МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Объектно-ориентированное программирование

Отчет по лабораторной работе №4.2

Перегрузка операторов в языке Python

Выполнил студент группы

ИВТ-б-о-21-1

Урусов М.А. « »_____20__г.

Подпись студента_____

Работа защищена « »_____20__г.

Проверил доцент
Кафедры инфокоммуникаций, старший преподаватель
Воронкин Р.А.

_____
(подпись)

Ставрополь 2023

Наследование и полиморфизм в языке Python.

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

Порядок выполнения работы:

Задание 1.

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

Код программы:

```
    if not result:
        logging.info(f"Самолеты типа {jet} не найдены")
    return result


1 usage  new *
def save_planes(file_name, staff):
    try:
        with open(file_name, "w", encoding="utf-8") as fout:
            json.dump(staff, fout, ensure_ascii=False, indent=4)
            logging.info(f"Данные успешно сохранены в файл {file_name}")
    except Exception as e:
        logging.error(f"Ошибка при сохранении данных: {e}")


1 usage  new *
def load_planes(file_name):
    try:
        with open(file_name, "r", encoding="utf-8") as fin:
            return json.load(fin)
    except FileNotFoundError:
        logging.warning(f"Файл {file_name} не найден. Будет создан новый файл.")
        return []
    except Exception as e:
        logging.error(f"Ошибка при загрузке данных: {e}")
        return []


1 usage  new *
def main(command_line=None):
    parser = argparse.ArgumentParser("planes")
    subparsers = parser.add_subparsers(dest="command")

    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument( *name_or_flags: "filename", action="store", help="The data file name")

    add = subparsers.add_parser( name: "add", parents=[file_parser], help="Add a new plane")
    add.add_argument( *name_or_flags: "-d", "--destination", action="store", required=True, help="The plane's destination")
    add.add_argument( *name_or_flags: "-n", "--num", action="store", type=int, required=True, help="The plane's number")
```

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

import argparse
import json
import os.path
import pathlib
import logging

# Настройка логгирования
logging.basicConfig(filename='planes.log', level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s',
                    datefmt='%Y-%m-%d %H:%M:%S')


1 usage  new *
def add_plane(staff, destination, num, typ):
    staff.append({"destination": destination, "num": num, "typ": typ})
    logging.info(f"Добавлен самолет: пункт назначения {destination}, номер {num}, тип {typ}")
    return staff


2 usages  new *
def display_planes(staff):
    if staff:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format( *args: '-' * 4, '-' * 30, '-' * 20, '-' * 15)
        print(line)
        print('| {:^4} | {:^30} | {:^20} | {:^15} |'.format( *args: "No", "Пункт назначения", "Номер рейса", "Тип самолета"))
        print(line)
        for idx, plane in enumerate(staff, 1):
            print('| {:>4} | {:<30} | {:<20} | {:>15} |'.format( *args: idx, plane.get('destination', ''), plane.get('num', 0),
                                                                  plane.get('typ', '')))
            print(line)
    else:
        print("Список самолетов пуст")
        logging.info("Попытка отобразить пустой список самолетов")


