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

Погорелов Д.Н « »_____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 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.

```
    self.__denominator = value
    self.__reduce()

# Привести дробь к строке.
def __str__(self):
    return f"{self.__numerator} / {self.__denominator}"

def __repr__(self):
    return self.__str__()

# Привести дробь к вещественному значению.
def __float__(self):
    return self.__numerator / self.__denominator

# Привести дробь к логическому значению.
def __bool__(self):
    return self.__numerator != 0

# Сложение обыкновенных дробей.
def __iadd__(self, rhs): # +=
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator + \
            self.denominator * rhs.numerator
        b = self.denominator * rhs.denominator
```

Рисунок 1 – Код с примера лабораторной работы

Рисунок 2 – Результат работы кода

```
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
```

3. Приступил к выполнению индивидуальных заданий своего варианта.

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.

```python
class Number:

    def __init__(self, first, second):
        self.first = first
        self.second = second
        if self.first == 0:
            raise ValueError

    def __pow__(self, other):
        a = self.first + self.second
        b = other.first + other.second
        return a ** b


if __name__ == "__main__":
    num1 = Number(1.5, 0)
    num2 = Number(2, 0)
    print(f"Число возведенное в степень равняется: {num1 ** num2}")
```

Рисунок 3 – Рефакторинг кода для первого задания

Рисунок 4 – Результат работы кода

```
Число возведенное в степень равняется: 2.25

Process finished with exit code 0
```

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.

```
class BitString:
    def __init__(self, x):
        # Инициализация
        self.size = x
        self.x = [0] * self.size

    def set(self, x):
        # Установка значения
        self.x = list(map(int, f'{x:b}'.rjust(self.size, '0')))

    def __invert__(self):
        # Оператор not (~)
        self.x = [int(not i) for i in self.x]
        return self

    def __or__(self, other):
        # Оператор or (|)
        x = [a | b for a, b in zip(self.x, other.x)]
        return ''.join(map(str, x))
```

Рисунок 5 – Участок кода

```
def __xor__(self, other):
    # Оператор xor (^)
    x = [a ^ b for a, b in zip(self.x, other.x)]
    return ''.join(map(str, x))

def __and__(self, other):
    # Оператор and (&)
    x = [a & b for a, b in zip(other.x, self.x)]
    return ''.join(map(str, x))

def __lshift__(self, x):
    # Оператор сдвиг влево (<<)
    del (self.x[0:x])
    self.x += [0] * x
    return self

def __rshift__(self, x):
    # Оператор сдвиг вправо (>>)
    del (self.x[len(self.x) - x:])
    self.x = [0] * x + self.x
    return self

def __str__(self):
    # Вывод результата в консоль
    return ''.join(map(str, self.x))
```

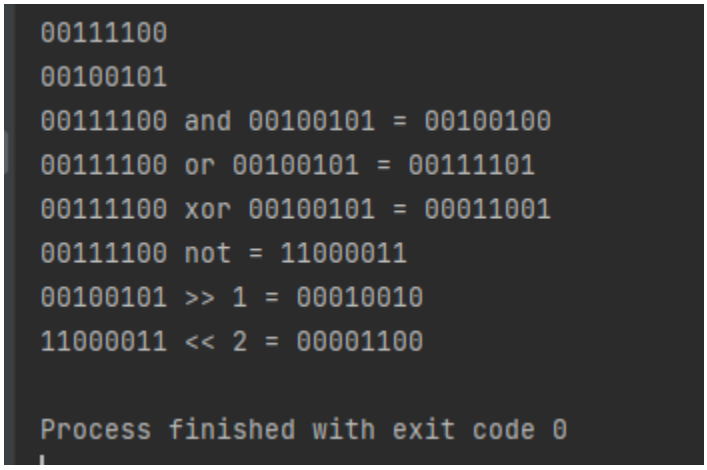
Рисунок 6 – Реализация операторов

```
if __name__ == "__main__":
    x = BitString(8)  # Размер списка 1 - 8 бит
    y = BitString(8)  # Размер списка 2 - 8 бит

    x.set(60)  # Первая цифра 00111100
    print(x)
    y.set(37)  # Вторая цифра 00100101
    print(y)

    print(f'{x} and {y} = {x & y}')
    print(f'{x} or {y} = {x | y}')
    print(f'{x} xor {y} = {x ^ y}')
    print(f'{x} not = {~x}')
    print(f'{y} >> 1 = {y >> 1}')
    print(f'{x} << 2 = {x << 2}')
```

Рисунок 7 – Конструкция вывода

2. Затем запустил код, чтобы проверить его работу.

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 << 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__?

- __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, a_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.