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 №4.3
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: перегрузка операторов в языке Python.

Цель работы: приобретение навыков по перегрузке операторов при написании программ с помощью языка программирования Python версии 3.x.

Ход выполнения работы:

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

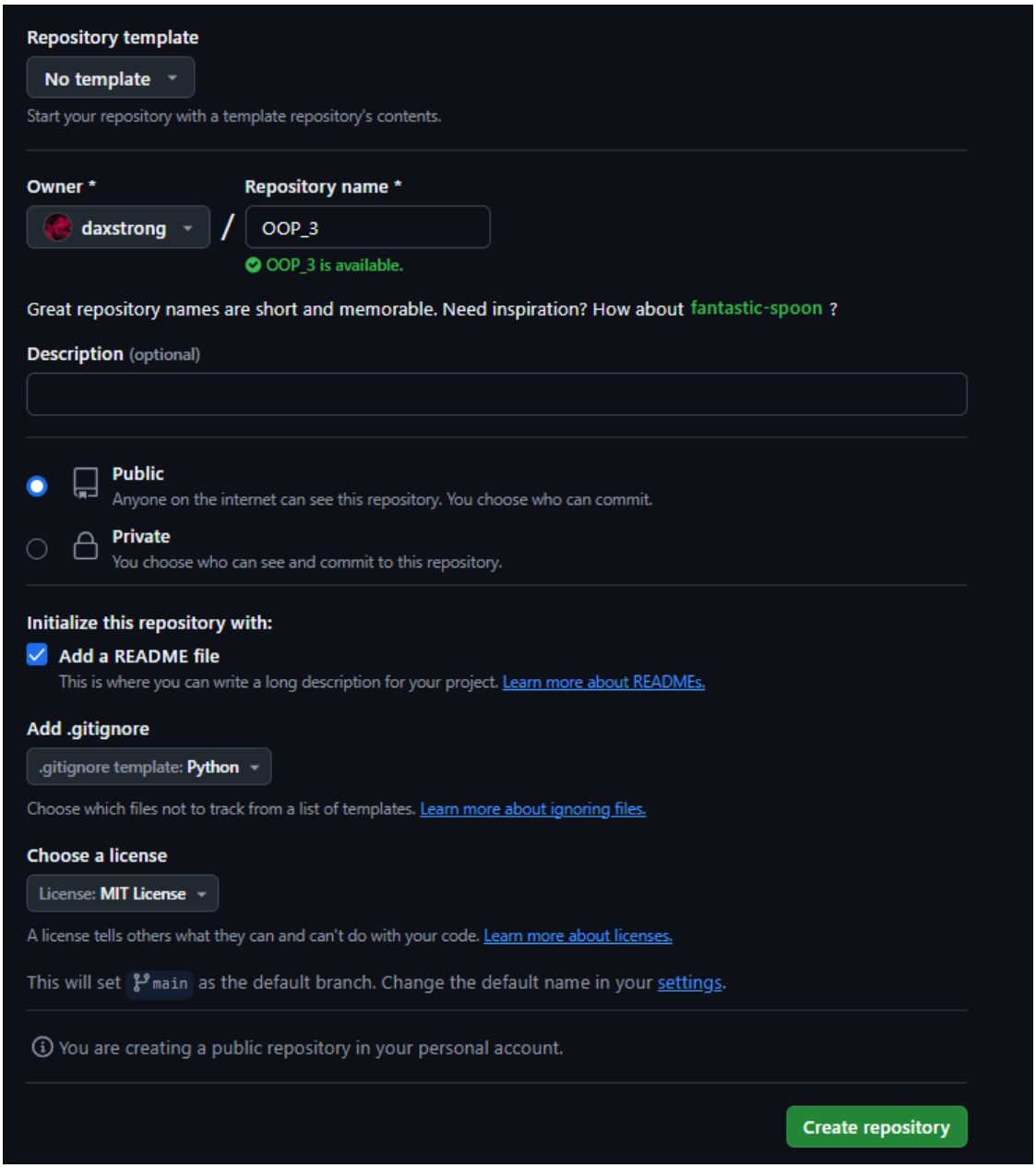

Рисунок 1 – Создание репозитория с заданными настройками

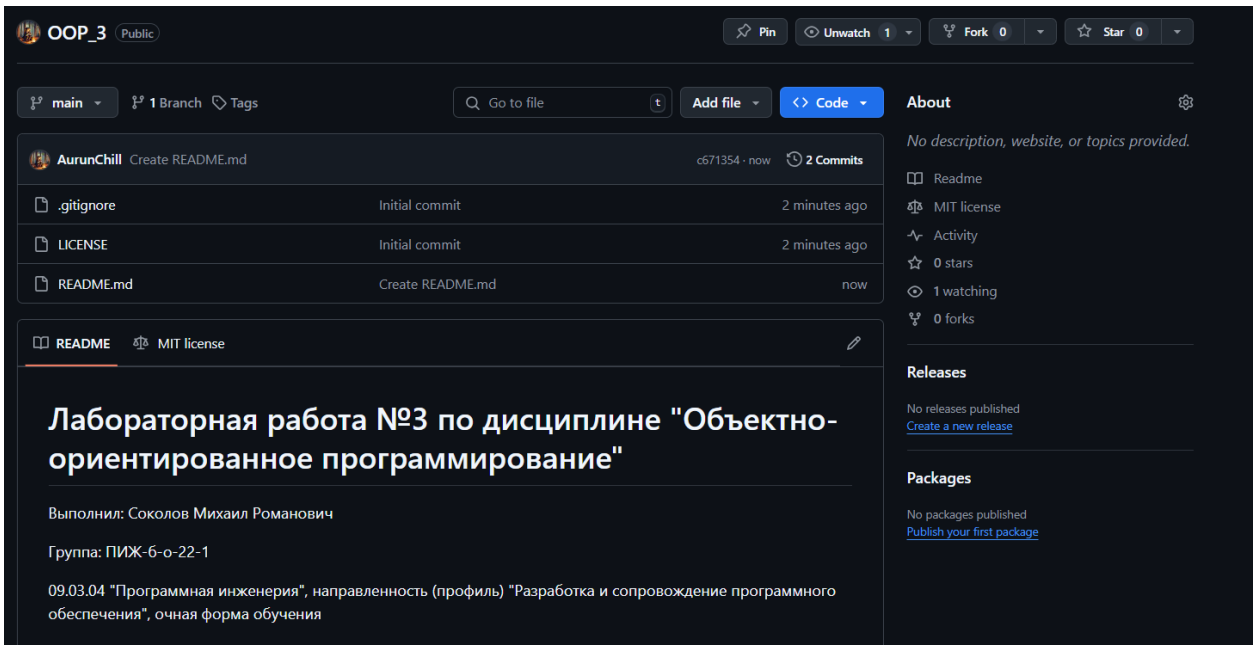

Рисунок 2 – Созданный репозиторий

```
PS C:\Users\Ilya\Documents\labs\OOP> git clone https://github.com/daxstrong/OOP_3.git
Cloning into 'OOP_3'...
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 (from 0)
Receiving objects: 100% (5/5), done.
PS C:\Users\Ilya\Documents\labs\OOP>
```

Рисунок 3 – Клонирование репозитория

```
PS C:\Users\Ilya\Documents\labs\OOP\OOP_3> git checkout -b develop
Switched to a new branch 'develop'
PS C:\Users\Ilya\Documents\labs\OOP\OOP_3>
```

