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 №4.2
дисциплины «Объектно-ориентированное программирование»
Вариант 9

Выполнил:
Иващенко Олег Андреевич
3 курс, группа ИВТ-б-о-22-1,
09.03.02 «Информационные и вычислительные машины»,
направленность (профиль)
«Программное обеспечение средств вычислительной техники и автоматизированных систем»

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович, доцент департамента цифровых, робототехнических систем и электроники

_____
(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:

Пример. Изменить класс Rational из примера 1 лабораторной работы 4.1, используя перегрузку операторов.

Листинг 1 – Код программы примера

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

class Rational:

    def __init__(self, a=0, b=1):
        a = int(a)
        b = int(b)

        if b == 0:
            raise ValueError("Illegal value of the denominator")

        self.__numerator = a
        self.__denominator = b

        self.__reduce()

    # Сокращение дроби.
    def __reduce(self):
        # Функция для нахождения наибольшего общего делителя
        def gcd(a, b):
            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
```

```
# Клонировать дробь.
def __clone(self):
    return Rational(self.__numerator, self.__denominator)

@property
def numerator(self):
    return self.__numerator

@numerator.setter
def numerator(self, value):
    self.__numerator = int(value)
    self.__reduce()

@property
def denominator(self):
    return self.__denominator

@denominator.setter
def denominator(self, value):
    value = int(value)
    if value == 0:
        raise ValueError("Illegal value of the denominator")

    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

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

def __add__(self, rhs): # +
    return self.__clone().__iadd__(rhs)
```

```
# Вычитание обыкновенных дробей.
def __isub__(self, rhs): # -=
    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): # -
    return self.__clone().__isub__(rhs)

# Умножение обыкновенных дробей.
def __imul__(self, rhs): # *=
    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): # *
    return self.__clone().__imul__(rhs)

# Деление обыкновенных дробей.
def __itruediv__(self, rhs): # /=
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator
        b = self.denominator * rhs.numerator

        if b == 0:
            raise ValueError("Illegal value of the denominator")

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

def __truediv__(self, rhs): # /
    return self.__clone().__itruediv__(rhs)

# Отношение обыкновенных дробей.
def __eq__(self, rhs): # ==
    if isinstance(rhs, Rational):
        return (self.numerator == rhs.numerator) and \
        (self.denominator == rhs.denominator)
    else:
        return False
```

```
    def __ne__(self, rhs): # !=
        if isinstance(rhs, Rational):
            return not self.__eq__(rhs)
        else:
            return False

    def __gt__(self, rhs): # >
        if isinstance(rhs, Rational):
            return self.__float__() > rhs.__float__()
        else:
            return False

    def __lt__(self, rhs): # <
        if isinstance(rhs, Rational):
            return self.__float__() < rhs.__float__()
        else:
            return False

    def __ge__(self, rhs): # >=
        if isinstance(rhs, Rational):
            return not self.__lt__(rhs)
        else:
            return False

    def __le__(self, rhs): # <=
        if isinstance(rhs, Rational):
            return not self.__gt__(rhs)
        else:
            return False


if __name__ == '__main__':
    r1 = Rational(3, 4)
    print(f"r1 = {r1}")

    r2 = Rational(5, 6)
    print(f"r2 = {r2}")

    print(f"r1 + r2 = {r1 + r2}")
    print(f"r1 - r2 = {r1 - r2}")
    print(f"r1 * r2 = {r1 * r2}")
    print(f"r1 / r2 = {r1 / r2}")

