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 № 18

Выполнил:
Текеева Мадина Азрет-Алиевна
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)

Проверил:
Доцент департамента цифровых, робототехнических систем и электроники института перспективной инженерии Воронкин Р.А

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г.

Тема: Перегрузка операторов в языке Python.

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

Репозиторий: https://github.com/bickrosss/TM_oop_lab2

Порядок выполнения работы:

1. Изменили класс Rational используя перегрузку операторов.

```
class Rational:
    def __init__(self, a: int = 0, b: int = 1) -> None:
        a = int(a)
        b = int(b)

        if b == 0:
            raise ValueError("Denominator cannot be zero")

        self.__numerator = abs(a)
        self.__denominator = abs(b)

        self.__reduce()

    def __reduce(self) -> None:
        def gcd(a: int, b: int) -> int:
            if a == 0:
                return b
            elif b == 0:
                return a
            elif a >= b:
                return gcd(a % b, b)
            else:
                return gcd(a, b % a)

        sign = 1
        if (self.__numerator > 0 and self.__denominator < 0) or (
            self.__numerator < 0 and self.__denominator > 0
        ):
            sign = -1
        a, b = abs(self.__numerator), abs(self.__denominator)
        c = gcd(a, b)

        self.__numerator = sign * (a // c)
        self.__denominator = b // c

    def __clone(self) -> "Rational":
        return Rational(self.__numerator, self.__denominator)

    @property
```

```
    def numerator(self) -> int:
        return self.__numerator

    @numerator.setter
    def numerator(self, value: int) -> None:
        self.__numerator = int(value)
        self.__reduce()

    @property
    def denominator(self) -> int:
        return self.__denominator

    @denominator.setter
    def denominator(self, value: int) -> None:
        value = int(value)
        if value == 0:
            raise ValueError("Denominator cannot be zero")
        self.__denominator = value
        self.__reduce()

    def __str__(self) -> str:
        return f"{self.__numerator}/{self.__denominator}"

    def __repr__(self) -> str:
        return self.__str__()

    def __float__(self) -> float:
        return self.__numerator / self.__denominator

    def __bool__(self) -> bool:
        return self.__numerator != 0

    def __iadd__(self, rhs: "Rational") -> "Rational":
        if isinstance(rhs, Rational):
            a = self.numerator * rhs.denominator + self.denominator *
rhs.numerator
            b = self.denominator * rhs.denominator

            self.__numerator, self.__denominator = a, b
            self.__reduce()
            return self
        else:
            raise ValueError("Illegal type of  the argument")

    def __add__(self, rhs: "Rational") -> "Rational":
        return self.__clone().__iadd__(rhs)

    def __isub__(self, rhs: "Rational") -> "Rational":
        if isinstance(rhs, Rational):
            a = self.numerator * rhs.denominator - self.denominator *
rhs.numerator
```

```
            b = self.denominator * rhs.denominator

            self.__numerator, self.__denominator = a, b
            self.__reduce()
            return self
