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 20

Выполнила:
Михеева Елена Александровна
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 Р.А.-доцент департамента цифровых, роботехнических систем и электроники института перспективной инженерии

_____
(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. Приступили к выполнению лабораторной работы. Задание 1: выполнить индивидуальное задание 1 лабораторной работы 1, максимально задействовав имеющиеся в Python средства перегрузки операторов.

```python
class GeometricProgression:
    def __init__(self, first=1.0, second=1.0):
        """
        Конструктор для инициализации 1-го элемента прогрессии
        и постоянного отношения.
        Проверяет корректность введённых значений.
        """
        if not isinstance(first, (int, float)):
            raise ValueError("Первый элемент прогрессии должен быть числом.")
        if not isinstance(second, (int, float)):
            raise ValueError("Постоянное отношение должно быть числом.")

        self.first = float(first)  # Первый элемент прогрессии a0
        self.second = float(second)  # Постоянное отношение r

    def __getitem__(self, j):
        """
        Перегрузка оператора [], для получения j-го элемента прогрессии.
        """
        if not isinstance(j, int) or j < 0:
            raise ValueError("Индекс должен быть положительным целым числом.")
        return self.first * (self.second ** j)

    def __mul__(self, scalar):
        """
        Перегрузка оператора * для умножения прогрессии на скаляр.
        Возвращает новую прогрессию с умноженным первым элементом и отношением.
        """
        if not isinstance(scalar, (int, float)):
            raise ValueError("Множитель должен быть числом.")
        return GeometricProgression(self.first * scalar, self.second * scalar)

    def __eq__(self, other):
        """
        Перегрузка оператора == для сравнения двух прогрессий.
        Прогрессии считаются равными, если 1-ый элемент и отношение одинаковы.
        """
        if not isinstance(other, GeometricProgression):
            return False
        return self.first == other.first and self.second == other.second

    def __str__(self):
        """
        Перегрузка оператора str для удобного вывода информации о прогрессии.
        """
        return (
            f"1-ый элемент: {self.first}, Постоянное отношение: {self.second}"
        )

    def read(self):
        """
        Ввод данных с клавиатуры для первого элемента и постоянного отношения.
        """
        self.first = float(input("Введите первый элемент прогрессии (a0): "))
        self.second = float(input("Введите постоянное отношение (r): "))

    def display(self):
        """
        Вывод данных на экран, включая j-й элемент прогрессии.
        """
        print(self)  # Используем перегруженный __str__ для вывода
        j = int(input("Введите номер элемента прогрессии для вычисления: "))

        if not isinstance(j, int) or j < 0:
            raise ValueError("Индекс должен быть положительным целым числом.")

        print(f"{j}-й элемент прогрессии: {self[j]}")


# Внешняя функция для создания объекта GeometricProgression
def make_geometric_progression(first, second):
    try:
        return GeometricProgression(first, second)
    except ValueError as e:
        print(f"Ошибка: {e}")
        exit(1)


if __name__ == '__main__':
    # Создаём первый объект прогрессии
    gp1 = GeometricProgression(2, 3)  # Первый элемент = 2, отношение = 3
    gp1.display()  # Выводим начальные значения и вычисляем элемент прогрессии

    # Ввод значений для второго объекта с клавиатуры
    gp2 = GeometricProgression()
    gp2.read()  # Чтение значений
    gp2.display()  # Выводим значения 2-ой прогрессии и вычисляем элемент

    # Сравнение прогрессий
    print("Прогрессии равны?", gp1 == gp2)

    # Умножение прогрессии на скаляр
    gp3 = gp1 * 2  # Умножаем прогрессию на 2
    print("Новая прогрессия после умножения:", gp3)
```

Рисунок 1. Код программы задания №1

```
(base) elenamiheeva@MacBook-Pro-Elena Oop_2 % python3 program/ex_1.py
1-ый элемент: 2.0, Постоянное отношение: 3.0
Введите номер элемента прогрессии для вычисления: 7
7-й элемент прогрессии: 4374.0
Введите первый элемент прогрессии (a0): 3.3
Введите постоянное отношение (r): 23
1-ый элемент: 3.3, Постоянное отношение: 23.0
Введите номер элемента прогрессии для вычисления: 56
56-й элемент прогрессии: 5.960363641898442e+76
Прогрессии равны? False
Новая прогрессия после умножения: 1-ый элемент: 4.0, Постоянное отношение: 6.0
```

