

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 №2.2
дисциплины «Основы программной инженерии»

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

(подпись)

Руководитель практики:
Богданов С.С., ассистент кафедры
инфокоммуникаций

(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.

Ход выполнения работы:

1. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.:

The screenshot shows the GitHub 'Create repository' page. At the top, there's a 'Repository template' section with a 'No template' dropdown. Below it, a message says 'Start your repository with a template repository's contents.' The 'Owner' is set to 'daxstrong' and the 'Repository name' is 'lr2_2', with a green checkmark indicating it's available. A suggestion for 'musical-fiesta' is shown. The 'Description' field is empty. Under 'Visibility', 'Public' is selected. The 'Initialize this repository with:' section has 'Add a README file' checked. The '.gitignore' section shows the 'Python' template selected. The 'Choose a license' section has 'MIT License' selected. At the bottom, it says 'This will set main as the default branch.' and a note: 'You are creating a public repository in your personal account.' A green 'Create repository' button is at the bottom right.

Рисунок 1 – Создание репозитория с заданными настройками

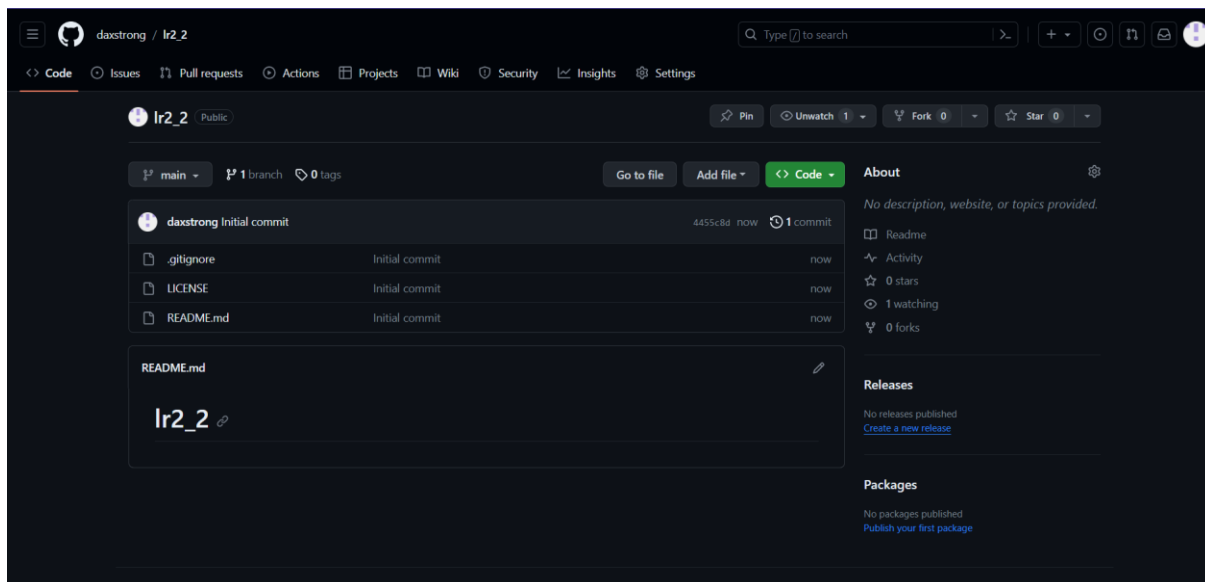


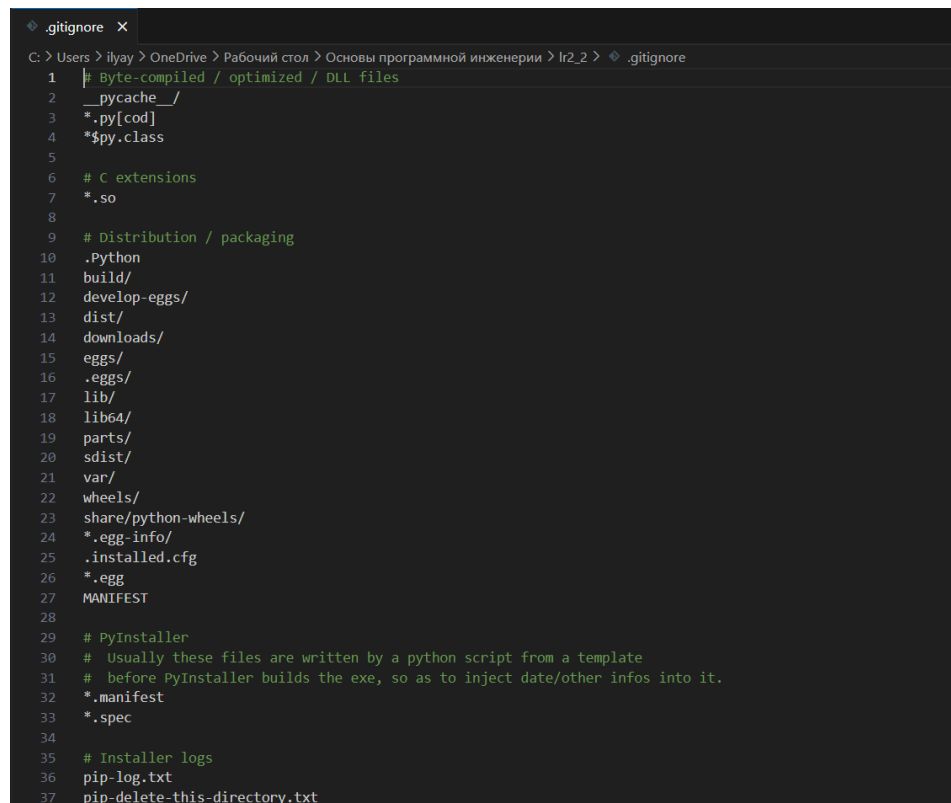
Рисунок 2 – Созданный репозиторий

```
ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/Основы программной инженерии
$ git clone https://github.com/daxstrong/lr2_2.git
Cloning into 'lr2_2'...
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 3 – Клонирование репозитория

