

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

Выполнил:

Джараян Арег Александрович  
2 курс, группа ПИЖ-б-о-22-1,  
09.03.04 «Программная инженерия»,  
направленность (профиль) «Разработка и  
сопровождение программного  
обеспечения», очная форма обучения

\_\_\_\_\_  
(подпись)

Проверил Воронкин Роман Александрович

\_\_\_\_\_  
(подпись)

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

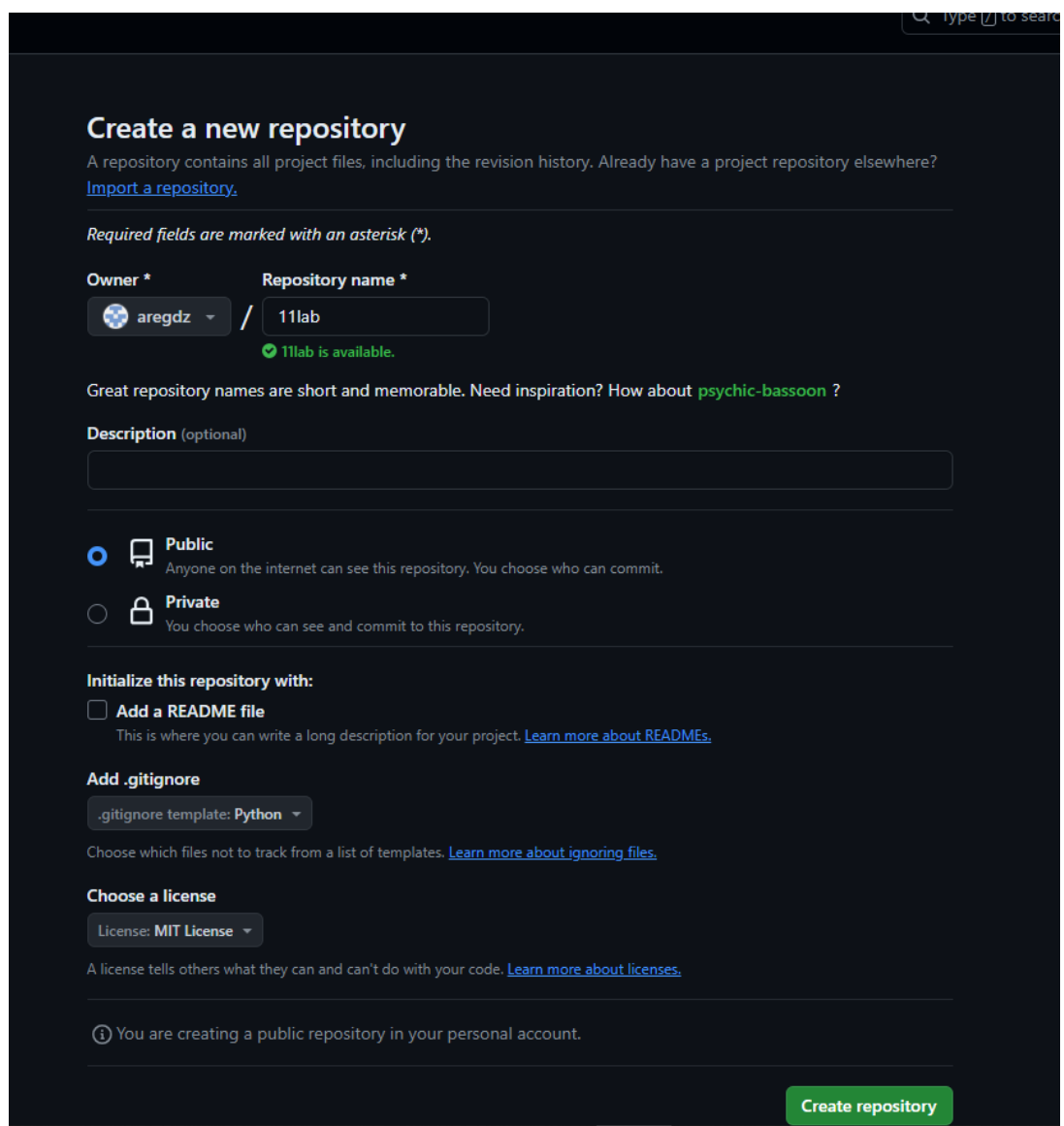
Ставрополь, 2024 г.

