

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 №5**  
**дисциплины «Основы программной инженерии»**

Выполнила:  
Панюкова Ксения Юрьевна  
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:  
Воронкин Р. А., доцент кафедры  
инфокоммуникаций

---

(подпись)

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

Ставрополь, 2023 г.

## Ход работы

### 1. Я изучила теоретический материал работы

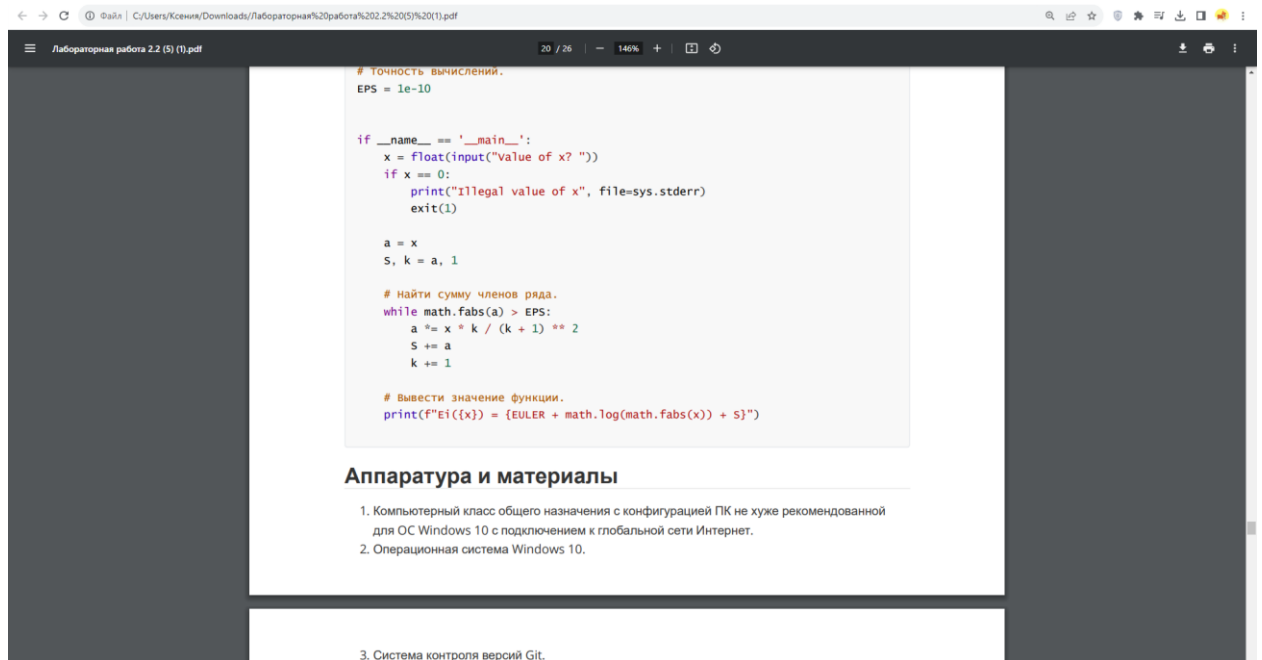



Рисунок 1.1 – Изучение материала для лабораторной работы

2. Создала общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python

## Create a new repository



A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (\*).

Owner \*  
 MrFinicheck / Repository name \*  
EveryoneLovesPython  
EveryoneLovesPython is available.

Great repository names are short and memorable. Need inspiration? How about [expert-octo-doodle](#) ?

Description (optional)

- ☒  **Public**  
Anyone on the internet can see this repository. You choose who can commit.
- ☐  **Private**  
You choose who can see and commit to this repository.

Initialize this repository with:

- ☐ **Add a README file**  
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore


.gitignore template: Python

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: MIT License


A license tells others what they can and can't do with your code. [Learn more about licenses.](#)


 You are creating a public repository in your personal account.


Create repository


## Рисунок 2.1 – Настройка репозитория

[Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)


 **EveryoneLovesPython** Public [Pin](#) [Unwatch](#) 1 [Fork](#) 0 [Star](#) 0


 **main**


 1 branch

 0 tags


[Go to file](#) [Add file](#) [Code](#)

 **MrFinicheck** Initial commit f8437c6 now 1 commit


 .gitignore Initial commit now


 LICENSE Initial commit now


Help people interested in this repository understand your project by adding a README. [Add a README](#)


**About** 

No description, website, or topics provided.

 Activity

 0 stars

 1 watching

 0 forks

**Releases**

No releases published  
[Create a new release](#)

**Packages**

No packages published  
[Publish your first package](#)

 © 2023 GitHub, Inc. [Terms](#) [Privacy](#) [Security](#) [Status](#) [Docs](#) [Contact GitHub](#) [Pricing](#) [API](#) [Training](#) [Blog](#) [About](#)

