

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Департамент цифровых, робототехнических систем и электроники института
перспективной инженерии

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №4.4
дисциплины «Объектно-ориентированное программирование»

Выполнил:
Юрьев Илья Евгеньевич
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:
Хацукова А.И., ассистент
департамента цифровых
робототехнических систем и
электроники

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

Тема: 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 выполнения работы:

создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и .gitignore файл для языка программирования Python:

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner * **Repository name ***

daxstrong ▾ / OOP_4

✔ OOP_4 is available.

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

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.](#)

This will set **main** as the default branch. Change the default name in your [settings](#).

You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание репозитория с заданными настройками

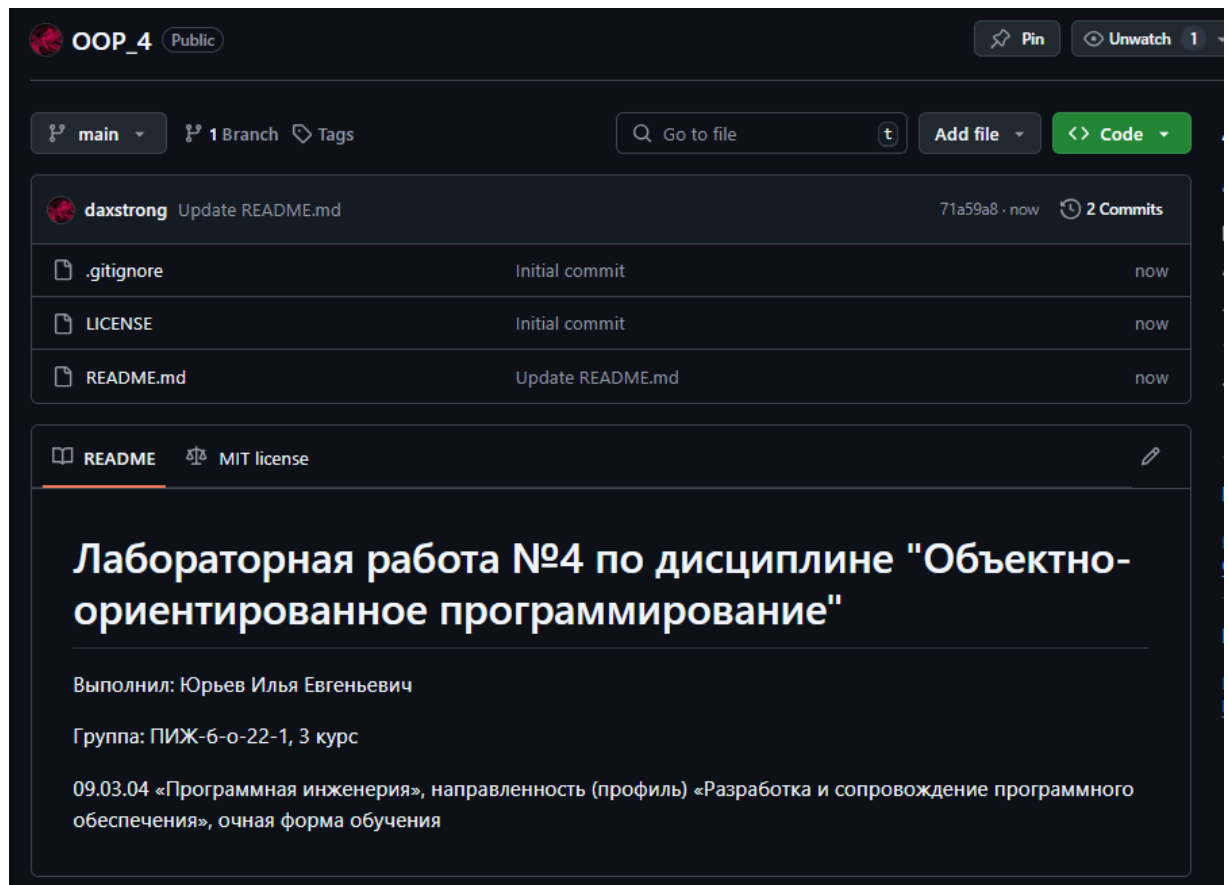


