

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 Перегрузка операторов в языке Python.

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

Ход работы.

1. Создание нового репозитория с лицензией MIT.

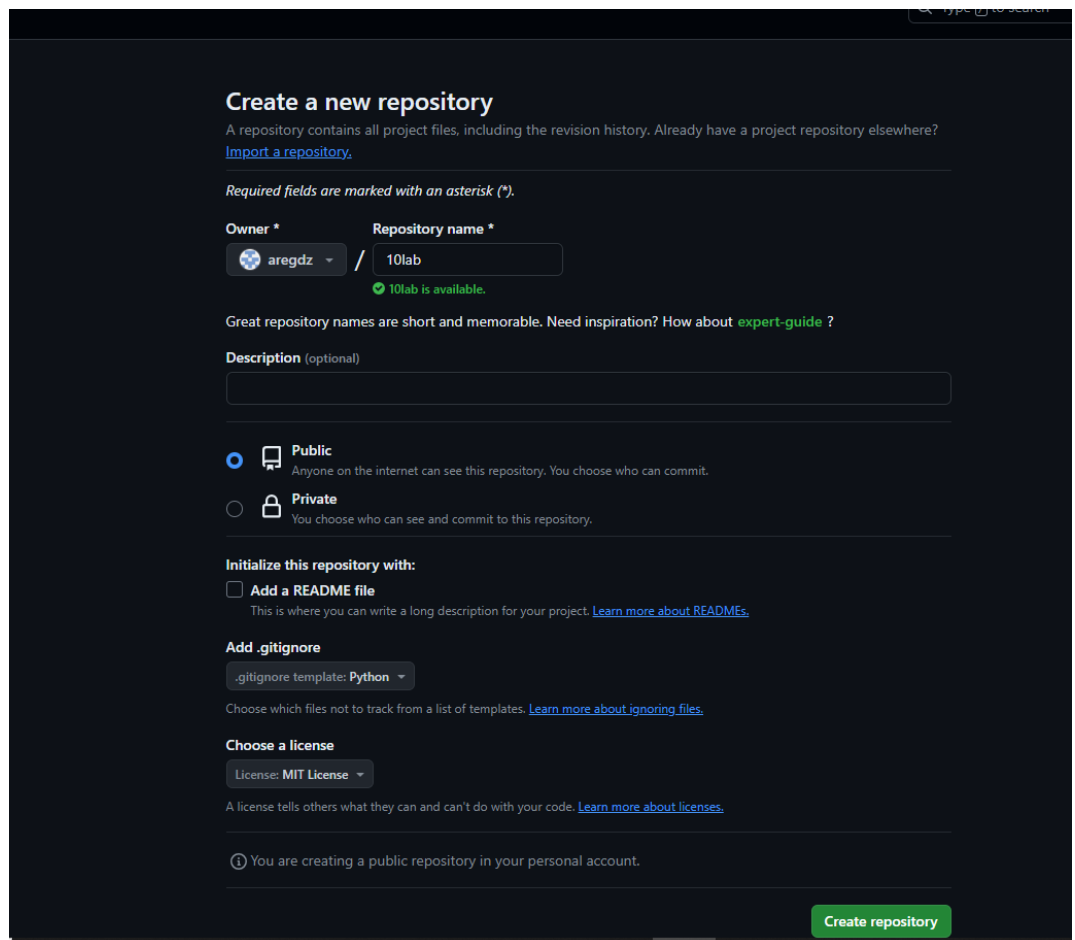


Рисунок 1 – создание репозитория

2. Клонировал репозиторий на рабочий ПК.

```

aregd@DESKTOP-5KV9QA9 MINGW64 ~
$ cd "D:\Рабочий стол\4 семестр\опи\10"

aregd@DESKTOP-5KV9QA9 MINGW64 /d/Рабочий стол/4 семестр/опи/10
$ git clone https://github.com/aregdz/10lab.git
Cloning into '10lab'...
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/10
$ cd 10lab

aregd@DESKTOP-5KV9QA9 MINGW64 /d/Рабочий стол/4 семестр/опи/10/10lab (main)
$ |

