

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

Кафедра инфокоммуникаций

Отчет по лабораторной работе №2
Перегрузка операторов в языке Python

по дисциплине «Объектно-ориентированное программирование»

Выполнил студент группы ИВТ-б-о-20-1

Павленко М.С. « »_____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. Изучив методические указания, приступил к разбору примера.

```
(base) C:\Users\zligo\Documents\GitHub\проекты\ООП\Лаба 1>python "Пример 1".py
3/4
Введите обыкновенную дробь: 5/6
5/6
19/12
1/12
5/8
10/9
```

Рисунок 1.1 – Проверка правильности работы кода

2. Затем начал выполнять задания для моего варианта.

```
✓ class TaskOne:
✓     def __init__(self, first, second):
        self.first = first
        self.second = second
        self.summ = self.first * self.second

✓     def __add__(self, other):
        return self.summ + other.summ

✓ if __name__ == '__main__':
    r1 = TaskOne(100, 2)
    r2 = TaskOne(200, 1)
    print(f'r1 + r2 = {r1 + r2}')
```

Рисунок 1.2 – Код первого индивидуального задания

```
(base) C:\Users\zligo\Documents\GitHub\ООП-4.2>python "Задание 1.py"
r1 + r2 = 400
```

Рисунок 1.3 – Проверка кода первого задания

```

class Money:
    const_len = 100
    def __init__(self, number):
        self.lst = []
        self.number = str(number)
        for i in self.number:
            self.lst.append(i)
        self.size(self.lst)

    def size(self, lst):
        size = len(lst)
        if size > Money.const_len:
            print("Первышена максимальная длина списка")
            exit(1)

    def __add__(self, other):
        summ_lst = []
        lst1 = self.lst[::-1]
        lst2 = other.lst[::-1]
        lst1 = int("".join(lst1))
        lst2 = int("".join(lst2))
        summ_str = str(lst1 + lst2)
        for i in summ_str:
            summ_lst.append(i)
        self.size(summ_lst)
        return "".join(summ_lst[::-1])

    def __sub__(self, other):
        summ_lst = []
        lst1 = self.lst[::-1]
        lst2 = other.lst[::-1]
        lst1 = int("".join(lst1))

```

Рисунок 1.4 – Код второго задания

```

(base) C:\Users\zligo\Documents\GitHub\OOP-4.2>python "Задание 2.py"
r1 + r2 = 23455
r1 - r2 = 09901

```

Рисунок 1.5 – Проверка кода второго задания

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 ($x \wedge y$).

`__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)` - `<<=` .

`__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 выполнения лабораторной работы были приобретены простейшие навыки по работе с методами перегрузки операторов в языке программирования Python.