    print(f"r1 == r2: {r1 == r2}")
    print(f"r1 != r2: {r1 != r2}")
    print(f"r1 > r2: {r1 > r2}")
    print(f"r1 < r2: {r1 < r2}")
    print(f"r1 >= r2: {r1 >= r2}")
    print(f"r1 <= r2: {r1 <= r2}")
```

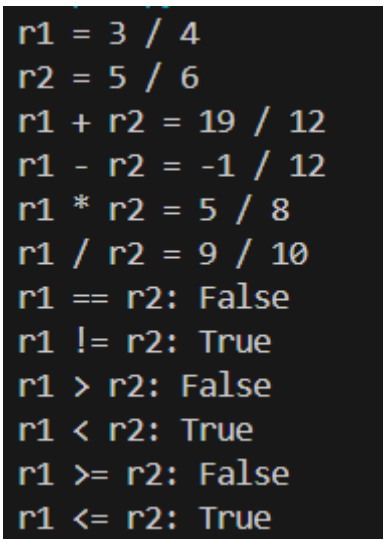

Рисунок 1 – Результат выполнения программы

Индивидуальное задание 1. Выполнить индивидуальное задание 1 лабораторной работы 4.1, максимально задействовав имеющиеся в Python средства перегрузки операторов.

Листинг 2 – Код к программе индивидуального задания 1

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

'''
Поле first — целое положительное число, часы; поле second — целое
положительное число, минуты. Реализовать метод
minutes() — приведение времени в минуты.

Выполнил студент группы ИВТ-б-о-22-1 Иващенко О.А.
'''


class Time:
    def __init__(self, hours=0, minutes=0):
        '''
        Метод инициализации значений.
        '''
        if not self.is_valid(hours) or not self.is_valid(minutes):
            raise ValueError("Часы и минуты должны быть неотрицательными целыми числами")
        self.hours = hours
        self.minutes = minutes

    @staticmethod
    def is_valid(value):
        '''
        Проверяет аргумент на правильность введённых данных.
        '''
```

```
        return isinstance(value, int) and value >= 0

    def total_minutes(self):
        '''
        Метод приведения времени в минуты.
        '''
        return self.hours * 60 + self.minutes

    def __add__(self, other):
        '''
        Перегрузка оператора сложения.
        '''
        if isinstance(other, Time):
            total_minutes = self.total_minutes() + other.total_minutes()
        elif isinstance(other, int):
            total_minutes = self.total_minutes() + other
        else:
            return NotImplemented
        return Time(total_minutes // 60, total_minutes % 60)

    def __sub__(self, other):
        '''
        Перегрузка оператора вычитания.
        '''
        if isinstance(other, Time):
            total_minutes = self.total_minutes() - other.total_minutes()
        elif isinstance(other, int):
            total_minutes = self.total_minutes() - other
        else:
            return NotImplemented
        return Time(total_minutes // 60, total_minutes % 60)

    def __int__(self):
        '''
        Приведение объекта к целому числу (общее количество минут).
        '''
        return self.total_minutes()

    def __str__(self):
        '''
        Строковое представление объекта.
        '''
        return f"{self.hours} ч. {self.minutes} мин."

    def __eq__(self, other):
        '''
        Перегрузка оператора равенства.
        '''
        if isinstance(other, Time):
            return self.total_minutes() == other.total_minutes()
        return False
```

```
    def __lt__(self, other):
        '''
        Перегрузка оператора меньше.
        '''
        if isinstance(other, Time):
            return self.total_minutes() < other.total_minutes()
        return NotImplemented

    def __le__(self, other):
        '''
        Перегрузка оператора меньше или равно.
        '''
        return self == other or self < other


if __name__ == "__main__":
    try:
        hours = int(input("[1] Введите количество часов: "))
        minutes = int(input("[1] Введите количество минут: "))
        time1 = Time(hours, minutes)

        hours = int(input("[2] Введите количество часов: "))
        minutes = int(input("[2] Введите количество минут: "))
        time2 = Time(hours, minutes)

        print(f"\nВремя 1: {time1.__str__()}, всего минут: {int(time1)}")
        print(f"Время 2: {time2.__str__()}, всего минут: {int(time2)}\n")

        print(f"[=] Эквивалентно ли время: {time1.__eq__(time2)}")
        print(f"[+] Сумма времён: {time1.__add__(time2).__str__()}")
        if time1.total_minutes() > time2.total_minutes():
            print(f"[-] Разность времён: {time1.__sub__(time2).__str__()}")
        else:
            print(f"[-] Разность времён: {time2.__sub__(time1).__str__()}")
        print(f"[<] Время 1 < Время 2: {time1.__lt__(time2).__str__()}")
        print(f"[<=] Время 1 <= Время 2: {time1.__le__(time2).__str__()}")

    except ValueError as e:
        print(f"Ошибка: {e}")
```