```

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/10/10lab (main)
$ git checkout -b develop
Switched to a new branch 'develop'

aregd@DESKTOP-5KV9QA9 MINGW64 /d/Рабочий стол/4 семестр/опи/10/10lab (develop)
$ Sss
```

Рисунок 4 – организация ветки

```
(venv) PS D:\Рабочий стол\4 семестр\опи\10\10lab
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("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

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

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

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

    @property
    def denominator(self):
        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()

    # Привести дробь к строке.
    def __str__(self):
        return f"{self.__numerator} / {self.__denominator}"

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

    # Привести дробь к вещественному значению.
    def __float__(self):
        return self.__numerator / self.__denominator

    # Привести дробь к логическому значению.
    def __bool__(self):
        return self.__numerator != 0

    # Сложение обыкновенных дробей.
    def __iadd__(self, rhs): # +=
        if isinstance(rhs, Rational):
            a = self.numerator * rhs.denominator + \
                self.denominator * rhs.numerator
            b = self.denominator * rhs.denominator
            self.__numerator, self.__denominator = a, b
            self.__reduce()
            return self
        else:
            raise ValueError("Illegal type of the argument")

    def __add__(self, rhs): # +
        return self.__clone().__iadd__(rhs)

```

```

def __isub__(self, rhs): # -=
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator - \
            self.denominator * rhs.numerator
        b = self.denominator * rhs.denominator
        self.__numerator, self.__denominator = a, b
        self.__reduce()
        return self
    else:
        raise ValueError("Illegal type of the argument")

def __sub__(self, rhs): # -
    return self.__clone().__isub__(rhs)

def __imul__(self, rhs): # *=
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.numerator
        b = self.denominator * rhs.denominator
        self.__numerator, self.__denominator = a, b
        self.__reduce()
        return self
    else:
        raise ValueError("Illegal type of the argument")

def __mul__(self, rhs): # *
    return self.__clone().__imul__(rhs)

def __itruediv__(self, rhs): # /=
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator
        b = self.denominator * rhs.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 __truediv__(self, rhs): # /
    return self.__clone().__itruediv__(rhs)

def __eq__(self, rhs): # ==
    if isinstance(rhs, Rational):
        return (self.numerator == rhs.numerator) and \
            (self.denominator == rhs.denominator)
    else:
        return False

def __ne__(self, rhs): # !=
    if isinstance(rhs, Rational):
        return not self.__eq__(rhs)
    else:
        return False

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

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

def __ge__(self, rhs): # >=
    if isinstance(rhs, Rational):
        return not self.__lt__(rhs)
    else:

```

