

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»

Выполнил:
Звездин Алексей Сергеевич
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 изучил теоретический материал работы

Лабораторная работа 2.2 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 работы

Диаграммы деятельности. Унифицированный язык моделирования (UML) является стандартным инструментом для создания «чертежей» программного обеспечения. 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. Несмотря на обилие выразительных возможностей, этот язык прост для понимания и использования.

Диаграммы деятельности - это один из пяти видов диаграмм, применяемых в UML для моделирования динамических аспектов поведения системы. Диаграмма деятельности - это, по существу, блок-схема, которая показывает, как поток управления переходит от одной деятельности к другой, однако, по сравнению с последней, у ней есть явные преимущества: поддержка многопоточности и объектно-ориентированного проектирования.

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, передаче сигнала, создании или уничтожении объекта либо в простом вычислении - скажем, значения выражения. Графически диаграмма деятельности представляется в виде графа, имеющего вершины и ребра.

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. Как показано на рис. 4.1, состояния действия изображаются прямоугольниками с закругленными краями. 1 из 26

Рисунок 1.1 – Изучение материала для лабораторной работы

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

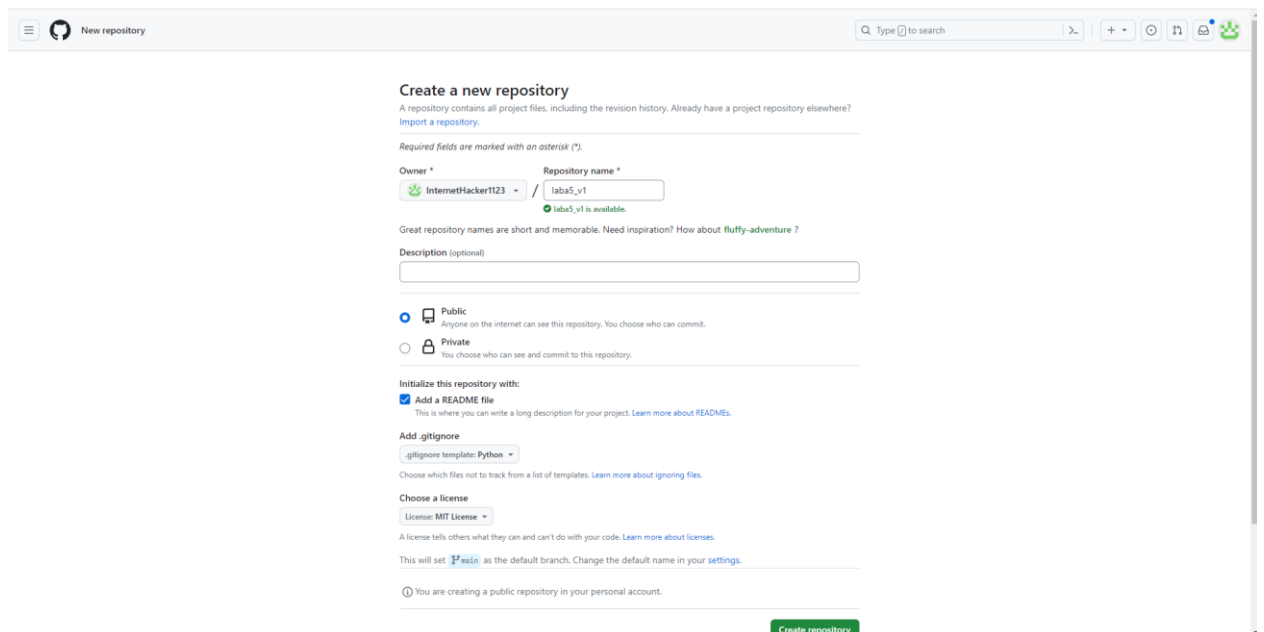


Рисунок 2.1 – Настройка репозитория

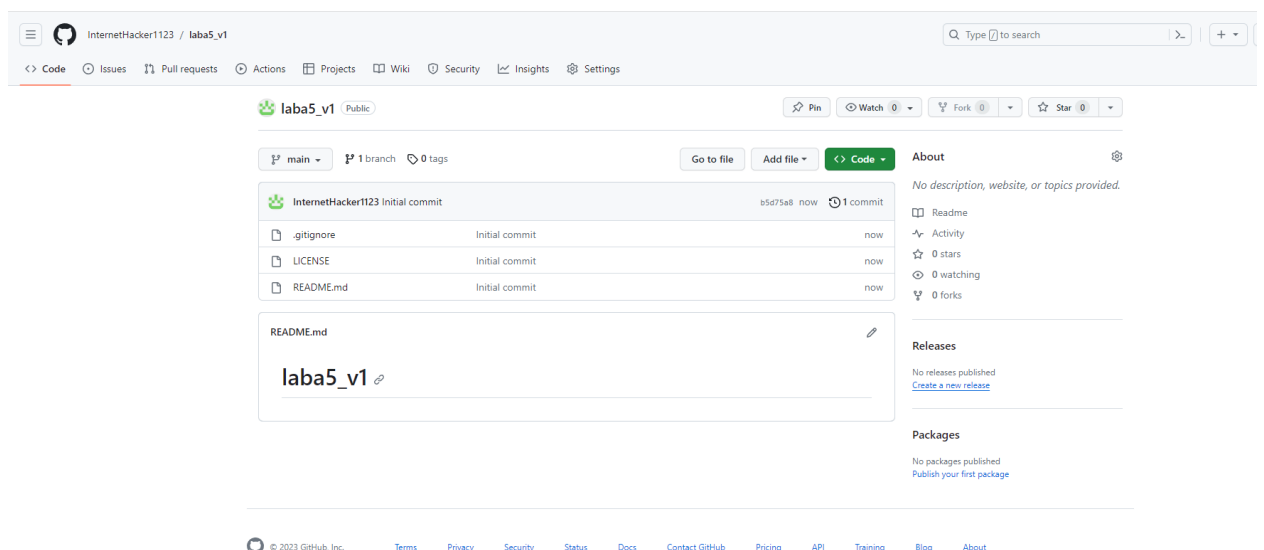
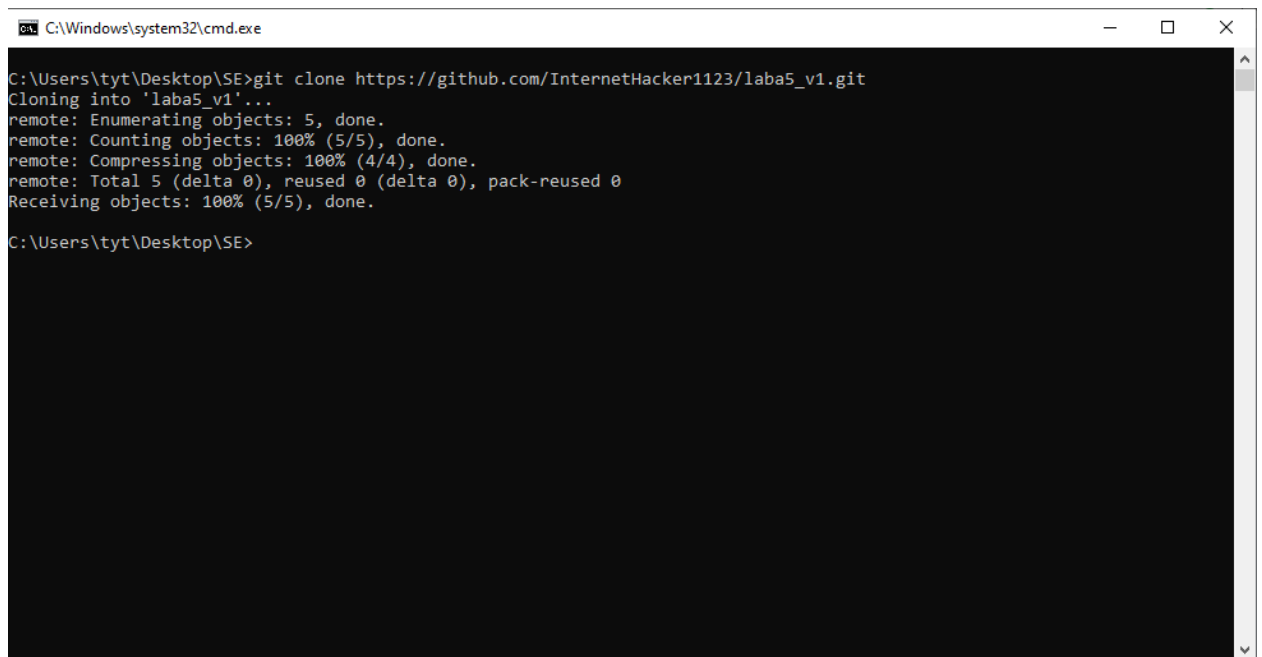


