Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №3
дисциплины «Анализ данных»
Вариант №5

Выполнил:
Михеева Елена Александровна
2 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)

Руководитель практики:
Воронкин Р.А., канд. техн. наук,
доцент, доцент кафедры
инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Работа с данными формата JSON в языке Python

Цель: приобретение построения приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

1. Был проработан пример из лабораторной работы.

```
(venv) (base) elenamiheeva@MacBook-Pro-Elena Data_Analysis_2.17 % python3 program/workers.py -h
usage: workers [-h] [--version] {add,display,select} ...

positional arguments:
  {add,display,select}
    add                 Add a new worker
    display             Display all workers
    select              Select the workers

options:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
(venv) (base) elenamiheeva@MacBook-Pro-Elena Data_Analysis_2.17 % python3 program/workers.py add -h
usage: workers add [-h] -n NAME [-p POST] -y YEAR filename

positional arguments:
  filename              The data file name

options:
  -h, --help            show this help message and exit
  -n NAME, --name NAME  The worker's name
  -p POST, --post POST  The worker's post
  -y YEAR, --year YEAR  The year of hiring
```

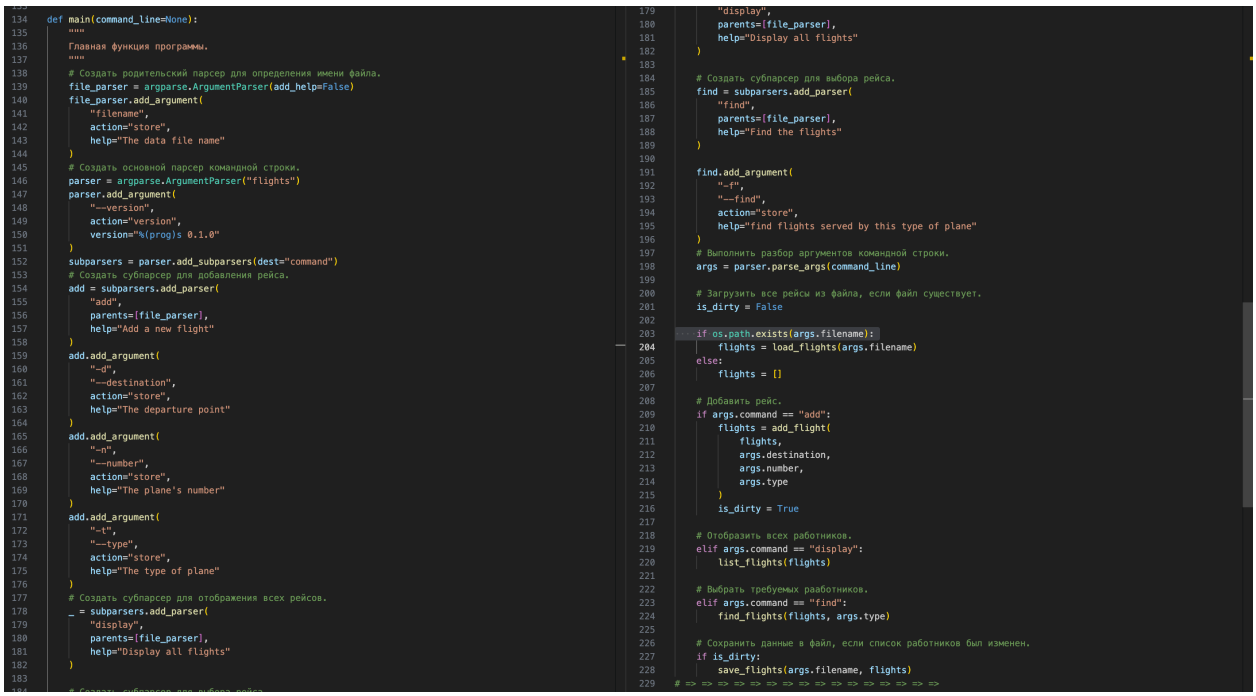
Рисунок 1. Тестирование примера 1

```
(venv) (base) elenamiheeva@MacBook-Pro-Elena Data_Analysis_2.17 % python3 program/workers.py add -n ridor -p singer -y 2012 data.json
(venv) (base) elenamiheeva@MacBook-Pro-Elena Data_Analysis_2.17 % python3 program/workers.py display data.json
+------+--------------------------------+----------------------+-----------+
|  №   |             Ф.И.О.             |      Должность       |   Год     |
+------+--------------------------------+----------------------+-----------+
|    1 | helen                          | boss                 |     2017 |
+------+--------------------------------+----------------------+-----------+
|    2 | sidor                          | singer               |     2012 |
+------+--------------------------------+----------------------+-----------+
|    3 | ridor                          | singer               |     2012 |
+------+--------------------------------+----------------------+-----------+
|    4 | gidor                          | singer               |     2012 |
+------+--------------------------------+----------------------+-----------+
|    5 | ridor                          | singer               |     2012 |
+------+--------------------------------+----------------------+-----------+
```