```

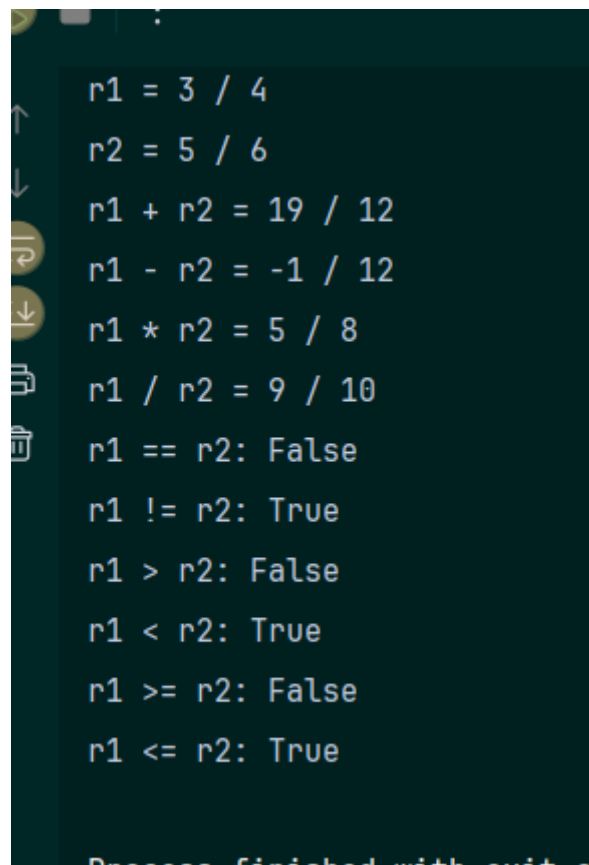
        return False

    def __le__(self, rhs): # <=
        if isinstance(rhs, Rational):
            return not self.__gt__(rhs)
        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 6 – пример 1



```

r1 = 3 / 4
r2 = 5 / 6
r1 + r2 = 19 / 12
r1 - r2 = -1 / 12
r1 * r2 = 5 / 8
r1 / r2 = 9 / 10
r1 == r2: False
r1 != r2: True
r1 > r2: False
r1 < r2: True
r1 >= r2: False
r1 <= r2: True
Process finished with exit co

```

Рисунок 7 – выполнение примера 1

5. Выполнить индивидуальное задание 1 лабораторной работы 4.1, максимально задействовав имеющиеся в Python средства перегрузки операторов.

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

class Pair:
    def __init__(self, first, second):
        self.set_first(first)
        self.set_second(second)

    def set_first(self, value):
        if isinstance(value, int) and value > 0:
            self.first = value
        else:
            raise ValueError("НЕ правильное значение.")

    def set_second(self, value):
        if isinstance(value, float) and value > 0:
            self.second = value
        else:
            raise ValueError("НЕ правильное значение.")

    def read(self):
        first = int(input("Введите целое положительное число для 'first': "))
        second = float(
            input("Введите дробное положительное число для 'second': ")
        )
        self.set_first(first)
        self.set_second(second)

    def display(self):
        print(
            f"Калорийность 100 г продукта: {self.first} ккал,\nМасса продукта: {self.second} кг."
        )

    def power(self):
        return self.first * self.second * 10

def make_pair(first, second):
    try:
        pair = Pair(first, second)
        return pair
    except ValueError as ve:
        print(ve)
        return None

if __name__ == "__main__":
    try:
        first = int(input("Введите калорийность 100 г продукта: "))
        second = float(input("Введите массу продукта в килограммах: "))
        my_pair = make_pair(first, second)
        if my_pair:
            my_pair.display()
            print(f"Общая калорийность продукта: {my_pair.power()} ккал.")
    except Exception as e:
        print("Введены некорректные данные.")
```

Рисунок 8 – Выполнение задания 1



```
"D:\Рабочий стол\4 семестр\опи\9\9lab\venv\Scripts
Введите калорийность 100 г продукта: 455
Введите массу продукта в килограммах: 7
Калорийность 100 г продукта: 455 ккал,
Масса продукта: 7.0 кг.
Общая калорийность продукта: 31850.0 ккал.

Process finished with exit code 0
```

Рисунок 9 – пример выполнение задания 1

6. Создать класс Fraction для работы с беззнаковыми дробными десятичными числами. Число должно быть представлено двумя списками типа int: целая и дробная часть, каждый элемент — десятичная цифра. Для целой части младшая цифра имеет меньший индекс, для дробной части старшая цифра имеет меньший индекс (десятые — в нулевом элементе, сотые — в первом, и т. д.). Реальный размер списков задается как аргумент конструктора инициализации. Реализовать арифметические операции сложения, вычитания и умножения, и операции сравнения.