## Рисунок 2.2 – Готовый репозиторий

### 3. Выполняю клонирование созданного репозитория

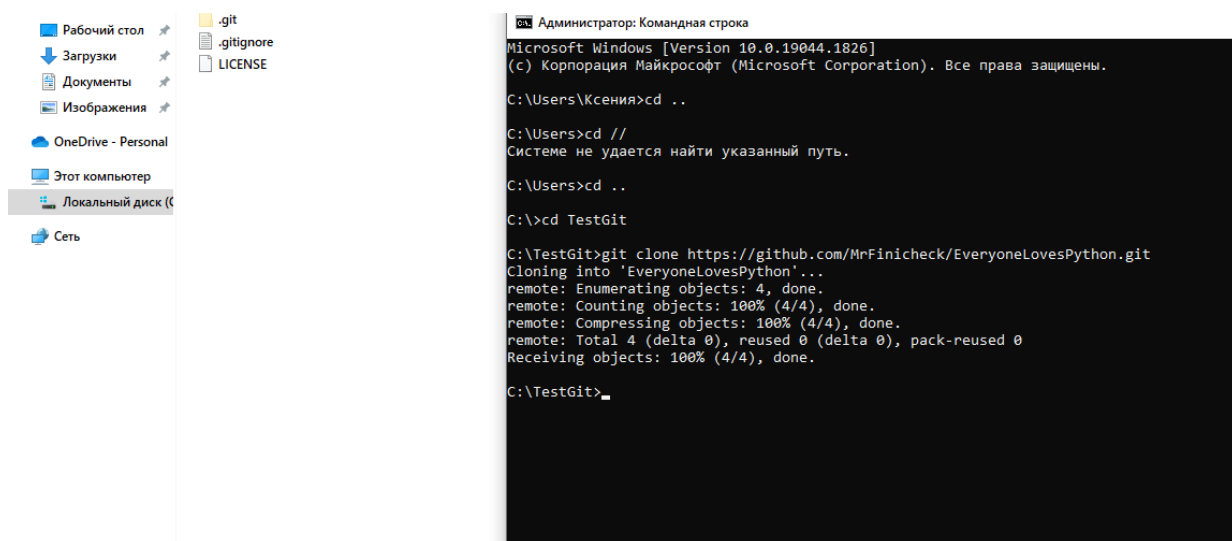


Рисунок 3.1 – Клонирование репозитория на локальный диск

### 4. Дополнила файл .gitignore необходимыми правилами для работы с IDE PyCharm

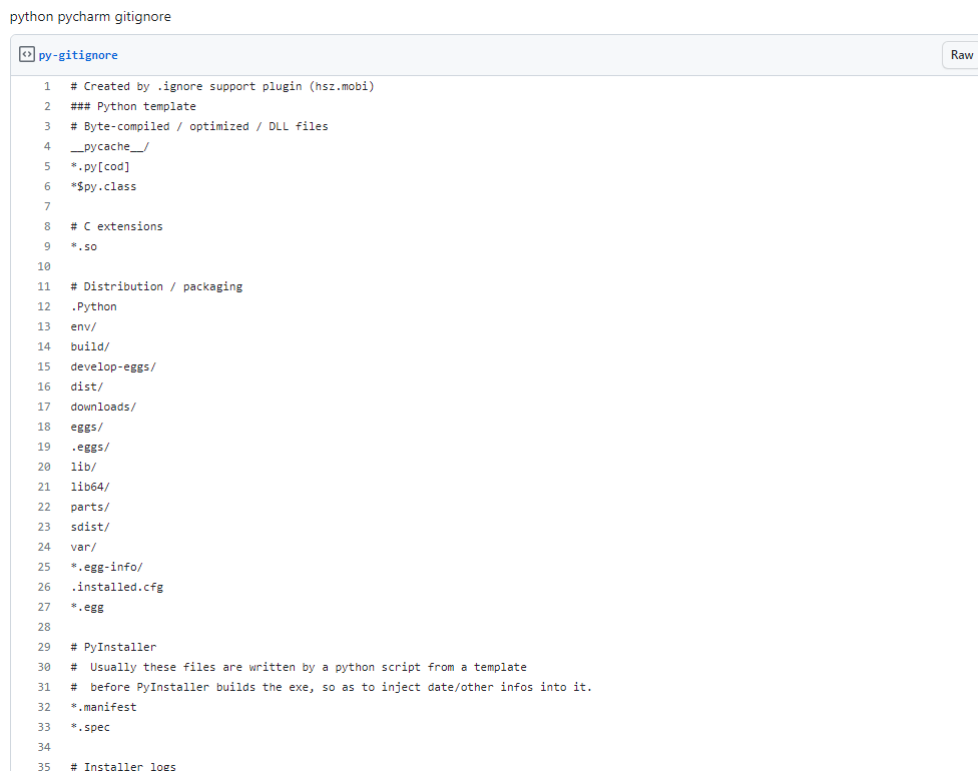


Рисунок 4.1 – .gitignore для IDE PyCharm

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

```
C:\TestGit\EveryoneLovesPython>git branch develop  
  
C:\TestGit\EveryoneLovesPython>git checkout develop  
Switched to branch 'develop'  
  
C:\TestGit\EveryoneLovesPython>
```

Рисунок 5.1 – Создание ветки develop от ветки main

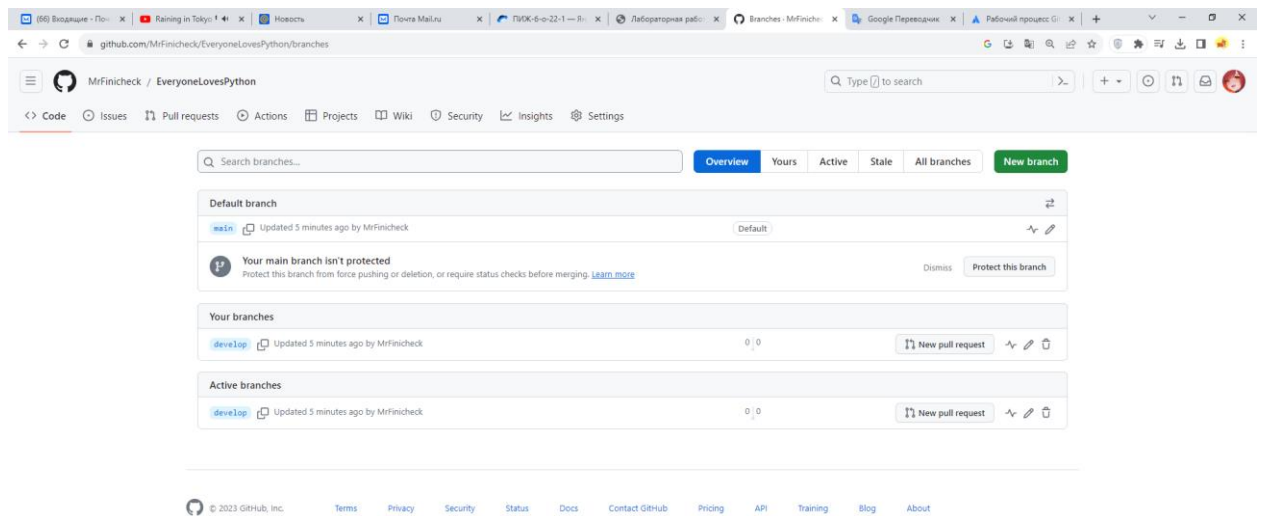


Рисунок 5.2 – Ветка develop на GitHub

6. Самостоятельно изучила рекомендации к оформлению исходного кода на языке Python PEP-8. Выполнила оформление исходного примеров лабораторной работы и индивидуальных созданий в соответствии с PEP-8