Рисунок 2.2 – Готовый репозиторий

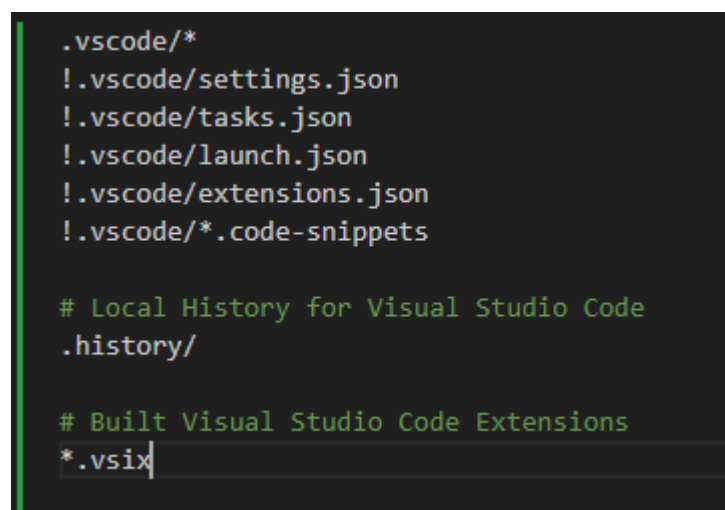
3. Выполняю клонирование созданного репозитория



```
C:\Windows\system32\cmd.exe
C:\Users\tyt\Desktop\SE>git clone https://github.com/InternetHacker1123/laba5_v1.git
Cloning into 'laba5_v1'...
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.
C:\Users\tyt\Desktop\SE>
```

Рисунок 3.1 – Клонирование репозитория на локальный диск

4. Дополнил файл .gitignore необходимыми правилами для работы с VScode



```
.vscode/*
!.vscode/settings.json
!.vscode/tasks.json
!.vscode/launch.json
!.vscode/extensions.json
!.vscode/*.code-snippets

# Local History for Visual Studio Code
.history/

# Built Visual Studio Code Extensions
*.vsix
```

Рисунок 4.1 – .gitignore для VScode

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

```
C:\Windows\system32\cmd.exe

C:\Users\tyt\Desktop\SE\laba5_v1>git branch develop
C:\Users\tyt\Desktop\SE\laba5_v1>git checkout develop
Switched to branch 'develop'

C:\Users\tyt\Desktop\SE\laba5_v1>git branch
* develop
  main

C:\Users\tyt\Desktop\SE\laba5_v1>git push -u origin develop
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:   https://github.com/InternetHacker1123/laba5_v1/pull/new/develop
remote:
To https://github.com/InternetHacker1123/laba5_v1.git
 * [new branch]      develop -> develop
branch 'develop' set up to track 'origin/develop'.
```

Рисунок 5.1 – Создание ветки develop от ветки main

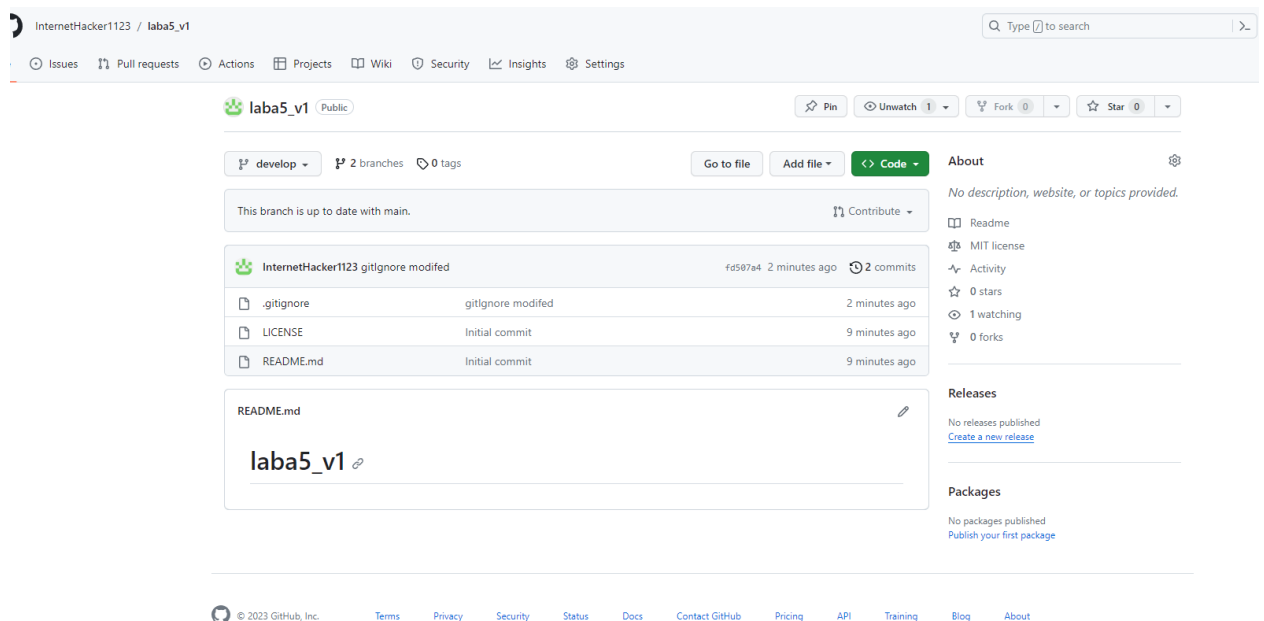


Рисунок 5.2 – Ветка develop на GitHub

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

Внешний вид кода

Отступы

Используйте 4 пробела на каждый уровень отступа.