```
ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/Основы программной инженерии/lr2_2 (main)
$ git checkout -b develop
Switched to a new branch 'develop'
```

Рисунок 4 – Создание ветки develop



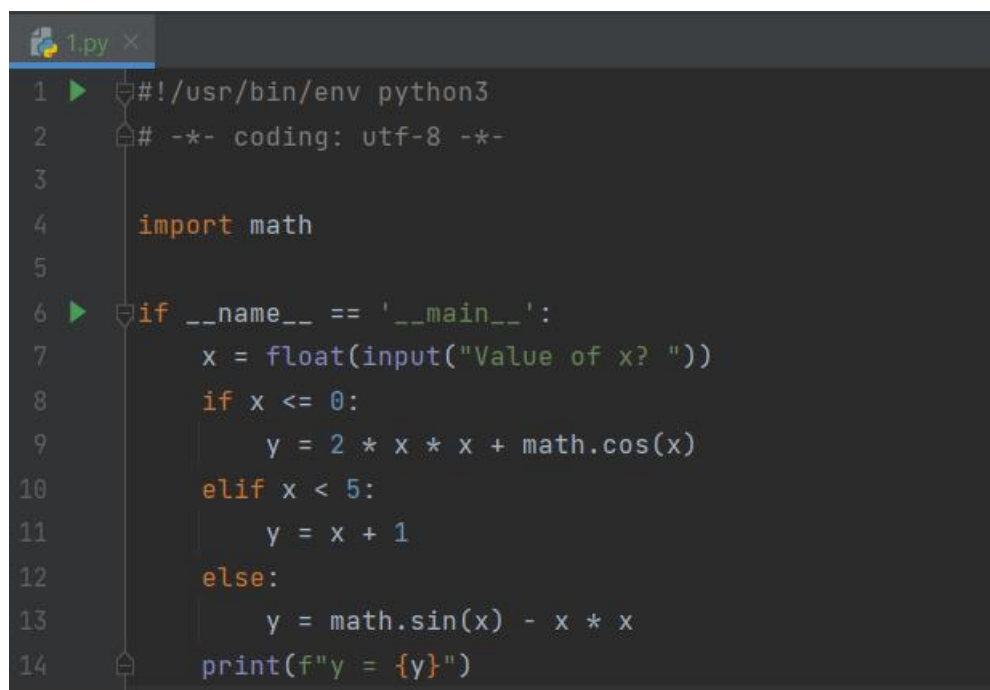
```

1  # Byte-compiled / optimized / DLL files
2  __pycache__/
3  *.py[cod]
4  *$py.class
5
6  # C extensions
7  *.so
8
9  # Distribution / packaging
10 .Python
11 build/
12 develop-eggs/
13 dist/
14 downloads/
15 eggs/
16 .eggs/
17 lib/
18 lib64/
19 parts/
20 sdist/
21 var/
22 wheels/
23 share/python-wheels/
24 *.egg-info/
25 .installed.cfg
26 *.egg
27 MANIFEST
28
29 # PyInstaller
30 # Usually these files are written by a python script from a template
31 # before PyInstaller builds the exe, so as to inject date/other infos into it.
32 *.manifest
33 *.spec
34
35 # Installer logs
36 pip-log.txt
37 pip-delete-this-directory.txt

```

Рисунок 5 – содержимое файла .gitignore

2. Проработать примеры лабораторной работы, оформляя код согласно PEP-8. Построить UML-диаграммы деятельности для 4 и 5 заданий:

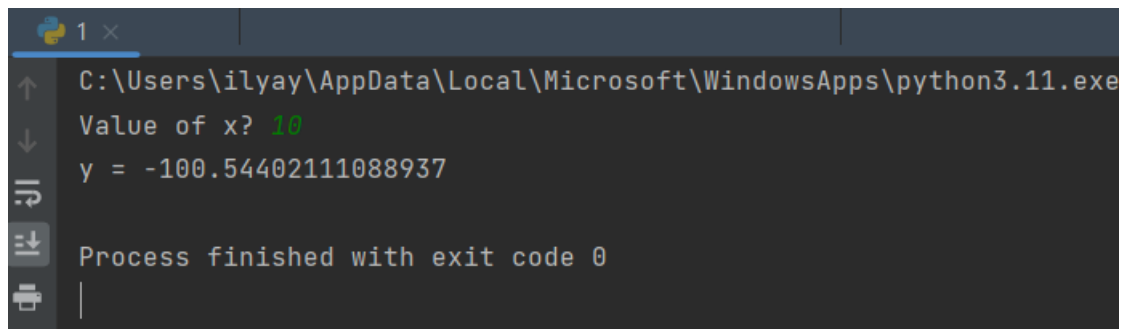