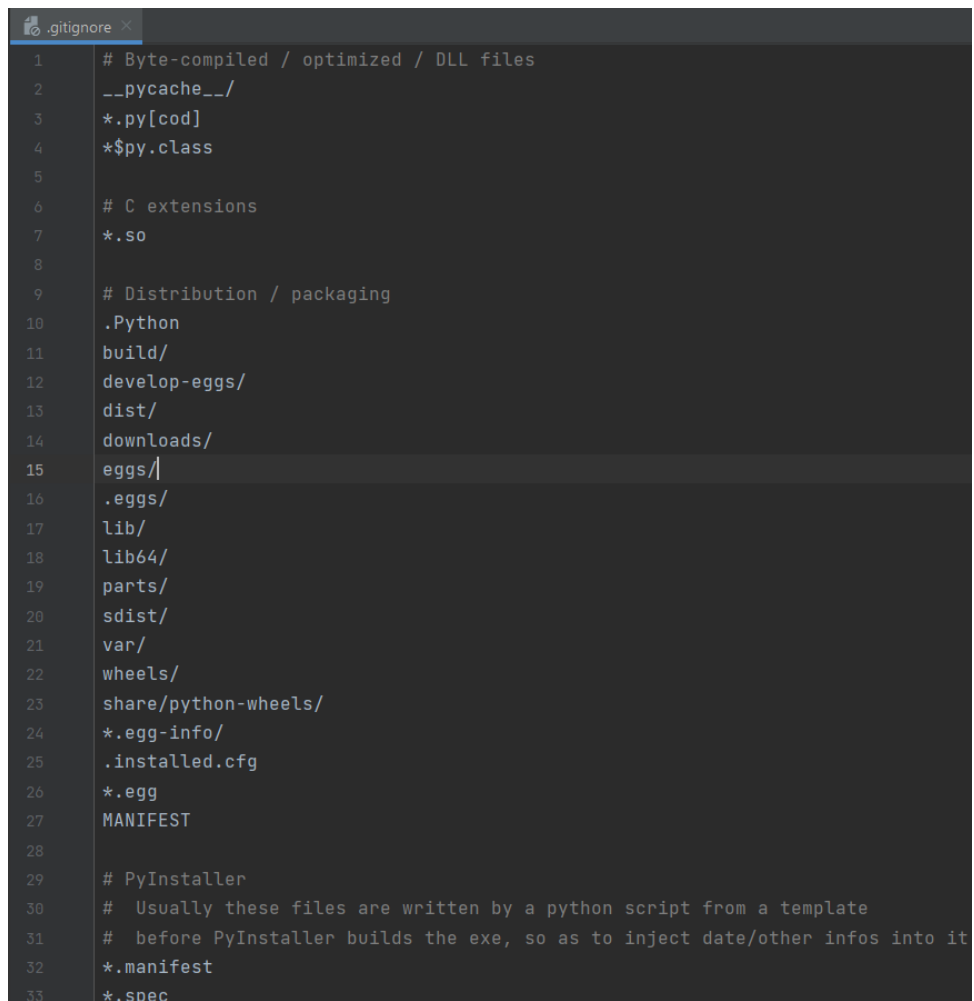
Рисунок 2 – Созданный репозиторий

```
PS C:\Users\Ilya\Documents\labs\OOP> git clone https://github.com/daxstrong/OOP_4.git
Cloning into 'OOP_4'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (8/8), 4.16 KiB | 2.08 MiB/s, done.
Resolving deltas: 100% (1/1), done.
PS C:\Users\Ilya\Documents\labs\OOP> |
```

Рисунок 3 – Клонирование репозитория

```
PS C:\Users\Ilya\Documents\labs\OOP\OOP_4> git checkout -b develop
Switched to a new branch 'develop'
PS C:\Users\Ilya\Documents\labs\OOP\OOP_4> |
```

Рисунок 4 – Создание ветки develop, где будут происходить изменения проекта до его полного релиза



The image shows a code editor window with a tab labeled ".gitignore". The file contains a list of patterns to be ignored by Git, with line numbers 1 through 33 on the left margin. The patterns are as follows:

```
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/
24 *.egg-info/
25 .installed.cfg
26 *.egg
27 MANIFEST
28
29 # PyInstaller
30 # Usually these files are written by a python script from a template
31 # before PyInstaller builds the exe, so as to inject date/other infos into it
32 *.manifest
33 *.spec
```