Продолжительные строки должны выравнивать обернутые элементы либо вертикально, используя неявную линию в скобках (круглых, квадратных или фигурных), либо с использованием висячего отступа. При использовании висячего отступа следует применять следующие соображения: на первой линии не должно быть аргументов, а остальные строки должны четко восприниматься как продолжение линии.

Правильно:

```
# Выровнено по открывающему разделителю
foo = long_function_name(var_one, var_two,
                          var_three, var_four)

# Больше отступов включено для отличия его от остальных
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
```

Неправильно:

```
# Аргументы на первой линии запрещены, если не используется вертикальное выравнивание
foo = long_function_name(var_one, var_two,
                          var_three, var_four)

# Больше отступов требуется, для отличия его от остальных
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
```

Рисунок 6.1 – Изучение Python PEP-8

7. Создал проект PyCharm в папке репозитория

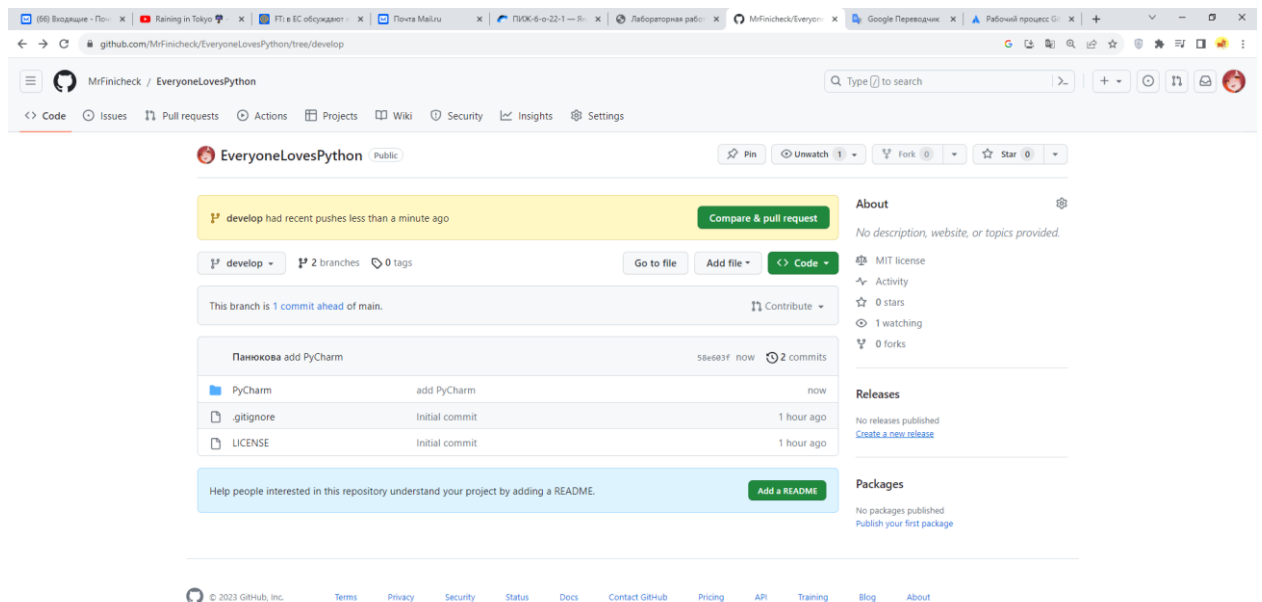
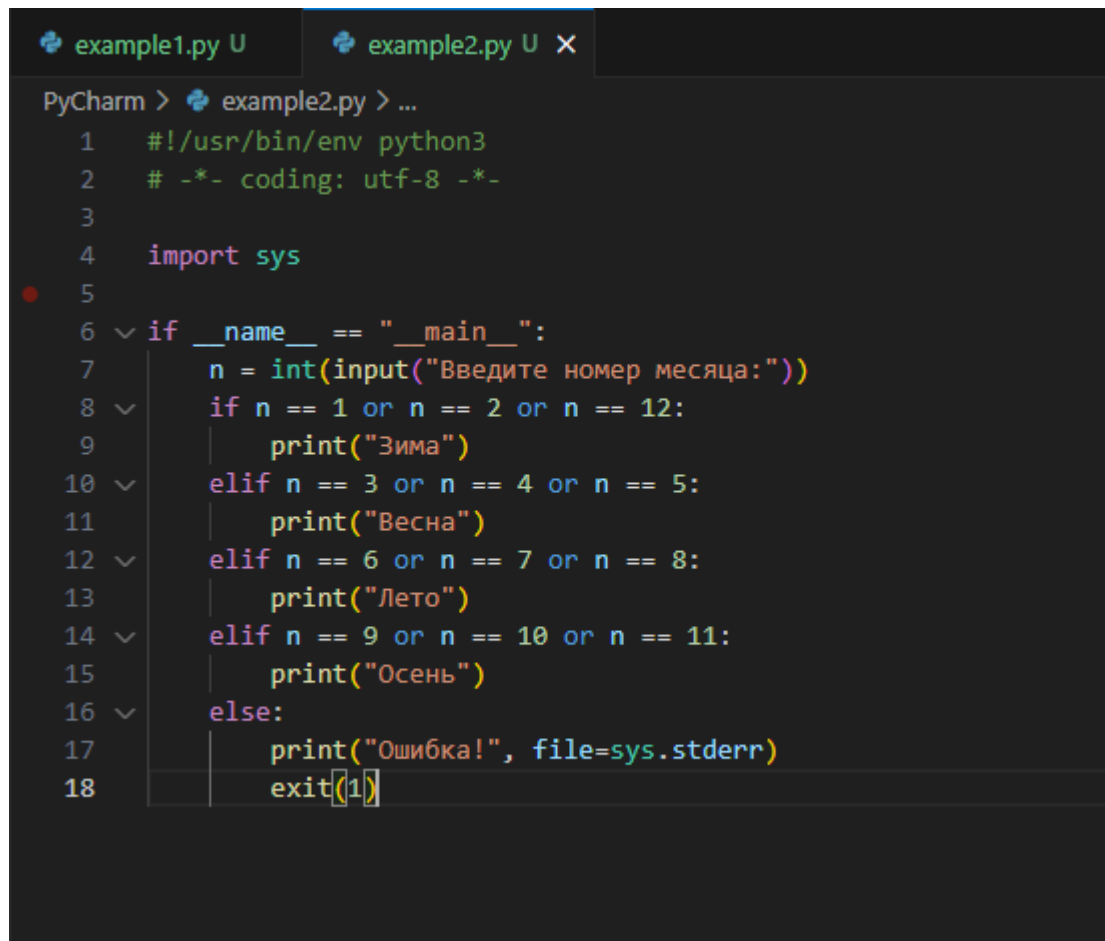