        else:
            raise ValueError("Illegal type of  the argument")

    def __sub__(self, rhs: "Rational") -> "Rational":
        return self.__clone().__isub__(rhs)

    def __imul__(self, rhs: "Rational") -> "Rational":
        if isinstance(rhs, Rational):
            a = self.numerator * rhs.numerator
            b = self.denominator * rhs.denominator
            self.__numerator, self.__denominator = a, b
            self.__reduce()
            return self
        else:
            raise ValueError("Illegal type of  the argument")

    def __mul__(self, rhs: "Rational") -> "Rational":
        return self.__clone().__imul__(rhs)

    def __itruediv__(self, rhs: "Rational") -> "Rational":
        if isinstance(rhs, Rational):
            a = self.numerator * rhs.numerator
            b = self.denominator * rhs.denominator

            if b == 0:
                raise ValueError("Illegal type of  the argument")

            self.__numerator, self.__denominator = a, b
            self.__reduce()
            return self
        else:
            raise ValueError("Illegal type of  the argument")

    def __truediv__(self, rhs: "Rational") -> "Rational":
        return self.__clone().__itruediv__(rhs)

    def __eq__(self, rhs: "Rational") -> bool:
        if isinstance(rhs, Rational):
            return (
                self.numerator == rhs.numerator and self.denominator ==
rhs.denominator
            )
        return False

    def __ne__(self, rhs: "Rational") -> bool:
        if isinstance(rhs, Rational):
```

```
            return not self.__eq__(rhs)
        return False

    def __gt__(self, rhs: "Rational") -> bool:
        if isinstance(rhs, Rational):
            return self.__float__() > rhs.__float__()
        return False

    def __lt__(self, rhs: "Rational") -> bool:
        if isinstance(rhs, Rational):
            return self.__float__() < rhs.__float__()
        return False

    def __ge__(self, rhs: "Rational") -> bool:
        if isinstance(rhs, Rational):
            return not self.__lt__(rhs)
        return False

    def __le__(self, rhs: "Rational") -> bool:
        if isinstance(rhs, Rational):
            return not self.__gt__(rhs)
        return False


if __name__ == "__main__":

    r1 = Rational(3, 4)
    print(f"Первая дробь r1={r1}:\n", end="")

    r2 = Rational(5, 6)
    print(f"Вторая дробь r2={r2}: ", end="")

    print(f"Сложение: r1 + r2 = {r1+r2}\n", end="")

    print(f"Вычитание: r1 - r2 = {r1-r2}\n", end="")

    print(f"Умножение: r1 * r2 = {r1*r2}\n", end="")

    print(f"Деление: r1 / r2 = {r1/r2}", end="")

    print(f"Равенство: r1 == r2 = {r1==r2}")

    print(f"Неравенство: r1 != r2 = {r1!=r2}")

    print(f"Больше: r1 > r2 = {r1>r2}")

    print(f"Меньше: r1 < r2 = {r1<r2}")

    print(f"Больше или равно: r1 >= r2 = {r1>=r2}")

    print(f"Меньше или равно: r1 <= r2 = {r1<=r2}")
```

```
@bickrosss →/workspaces/TM_oop_lab2 (main) $ python examples/rational_example.py
Первая дробь r1=3/4:
Вторая дробь r2=5/6: Сложение: r1 + r2 = 19/12
Вычитание: r1 - r2 = -1/12
Умножение: r1 * r2 = 5/8
Деление: r1 / r2 = 5/8Равенство: r1 == r2 = False
Неравенство: r1 != r2 = True
Больше: r1 > r2 = False
Меньше: r1 < r2 = True
Больше или равно: r1 >= r2 = False
Меньше или равно: r1 <= r2 = True
```

