

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 №2.2**  
**дисциплины «Искусственный интеллект в профессиональной сфере»**

#  
#  
#  
#  
#

Выполнил:  
Оганесов Артур Витальевич  
3 курс, группа ЭНЭ-б-о-22-1,  
11.03.04 «Электроника и  
наноэлектроника», направленность  
(профиль) «Промышленная  
электроника», очная форма обучения

---

(подпись)

Проверил:  
Воронкин Роман Александрович,  
доцент

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2024 г.

Тема работы: 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.

Аппаратура и материалы: ПК, операционная система Windows 10, Git, браузер для доступа к web-сервису GitHub, Anaconda, PyCharm Community Edition.

Ход работы:

1. Изучил теоретический материал работы, рекомендации к оформлению исходного кода на языке Python PEP-8.
2. На основе полученных знаний создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT и выбранный мной язык программирования (python).

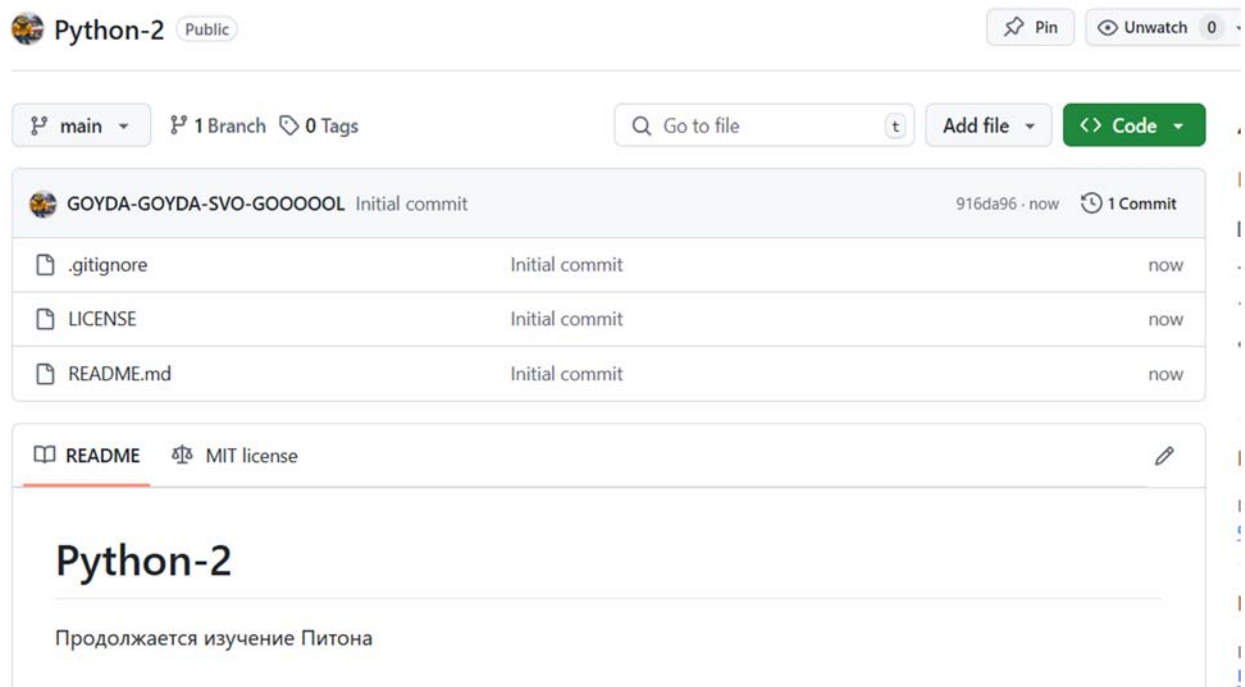
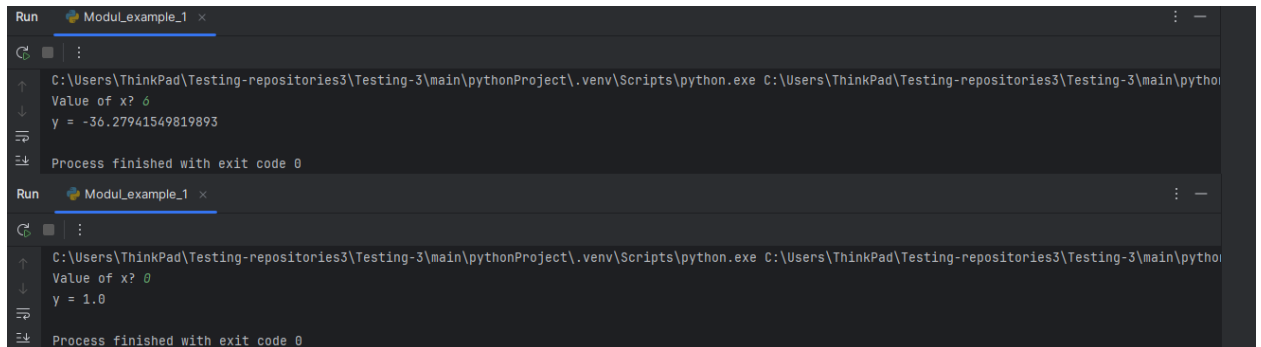