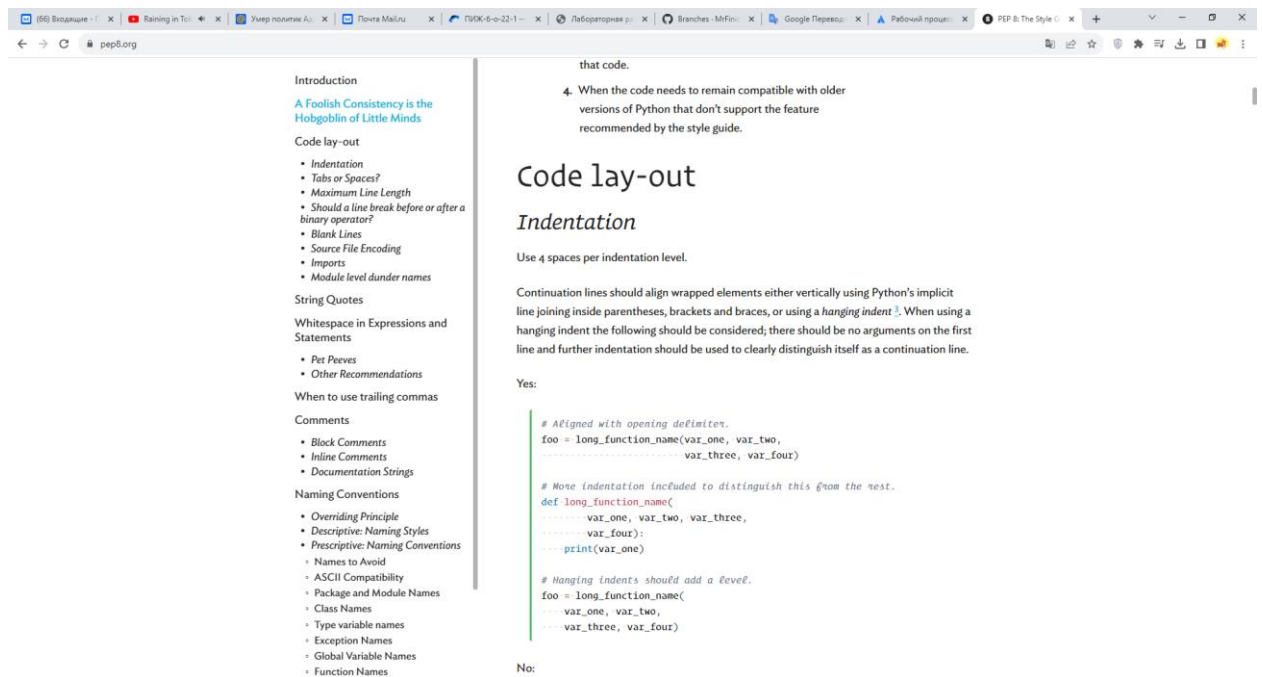


Рисунок 6.1 – Изучение Python PEP-8

## 7. Создала проект PyCharm в папке репозитория

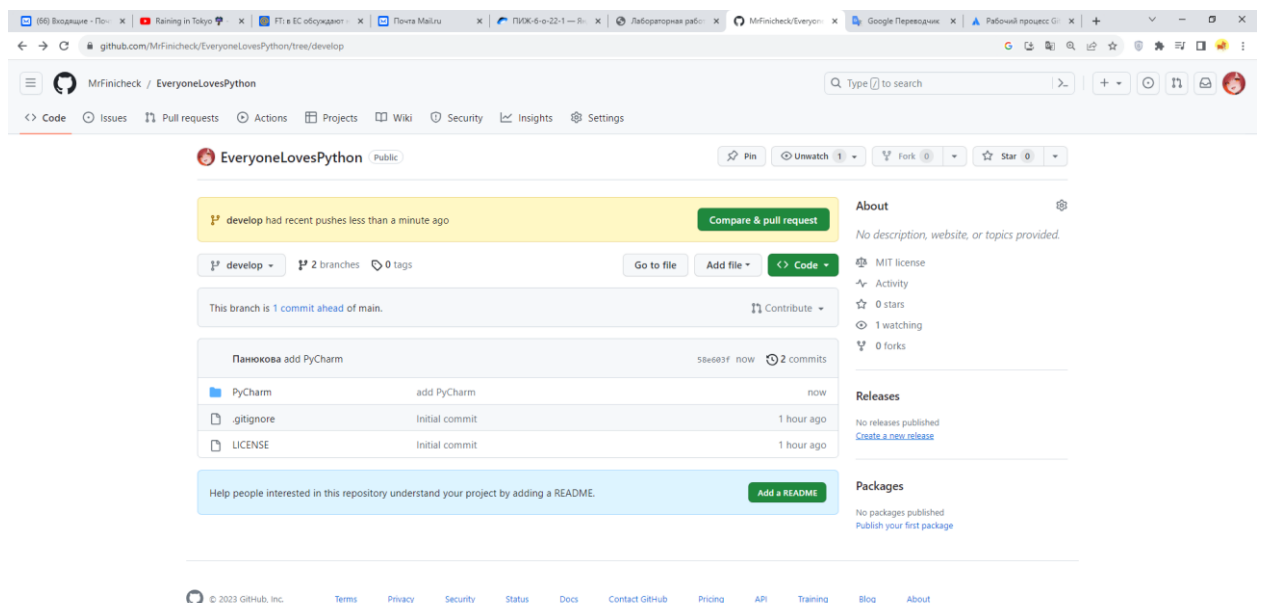


Рисунок 7.1 – Репозиторий с проектом PyCharm

8. Проработала примеры лабораторной работы. Создала для каждого примера отдельный модуль языка Python. Зафиксировала изменения в репозитории

```
PP pythonProject2 Version control
main.py example1.py x advancedtask.py user.py arithmetic.py
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 8.1 – Проработка примера 1