Рисунок 1. Работа примера

2. Выполнили задание 1.

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

Класс:

```python
from math import gcd as gsd


class Pair:
    def __init__(self, first=0, second=0):
        if not isinstance(first, (int, float)):
            print("Ошибка: first должно быть числом")
            exit(1)
        if not isinstance(second, int) or second < 0:
            print("Ошибка: second должно быть положительным целым числом")
            exit(1)

        self.first = int(first)
        self.second = int(second)

    def read(self):
        try:
            self.first = int(input("Введите целую часть числа: "))
            second_input = int(input("Введите дробную часть числа: "))
            if second_input < 0:
                print("Ошибка: дробная часть должна быть положительной")
                exit(1)
            self.second = second_input
        except ValueError:
            print("Ошибка: введите целые числа")
            exit(1)

    def display(self):
```

```
        print(f"Число: {self.first}.{self.second}")

    def __mul__(self, other):
        """Перегрузка оператора * """
        if not isinstance(other, int):
            raise TypeError("Множитель должен быть целым числом")

        full_number = float(f"{self.first}.{self.second}")
        result_number = full_number * other
        result_str = str(result_number)
        parts = result_str.split('.')
        new_first = int(parts[0])
        new_second = int(parts[1]) if len(parts) > 1 else 0
        return Pair(new_first, new_second)

    def __rmul__(self, other):
        """Перегрузка оператора * (справа)"""
        return self.__mul__(other)

    def __str__(self):
        """Перегрузка str()"""
        return f"{self.first}.{self.second}"

    def __repr__(self):
        """Перегрузка repr()"""
        return f"Pair({self.first}, {self.second})"

    def __eq__(self, other):
        """Перегрузка оператора =="""
        if not isinstance(other, Pair):
            return False
        return self.first == other.first and self.second == other.second

    def __float__(self):
        """Перегрузка float()"""
        return float(f"{self.first}.{self.second}")

    # Старый метод multiply оставляем для обратной совместимости
    def multiply(self, number):
        return self.__mul__(number)


def make_pair(first, second):
    if not isinstance(first, (int, float)):
        print("Ошибка: first должно быть числом")
        exit(1)
    if not isinstance(second, int) or second < 0:
        print("Ошибка: second должно быть положительным целым числом")
        exit(1)

    return Pair(first, second)
```

```
1. Создание объектов:
pair1 = 3.5
pair2 = 2.44
pair3 = 7.8
repr(pair1) = Pair(3, 5)

2. Перегрузка оператора умножения *:
3.5 * 3 = 10.5
2 * 2.44 = 4.88
7.8 * 0 = 0.0
3.5 * -2 = -7.0

3. Перегрузка оператора сравнения ==:
3.5 == 3.5 : True
3.5 == 2.44 : False
3.5 == 'строка' : False

4. Перегрузка float():
float(3.5) = 3.5
Тип: <class 'float'>

5. Комбинированные операции:
(3.5 * 2 * 3) = 21.0

6. Ввод с клавиатуры и операции:
Введите целую часть числа: 7
Введите дробную часть числа: 88
Введенное число: 7.88
Введите целое число для умножения: 6
Результат: 7.88 * 6 = 47.28

7. Демонстрация всех возможностей:
   1.5 *  1 = 1.5
   1.5 *  2 = 3.0
   1.5 * -1 = -1.5
   1.5 *  0 = 0.0
```

Рисунок 2. Результат

```
========================================== test session starts ==========================================
platform linux -- Python 3.12.1, pytest-8.4.2, pluggy-1.6.0 -- /usr/local/py-utils/venvs/pytest/bin/python
cachedir: .pytest_cache
rootdir: /workspaces/TM_oop_lab2
configfile: pyproject.toml
plugins: anyio-4.11.0
collected 11 items

tests/test_pair.py::test_multiplication_operator PASSED                                          [  9%]
tests/test_pair.py::test_right_multiplication PASSED                                             [ 18%]
tests/test_pair.py::test_multiplication_negative PASSED                                          [ 27%]
tests/test_pair.py::test_multiplication_zero PASSED                                              [ 36%]
tests/test_pair.py::test_equality_operator PASSED                                                [ 45%]
tests/test_pair.py::test_float_conversion PASSED                                                 [ 54%]
tests/test_pair.py::test_string_representation PASSED                                            [ 63%]
tests/test_pair.py::test_repr_representation PASSED                                              [ 72%]
tests/test_pair.py::test_multiplication_type_error PASSED                                        [ 81%]
tests/test_pair.py::test_make_pair_function PASSED                                               [ 90%]
tests/test_pair.py::test_backward_compatibility PASSED                                           [100%]

========================================== 11 passed in 0.03s ==========================================
```

Рисунок 3. Тест

3. Решили задание 2.

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 Octal для работы с беззнаковыми целыми восьмеричными числами, используя для представления числа список из 100 элементов типа int, каждый элемент которого является восьмеричной цифрой. Младшая цифра имеет меньший индекс (единицы - в нулевом элементе списка). Реальный размер списка задается как аргумент конструктора инициализации. Реализовать арифметические операции, аналогичные встроенным для целых и операции сравнения.