```
class Fraction:
    MAX_SIZE = 100

    def __init__(self, size):
        if size > Fraction.MAX_SIZE:
            raise ValueError("Введите размер меньше:")
        self.integer_part = [0] * size
        self.decimal_part = [0] * size
        self.size = size
        self.count = 0

    def get_size(self):
        return self.size

    def get_count(self):
        return self.count

    def __add__(self, other):
        result = Fraction(self.size)
        carry = 0
```

```

        for i in range(self.size - 1, -1, -1):
            total = self.integer_part[i] + other.integer_part[i] + carry
            result.integer_part[i] = total % 10
            carry = total // 10

        # Начнем сложение дробной части с самого начала списка
        for i in range(self.size):
            total = self.decimal_part[i] + other.decimal_part[i] + carry
            result.decimal_part[i] = total % 10
            carry = total // 10

        return result

def __sub__(self, other):
    result = Fraction(self.size)
    borrow = 0

    for i in range(self.size - 1, -1, -1):
        sub = self.integer_part[i] - other.integer_part[i] - borrow
        if sub < 0:
            borrow = 1
            sub += 10
        else:
            borrow = 0
        result.integer_part[i] = sub

    # Вычитаем дробные части
    for i in range(self.size):
        sub = self.decimal_part[i] - other.decimal_part[i] - borrow
        if sub < 0:
            borrow = 1
            sub += 10
        else:
            borrow = 0
        result.decimal_part[i] = sub

    return result

# Пропущено умножение и сравнение...

def __str__(self):
    integer_str = ''.join(map(str, reversed(self.integer_part)))
    decimal_str = ''.join(map(str, self.decimal_part))
    return f"{integer_str}.{decimal_str}"

# Пример использования
fraction1 = Fraction(5)
fraction1.integer_part = [1, 2, 3, 4, 5]
fraction1.decimal_part = [6, 7, 8, 9, 0]
fraction2 = Fraction(5)
fraction2.integer_part = [5, 4, 3, 2, 1]
fraction2.decimal_part = [0, 9, 8, 7, 6]

result_add = fraction1 + fraction2
result_sub = fraction1 - fraction2
print(f"Addition: {result_add}")
print(f"Subtraction: {result_sub}")

```

Рисунок 10 – выполнение задания 2

```
run 2zadanie x

"П\Рабочий стол\4 семестр\опи\10\10lab\venv\
Services Alt+8 ,6.66777

Результат умножения:: 42085.58914

Process finished with exit code 0
```

Рисунок 11 – пример выполнения задания 2

```
3 files reformatted.

isort.....
[develop f61977a] 2
14 files changed, 411 insertions(+)
create mode 100644 .flake8
create mode 100644 .idea/.gitignore
create mode 100644 .idea/10lab.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 1zadanie.py
create mode 100644 2zadanie.py
create mode 100644 README (1).md
create mode 100644 environment.yml
create mode 100644 primer1.py
create mode 100644 pyproject.toml
(venv) PS D:\Рабочий стол\4 семестр\опи\10\10lab>

