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

Институт цифрового развития
Кафедра инфокоммуникаций

«Перегрузка операторов в языке Python»

ОТЧЕТ
по лабораторной работе №4.2
дисциплины
«Основы программной инженерии»

Выполнил:
Мизин Глеб Егорович
2 курс, группа ПИЖ-б-о-21-1,
011.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

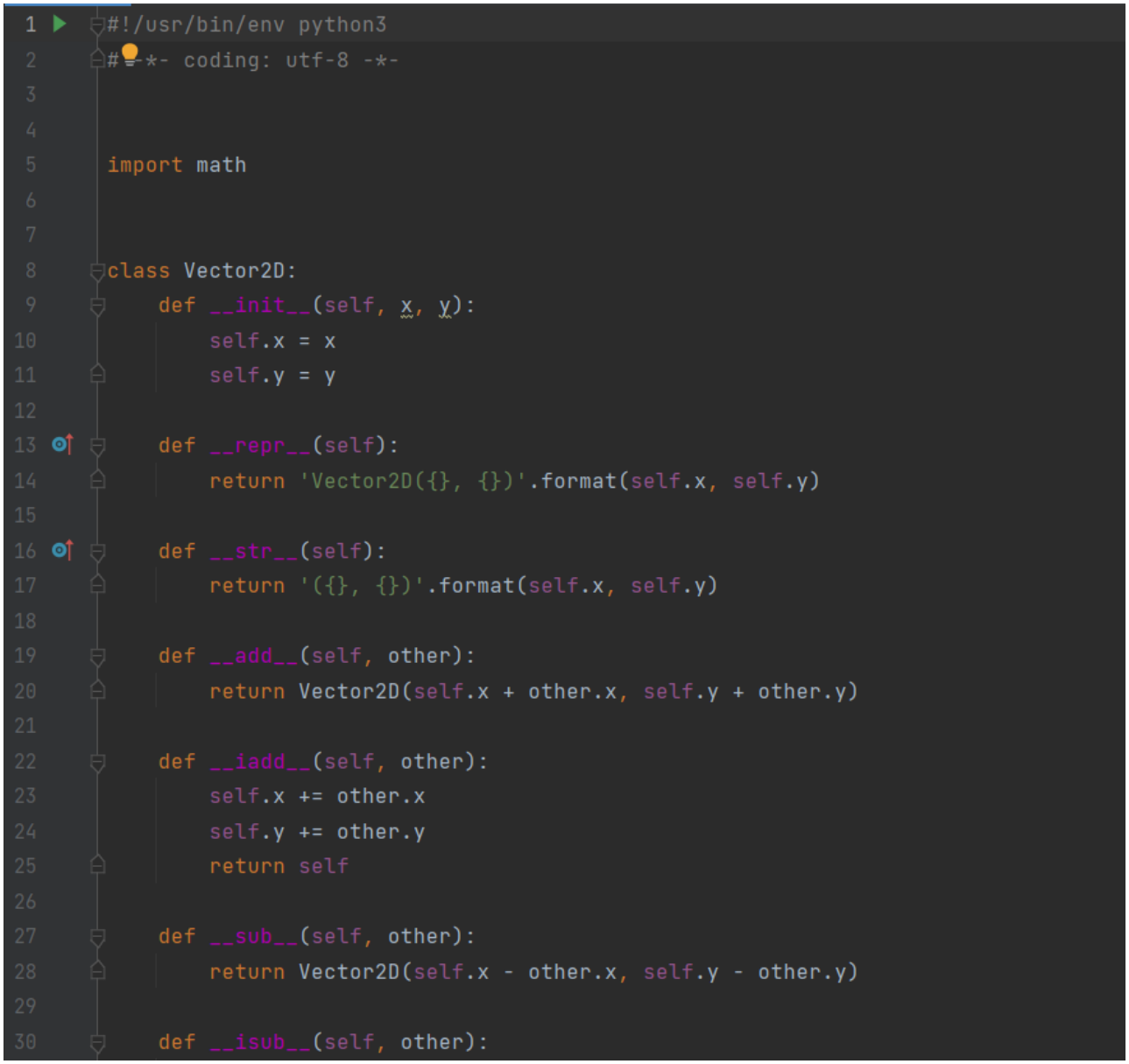
_____
(подпись)