Рисунок 7.1 – Репозиторий с проектом PyCharm

8. Проработал примеры лабораторной работы. Создал для каждого примера отдельный модуль языка Python. Зафиксировал изменения в репозитории

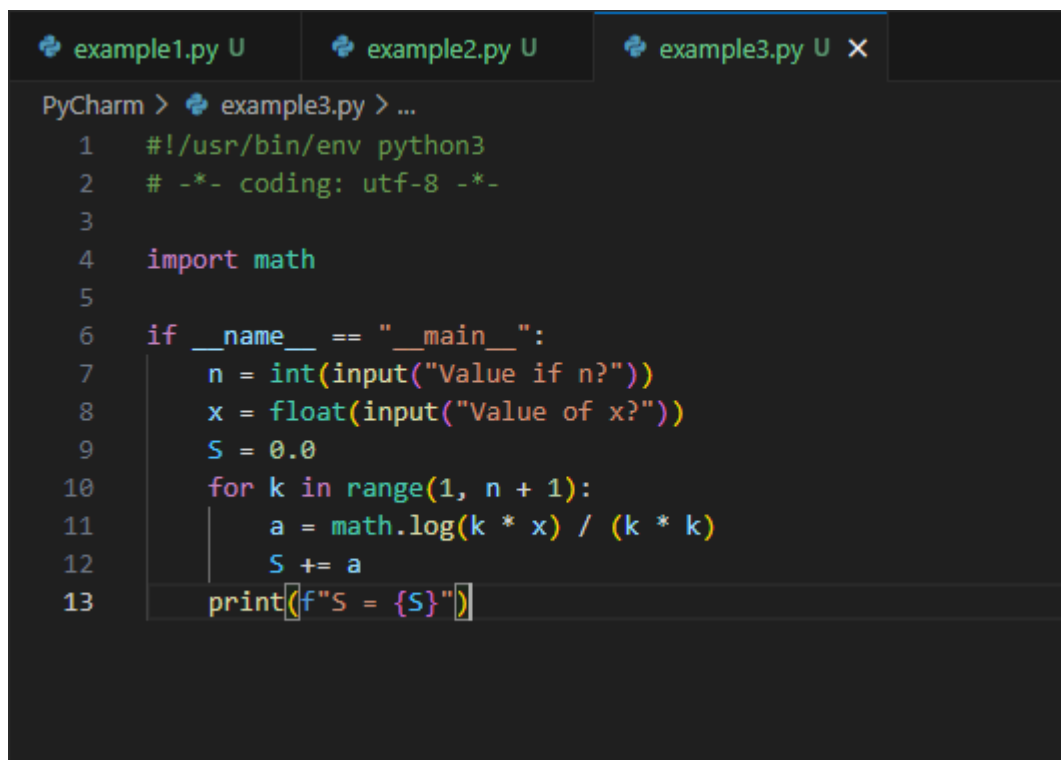
```
example1.py U X
PyCharm > example1.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import math
5
6  if __name__ == "__main__":
7      x = float(input("Value of x? "))
8      if x <= 0:
9          y = 2 * x * x + math.cos(x)
10     elif x < 5:
11         y = x + 1
12     else:
13         y = math.sin(x) - x * x
14     print(f"y = {y}")
```

Рисунок 8.1 – Проработка примера 1



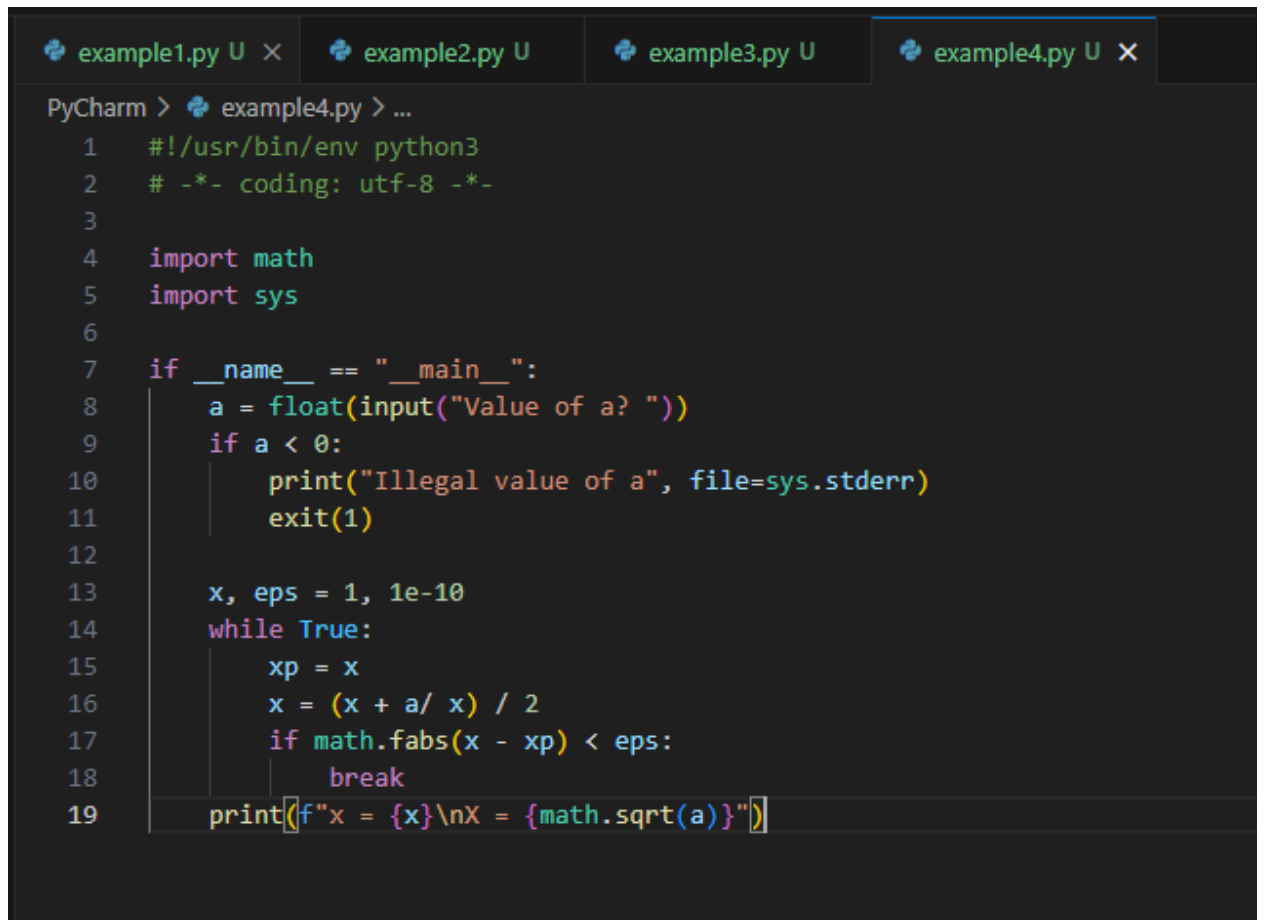
```
example1.py U example2.py U X
PyCharm > example2.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import sys
5
6  if __name__ == "__main__":
7      n = int(input("Введите номер месяца:"))
8      if n == 1 or n == 2 or n == 12:
9          print("Зима")
10     elif n == 3 or n == 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