Рисунок 1 – Новый репозиторий

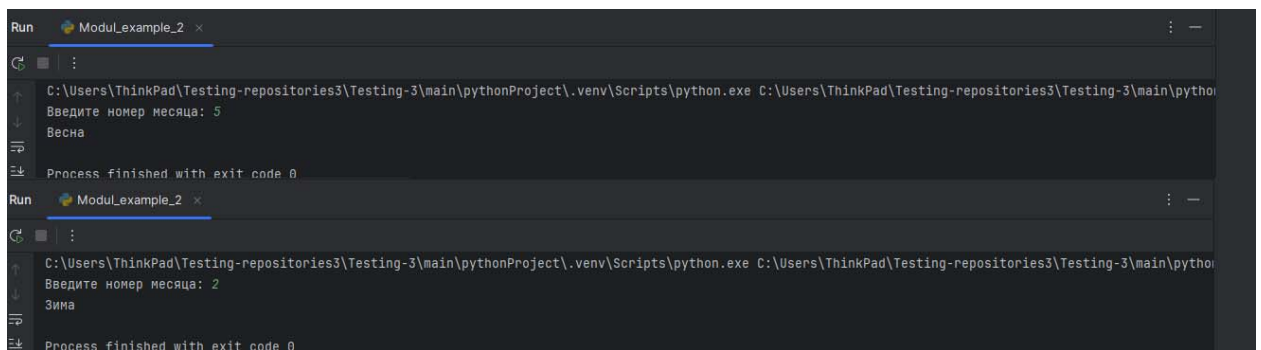
3. Привел скриншоты результатов выполнения каждой из программ примеров при различных исходных данных, вводимых с клавиатуры.



```
Run ModulExample_1 x
C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\pythonProject\.venv\Scripts\python.exe C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\python
Value of x? 0
y = -36.27941549819893
Process finished with exit code 0

Run ModulExample_1 x
C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\pythonProject\.venv\Scripts\python.exe C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\python
Value of x? 0
y = 1.0
Process finished with exit code 0
```

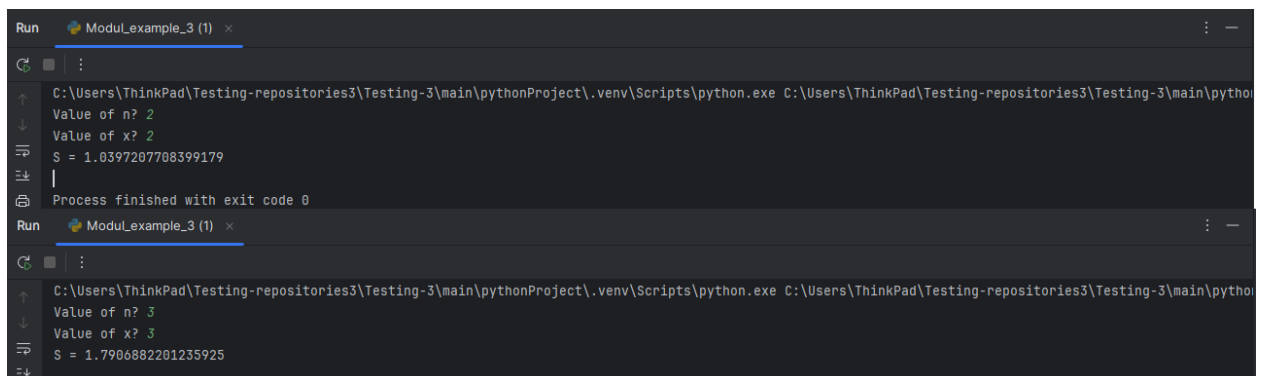
Рисунок 2 – Пример модуля 1



```
Run ModulExample_2 x
C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\pythonProject\.venv\Scripts\python.exe C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\python
Введите номер месяца: 5
Весна
Process finished with exit code 0

Run ModulExample_2 x
C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\pythonProject\.venv\Scripts\python.exe C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\python
Введите номер месяца: 2
Зима
Process finished with exit code 0
```

Рисунок 3 – Пример модуля 2



```
Run ModulExample_3 (1) x
C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\pythonProject\.venv\Scripts\python.exe C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\python
Value of n? 2
Value of x? 2
S = 1.0397207708399179
Process finished with exit code 0

Run ModulExample_3 (1) x
C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\pythonProject\.venv\Scripts\python.exe C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\python
Value of n? 3
Value of x? 3
S = 1.7906882201235925
```