**Тема:** Лабораторная работа 4.2 Наследование и полиморфизм в языке Python.

**Цель работы:** приобретение навыков по созданию иерархии классов при написании программ с помощью языка программирования Python версии 3.x.

Ход работы.

1. Создание нового репозитория с лицензией MIT.



**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 \*** aregdz / **Repository name \*** 11lab  
✔ 11lab is available.

Great repository names are short and memorable. Need inspiration? How about [psychic-bassoon](#) ?

**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 1 – создание репозитория

2. Клонировал репозиторий на рабочий ПК.

```
aregd@DESKTOP-5KV9QA9 MINGW64 /d/Рабочий стол/4 семестр/опи/10/10lab (dev)
$ cd "D:\Рабочий стол\4 семестр\опи\11"

aregd@DESKTOP-5KV9QA9 MINGW64 /d/Рабочий стол/4 семестр/опи/11
$ git clone https://github.com/aregdz/11lab.git
Cloning into '11lab'...
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 /d/Рабочий стол/4 семестр/опи/11
$ |
```

Рисунок 2 – клонирование репозитория

3. Дополнил файл .gitignore необходимыми инструкциями.

```
1  # Byte-compiled / optimized / DLL files
2  __pycache__/
3  *.py[cod]
4  *$py.class
5
6  # C extensions
7  *.so
8
9  # Distribution / packaging
10 .Python
11 build/
12 develop-eggs/
13 dist/
14 downloads/
15 eggs/
16 .eggs/
17 lib/
18 lib64/
19 parts/
20 sdist/
21 var/
22 wheels/
23 share/python-wheels/
```

Рисунок 4 – Файл .gitignore

```
aregd@DESKTOP-5KV9QA9 MINGW64 /d/Рабочий стол/4 семестр/опи/11
$ cd 11lab

aregd@DESKTOP-5KV9QA9 MINGW64 /d/Рабочий стол/4 семестр/опи/11/11lab (main)
$ git checkout -b develop
Switched to a new branch 'develop'

aregd@DESKTOP-5KV9QA9 MINGW64 /d/Рабочий стол/4 семестр/опи/11/11lab (develop)
$
```

Рисунок 4 – организация ветки

```
(venv) PS D:\Рабочий стол\4 семестр\опи\11\11lab
Package      Version
-----
black        24.4.0
cfgv         3.4.0
click        8.1.7
colorama     0.4.6
distlib      0.3.8
pyflakes     3.2.0
PyYAML       6.0.1
setuptools   69.5.1
virtualenv   20.25.2
```

Рисунок 5 – создание виртуального окружения

#### 4.Пример 1.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

class Rational:
    def __init__(self, a=0, b=1):
        a = int(a)
        b = int(b)
        if b == 0:
            raise ValueError("Denominator cannot be zero.")
        self.__numerator = abs(a)
        self.__denominator = abs(b)
        self.__reduce()

    # Сокращение дроби
    def __reduce(self):
        # Функция для нахождения наибольшего общего делителя
```

```

def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a
c = gcd(self.__numerator, self.__denominator)
self.__numerator //= c
self.__denominator //= c

@property
def numerator(self):
    return self.__numerator

@property
def denominator(self):
    return self.__denominator

# Прочитать значение дроби с клавиатуры. Дробь вводится как a/b.
def read(self, prompt=None):
    line = input(prompt) if prompt is not None else input()
    parts = list(map(int, line.split('/', maxsplit=1)))
    if parts[1] == 0:
        raise ValueError("Denominator cannot be zero.")
    self.__numerator = abs(parts[0])
    self.__denominator = abs(parts[1])
    self.__reduce()

# Вывести дробь на экран
def display(self):
    print(f"{self.__numerator}/{self.__denominator}")

# Сложение обыкновенных дробей.
def add(self, rhs):
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator + \
            self.denominator * rhs.numerator
        b = self.denominator * rhs.denominator
        return Rational(a, b)
    else:
        raise ValueError("Operand must be a Rational.")

