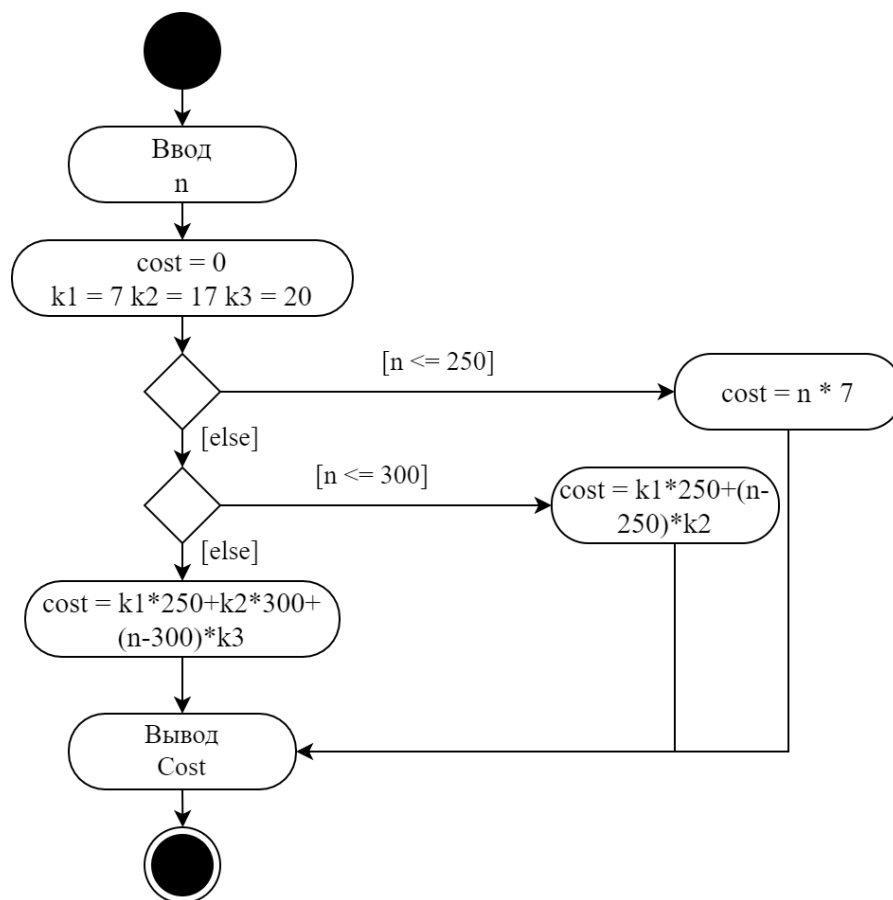
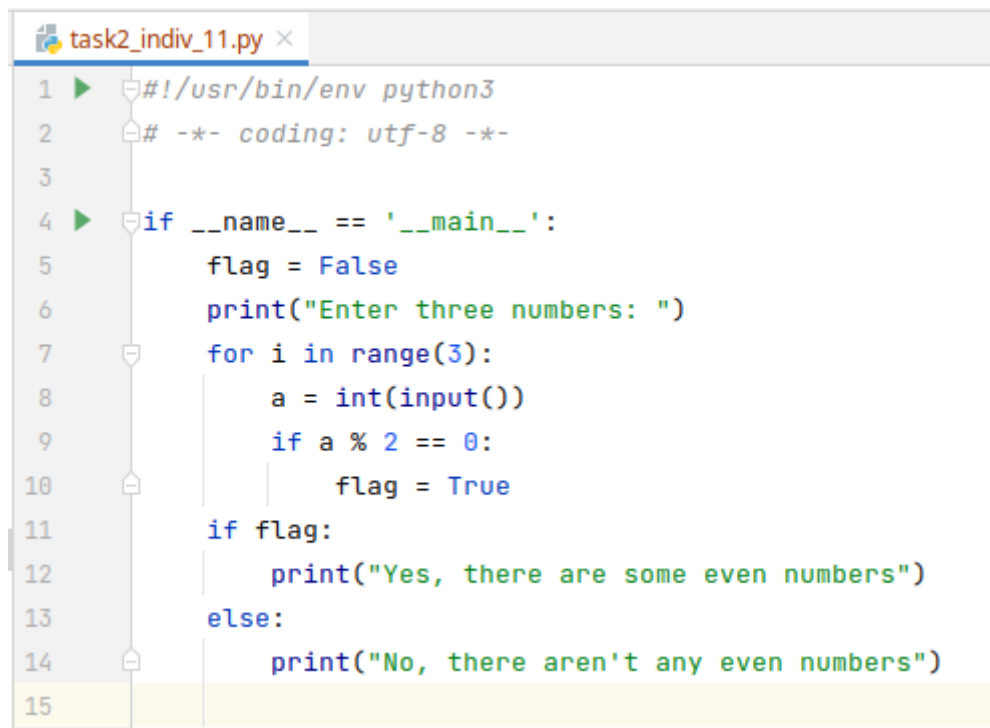