Рисунок 4 –Пример модуля 3

```

Run ModuLexample_4 x
C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\pythonProject\.venv\Scripts\python.exe C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\pythonProject\modul_4.py
Value of a? 1
x = 1.0
X = 1.0
Process finished with exit code 0

Run ModuLexample_4 x
C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\pythonProject\.venv\Scripts\python.exe C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\pythonProject\modul_4.py
Value of a? 3
x = 1.7320508075688772
X = 1.7320508075688772
Process finished with exit code 0

```

Рисунок 5 – Пример модуля 4

```

Run ModuLexample_5 x
C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\pythonProject\.venv\Scripts\python.exe C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\pythonProject\modul_5.py
Value of x? 2
Ei(2.0) = 4.954234355999365
Process finished with exit code 0

Run ModuLexample_5 x
C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\pythonProject\.venv\Scripts\python.exe C:\Users\ThinkPad\Testing-repositories3\Testing-3\main\pythonProject\modul_5.py
Value of x? 5
Ei(5.0) = 40.18527535579794
Process finished with exit code 0

```

Рисунок 6 – Пример модуля 5

4. Для примеров 5 и 6 построил UML-диаграмму деятельности.

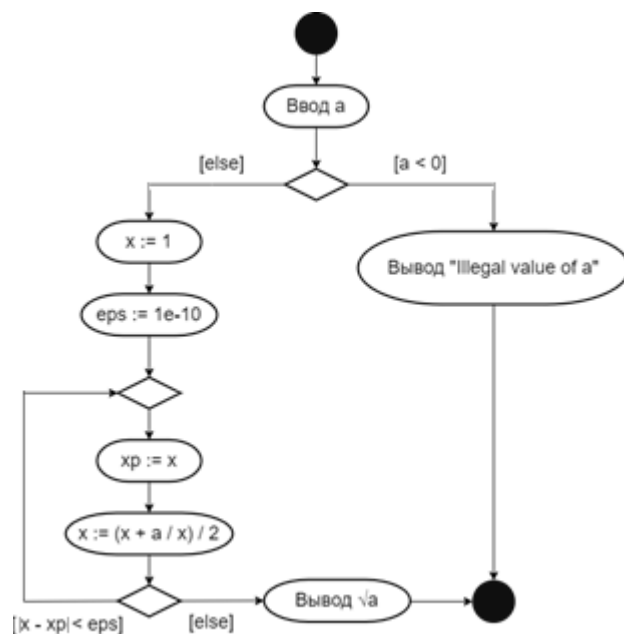


Рисунок 7 – UML-диаграмма модуля 4

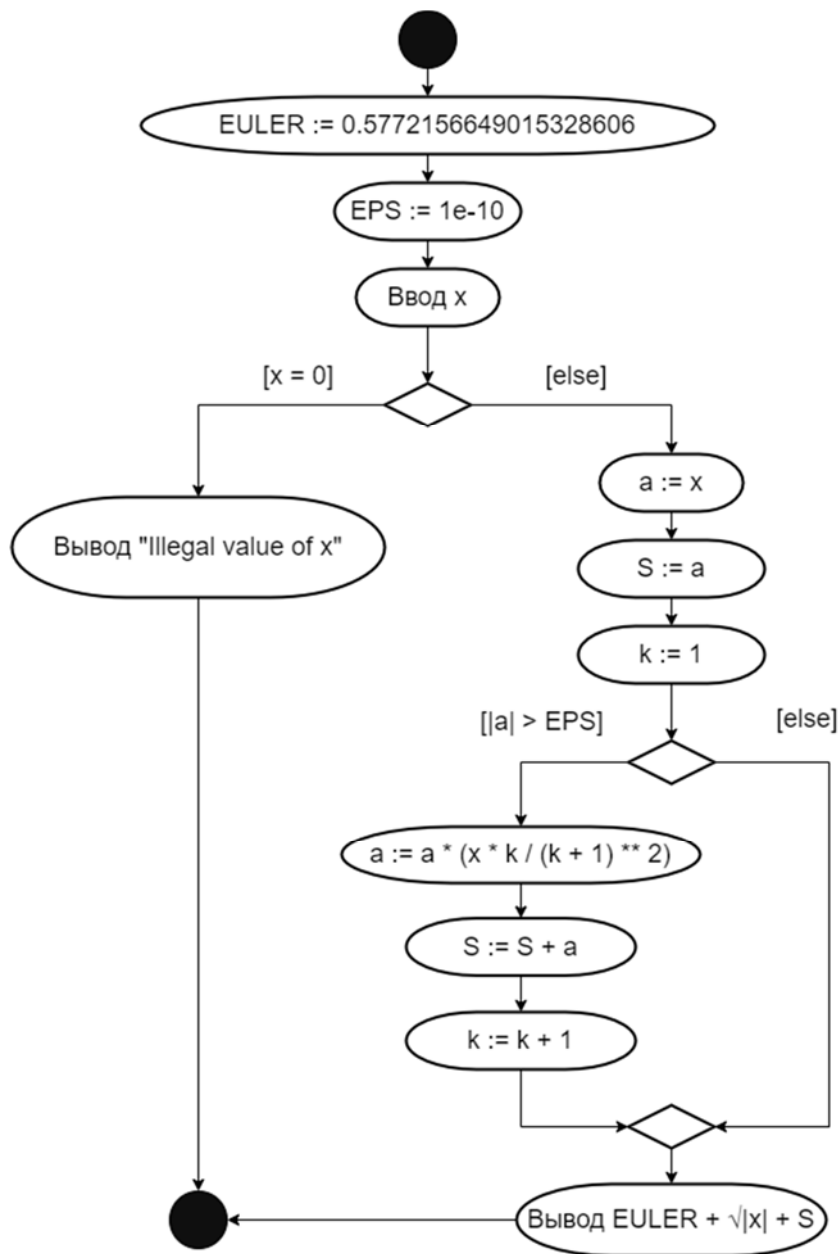


Рисунок 8 – UML-диаграмма модуля 5

## 5. Индивидуальные задания, вариант 10.

10. Вводится число карандашей  $N \leq 10$ . Вывести фразу **я купил N карандашей**, согласовав слово "карандаш" с числом  $N$ .

10. Вывести на экран большее из трёх заданных чисел.

10. Сколько можно купить быков, коров и телят, платя за быка 10 р., за корову - 5 р., а за теленка - 0,5 р., если на 100 р. надо купить 100 голов скота?

