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 №3
дисциплины «Объектно-ориентированное программирование»

Выполнил:
Плугатырев Владислав Алексеевич
3 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и
сопровождение программного
обеспечения», очная форма обучения

_____
(подпись)

Доцент кафедры инфокоммуникаций:
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Элементы объектно-ориентированного программирования в языке Python.

Цель работы: приобретение навыков по работе с классами и объектами при написании программ с помощью языка программирования Python версии 3.x.

Ход работы

1. Создание репозитория.

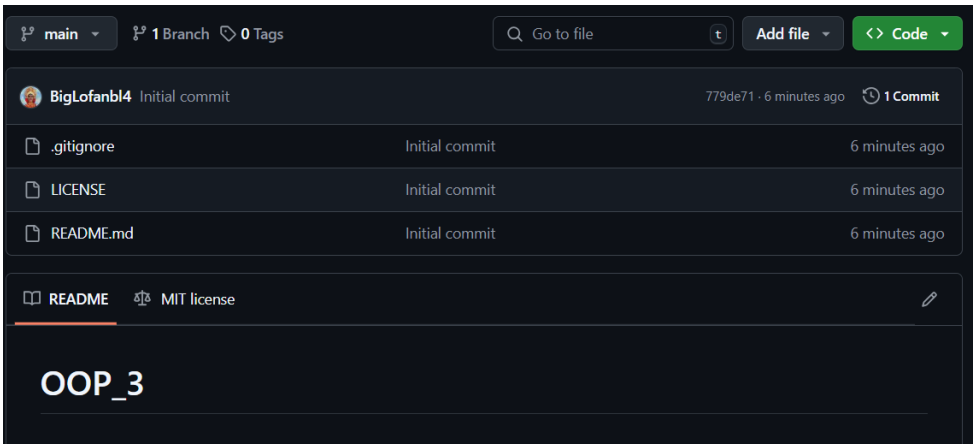

Рисунок 1.1 – Созданный репозиторий

2. Выполнение примеров из лабораторной работы.

```python
class Rational:
    def __init__(self, a=0, b=1):
        a = int(a)
        b = int(b)
        if b == 0:
            raise ValueError("Illegal value of the denominator")
        self.__numerator = a
        self.__denominator = b
        self.__reduce()

    # Сокращение дроби.
    def __reduce(self):
        # Функция для нахождения наибольшего общего делителя
        def gcd(a, b):
            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
```

Рисунок 1.2 – Код примера

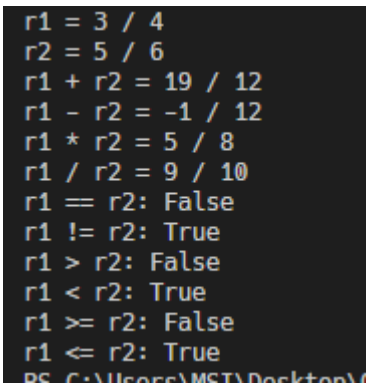

Рисунок 1.3 – Результат работы примера

3. Выполнение первого задания: выполнить индивидуальное задание 1 лабораторной работы 4.1, максимально задействовав имеющиеся в Python средства перегрузки операторов.

```
class RightTriangle:
    def __init__(self, first, second):
        if not isinstance(first, (int, float)) or not isinstance(
            second, (int, float)
        ):
            raise ValueError("The first and second inputs must be numbers")
        if first <= 0 or second <= 0:
            raise ValueError("The numbers must be > 0")
        self.first = first
        self.second = second

    def read(self):
        try:
            first = float(input("Enter first value > 0: "))
            second = float(input("Enter second value > 0: "))
            if first <= 0 or second <= 0:
                raise ValueError("The numbers must be > 0")
            self.first = first
            self.second = second
        except ValueError as error:
            print(error)
            self.read()

    # заменили метод display на __str__
    def __str__(self):
        return f"The catheter a={self.first}, the catheter b={self.second}"

    # заменили метод hypotenuse на __call__
    def __call__(self):
        return math.sqrt(self.first**2 + self.second**2)
```

Рисунок 1.4 – Код программы

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

import unittest

from ind1 import RightTriangle


class RightTriangleTest(unittest.TestCase):
    def test_valid_initialization(self):
        triangle = RightTriangle(4, 6)
        self.assertEqual(triangle.first, 4)
        self.assertEqual(triangle.second, 6)

    def test_invalid_initialization(self):
        self.assertRaises(ValueError, RightTriangle, -1, 4)
        self.assertRaises(ValueError, RightTriangle, 4, -1)
        self.assertRaises(ValueError, RightTriangle, "bruh", 4)
        self.assertRaises(ValueError, RightTriangle, 4, "bruh")

    def test_hypotenuse(self):
        triangle = RightTriangle(3, 4)
        self.assertAlmostEqual(triangle(), 5.0)


if __name__ == "__main__":
    unittest.main()
```

Рисунок 1.5 – Юниттест задания

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```

Рисунок 1.6 – Результат тестирования