1 usage  new *
def select_planes(staff, jet):
    result = [plane for plane in staff if jet == plane.get('typ', '')]
```

Результат работы программы:

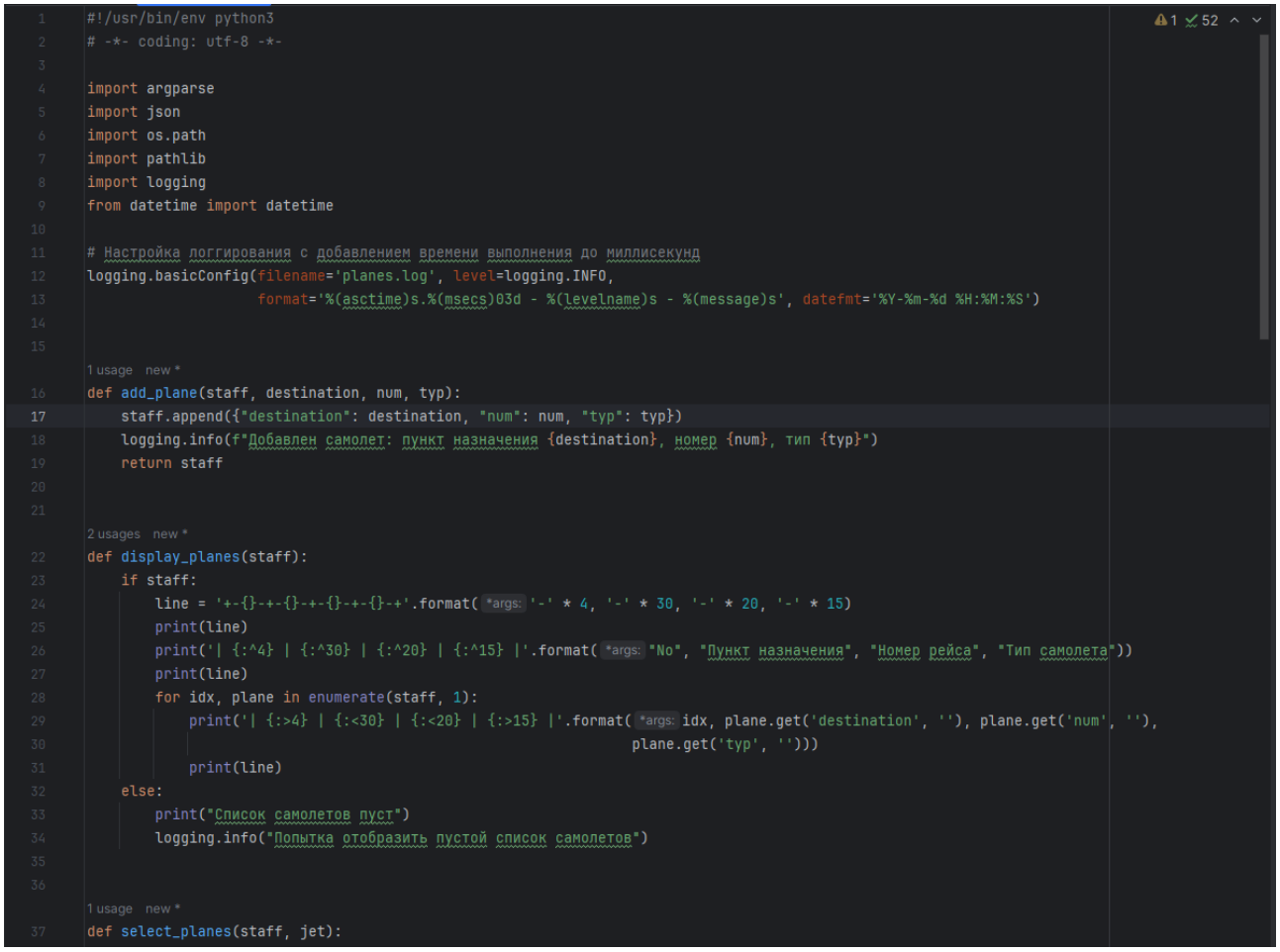
```
C:\Users\den-n\AppData\Local\Programs\Python\Python312\python.exe D:\git\Git\Lab_4.2\Задания\Task_1.py
Число: 12.34
Введите целую часть числа: 41
Введите дробную часть числа (как положительное целое число): 654
Число: 41.654
Введите множитель: 2
После умножения:
Число: 83.308