Рисунок 14 – Задания 1, 2 и 3

```

Python Console
...     if 11 <= n % 100 <= 19:
...         return "карандашей"
...     elif n % 10 == 1:
...         return "карандаш"
...     elif 2 <= n % 10 <= 4:
...         return "карандаша"
...     else:
...         return "карандашей"
... def main():
...     n = int(input("Введите количество карандашей: "))
...     pencil_ekran = pencil(n)
...     print(f"Я купил {n} {pencil_ekran}.")
... if __name__ == "__main__":
...     main()
...
Введите количество карандашей: >? 8
Я купил 8 карандашей.

```

Рисунок 9 – Решение 1

```

import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['C:\\Users\\Артур\\PycharmProjects\\pythonProject'])

Python Console
...     num1 = int(input("Введи первое число (n < 100): "))
...     num2 = int(input("Введи второе число (n < 100): "))
...     num3 = int(input("Введи третье число (n < 100): "))
...     if num1 >= 100 or num2 >= 100 or num3 >= 100:
...         print("Все числа должны быть меньше 100, переделывай.")
...         return
...     max_num = max(num1, num2, num3)
...     print(f"Наибольшее число: {max_num}")
... if __name__ == "__main__":
...     main()
...
Введи первое число (n < 100): >? 77
Введи второе число (n < 100): >? 3
Введи третье число (n < 100): >? 94
Наибольшее число: 94

```

Рисунок 10 – Решение 2

```

Python Console
>>>
>>> for x in range(0, 11): # Максимум 10 быков, так как 10 * 10 = 100
...     for y in range(0, 21): # Максимум 20 коров, так как 5 * 20 = 100
...         z = 100 - x - y
...         if z >= 0 and (10 * x + 5 * y + 0.5 * z == 100):
...             print(f"Быков: {x}, Коров: {y}, Телят: {z}")
...
Быков: 1, Коров: 9, Телят: 90

```

Рисунок 11 – Решение 3

6. Привел в отчете UML-диаграммы деятельности решения индивидуальных заданий.