```
main.py example1.py example2.py x advancedtask.py user.py arithmetic.py
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 import sys
5
6 if __name__ == '__main__':
7     n = int(input("Введите номер месяца: "))
8     if n == 1 or n == 2 or n == 12:
9         print("Зима")
10    elif n == 3 or n == 4 or n == 5:
11        print("Весна")
12    elif n == 6 or n == 7 or n == 8:
13        print("Лето")
14    elif n == 9 or n == 10 or n == 11:
15        print("Осень")
16    else:
17        print("Ошибка!", file=sys.stderr)
18    exit(1)
```

Рисунок 8.2 – Проработка примера 2

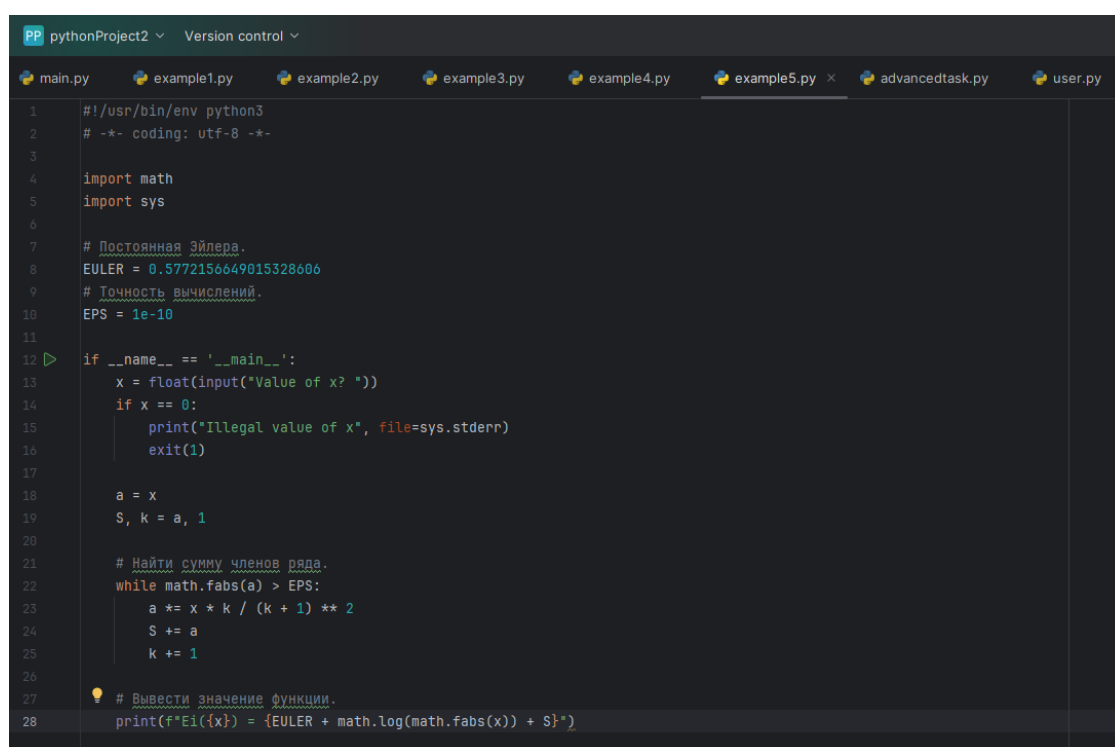
```
main.py example1.py example2.py example3.py x advancedtask.py user.py arithmetic.py
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 import math
5
6 if __name__ == '__main__':
7     n = int(input("Value of n? "))
8     x = float(input("Value of x? "))
9     S = 0.0
10    for k in range(1, n + 1):
11        a = math.log(k * x) / (k * k)
12        S += a
13    print(f"S = {S}")
```

Рисунок 8.3 – Проработка примера 3



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

Рисунок 8.4 – Проработка примера 4



```
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
18     a = x
19     S, k = a, 1
20
21     # Найти сумму членов ряда.
22     while math.fabs(a) > EPS:
23         a *= x * k / (k + 1) ** 2
24         S += a
25         k += 1
26
27     # Вывести значение функции.
28     print(f'Ei({x}) = {EULER + math.log(math.fabs(x)) + S}')
```

Рисунок 8.5 – Проработка примера 5



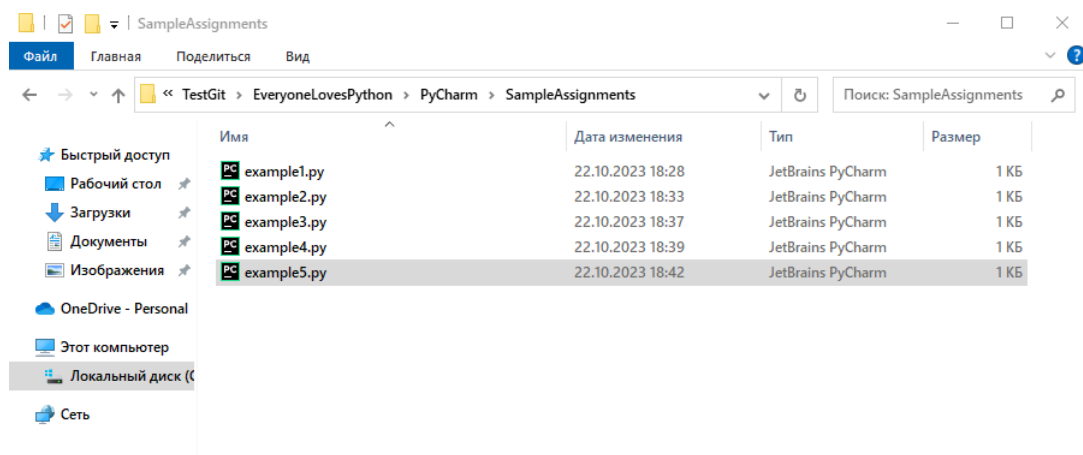


Рисунок 8.6 – Создание отдельных модулей для каждого из примеров