```

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

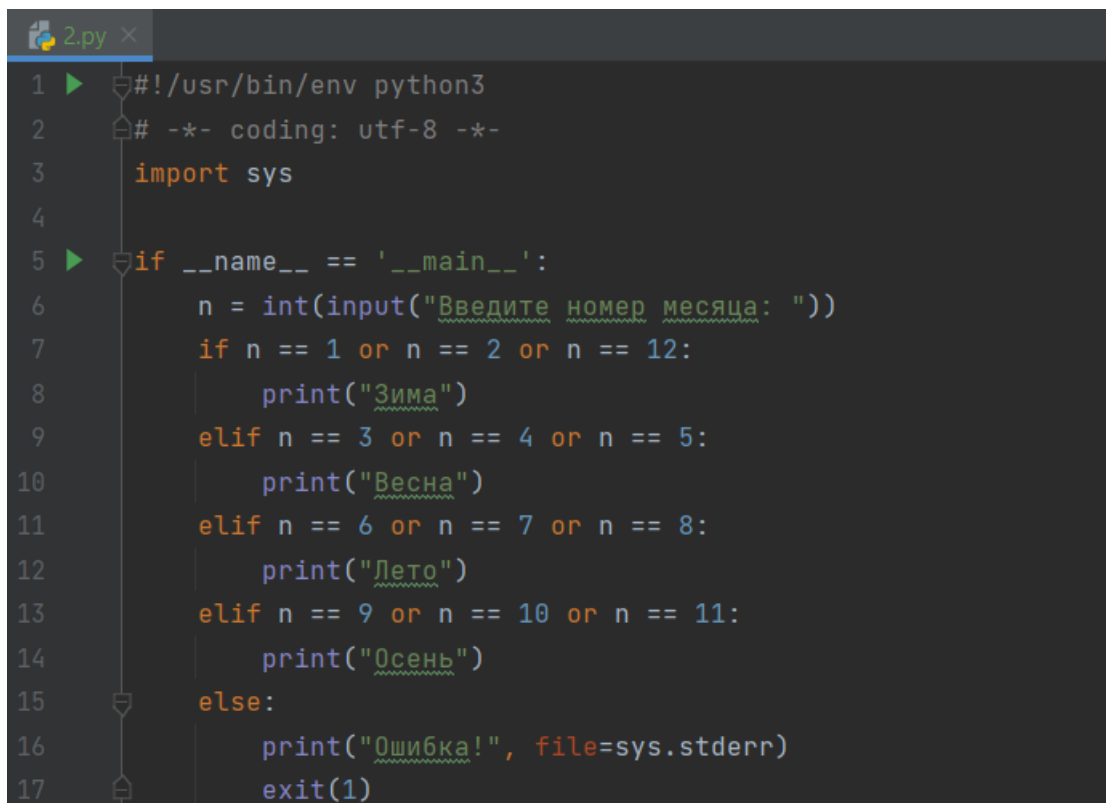
```

Рисунок 6 – Нахождение значения функции (задание №1)



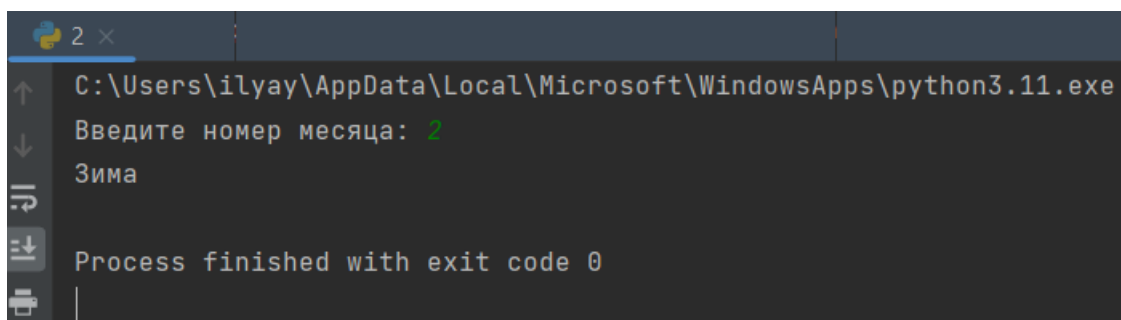
```
1 x
C:\Users\ilyay\AppData\Local\Microsoft\WindowsApps\python3.11.exe
Value of x? 10
y = -100.54402111088937
Process finished with exit code 0
```

Рисунок 7 – Вывод программы (задание №1)



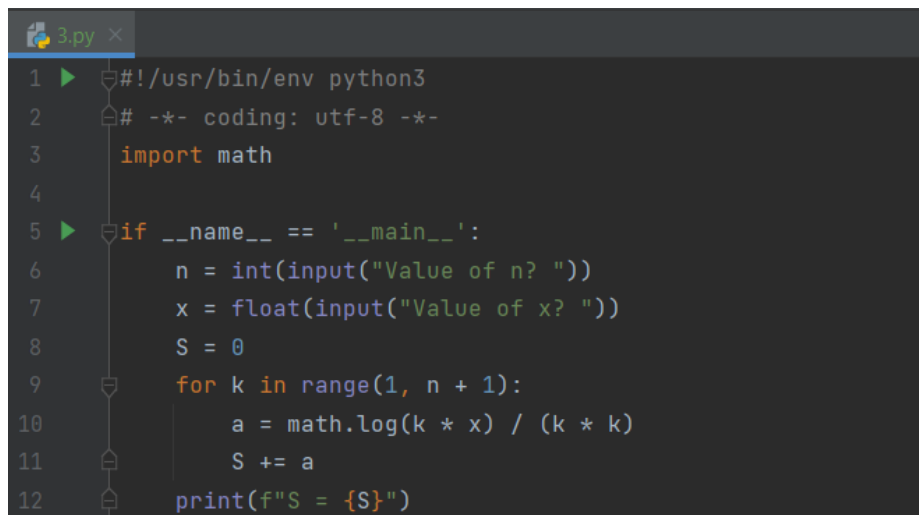
```
2.py x
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 import sys
4
5 if __name__ == '__main__':
6     n = int(input("Введите номер месяца: "))
7     if n == 1 or n == 2 or n == 12:
8         print("Зима")
9     elif n == 3 or n == 4 or n == 5:
10        print("Весна")
11    elif n == 6 or n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

Рисунок 8 – Нахождение времени года по номеру месяца (задание №2)



```
2 x
C:\Users\ilyay\AppData\Local\Microsoft\WindowsApps\python3.11.exe
Введите номер месяца: 2
Зима
Process finished with exit code 0
```

Рисунок 9 – Вывод времени года по номеру месяца (задание №2)

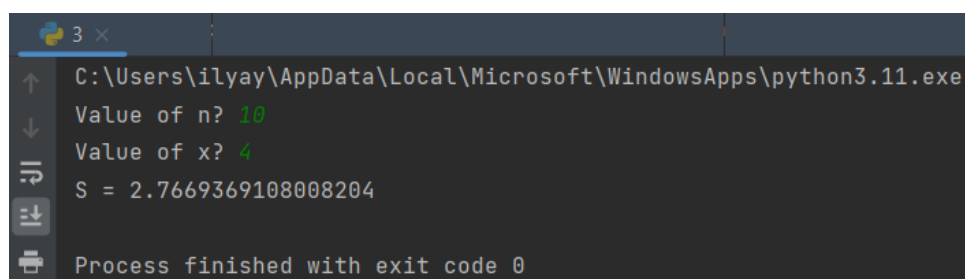


```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  import math
4
5  if __name__ == '__main__':
6      n = int(input("Value of n? "))
7      x = float(input("Value of x? "))
8      S = 0
9      for k in range(1, n + 1):
10         a = math.log(k * x) / (k * x)
11         S += a
12     print(f"S = {S}")

```

Рисунок 10 – Вычисление конечной суммы (задание №3)

