

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное учреждение
высшего образования**

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития

Кафедра информационных систем и технологий

Отчет по лабораторной работе №5.

Дисциплина: «Основы программной инженерии»

Выполнил:

Студент группы ПИЖ-б-о-22-1,
направление подготовки: 09.03.04

«Программная инженерия»

ФИО: Джараян Арег Александрович

Проверил:

Воронкин Р. А.

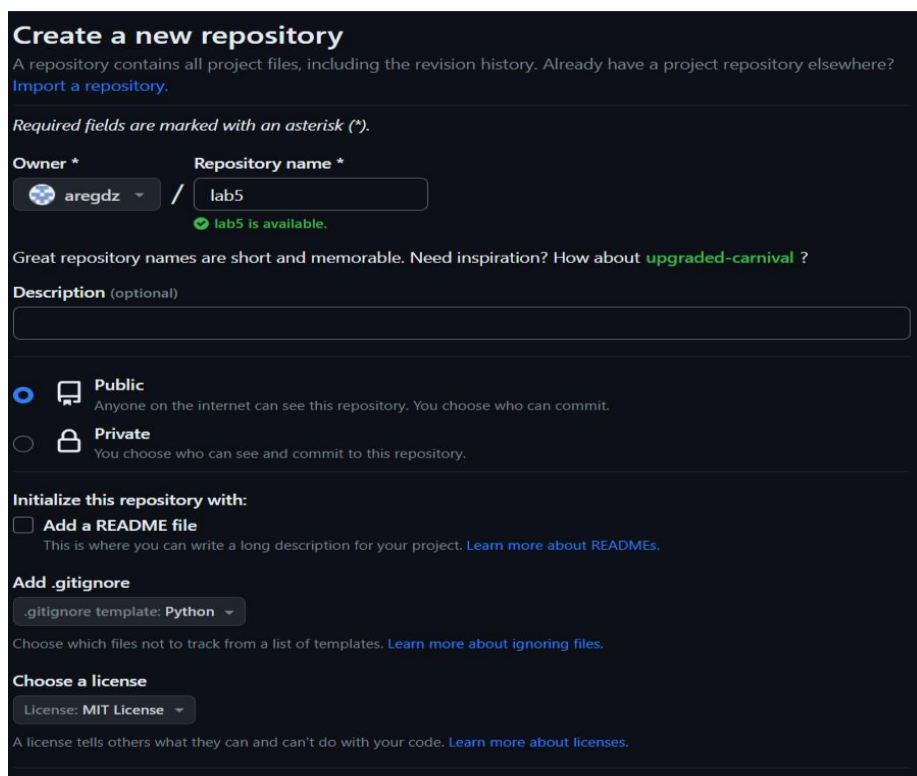
Ставрополь 2022

Тема: Лабораторная работа 2.1 Основы языка Python

Цель работы: исследование процесса установки и базовых возможностей языка Python версии 3.x.

Выполнение работы:

1. Изучил теоретический материал работы.
2. Создал репозиторий на git.hub.



The screenshot shows the GitHub 'Create a new repository' page. At the top, it says 'Create a new repository' and provides a brief explanation of what a repository is. Below this, there's a section for 'Required fields are marked with an asterisk (*)'. The 'Owner' field is set to 'aregdz' and the 'Repository name' field is 'lab5', with a green checkmark indicating 'lab5 is available'. There's a 'Description (optional)' text area below. The 'Visibility' section has 'Public' selected by default. The 'Initialize this repository with:' section has 'Add a README file' checked. The '.gitignore' section has 'Python' selected as the template. The 'Choose a license' section has 'MIT License' selected. At the bottom, there's a link to 'Learn more about licenses'.

Рисунок 1 – создание репозитория

3. Клонировал репозиторий.

MINGW64:/c/Users/aregd/OneDrive/Рабочий стол/git 5

```
aregd@DESKTOP-5KV9QA9 MINGW64 ~
$ cd "C:\Users\aregd\OneDrive\Рабочий стол\git 5"

aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/git 5
$ git clone https://github.com/aregdz/lab5.git
Cloning into 'lab5'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.

aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/git 5
$ |
```

Рисунок 2 – клонирование репозитория 4.

Дополнить файл gitignore необходимыми правилами.

```
.gitignore – Блокнот
Файл Правка Формат Вид Справка
# Created by .ignore support plugin (hsz.mobi)
### Python template
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
env/
build/
develop-eggs/
dist/
downloads/
```

Рисунок 3 - .gitignore для IDE PyCharm

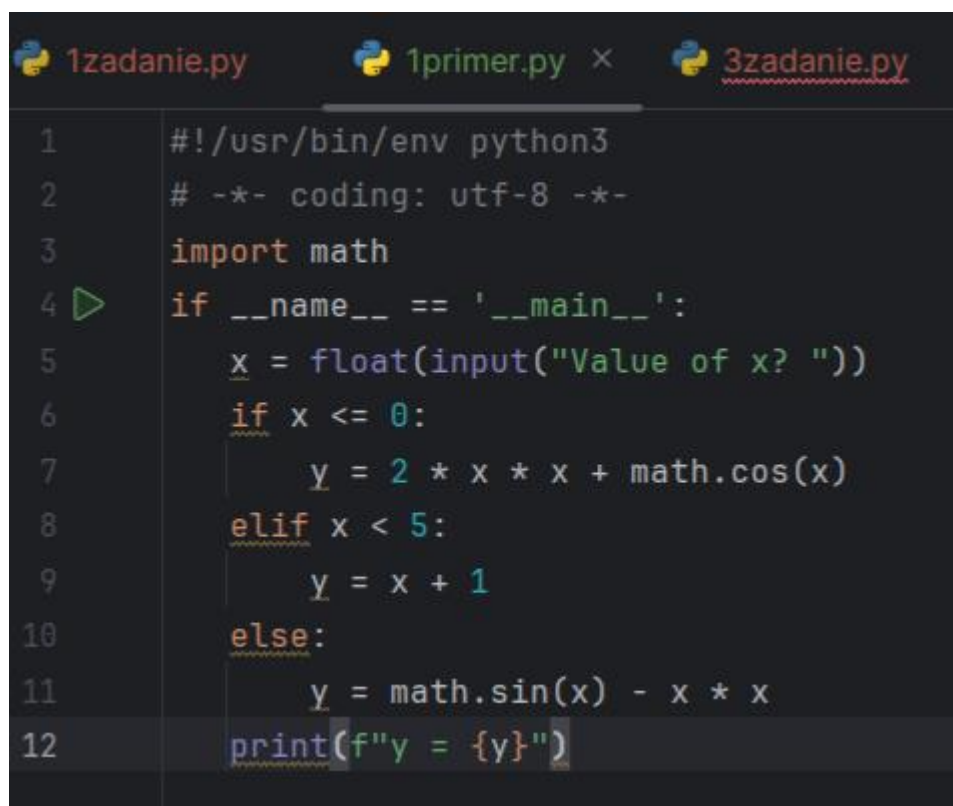
5. Организовать свой репозиторий в соответствии с моделью ветвления git-flow.

```
aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/git 5/lab5 (main)
$ git checkout -b develop
Switched to a new branch 'develop'

aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/git 5/lab5 (develop)
$ |
```

