

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

Лабораторная работа 2

Перегрузка операторов в языке Python

Выполнил студент группы ИВТ-б-о-20-1

Симанский М.Ю « » _____ 20__ г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

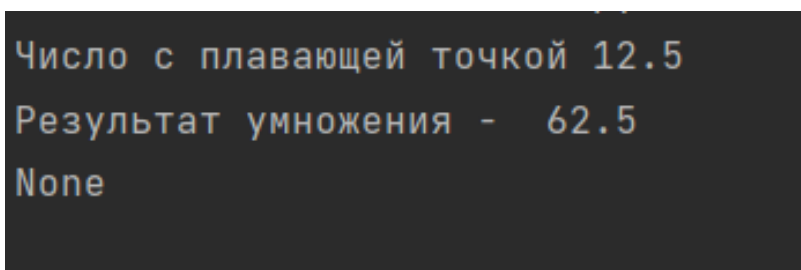
Проверил Воронкин Р.А. _____

(подпись)

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

Задание 1

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



```
Число с плавающей точкой 12.5
Результат умножения - 62.5
None
```

Рисунок 1 – Результат выполнения

Задание 2

Создать класс BitString для работы с битовыми строками не более чем из 100 бит. Битовая строка должна быть представлена списком типа int, каждый элемент которого принимает значение 0 или 1. Реальный размер списка задается как аргумент конструктора инициализации. Должны быть реализованы все традиционные операции для работы с битовыми строками: and, or, xor, not. Реализовать сдвиг влево и сдвиг вправо на заданное количество битов (рис. 2).

```
Результат вычисления включающее ИЛИ 109971607030283
Результат вычисления сдвига направо 6875687500000
Результат вычисления сдвига налево 1760176000000176
Результат вычисления ИЛИ 111050403069739
Длина числа 15
Битовое число 110011000000011
Битовое число 111011010100011
```

Рисунок 2 – Результат выполнения

Ответы на контрольные вопросы

1. Перегрузка операторов — один из способов реализации полиморфизма, когда мы можем задать свою реализацию какого-либо метода в своём классе.

2. `__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 ($x \wedge y$).
`__or__(self, other)` - битовое ИЛИ ($x | y$).

3. операция $x + y$ будет сначала пытаться вызвать `x.__add__(y)`, и только в том случае, если это не получилось, будет пытаться вызвать `y.__radd__(x)`. Аналогично для остальных методов.
4. `__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__`
5. `__str__(self)` - вызывается функциями `str`, `print` и `format`. Возвращает строковое представление объекта.