

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

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2
дисциплины «Объектно-ориентированное программирование»
Вариант 3

Выполнил:
Говоров Егор Юрьевич
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем»,
очная форма обучения

(подпись)

Проверил:
Воронкин Роман Александрович

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

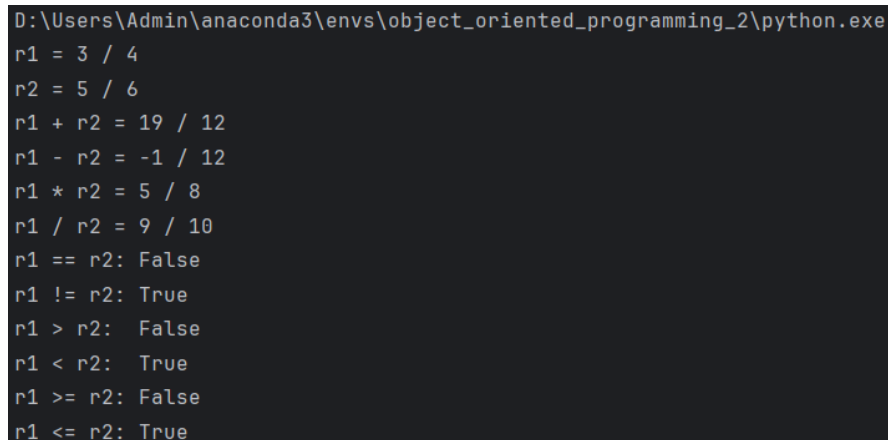
Ставрополь, 2024 г.

Тема: перегрузка операторов в языке Python

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

Порядок выполнения работы:

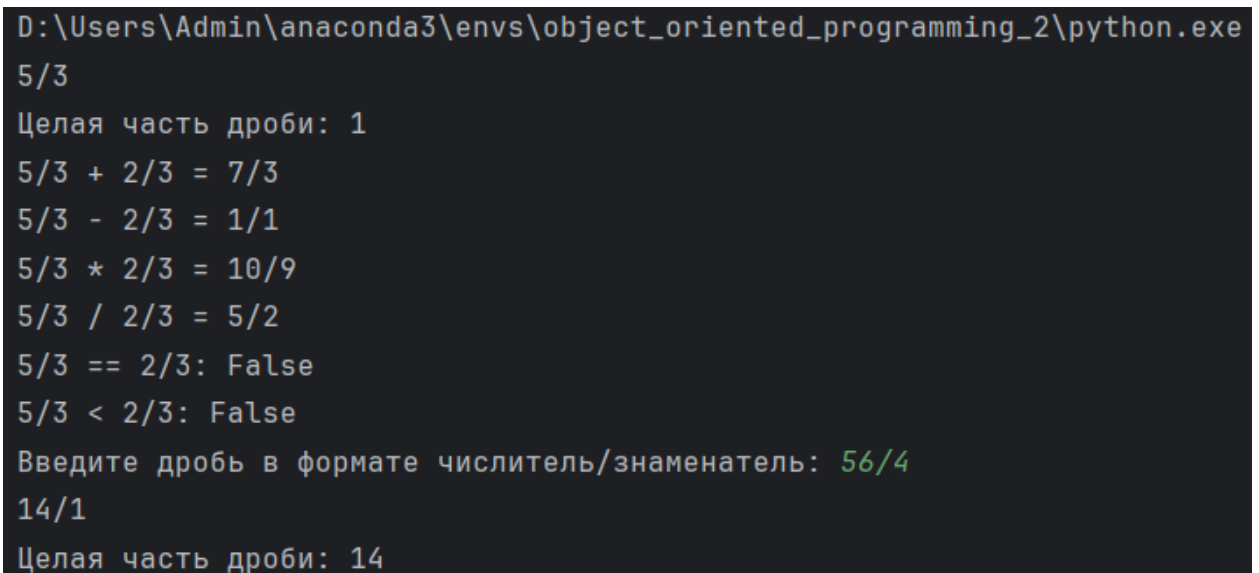
1. Создал новый репозиторий, клонировал его, в нем создал ветку developer и перешел на нее.
2. Проработал пример:



```
D:\Users\Admin\anaconda3\envs\object_oriented_programming_2\python.exe
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 1. Результат работы примера

3. Выполнил индивидуальное задание 1: Выполнить индивидуальное задание 1 лабораторной работы, максимально задействовав имеющиеся в Python средства перегрузки операторов