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

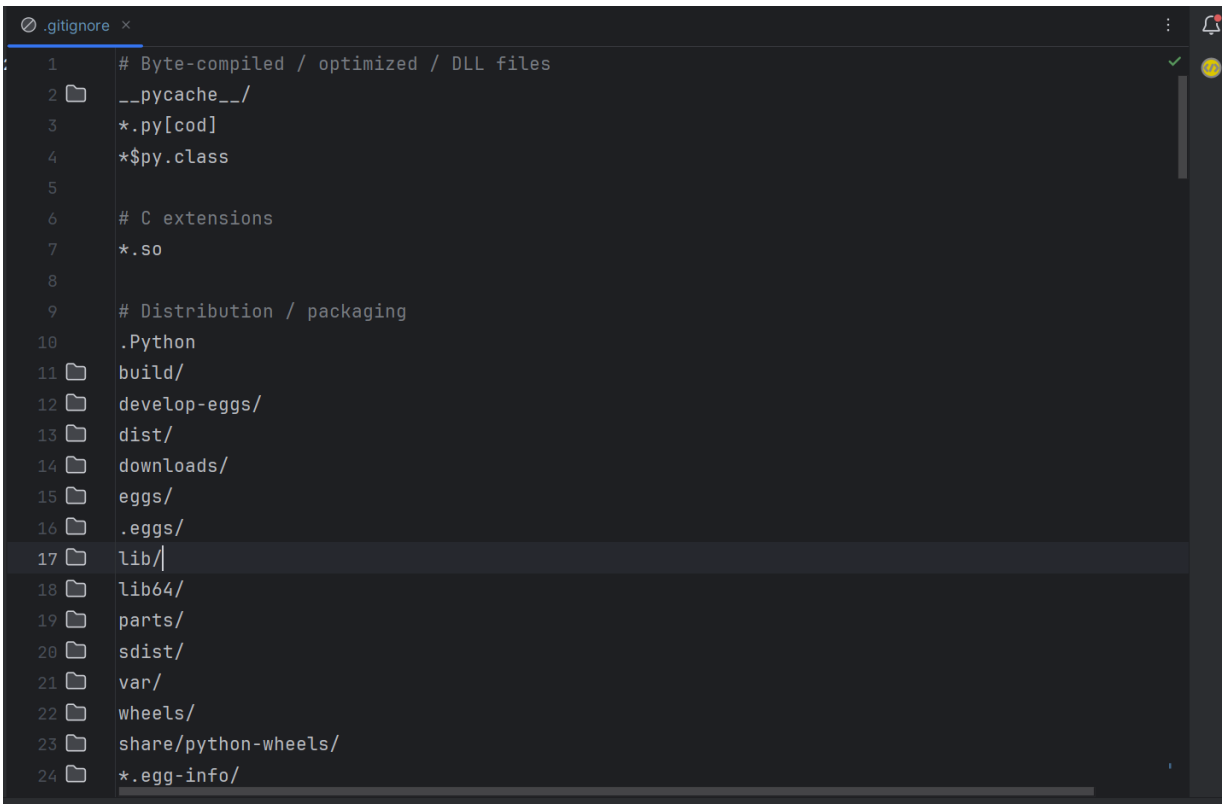

Рисунок 5 – Часть .gitignore, созданного GitHub

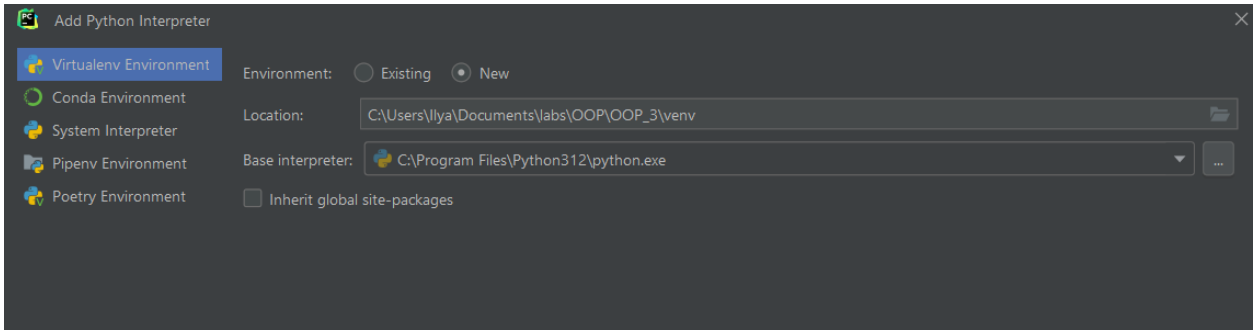

Рисунок 6 – Создание виртуального окружения

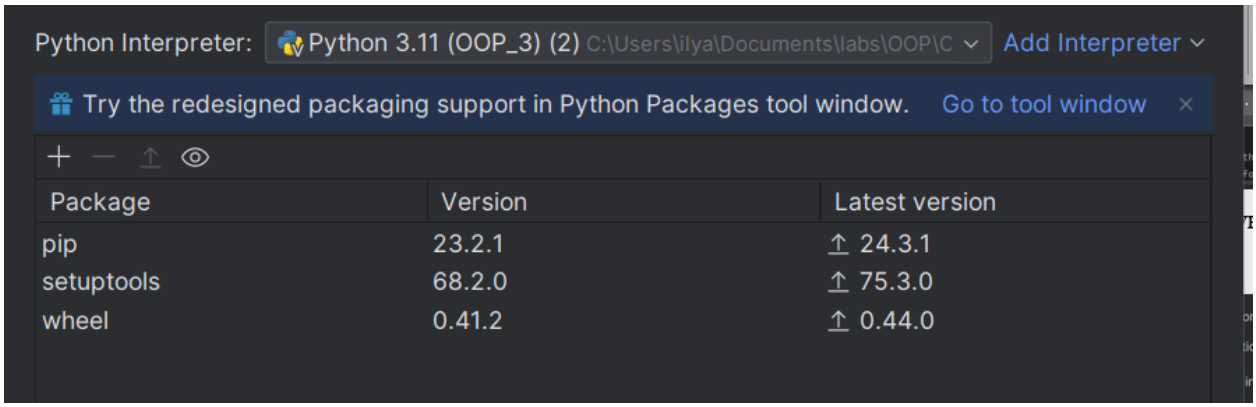

Рисунок 7 – Созданное окружение

адание лабораторной работы №1. Изменить класс Rational из примера №1 лабораторной работы №4.1, используя перегрузку операторов:

Листинг 1 – Код класса Rational с использованием перегрузки операторов

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

class Rational:
    def __init__(self, a: int = 0, b: int = 1):
        if b == 0:
            raise ValueError()
        self.__numerator = abs(int(a))
        self.__denominator = abs(int(b))
        self.__reduce()