Рисунок 2. Результат работы программы задания №1

2. Задание 2: Одна запись в списке запланированных дел представляет собой словарь DailyItem, которая содержит время начала и окончания работы, описание и признак выполнения. Реализовать класс DailySchedule, представляющий собой план работ на день. Реализовать методы

добавления, удаления и изменения планируемой работы. При добавлении проверять корректность временных рамок (они не должны пересекаться с уже запланированными мероприятиями). Реализовать метод поиска свободного промежутка времени. Условие поиска задает размер искомого интервала, а также временные рамки, в которые он должен попадать. Метод поиска возвращает словарь Dailyltern с пустым описанием вида работ. Реализовать операцию генерации объекта Redo (еще раз), содержащего список дел, не выполненных в течение дня, из объекта типа DailySchedule.

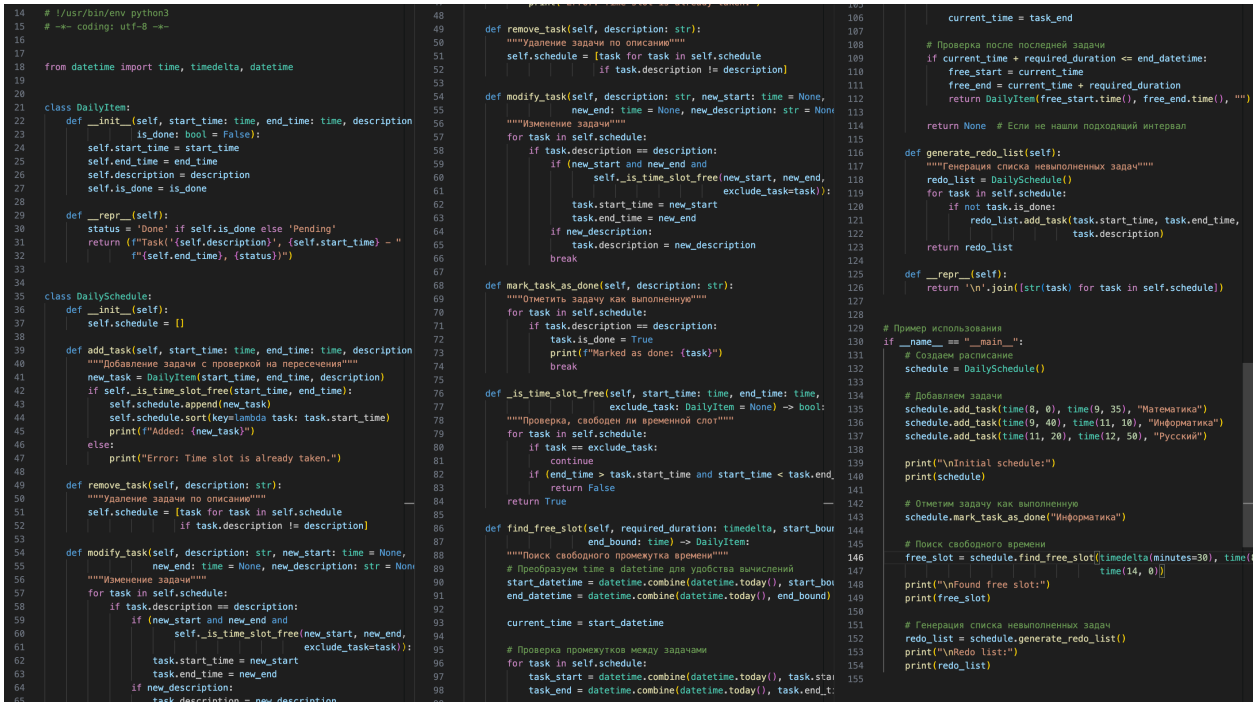

Рисунок 3. Код программы задания №2

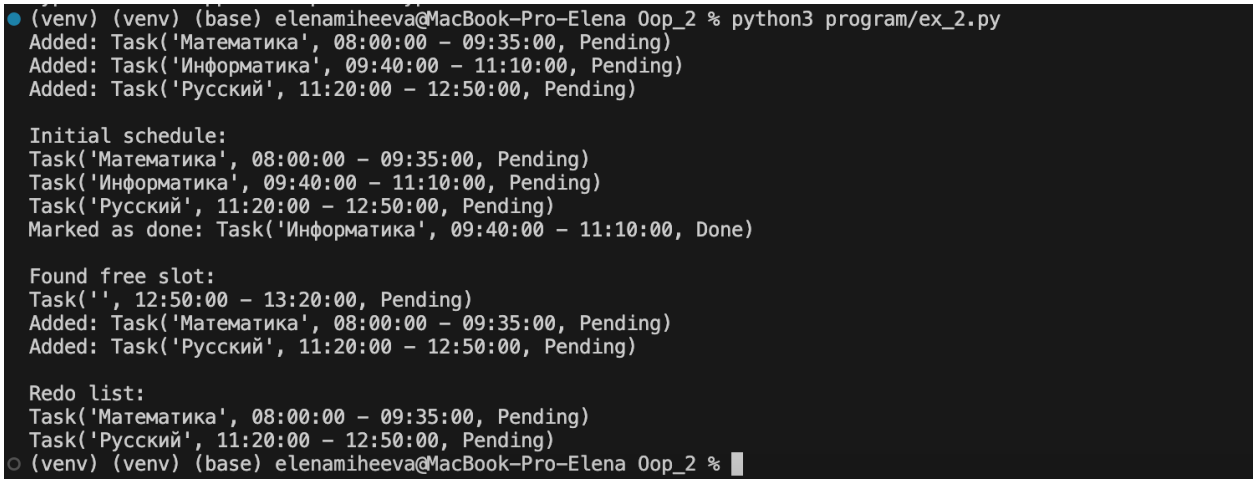

Рисунок 4. Результат работы программы задания №2

Ответы на контрольные вопросы:

1. Какие средства существуют в Python для перегрузки операций?

В Python перегрузка операций осуществляется с помощью специальных методов, известных как магические методы или дандер-методы (double underscore methods). Эти методы позволяют настраивать поведение встроенных операторов, таких как арифметические операции (например, сложение, вычитание, умножение и деление) и операции сравнения (например, равенство и неравенство). Основные магические методы включают __add__, __sub__, __mul__, __eq__ и многие другие, которые можно реализовать в классах для изменения их стандартного поведения при использовании операторов.

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

Для перегрузки арифметических операций в Python используются методы, такие как __add__ для сложения, __sub__ для вычитания, __mul__ для умножения, __truediv__ для деления и __pow__ для возведения в степень. В свою очередь, для перегрузки операций отношения применяются методы __eq__ для проверки равенства, __ne__ для неравенства, __lt__ для проверки "меньше", __le__ для "меньше или равно", __gt__ для "больше" и __ge__ для "больше или равно". Реализация этих методов позволяет пользователю определять, как объекты пользовательских классов взаимодействуют с стандартными операторами.