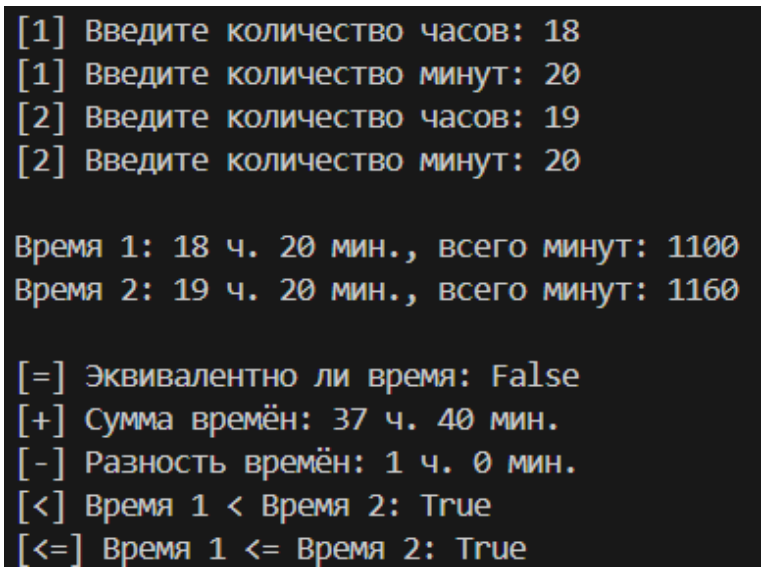

Рисунок 2 – Результат выполнения программы индивидуального задания 1

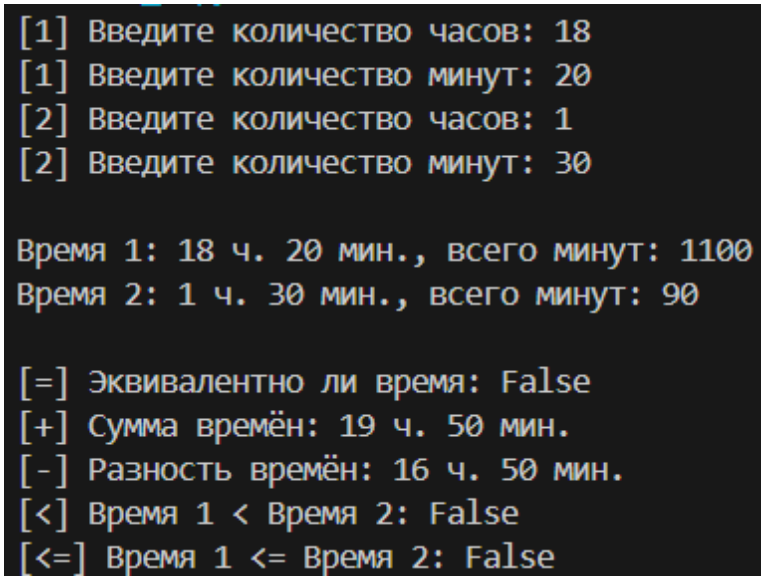

Рисунок 3 – Результат выполнения программы индивидуального задания 1

Индивидуальное 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. Карточка иностранного слова представляет собой словарейу, содержащую иностранное слово и его перевод. Для моделирования электронного словаря иностранных слов реализовать класс Dictionary. Данный класс имеет поле-название словаря и содержит список словарей WordCard, представляющий собой карточки иностранного слова. Название словаря задаётся при создании нового словаря, но должна быть предоставлена возможность его изменения во время работы. Карточки добавляются в словарь и удаляются из него. Реализовать поиск определённого слова как отдельный метод. Аргументом операции индексирования должно быть иностранное слово. В словаре не должно быть карточек-дублей. Реализовать операции объединения, пересечения и вычитания словарей. При реализации должен создаваться новый словарь, а исходных словари не должны изменяться. При объединении новый словарь должен содержать без повторений все слова, содержащиеся в обоих словаря-операндах. При пересечении новый словарь должен состоять из тех слов, которые имеются в обоих словарях-операндах. При вычитании новый словарь должен содержать слова первого словаря-операнда, отсутствующие во втором.

Листинг 3 – Код программы индивидуального задания 2

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

"""
Создать электронный словарь, который хранит карточки с иностранными
словами и их переводами, ограничивает их количество (size), поддерживает
добавление, удаление, поиск переводов, операции объединения, пересечения
и разности словарей, а также исключает дубли.
"""


class WordCard:
    def __init__(self, foreign_word, translation):
        """
        Конструктор карточки иностранного слова.
        """
        self.foreign_word = foreign_word
        self.translation = translation

    def __eq__(self, other):
```

```
        """
        Проверяет равенство двух карточек по иностранному слову.
        """
        if isinstance(other, WordCard):
            return self.foreign_word == other.foreign_word
        return False

    def __hash__(self):
        """
        Хеш-функция для поддержки операций с множествами.
        """
        return hash(self.foreign_word)


class Dictionary:
    MAX_SIZE = 100

    def __init__(self, name, size=MAX_SIZE):
        """
        Конструктор класса Dictionary.