# Вычитание обыкновенных дробей
def sub(self, rhs):
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator - \
            self.denominator * rhs.numerator
        b = self.denominator * rhs.denominator
        return Rational(a, b)
    else:
        raise ValueError("Operand must be a Rational.")

# Умножение обыкновенных дробей.
def mul(self, rhs):
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.numerator
        b = self.denominator * rhs.denominator
        return Rational(a, b)
    else:
        raise ValueError("Operand must be a Rational.")

# Деление обыкновенных дробей.
def div(self, rhs):
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator
        b = self.denominator * rhs.numerator
        if b == 0:
            raise ValueError("Denominator cannot be zero.")
        return Rational(a, b)
    else:
        raise ValueError("Operand must be a Rational.")

# Отношение обыкновенных дробей.

```

```

def equals(self, rhs):
    return self.numerator == rhs.numerator and \
           self.denominator == rhs.denominator

def greater(self, rhs):
    return self.numerator / self.denominator > \
           rhs.numerator / rhs.denominator

def less(self, rhs):
    return self.numerator / self.denominator < \
           rhs.numerator / rhs.denominator

if __name__ == '__main__':
    r1 = Rational(3, 4)
    r1.display()
    r2 = Rational()
    r2.read("Введите обыкновенную дробь: ")
    r2.display()
    r3 = r2.add(r1)
    r3.display()
    r4 = r2.sub(r1)
    r4.display()
    r5 = r2.mul(r1)
    r5.display()
    r6 = r2.div(r1)
    r6.display()

```

Рисунок 6 – пример 1

```

"D:\Рабочий стол\4 семестр\опи\11\11lab\venv\Scripts\python.exe"
3/4
Введите обыкновенную дробь: 4/8
1/2
5/4
1/4
3/8
2/3

Process finished with exit code 0

```

Рисунок 7 – выполнение примера 1

## 5.Пример 2.

```

# Python program showing
# abstract base class work

```

```

from abc import ABC, abstractmethod

class Polygon(ABC):
    @abstractmethod
    def noofsides(self):
        pass

class Triangle(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 3 sides")

class Pentagon(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 5 sides")

class Hexagon(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 6 sides")

class Quadrilateral(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 4 sides")

# Driver code
R = Triangle()
R.noofsides()

K = Quadrilateral()
K.noofsides()

R = Pentagon()
R.noofsides()

# Пример 3:
K = Hexagon()
K.noofsides()

```

Рисунок 8 - пример 2



```

"D:\Рабочий стол\4 семестр\опи\11\11lab\venv\Script
I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides

Process finished with exit code 0

```

Рисунок 9 – пример выполнения примера 2

### 6. Пример 3.

```
# Python program showing
# abstract base class work

from abc import ABC, abstractmethod

class Animal(ABC):
    def move(self):
        pass

class Human(Animal):
    def move(self):
        print("I can walk and run")

class Snake(Animal):
    def move(self):
        print("I can crawl")

class Dog(Animal):
    def move(self):
        print("I can bark")

class Lion(Animal):
    def move(self):
        print("I can roar")

# Driver code
R = Human()
R.move()

K = Snake()
K.move()

R = Dog()
R.move()

