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

Институт перспективной инженерии

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №2
дисциплины «Объектно-ориентированное программирование»
Вариант 7

Выполнил:
Мотовилов Вадим Борисович
3 курс, группа ИВТ-б-о-23-2,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р.А. ктн доцент департамента перспективной инженерии

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2026 г.

Тема: Перегрузка операторов в зыке Python.

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

Порядок выполнения работы:

7. Реализовать класс Rational, используя два списка из 100 элементов типа int для представления числителя и знаменателя. Каждый элемент является десятичной цифрой. Младшая цифра имеет меньший индекс (единицы — в нулевом элементе списка). Реальный размер списка задается как аргумент конструктора инициализации.

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

class Rational:
    MAX_SIZE = 100

    def __init__(self, size=10, value="0"):
        if size > self.MAX_SIZE:
            print(f"Ошибка: размер не может превышать {self.MAX_SIZE}")
            size = self.MAX_SIZE

        self._size = size
        self.count = 0
        self.count_den = 0

        self.numerator = [0] * self.MAX_SIZE
        self.denominator = [0] * self.MAX_SIZE

        self._set_from_string(value)

    def _set_from_string(self, value):
        if '/' in value:
            parts = value.split('/')
            if len(parts) != 2:
                print("Ошибка: неверный формат строки")
                return
            num_str, den_str = parts[0].strip(), parts[1].strip()
        else:
            num_str = value.strip()
            den_str = "1"

        self._set_digits(self.numerator, num_str, True)
        self.count = min(len(num_str), self._size)

        self._set_digits(self.denominator, den_str, False)
        self.count_den = min(len(den_str), self._size)

    def _set_digits(self, arr, num_str, is_numerator):
        num_str = num_str.lstrip('0')
```

```
        if not num_str:
            num_str = "0"

        if not num_str.isdigit():
            print(f"Ошибка: {'числитель' if is_numerator else 'знаменатель'} должен содержать
только цифры")
            return

        if len(num_str) > self._size:
            print(f"Внимание: {'числитель' if is_numerator else 'знаменатель'} обрезан до
{self._size} цифр")
            num_str = num_str[-self._size:]

        for i, digit in enumerate(reversed(num_str)):
            if i < self._size:
                arr[i] = int(digit)

    def read(self):
        while True:
            value = input(f"Введите рациональное число (размер до {self._size} цифр, формат
'a/b' или 'a'): ")
            if '/' in value:
                parts = value.split('/')
                if len(parts) == 2 and parts[0].strip().isdigit() and parts[1].strip().isdigit():
                    self._set_from_string(value)
                    break
                else:
                    print("Неверный формат. Используйте 'числитель/знаменатель' или просто
число")
            elif value.strip().isdigit():
                self._set_from_string(value)
                break
            else:
                print("Введите только цифры и символ '/'")

    def display(self):
        num_str = self._get_digits_str(self.numerator, self.count)
        den_str = self._get_digits_str(self.denominator, self.count_den)

        if den_str == "1":
            print(f"Rational: {num_str}")
        else:
            print(f"Rational: {num_str}/{den_str}")

    def _get_digits_str(self, arr, count):
        if count == 0:
            return "0"

        digits = []
        for i in range(count-1, -1, -1):
            digits.append(str(arr[i]))
```

```
        result = ''.join(digits)
        return result if result else "0"

    def get_size(self):
        return self._size

    def __getitem__(self, index):
        if isinstance(index, slice):
            start = index.start or 0
            stop = min(index.stop or self.count, self.count)
            step = index.step or 1
            return [self.numerator[i] for i in range(start, stop, step)]
        else:
            if 0 <= index < self.count:
                return self.numerator[index]
            elif 0 <= index < self.MAX_SIZE:
                return 0
            else:
                raise IndexError(f"Индекс {index} вне диапазона")

    def get_denominator_digit(self, index):
        if 0 <= index < self.count_den:
            return self.denominator[index]
        elif 0 <= index < self.MAX_SIZE:
            return 0
        else:
            raise IndexError(f"Индекс {index} вне диапазона")

    def __str__(self):
        num_str = self._get_digits_str(self.numerator, self.count)
        den_str = self._get_digits_str(self.denominator, self.count_den)