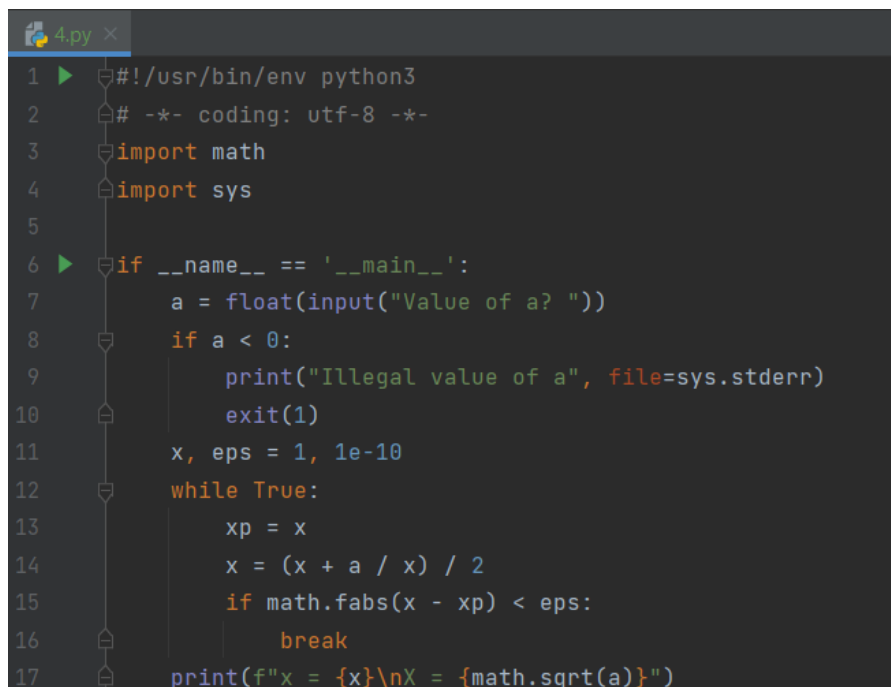


```

C:\Users\ilyay\AppData\Local\Microsoft\WindowsApps\python3.11.exe
Value of n? 10
Value of x? 4
S = 2.7669369108008204
Process finished with exit code 0

```

Рисунок 11 – Вывод конечной суммы (задание №3)



```

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

```

Рисунок 12 – Нахождение квадратного корня числа и его сравнение с результатом метода sqrt() стандартной библиотеки Python (задание №4)

```
4 x
C:\Users\ilyay\AppData\Local\Microsoft\WindowsApps\python3.11.exe
Value of a? 15
x = 3.872983346207417
X = 3.872983346207417
Process finished with exit code 0
```

Рисунок 13 – Вывод программы (задание №4)

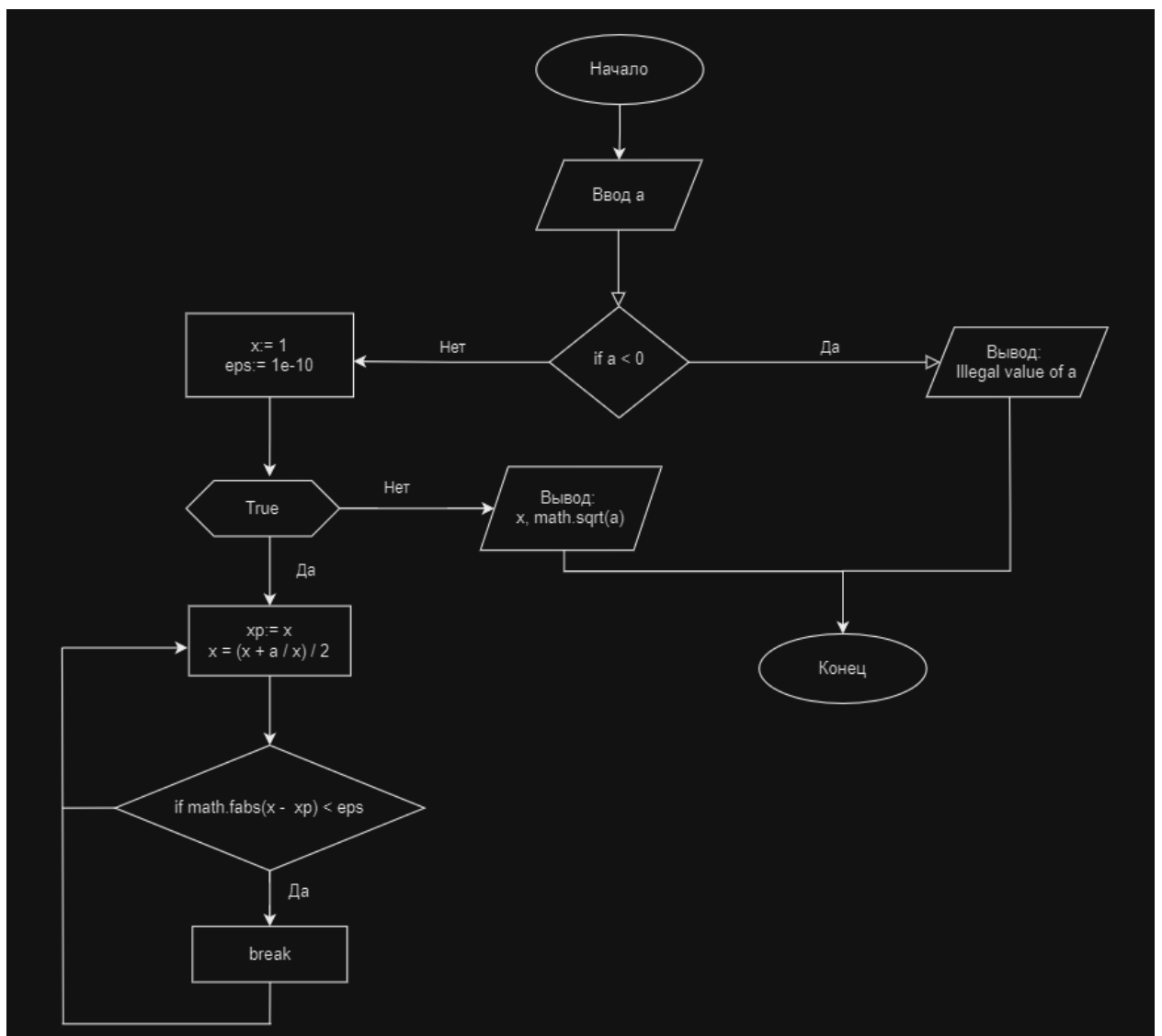


Рисунок 14 – Диаграмма для программы задания №4

```

5.py x
1 ▶ #!/usr/bin/env python3
2   #- coding: utf-8 -*-
3
4   import math
5   import sys
6
7
8   # Постоянная Эйлера.
9   EULER = 0.5772156649015329
10  # Точность вычислений.
11  EPS = 1e-10
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}")