K = Lion()
K.move()
```

Рисунок 9 – пример 3

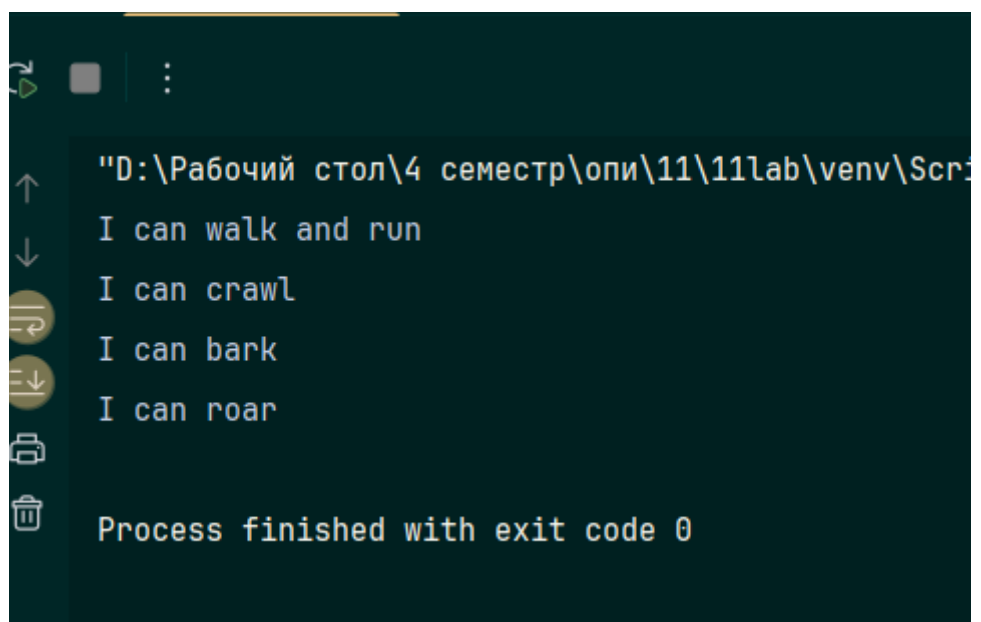


Рисунок 10 – пример выполнения примера 3



7. Создать класс Triad (тройка чисел); определить методы изменения полей и вычисления суммы чисел. Определить производный класс Triangle с полями-сторонами. Определить методы вычисления углов и площади треугольника.

```
import math

class Triad:
    def __init__(self, x=1, y=1, z=1):
        self.x = x
        self.y = y
        self.z = z

    def x_get(self):
        return self.x

    def y_get(self):
        return self.y

    def z_get(self):
        return self.z

    def x_set(self, x):
        self.x = x

    def y_set(self, y):
        self.y = y

    def z_set(self, z):
        self.z = z

    def adxyz(self):
        return self.x + self.y + self.z

class Triangle(Triad):
    def __init__(self, x, y, z):
        super().__init__(x, y, z)

    def s(self):
        p = (self.x + self.y + self.z) / 2
        return math.sqrt(p * (p - self.x) * (p - self.y) * (p - self.z))

    def angle_A(self):
        a, b, c = self.x, self.y, self.z
        return math.degrees(math.acos((b**2 + c**2 - a**2) / (2 * b * c)))

    def angle_B(self):
        a, b, c = self.x, self.y, self.z
        return math.degrees(math.acos((c**2 + a**2 - b**2) / (2 * c * a)))

    def angle_C(self):
        a, b, c = self.x, self.y, self.z
        return math.degrees(math.acos((a**2 + b**2 - c**2) / (2 * a * b)))

    def __str__(self):
        return f"x = {self.x}; y = {self.y}; z={self.z}"

j = Triangle(25, 23, 35)
print(j.s())
print(j.x_get())
j.y_set(34)
print(j.angle_A())
print(j.angle_B())
print(j.angle_C())
print(j)
```

Рисунок 11 – Выполнение задания 1

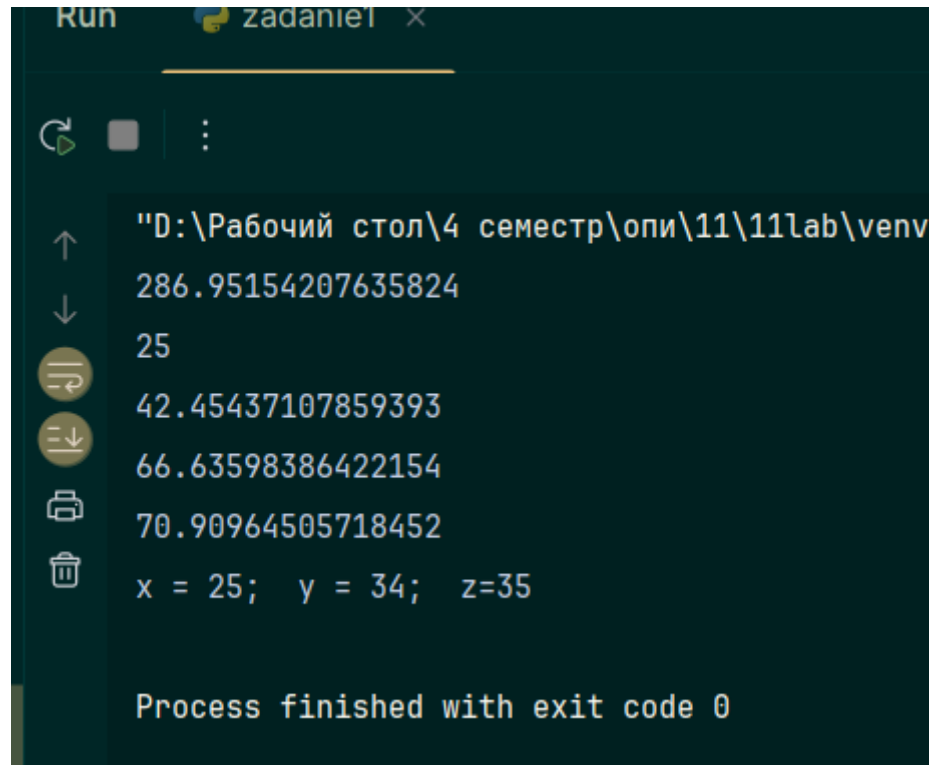


Рисунок 12– пример выполнение задания 1

6. Создать абстрактный базовый класс Pair с виртуальными арифметическими операциями. Создать производные классы FuzzyNumber (нечеткое число) и Complex (комплексное число).

