

**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

Бобров Н.В. « » _____ 20__ г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

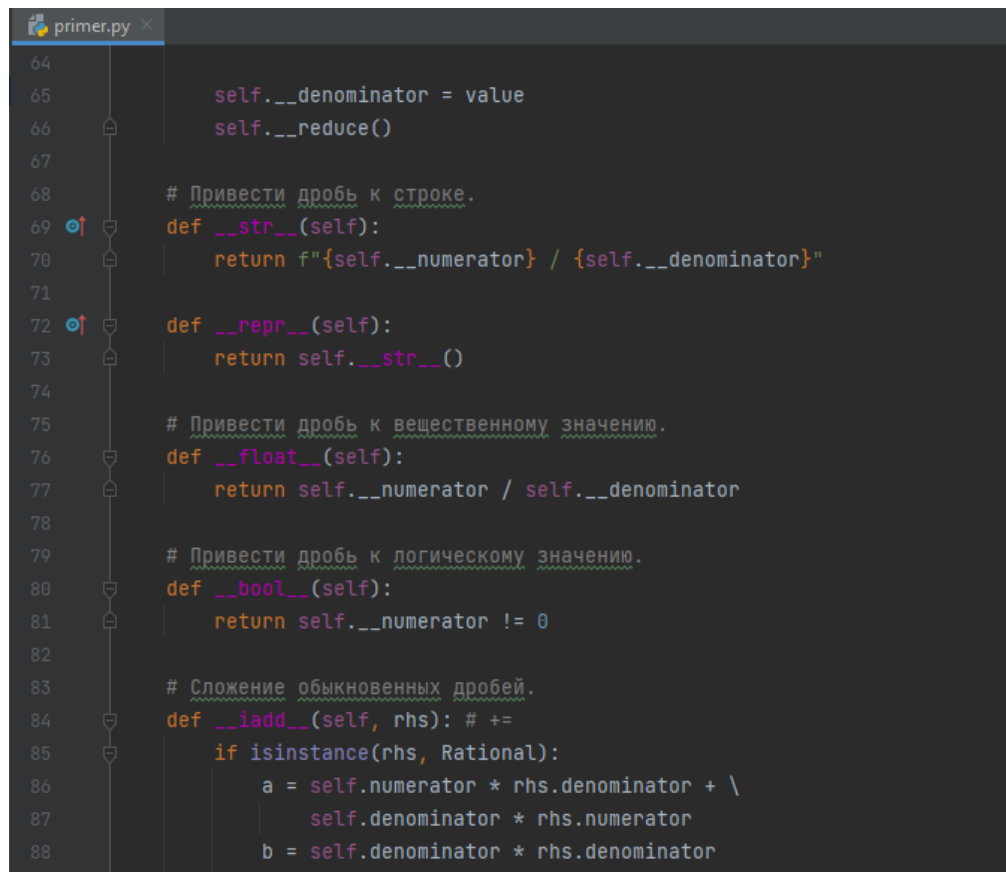
Проверил Воронкин Р.А. _____

(подпись)

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

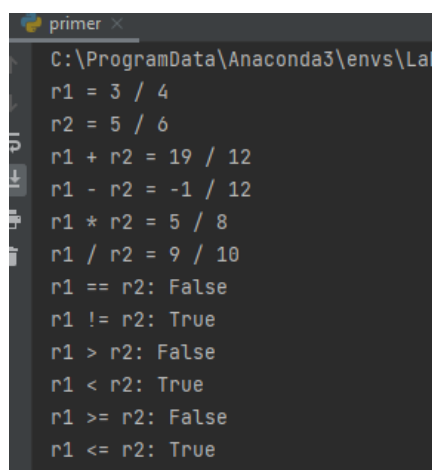
Ход работы:

1. Создал общедоступный репозиторий на Github и клонировал его на свой локальный сервер.
2. Изучив методический материал, приступил к выполнению лабораторной работы. Выполнил пример и запустил его.



```
64
65     self.__denominator = value
66     self.__reduce()
67
68     # Привести дробь к строке.
69     def __str__(self):
70         return f"{self.__numerator} / {self.__denominator}"
71
72     def __repr__(self):
73         return self.__str__()
74
75     # Привести дробь к вещественному значению.
76     def __float__(self):
77         return self.__numerator / self.__denominator
78
79     # Привести дробь к логическому значению.
80     def __bool__(self):
81         return self.__numerator != 0
82
83     # Сложение обыкновенных дробей.
84     def __iadd__(self, rhs): # +=
85         if isinstance(rhs, Rational):
86             a = self.numerator * rhs.denominator + \
87                 self.denominator * rhs.numerator
88             b = self.denominator * rhs.denominator
```

Рисунок 1 – Код с примера лабораторной работы



```
primer >
C:\ProgramData\Anaconda3\envs\Lab
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 2 – Результат работы кода

3. Приступил к выполнению индивидуальных заданий своего варианта.

Вариант 1

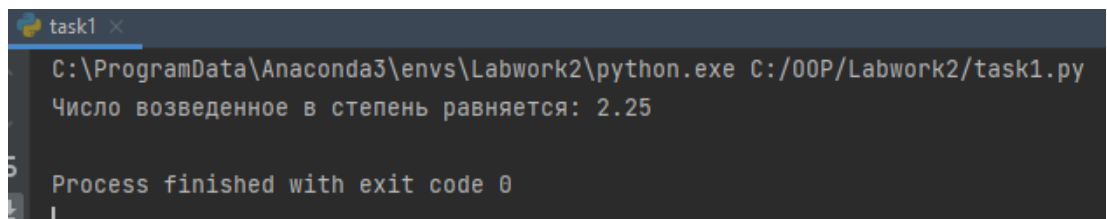
Задание 1.

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

1. Выполнил рефакторинг кода задания 1 лабораторной работы 4.1, максимально задействовав средства перегрузки операторов.