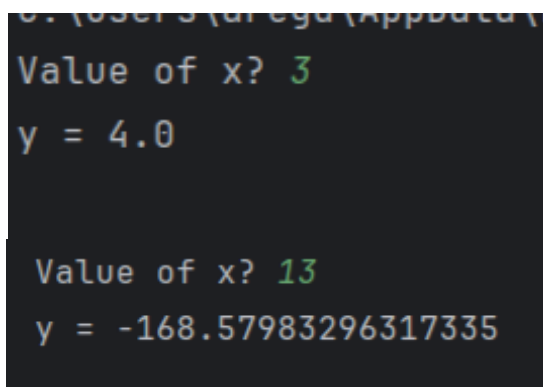
Рисунок 4 – создание ветки develop

6. Проработать примеры лабораторной работы. Создать для каждого примера отдельный модуль языка Python. Зафиксировать изменения в репозитории. Привести в отчете скриншоты результатов выполнения каждой из программ примеров при различных исходных данных вводимых с клавиатуры.



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

Рисунок 5 – пример 1



```
Value of x? 3
y = 4.0

Value of x? 13
y = -168.57983296317335
```

Рисунок 6 – примеры выполнения для примера 1

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

Рисунок 7 – пример 2

```
C:\Users\aregd\AppData\Local\Progra
Введите номер месяца: 2
Зима

Process finished with exit code 0
C:\Users\aregd\AppData\Local\Progra
Введите номер месяца: 4
Весна

Process finished with exit code 0
```

Рисунок 8 – примеры выполнения для 2 примера

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

Рисунок 9 – пример 3

```
C:\Users\aregd\AppData\Local\Programs\Python\Python39\python.exe 1zadanie.py
Value of n? 2
Value of x? 5
S = 2.1850841856826118

Process finished with exit code 0

C:\Users\aregd\AppData\Local\Programs\Python\Python39\python.exe 1zadanie.py
Value of n? 6
Value of x? 10
S = 3.930196661061532

Process finished with exit code 0
```

Рисунок 10 – примеры выполнения для примера 3

```

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

```

Рисунок 10 – пример 4

```

C:\Users\aregd\AppData\Local\Programs\Python\Python38-32\python.exe
Value of a? 2
x = 1.414213562373095
X = 1.4142135623730951

Process finished with exit code 0

C:\Users\aregd\AppData\Local\Programs\Python\Python38-32\python.exe
Value of a? 8
x = 2.82842712474619
X = 2.8284271247461903

view is read-only
Process finished with exit code 0

```

Рисунок 11 – примеры выполнения для примера 4

```
1zadanie.py 1primer.py 2primer.py 3primer.py

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  import math
4  import sys
5  # Постоянная Эйлера.
6  EULER = 0.5772156649015329
7  # Точность вычислений.
8  EPS = 1e-10
9  if __name__ == '__main__':
10     x = float(input("Value of x? "))
11     if x == 0:
12         print("Illegal value of x", file=sys.stderr)
13         exit(1)
14     a = x
15     S, k = a, 1
16     # Найти сумму членов ряда.
17     while math.fabs(a) > EPS:
18         a *= x * k / (k + 1) ** 2
19         S += a
20         k += 1
21     # Вывести значение функции.
22     print(f"Ei({x}) = {EULER + math.log(math.fabs(x)) + S}")
```

Рисунок 12 – пример 5

```
C:\Users\aregd\AppData\Local\Programs
Value of x? 12
Ei(12.0) = 14959.532666397517

Process finished with exit code 0
C:\Users\aregd\AppData\Local\Progra
Value of x? 11
Ei(11.0) = 6071.406374098595

Process finished with exit code 0
```

Рисунок 13 – примеры выполнения для примера 5

7. Для примеров 4 и 5 построить UML-диаграмму деятельности. Для построения диаграмм деятельности использовать веб-сервис Google <https://www.diagrams.net/>.