```
from abc import ABC, abstractmethod

class Pair(ABC):
    def __init__(self, first, second):
        self.first = first
        self.second = second

    @abstractmethod
    def add(self, other):
        pass

    @abstractmethod
    def subtract(self, other):
        pass

    @abstractmethod
    def multiply(self, other):
        pass

    @abstractmethod
    def divide(self, other):
        pass

class FuzzyNumber(Pair):
    def add(self, other):
```

```

        return FuzzyNumber(self.first + other.first, self.second + other.second)

    def subtract(self, other):
        return FuzzyNumber(self.first - other.first, self.second - other.second)

    def multiply(self, other):
        return FuzzyNumber(self.first * other.first, self.second * other.second)

    def divide(self, other):
        return FuzzyNumber(self.first / other.first, self.second / other.second)

class ComplexNumber(Pair):
    def add(self, other):
        return ComplexNumber(self.first + other.first, self.second + other.second)

    def subtract(self, other):
        return ComplexNumber(self.first - other.first, self.second - other.second)

    def multiply(self, other):
        return ComplexNumber(self.first * other.first - self.second * other.second,
self.first * other.second + self.second * other.first)

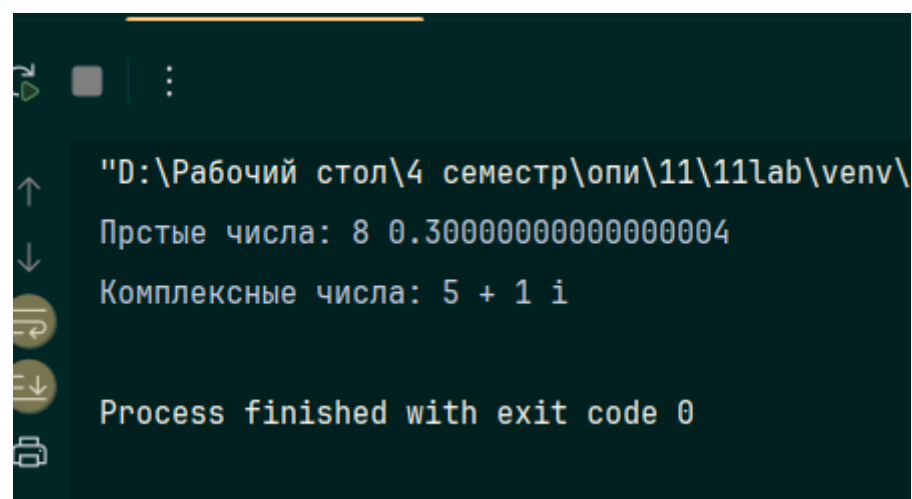
    def divide(self, other):
        denominator = other.first ** 2 + other.second ** 2
        real = (self.first * other.first + self.second * other.second) / denominator
        imaginary = (self.second * other.first - self.first * other.second) /
denominator
        return ComplexNumber(real, imaginary)