Рисунок 5 – Часть .gitignore, созданного GitHub

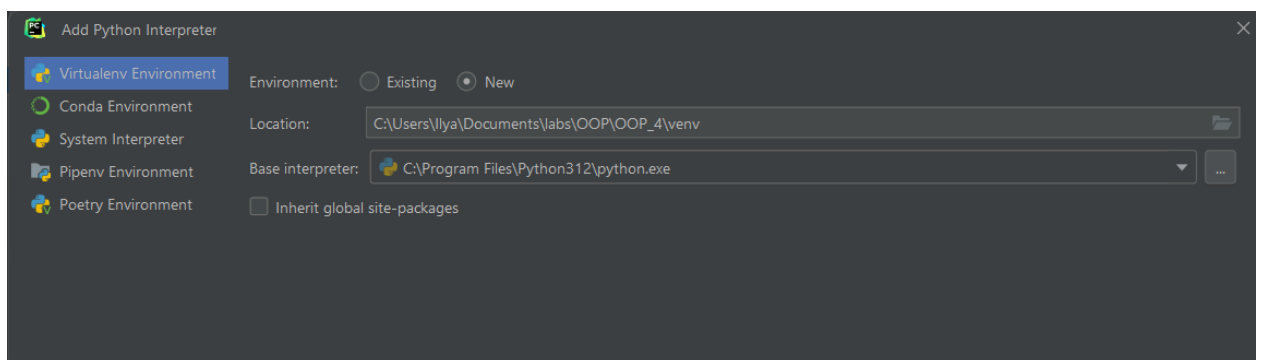


Рисунок 6 – Создание виртуального окружения

Try the redesigned packaging support in Python Packages tool window. [Go to tool window](#) ×

Package	Version	Latest version
pip	23.2.1	⬆ 24.3.1
setuptools	68.2.0	⬆ 75.3.0
wheel	0.41.2	⬆ 0.44.0

Рисунок 7 – Созданное окружение

доработка примера №1 из лабораторной работы. Пример реализации абстрактного класса Polygon:

Листинг 1 – Код примера из лабораторной работы №1

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

# 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")

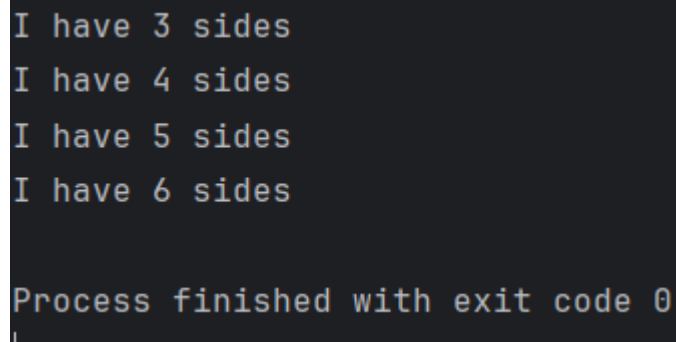
def main():
    # Driver code
```

```

R = Triangle()
R.noofsides()
K = Quadrilateral()
K.noofsides()
R = Pentagon()
R.noofsides()
K = Hexagon()
K.noofsides()

if __name__ == '__main__':
    main()

```



```

I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides

Process finished with exit code 0

```

Рисунок 8 – Пример вывода программы

ешение примера №2. Пример реализации абстрактного класса Animal:

Листинг 2 – Код примера из лабораторной работы №2

```

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

# 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")

def main():
    # Driver code
    R = Human()
    R.move()

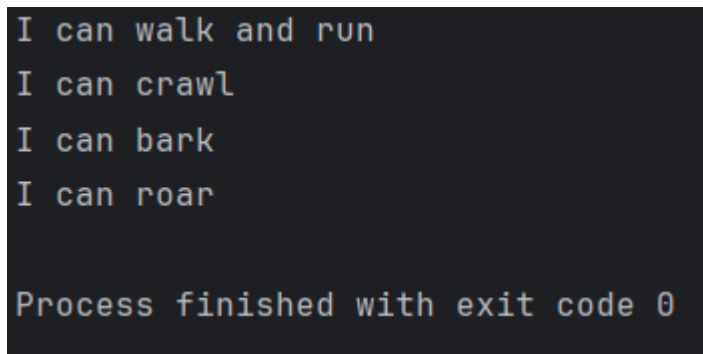
    K = Snake()
    K.move()

    R = Dog()
    R.move()