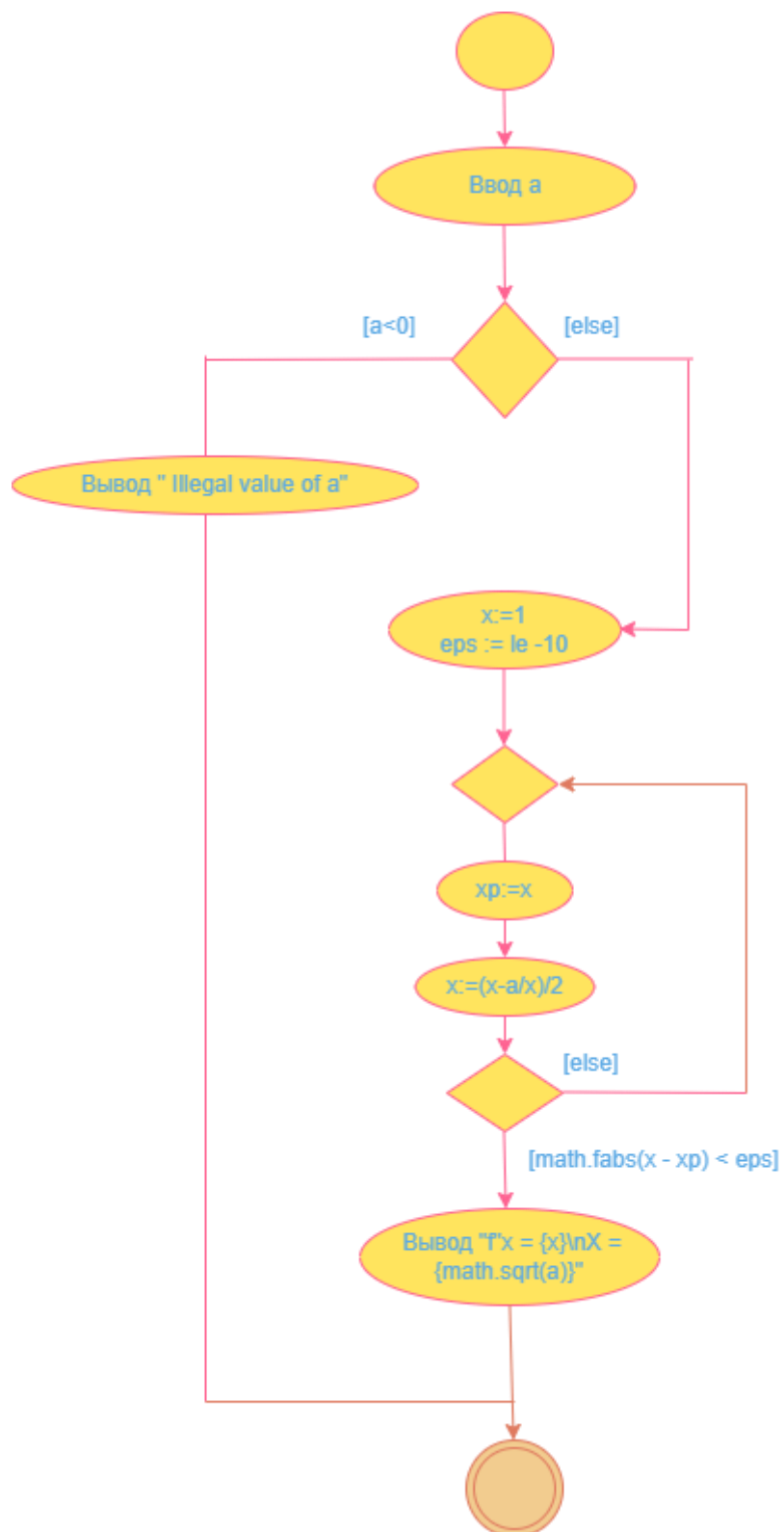


Рисунок 14 - UML-диаграмму деятельности для примера 4

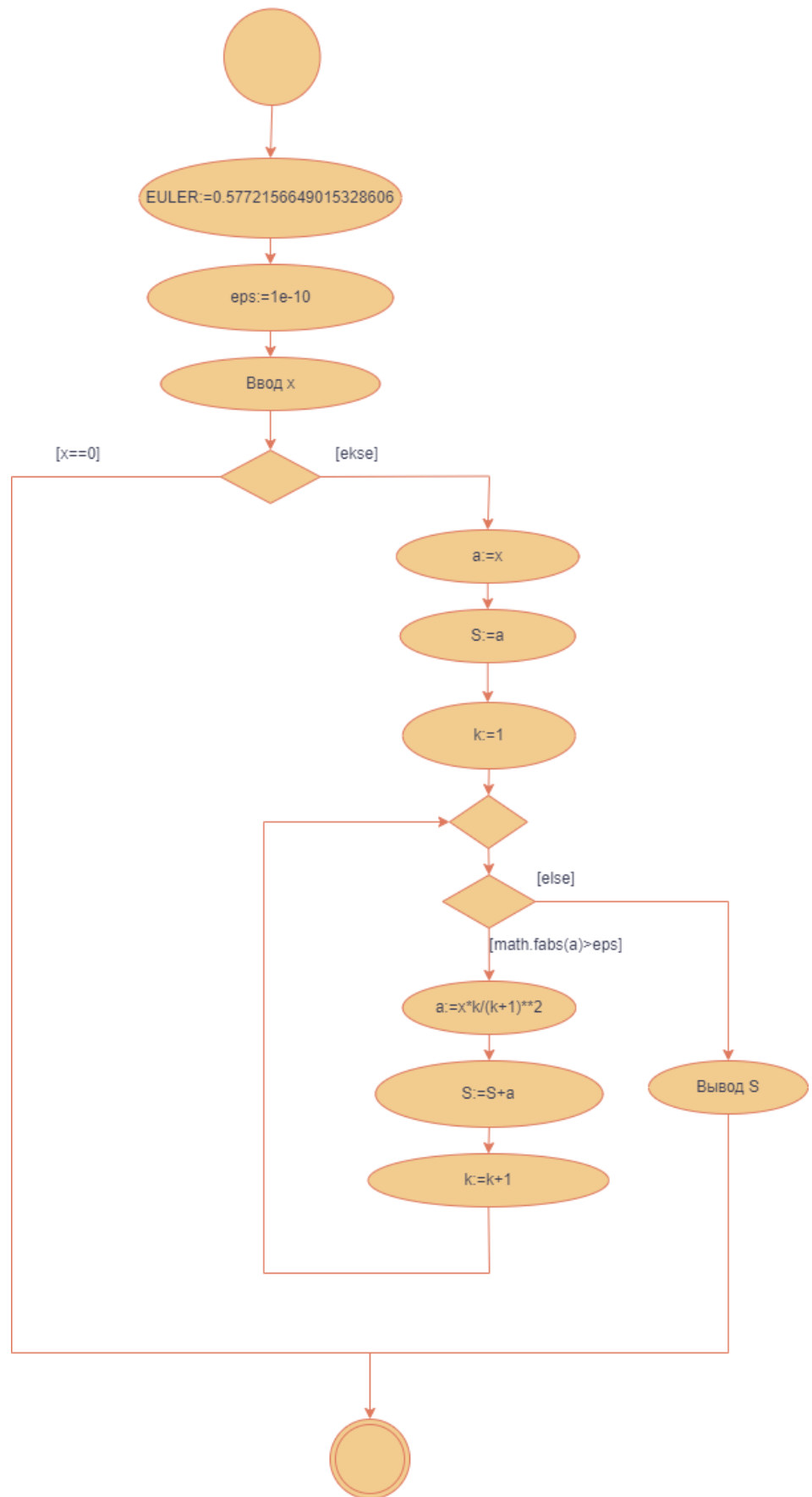
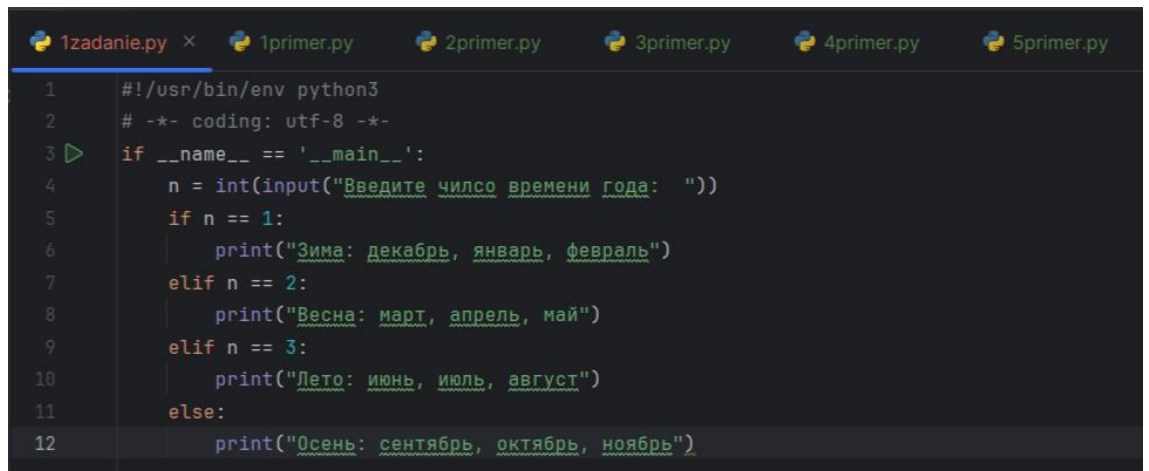