```
The catheter a=3, the catheter b=4
5.0
Enter first value > 0: 4
Enter second value > 0: 5
The catheter a=4.0, the catheter b=5.0
6.4031242374328485
```

Рисунок 1.7 – Результат работы программы

4. Выполнение второго задания: Информационная запись о книге в библиотеке содержит следующие поля: автор, название, год издания, издательство, цена. Для моделирования учетной карточки абонента реализовать класс Subscriber, содержащий фамилию абонента, его библиотечный номер и список взятых в библиотеке книг. Один элемент списка состоит из информационной записи о книге, даты выдачи, требуемой даты возврата и признака возврата. Реализовать методы добавления книг в список и удаления книг из него; метод поиска книг, подлежащих возврату; методы поиска по автору, издательству и году издания; метод вычисления стоимости всех подлежащих возврату книг. Реализовать операцию слияния двух учетных карточек, операцию пересечения и вычисления разности. Реализовать операцию генерации конкретного объекта Debt (долг), содержащего список книг, подлежащих возврату из объекта типа Subscriber.

```python
class Subscriber:
    MAX_BOOKS = 100

    def __init__(self, surname, id, books=None, size=MAX_BOOKS):
        if books is None:
            books = []
        if len(books) > size:
            raise ValueError("Amount of books > max size")
        self.surname = surname
        self.id = id
        self.books = books
        self.__size = size
        self.__count = len(books)

    @property
    def size(self):
        return self.__size

    @property
    def count(self):
        return self.__count

    def add_book(self, book, issue_date, return_date):
        if self.__count < self.__size:
            self.books.append(
                {
                    "book": book,
                    "issue_date": issue_date,
                    "return_date": return_date,
                    "returned": False,
                }
            )
            self.__count += 1
        else:
            print("Reached maximum size")

    def remove_book(self, book):
        for item in self.books:
            if item["book"] == book:
                self.books.remove(item)
                self.__count -= 1
                return

    def books_to_return(self):
        return [i for i in self.books if not i["returned"]]
```

Рисунок 1.8 – Код программы

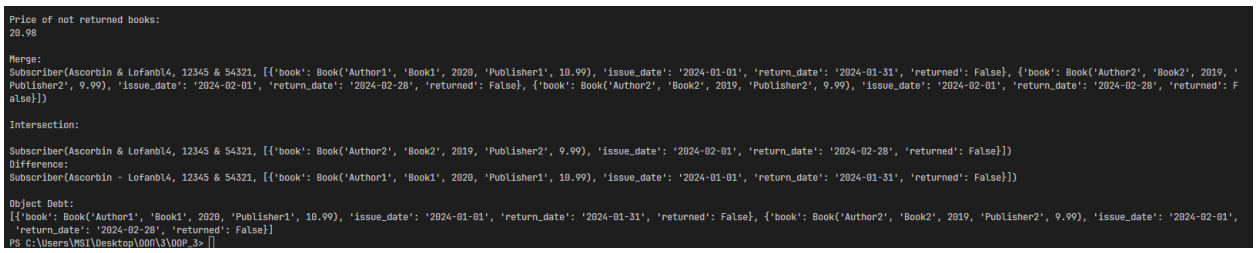

Рисунок1.9 – Результат работы программы

```python
class TestSubscriber(unittest.TestCase):
    def setUp(self):
        self.book1 = Book("Author1", "Book1", 2020, "Publisher1", 10.99)
        self.book2 = Book("Author2", "Book2", 2019, "Publisher2", 9.99)

        self.subscriber1 = Subscriber("Ascorbin", "12345")
        self.subscriber2 = Subscriber("Lofanbl4", "54321")

    def test_initialization(self):
        self.assertEqual(self.subscriber1.surname, "Ascorbin")
        self.assertEqual(self.subscriber1.id, "12345")
        self.assertEqual(self.subscriber1.size, Subscriber.MAX_BOOKS)
        self.assertEqual(self.subscriber1.count, 0)

    def test_add_book(self):
        self.subscriber1.add_book(self.book1, "2024-01-01", "2024-01-31")
        self.assertEqual(self.subscriber1.count, 1)