Класс:

```
class Octal:
    MAX_SIZE = 100

    def __init__(self, value=0, size=10):
        self.size = min(size, Octal.MAX_SIZE)
        self.count = 0
        self.digits = [0] * self.size

        if isinstance(value, str):
            self._init_from_string(value)
        else:
            self._init_from_int(value)

    def _init_from_string(self, value_str):
        """Инициализация из строки"""
        if value_str.startswith('0o'):
            value_str = value_str[2:]

        self.count = min(len(value_str), self.size)
        for i in range(self.count):
```

```
            digit_char = value_str[-(i + 1)]
            digit = int(digit_char)
            if digit < 0 or digit > 7:
                raise ValueError("Восьмеричное число должно содержать цифры 0-7")
            self.digits[i] = digit

    def _init_from_int(self, value_int):
        """Инициализация из целого числа"""
        if value_int < 0:
            raise ValueError("Число должно быть беззнаковым")

        value = value_int
        self.count = 0

        while value > 0 and self.count < self.size:
            self.digits[self.count] = value % 8
            value //= 8
            self.count += 1

        if self.count == 0:
            self.count = 1
            self.digits[0] = 0

    def __getitem__(self, index):
        """Перегрузка операции индексирования []"""
        if index < 0 or index >= self.size:
            raise IndexError("Индекс за пределами списка")
        return self.digits[index]

    def __setitem__(self, index, value):
        """Перегрузка операции присваивания по индексу"""
        if index < 0 or index >= self.size:
            raise IndexError("Индекс за пределами списка")
        if value < 0 or value > 7:
            raise ValueError("Восьмеричная цифра должна быть от 0 до 7")
        self.digits[index] = value
        # Обновляем count если нужно
        if index >= self.count and value != 0:
            self.count = index + 1

    def to_int(self):
        """Преобразование в целое число"""
        result = 0
        for i in range(self.count):
            result += self.digits[i] * (8 ** i)
        return result

    def to_string(self):
        """Преобразование в строку"""
        if self.count == 0:
            return "0"
```

```
        result = ""
        for i in range(self.count - 1, -1, -1):
            result += str(self.digits[i])
        return result

    def display(self):
        """Вывод на экран"""
        print(f"Octal: {self.to_string()} (size: {self.size}, count:
{self.count})")

    def read(self):
        """Ввод с клавиатуры"""
        value_str = input("Введите восьмеричное число: ")
        self._init_from_string(value_str)

    def add(self, other):
        """Сложение"""
        result_int = self.to_int() + other.to_int()
        result_size = max(self.size, other.size)
        return Octal(result_int, result_size)
    def __eq__(self, other):
        return self.equals(other)

    def __lt__(self, other):
        return self.less(other)

    def __gt__(self, other):
        return self.greater(other)

    def __le__(self, other):
        return self.less(other) or self.equals(other)

    def __ge__(self, other):
        return self.greater(other) or self.equals(other)
    def subtract(self, other):
        """Вычитание"""
        if self < other:
            raise ValueError("Результат не может быть отрицательным")
        result_int = self.to_int() - other.to_int()
        result_size = max(self.size, other.size)
        return Octal(result_int, result_size)

    def multiply(self, other):
        """Умножение"""
        result_int = self.to_int() * other.to_int()
        result_size = max(self.size, other.size)
        return Octal(result_int, result_size)

    def divide(self, other):
        """Целочисленное деление"""
        if other.to_int() == 0:
```

```
            raise ZeroDivisionError("Деление на ноль")