        """
        self.name = name
        self.size = min(size, self.MAX_SIZE)
        self.cards = []
        self.count = 0

    def size(self):
        """
        Возвращает установленную длину словаря.
        """
        return self.size

    def add_card(self, foreign_word, translation):
        """
        Добавляет карточку в словарь.
        """
        if self.count >= self.size:
            raise ValueError("Словарь переполнен")

        new_card = WordCard(foreign_word, translation)

        # Проверка на наличие дублей
        if new_card in self.cards:
            print(f"Слово '{foreign_word}' уже есть в словаре")
            return

        self.cards.append(new_card)
        self.count += 1

    def remove_card(self, foreign_word):
        """
        Удаляет карточку из словаря.
```

```
        """
        self.cards = [card for card in self.cards if card.foreign_word != foreign_word]
        self.count = len(self.cards)

    def find_translation(self, foreign_word):
        """
        Ищет перевод для иностранного слова.
        """
        for card in self.cards:
            if card.foreign_word == foreign_word:
                return card.translation
        return None

    def __getitem__(self, foreign_word):
        """
        Перегрузка операции индексирования для поиска перевода.
        """
        return self.find_translation(foreign_word)

    def __setitem__(self, foreign_word, translation):
        """
        Перегрузка индексирования для добавления или обновления карточки.
        """
        for card in self.cards:
            if card.foreign_word == foreign_word:
                card.translation = translation
                return
        self.add_card(foreign_word, translation)

    def __add__(self, other):
        """
        Операция объединения словарей.
        """
        if not isinstance(other, Dictionary):
            return NotImplemented
        new_dict = Dictionary(f"{self.name} + {other.name}", size=self.MAX_SIZE)
        new_dict.cards = list(set(self.cards + other.cards))
        new_dict.count = len(new_dict.cards)
        return new_dict

    def __and__(self, other):
        """
        Операция пересечения словарей.
        """
        if not isinstance(other, Dictionary):
            return NotImplemented
        new_dict = Dictionary(f"{self.name} & {other.name}", size=self.MAX_SIZE)
        new_dict.cards = list(set(self.cards) & set(other.cards))
        new_dict.count = len(new_dict.cards)
        return new_dict

