

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, робототехнических 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.

```
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 DailyItem с пустым описанием вида работ. Реализовать операцию генерации объекта Redo (еще раз), содержащего список дел, не выполненных в течение дня, из объекта типа DailySchedule.

```

14 # /usr/bin/env python3
15 # -*- coding: utf-8 -*-
16
17 from datetime import time, timedelta, datetime
18
19 class DailyItem:
20     def __init__(self, start_time: time, end_time: time, description: str, is_done: bool = False):
21         self.start_time = start_time
22         self.end_time = end_time
23         self.description = description
24         self.is_done = is_done
25
26     def __repr__(self):
27         status = 'Done' if self.is_done else 'Pending'
28         return f"Task('{self.description}', {self.start_time} - {self.end_time}, {status})"
29
30 class DailySchedule:
31     def __init__(self):
32         self.schedule = []
33
34     def add_task(self, start_time: time, end_time: time, description: str):
35         """Добавление задачи с проверкой на пересечение"""
36         new_task = DailyItem(start_time, end_time, description)
37         if self.is_time_slot_free(start_time, end_time):
38             self.schedule.append(new_task)
39         else:
40             print("Error: Time slot is already taken.")
41
42     def remove_task(self, description: str):
43         """Удаление задачи по описанию"""
44         self.schedule = [task for task in self.schedule if task.description != description]
45
46     def modify_task(self, description: str, new_start: time = None, new_end: time = None, new_description: str = None):
47         """Изменение задачи"""
48         for task in self.schedule:
49             if task.description == description:
50                 if new_start and new_end:
51                     task.start_time = new_start
52                     task.end_time = new_end
53                 if new_description:
54                     task.description = new_description
55
56     def mark_task_as_done(self, description: str):
57         """Отметить задачу как выполненную"""
58         for task in self.schedule:
59             if task.description == description:
60                 task.is_done = True
61                 print(f"Marked as done: {task}")
62                 break
63
64     def is_time_slot_free(self, start_time: time, end_time: time, exclude_task: DailyItem = None) -> bool:
65         """Проверка, свободен ли временной слот"""
66         for task in self.schedule:
67             if task == exclude_task:
68                 continue
69             if (end_time > task.start_time and start_time < task.end_time):
70                 return False
71         return True
72
73     def find_free_slot(self, required_duration: timedelta, start_bound: time, end_bound: time) -> DailyItem:
74         """Поиск свободного промежутка времени"""
75         # Преобразуем time в datetime для удобства вычислений
76         start_datetime = datetime.combine(datetime.today(), start_bound)
77         end_datetime = datetime.combine(datetime.today(), end_bound)
78         current_time = start_datetime
79
80         # Проверка промежутков между задачами
81         for task in self.schedule:
82             task_start = datetime.combine(datetime.today(), task.start_time)
83             task_end = datetime.combine(datetime.today(), task.end_time)
84
85             if current_time < task_start:
86                 # Проверка после последней задачи
87                 if current_time + required_duration <= task_end:
88                     free_start = current_time
89                     free_end = current_time + required_duration
90                     return DailyItem(free_start, free_end, "", True)
91
92             current_time = task_end
93
94     def generate_redo_list(self):
95         """Генерация списка невыполненных задач"""
96         redo_list = DailySchedule()
97         for task in self.schedule:
98             if not task.is_done:
99                 redo_list.add_task(task.start_time, task.end_time, task.description)
100
101     def __repr__(self):
102         return '\n'.join([str(task) for task in self.schedule])
103
104 # Пример использования
105 if __name__ == "__main__":
106     # Создаем расписание
107     schedule = DailySchedule()
108
109     # Добавляем задачи
110     schedule.add_task(time(8, 0), time(9, 35), "Математика")
111     schedule.add_task(time(9, 40), time(11, 10), "Информатика")
112     schedule.add_task(time(11, 20), time(12, 50), "Русский")
113
114     print("Initial schedule:")
115     print(schedule)
116
117     # Отметим задачу как выполненную
118     schedule.mark_task_as_done("Информатика")
119
120     # Поиск свободного времени
121     free_slot = schedule.find_free_slot(timedelta(minutes=30), time(12, 0), time(14, 0))
122     print(f"Found free slot: {free_slot}")
123
124     # Генерация списка невыполненных задач
125     redo_list = schedule.generate_redo_list()
126     print("Redo list:")
127     print(redo_list)

```

Рисунок 3. Код программы задания №2

```

(venv) (venv) (base) elenamiheeva@MacBook-Pro-Elena Oop_2 % python3 program/ex_2.py
Added: Task('Математика', 08:00:00 - 09:35:00, Pending)
Added: Task('Информатика', 09:40:00 - 11:10:00, Pending)
Added: Task('Русский', 11:20:00 - 12:50:00, Pending)

Initial schedule:
Task('Математика', 08:00:00 - 09:35:00, Pending)
Task('Информатика', 09:40:00 - 11:10:00, Pending)
Task('Русский', 11:20:00 - 12:50:00, Pending)
Marked as done: Task('Информатика', 09:40:00 - 11:10:00, Done)

Found free slot:
Task('', 12:50:00 - 13:20:00, Pending)
Added: Task('Математика', 08:00:00 - 09:35:00, Pending)
Added: Task('Русский', 11:20:00 - 12:50:00, Pending)

Redo list:
Task('Математика', 08:00:00 - 09:35:00, Pending)
Task('Русский', 11:20:00 - 12:50:00, Pending)
(venv) (venv) (base) elenamiheeva@MacBook-Pro-Elena Oop_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.