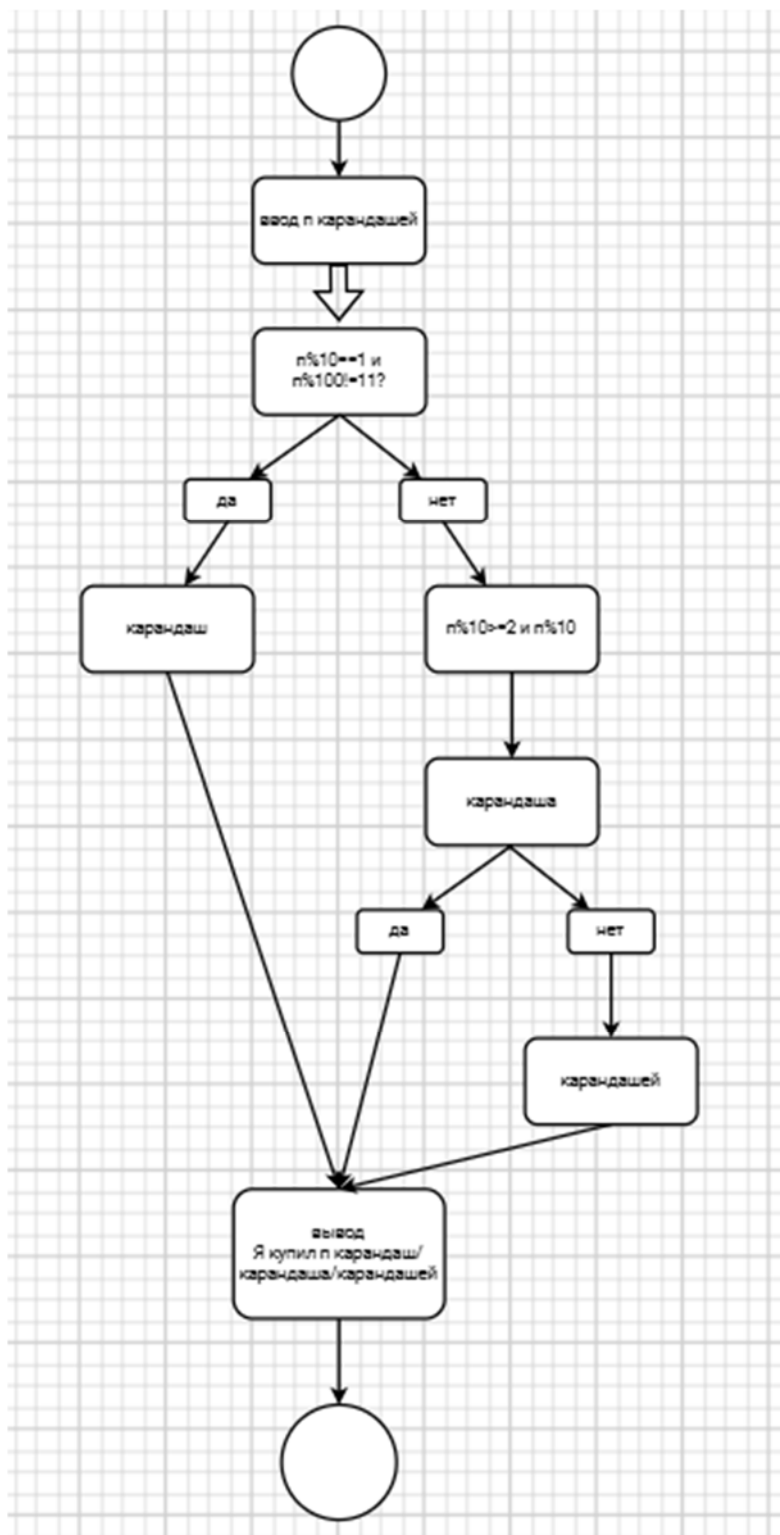


Рисунок 12 – UML-диаграмма 1

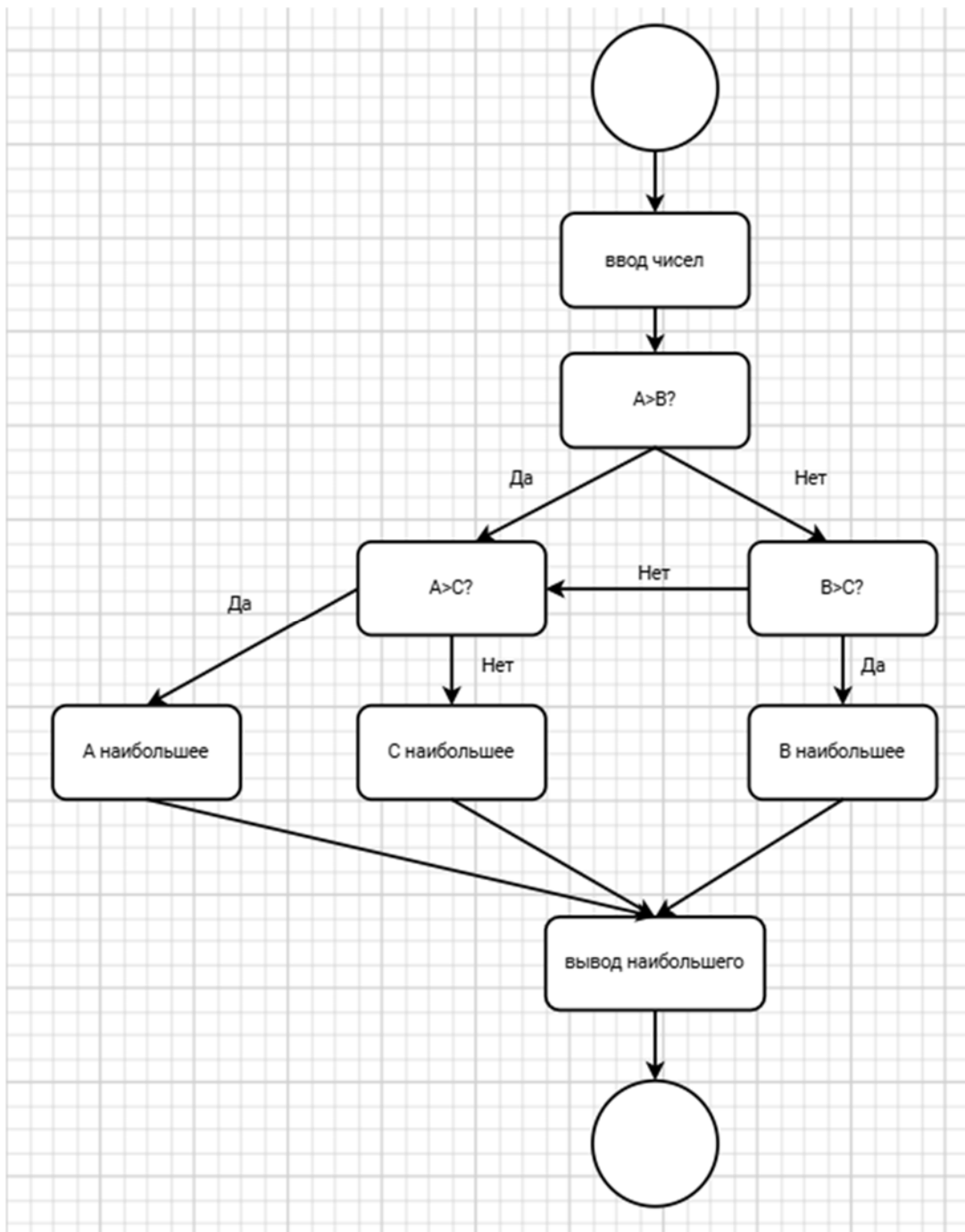


Рисунок 13 – UML-диаграмма 2



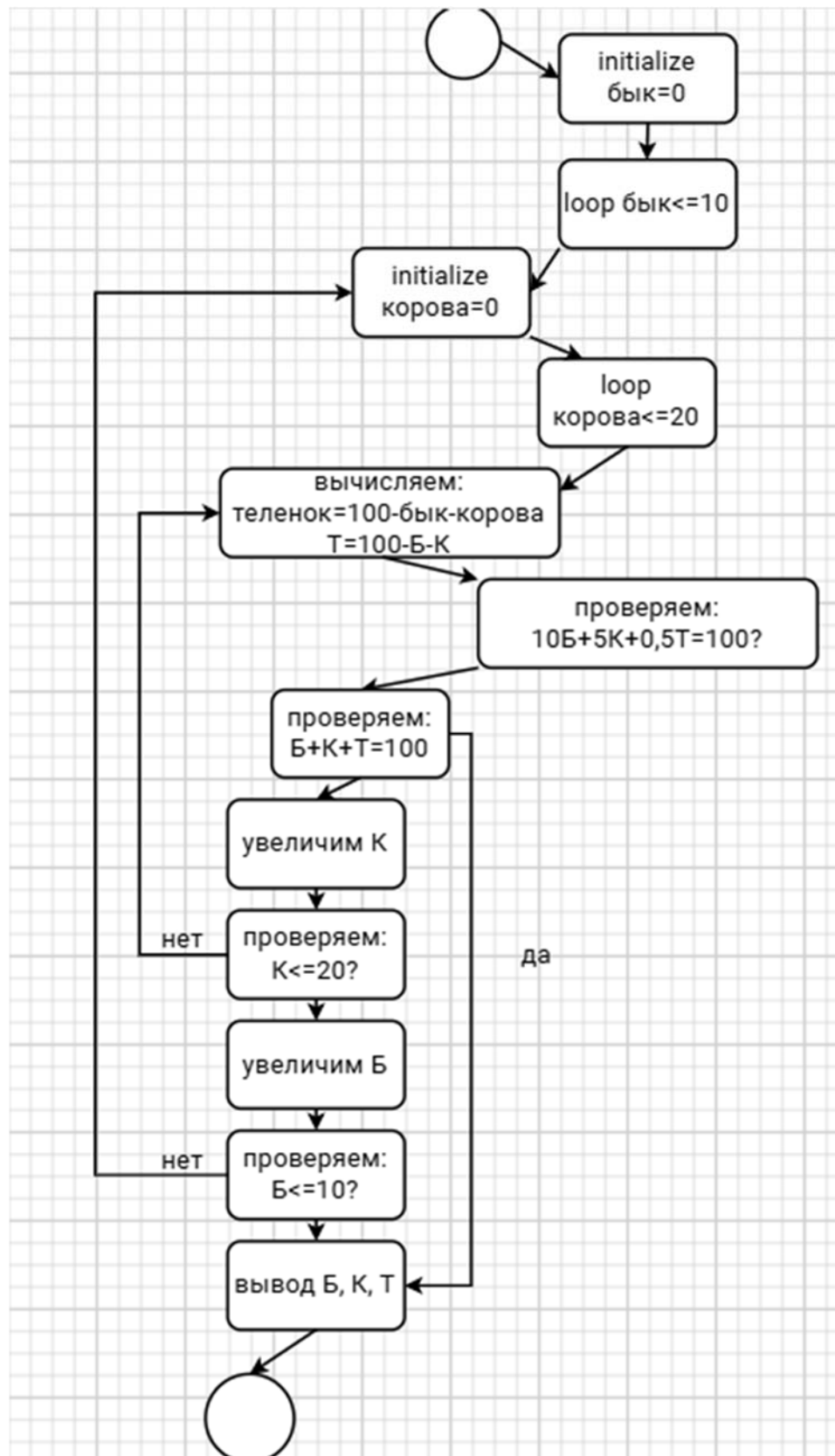


Рисунок 14 – UML-диаграмма 3

Ответы на вопросы:

1. Для чего нужны диаграммы деятельности UML?

UML-диаграмма — это просто схема, которую сделали по правилам языка UML. Обычно в ней есть элементы и связи между ними, а также дополнительные объекты, которые помогают лучше понять диаграмму.

С помощью UML-схем можно описывать разные процессы и явления.

2. Что такое состояние действия и состояние деятельности?

Состояние действия (action state) — состояние, которое представляет вычисление атомарного действия, как правило — вызов операции. Переход из состояния действия происходит после завершения входного действия. Состояние действия не может иметь внутренних переходов, поскольку оно является элементарным. Обычное использование состояния действия заключается в моделировании шага выполнения алгоритма или процедуры в рамках одного потока управления.

Состояние деятельности (activity state) — состояние в графе деятельности, которое служит для представления процедурной последовательности действий, требующих определённого времени. Переход из состояния деятельности происходит после выполнения специфицированной в нём деятельности, при этом ключевое слово do в имени деятельности не указывается. Состояние деятельности