    def __sub__(self, other):
```

```
        """
        Операция вычитания словарей.
        """
        if not isinstance(other, Dictionary):
            return NotImplemented
        new_dict = Dictionary(f"{self.name} - {other.name}", size=self.MAX_SIZE)
        new_dict.cards = [card for card in self.cards if card not in other.cards]
        new_dict.count = len(new_dict.cards)
        return new_dict

    def __str__(self):
        """
        Строковое представление словаря.
        """
        cards_str = "\n".join(f"{card.foreign_word}: {card.translation}" for card in self.cards)
        return f"Словарь '{self.name}' ({self.count}/{self.size}):\n{cards_str}"


# Пример использования:
if __name__ == "__main__":
    dict1 = Dictionary("Еда")
    dict1.add_card("Apple", "Яблоко")
    dict1.add_card("Bread", "Хлеб")
    dict1.add_card("Milk", "Молоко")

    dict2 = Dictionary("Природа")
    dict2.add_card("Apple", "Яблоко")
    dict2.add_card("Tree", "Дерево")
    dict2.add_card("River", "Река")

    print(f"{dict1}\n") # Вывод словаря 1
    print(f"{dict2}\n") # Вывод словаря 2

    print("\nОбъединение словарей:")
    print(dict1.__add__(dict2))

    print("\nПересечение словарей:")
    print(dict1.__and__(dict2))

    print("\nРазность словарей:")
    print(dict1.__sub__(dict2))

    dict1.name = "Продовольствие" # Изменение названия словаря
    print(f"\n{dict1}")
```

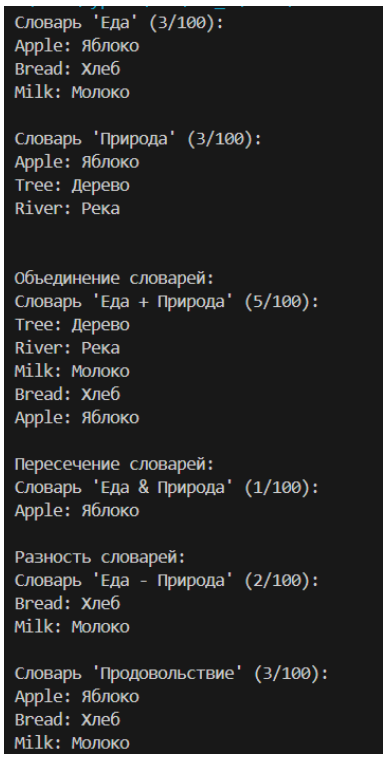

Рисунок 4 – Результат выполнения программы индивидуального задания 2

Ответы на контрольные вопросы:

1. Какие средства существуют в Python для перегрузки операций?

Для перегрузки операций в Python используются магические методы (__ad__, __eq__, __len__ и т.д.) для перегрузки стандартных операций.

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

Арифметические: __add__, __sub__, __mul__, __truediv__, __floordiv__, __mod__, __pow__.

Операции отношения: __eq__, __ne__, __lt__, __le__, __gt__, __ge__.

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

__add__ - при использовании + с объектами слева;

__iadd__ - при использовании +=;

__radd__ - при использовании + с объектами справа, если слева на определён __add__.

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

__new__ создаёт новый объект, вызывается перед __init__.

__init__ - инициализирует объект после его создания.

5. Чем отличаются методы __str__ и __repr__?

__str__ - читаемое представление объекта для пользователя.

__repr__ - формальное представление объекта для разработчика, используется в консоли.

Выводы: В процессе выполнения лабораторной работы были приобретены навыки по перегрузке операторов при написании программ с помощью языка программирования Python версии 3.x.

Ссылка на репозиторий: IUnnamedUserI/OOP_2: Объектно-ориентированное программирование. Лабораторная работа №2