

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное автономное  
образовательное учреждение высшего образования  
«Северо-Кавказский федеральный университет»**

**Кафедра инфокоммуникаций**

**Отчет по лабораторной работе № 4.2  
«Перегрузка операторов в языке Python»**

**по дисциплине «Объектно-ориентированное программирование»**

Выполнил студент группы ИВТ-б-о-20-1

Дыбов Д.В. « » \_\_\_\_\_ 20\_\_ г.

Подпись студента \_\_\_\_\_

Работа защищена « » \_\_\_\_\_ 20\_\_ г.

Проверил Воронкин Р.А. \_\_\_\_\_  
(подпись)

Ставрополь 2022

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

### Ход Работы

1. Создал новый репозиторий для лабораторной работы №4.2;
2. Клонировал созданный репозиторий на компьютер;
3. Создал новый PyCharm проект в папке репозитория;
4. Проработал пример:

```
r1 = 3 / 4
r2 = 5 / 6
r1 + r2 = 19 / 12
r1 - r2 = -1 / 4
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 code 0
```

Рисунок 1 – Результат выполнения примера

5. Выполнил первое индивидуальное задание;
6. Проверил задание на работоспособность:

```
Целочисленное деление: (2, 8)

Process finished with exit code 0
```

Рисунок 2 – Результат выполнения первого индивидуального задания

7. Выполнил второе индивидуальное задание;

8. Проверил задание на работоспособность:

```
r1 + r2 = 5fx0
r1 - r2 = fex0
r1 * r2 = 6d2x0
r1 < r2 = False
r1 > r2 = True
r1 == r2 = False

Process finished with exit code 0
```

Рисунок 3 – Результат выполнения второго индивидуального задания

9. Проверил пример на наличие ошибок:

# 100%

[Guido](#)

---

Lines: 124  
Hints: 0

## PythonChecker Makes Your Code Great Again

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

class Rational:

    def __init__(self, a=0, b=1):
        a = int(a)
        b = int(b)
```

Рисунок 4 – Проверка примера на наличие ошибок

10. Проверил первое индивидуальное задание на наличие ошибок:

100%

[Guido](#)

Lines: 31

Hints: 0

## PythonChecker Makes Your Code Great Again

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

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

Поле first — целое положительное число, числитель; поле second — целое положительное
```

Рисунок 5 – Проверка примера на наличие ошибок

11. Проверил первое индивидуальное задание на наличие ошибок:

100%

[Guido](#)

Lines: 95

Hints: 0

## PythonChecker Makes Your Code Great Again

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

"""
Создать класс Нех для работы с беззнаковыми целыми шестнадцатеричными числами,
используя для представления числа список из 100 элементов типа int, каждый из которых
является шестнадцатеричной цифрой. Младшая цифра имеет меньший индекс. Реальный
размер списка задается как аргумент конструктора инициализации. Реализовать
```

Рисунок 6 – Проверка примера на наличие ошибок

### Ответы на контрольные вопросы

1. Какие средства существуют в Python для перегрузки операций?

Перегрузка осуществляется при помощи специальных методов.

Методы группируются по следующим категориям:

- методы для всех видов операций;
- методы перегрузки операторов работы с коллекциями;
- методы для числовых операций в двоичной форме;
- методы для других операций над числами;
- методы для операций с дескрипторами;
- методы для операций, используемых с диспетчерами контекста.

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

`__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 ( $x \wedge y$ ).

`__radd__(self, other)`,

`__rsub__(self, other)`,

`__rmul__(self, other)`,

`__rtruediv__(self, other)`,

`__rmod__(self, other)`,

`__rdivmod__(self, other)`,

`__rpow__(self, other)`,

`__rlshift__(self, other)`,

`__rrshift__(self, other)`,

`__rand__(self, other)`,

`__rxor__(self, other)`,

`__ror__(self, other)` - делают то же самое, что и арифметические операторы, перечисленные выше, но для аргументов, находящихся справа, и только в случае, если для левого операнда не определён соответствующий метод.

`__iadd__(self, other)` -  $+=$ .

`__isub__(self, other) - -= .`  
`__imul__(self, other) - *= .`  
`__itruediv__(self, other) - /= .`  
`__ifloordiv__(self, other) - //= .`  
`__imod__(self, other) - %= .`  
`__ipow__(self, other[, modulo]) - **= .`  
`__lshift__(self, other) - <<= .`  
`__rshift__(self, other) - >>= .`  
`__iand__(self, other) - &= .`  
`__ixor__(self, other) - ^= .`  
`__ior__(self, other) - |= .`

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

1) `__add__` - `a + b`

2) `__iadd__` - `a += b`

3) `__radd__` - Если не получилось вызвать метод `__add__`

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

Метод `__new__` используется, когда нужно управлять процессом создания нового экземпляра, а `__init__` – когда контролируется его инициализация.

5. Чем отличаются методы `__str__` и `__repr__`?

`__str__` должен возвращать строковый объект, тогда как `__repr__` может возвращать любое выражение в Python

**Вывод:** в ходе занятия были приобретены навыки по перегрузке операторов при написании программ с помощью языка программирования Python версии 3.x.