Проверил:
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

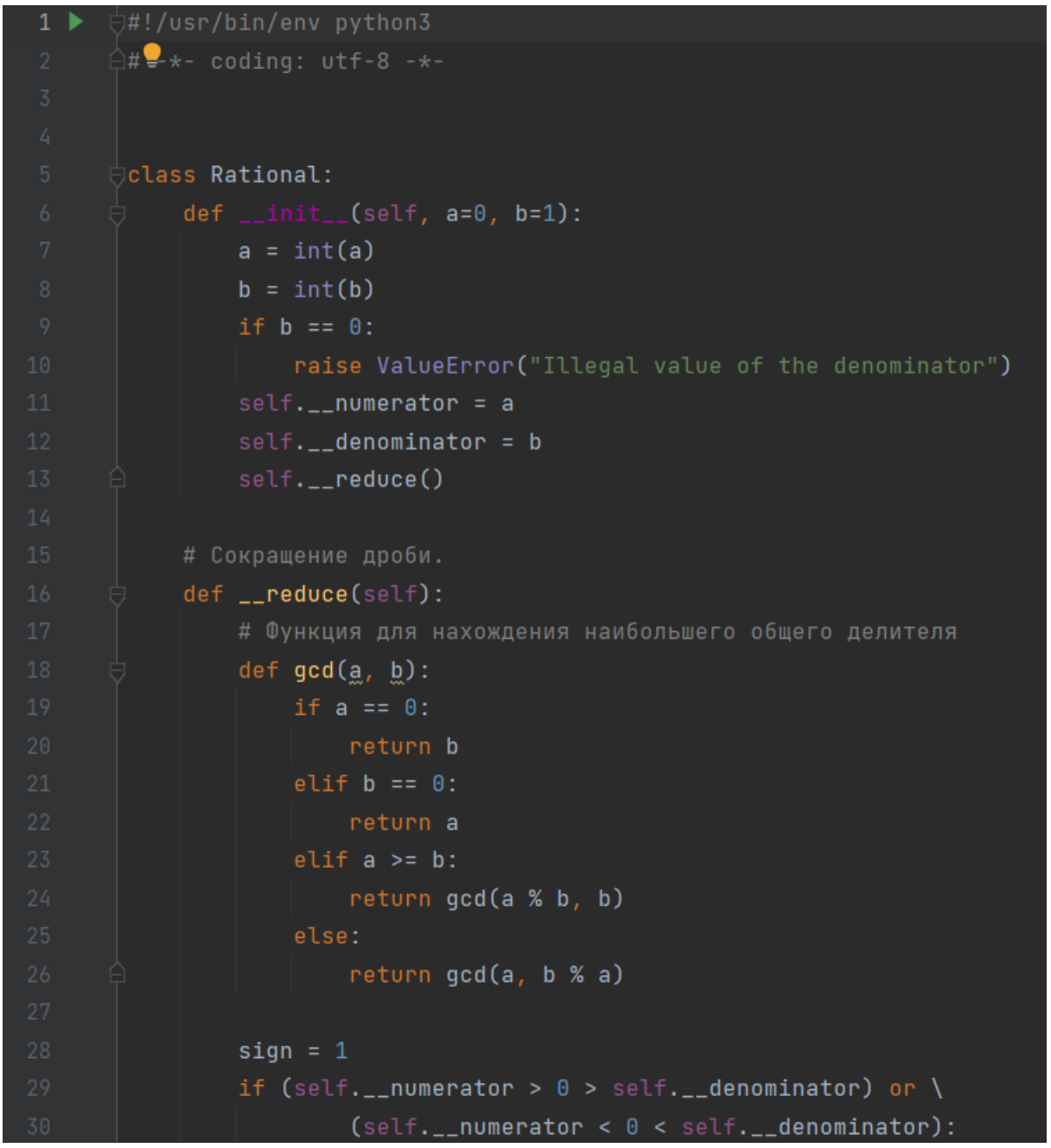
Ставрополь, 2023 г.

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



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


import math


class Vector2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return 'Vector2D({}, {})'.format(self.x, self.y)

    def __str__(self):
        return '({}, {})'.format(self.x, self.y)

    def __add__(self, other):
        return Vector2D(self.x + other.x, self.y + other.y)

    def __iadd__(self, other):
        self.x += other.x
        self.y += other.y
        return self

    def __sub__(self, other):
        return Vector2D(self.x - other.x, self.y - other.y)

    def __isub__(self, other):
```

Рисунок 1 – Пример 1



```python
#!/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 > self.__denominator) or \
                (self.__numerator < 0 < self.__denominator):
```

Рисунок 2 – Пример 2