```

Рисунок 15 – Вычисление значения специальной (интегральной показательной) функции (задание №5)

```

5 x
C:\Users\ilyay\AppData\Local\Microsoft\WindowsApps\python3.11.exe
Value of x? 4
Ei(4.0) = 19.63087447005282
Process finished with exit code 0

```

Рисунок 16 – Вывод программы (задание №5)

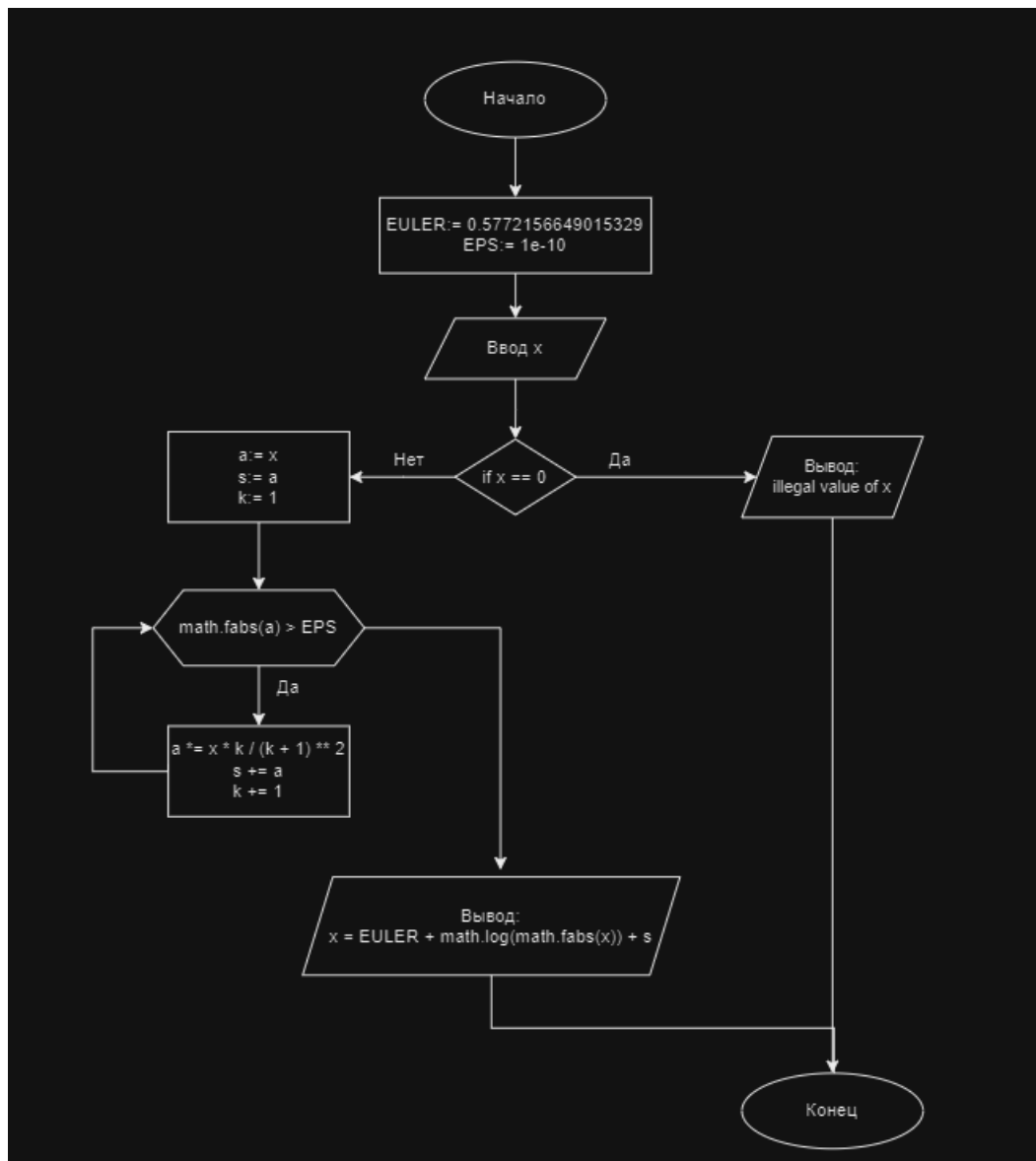


Рисунок 17 – Диаграмма для программы задания №5

3. Выполним индивидуальные задания и построим UML-диаграммы для них:

```
individual1.py x
1  ▶  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  ▶  if __name__ == '__main__':
5      wallpaper_price = float(input("Введите цену рулона обоев: "))
6      paint_can_price = float(input("Введите цену банки краски: "))
7
8      wallpaper_cost = 8 * wallpaper_price
9      paint_cost = 2 * paint_can_price
10
11     total_cost = wallpaper_cost + paint_cost
12
13     discount = 0
14     if 200 <= total_cost < 500:
15         discount = 0.03
16     elif 500 <= total_cost < 800:
17         discount = 0.05
18     elif 800 <= total_cost < 1000:
19         discount = 0.07
20     elif total_cost >= 1000:
21         discount = 0.09
22
23     final_amount = total_cost * (1 - discount)
24
25     print(f"Покупатель заплатил {final_amount} рублей.")
```

Рисунок 18 – Нахождение стоимости покупки с учетом скидки (задание №1)

```
individual1 x |
C:\Users\ilyay\AppData\Local\Microsoft\WindowsApps\python3.11.exe
Введите цену рулона обоев: 100
Введите цену банки краски: 50
Покупатель заплатил 837.0 рублей.
```

Рисунок 19 – Вывод программы (задание №1)

