**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**

**по дисциплине «Технологии программирования и алгоритмизации»**

Выполнил студент группы ИВТ-б-о-20-1

Хашиев Х.М. «  »_____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: Примеры

**https://github.com/Mirror-Shard/L4.2**

1. Изучил теоретический материал, переписал код примера, запустил его:

```
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
```

Рисунок 1 – Работа примера

## Задание 1

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

1. Написал программу, которая из двух чисел создаёт интервал. Перегрузил оператор in, теперь наличие числа можно проверить ещё проще:

```
def __contains__(self, item):
    return item in self.span
```

Рисунок 2 – Перегрузка оператора

2. Программа работает также успешно

```
Введите левую границу интервала: 2
Введите правую границу интервала: 7
[2, 3, 4, 5, 6]
Введите число: 4
Число присутствует в интервале
```

Рисунок 3 – Функционал задания 1

**Задание 2**

Дополнительно к требуемым в заданиях операциям перегрузить операцию индексирования []. Максимально возможный размер списка задать константой. В отдельном поле size должно храниться максимальное для данного объекта количество элементов списка; реализовать метод size(), возвращающий установленную длину. Если количество элементов списка изменяется во время работы, определить в классе поле count. Первоначальные значения size и count устанавливаются конструктором.

Реализовать класс Money, используя для представления суммы денег список словарей. Словарь имеет два ключа: номинал купюры и количество купюр данного достоинства. Номиналы представить как строку. Элемент списка словарей с меньшим индексом содержит меньший номинал.

1. Создание и сумма двух пачек купюр:

```
### Создание  ###
Купюры в кошельке:
Купюра номиналом - fifty. Количество этих купюр - 3
Купюра номиналом - hundred. Количество этих купюр - 1

### Сумма ###
Купюры в кошельке:
Купюра номиналом - fifty. Количество этих купюр - 3
Купюра номиналом - hundred. Количество этих купюр - 2
```

Рисунок 4 – Сумма __add__

2. Теперь отнимаем из кошелька пачку b:

```
### Разность ###
Купюры в кошельке:
Купюра номиналом - fifty. Количество этих купюр - 3
```

Рисунок 5 – Разность __sub__

3. Свойство size было задано с помощью @property, оно не изменяется:

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

Рисунок 6 – Константа в классе

**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 ^ y).

__radd__(self, other) ,

__rsub__(self, other) ,

__rmul__(self, other) ,

__rtruediv__(self, other) ,

__rfloordiv__(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 длялевого 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]) - **= .

__ilshift__(self, other) - <<= .

__irshift__(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 работы приобрёл навыки по работе с перегрузкой операторов с помощью языка программирования Python версии 3.