```
D:\Users\Admin\anaconda3\envs\object_oriented_programming_2\python.exe
5/3
Целая часть дроби: 1
5/3 + 2/3 = 7/3
5/3 - 2/3 = 1/1
5/3 * 2/3 = 10/9
5/3 / 2/3 = 5/2
5/3 == 2/3: False
5/3 < 2/3: False
Введите дробь в формате числитель/знаменатель: 56/4
14/1
Целая часть дроби: 14
```

Рисунок 2. Результат работы индивидуального задания 1

4. Выполнил индивидуальное задание 2 вариант 3: 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.

В тех задачах, где возможно, реализовать конструктор инициализации строкой.

Создать класс Hex для работы с беззнаковыми целыми шестнадцатеричными числами, используя для представления числа список из 100 элементов типа int, каждый из которых является шестнадцатеричной цифрой. Младшая цифра имеет меньший индекс. Реальный размер списка задается как аргумент конструктора инициализации. Реализовать арифметические операции, аналогичные встроенным для целых и операции сравнения.

```
D:\Users\Admin\anaconda3\envs\object_oriented_programming_2\python.exe
Hex1: 0x1A3F
Hex2: 0x2B4
Сумма: 0x01CF3
Разность: 0x178B
Hex1 < Hex2: False
```

Рисунок 3. Результат работы индивидуального задания 2

Ответы на контрольные вопросы:

1. Какие средства существуют в Python для перегрузки операций?

Python можно перегружать операции с помощью специальных методов.

Эти методы начинаются и заканчиваются двумя символами подчеркивания (`__add__`, `__sub__`, и т.д.). Перегрузка позволяет изменять поведение стандартных операций для пользовательских объектов, таких как сложение, умножение, сравнение и другие.

2. Какие существуют методы для перегрузки арифметических операций и операций отношения в языке Python?

Перегрузка арифметических операторов

`__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 ($x | y$).

`__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])` - **= .
`__lshift__(self, other)` - <<= .
`__rshift__(self, other)` - >>= .
`__iand__(self, other)` - &= .
`__ixor__(self, other)` - ^= .
`__ior__(self, other)` - |= .
`__neg__(self)` - унарный -.
`__pos__(self)` - унарный +.
`__abs__(self)` - модуль (abs()).
`__invert__(self)` - инверсия (~).
`__complex__(self)` - приведение к complex.
`__int__(self)` - приведение к int.
`__float__(self)` - приведение к float.
`__round__(self[, n])` - округление.

Перегрузка операторов отношения:

`__lt__(self, other)` - $x < y$ вызывает `x.__lt__(y)` .
`__le__(self, other)` - $x \leq y$ вызывает `x.__le__(y)` .
`__eq__(self, other)` - $x == y$ вызывает `x.__eq__(y)` .
`__ne__(self, other)` - $x != y$ вызывает `x.__ne__(y)` .
`__gt__(self, other)` - $x > y$ вызывает `x.__gt__(y)` .
`__ge__(self, other)` - $x \geq y$ вызывает `x.__ge__(y)` .

3. В каких случаях будут вызваны следующие методы: `__add__`, `__iadd__` и `__radd__`? Приведите примеры.

`__add__(self, other)` вызывается при обычной операции сложения объектов (a+b).

`__iadd__(self, other)` вызывается при операции сокращённого сложения с присваиванием (a += b).

`__radd__(self, other)` вызывается, если левый операнд не поддерживает операцию сложения, а правый операнд пытается выполнить операцию через `+`. Например, если операция `other + self` не реализована для объекта `other`, будет вызван метод `self._radd_(other)`:

4. Для каких целей предназначен метод `__new__`? Чем он отличается от метода `__init__`?

Метод `__new__` предназначен для создания нового экземпляра класса. Это метод, который отвечает за выделение памяти для объекта и является первой стадией создания объекта. Его основная задача — вернуть новый экземпляр класса.

`__new__` используется в основном для создания экземпляров неизменяемых типов (например, `int`, `str`, `tuple`).

В отличие от метода `__init__`, который инициализирует уже существующий объект, `__new__` создаёт сам объект.

5. Чем отличаются методы `__str__` и `__repr__`?

`__str__(self)` предназначен для возвращения "человеко-читаемого" строкового представления объекта. Этот метод используется функцией `str()` и

выводится, когда объект передается в функцию `print()`. Цель этого метода — предоставление удобного для восприятия пользователем описания объекта.

`__repr__(self)` должен возвращать строковое представление объекта, которое предназначено для программистов и должно быть максимально информативным. Этот метод используется функцией `repr()` и должен возвращать строку, которая при необходимости может быть использована для создания идентичного объекта (если это возможно). Если `__str__` не определен, используется `__repr__`.

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