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

Институт цифрового развития
Кафедра инфокоммуникаций

**«Лабораторная работа 4.2 Перегрузка операторов в языке Python»**

**ОТЧЕТ**
**по лабораторной работе №30**
**дисциплины**
**«Основы программной инженерии»**

Выполнил:
Луценко Дмитрий Андреевич
2 курс, группа ПИЖ-б-о-21-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и сопровождение программного обеспечения», очная форма обучения

_____
(подпись)

Проверил:
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

**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:**

## Задание 1

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, a, b):
        if not isinstance(a, int) or not isinstance(b, int) or b < 0 or a <
0:
            raise ValueError()

        self.__first = a
        self.__second = b

    @property
    def first(self):
        return self.__first

    @property
    def second(self):
        return self.__second

    def read(self, prompt=None):
        line = input() if prompt is None else input(prompt)
        parts = list(map(int, line.split('.', maxsplit=1)))

        if parts[1] < 0 or parts[0] < 0:
            raise ValueError()

        self.__first = parts[0]
        self.__second = parts[1]

    def __mul__(self, number):
        if isinstance(number, int):
            result = (self.first * 100 + self.second) * number
            return Pair(result // 100, result % 100)
        else:
            raise ValueError()

    def __str__(self):
        return f"first: {self.__first}, second: {self.__second}"


def make_Pair(a, b):
    try:
        return Pair(a, b)
```

```
    except ValueError as e:
        print(str(e))
        exit(1)


if __name__ == '__main__':
    pair = make_Pair(1, 90)
    print(pair)
    number = int(input("Введите целое число: "))
    result = pair * number
    print(result)
```

Листинг 1 – Код программы индивидуального задания

```
first: 1, second: 90
Введите целое число: 5
first: 9, second: 50
```

Рисунок 1 – Результат выполнения программы

## 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.

Создать класс Octal для работы с беззнаковыми целыми восьмеричными числами, используя для представления числа список из 100 элементов типа int, каждый элемент которого является восьмеричной цифрой. Младшая цифра имеет меньший индекс (единицы — в нулевом элементе списка). Реальный размер списка задается как аргумент конструктора инициализации. Реализовать арифметические операции, аналогичные встроенным для целых  и операции сравнения.

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

class Octal:
    def __init__(self, size):
        if size <= 0 or size > 100:
            raise ValueError()
        self.digits = [0] * size

    def __str__(self):
        octal_string = ''.join(map(str, self.digits[::-1]))
        return octal_string

    def __add__(self, rhs):
```

```
        if not isinstance(rhs, Octal):
            raise ValueError()

        size = max(len(self.digits), len(rhs.digits))
        result = Octal(size)
        self.add_lenght(rhs)

        carry = 0
        for i, item in enumerate(result):
            sum_digits = self.digits[i] + rhs.digits[i] + carry
            result.digits[i] = sum_digits % 8
            carry = sum_digits // 8

        if carry > 0:
            result.digits.append(carry)

        return result

    def __lt__(self, rhs):
        if not isinstance(rhs, Octal):
            raise ValueError()

        size = max(len(self.digits), len(rhs.digits))
        self.add_lenght(rhs)

        for i in range(size - 1, -1, -1):
            if self.digits[i] < rhs.digits[i]:
                return True
            elif self.digits[i] > rhs.digits[i]:
                return False

        return False

    def __eq__(self, rhs):
        if not isinstance(rhs, Octal):
            raise ValueError()

        self.add_lenght(rhs)
        return self.digits == rhs.digits

    def __getitem__(self, index):
        if 0 <= index <= len(self.digits):
            return self.digits[index]
        else:
            raise IndexError("Index out of range")

    def add_lenght(self, rhs):
        while len(self.digits) != len(rhs.digits):
            if len(self.digits) > len(rhs.digits):
                rhs.digits.append(0)
            if len(self.digits) < len(rhs.digits):
                self.digits.append(0)


if __name__ == '__main__':
    num1 = Octal(4)
    num1.digits = [7, 5, 3, 1]
    print(f"num1: {num1}")

    num2 = Octal(3)
    num2.digits = [2, 4, 6]
    print(f"num2: {num2}")

    num3 = Octal(3)
```

```
    num3.digits = [7, 3, 5]
    print(f"num3: {num3}")

    sum_num = num1 + num2
    print(f"num1 + num2: {sum_num}")

    print(num2 > num3)
    print(num2 == num3)
```

Листинг 1 – Листинг индивидуального задания 2

```
num1: 1357
num2: 642
num3: 537
num1 + num2: 2221
True
False
```

Рисунок 2 – Результат выполнения программы

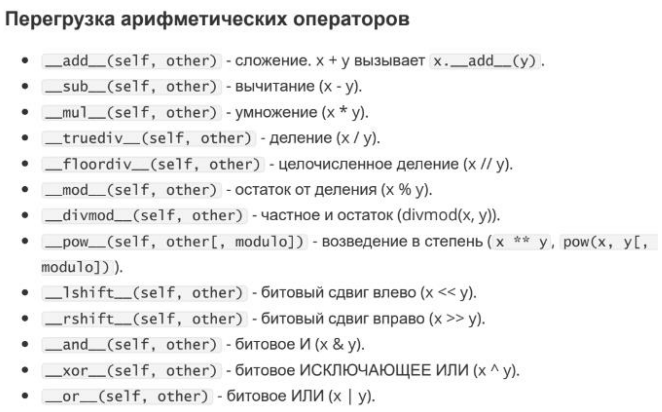
**Вывод:** в ходе выполнения лабораторной работы приобретены навыки по перегрузке операторов при написании программ с помощью языка программирования Python версии 3.x.

**Ответы на 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?**




**3. В каких случаях будут вызваны следующие методы: __add__, __iadd__ и __radd__ ? Приведите примеры.** Метод __add__ будет вызван при операции сложения, метод __iadd__ - это операция +=, а __radd__ делает то же самое, что и __add__, но для аргументов, находящихся справа, и только в случае, если для левого операнда не определён соответствующий метод.

**4. Для каких целей предназначен метод __new__? Чем он отличается от метода __init_?** Метод__new__(cls[, ...]) — управляет созданием экземпляра. В качестве обязательного аргумента принимает класс (не путать с экземпляром). Должен возвращать экземпляр класса для его последующей его передачи методу __init__. А метод __init__ - конструктор.

**5. Чем отличаются методы __str__ и __repr__ ?** Метод __str__(self) - вызывается функциями str, print и format. Возвращает строковое представление объекта, а метод __repr__(self) - вызывается встроенной функцией repr; возвращает "сырые" данные, использующиеся для внутреннего представления в python.