```
example1.py U example2.py U example3.py U X
PyCharm > example3.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import math
5
6  if __name__ == "__main__":
7      n = int(input("Value if 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



```
PyCharm > example4.py U X
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

```

4 import math
5 import sys
6
7 # Постоянная Эйлера.
8 EULER = 0.5772156649015329
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
19     a = x
20     S, k = a, 1
21
22
23     # Найти сумму членов ряда.
24     while math.fabs(a) < EPS:
25         a *= x * k / (k + 1) ** 2
26         S += a
27         k += 1
28
29     # Вывести значение функции.
30     print(f"Ei({x}) = {EULER + math.log(math.fabs(x)) + S}")
31

```

Рисунок 8.5 – Проработка примера 5

	Имя	Дата изменения	Тип	Размер
Доступ стол ты ения Desktop webgl	example1.py	24.10.2023 23:44	Исходный файл ...	1 КБ
	example2.py	24.10.2023 23:57	Исходный файл ...	1 КБ
	example3.py	25.10.2023 0:14	Исходный файл ...	1 КБ
	example4.py	25.10.2023 0:24	Исходный файл ...	1 КБ
	example5.py	25.10.2023 0:31	Исходный файл ...	1 КБ

Рисунок 8.6 – Создание отдельных модулей для каждого из примеров

```
C:\Windows\system32\cmd.exe

C:\Users\tyt\Desktop\SE\laba5_v1>git commit -m"add examples"
[develop 798ff3d] add examples
6 files changed, 101 insertions(+)
create mode 100644 .vscode/settings.json
create mode 100644 PyCharm/example1.py
create mode 100644 PyCharm/example2.py
create mode 100644 PyCharm/example3.py
create mode 100644 PyCharm/example4.py
create mode 100644 PyCharm/example5.py

C:\Users\tyt\Desktop\SE\laba5_v1>
```

git

Рисунок 8.7 – Фиксирование изменений в репозитории

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

```
PS C:\Users\tyt\Desktop\SE\laba5_v1> & C:/Users/tyt/AppData/Local/Programs/Python/Python311/python.exe c:/Users/tyt/Desktop/SE/laba5_v1/PyCharm/example1.py
Value of x? 10
y = -100.54402111088937
PS C:\Users\tyt\Desktop\SE\laba5_v1>
```

Рисунок 9.1 – Результат примера 1

```
PS C:\Users\tyt\Desktop\SE\laba5_v1> & C:/Users/tyt/AppData/Local/Programs/Python/Python311/python.exe c:/Users/tyt/Desktop/SE/laba5_v1/PyCharm/example2.py
Введите номер месяца:5
Весна
PS C:\Users\tyt\Desktop\SE\laba5_v1>
```

Рисунок 9.2 – Результат примера 2

```
PS C:\Users\tyt\Desktop\SE\laba5_v1> & C:/Users/tyt/AppData/Local/Programs/Python/Python311/python.exe c:/Users/tyt/Desktop/SE/laba5_v1/PyCharm/example3.py
Value if n?10
Value of x?20
S = 5.26119185280723
PS C:\Users\tyt\Desktop\SE\laba5_v1>
```

Рисунок 9.3 – Результат примера 3

```
PS C:\Users\tyt\Desktop\SE\laba5_v1> & C:/Users/tyt/AppData/Local/Programs/Python/Python311/python.exe c:/Users/tyt/Desktop/SE/laba5_v1/PyCharm/example4.py
Value of a? 3
x = 1.7320508075688772
X = 1.7320508075688772
PS C:\Users\tyt\Desktop\SE\laba5_v1> █
```

Рисунок 9.4 – Результат примера 4

```
PS C:\Users\tyt\Desktop\SE\laba5_v1> & C:/Users/tyt/AppData/Local/Programs/Python/Python311/python.exe c:/Users/tyt/Desktop/SE/laba5_v1/PyCharm/example5.py
Value of x?10
Ei(10.0) = 12.879800757895579
PS C:\Users\tyt\Desktop\SE\laba5_v1> █
```

Рисунок 9.5 – Результат примера 5

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

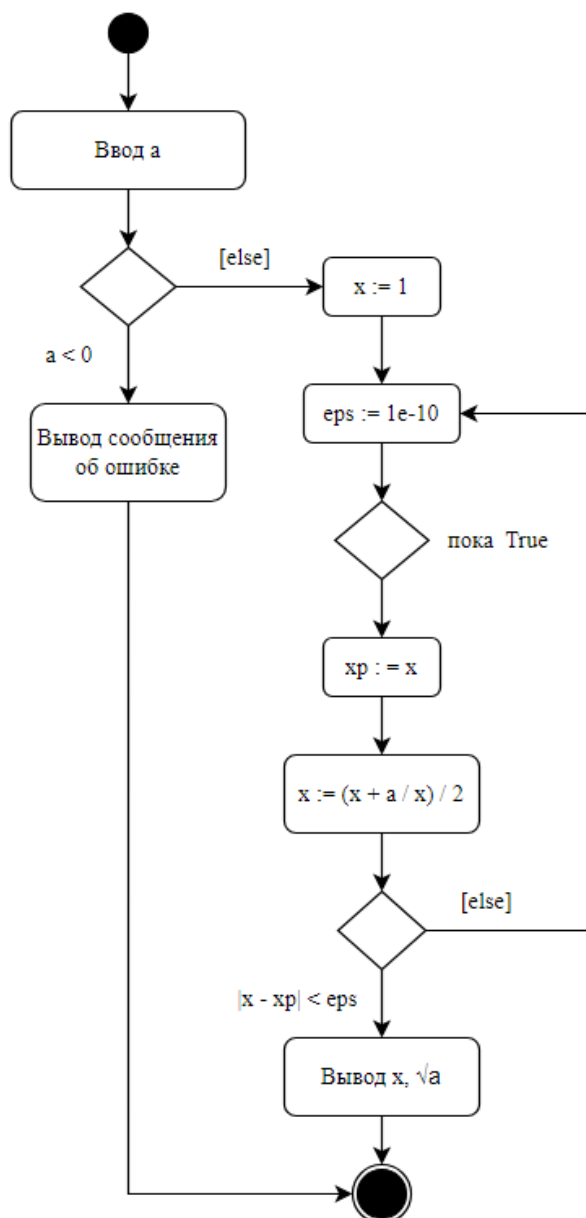


Рисунок 10.1 – UML-диаграмма деятельности для примера 4

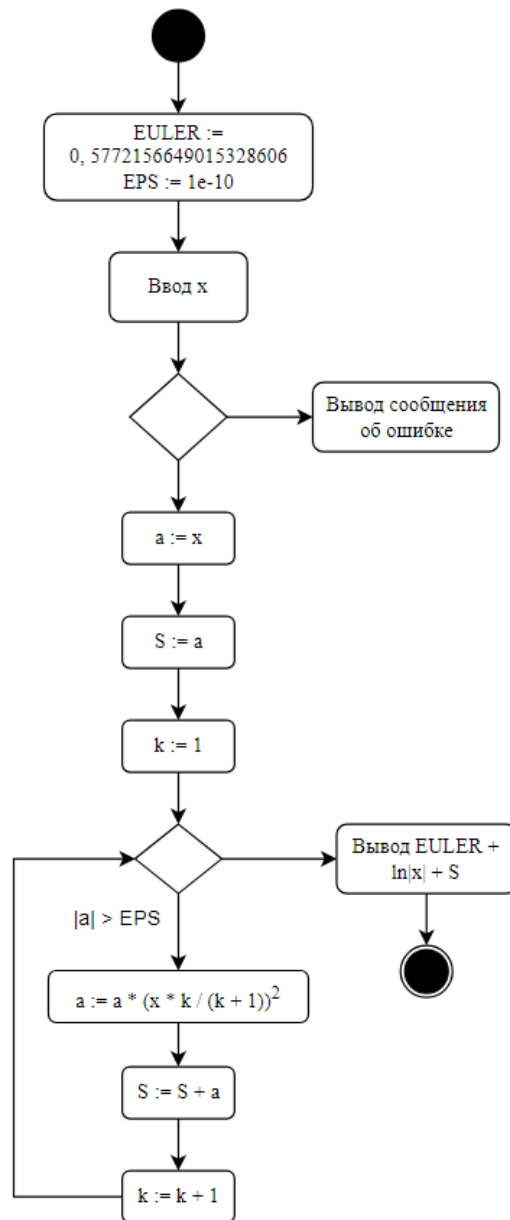


Рисунок 10.2 – UML-диаграмма деятельности для примера 5

11. Выполнил индивидуальные задания, согласно своему варианту