Process finished with exit code 0
```

Рисунок 1. Результат работы программы

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.

Используя класс Bill , реализовать класс ListPayer. Класс содержит список плательщиков за
телефонные услуги, дату создания списка, номер списка. Поля одного элемента списка —
это: плательщик (класс Bill), признак оплаты, дата платежа, сумма платежа. Реализовать
методы добавления плательщиков в список и удаления их из него; метод поиска
плательщика по номеру телефона и по фамилии, по дате платежа. Метод вычисления
полной стоимости платежей всего списка. Реализовать операцию объединения и операцию
пересечения списков. Реализовать операцию генерации конкретного объекта Group
(группа), содержащего список плательщиков, из объекта типа ListPayer. Должна быть
возможность выбирать группу плательщиков по признаку оплаты, по атрибутам, по дате
платежа, по номеру телефона.

Код программы:



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

import argparse
import json
import os.path
import pathlib
import logging
from datetime import datetime

# Настройка логгирования с добавлением времени выполнения до миллисекунд
logging.basicConfig(filename='planes.log', level=logging.INFO,
                    format='%(asctime)s.%(msecs)03d - %(levelname)s - %(message)s', datefmt='%Y-%m-%d %H:%M:%S')


1 usage  new *
def add_plane(staff, destination, num, typ):
    staff.append({"destination": destination, "num": num, "typ": typ})
    logging.info(f"Добавлен самолет: пункт назначения {destination}, номер {num}, тип {typ}")
    return staff


2 usages  new *
def display_planes(staff):
    if staff:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format( *args: '-' * 4, '-' * 30, '-' * 20, '-' * 15)
        print(line)
        print('| {:^4} | {:^30} | {:^20} | {:^15} |'.format( *args: "No", "Пункт назначения", "Номер рейса", "Тип самолета"))
        print(line)
        for idx, plane in enumerate(staff, 1):
            print('| {:>4} | {:<30} | {:<20} | {:>15} |'.format( *args: idx, plane.get('destination', ''), plane.get('num', ''),
                                                                  plane.get('typ', '')))
            print(line)
    else:
        print("Список самолетов пуст")
        logging.info("Попытка отобразить пустой список самолетов")


1 usage  new *
def select_planes(staff, jet):
```

```python
    result = [plane for plane in staff if jet == plane.get('typ', '')]
    if not result:
        logging.info(f"Самолеты типа {jet} не найдены")
    return result


1 usage  new *
def save_planes(file_name, staff):
    try:
        with open(file_name, "w", encoding="utf-8") as fout:
            json.dump(staff, fout, ensure_ascii=False, indent=4)
            logging.info(f"Данные успешно сохранены в файл {file_name}")
    except Exception as e:
        logging.error(f"Ошибка при сохранении данных: {e}")


1 usage  new *
def load_planes(file_name):
    try:
        with open(file_name, "r", encoding="utf-8") as fin:
            return json.load(fin)
    except FileNotFoundError:
        logging.warning(f"Файл {file_name} не найден. Будет создан новый файл.")
        return []
    except Exception as e:
        logging.error(f"Ошибка при загрузке данных: {e}")
        return []


1 usage  new *
def main(command_line=None):
    start_time = datetime.now()

    parser = argparse.ArgumentParser("planes")
    subparsers = parser.add_subparsers(dest="command")

    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument( *name_or_flags: "filename", action="store", help="The data file name")

    add = subparsers.add_parser( name: "add", parents=[file_parser], help="Add a new plane")
```

Результат работы программы:

```
C:\Users\den-n\AppData\Local\Programs\Python\Python312\python.exe D:\git\6it\Lab_4.2\Задания\Task_2.py
Общая сумма платежей: 250
Поиск по номеру телефона '123456789': [Иванов Иван, 123456789, 2023-02-01, 100, Оплачено]
Поиск по фамилии 'Петров Петр': [Петров Петр, 987654321, 2023-02-02, 150, Не оплачено]
Поиск по дате платежа '2023-02-01': [Иванов Иван, 123456789, 2023-02-01, 100, Оплачено]
Все плательщики: Номер списка: 001, Дата создания: 2023-01-01, Счета: [Иванов Иван, 123456789, 2023-02-01, 100, Оплачено, Петров Петр, 987654321, 2023-02-02, 150, Не оплачено]

Process finished with exit code 0
```

Рисунок 2. Результат работы программы

Ответы на вопросы:

1. Какие средства существуют в Python для перегрузки операций?

В python имеются методы, которые не вызываются напрямую, а вызываются встроенными функциями или операторами. С их помощью можно перегрузить операции.

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

Пример: __add___- сложение, __sub___- вычитание, __mul___- умножение.

3. В каких случаях будут вызваны следующие методы: __add__, __iadd___и __radd__?

__add___- вызывается при сложении двух чисел оператором «+». В случае, если это сделать не удаётся, вызываются __iadd___и __radd__, они делают то же самое, что и арифметические операторы, перечисленные выше, но для аргументов, находящихся справа, и только в случае, если для левого операнда не определён соответствующий метод.

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

Управляет созданием экземпляра. В качестве обязательного аргумента принимает класс (не путать с экземпляром). Должен возвращать экземпляр класса для его последующей его передачи методу __init___

5. Чем отличаются методы __str___и __repr__?

__str___- вызывается функциями str, print и format. Возвращает строковое представление объекта.

__repr___- вызывается встроенной функцией repr; возвращает "сырые" данные, использующиеся для внутреннего представления в python.

Вывод: в ходе работы были приобретены навыки по перегрузке операторов при написании программ с использованием языка программирования Python версии 3.x.