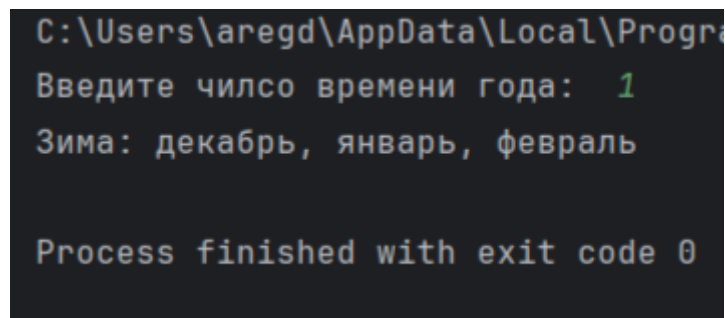
Рисунок 15 - UML-диаграмму деятельности для примера 5

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



```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  if __name__ == '__main__':
4      n = int(input("Введите число времени года: "))
5      if n == 1:
6          print("Зима: декабрь, январь, февраль")
7      elif n == 2:
8          print("Весна: март, апрель, май")
9      elif n == 3:
10         print("Лето: июнь, июль, август")
11     else:
12         print("Осень: сентябрь, октябрь, ноябрь")
```

Рисунок 16 – решение задания 1



```
C:\Users\aregd\AppData\Local\Programs\Python\Python39\python.exe
Введите число времени года: 1
Зима: декабрь, январь, февраль

Process finished with exit code 0
```