```

PyCharm > individual1.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import sys
5
6  if __name__ == '__main__':
7      num = int(input("Vvedite chislo"))
8      if num <= 0 and num >= 12:
9          print()
10         days = 0
11         polyg = 0
12         if num in [4, 6, 9, 11]:
13             days = 30
14         elif num == 2:
15             days = 28
16         else:
17             days = 31
18
19         if num <= 6:
20             polyg = 1
21         else:
22             polyg = 2
23         print(f"polygodie = {polyg}\nkol-vo dney = {days}")
24

```

Рисунок 11.1 – Код программы individual1.py в IDE PyCharm

```

PyCharm > individual2.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  if __name__ == '__main__':
5      a = 10
6      D = 9 + 4*a
7      if D > 0:
8          x1 = (5 + D**0.5) / 2
9          x2 = (5 - D**0.5) / 2
10         print(f"Для a = {a} неравенство имеет два корня: x1 = {x1} и x2 = {x2}")
11     elif D == 0:
12         x = 5 / 2
13         print(f"Для a = {a} неравенство имеет один корень: x = {x}")
14     else:
15         print(f"Для a = {a} неравенство не имеет корней")
16

```

Рисунок 11.2 – Код программы individual 2.py в IDE PyCharm

```
PyCharm > individual3.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  if __name__ == '__main__':
5      for i in range(100, 1000):
6          i = str(i)
7          sum = int(i[0]) + int(i[1]) + int(i[2])
8          if sum % 7 == 0 and int(i) % 7 == 0:
9              print(i)
10         else:
11             continue
```

Рисунок 11.3 – Код программы individual 3.py в IDE PyCharm

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

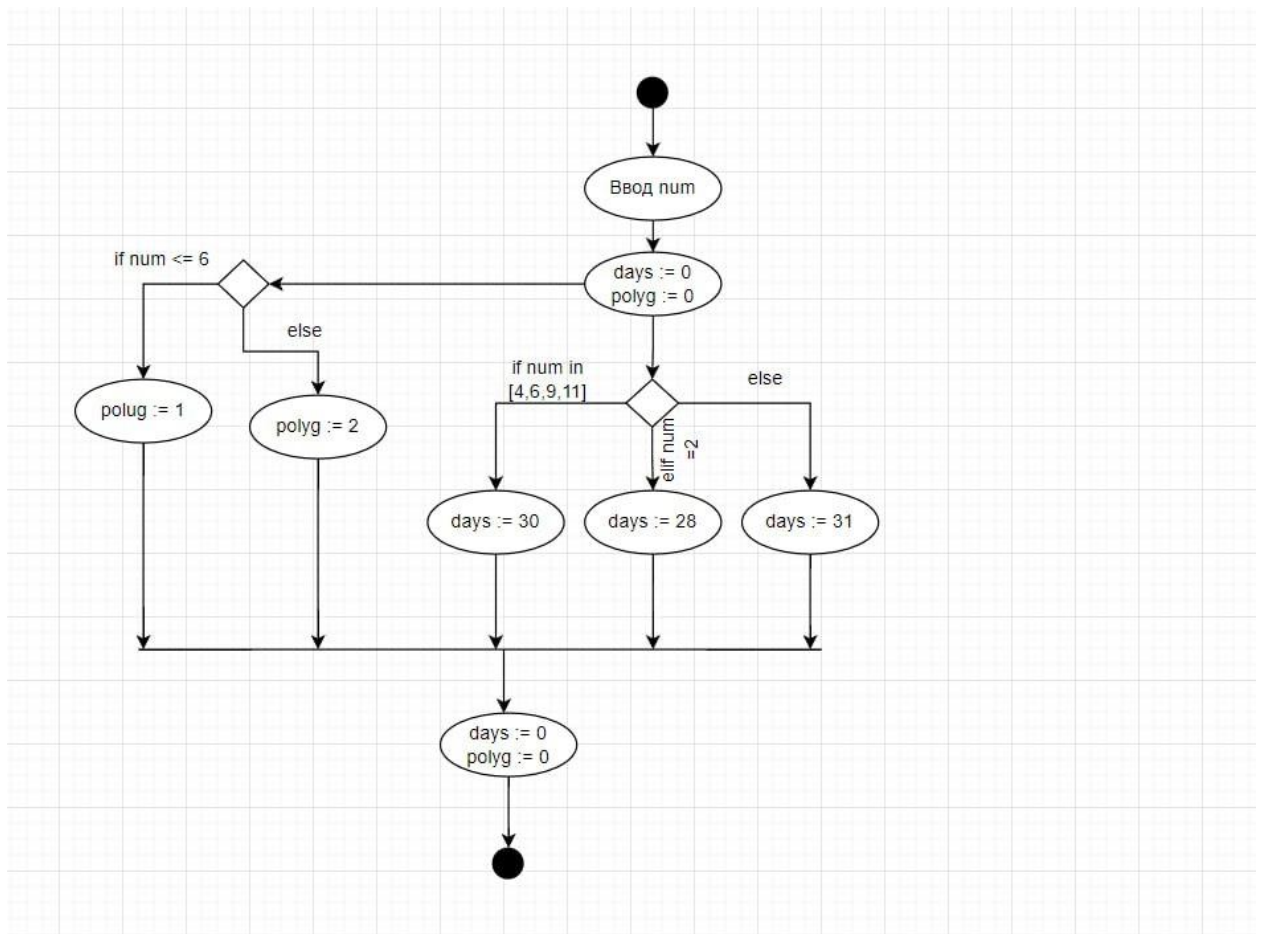


Рисунок 12.1 – UML-диаграмма деятельности для первого индивидуального задания

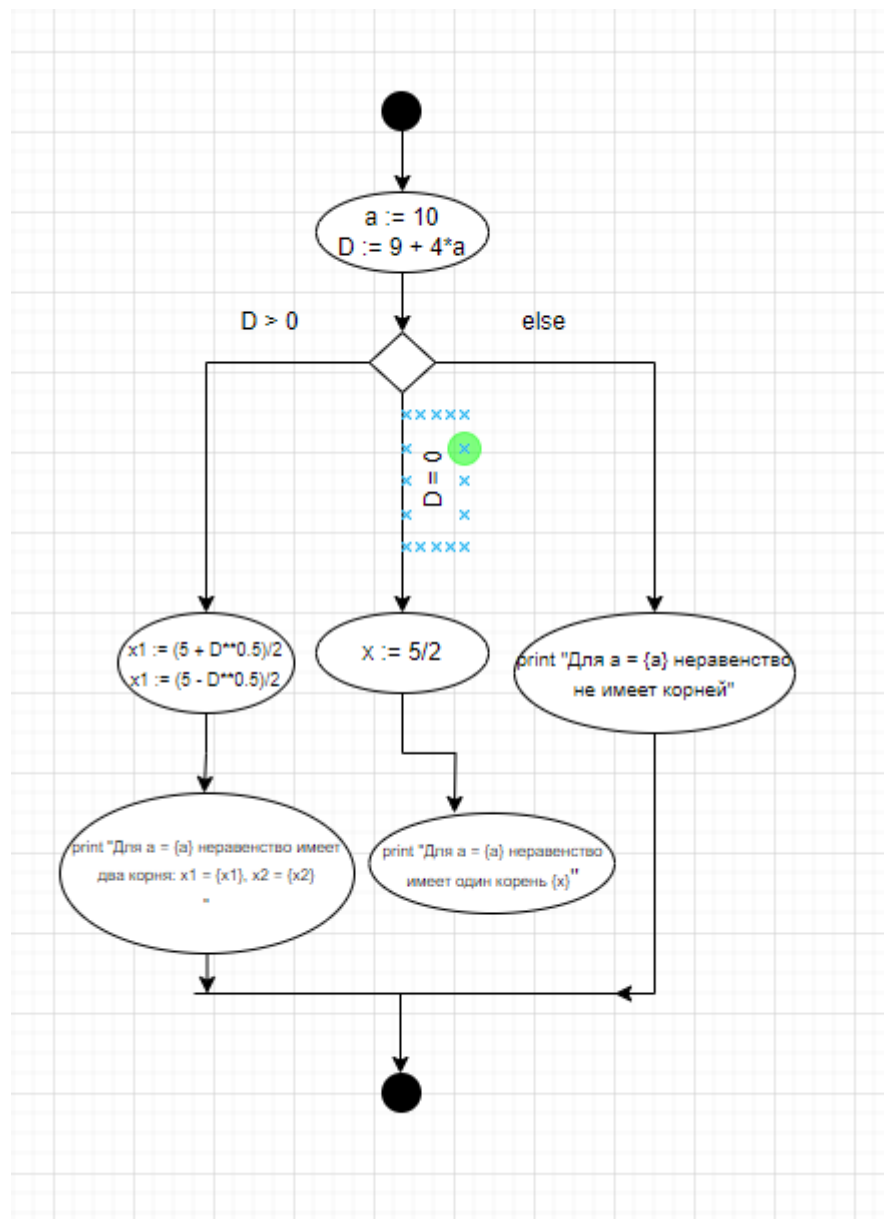


Рисунок 11.2 – UML-диаграмма деятельности для второго индивидуального задания

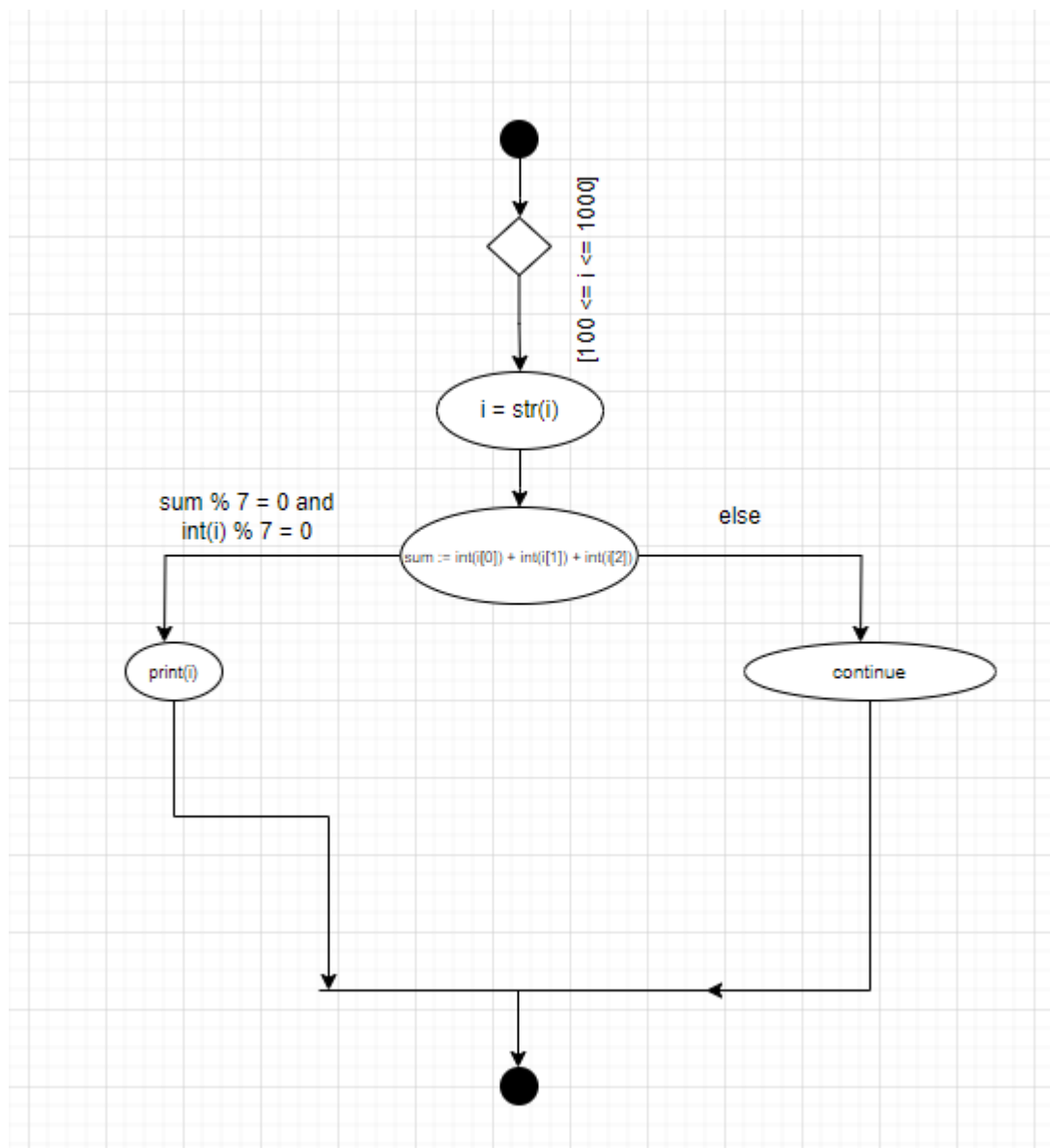


Рисунок 11.3 – UML-диаграмма деятельности для третьего индивидуального задания

13. Зафиксировал сделанные изменения в репозитории