0lab > 2zadanie.py
```

Рисунок 12 - фиксация изменений

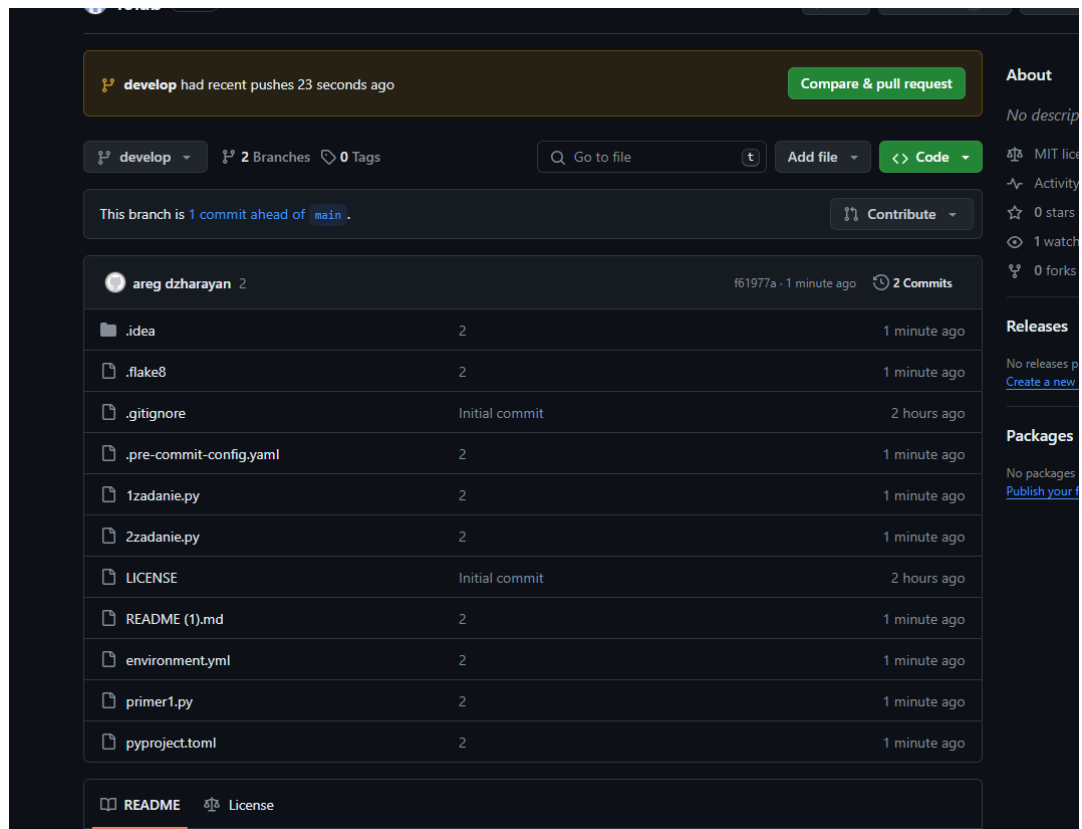


Рисунок 13 – ветка develop в git

Контрольные вопросы:

1. В Python для перегрузки операций существуют специальные методы, так называемые "магические методы" или "специальные методы". Они начинаются и заканчиваются двумя подчеркиваниями (например, `__add__`, `__sub__`, `__mul__` и т. д.). Эти методы позволяют определить поведение объектов при выполнении стандартных операций, таких как сложение, вычитание, умножение и другие.

2. Для перегрузки арифметических операций в Python используются следующие методы:

`__add__`: сложение (+)

`__sub__`: вычитание (-)

`__mul__`: умножение (\*)

`__truediv__`: деление (/)

`__floordiv__`: целочисленное деление (//)

`__mod__`: остаток от деления (%)

`__pow__`: возведение в степень (\*\*)

Для перегрузки операций отношения используются методы:

`__eq__`: равенство (==)

`__ne__`: неравенство (!=)

`__lt__`: меньше (<)

`__le__`: меньше или равно (<=)

`__gt__`: больше (>)

`__ge__`: больше или равно (>=)

3.Метод `__add__` вызывается при выполнении операции сложения (+). Метод `__iadd__` вызывается при выполнении операции +=, когда объект изменяется на месте. Метод `__radd__` вызывается, когда операнд справа от операции сложения (+) не поддерживает эту операцию и при этом первый операнд поддерживает эту операцию.

4.Метод `__new__` предназначен для создания нового экземпляра класса. Он вызывается перед методом `__init__`. `__new__` используется для создания экземпляра класса с изменяемыми атрибутами до их инициализации в методе `__init__`. Метод `__new__` принимает класс в качестве первого аргумента и возвращает новый экземпляр класса. Отличие от метода `__init__` заключается в том, что `__new__` создает объект, а `__init__` инициализирует его.

5.Методы `__str__` и `__repr__` используются для представления объекта в виде строк. `__str__` возвращает "официальную" строковую версию объекта, которая используется для отображения объекта в принятой человеком форме. `__repr__` возвращает строковое представление объекта, которое можно использовать для его воссоздания. Оба метода возвращают строку, но `__str__` предназначен для конечного пользователя, в то время как `__repr__` предназначен для разработчика.