        if den_str == "1":
            return num_str
        else:
            return f"{num_str}/{den_str}"

if __name__ == '__main__':
    print("1. Создание объектов:")
    r1 = Rational(20, "123/456")
    print("   Rational(20, '123/456'): ", end="")
    r1.display()

    r2 = Rational(15, "789")
    print("   Rational(15, '789'):     ", end="")
    r2.display()

    r3 = Rational(5, "1000/25")
    print("   Rational(5, '1000/25'):  ", end="")
    r3.display()

    print(f"\n2. Размер объектов:")
```

```
    print(f"   r1.get_size() = {r1.get_size()}")
    print(f"   r2.get_size() = {r2.get_size()}")
    print(f"   r3.get_size() = {r3.get_size()}")

    print("\n3. Операция индексирования [] для числителя:")
    print(f"   r1[0] = {r1[0]} (единицы)")
    print(f"   r1[1] = {r1[1]} (десятки)")
    print(f"   r1[2] = {r1[2]} (сотни)")
    print(f"   r1[0:3] = {r1[0:3]} (первые 3 цифры)")

    print("\n4. Цифры знаменателя:")
    print(f"   r1 знаменатель[0] = {r1.get_denominator_digit(0)}")
    print(f"   r1 знаменатель[1] = {r1.get_denominator_digit(1)}")
    print(f"   r1 знаменатель[2] = {r1.get_denominator_digit(2)}")

    print("\n5. Ввод нового объекта с клавиатуры:")
    r4 = Rational(10)
    r4.read()
    print("   Введенное число: ", end="")
    r4.display()

    print("\n6. Работа с большими числами:")
    big_num = "12345678901234567890"
    r5 = Rational(15, big_num)
    print(f"   Исходное: {big_num}")
    print(f"   В объекте: {r5}")
    print(f"   Фактическое количество цифр: числитель={r5.count},
знаменатель={r5.count_den}")

    print("\n7. Граничные случаи:")
    r6 = Rational(3, "0")
    print(f"   Rational(3, '0'): {r6}")

    r7 = Rational(3, "007/002")
    print(f"   Rational(3, '007/002'): {r7}")
```

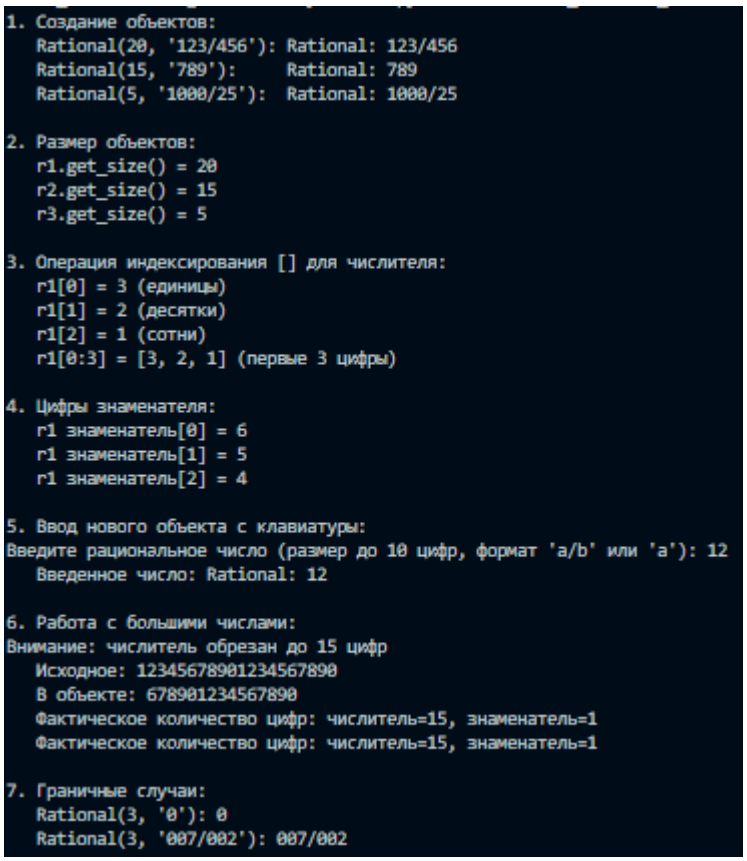

Рисунок 1. Пример работы кода

Контрольные вопросы:

1. Какие средства существуют в Python для перегрузки операций?

В Python перегрузка операций осуществляется через специальные методы (или магические методы). Эти методы позволяют определять поведение операторов для пользовательских классов. Например, для перегрузки арифметических операций, таких как сложение или вычитание, используются методы, начинающиеся и заканчивающиеся двойным подчеркиванием (например, __add__, __sub__ и т.д.).

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

Для перегрузки арифметических операций в Python существуют следующие методы:

__add__(self, other) — перегрузка оператора +

__sub__(self, other) — перегрузка оператора -

__mul__(self, other) — перегрузка оператора *

__truediv__(self, other) — перегрузка оператора /

__floordiv__(self, other) — перегрузка оператора //

__mod__(self, other) — перегрузка оператора %

__pow__(self, other) — перегрузка оператора **

Для перегрузки операций отношения:

__eq__(self, other) — перегрузка оператора ==

__ne__(self, other) — перегрузка оператора !=

__lt__(self, other) — перегрузка оператора <

__le__(self, other) — перегрузка оператора <=

__gt__(self, other) — перегрузка оператора >

__ge__(self, other) — перегрузка оператора >=

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

__add__: Этот метод вызывается, когда используется оператор +. Например:

```
class MyNumber:
    def __init__(self, value):
        self.value = value
    def __add__(self, other):
        return MyNumber(self.value + other.value)
a = MyNumber(10)
b = MyNumber(20)
c = a + b  # Вызывает a.__add__(b)
print(c.value)  # 30
```

__iadd__: Этот метод вызывается, когда используется оператор +=. Он позволяет изменять объект на месте. Например:

```
class MyNumber:
    def __init__(self, value):
        self.value = value
    def __iadd__(self, other):
        self.value += other.value
```

```
        return self
a = MyNumber(10)
b = MyNumber(20)
a += b  # Вызывает a.__iadd__(b)
print(a.value)  # 30
```

__radd__: Этот метод вызывается, когда объект находится справа от оператора +, и левый операнд не поддерживает операцию. Например:

```
class MyNumber:
    def __init__(self, value):
        self.value = value
    def __radd__(self, other):
        return MyNumber(self.value + other)
a = MyNumber(10)
c = 5 + a  # Вызывает a.__radd__(5)
print(c.value)  # 15
```

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

Метод __new__ используется для создания нового экземпляра класса. Он вызывается перед __init__ и отвечает за выделение памяти для нового объекта. __new__ должен возвращать новый объект, который будет передан в __init__.

Метод __init__ используется для инициализации уже созданного объекта. Он не создает новый объект, а настраивает его состояние.

Пример:

```
class MyClass:
    def __new__(cls, *args, **kwargs):
        print("Создание нового объекта")
        return super(MyClass, cls).__new__(cls)
    def __init__(self, value):
        print("Инициализация объекта")
```

```
        self.value = value
obj = MyClass(10)  # Сначала вызовется __new__, затем __init__
```

5. Чем отличаются методы __str__ и __repr__?

__str__: Этот метод предназначен для возвращения "читаемого" строкового представления объекта, которое будет понятным для пользователя. Он вызывается, когда используется функция str() или при печати объекта.

__repr__: Этот метод предназначен для возвращения "официального" строкового представления объекта, которое должно быть понятным для разработчиков. Он вызывается, когда используется функция repr() или когда объект вводится в интерактивном режиме.

Пример:

```
class MyClass:
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return f"Строковое представление: {self.value}"
    def __repr__(self):
        return f"MyClass({self.value})"
obj = MyClass(10)
print(str(obj))   # Вывод: Строковое представление: 10
print(repr(obj))  # Вывод: MyClass(10)
```

Таким образом, __str__ предназначен для пользователя, а __repr__ — для разработчиков.

Вывод: в результате выполнения лабораторной работы были приобретены навыки по работе с классами и объектами при написании программ с помощью языка программирования Python версии 3.x.