```
C:\TestGit\EveryoneLovesPython>git add PyCharm

C:\TestGit\EveryoneLovesPython>git commit -m"adding examples"
[develop 2d7ce83] adding examples
6 files changed, 93 insertions(+)
create mode 100644 PyCharm/SampleAssignments/example1.py
create mode 100644 PyCharm/SampleAssignments/example2.py
create mode 100644 PyCharm/SampleAssignments/example3.py
create mode 100644 PyCharm/SampleAssignments/example4.py
create mode 100644 PyCharm/SampleAssignments/example5.py
delete mode 100644 PyCharm/python.txt
```

Рисунок 8.7 – Фиксирование изменений в репозитории

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

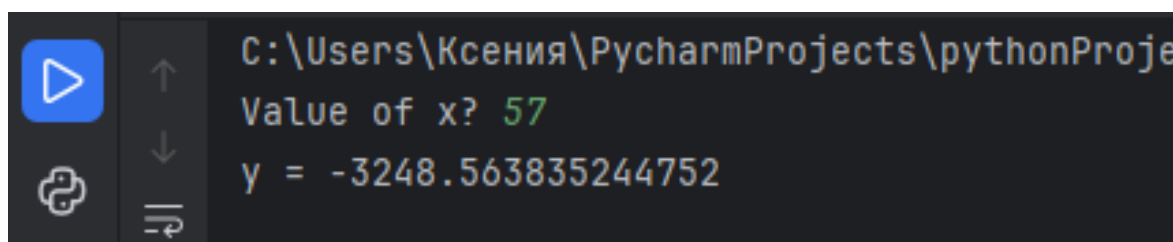
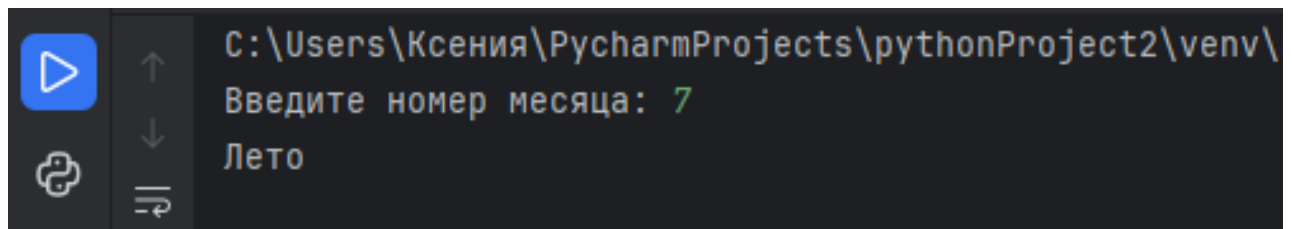
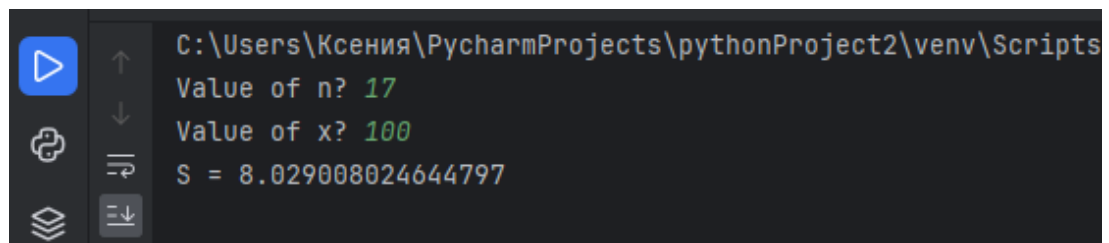


Рисунок 9.1 – Результат примера 1



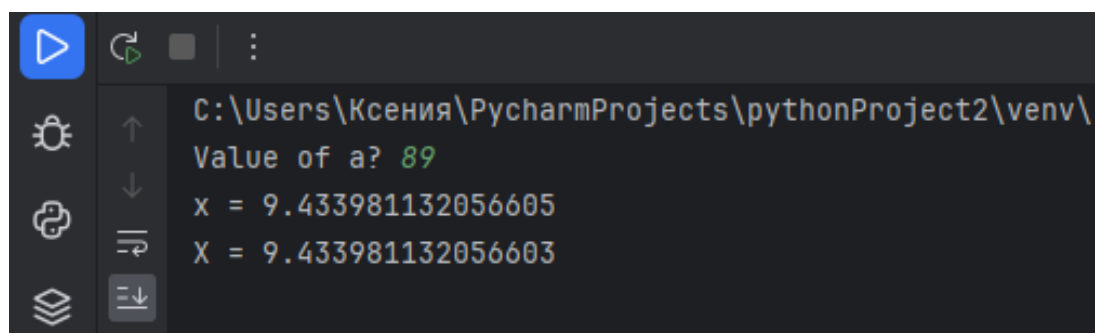
```
C:\Users\Ксения\PycharmProjects\pythonProject2\venv\
Введите номер месяца: 7
Лето
```

Рисунок 9.2 – Результат примера 2



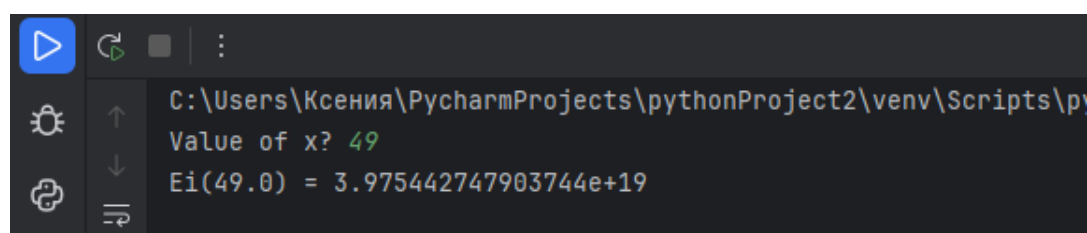
```
C:\Users\Ксения\PycharmProjects\pythonProject2\venv\Scripts
Value of n? 17
Value of x? 100
S = 8.029008024644797
```

Рисунок 9.3 – Результат примера 3