    K = Lion()
    K.move()

if __name__ == '__main__':
    main()

```



```

I can walk and run
I can crawl
I can bark
I can roar

Process finished with exit code 0

```

Рисунок 9 – Пример выполнения программы

ешить задачу: В некой игре-стратегии есть солдаты и герои. У всех есть свойство, содержащее уникальный номер объекта, и свойство, в котором хранится принадлежность команде. У солдат есть метод "иду за героем", который в качестве аргумента принимает объект типа "герой". У героев есть метод увеличения собственного уровня. В основной ветке программы создается по одному герою для каждой команды. В цикле генерируются объекты-солдаты. Их принадлежность команде определяется случайно. Солдаты разных команд добавляются в разные списки. Измеряется длина списков солдат противоборствующих команд и выводится на экран. У героя,

принадлежащего команде с более длинным списком, увеличивается уровень. Отправьте одного из солдат первого героя следовать за ним. Выведите на экран идентификационные номера этих двух юнитов:

Листинг 3 – Код решения задачи

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

from abc import ABC, abstractmethod
from enum import Enum
import random

# Enum для определения команд
class Team(Enum):
    RED = 1
    BLUE = 2

# Абстрактный класс для юнитов
class Unit(ABC):
    def __init__(self, id, team):
        self.id = id
        self.team = team

    @abstractmethod
    def follow(self, obj):
        pass

# Класс для солдат
class Soldier(Unit):
    def follow(self, hero):
        print(f'Солдат {self.id} следует за героем {hero.id}')

# Класс для героев
class Hero(Unit):
    def __init__(self, id, team):
        super().__init__(id, team)
        self.level = 1

    def follow(self, obj):
        pass

    def increase_level(self):
        self.level += 1
        print(f'Герой {self.id} повышен до уровня {self.level}\n')

# Функция для создания солдат
def create_soldiers(num_soldiers):
    soldiers = {Team.RED: [], Team.BLUE: []}
    for i in range(num_soldiers):
        team = random.choice(list(Team))
        soldier = Soldier(i, team)
        soldiers[team].append(soldier)
    return soldiers
```



```

def main():
    # Создаем по одному герою для каждой команды
    heroes = {Team.RED: Hero('Красный', Team.RED), Team.BLUE: Hero('Синий',
Team.BLUE)}

    # Генерируем случайное количество солдат для каждой команды
    soldiers = create_soldiers(100)

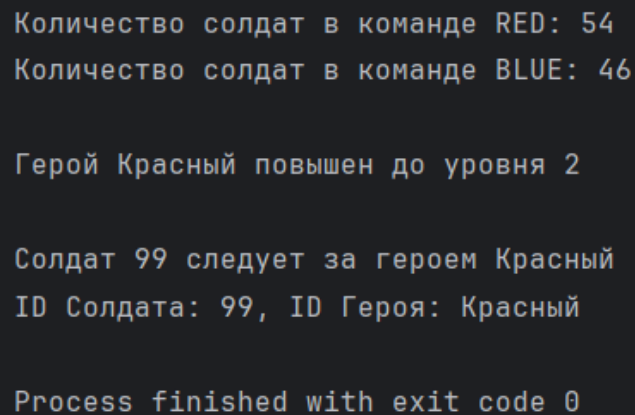
    # Выводим количество солдат в каждой команде
    print(f'Количество солдат в команде RED: {len(soldiers[Team.RED])}')
    print(f'Количество солдат в команде BLUE: {len(soldiers[Team.BLUE])}\n')

    # Определяем, у какой команды больше солдат, и повышаем уровень
соответствующего героя
    if len(soldiers[Team.RED]) > len(soldiers[Team.BLUE]):
        heroes[Team.RED].increase_level()
    elif len(soldiers[Team.RED]) < len(soldiers[Team.BLUE]):
        heroes[Team.BLUE].increase_level()
    else:
        print('Количество солдат в командах равно. Никто не получает
повышение\n')

    # Отправляем одного из солдат первого героя (RED) следовать за ним
    if soldiers[Team.RED]:
        soldier = random.choice(soldiers[Team.RED])
        soldier.follow(heroes[Team.RED])
        print(f'ID Солдата: {soldier.id}, ID Героя: {heroes[Team.RED].id}')

if __name__ == '__main__':
    main()

```



```

Количество солдат в команде RED: 54
Количество солдат в команде BLUE: 46

Герой Красный повышен до уровня 2

Солдат 99 следует за героем Красный
ID Солдата: 99, ID Героя: Красный

Process finished with exit code 0

```

Рисунок 10 – Пример вывода программы (1)

```
Количество солдат в команде RED: 47
Количество солдат в команде BLUE: 53

Герой Синий повышен до уровня 2
|
Солдат 89 следует за героем Красный
ID Солдата: 89, ID Героя: Красный

Process finished with exit code 0
```