    # Сокращение дроби
    def __reduce(self):
        # Функция для нахождения нибольшего общего делителя
        def gcd(a: int, b: int):
            if a == 0:
                return b
            elif b == 0:
                return a
            elif a >= b:
                return gcd(a % b, b)
            else:
                return gcd(a, b % a)

        sign = 1
        if (self.__numerator > 0 and self.__denominator < 0) or \
                (self.__numerator < 0 and self.__denominator > 0):
            sign = -1

        a, b = abs(self.__numerator) , abs(self.__denominator)
        c = gcd(a, b)

        self.__numerator = sign * (a // c)
        self.__denominator = b // c

    def __clone(self):
        return Rational(self.__numerator, self.__denominator)

    @property
    def numerator(self) -> int:
        return self.__numerator

    @numerator.setter
    def numerator(self, value):
        self.__numerator = int(value)
        self.__reduce()

    @property
    def denominator(self) -> int:
        return self.__denominator

    @denominator.setter
    def denominator(self, value):
        value = int(value)
        if value == 0:
            raise ValueError('Illegal value of the denominator')
```

```
        self.__denominator = value
        self.__reduce()

    def __str__(self):
        return f'{self.__numerator} / {self.__denominator}'

    def __repr__(self):
        return self.__str__()

    def __float__(self):
        return self.__numerator / self.__denominator

    def __bool__(self):
        return self.__numerator != 0

    def __iadd__(self, other):
        if isinstance(other, Rational):
            a = self.numerator * other.denominator / self.denominator *
other.numerator
            b = self.denominator * other.denominator

            self.__numerator, self.__denominator = a, b
            self.__reduce()
            return self
        else:
            raise ValueError('Illegal type of the argument')

    def __add__(self, other):
        return self.__clone().__iadd__(other)

    def __isub__(self, other):
        if isinstance(other, Rational):
            a = self.numerator * other.denominator - self.denominator *
other.numerator
            b = self.denominator * other.denominator
            self.__numerator, self.__denominator = a, b
            self.__reduce()
            return self
        else:
            raise ValueError('Illegal tpe of the argument')

    def __sub__(self, other):
        return self.__clone().__isub__(other)

    def __imul__(self, other):
        if isinstance(other, Rational):
            a = self.numerator * other.numerator
            b = self.denominator * other.denominator

            self.__numerator, self.__denominator = a, b
            self.__reduce()
            return self
        else:
            raise ValueError('Illegal type of the argument')

    def   mul  (self, other):
        return self.__clone().__imul__(other)

    def __itruediv__(self, other):
        if isinstance(other, Rational):
            a = self.numerator * other.denominator
            b = self.denominator * other.numerator
```

```
            if b == 0:
                raise ValueError('Illegal value of the denominator')
            self.__numerator, self.__denominator = a, b
            self.__reduce()
            return self
        else:
            raise ValueError('Illegal type of the argument')

    def __eq__(self, other):
        if isinstance(other, Rational):
            return (self.numerator == other.numerator) and (self.denominator
== other.denominator)

    def __ne__(self, other):
        if isinstance(other, Rational):
            return not self.__eq__(other)
        else:
            return False

    def __gt__(self, other):
        if isinstance(other, Rational):
            return self.__float__() > other.__float__()
        else:
            return False

    def __lt__(self, other):
        if isinstance(other, Rational):
            return self.__float__() < other.__float__()
        else:
            return False

    def __ge__(self, other):
        if isinstance(other, Rational):
            return not self.__lt__(other)
        else:
            return False

    def __le__(self, other):
        if isinstance(other, Rational):
            return not self.__gt__(other)
        else:
            return False