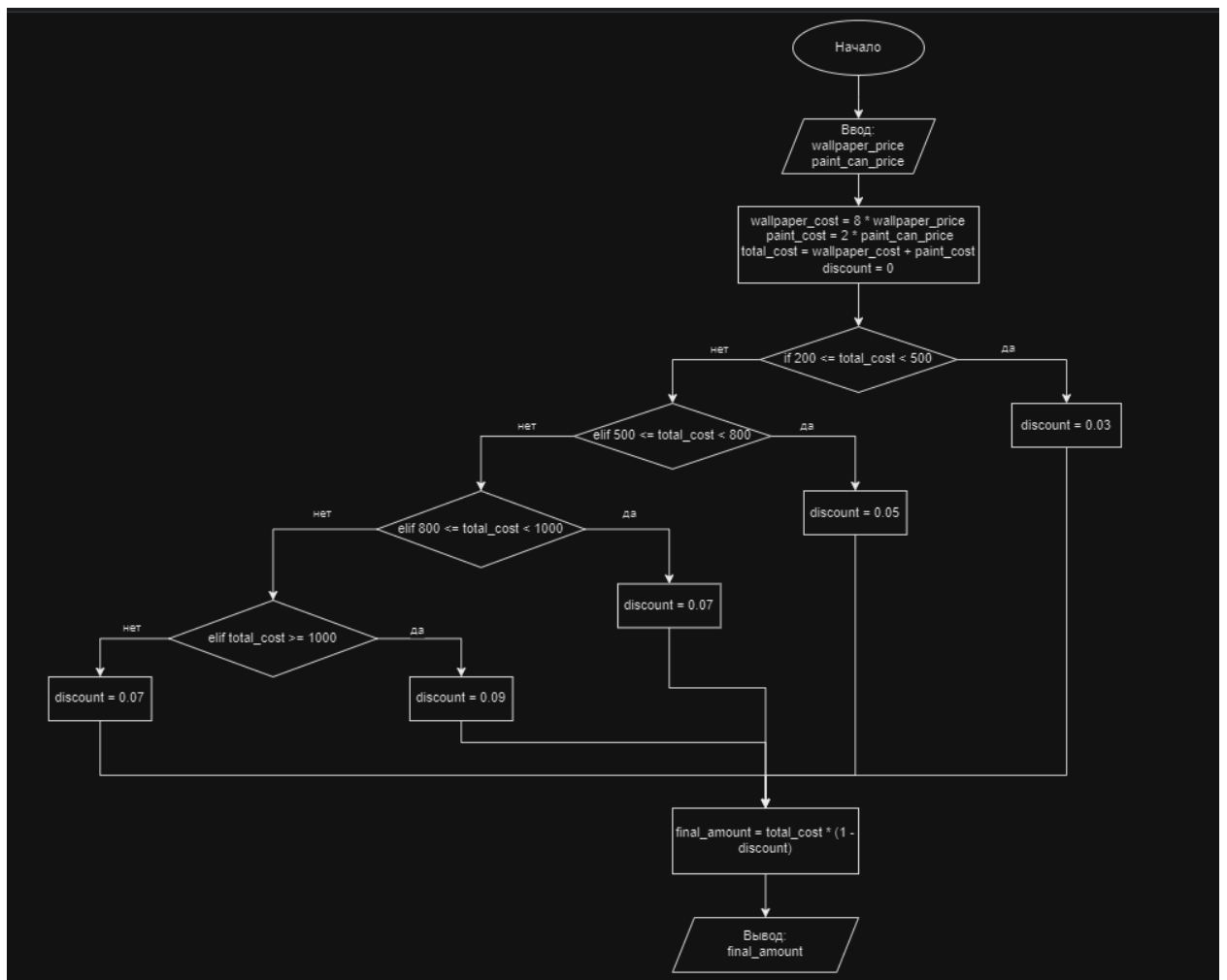


Рисунок 20 – Диаграмма программы индивидуального задания №1

```
individual2.py x
1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4      import math
5
6
7  ▶  if __name__ == '__main__':
8      x1 = float(input("Введите x-координату центра первой окружности: "))
9      y1 = float(input("Введите y-координату центра первой окружности: "))
10     r1 = float(input("Введите радиус первой окружности: "))
11
12     x2 = float(input("Введите x-координату центра второй окружности: "))
13     y2 = float(input("Введите y-координату центра второй окружности: "))
14     r2 = float(input("Введите радиус второй окружности: "))
15
16     distance = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
17
18     if distance > r1 + r2:
19         print("Окружности не пересекаются")
20     elif distance < abs(r1 - r2):
21         print("Одна окружность внутри другой, бесконечно много пересечений")
22     elif distance == 0 and r1 == r2:
23         print("Окружности совпадают, бесконечно много пересечений")
24     else:
25         alpha = math.acos((r1 ** 2 + distance ** 2 - r2 ** 2) / (2 * r1 * distance))
26         beta = math.atan2((y2 - y1), (x2 - x1))
27
28         intersection_point1 = (x1 + r1 * math.cos(beta + alpha), y1 + r1 * math.sin(beta + alpha))
29         intersection_point2 = (x1 + r1 * math.cos(beta - alpha), y1 + r1 * math.sin(beta - alpha))
30
31         if intersection_point1 == intersection_point2:
32             print("Окружности касаются в одной точке")
33         else:
34             print(f"Окружности пересекаются в двух точках: {intersection_point1}, {intersection_point2}")
35
```

Рисунок 21 – Проверка на пересечение двух окружностей (задание №2)

```
individual2 x
C:\Users\ilyay\AppData\Local\Microsoft\WindowsApps\python3.11.exe
Введите x-координату центра первой окружности: 12
Введите y-координату центра первой окружности: 5
Введите радиус первой окружности: 4
Введите x-координату центра второй окружности: 24
Введите y-координату центра второй окружности: 6
Введите радиус второй окружности: 2
Окружности не пересекаются
```

Рисунок 22 – Вывод программы (задание №2)