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

Метод __add__ вызывается при использовании оператора + между объектами, что позволяет определить, как два объекта складываются, как показано в примере с классом Vector, где v1 + v2 вызывает v1.__add__(v2). Метод __iadd__ вызывается при использовании оператора +=, и он изменяет текущий объект, как видно в примере, где v1 += v2 вызывает v1.__iadd__(v2). Метод __radd__ вызывается, когда левый операнд не поддерживает операцию

и Python пытается выполнить операцию с правым операндом, как в случае, когда к кортежу (3, 4) добавляется объект Vector, вызывая v1.__radd__(v2).

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

Метод __new__ предназначен для создания нового объекта и вызывается перед методом __init__. Он должен возвращать новый экземпляр класса и особенно полезен при работе с классами, наследуемыми от встроенных типов, а также при реализации паттерна Singleton. В отличие от него, метод __init__ служит для инициализации уже созданного объекта, устанавливая его начальные значения и выполняя дополнительные настройки, что делает __new__ и __init__ различными по своей функциональности, хотя и работающими совместно при создании экземпляров класса.

5. Чем отличаются методы __str__ и __repr__ ?

Метод __str__ предназначен для создания строкового представления объекта, удобного для пользователя, и вызывается при использовании функции print() или str(). В то же время метод __repr__ служит для создания официального строкового представления объекта, которое должно быть максимально информативным и точным, и вызывается, когда используется функция repr() или при вводе объекта в интерактивной консоли. Основное различие: __str__ фокусируется на читабельности, поэтому его вывод может приносить некоторые детали в жертву ясности. __repr__ отдает приоритет однозначности, предоставляя подробное представление состояния объекта, часто включая всю необходимую информацию для реконструкции объекта

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