Рисунок 11 – Пример вывода программы (2)

```
Количество солдат в команде RED: 2
Количество солдат в команде BLUE: 2

Количество солдат в командах равно. Никто не получает повышение

Солдат 2 следует за героем Красный
ID Солдата: 2, ID Героя: Красный

Process finished with exit code 0
```

Рисунок 12 – Пример вывода программы (3)

ешить индивидуальное задание №1 (вариант №12). Реализовать класс-оболочку Number для числового типа float. Реализовать методы сложения и деления. Создать производный класс Real, в котором реализовать метод возведения в производную степень, и метод для вычисления логарифма числа:

Листинг 4 – Код индивидуального задания №1

```
import math

class Number:
    """Класс-оболочка для числового типа float."""

    def __init__(self, value):
        """Инициализация с приведением к типу float."""
        self.value = float(value)

    def __add__(self, other):
        """Метод сложения. Принимает другой объект Number или число."""
        if isinstance(other, Number):
            return Number(self.value + other.value)
        else:
            return Number(self.value + float(other))
```

```

def __truediv__(self, other):
    """Метод деления. Принимает другой объект Number или число."""
    if isinstance(other, Number):
        if other.value == 0:
            raise ZeroDivisionError("Деление на ноль.")
        return Number(self.value / other.value)
    else:
        if float(other) == 0:
            raise ZeroDivisionError("Деление на ноль.")
        return Number(self.value / float(other))

def __repr__(self):
    """Представление объекта для вывода."""
    return f"Number({self.value})"

class Real(Number):
    """Производный класс от Number с дополнительными методами."""

    def power(self, exponent):
        """Метод возведения числа в произвольную степень."""
        return Real(self.value ** exponent)

    def logarithm(self, base=math.e):
        """Метод вычисления логарифма числа по заданному основанию."""
        if self.value <= 0:
            raise ValueError("Логарифм не определен для неположительных чисел.")
        return Real(math.log(self.value, base))

    def __repr__(self):
        """Представление объекта для вывода."""
        return f"Real({self.value})"

if __name__ == '__main__':
    # Создание объектов класса Number
    num1 = Number(10.5)
    num2 = Number(2.5)

    # Демонстрация сложения
    sum_result = num1 + num2
    print(f"{num1} + {num2} = {sum_result}")

    # Демонстрация деления
    div_result = num1 / num2
    print(f"{num1} / {num2} = {div_result}")

    # Создание объекта класса Real
    real_num = Real(16.0)

    # Демонстрация возведения в степень
    power_result = real_num.power(0.5)
    print(f"{real_num} в степени 0.5 = {power_result}")

    # Демонстрация вычисления логарифма
    log_result = real_num.logarithm(math.sqrt(16)) # Логарифм по основанию 4
    print(f"Логарифм {real_num} по основанию 4 = {log_result}")

```

```

C:\Users\ilya\Documents\labs\OOP\OOP_4\indiv_task.py
Number(10.5) + Number(2.5) = Number(13.0)
Number(10.5) / Number(2.5) = Number(4.2)
Real(16.0) в степени 0.5 = Real(4.0)
Логарифм Real(16.0) по основанию 4 = Real(2.0)

```

Рисунок 13 – Пример вывода кода индивидуального задания №1

ешить индивидуальное задание №2 (вариант №12). Создать абстрактный базовый класс Integer (целое) с виртуальными арифметическими операциями и функцией вывода на экран. Определить производные классы Decimal (десятичное) и Binary (двоичное), реализующие собственные арифметические операции и функцию вывода на экран. Число представляется массивом, каждый элемент которого – цифра:

Листинг 5 – Код индивидуального задания №2

```

from abc import ABC, abstractmethod

class Integer(ABC):
    def __init__(self, digits):
        """Initialize with a list of digits."""
        self.digits = digits

    @abstractmethod
    def add(self, other):
        """Add another Integer and return a new Integer."""
        pass

    @abstractmethod
    def subtract(self, other):
        """Subtract another Integer and return a new Integer."""
        pass

    @abstractmethod
    def display(self):
        """Display the number."""
        pass

class Decimal(Integer):
    def __init__(self, digits):
        super().__init__(digits)

    def to_int(self):
        """Convert digit list to integer."""
        return int(''.join(map(str, self.digits)))

    @classmethod
    def from_int(cls, number):
        """Create Decimal from integer."""