Рисунок 17 – результат выполнения задания 1

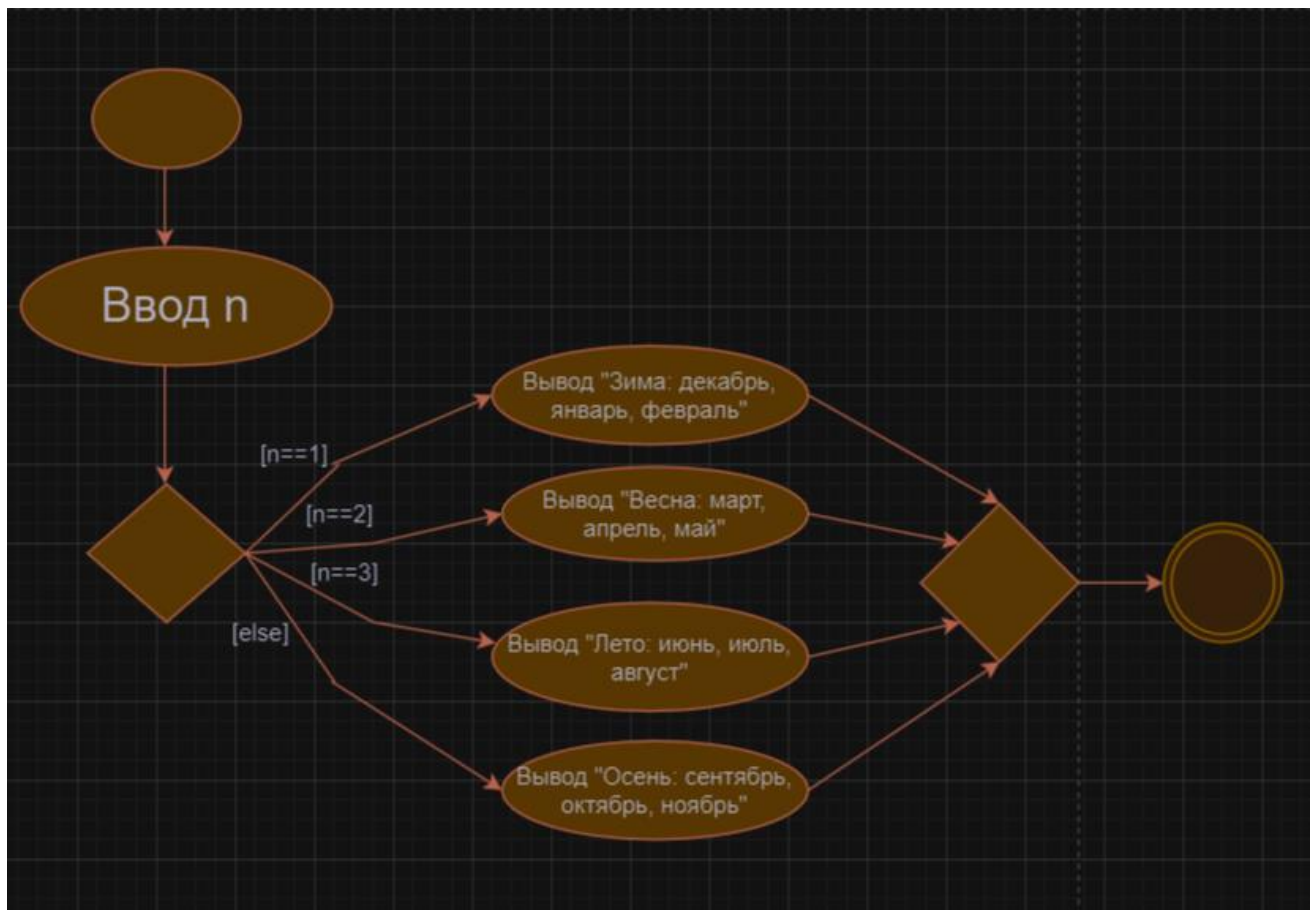


Рисунок 18 - UML-диаграмму деятельности для задания 1

9. Определить принадлежит ли точка $A(a,b)$ кольцу определяемому окружностями $x^2+y^2=1$ $x^2+y^2=0.25$.

```

1zadanie.py 2zadanie.py x 1primer.py 2primer.py
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  if __name__ == '__main__':
4      a = float(input("Введите координату по оси x "))
5      b = float(input("Введите координату по оси y "))
6      distance = a**2 + b**2
7      if 0.25 <= distance <= 1:
8          print("Точка принадлежит кольцу")
9      else:
10         print("Точка не принадлежит кольцу")
11
  
```

Рисунок 19 – решение задания 2

```
Введите координату по оси x 1
Введите координату по оси y 2
Точка не принадлежит кольцу

Process finished with exit code 0
```

Рисунок 20 – результат выполнения задания 2

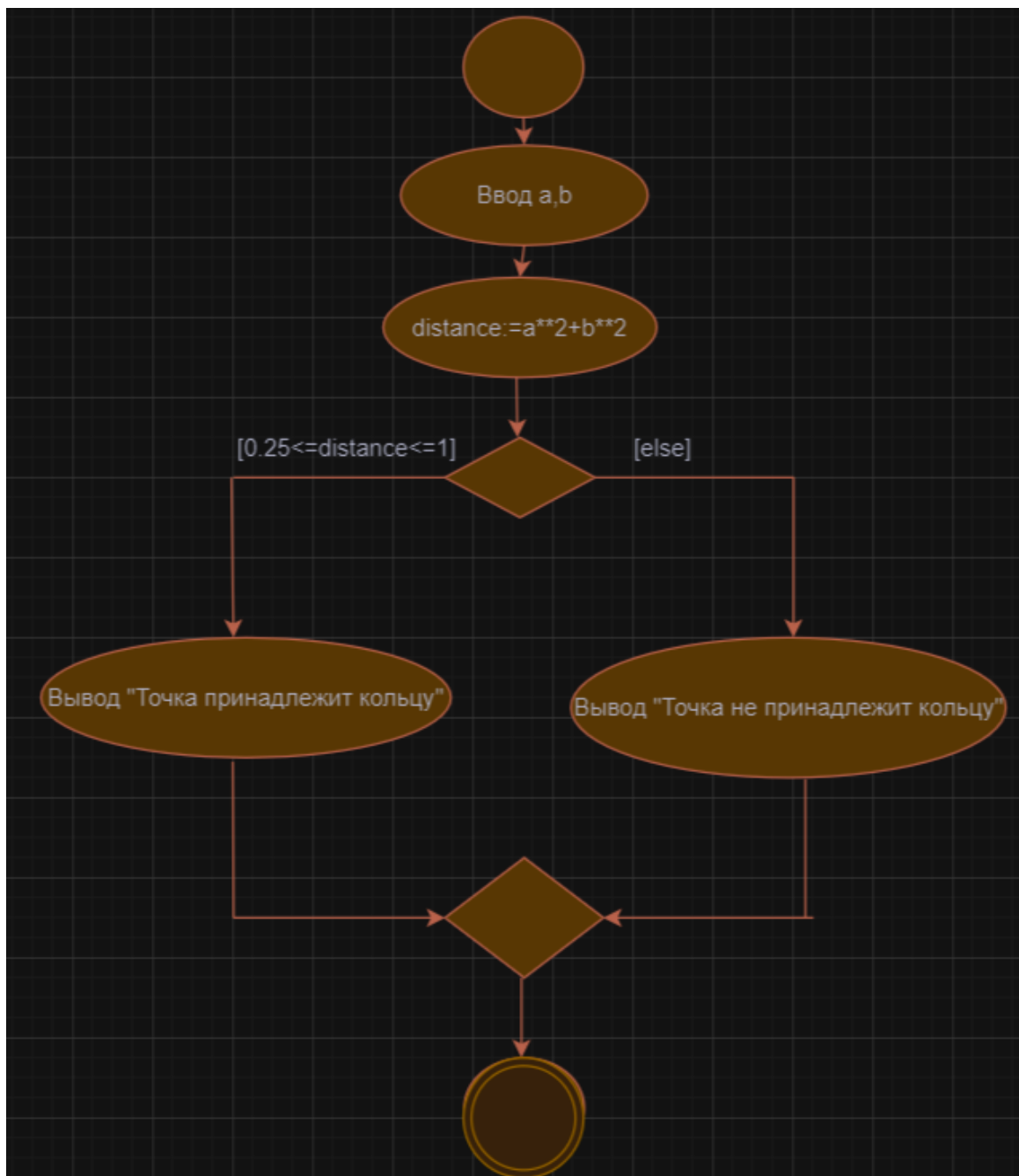
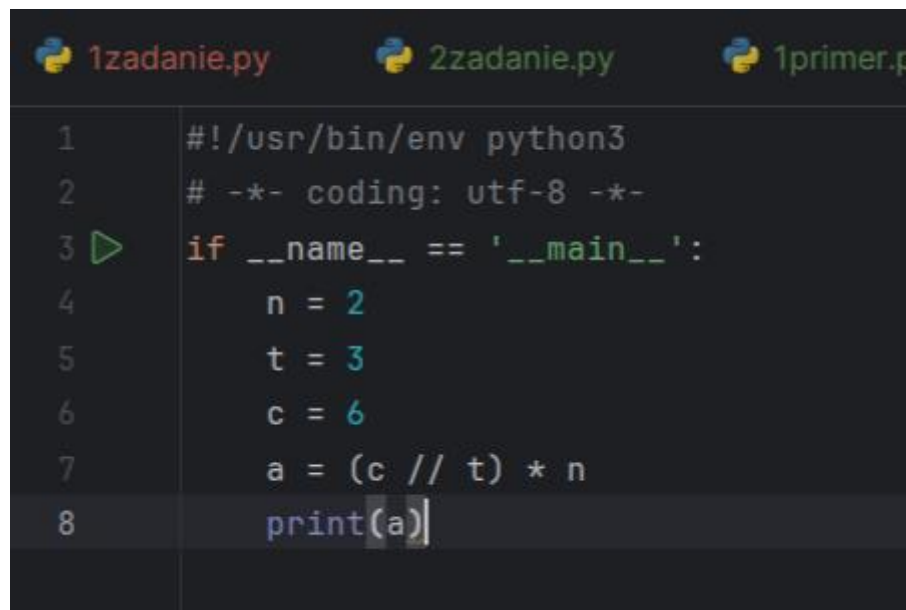