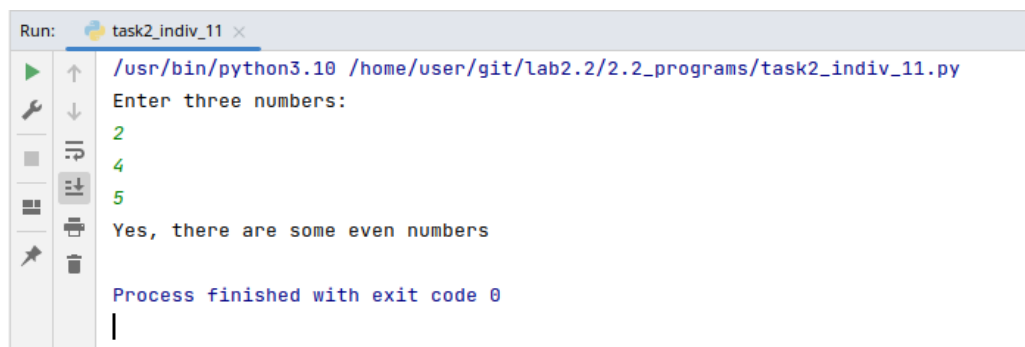
Рисунок 19 – UML-диаграмма деятельности программы индивидуального задания 1

Задание 2. Определить, есть ли среди трёх заданных чисел чётные.
Рисунки 20-23.



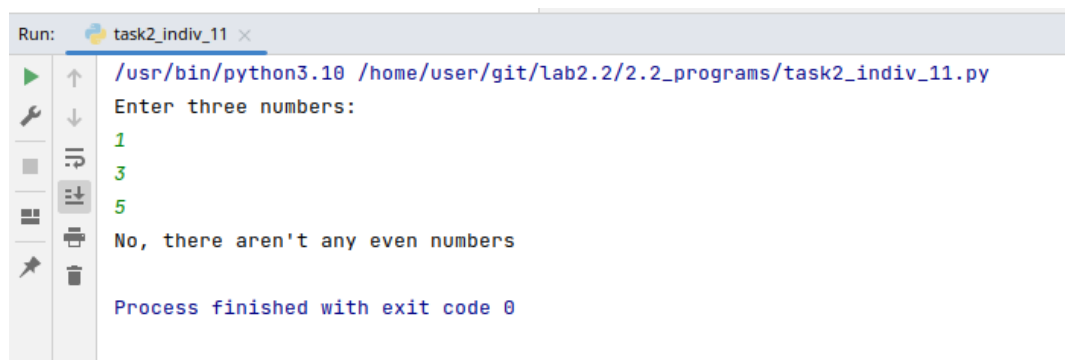
```
task2_indiv_11.py x
1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4  ▶  if __name__ == '__main__':
5      flag = False
6      print("Enter three numbers: ")
7      for i in range(3):
8          a = int(input())
9          if a % 2 == 0:
10             flag = True
11
12     if flag:
13         print("Yes, there are some even numbers")
14     else:
15         print("No, there aren't any even numbers")
```

Рисунок 20 – Листинг программы индивидуального задания 2



```
Run: task2_indiv_11 x
/usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/task2_indiv_11.py
Enter three numbers:
2
4
5
Yes, there are some even numbers
Process finished with exit code 0
|
```

Рисунок 21 – Результат 1 выполнения программы индивидуального задания 2



```
Run: task2_indiv_11 x
/usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/task2_indiv_11.py
Enter three numbers:
1
3
5
No, there aren't any even numbers
Process finished with exit code 0
```