```

```

        return cls([int(d) for d in str(number)])

def add(self, other):
    if not isinstance(other, Decimal):
        raise TypeError("Can only add Decimal with Decimal")
    result = self.to_int() + other.to_int()
    return Decimal.from_int(result)

def subtract(self, other):
    if not isinstance(other, Decimal):
        raise TypeError("Can only subtract Decimal with Decimal")
    result = self.to_int() - other.to_int()
    if result < 0:
        raise ValueError("Result cannot be negative in Decimal")
    return Decimal.from_int(result)

def display(self):
    """Display the decimal number."""
    print(''.join(map(str, self.digits)))

class Binary(Integer):
    def __init__(self, digits):
        super().__init__(digits)

    def to_int(self):
        """Convert binary digit list to integer."""
        return int(''.join(map(str, self.digits)), 2)

    @classmethod
    def from_int(cls, number):
        """Create Binary from integer."""
        binary_str = bin(number)[2:] # Remove '0b' prefix
        return cls([int(d) for d in binary_str])

    def add(self, other):
        if not isinstance(other, Binary):
            raise TypeError("Can only add Binary with Binary")
        result = self.to_int() + other.to_int()
        return Binary.from_int(result)

    def subtract(self, other):
        if not isinstance(other, Binary):
            raise TypeError("Can only subtract Binary with Binary")
        result = self.to_int() - other.to_int()
        if result < 0:
            raise ValueError("Result cannot be negative in Binary")
        return Binary.from_int(result)

    def display(self):
        """Display the binary number."""
        print(''.join(map(str, self.digits)))

if __name__ == "__main__":
    # Создаем два десятичных числа
    dec1 = Decimal([1, 2, 3]) # 123
    dec2 = Decimal([4, 5, 6]) # 456

    # Добавляем десятичные числа
    dec_sum = dec1.add(dec2)
    print("Decimal Sum:")
    dec_sum.display() # Ожидается 579

    # Вычитаем десятичные числа

```

```

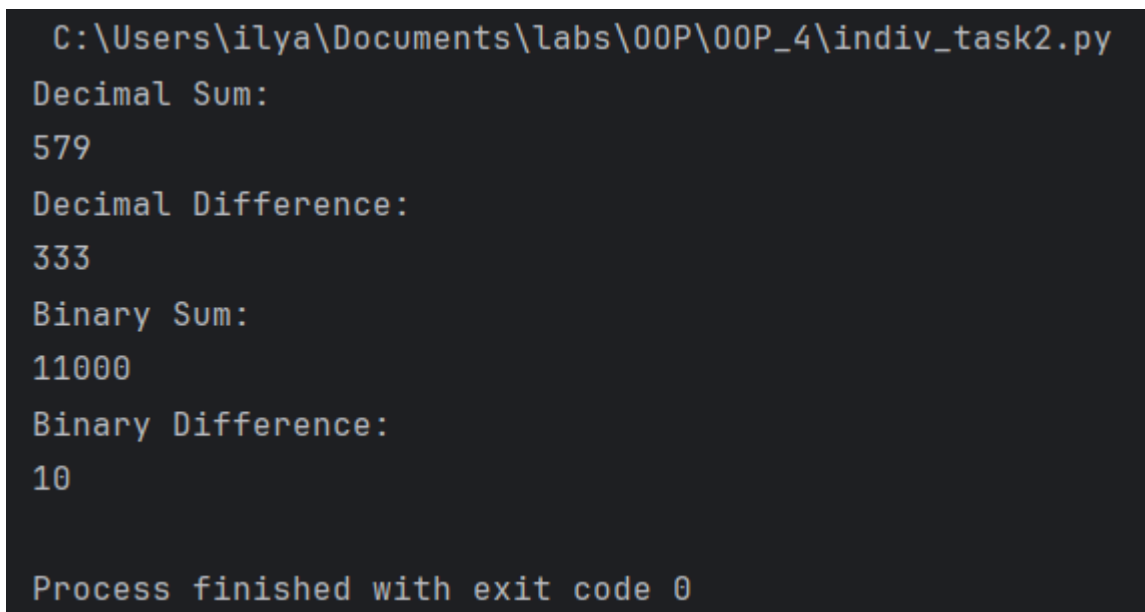
dec_diff = dec2.subtract(dec1)
print("Decimal Difference:")
dec_diff.display() # Ожидается 333

# Создаем два двоичных числа
bin1 = Binary([1, 0, 1, 1]) # 11 в десятичной
bin2 = Binary([1, 1, 0, 1]) # 13 в десятичной

# Добавляем двоичные числа
bin_sum = bin1.add(bin2)
print("Binary Sum:")
bin_sum.display() # Ожидается 11000 (24 в десятичной)

# Вычитаем двоичные числа
bin_diff = bin2.subtract(bin1)
print("Binary Difference:")
bin_diff.display() # Ожидается 10 (2 в десятичной)