```
C:\Windows\system32\cmd.exe

C:\Users\tyt\Desktop\SE\laba5_v1>code .

C:\Users\tyt\Desktop\SE\laba5_v1>git add .
warning: in the working copy of 'exercise2.drawio', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'exercise3.drawio', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'exerise1.drawio', LF will be replaced by CRLF the next time Git touches it

C:\Users\tyt\Desktop\SE\laba5_v1>git commit -m"all add"
[develop acc90c4] all add
 6 files changed, 349 insertions(+)
 create mode 100644 PyCharm/individual1.py
 create mode 100644 PyCharm/individual2.py
 create mode 100644 PyCharm/individual3.py
 create mode 100644 exercise2.drawio
 create mode 100644 exercise3.drawio
 create mode 100644 exerise1.drawio

C:\Users\tyt\Desktop\SE\laba5_v1>
```

Рисунок 13.1 – Коммит файлов в репозитории git

14. Выполнил слияние ветки для разработки с веткой main / master

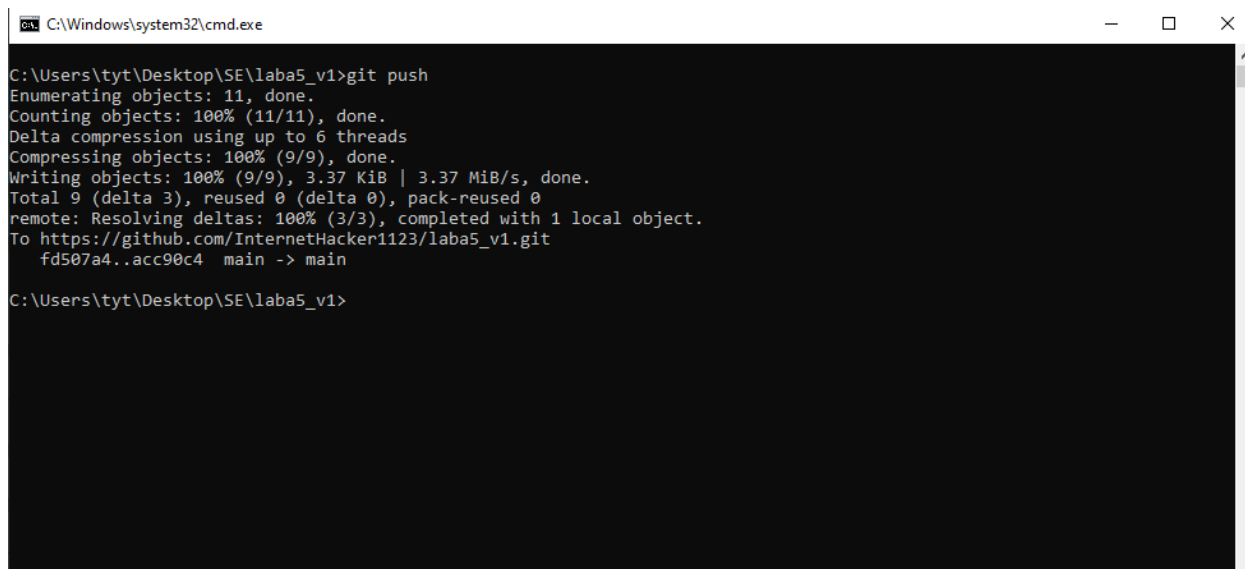
```
C:\Windows\system32\cmd.exe

C:\Users\tyt\Desktop\SE\laba5_v1>git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

C:\Users\tyt\Desktop\SE\laba5_v1>git merge develop
Updating fd507a4..acc90c4
Fast-forward
 .vscode/settings.json | 6 +++
 PyCharm/example1.py | 14 ++++++
 PyCharm/example2.py | 18 ++++++++
 PyCharm/example3.py | 13 ++++++
 PyCharm/example4.py | 19 ++++++++
 PyCharm/example5.py | 31 ++++++++
 PyCharm/individual1.py | 23 ++++++++
 PyCharm/individual2.py | 15 ++++++
 PyCharm/individual3.py | 11 ++++++
 exercise2.drawio | 100 ++++++++
 exercise3.drawio | 100 ++++++++
 exerise1.drawio | 100 ++++++++
12 files changed, 450 insertions(+)
 create mode 100644 .vscode/settings.json
 create mode 100644 PyCharm/example1.py
 create mode 100644 PyCharm/example2.py
 create mode 100644 PyCharm/example3.py
 create mode 100644 PyCharm/example4.py
 create mode 100644 PyCharm/example5.py
 create mode 100644 PyCharm/individual1.py
 create mode 100644 PyCharm/individual2.py
 create mode 100644 PyCharm/individual3.py
```

Рисунок 14.1 – Слияние ветки main с веткой develop

15. Отправил сделанные изменения на сервер GitHub



```
C:\Windows\system32\cmd.exe
C:\Users\tyt\Desktop\SE\laba5_v1>git push
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 6 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 3.37 KiB | 3.37 MiB/s, done.
Total 9 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 1 local object.
To https://github.com/InternetHacker1123/laba5_v1.git
   fd507a4..acc90c4  main -> main
C:\Users\tyt\Desktop\SE\laba5_v1>
```

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, послылке 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`.