Рисунок 22 – Результат 2 выполнения программы индивидуального задания 2

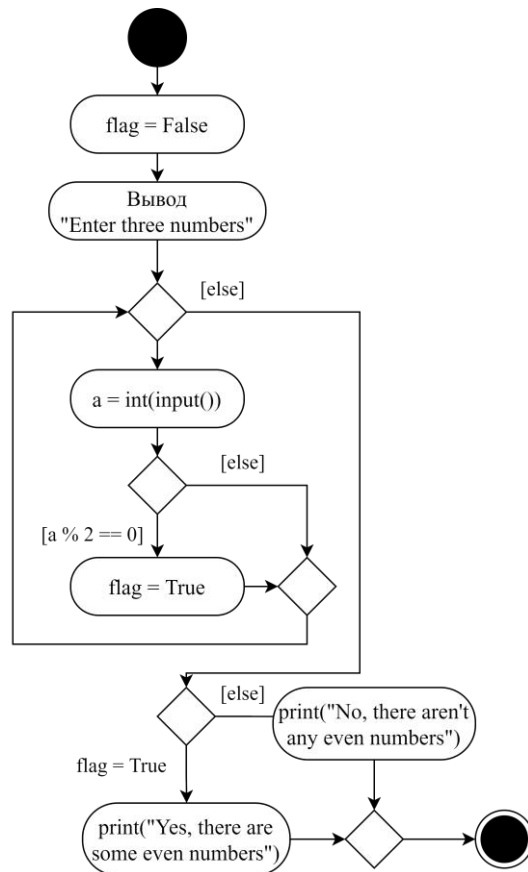


Рисунок 23 – UML-диаграмма деятельности программы индивидуального задания 2

Задание 3. Составьте программу, которая печатает таблицу умножения натуральных чисел в десятичной системе счисления. Рисунки 22, 23, 24.

```

task3_indiv_11.py x
1 ▶ #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4 ▶ if __name__ == '__main__':
5     for i in range(2, 9+1):
6         for j in range(1, 9+1):
7             print("{0} * {1} = {2}".format(i, j, i*j))
8
  
```

Рисунок 22 – Листинг программы индивидуального задания 3

```
Run: task3_indiv_11 x
7 * 8 = 56
7 * 9 = 63
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
Process finished with exit code 0
```

Рисунок 23 – Результат выполнения программы индивидуального задания 3

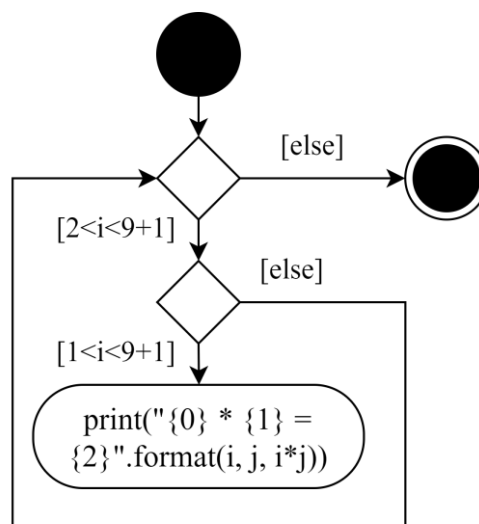


Рисунок 24 – UML-диаграмма деятельности программы индивидуального задания 3

Зафиксируйте сделанные изменения в репозитории (индивидуальные задания), рисунок 25.