```



```

C:\Users\ilya\Documents\labs\OOP\OOP_4\indiv_task2.py
Decimal Sum:
579
Decimal Difference:
333
Binary Sum:
11000
Binary Difference:
10

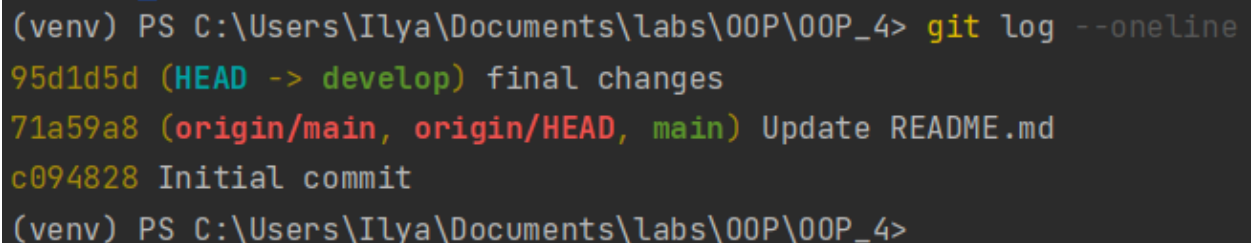
Process finished with exit code 0

```

Рисунок 14 – Примеры выполнения кода индивидуального задания №2

ольем ветки develop и main/master и отправим изменения на удаленный репозиторий:

Ссылка: https://github.com/daxstrong/OOP_4



```

(venv) PS C:\Users\Ilya\Documents\labs\OOP\OOP_4> git log --oneline
95d1d5d (HEAD -> develop) final changes
71a59a8 (origin/main, origin/HEAD, main) Update README.md
c094828 Initial commit
(venv) PS C:\Users\Ilya\Documents\labs\OOP\OOP_4>

```

Рисунок 15 – История коммитов

```
PS C:\Users\Ilya\Documents\labs\00P\00P_4> git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\Users\Ilya\Documents\labs\00P\00P_4> git merge develop
Updating 71a59a8..c399c73
Fast-forward
 .idea/.gitignore | 8 ++
 rename README.md => 00P_4/README.md (100%)
 create mode 100644 00P_4/ex1.py
 create mode 100644 00P_4/ex1_1.py
 create mode 100644 00P_4/ex2.py
 create mode 100644 00P_4/indiv_task.py
 create mode 100644 00P_4/indiv_task2.py
PS C:\Users\Ilya\Documents\labs\00P\00P_4>
```

Рисунок 16 – Слияние веток main и develop

```
PS C:\Users\Ilya\Documents\labs\00P\00P_4> git push origin main
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 12 threads
Compressing objects: 100% (20/20), done.
Writing objects: 100% (22/22), 6.99 KiB | 2.33 MiB/s, done.
Total 22 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
```