if __name__ == '__main__':
    r1 = Rational(3, 4)
    print(f"r1 = {r1}")
    r2 = Rational(5, 6)
    print(f"r2 = {r2}")
    print(f"r1 + r2 = {r1 + r2}")
    print(f"r1 - r2 = {r1 - r2}")
    print(f"r1 * r2 = {r1 * r2}")
    print(f"r1 / r2 = {r1 / r2}")
    print(f"r1 == r2: {r1 == r2}")
    print(f"r1 != r2: {r1 != r2}")
    print(f"r1 > r2: {r1 > r2}")
    print(f"r1 < r2: {r1 < r2}")
    print(f"r1 >= r2: {r1 >= r2}")
    print(f"r1 <= r2: {r1 <= r2}")
```

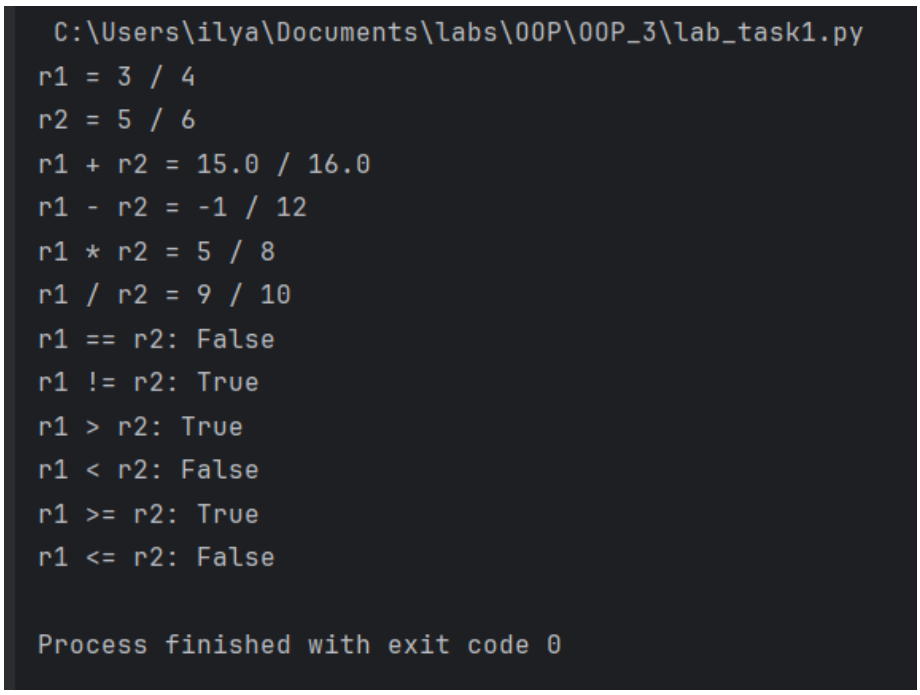

Рисунок 8 – Результат работы программы с новым классом Rational

ыполнить индивидуальные задания (вариант №12):

Задание №1. Выполнить индивидуальное задание №1 лабораторной работы №4.1, максимально задействовав имеющиеся в Python средства перегрузки операторов:

Листинг 2 – Новый код класса Point

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

import math
from typing import Any


def _is_number(value) -> bool:
    """
    Проверяет, можно ли преобразовать значение в число с плавающей запятой.
    :param value: Значение для проверки
    :return: True, если значение является числом, иначе False
    """
    try:
        float(value)
        return True
    except ValueError:
        return False


class Point:
    def __init__(self, first: float = 0.0, second: float = 0.0):
        """
        Инициализирует точку с двумя координатами (first и second).
        :param first: Координата X точки
```

```
        :param second: Координата Y точки
        """
        if not _is_number(first) or not _is_number(second):
            raise ValueError("Координаты должны быть числами!")

        self.first = float(first)
        self.second = float(second)

    def read(self):
        """
        Ввод координат точки с клавиатуры.
        """
        first_input = input('Введите координату X: ')
        if not _is_number(first_input):
            raise ValueError('Координата должна быть числом!')

        second_input = input('Введите координату Y: ')
        if not _is_number(second_input):
            raise ValueError('Координата должна быть числом!')

        self.first = float(first_input)
        self.second = float(second_input)

    def display(self):
        """
        Вывод координат точки на экран.
        """
        print(f'Точка: ({self.first}, {self.second})')

    def distance(self) -> float:
        """
        Вычисляет расстояние от точки до начала координат.
        :return: Расстояние от начала координат до точки
        """
        return math.sqrt(self.first ** 2 + self.second ** 2)

    def __add__(self, other: Any) -> 'Point':
        if isinstance(other, Point):
            return Point(self.first + other.first, self.second +
other.second)
        return NotImplemented

    def __sub__(self, other: Any) -> 'Point':
        if isinstance(other, Point):
            return Point(self.first - other.first, self.second -
other.second)
        return NotImplemented

    def __mul__(self, scalar: float) -> 'Point':
        if _is_number(scalar):