не может иметь внутренних переходов, поскольку оно является элементарным. Деятельность, описанная в состоянии деятельности, не может быть прервана никакими внешними событиями. Обычное использование состояния деятельности заключается в моделировании подпроцесса выполнения отдельных алгоритмов или процедур.

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

Для обозначения переходов и ветвлений в диаграммах деятельности используются следующие нотации:

Переход — на диаграмме деятельности изображается сплошной линией со стрелкой. Если из состояния действия выходит единственный переход, то он может быть никак не помечен. Если же таких переходов несколько, то сработать может только один из них. Для каждого из таких переходов должно быть явно записано сторожевое условие в прямых скобках.

Ветвление — на диаграмме деятельности обозначается небольшим ромбом, внутри которого нет никакого текста. В этот ромб может входить только одна стрелка от того состояния действия, после выполнения которого поток управления должен быть продолжен по одной из взаимно исключающих ветвей.

#### 4. Какой алгоритм является алгоритмом разветвляющейся структуры?

Алгоритмом разветвляющейся структуры называют такой алгоритм, в котором в зависимости от выполнения некоторого логического условия происходит разветвление вычислений по одному из нескольких возможных направлений.

#### Чем отличается разветвляющийся алгоритм от линейного?

Разветвляющийся алгоритм отличается от линейного тем, что в нём в зависимости от условия выполняется либо одна, либо другая последовательность действий. В линейном алгоритме все действия выполняются последовательно друг за другом.

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

Условный оператор или оператор ветвления - это оператор, конструкция языка программирования, обеспечивающая выполнение определённой команды (набора команд) только при условии истинности некоторого логического выражения, либо выполнение одной из нескольких команд (наборов команд) в зависимости от значения некоторого выражения.

Существует две основные формы условной инструкции, встречающиеся в реальных языках программирования: условный оператор (оператор if) и оператор многозначного выбора (переключатель, case, switch).

#### 7. Какие операторы сравнения используются в Python?

`==`. Возвращает True, если оба операнда равны. Иначе возвращает False.

`!=`. Возвращает True, если оба операнда НЕ равны. Иначе возвращает False.

`>` (больше чем). Возвращает True, если первый операнд больше второго.

`<` (меньше чем). Возвращает True, если первый операнд меньше второго.

`>=` (больше и равно). Возвращает True, если первый операнд больше или равен второму.

`<=` (меньше или равно). Возвращает True, если первый операнд меньше или равен второму.

8. Что называется простым условием? Приведите примеры.