```
10 class Number:
11
12     def __init__(self, first, second):
13         self.first = first
14         self.second = second
15         if self.first == 0:
16             raise ValueError
17
18     def __pow__(self, other):
19         a = self.first + self.second
20         b = other.first + other.second
21         return a ** b
22
23
24 if __name__ == "__main__":
25     num1 = Number(1.5, 0)
26     num2 = Number(2, 0)
27     print(f"Число возведенное в степень равняется: {num1 ** num2}")
```

Рисунок 3 – Рефакторинг кода для первого задания



```
task1 x
C:\ProgramData\Anaconda3\envs\Labwork2\python.exe C:/00P/Labwork2/task1.py
Число возведенное в степень равняется: 2.25
Process finished with exit code 0
```

Рисунок 4 – Результат работы кода

Задание 2.

Условие: 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 `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.

1. Написал код для решения задачи.

```
14 class BitString:
15     def __init__(self, x):
16         # Инициализация
17         self.size = x
18         self.x = [0] * self.size
19
20     def set(self, x):
21         # Установка значения
22         self.x = list(map(int, f'{x:b}'.rjust(self.size, '0')))
23
24     def __invert__(self):
25         # Оператор not (~)
26         self.x = [int(not i) for i in self.x]
27         return self
28
29     def __or__(self, other):
30         # Оператор or (|)
31         x = [a | b for a, b in zip(self.x, other.x)]
32         return ''.join(map(str, x))
33
```

Рисунок 5 – Участок кода

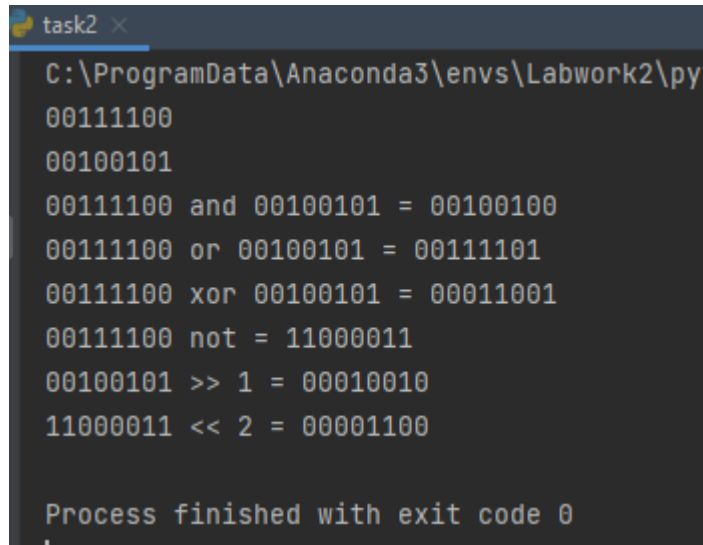
```
task2.py x
34 def __xor__(self, other):
35     # Оператор xor (^)
36     x = [a ^ b for a, b in zip(self.x, other.x)]
37     return ''.join(map(str, x))
38
39 def __and__(self, other):
40     # Оператор and (&)
41     x = [a & b for a, b in zip(other.x, self.x)]
42     return ''.join(map(str, x))
43
44 def __lshift__(self, x):
45     # Оператор сдвиг влево (<<)
46     del (self.x[0:x])
47     self.x += [0] * x
48     return self
49
50 def __rshift__(self, x):
51     # Оператор сдвиг вправо (>>)
52     del (self.x[len(self.x) - x:])
53     self.x = [0] * x + self.x
54     return self
55
56 def __str__(self):
57     # Вывод результата в консоль
58     return ''.join(map(str, self.x))
59
```

Рисунок 6 – Реализация операторов

```
61 if __name__ == "__main__":
62     x = BitString(8) # Размер списка 1 - 8 бит
63     y = BitString(8) # Размер списка 2 - 8 бит
64
65     x.set(60) # Первая цифра 00111100
66     print(x)
67     y.set(37) # Вторая цифра 00100101
68     print(y)
69
70     print(f'{x} and {y} = {x & y}')
71     print(f'{x} or {y} = {x | y}')
72     print(f'{x} xor {y} = {x ^ y}')
73     print(f'{x} not = {~x}')
74     print(f'{y} >> 1 = {y >> 1}')
75     print(f'{x} << 2 = {x << 2}')
```

Рисунок 7 – Конструкция вывода

2. Затем запустил код, чтобы проверить его работу.



```
task2 x
C:\ProgramData\Anaconda3\envs\Labwork2\python
00111100
00100101
00111100 and 00100101 = 00100100
00111100 or 00100101 = 00111101
00111100 xor 00100101 = 00011001
00111100 not = 11000011
00100101 >> 1 = 00010010
11000011 << 2 = 00001100

Process finished with exit code 0
```

Рисунок 8 – Работа кода

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 ($x \ll y$).

`__rshift__(self, other)` - битовый сдвиг вправо ($x \gg y$).

`__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)`,

`__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 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])` - $**=$.

`__ilshift__(self, other)` - $\ll=$.

`__irshift__(self, other)` - $\gg=$.

`__iand__(self, other) - &= .`

`__ixor__(self, other) - ^= .`

`__ior__(self, other) - |= .`

3. В каких случаях будут вызваны следующие методы: `__add__`,

`__iadd__` и `__radd__`?

– `__add__` - `a + b`

– `__iadd__` - `a += b`

– `__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 выполнения лабораторной работы были приобретены навыки по перегрузке операторов при написании программ с помощью языка программирования Python версии 3.x.