        result_int = self.to_int() // other.to_int()
        result_size = max(self.size, other.size)
        return Octal(result_int, result_size)

    def equals(self, other):
        return self.to_int() == other.to_int()

    def greater(self, other):
        return self.to_int() > other.to_int()

    def less(self, other):
        return self.to_int() < other.to_int()

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

    def __repr__(self):
        return f"Octal('{self.to_string()}', size={self.size})"


if __name__ == "__main__":
    print("=== Демонстрация класса Octal ===")
    print()

    print("1. Создание объектов:")
    oct1 = Octal(42)
    oct1.display()

    oct2 = Octal("52")
    oct2.display()

    oct3 = Octal("0o63", size=5)
    oct3.display()
    print()

    print("2. Тестирование индексирования:")
    print(f"oct1[0] = {oct1[0]} (единицы)")
    print(f"oct1[1] = {oct1[1]} (восьмерки)")

    oct4 = Octal(10)
    print(f"До изменения: oct4 = {oct4}")
    oct4[0] = 2
    print(f"После oct4[0] = 2: oct4 = {oct4}")
    print()

    print("3. Арифметические операции:")
    a = Octal("12")
    b = Octal("7")

    sum_result = a.add(b)
```

```
print(f"{a} + {b} = {sum_result}")

diff_result = a.subtract(b)
print(f"{a} - {b} = {diff_result}")

mul_result = a.multiply(b)
print(f"{a} * {b} = {mul_result}")

div_result = a.divide(b)
print(f"{a} / {b} = {div_result}")
print()

print("4. Операции сравнения:")
x = Octal("15")
y = Octal("12")

print(f"{x} == {y}: {x.equals(y)}")
print(f"{x} > {y}: {x.greater(y)}")
print(f"{x} < {y}: {x.less(y)}")
print()

print("5. Работа с размером:")
large = Octal("777", size=5)
large.display()
print(f"large.size() = {large.size}")
print(f"large.count = {large.count}")
print()

print("6. Ввод с клавиатуры:")
try:
    user_oct = Octal(size=10)
    user_oct.read()
    user_oct.display()
except Exception as e:
    print(f"Ошибка: {e}")
```

```
1. Создание объектов Octal:

Octal(42) = Octal: 52 (size: 10, count: 2)
Octal('52') = Octal: 52 (size: 10, count: 2)
Octal('0o63') = Octal: 63 (size: 10, count: 2)
Octal('777', size=5) = Octal: 777 (size: 5, count: 3)

2. Операция индексирования []:

test_oct = Octal('1234')
test_oct[0] = 4 (единицы)
test_oct[1] = 3 (восьмерки)
test_oct[2] = 2 (64-ки)
test_oct[3] = 1 (512-ки)

Изменение через индексирование:
До: mod_oct = 100
После mod_oct[0] = 5: mod_oct = 105

3. Поля size и count:

Octal('12345', size=8):
  size = 8 (максимальный размер)
  count = 5 (текущее количество цифр)
  MAX_SIZE = 100 (глобальная константа)

4. Арифметические операции:

a = 12 (десятичное: 10)
b = 7 (десятичное: 7)

a + b = 21 (десятичное: 17)
a - b = 3 (десятичное: 3)
a * b = 106 (десятичное: 70)
a / b = 1 (десятичное: 1)
```

Рисунок 4. Результат

```
@bickrosss →/workspaces/TM_oop_lab2 (main) $ pytest tests/test_octal.py
============================================= test session starts ==============================================
platform linux -- Python 3.12.1, pytest-8.4.2, pluggy-1.6.0 -- /usr/local/py-utils/venvs/pytest/bin/python
cachedir: .pytest_cache
rootdir: /workspaces/TM_oop_lab2
configfile: pyproject.toml
plugins: anyio-4.11.0
collected 5 items

tests/test_octal.py::test_octal_creation PASSED                                                      [ 20%]
tests/test_octal.py::test_octal_indexing PASSED                                                      [ 40%]
tests/test_octal.py::test_octal_arithmetic PASSED                                                    [ 60%]
tests/test_octal.py::test_octal_comparison PASSED                                                    [ 80%]
tests/test_octal.py::test_octal_size PASSED                                                          [100%]

==============================================_5 passed in 0.02s ===============================================
```

Рисунок 5. Тест

Вывод: в ходе выполнения лабораторной работы были приобретены навыки по перегрузке операторов при написании программ.