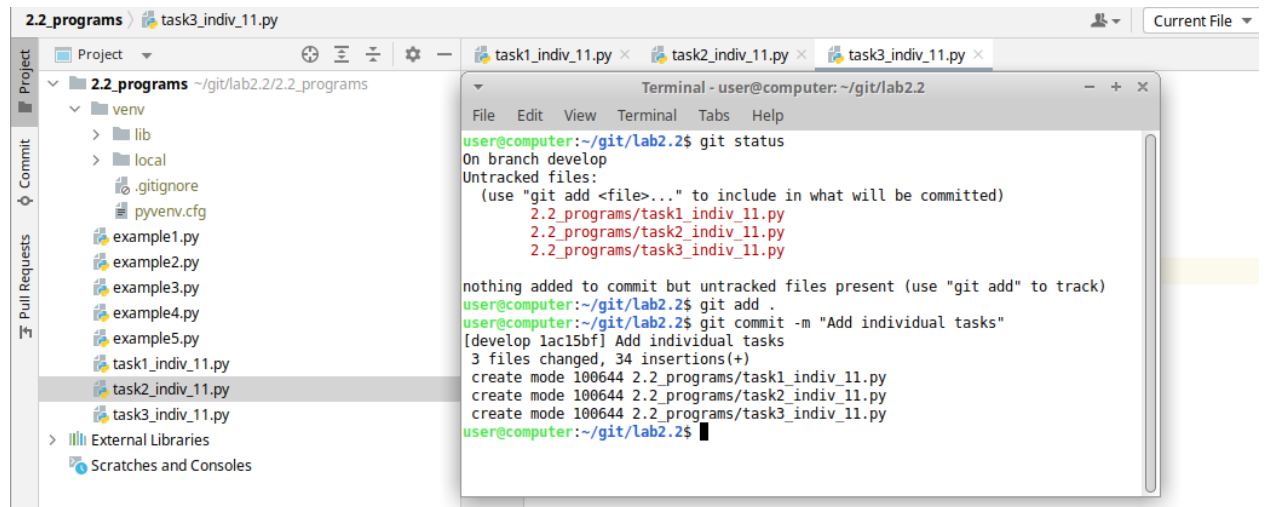


Рисунок 25 – Окно командной строки

Выполните слияние ветки для разработки с веткой main / master. Отправьте сделанные изменения на сервер GitHub. Рисунки 26, 27.



Рисунок 26 – Окно командной строки

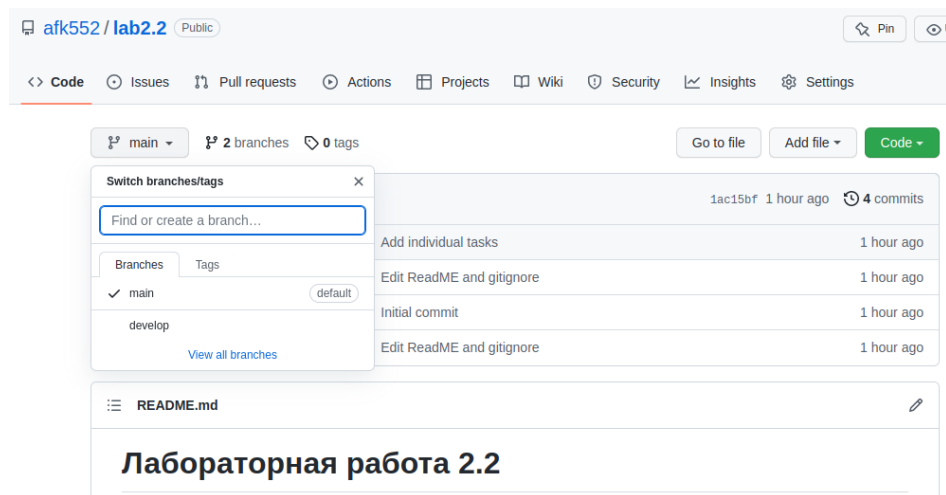


Рисунок 27 – Удаленный репозиторий на GitHub

Составить UML-диаграмму деятельности, программу и произвести вычисления значения специальной функции по ее разложению в ряд с точностью, аргумент функции вводится с клавиатуры.

Интегральный косинус:

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

Формула интегрального косинуса в расписанном виде:

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

Программа и результаты вычислений изображены на рисунках 28, 29. UML-диаграмма деятельности программы – на рисунке 30.

```

advanced_2.py x
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  # Точность вычислений.
10 EPS = 1e-10
11
12 if __name__ == '__main__':
13     x = float(input("Введите x: "))
14     if x == 0:
15         print("Неверно задан X", file=sys.stderr)
16         exit(1)
17     a = x
18     S, n = a, 1
19     # Сумма членов ряда
20     while math.fabs(a) > EPS:
21         a *= -1*x**2*(2*n)/(2*n+2)**2
22         S += a
23         n += 1
24     # Значение функции
25     print(f"Ci({x}) = {EULER + math.log(math.fabs(x)) + S}")
26