            return Point(self.first * scalar, self.second * scalar)
        return NotImplemented

    def __rmul__(self, scalar: float) -> 'Point':
        return self.__mul__(scalar)

    def __eq__(self, other: Any) -> bool:
        if isinstance(other, Point):
            return self.first == other.first and self.second == other.second
        return NotImplemented

    def __ne__(self, other: Any) -> bool:
        eq = self.__eq__(other)
```

```
        if eq is NotImplemented:
            return NotImplemented
        return not eq

    def __str__(self) -> str:
        return f'Точка({self.first}, {self.second})'

    def __repr__(self) -> str:
        return f'Point(first={self.first}, second={self.second})'


def make_point(first: float, second: float) -> Point:
    """
    Создает объект Point с заданными координатами.
    :param first: Координата X
    :param second: Координата Y
    :return: Объект Point
    :raises ValueError: Если аргументы не являются числами
    """
    if not _is_number(first) or not _is_number(second):
        raise ValueError("Координаты должны быть числами!")
    return Point(first, second)


if __name__ == '__main__':
    # Пример использования класса Point с перегруженными операторами
    try:
        # Создание точек с помощью функции make_point
        point1 = make_point(3.0, 4.0)
        point2 = make_point(1.0, 2.0)
        print("Исходные точки:")
        point1.display()
        point2.display()

        # Сложение точек
        point3 = point1 + point2
        print("\nСумма точек:")
        point3.display()

        # Вычитание точек
        point4 = point1 - point2
        print("\nРазность точек:")
        point4.display()

        # Умножение точки на скаляр
        scalar = 2
        point5 = point1 * scalar
        print(f"\nУмножение точки {point1} на скаляр {scalar}:")
        point5.display()

        # Проверка равенства точек
        print("\nПроверка равенства точек:")
        print(f"point1 == point2: {point1 == point2}")
        print(f"point1 == make_point(3.0, 4.0): {point1 == make_point(3.0,
4.0)}")

        # Ввод координат с клавиатуры для point2
        print("\nВвод новых координат для point2:")
        point2.read()
        point2.display()

        # Вычисление расстояния до начала координат
        distance = point2.distance()
```

```
        print(f'Расстояние точки {point2} до начала координат:
{distance:.2f}')

    except ValueError as e:
        print("Ошибка:", e)
```

```
 C:\Users\ilya\Documents\labs\OOP\OOP_3\indiv_task1.py
Исходные точки:
Точка: (3.0, 4.0)
Точка: (1.0, 2.0)

Сумма точек:
Точка: (4.0, 6.0)

Разность точек:
Точка: (2.0, 2.0)

Умножение точки Точка(3.0, 4.0) на скаляр 2:
Точка: (6.0, 8.0)

Проверка равенства точек:
point1 == point2: False
point1 == make_point(3.0, 4.0): True

Ввод новых координат для point2:
Введите координату X: 12
Введите координату Y: 3
Точка: (12.0, 3.0)
Расстояние точки Точка(12.0, 3.0) до начала координат: 12.37
```