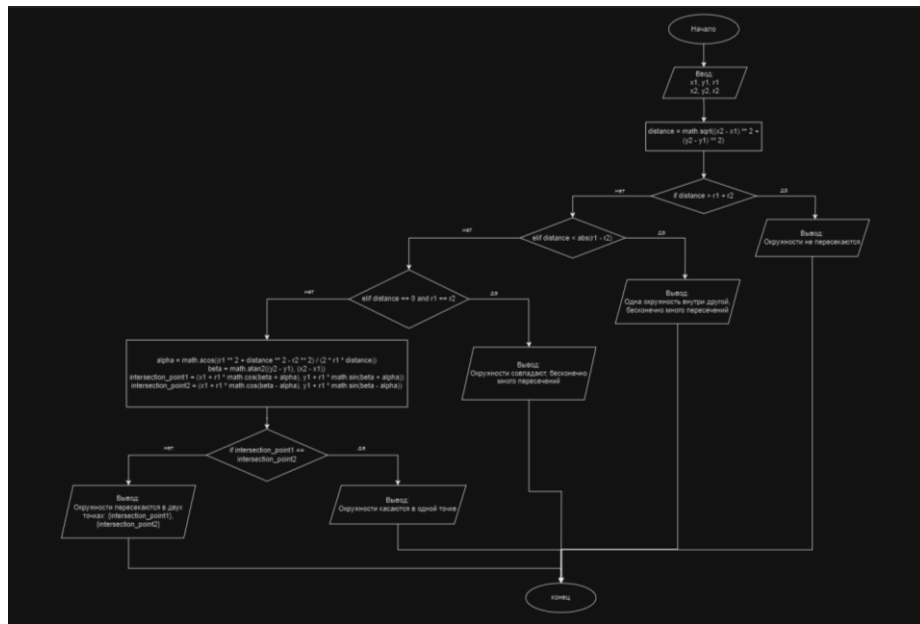


Рисунок 23 – Диаграмма программы индивидуального задания №2

```

individual3.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  if __name__ == '__main__':
5      payment = int(input("Введите сумму, которую отдаёт покупатель: "))
6
7      available_notes = [500, 100, 10, 5, 2, 1]
8
9      print("Покупатель отдаст следующее количество купюр разного достоинства:")
10
11     for note in available_notes:
12         if payment >= note:
13             count = payment // note
14             payment %= note
15             print(f"{note} р.: {count} шт.")
16

```

Рисунок 24 – Определение количества купюр каждого номинала, которые покупатель отдаст, начиная с самой большой (задание №3)

```
individual3 x |
C:\Users\ilyay\AppData\Local\Microsoft\WindowsApps\python3.11.exe
Введите сумму, которую отдаёт покупатель: 2999
Покупатель отдаст следующее количество купюр разного достоинства:
500 р.: 5 шт.
100 р.: 4 шт.
10 р.: 9 шт.
5 р.: 1 шт.
2 р.: 2 шт.
```

Рисунок 25 – Вывод программы (задание №3)