```

Рисунок 28 – Листинг программы задания повышенной сложности 2

```

Run: advanced_2 x
  /usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/advanced_2.py
  Введите x: 1
  Ci(1.0) = 1.4648998231348875
  Process finished with exit code 0

Run: advanced_2 x
  /usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/advanced_2.py
  Введите x: 3
  Ci(3.0) = 3.0647389285299282

Run: advanced_2 x
  /usr/bin/python3.10 /home/user/git/lab2.2/2.2_programs/advanced_2.py
  Введите x: 0
  Неверно задан X
  Process finished with exit code 1

```

Рисунок 29 – Результат выполнения программы задания повышенной сложности 2

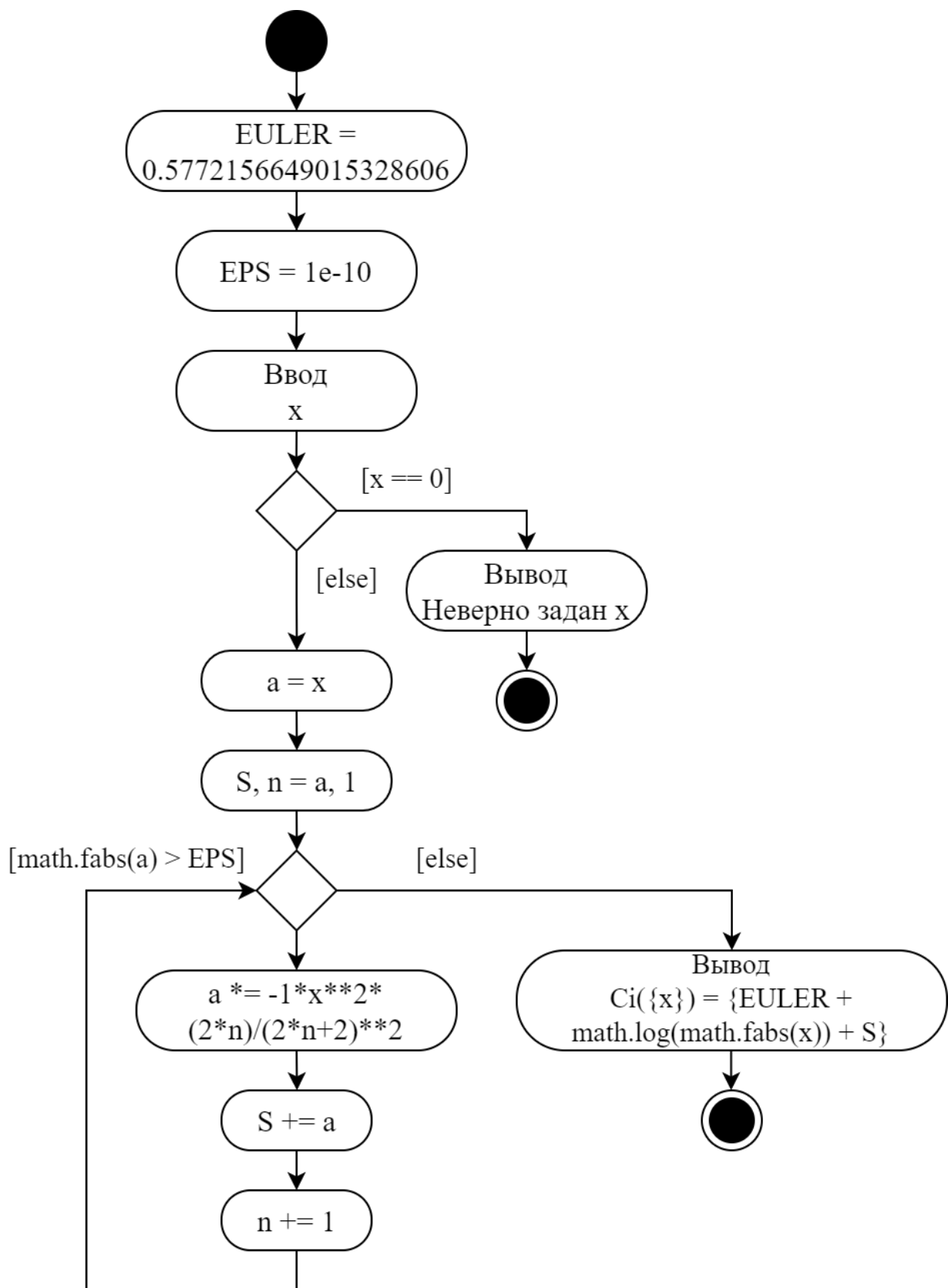


Рисунок 30 – UML-диаграмма деятельности программы задания повышенной сложности 2

Вывод: В результате выполнения работы были изучены циклические и разветвляющие структуры, а именно операторы языка Python if, while, for, break и continue и примеры их использования в программировании.