Рисунок 9 – Результат работы нового класса Point

Задание №2. Товарный чек содержит список товаров, купленных покупателем в магазине. Один элемент списка представляет собой пару: товар-сумма. Товар – это класс Goods с полями кода и наименования товара, цены за единицу товара, количества покупаемых единиц товара. В классе должны быть реализованы методы доступа к полям для получения и изменения информации, а также метод вычисления суммы оплаты за товар. Для моделирования товарного чека реализовать класс Receipt, полями которого являются номер товарного чека, дата и время его создания, список покупаемых

товаров. В классе Receipt реализовать методы добавления, изменения и удаления записи о покупаемом товаре, метод поиска информации об определенном виде товара по его коду, а также метод подсчета общей суммы, на которую были осуществлены покупки. Методы добавления и изменения принимают в качестве аргумента объект класса Goods. Метод поиска возвращает объект класса Goods в качестве результата:

Листинг 3 – Код решения индивидуального задания №2

```
from datetime import datetime


class Goods:
    def __init__(self, code, name, unit_price, quantity):
        self._code = code
        self. name = name
        self._unit_price = unit_price
        self._quantity = quantity

    # Property for code
    @property
    def code(self):
        return self._code

    @code.setter
    def code(self, value):
        self._code = value

    # Property for name
    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value

    # Property for unit_price
    @property
    def unit_price(self):
        return self._unit_price

    @unit_price.setter
    def unit_price(self, value):
        self._unit_price = value

    # Property for quantity
    @property
    def quantity(self):
        return self._quantity

    @quantity.setter
    def quantity(self, value):
        self._quantity = value

    # Calculate total price for the goods
    def total_price(self):
```

```
        return self.unit_price * self.quantity

    def __str__(self):
        return (f"Goods(code={self.code}, name={self.name}, "
                f"unit_price={self.unit_price}, quantity={self.quantity})")


class Receipt:
    def __init__(self, number):
        self.number = number
        self.datetime = datetime.now()
        self.goods_list = []

    # Add a Goods object to the receipt
    def add_goods(self, goods):
        self.goods_list.append(goods)

    # Edit an existing Goods object identified by code
    def edit_goods(self, goods):
        for idx, g in enumerate(self.goods_list):
            if g.code == goods.code:
                self.goods_list[idx] = goods
                return True
        return False

    # Delete a Goods object by code
    def delete_goods(self, code):
        for g in self.goods_list:
            if g.code == code:
                self.goods_list.remove(g)
                return True
        return False

    # Search for a Goods object by code
    def search_goods(self, code):
        for g in self.goods_list:
            if g.code == code:
                return g
        return None

    # Calculate the total amount for the receipt
    def total_amount(self):
        return sum(g.total_price() for g in self.goods_list)

    def __str__(self):
        goods_str = "\n".join(str(g) for g in self.goods_list)
        return (f"Receipt Number: {self.number}\nDateTime: {self.datetime}\n"
                f"Goods:\n{goods_str}\nTotal Amount: {self.total_amount()}")


if __name__ == '__main__':
    # Create a receipt
    receipt = Receipt(number=1)

    # Create some goods
    apple = Goods(code=101, name="Apple", unit price=0.5, quantity=10)
    bread = Goods(code=102, name="Bread", unit_price=1.5, quantity=2)
    milk = Goods(code=103, name="Milk", unit_price=0.99, quantity=5)

    # Add goods to receipt
    receipt.add_goods(apple)
    receipt.add_goods(bread)
    receipt.add_goods(milk)
```

```
print("After adding goods:")
print(receipt)

# Edit quantity of bread
bread.quantity = 3
receipt.edit_goods(bread)

print("\nAfter editing bread quantity:")
print(receipt)

# Delete milk from receipt
receipt.delete_goods(103)

print("\nAfter deleting milk:")
print(receipt)

# Search for apple
searched_goods = receipt.search_goods(101)
print("\nSearched Goods:")
print(searched_goods if searched_goods else "Goods not found.")

# Total amount
total = receipt.total_amount()
print(f"\nTotal Amount: {total}")
```

```
C:\Users\ilya\Documents\labs\OOP\OOP_3\indiv_task2.py
After adding goods:
Receipt Number: 1
DateTime: 2024-11-07 16:46:12.893985
Goods:
Goods(code=101, name=Apple, unit_price=0.5, quantity=10)
Goods(code=102, name=Bread, unit_price=1.5, quantity=2)
Goods(code=103, name=Milk, unit_price=0.99, quantity=5)
Total Amount: 12.95

After editing bread quantity:
Receipt Number: 1
DateTime: 2024-11-07 16:46:12.893985
Goods:
Goods(code=101, name=Apple, unit_price=0.5, quantity=10)
Goods(code=102, name=Bread, unit_price=1.5, quantity=3)
Goods(code=103, name=Milk, unit_price=0.99, quantity=5)
Total Amount: 14.45

After deleting milk:
Receipt Number: 1
DateTime: 2024-11-07 16:46:12.893985
Goods:
Goods(code=101, name=Apple, unit_price=0.5, quantity=10)
Goods(code=102, name=Bread, unit_price=1.5, quantity=3)
Total Amount: 9.5

Searched Goods:
Goods(code=101, name=Apple, unit_price=0.5, quantity=10)

Total Amount: 9.5
```

Рисунок 10 – Пример выполнения программы индивидуального задания №2

охраним изменения в git и выложим в сервис GitHub:

```
(venv) PS C:\Users\Ilya\Documents\labs\OOP\OOP_3> git log --oneline
0b50417 (HEAD -> develop) final changes
3ecaf5c (origin/main, origin/HEAD, main) Initial commit
(venv) PS C:\Users\Ilya\Documents\labs\OOP\OOP_3>
```

Рисунок 11 – История коммитов

```
(venv) PS C:\Users\Ilya\Documents\labs\OOP\OOP_3> git checkout main
 10 files changed, 512 insertions(+), 1 deletion(-)
 create mode 100644 .idea/.gitignore
 create mode 100644 .idea/OOP_3.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 indiv_task1.py
 create mode 100644 indiv_task2.py
 create mode 100644 lab_task1.py
```

Рисунок 12 – Переключение на ветку main и объединение с веткой develop

```
(venv) PS C:\Users\Ilya\Documents\labs\OOP\OOP_3> git push origin main
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 12 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (14/14), 5.44 KiB | 2.72 MiB/s, done.
Total 14 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/daxstrong/OOP_3.git
   3ecaf5c..0b50417  main -> main
(venv) PS C:\Users\Ilya\Documents\labs\OOP\OOP_3>
```