fuzzy1 = FuzzyNumber(3, 0.1)
fuzzy2 = FuzzyNumber(5, 0.2)
fuzzy_sum = fuzzy1.add(fuzzy2)
print("Прстые числа:", fuzzy_sum.first, fuzzy_sum.second)

complex1 = ComplexNumber(2, 3)
complex2 = ComplexNumber(1, -1)
complex_product = complex1.multiply(complex2)
print("Комплексные числа:", complex_product.first, "+", complex_product.second, "i")

```

Рисунок 13 – выполнение задания 2



```

"D:\Рабочий стол\4 семестр\опи\11\11lab\venv\
Прстые числа: 8 0.30000000000000004
Комплексные числа: 5 + 1 i
Process finished with exit code 0

```

Рисунок 14 – пример выполнения задания 2

```
(venv) PS D:\Рабочий стол\4 семестр\опи\11\11lab> git commit -m"1 коммит"
black.....asset
flake8.....asset
isort.....asset
[develop 9b58dd4] 1 коммит
 15 files changed, 447 insertions(+)
 create mode 100644 .idea/.gitignore
 create mode 100644 .idea/11lab.iml
 create mode 100644 .idea/inspectionProfiles/profiles_settings.xml
 create mode 100644 .idea/misc.xml
 create mode 100644 .idea/modules.xml
 create mode 100644 .idea/vcs.xml
 create mode 100644 .pre-commit-config.yaml
 create mode 100644 2primer.py
 create mode 100644 2zadanie.py
 create mode 100644 3primer.py
```

Рисунок 15 - фиксация изменений

```
(venv) PS D:\Рабочий стол\4 семестр\опи\11\11lab> git commit -m"1 коммит"
black.....asset
flake8.....asset
isort.....asset
[develop 9b58dd4] 1 коммит
 15 files changed, 447 insertions(+)
 create mode 100644 .idea/.gitignore
 create mode 100644 .idea/11lab.iml
 create mode 100644 .idea/inspectionProfiles/profiles_settings.xml
 create mode 100644 .idea/misc.xml
 create mode 100644 .idea/modules.xml
 create mode 100644 .idea/vcs.xml
 create mode 100644 .pre-commit-config.yaml
 create mode 100644 2primer.py
 create mode 100644 2zadanie.py
 create mode 100644 3primer.py
```

Рисунок 16 – ветка develop в git

Контрольные вопросы:

1.Наследование в Python - это механизм, позволяющий классам наследовать атрибуты и методы родительского класса. Это реализуется путем указания родительского класса в определении дочернего класса. Дочерний класс получает все атрибуты и методы родительского класса и может добавить

собственные атрибуты и методы или переопределить методы родительского класса.

2.Полиморфизм в Python означает способность объектов разных классов обладать одинаковым интерфейсом. Это позволяет использовать объекты разных типов с одинаковыми методами без необходимости знать их конкретный тип. В Python полиморфизм обычно реализуется с помощью перегрузки операторов и методов.

3."Утиная" типизация в Python - это принцип, согласно которому тип объекта определяется его поведением, а не его явным объявлением типа. Это означает, что если объект ведет себя как утка (то есть имеет методы и атрибуты, которые можно ожидать от утки), то он считается уткой, независимо от его фактического типа.

4.Модуль abc в Python предоставляет инструменты для создания абстрактных базовых классов (ABC). Абстрактные базовые классы могут содержать абстрактные методы, которые должны быть реализованы в производных классах. Это помогает обеспечить единый интерфейс для классов, реализующих определенный функционал.

5.Для создания абстрактного метода в классе Python нужно импортировать модуль abc и использовать декоратор @abstractmethod перед определением метода в абстрактном базовом классе.

6.Для создания абстрактного свойства в Python также можно использовать модуль abc. Это достигается путем определения абстрактного метода и использования декоратора @property перед ним.

7.Функция isinstance() в Python используется для проверки принадлежности объекта к определенному классу или типу данных. Она возвращает True, если объект является экземпляром указанного класса или его подкласса, и False в противном случае.