Рисунок 17 – Отправка изменений на удаленный репозиторий

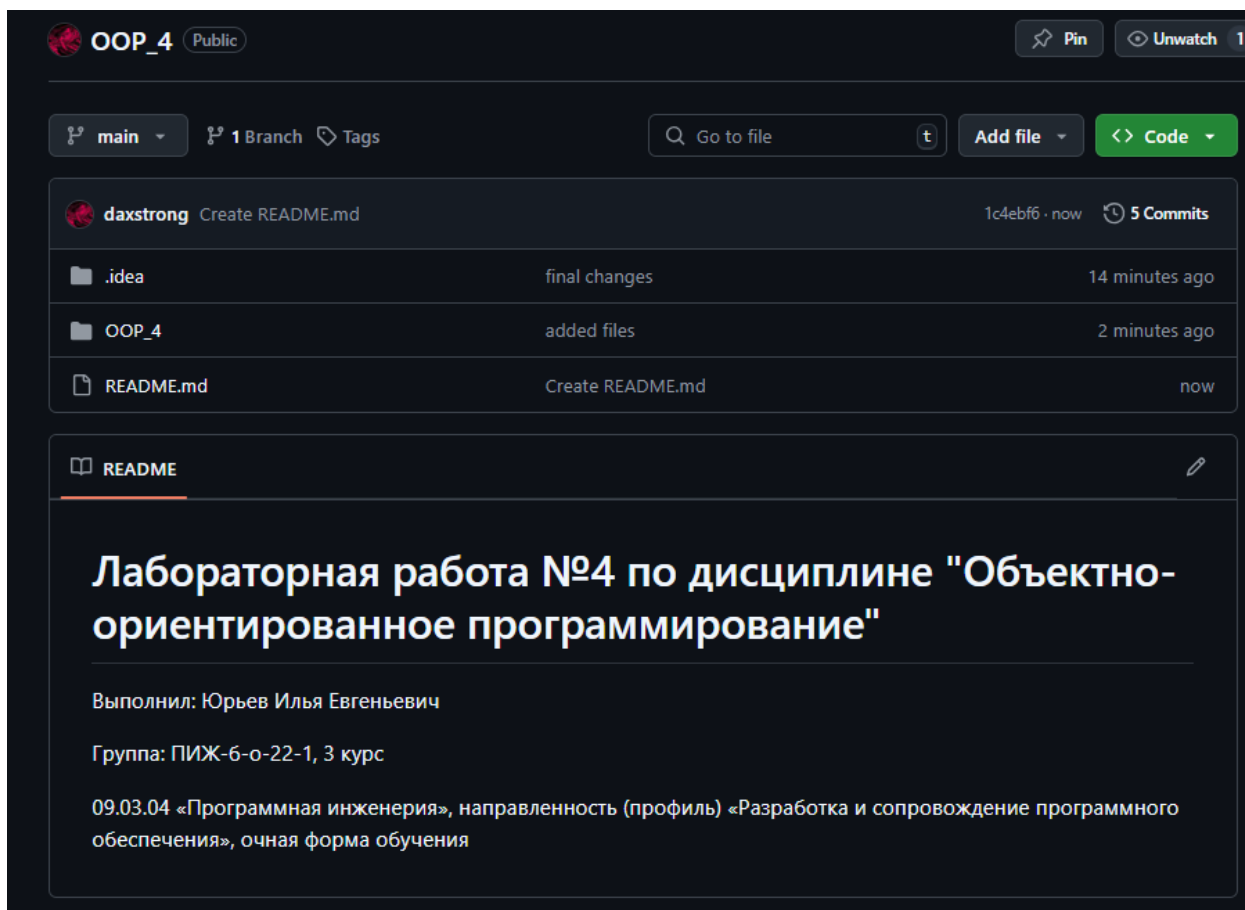


Рисунок 18 – Изменения в GitHub

Вывод: приобрели навыки по созданию иерархии классов (родители – наследники) при написании различных программ с помощью языка программирования Python версии 3.x.

Ответы на контрольные вопросы:

1. Что такое наследование как оно реализовано в языке Python?

Наследование — это возможность создания нового класса на основе уже существующего класса, чтобы унаследовать его атрибуты и методы. В Python наследование реализуется путем указания базового класса в скобках после имени нового класса.

2. Что такое полиморфизм и как он реализован в языке Python?

Полиморфизм — это возможность объектов с разными типами быть обработанными одним и тем же общим способом. В Python полиморфизм реализуется автоматически благодаря динамической типизации языка.

3. Что такое "утиная" типизация в языке программирования Python?

"Утиная" типизация в Python означает, что тип объекта важнее, чем его класс. Это означает, что важно, какой метод или свойство объекта может быть вызван, а не какой класс он имеет

4. Каково назначение модуля abc языка программирования Python?

Модуль abc (Abstract Base Classes) предоставляет инструменты для создания абстрактных базовых классов, которые могут содержать абстрактные методы и свойства, которые должны быть реализованы в дочерних классах.

5. Как сделать некоторый метод класса абстрактным?

Для того чтобы сделать метод абстрактным, нужно использовать декоратор `@abstractmethod` перед его объявлением в абстрактном базовом классе.

6. Как сделать некоторое свойство класса абстрактным?

Для того чтобы сделать свойство абстрактным, можно внести его в абстрактный класс с помощью декоратора `@abstractmethod`.

7. Каково назначение функции `isinstance`?

Функция `isinstance` используется для проверки является ли указанный объект экземпляром указанного класса или его подкласса. Она возвращает `True`, если объект является экземпляром класса, и `False` в противном случае.