```
C:\Users\Ксения\PycharmProjects\pythonProject2\venv\
Value of a? 89
x = 9.433981132056605
X = 9.433981132056603
```

Рисунок 9.4 – Результат примера 4



```
C:\Users\Ксения\PycharmProjects\pythonProject2\venv\Scripts\p
Value of x? 49
Ei(49.0) = 3.975442747903744e+19
```

Рисунок 9.5 – Результат примера 5

10. Для примеров 4 и 5 построила UML-диаграмму деятельности, используя веб-сервис Google

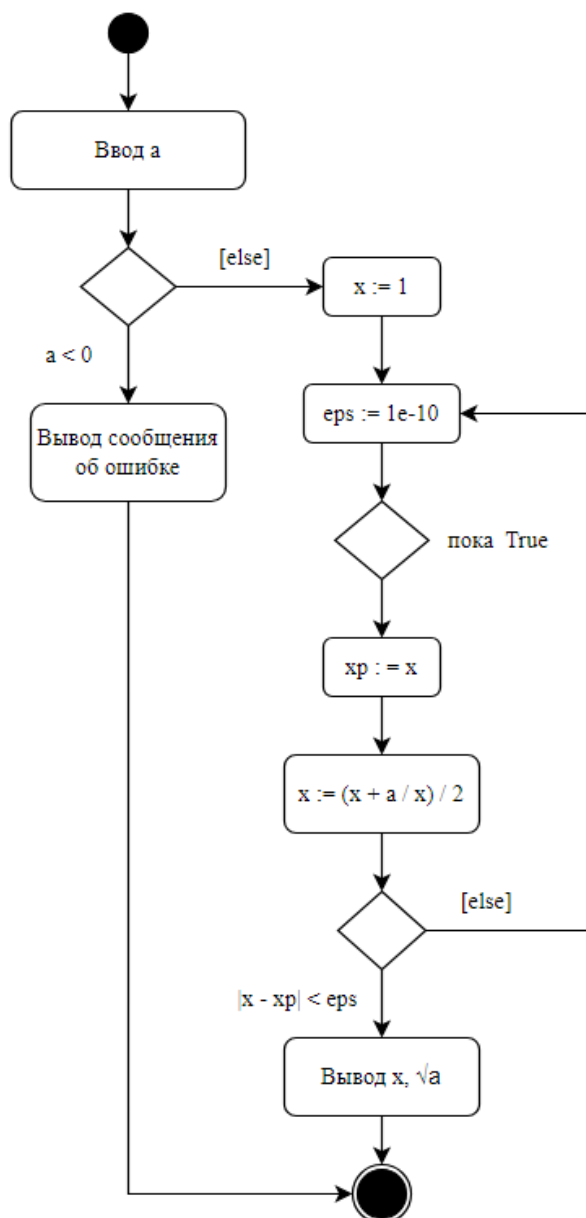


Рисунок 10.1 – UML-диаграмма деятельности для примера 4

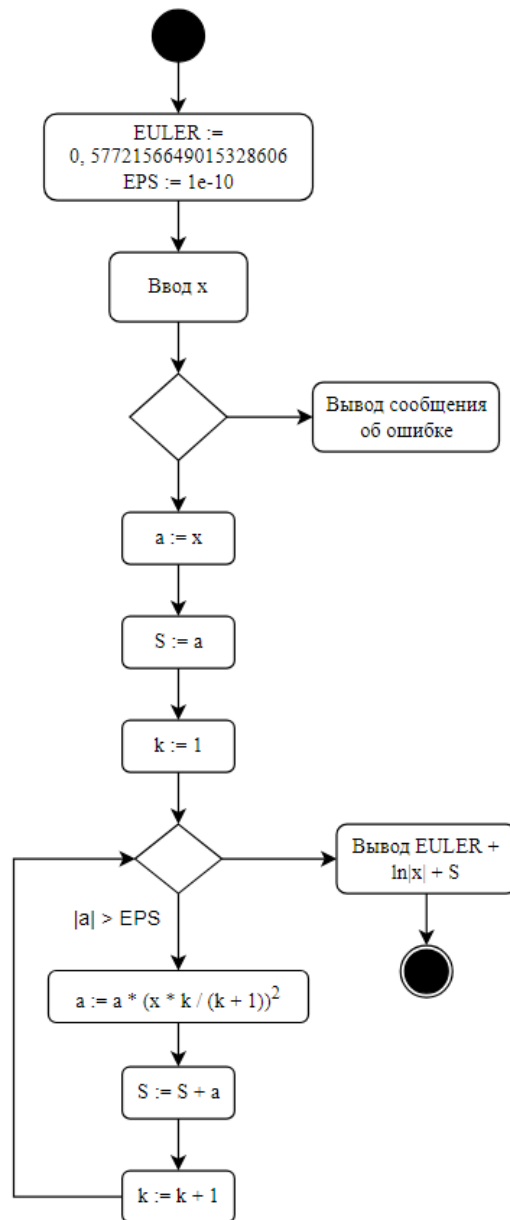


Рисунок 10.2 – UML-диаграмма деятельности для примера 5

11. Выполнила индивидуальные задания, согласно своему варианту