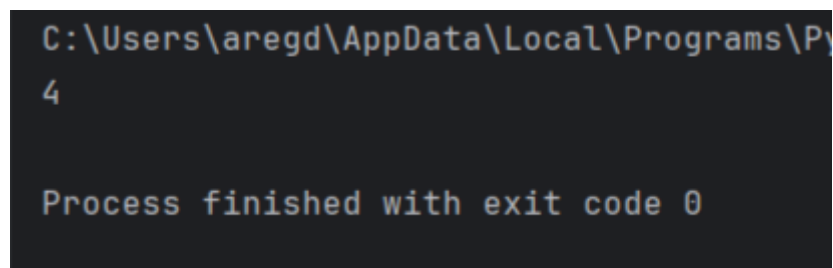
Рисунок 21 - UML-диаграмму деятельности для задания 2

10. Одноклеточная амеба каждые три часа делится на 2 клетки. Определить, сколько будет клеток через 6 часов.

A screenshot of a Python IDE with a dark theme. At the top, there are three tabs: '1zadanie.py' (active), '2zadanie.py', and '1primer.p'. The code in the active tab is as follows:

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  if __name__ == '__main__':
4      n = 2
5      t = 3
6      c = 6
7      a = (c // t) * n
8      print(a)
```

Рисунок 22 – решение задания 3

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Users\aregd\AppData\Local\Programs\Py'. The output of the program is displayed as:

```
4

Process finished with exit code 0
```

Рисунок 23 – результат выполнения задания 3

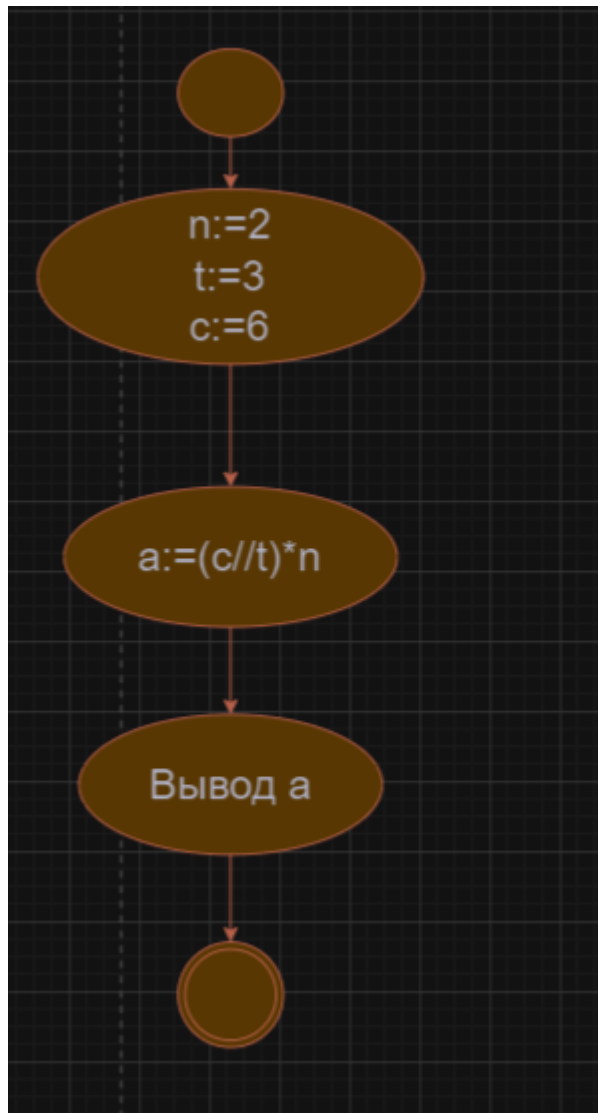


Рисунок 24 - UML-диаграмму деятельности для задания 3

11. Составить UML-диаграмму деятельности, программу и произвести вычисления вычисление значения специальной функции по ее разложению в ряд с точностью 10^{-10} , аргумент функции вводится с клавиатуры.

5. Первый интеграл Френеля:

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

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

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

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

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

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

$$a_0 = \frac{(-1)^0 x^0}{0! (4 \cdot 0 + 1)} = 1$$