    def test_remove_book(self):
        self.subscriber1.add_book(self.book1, "2024-01-01", "2024-01-31")
        self.subscriber1.remove_book(self.book1)
        self.assertEqual(self.subscriber1.count, 0)

    def test_books_to_return(self):
        self.subscriber1.add_book(self.book1, "2024-01-01", "2024-01-31")
        self.subscriber1.add_book(self.book2, "2024-02-01", "2024-02-28")
        books_to_return = self.subscriber1.books_to_return()
        self.assertEqual(len(books_to_return), 2)

    def test_find_by_author(self):
        self.subscriber1.add_book(self.book1, "2024-01-01", "2024-01-31")
        books_by_author = self.subscriber1.find_by_author("Author1")
        self.assertEqual(len(books_by_author), 1)
```

Рисунок 2.0 – Юниттест программы

```
test_add_book (__main__.TestSubscriber.test_add_book) ... ok
test_add_subscribers (__main__.TestSubscriber.test_add_subscribers) ... ok
test_and_subscribers (__main__.TestSubscriber.test_and_subscribers) ... ok
test_books_to_return (__main__.TestSubscriber.test_books_to_return) ... ok
test_find_by_author (__main__.TestSubscriber.test_find_by_author) ... ok
test_find_by_publisher (__main__.TestSubscriber.test_find_by_publisher) ... ok
test_find_by_year (__main__.TestSubscriber.test_find_by_year) ... ok
test_generate_debt (__main__.TestSubscriber.test_generate_debt) ... ok
test_get_item (__main__.TestSubscriber.test_get_item) ... ok
test_initialization (__main__.TestSubscriber.test_initialization) ... ok
test_price_books_to_return (__main__.TestSubscriber.test_price_books_to_return) ... ok
test_remove_book (__main__.TestSubscriber.test_remove_book) ... ok
test_sub_subscribers (__main__.TestSubscriber.test_sub_subscribers) ... ok

----------------------------------------------------------------------
Ran 13 tests in 0.001s

OK
```

Рисунок 2.1 – Результаты теста

Ответы на контрольные вопросы

1. Средства перегрузки операций в Python: В Python перегрузка операций реализуется через специальные методы, часто называемые "магическими" методами. Эти методы имеют двойные подчеркивания в начале и конце их имен (например, «__add__» для операции сложения, «__eq__» для операции равенства).

2. Методы перегрузки арифметических операций и операций отношения: - «Арифметические операции»: «__add__» (сложение), «__sub__» (вычитание), «__mul__» (умножение), «__truediv__» (деление), «__floordiv__» (целочисленное деление), «__mod__» (модуль), «__pow__» (возведение в степень), «__neg__» (отрицание), «__pos__» (унарный плюс). - «Операции отношения»: «__eq__» (равенство), «__ne__» (неравенство), «__lt__» (меньше), «__le__» (меньше или равно), «__gt__» (больше), «__ge__» (больше или равно).

3. Случаи вызова «__add__», «__iadd__», и «__radd__»: - «__add__(self, other)»: Вызывается, когда объект класса находится в левой части операции сложения (например, «a + b»). - «__iadd__(self, other)»: Вызывается для реализации операции сложения с присваиванием (например, «a += b»). Обычно метод должен возвращать «self» после модификации. - «__radd__(self, other)»: Вызывается, когда объект класса находится в правой части операции сложения и левый операнд не поддерживает сложение с правым (например, «b + a», где «b» не имеет метода «__add__» для работы с «a»).

4. Назначение метода «__new__» и его отличия от «__init__»: «__new__» – это статический метод, который вызывается для создания нового экземпляра класса перед «__init__». Этот метод возвращает новый экземпляр класса (новый объект). Он обычно используется для изменения процесса создания объекта у неизменяемых типов (как «tuple», «str») или при реализации паттерна Singleton. «__init__» – метод, который вызывается после создания объекта (через «__new__») для его инициализации с начальными значениями.

5. Отличия между «__str__» и «__repr__»: «__str__» – возвращает строковое представление объекта, предназначенное для отображения пользователю. Более "дружественное" представление. «__repr__» – возвращает официальное строковое представление объекта, которое должно быть максимально точным. Идеально, чтобы результат этого метода можно было использовать для воссоздания объекта при его вызове через «eval()». «__str__» больше ориентирован на конечного пользователя, в то время как «__repr__» предназначен для программистов. Если «__str__» не определен, Python в качестве запасного варианта использует «__repr__».