```
user.py arithmetic.py numbers.py ×
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  if __name__ == '__main__':
4      number_1 = int(input("Enter first number - "))
5      number_2 = int(input("Enter second number - "))
6      number_3 = int(input("Enter third number - "))
7      number_4 = int(input("Enter the fourth number - "))
8      sum_1_2 = number_1 + number_2
9      sum_3_4 = number_3 + number_4
10     division_result = sum_1_2 / sum_3_4
11     print("Result: %.2f" %
12           division_result)
13
```

Рисунок 11.1 – Код программы Task1.py в IDE PyCharm

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import sys
5
6  if __name__ == '__main__':
7      a = int(input("Введите значение a: "))
8      b = int(input("Введите значение b: "))
9      c = int(input("Введите значение c: "))
10
11     # Формула дискриминанта.
12     D = (b ** 2) - (4 * a * c)
13
14     if D > 0:
15         print(f"Первый x = {(-b + (D ** (1 / 2))) / (2 * a)}\nВторой x = {(-b - (D ** (1 / 2))) / (2 * a)}")
16     elif D == 0:
17         print(f"x равен {-b / (2 * a)}")
18     else:
19         print("Отсутствуют действительные корни", file=sys.stderr)
20
```

Рисунок 11.2 – Код программы Task2.py в IDE PyCharm

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import sys
5  import math
6
7  if __name__ == '__main__':
8      c = int(input("Введите число от -8 до 8: "))
9      result = ""
10
11     if math.fabs(c) < 9:
12
13         if c < 0:
14             result = "Минус "
15         if math.fabs(c) == 0:
16             result += "ноль"
17         elif math.fabs(c) == 1:
18             result += "один"
19         elif math.fabs(c) == 2:
20             result += "два"
21         elif math.fabs(c) == 3:
22             result += "три"
23         elif math.fabs(c) == 4:
24             result += "четыре"
25         elif math.fabs(c) == 5:
26             result += "пять"
27         elif math.fabs(c) == 6:
28             result += "шесть"
29         elif math.fabs(c) == 7:
30             result += "семь"
31         elif math.fabs(c) == 8:
32             result += "восемь"
33         print(result)
34     else:
35         print("Вы ввели неверное число", file=sys.stderr)
36

```

Рисунок 11.3 – Код программы Task3.py в IDE PyCharm

12. Привела в отчете скриншоты работы программ и UML-диаграммы деятельности решения индивидуальных заданий

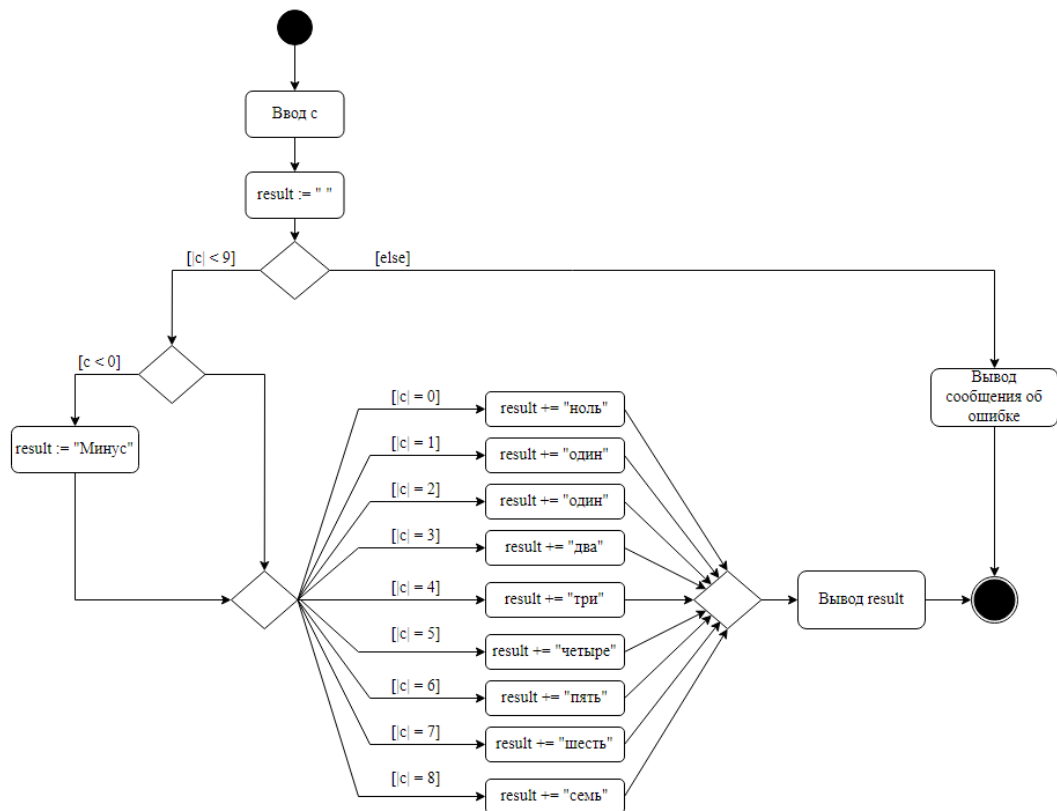


Рисунок 12.1 – UML-диаграмма деятельности для первого индивидуального задания

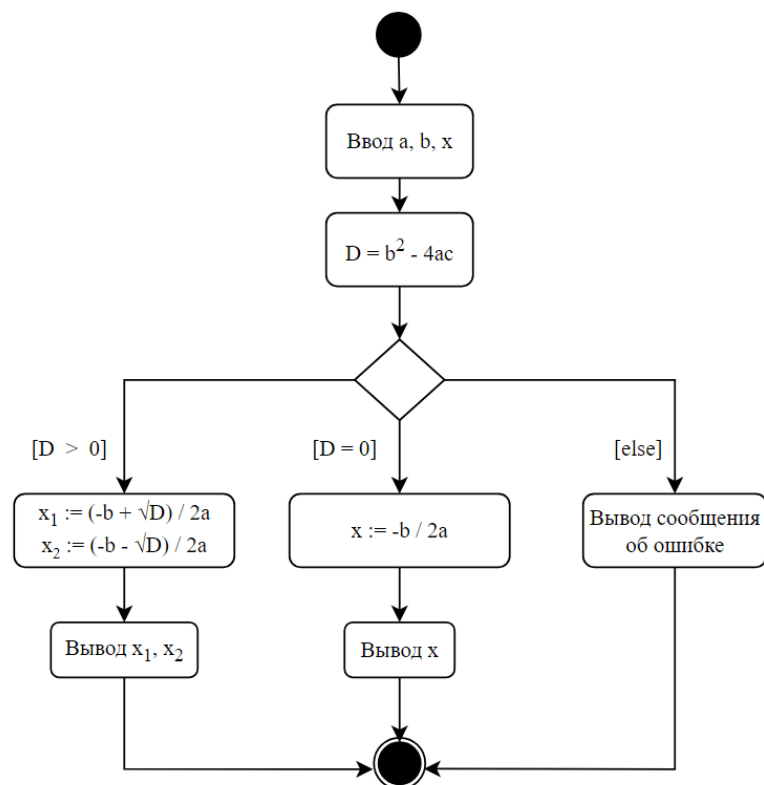


Рисунок 11.2 – UML-диаграмма деятельности для второго индивидуального задания

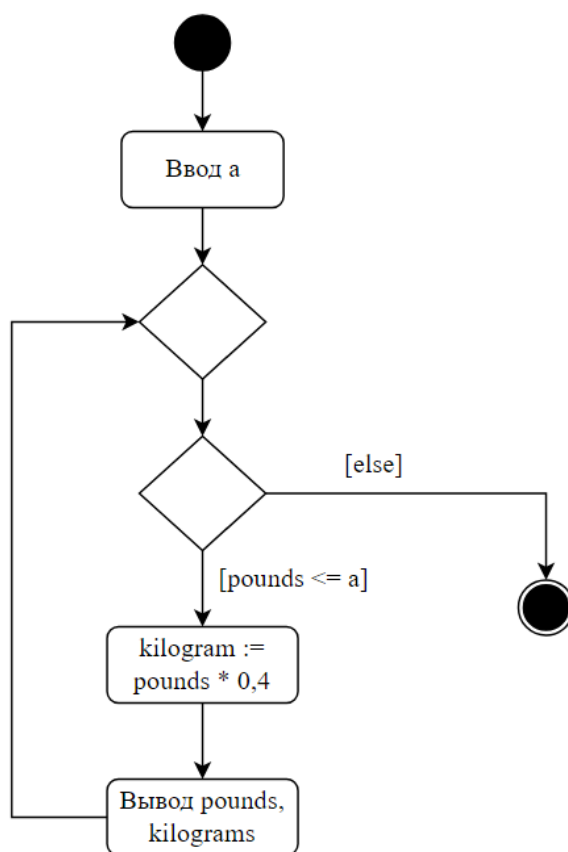


Рисунок 11.3 – UML-диаграмма деятельности для третьего индивидуального задания

13. Зафиксировала сделанные изменения в репозитории