```

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}")

```

Рисунок 25 – решение задания повышенной сложности

```

C:\Users\aregd\AppData\Local\Program
Value of x? 2
C(x) = 0.6675968481482645

Process finished with exit code 0

```

Рисунок 26 – результат выполнения задания повышенной сложности

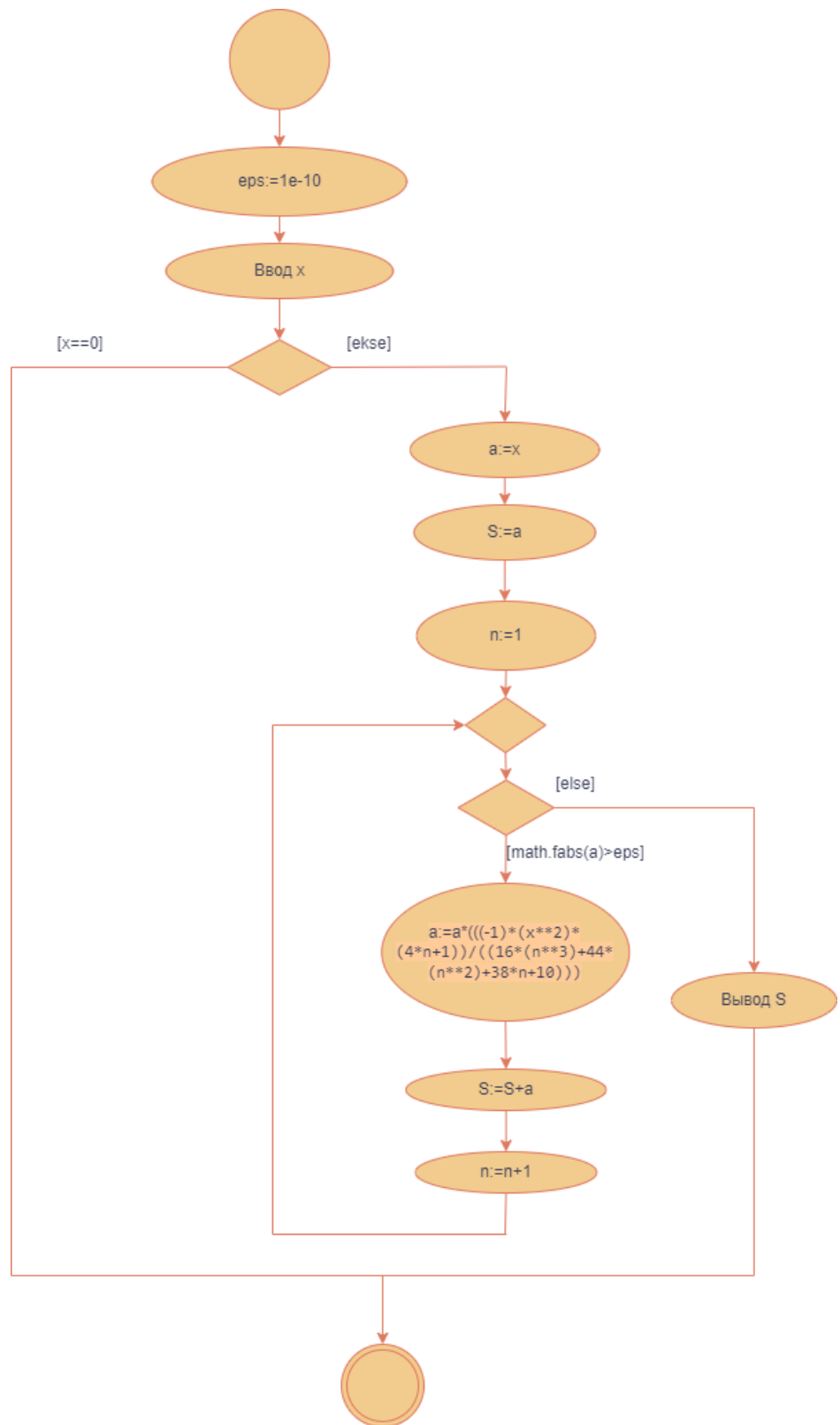


Рисунок 27 - UML-диаграмму деятельности для задания повышенной сложности

12. Зафиксировал все изменения в ветку develop.

```

warning: in the working copy of 'PyCharm/.idea/inspectionProfiles/profiles_se
placed by CRLF the next time Git touches it

aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/git 5/lab5 (develop)
$ git status
On branch develop
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore
        new file:   PyCharm/.idea/.gitignore
        new file:   PyCharm/.idea/.name
        new file:   PyCharm/.idea/git5.iml
        new file:   PyCharm/.idea/inspectionProfiles/profiles_settings.xml
        new file:   PyCharm/.idea/misc.xml
        new file:   PyCharm/.idea/modules.xml
        new file:   PyCharm/.idea/vcs.xml
        new file:   PyCharm/1primer.py
        new file:   PyCharm/1zadanie.py
        new file:   PyCharm/2primer.py
        new file:   PyCharm/2zadanie.py
        new file:   PyCharm/3primer.py
        new file:   PyCharm/3zadanie.py
        new file:   PyCharm/4primer.py
        new file:   PyCharm/5primer.py
        new file:   PyCharm/poslednee.py

aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/git 5/lab5 (develop)
$ git commit -m "Фиксация"
[develop a96591e] Фиксация
17 files changed, 276 insertions(+), 80 deletions(-)
create mode 100644 PyCharm/.idea/.gitignore
create mode 100644 PyCharm/.idea/.name
create mode 100644 PyCharm/.idea/git5.iml
create mode 100644 PyCharm/.idea/inspectionProfiles/profiles_settings.xml
create mode 100644 PyCharm/.idea/misc.xml
create mode 100644 PyCharm/.idea/modules.xml
create mode 100644 PyCharm/.idea/vcs.xml
create mode 100644 PyCharm/1primer.py
create mode 100644 PyCharm/1zadanie.py
create mode 100644 PyCharm/2primer.py
create mode 100644 PyCharm/2zadanie.py
create mode 100644 PyCharm/3primer.py
create mode 100644 PyCharm/3zadanie.py
create mode 100644 PyCharm/4primer.py
create mode 100644 PyCharm/5primer.py
create mode 100644 PyCharm/poslednee.py

aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/git 5/lab5 (develop)
$ ^[[200~
bash: $'\E[200~': command not found

aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/git 5/lab5 (develop)

```

Рисунок 28 – фиксация изменений в ветку develop

13. Слил ветки.

```

aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/git 5/lab5 (develop)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/git 5/lab5 (main)
$ git merge develop
Updating b59fd80..a96591e
Fast-forward
 .gitignore | 185 ++++++-----
 PyCharm/.idea/.gitignore | 8 +
 PyCharm/.idea/.name | 1 +
 PyCharm/.idea/git5.iml | 8 +
 .../.idea/inspectionProfiles/profiles_settings.xml | 6 +
 PyCharm/.idea/misc.xml | 7 +
 PyCharm/.idea/modules.xml | 8 +
 PyCharm/.idea/vcs.xml | 6 +
 PyCharm/1primer.py | 12 ++
 PyCharm/1zadanie.py | 12 ++
 PyCharm/2primer.py | 16 ++
 PyCharm/2zadanie.py | 10 ++
 PyCharm/3primer.py | 11 ++
 PyCharm/3zadanie.py | 8 +
 PyCharm/4primer.py | 16 ++
 PyCharm/5primer.py | 22 +++
 PyCharm/poslednee.py | 20 +++
17 files changed, 276 insertions(+), 80 deletions(-)
create mode 100644 PyCharm/.idea/.gitignore
create mode 100644 PyCharm/.idea/.name
create mode 100644 PyCharm/.idea/git5.iml
create mode 100644 PyCharm/.idea/inspectionProfiles/profiles_settings.xml
create mode 100644 PyCharm/.idea/misc.xml
create mode 100644 PyCharm/.idea/modules.xml
create mode 100644 PyCharm/.idea/vcs.xml
create mode 100644 PyCharm/1primer.py
create mode 100644 PyCharm/1zadanie.py
create mode 100644 PyCharm/2primer.py
create mode 100644 PyCharm/2zadanie.py
create mode 100644 PyCharm/3primer.py
create mode 100644 PyCharm/3zadanie.py
create mode 100644 PyCharm/4primer.py
create mode 100644 PyCharm/5primer.py
create mode 100644 PyCharm/poslednee.py

```

Рисунок 29 – слитие ветки develop в ветку main

Контрольные вопросы:

1. Для чего нужны диаграммы деятельности UML?

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.

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?

Для описания этого потока используются переходы, показывающие путь из одного состояния действия или деятельности в другое. В UML переход представляется простой линией со стрелкой. Поток управления должен где-то начинаться и заканчиваться.

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. 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. 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?

Линейный алгоритм — это такой, в котором все операции выполняются последовательно одна за другой.

Алгоритмы разветвленной структуры применяются, когда в зависимости от некоторого условия необходимо выполнить либо одно, либо другое действие.

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

Оператор ветвления `if` позволяет выполнить определенный набор инструкций в зависимости от некоторого условия. Существует несколько форм конструкций – `if`, `if – else`, `if – elif – else`

7. Какие операторы сравнения используются в Python?

В языках программирования используются специальные знаки, подобные тем, которые используются в математике: `>` (больше), `<` (меньше), `>=` (больше или равно), `<=` (меньше или равно), `==` (равно), `!=` (не равно).

8. Что называется простым условием? Приведите примеры

Простым условием называется выражение, составленное из двух арифметических выражений или двух текстовых величин связанных одним из знаков. Например, логическое выражение типа `kByte >= 1023` является простым, так как в нём выполняется только одна логическая операция.

9. Что такое составное условие? Приведите примеры.

Составное условие – логическое выражение, содержащее несколько простых условий, объединенных логическими операциями. Например, "на улице идет снег или дождь", "переменная `news` больше 12 и меньше 20".

10. Какие логические операторы допускаются при составлении сложных условий?

В таких случаях используются специальные операторы, объединяющие два и более простых логических выражения. Широко используются два оператора – так называемые логические И (`and`) и ИЛИ (`or`).

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

Да, внутри оператора ветвления можно определить и другие ветвления

12. Какой алгоритм является алгоритмом циклической структуры?

Алгоритм циклической структуры – это алгоритм, в котором предусмотрено неоднократное выполнение одной и той же последовательности действий.

13. Типы циклов в языке Python.

В Python есть два вида циклов: `while` и `for`.

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

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

`range(stop)`

```
range(start, stop[, step])
```

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.

Самый простой вариант `range` - передать только значение `stop`. Если передаются два аргумента, то первый используется как `start`, а второй - как `stop`. И чтобы указать шаг последовательности надо передать три аргумента.

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

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

16. Могут ли быть циклы вложенными?

Существует возможность организовать цикл внутри тела другого цикла. Такой цикл будет называться вложенным циклом.

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

Чтобы организовать бесконечный цикл, используют конструкцию `while (true)`. При этом он, как и любой другой цикл, может быть прерван командой `break` или сам прекратит работу, когда его условие работы не равно `True`.

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 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`. 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`. Само же определение потоков `stdout` и `stderr` находится в стандартном пакете Python `sys`.

22. Каково назначение функции `exit`?

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