Простое условие — это два выражения, связанные одним из знаков отношений: `=` (равно), `>` (больше), `<` (меньше), `>=` (больше либо равно), `<=` (меньше либо равно), `<>` (не равно).

Примеры простых условий: `a > 0`; `a + c <= 0`.

9. Что такое составное условие? Приведите примеры.

Составные условия – условия, состоящие из двух или более простых условий, соединенных с помощью логических операций `and` (и), `or` (или), `not` (не). Простые условия при этом заключаются в круглые скобки.

Примеры составных условий: `(a>5) and (a<13)`; `(x>=0) or (a<>7)`.

10. Какие логические операторы допускаются при составлении сложных условий?

Для составления сложных условий в языке программирования Python можно использовать следующие логические операторы:

`and` — логическое «И» для двух условий. Возвращает True, если оба условия истинны, иначе возвращает False.

`or` — логическое «ИЛИ» для двух условий. Возвращает False, если оба условия ложны, иначе возвращает True.

`not` — логическое «НЕ» для одного условия. Возвращает False для истинного условия и наоборот.

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

Да, оператор ветвления может содержать внутри себя другие ветвления.

Такой оператор ветвления называется вложенным. В этом случае важно не перепутать, какая ветвь кода к какому оператору относится. Поэтому рекомендуется соблюдать отступы в исходном коде программы, чтобы не запутаться.

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.

«For» — применяется, когда нужно выполнить перебор элементов заранее известное количество раз.

«While» — применяется, когда численность итераций заранее не известна. Блок операторов выполняется до тех пор, пока не будет выполнено условие, указанное в цикле.

«do-while» — продолжается вплоть до момента, когда будет выполнено заданное условие. Применяется в ситуациях, при которых стоит задача как минимум однократного выполнения цикла.

14. Назовите назначение и способы применения функции range.

Назначение функции range() в Python — генерировать последовательность чисел в заданном диапазоне с заданным шагом.

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

Чтобы организовать перебор значений от 15 до 0 с шагом 2 с помощью функции range в Python, можно использовать следующий код: `for i in range(15, 0, 2):`

16. Могут ли быть циклы вложенными?

Да, циклы могут быть вложенными.

Это означает, что внутри тела цикла можно определить и вызвать другой цикл. Внутренний цикл выполнится и будет повторяться столько раз, сколько описано в условии внешнего. При выходе из основного цикла вложенный уже не будет исполняться, однако при выходе из вложенного происходит возврат к основному.

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

Бесконечный цикл образуется, если в цикле нет условия для выхода. Чаще всего таким становится цикл `while`, но в теории таким можно сделать и цикл `for` — например, если не менять итератор или установить невозможное условие для выхода.

Чтобы выйти из бесконечного цикла, можно использовать следующие методы:

Невыполнение условия. Итератор достиг нужного значения, или переменная, которую нужно было отыскать, найдена. Тогда цикл прекращается, программа начинает выполнять код, который был написан после него.

Пропуск итерации. Если по какой-то причине в цикле нужно закончить итерацию раньше времени и перейти на следующую, для этого есть специальная команда. Обычно она называется `continue`.

Выход из цикла. Для этого во многих языках программирования существует команда `break`. Она означает «Прерви выполнение подпрограммы и выйди из неё». Когда программа доходит до этой команды, она выходит из цикла или условия и начинает выполнять код, который идёт дальше. Никаких итераций больше не происходит.

#### 18. Для чего нужен оператор `break`?

Оператор `break` нужен для прерывания выполнения цикла или оператора. Он позволяет программе выйти из цикла до того, как будет выполнено обычное условие цикла.

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

Оператор `continue` используется в структуре управления циклом, когда вам нужно немедленно перейти к следующей итерации цикла. Его можно использовать с циклом `for` или циклом `while`.

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

`stdout` выводит выходные данные (чаще всего в формате текста) — результат выполнения программы. По умолчанию нацелен на запись на устройство отображения (монитор).

`stderr` выводит диагностические или отладочные сообщения, либо информацию об ошибках в работе программы. Чаще всего цель этого потока совпадает с `stdout`, однако, в отличие от него, цель потока `stderr` не меняется при перенаправлении вывода, то есть отладочные сообщения процесса, вывод которого перенаправлен, всё равно попадут пользователю

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

Использование `sys.stderr.write()`. Использование функции `print()`. Использование `os.write()`.

22. Каково назначение функции `exit`?

Функция `exit` выполняет немедленное завершение программы.

Вывод: в ходе выполнения лабораторной работы были приобретены навыки программирования разветвляющихся алгоритмов и алгоритмов циклической структуры. Освоены операторы языка Python версии 3.x `if`, `while`, `for`, `break` и `continue`, позволяющие реализовывать разветвляющиеся алгоритмы и алгоритмы циклической структуры.