```
def is_number(a):
    try:
        float(a)
    except ValueError:
        return False
    return True


class Coordinates:
    # Конструктор класса.
    def __init__(self, first=0.0, second=0.0):
        if is_number(first) and is_number(second):
            if first > 0 and second > 0:
                self.first = first
                self.second = second
        else:
            raise ValueError

    def read(self):
        self.first = float(input("Enter the first value: "))
        self.second = float(input("Enter the second value: "))

    # Переопределение метода, вызываемого при использовании оператора in для объекта класса.
    # Возвращает True, если значение x входит в интервал [self.first, self.second),
    # и False в противном случае
    def __contains__(self, x):
        return self.first <= x < self.second

    # Метод, который выводит на экран сообщение о том, принадлежит ли число x интервалу
```

Рисунок 3 – Задание 1

```python
class Money:
    # константа для максимального размера списка словарей
    MAX_SIZE = 10

    # конструктор класса
    def __init__(self, n=None):
        self.count = 0
        self._size = Money.MAX_SIZE
        self.money_list = []  # список словарей

        # если передана строка, инициализируем список словарей соответствующим образом
        # isinstance - Позволяет проверить принадлежность экземпляра к классу.
        if isinstance(n, str):
            denominations = n.split(", ")
            for d in denominations:
                money = d.split(": ")
                if money[0].isnumeric() and money[1].isnumeric():
                    denomination = int(money[0])
                    number = int(money[1])
                    if self.count < self._size:
                        self.money_list.append({"denomination": str(denomination), "number": number})
                        self.count += 1

        # иначе инициализируем список словарей нулями
        else:
            for i in range(self.count):
                self.money_list.append({"denomination": str(i), "number": 0})

        # сортировка списка по номиналу
        self.money_list = sorted(self.money_list, key=lambda d: int(d['denomination']))
```

Рисунок 4 – Задание 2

Контрольные вопросы

1. Какие средства существуют в Python для перегрузки операций?

Для перегрузки операций в Python используются специальные методы, которые начинаются и заканчиваются двойным подчеркиванием. Например, для перегрузки оператора сложения используется метод add, для оператора равенства - метод eq и т.д.

2. Какие существуют методы для перегрузки арифметических операций и операций отношения в языке Python?

Перегрузка арифметических операторов

- `__add__(self, other)` - сложение. x + y вызывает `x.__add__(y)`.
- `__sub__(self, other)` - вычитание (x - y).
- `__mul__(self, other)` - умножение (x * y).
- `__truediv__(self, other)` - деление (x / y).
- `__floordiv__(self, other)` - целочисленное деление (x // y).
- `__mod__(self, other)` - остаток от деления (x % y).
- `__divmod__(self, other)` - частное и остаток (divmod(x, y)).
- `__pow__(self, other[, modulo])` - возведение в степень (`x ** y`, `pow(x, y[, modulo])`).
- `__lshift__(self, other)` - битовый сдвиг влево (x << y).
- `__rshift__(self, other)` - битовый сдвиг вправо (x >> y).
- `__and__(self, other)` - битовое И (x & y).
- `__xor__(self, other)` - битовое ИСКЛЮЧАЮЩЕЕ ИЛИ (x ^ y).
- `__or__(self, other)` - битовое ИЛИ (x | y).

3. В каких случаях будут вызваны следующие методы: __add__, __iadd__ и __radd__? Приведите примеры.

Метод add вызывается при использовании оператора "+" для объектов данного класса. Метод iadd вызывается при использовании оператора "+=" для того же класса. Метод radd вызывается, когда объект данного класса слева от оператора "+". Примеры:

```
class Number:
    def __init__(self, value):
        self.value = value
```

```
    def __add__(self, other):
        return Number(self.value + other.value)

    def __iadd__(self, other):
        self.value += other.value
        return self

    def __radd__(self, other):
        return Number(self.value + other)
```

4. Для каких целей предназначен метод __new__? Чем он отличается от метода __init__?

Метод new предназначен для создания экземпляров класса, он возвращает новый объект класса. Метод init вызывается после создания экземпляра класса и выполняет инициализацию экземпляра. Отличие в том, что метод new создает объект, а init инициализирует его.

5. Чем отличаются методы __str__ и __repr__?

Можно сказать, что методы repr() и __repr__ взаимозаменяемы. Функция __str__ в Python делает то же самое, но ее поведение всё же немного отличается. Она предназначена для создания удобочитаемой версии, полезной для отслеживания или отображения информации об объекте. А метод __repr__ предназначен для предоставления «официального» текстового образа объекта, который можно использовать для воссоздания этого объекта.