```

C:\TestGit\EveryoneLovesPython>git add PyCharm

C:\TestGit\EveryoneLovesPython>git commit -m"add all tasks"
[develop 8d87e51] add all tasks
5 files changed, 94 insertions(+)
create mode 100644 PyCharm/AdvancedTasks/AdvancedTask.py
create mode 100644 "PyCharm/AdvancedTasks/\320\222\320\260\321\200\320\270\320\260\320\275\321\2026.txt"
create mode 100644 PyCharm/GeneralTasks/Task1.py
create mode 100644 PyCharm/GeneralTasks/Task2.py
create mode 100644 PyCharm/GeneralTasks/Task3.py

C:\TestGit\EveryoneLovesPython>git status
On branch develop
Your branch is ahead of 'origin/develop' by 2 commits.
(use "git push" to publish your local commits)

nothing to commit, working tree clean

```

Рисунок 13.1 – Коммит файлов в репозитории git

#### 14. Выполнила слияние ветки для разработки с веткой main / master

```

C:\TestGit\EveryoneLovesPython>git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

C:\TestGit\EveryoneLovesPython>git merge develop
Updating f8437c6..8d87e51
Fast-forward
 PyCharm/AdvancedTasks/AdvancedTask.py | 24 ++++++++
 ...0\321\200\320\270\320\260\320\275\321\2026.txt" | 0
 PyCharm/GeneralTasks/Task1.py | 37 ++++++++
 PyCharm/GeneralTasks/Task2.py | 21 ++++++++
 PyCharm/GeneralTasks/Task3.py | 12 ++++++
 PyCharm/SampleAssignments/example1.py | 15 ++++++
 PyCharm/SampleAssignments/example2.py | 18 ++++++++
 PyCharm/SampleAssignments/example3.py | 13 ++++++
 PyCharm/SampleAssignments/example4.py | 19 ++++++++
 PyCharm/SampleAssignments/example5.py | 28 ++++++++
10 files changed, 187 insertions(+)
create mode 100644 PyCharm/AdvancedTasks/AdvancedTask.py
create mode 100644 "PyCharm/AdvancedTasks/\320\222\320\260\321\200\320\270\320\260\320\275\321\2026.txt"
create mode 100644 PyCharm/GeneralTasks/Task1.py
create mode 100644 PyCharm/GeneralTasks/Task2.py
create mode 100644 PyCharm/GeneralTasks/Task3.py
create mode 100644 PyCharm/SampleAssignments/example1.py
create mode 100644 PyCharm/SampleAssignments/example2.py
create mode 100644 PyCharm/SampleAssignments/example3.py
create mode 100644 PyCharm/SampleAssignments/example4.py
create mode 100644 PyCharm/SampleAssignments/example5.py

```

Рисунок 14.1 – Слияние ветки main с веткой develop

#### 15. Отправила сделанные изменения на сервер GitHub

```

C:\TestGit\EveryoneLovesPython>git push
Enumerating objects: 20, done.
Counting objects: 100% (20/20), done.
Delta compression using up to 4 threads
Compressing objects: 100% (17/17), done.
Writing objects: 100% (18/18), 3.68 KiB | 1.84 MiB/s, done.
Total 18 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/MrFinicheck/EveryoneLovesPython.git
 f8437c6..8d87e51 main -> main

C:\TestGit\EveryoneLovesPython>

```

Рисунок 15.1 – Отправка изменений на сервер GitHub

## 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, отправке сигнала, создании или уничтожении объекта либо в простом вычислении — скажем, значения выражения

### 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`.