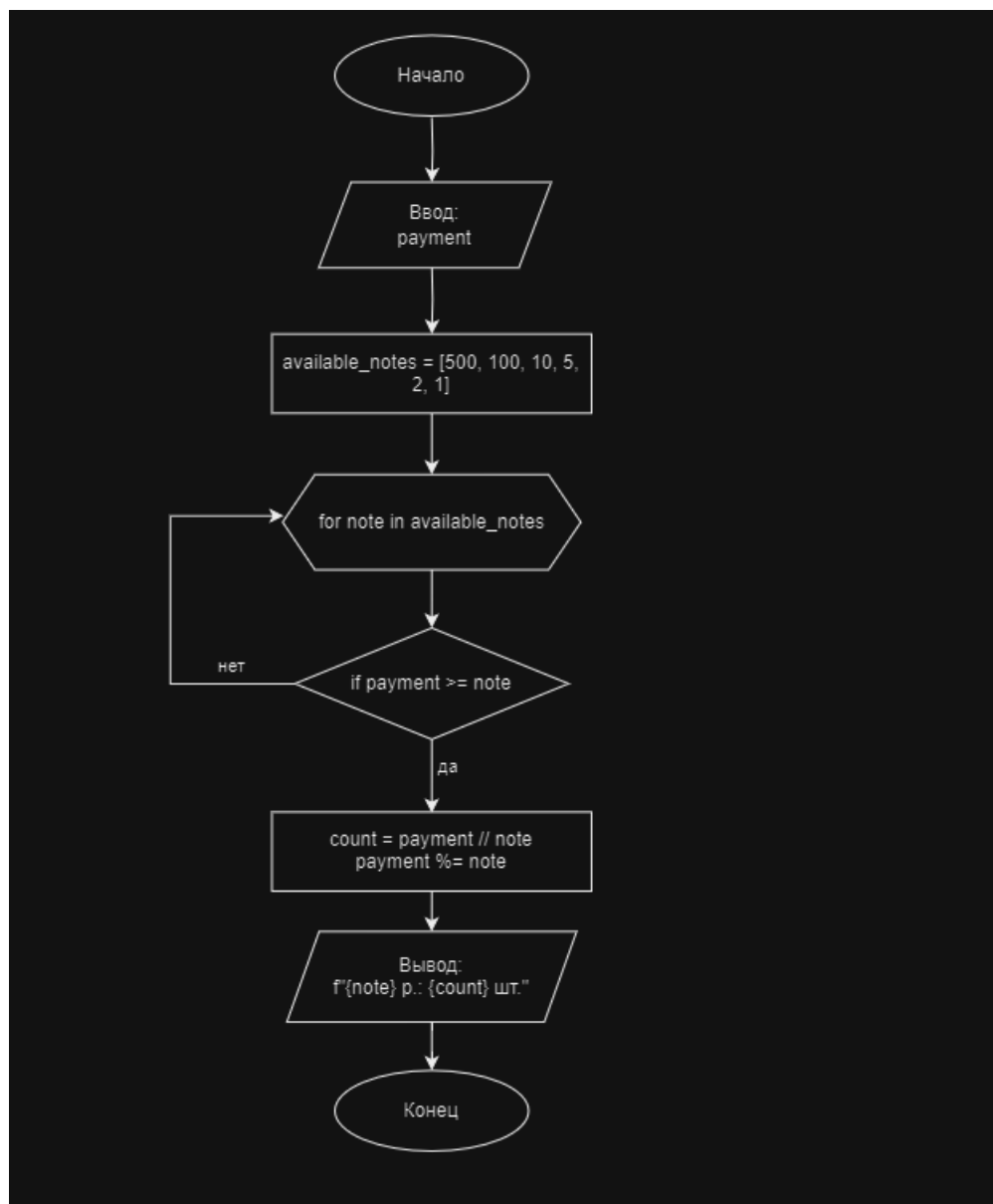


Рисунок 26 – Диаграмма программы индивидуального задания №3

4. Зафиксируем проделанные изменения, сольем ветки и отправим на удаленный репозиторий:

```
ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/Основы программной инженерии/lr2_2 (main)
$ git merge develop
Updating 4455c8d..57fdde2
Fast-forward
 .idea/.gitignore          | 8 ++++++
 .idea/.name               | 1 +
 .idea/inspectionProfiles/profiles_settings.xml | 6 +++++
 .idea/lr2_2.iml          | 8 ++++++
 .idea/misc.xml            | 4 +++
 .idea/modules.xml         | 8 ++++++
 .idea/vcs.xml             | 6 +++++
 1.py                     | 14 ++++++++
 2.py                     | 17 ++++++++
 3.py                     | 12 ++++++
 4.py                     | 17 ++++++++
 5.py                     | 25 ++++++++
 individual1.py           | 25 ++++++++
 individual2.py           | 34 ++++++++
 individual3.py           | 15 ++++++
15 files changed, 200 insertions(+)
create mode 100644 .idea/.gitignore
create mode 100644 .idea/.name
create mode 100644 .idea/inspectionProfiles/profiles_settings.xml
create mode 100644 .idea/lr2_2.iml
create mode 100644 .idea/misc.xml
create mode 100644 .idea/modules.xml
create mode 100644 .idea/vcs.xml
create mode 100644 1.py
create mode 100644 2.py
create mode 100644 3.py
create mode 100644 4.py
create mode 100644 5.py
create mode 100644 individual1.py
create mode 100644 individual2.py
create mode 100644 individual3.py
```

Рисунок 27 – Слияние веток main и develop

```
ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/Основы программной инженерии/lr2_2 (main)
$ git push origin main
Enumerating objects: 24, done.
Counting objects: 100% (24/24), done.
Delta compression using up to 12 threads
Compressing objects: 100% (20/20), done.
Writing objects: 100% (23/23), 4.72 KiB | 4.72 MiB/s, done.
Total 23 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/daxstrong/lr2_2.git
 4455c8d..b1d7fc7  main -> main
```

Рисунок 25 – Отправка изменений на удаленный репозиторий

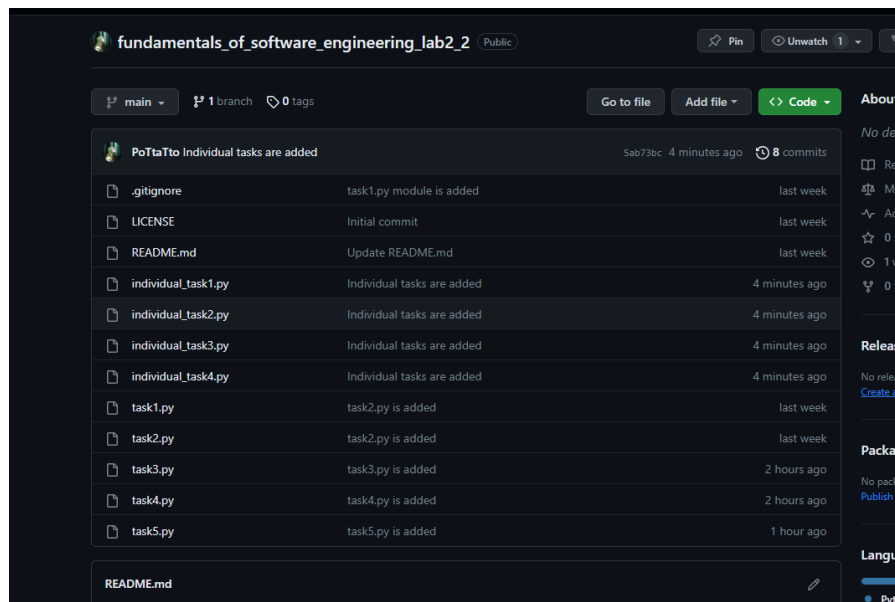


Рисунок 28 – Изменения удаленного репозитория

Ответы на контрольные вопросы:

1. Для чего нужны диаграммы деятельности UML?

Диаграмма деятельности (Activity diagram) показывает поток переходов от одной деятельности к другой. Деятельность (Activity) — это продолжающийся во времени неатомарный шаг вычислений в автомате. Деятельности в конечном счете приводят к выполнению некоего действия (Action), составленного из выполняемых атомарных вычислений, каждое из которых либо изменяет состояние системы, либо возвращает какое-то значение. Действие может заключаться в вызове другой операции, послыке сигнала, создании или уничтожении объекта либо в простом вычислении - скажем, значения выражения. Графически диаграмма деятельности представляется в виде графа, имеющего вершины и ребра.

2. Что такое 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 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. Какие нотации существуют для обозначения переходов и ветвлений в диаграммах деятельности?

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.

4. Какой алгоритм является алгоритмом 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. Чем отличается разветвляющийся алгоритм от линейного?

Разветвляющийся алгоритм отличается от линейного тем, что он позволяет программе принимать решения на основе различных условий.

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

Оператор ветвления `if` позволяет выполнить определенный набор инструкций в зависимости от некоторого условия. Существуют: `if`, `if-else`, `if-elif-else`.

7. Какие операторы сравнения используются в Python?

`>`, `<`, `<=`, `>=`, `!=`, `==`

8. Что называется простым условием? Приведите примеры:

Простые условия – это те, в которых выполняется только одна логическая операция, например: `var >= 1023`, `var == 0`, `var != 3` и т. д.

8. Что такое составное условие? Приведите примеры:

Может понадобиться получить ответа `True` или `False` в зависимости от результата выполнения двух и более простых выражений. Для этого используется составное условие из операторов `or`, `and` и `not`. Например: `var < 100 and var > 0`, `not var < 10 or var == 100` и т. д.

10. Какие логические операторы допускаются при составлении сложных условий?

В Python допускаются три основных логических оператора для составления сложных условий: `and`, `or` и `not`.

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

Да, может.

12. Какой алгоритм является алгоритмом 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:

Цикл со счетчиком (`for`), цикл с предусловием (`while`), цикл `for-each` (`for in`). Цикла с постусловием нет, но его можно сделать, используя другие алгоритмические конструкции и операторы.

14. Назовите назначение и способы применения функции `range`:

Функция `range` возвращает неизменяемую последовательность чисел в виде объекта `range`. 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. Обычно применяется для цикла с счетчиком.

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

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

16. Могут ли быть циклы вложенными?

Могут.

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

Например, можно воспользоваться конструкцией `while True`. Чтобы выйти необходимо воспользоваться оператором `break`.

18. Для чего нужен оператор `break`?

Оператор `break` предназначен для досрочного прерывания работы цикла.

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 стандартных потока вывода на консоль: буферизованный поток `stdout` для вывода данных и информационных сообщений, а также небуферизованный поток `stderr` для вывода сообщений об ошибках.

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

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.

22. Каково назначение функции `exit`?

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