Рисунок 13 – Отправка изменений на удаленный репозиторий

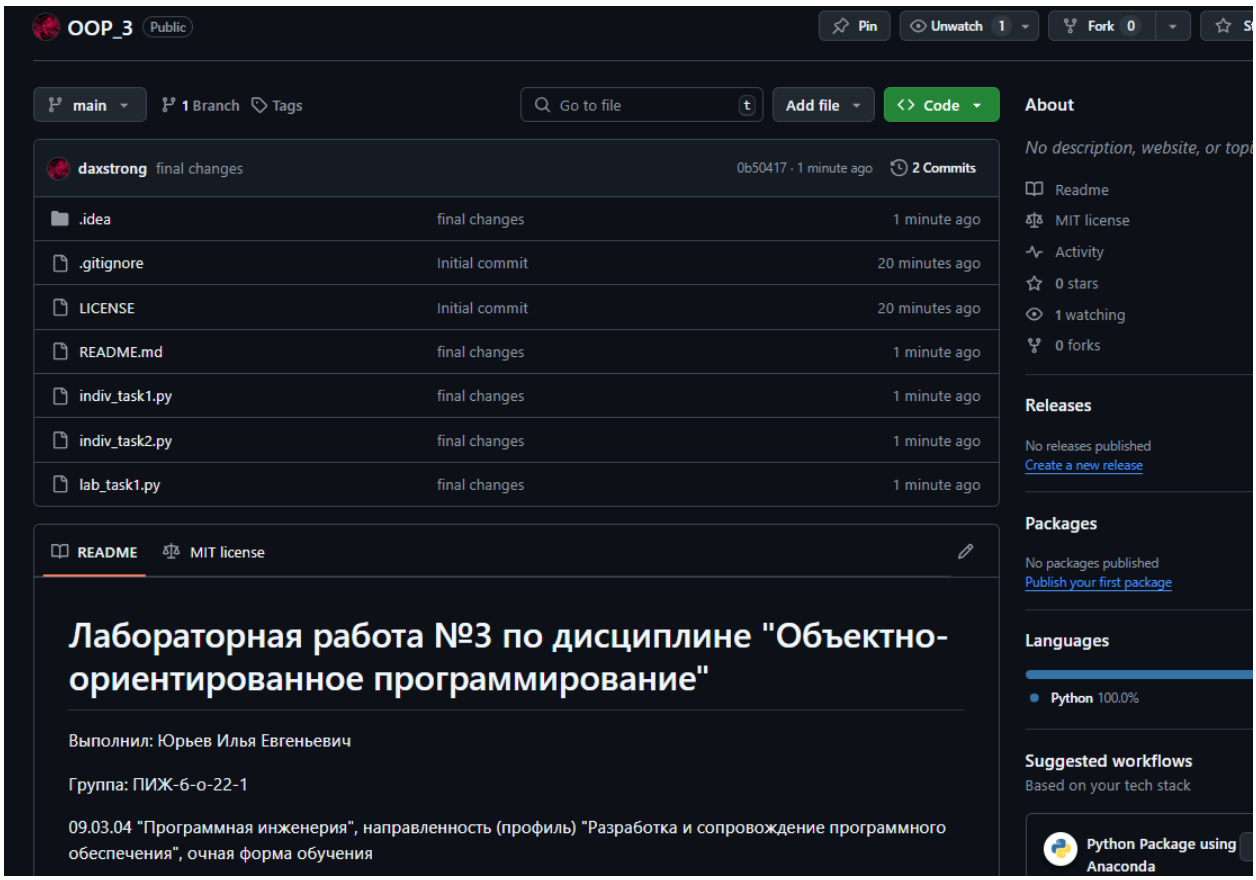

Рисунок 14 – Обновленный репозиторий

Вывод: приобрели навыки по написанию методов перегрузок операторов при написании различных программ с помощью языка программирования Python версии 3.x.

Ответы на контрольные вопросы:

акие средства существуют в Python для перегрузки операций?

Ответ: в Python для перегрузки операций существуют специальные методы, которые начинаются и заканчиваются двойными подчеркиваниями. Например, методы таких классов, как «__add__», «__sub__», «__mul__» и т.д., предназначены для перегрузки арифметических операций, в то время как методы, такие как «__lt__», «__gt__», «__eq__» и других, перегружают операции сравнения.

акие существуют методы для перегрузки арифметических операций и операций отношения в языке Python?

Ответ: метод «__add__» перегружает оператор сложения («+»), а «__truediv__» - оператор деления («/»). Для операций сравнения используются такие методы, как «__lt__» для "меньше чем" («<«), «__gt__» для "больше чем" («>«), «__eq__» для "равно" («==«) и другие.

каких случаях будут вызваны следующие методы: __add__ , __iadd__ и

Ответ: метод «__add__» вызывается, когда используется оператор сложения. Например, при выполнении «a + b» будет вызван «a.__add__(b)». Метод «__iadd__», отвечающий за оператор «+=«, вызывается при выполнении такой операции, как «a += b», он изменяет объект «a» на месте. Метод «__radd__» вызывается, когда левый операнд не поддерживает операцию, например, если «b» не может быть сложен с «a», будет вызван

ля каких целей предназначен метод __new__ ? Чем он отличается от метода

Ответ: метод «__new__» отвечает за создание нового экземпляра класса, в то время как метод «__init__» предназначен для инициализации этого экземпляра после его создания. Метод «__new__» вызывается первым и может использоваться для контролирования процесса создания объекта, например, для создания объектов неизменяемых типов (таких как строки или кортежи).

ем отличаются методы __str__ и __repr__ ?

Ответ: методы «__str__» и «__repr__» служат для представления объектов в виде строк, но предназначены для разных целей. Метод «__str__» должен возвращать строку, удобную для пользователя, предоставляя "красивое" представление объекта. Метод «__repr__» предназначен для создателя, и его цель — вернуть строку, которая может помочь в отладке и, желательно, быть валидным кодом Python, позволяя воссоздать объект.