Контрольные вопросы:

1. Для чего нужны диаграммы деятельности UML?

Диаграммы деятельности – это один из пяти видов диаграмм, применяемых в UML для моделирования динамических аспектов поведения системы. Диаграмма деятельности – это, блок-схема, которая показывает, как поток управления переходит от одной деятельности к другой. UML-диаграммы помогут при моделировании архитектуры больших проектов, в которой можно собрать как крупные, так и более мелкие детали и нарисовать каркас (схему) приложения.

2. Что такое состояние действия и состояние деятельности?

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.

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.

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

В UML переход представляется простой линией со стрелкой. Точка ветвления представляется ромбом. В точку ветвления может входить ровно один переход, а выходить – два или более.

4. Какой алгоритм является алгоритмом разветвляющейся структуры?

Алгоритм разветвляющейся структуры – это алгоритм, в котором вычислительный процесс осуществляется либо по одной, либо по другой ветви, в зависимости от выполнения некоторого условия.

5. Чем отличается разветвляющийся алгоритм от линейного?

Линейный алгоритм выполняется последовательно независимо от чего-либо, а алгоритм ветвления выполняется определенные действия в зависимости от выполнения условия или условий.

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

Оператор ветвления «if» позволяет выполнить определенный набор инструкций в зависимости от некоторого условия. Возможны следующие варианты использования:

- 1) Конструкция «if»
- 2) Конструкция «if» - «else»
- 3) Конструкция «if» - «elif» - «else»

7. Какие операторы сравнения используются в Python?

<, >, <=, >=, ==, !=.

8. Что называется простым условием? Приведите примеры.

Логические выражения являются простыми, если в них выполняется только одна логическая операция. Например, $a > 15$ или $a \neq b$.

9. Что такое составное условие? Приведите примеры.

В составных условиях используется 2 и более логические операции. Например, $\text{if } x > 2 \text{ or } y < 3$.

10. Какие логические операторы допускаются при составлении сложных условий?

«and» и «or».

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

Да, может.

12. Какой алгоритм является алгоритмом циклической структуры?

Алгоритм циклической структуры – это алгоритм, в котором происходит многократное повторение одного и того же участка программы. Такие повторяемые участки вычислительного процесса называются циклами.

13. Типы циклов в языке Python.

Циклы «for» и «while».

14. Назовите назначение и способы применения функции range

Функция range возвращает неизменяемую последовательность чисел в виде объекта range. Параметры функции:

start - с какого числа начинается последовательность. По умолчанию - 0

stop - до какого числа продолжается последовательность чисел

Указанное число не включается в диапазон.

step - с каким шагом растут числа. По умолчанию 1

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

`range(15, 0, 2).`

16. Могут ли быть циклы вложенными?

Да, могут.

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

Пример бесконечного цикла: `a = 0`

`while a >= 0:`

`if a == 8:`

`break`

`a += 2`

`print("A")`

Оператор «break» предназначен для досрочного прерывания работы цикла «while».

18. Для чего нужен оператор break?

Оператор «break» предназначен для досрочного прерывания работы цикла «while».

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

Оператор «continue» запускает цикл заново, при этом код, расположенный после данного оператора, не выполняется.

20. Для чего нужны стандартные потоки *stdout* и *stderr*?

В операционной системе по умолчанию присутствуют стандартные потоки 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*. 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. Как в Python организовать вывод в стандартный поток *stderr*?

Для того, чтобы использовать поток *stderr* необходимо передать его в параметре `file` функции `print`.

22. Каково назначение функции `exit`?

В Python завершить программу и передать операционной системе заданный код возврата можно посредством функции `exit`.