Рисунок 2. Выполнение примера 1

2. Приступили к выполнению индивидуального задания: использовать словарь, содержащий следующие ключи: название пункта назначения рейса; номер рейса; тип самолета. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в список, состоящий из словарей заданной структуры; записи должны быть размещены в алфавитном порядке по названиям пунктов назначения; вывод на экран пунктов назначения и номеров рейсов, обслуживаемых самолетом, тип которого введен с клавиатуры; если таких рейсов нет, выдать на дисплей соответствующее сообщение. Необходимо дополнительно реализовать интерфейс командной строки.

3. Был реализован интерфейс командной строки для программы.



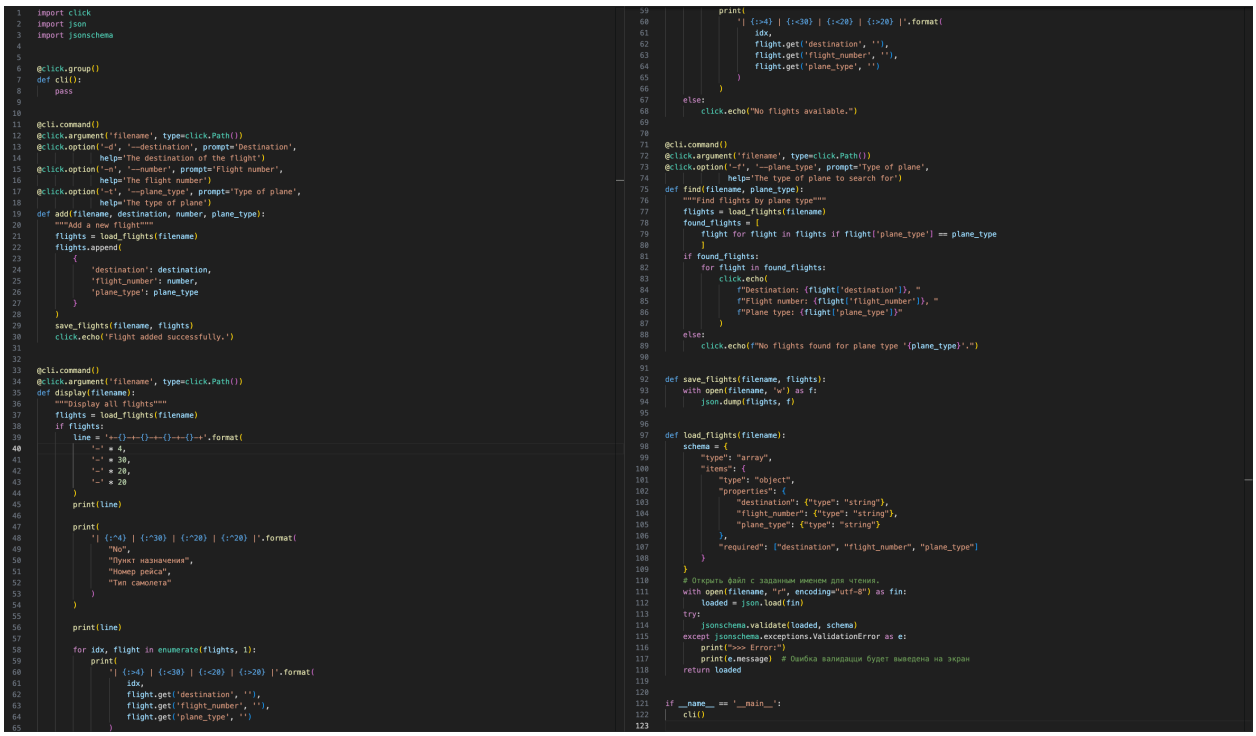
```python
def main(command_line=None):
    """
    Главная функция программы.
    """
    # Создать родительский парсер для определения имени файла.
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "filename",
        action="store",
        help="The data file name"
    )
    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("flights")
    parser.add_argument(
        "--version",
        action="version",
        version="%(prog)s 0.1.0"
    )
    subparsers = parser.add_subparsers(dest="command")
    # Создать субпарсер для добавления рейса.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new flight"
    )
    add.add_argument(
        "-d",
        "--destination",
        action="store",
        help="The departure point"
    )
    add.add_argument(
        "-n",
        "--number",
        action="store",
        help="The plane's number"
    )
    add.add_argument(
        "-t",
        "--type",
        action="store",
        help="The type of plane"
    )
    # Создать субпарсер для отображения всех рейсов.
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all flights"
    )

    # Создать субпарсер для выбора рейса.
    find = subparsers.add_parser(
        "find",
        parents=[file_parser],
        help="Find the flights"
    )

    find.add_argument(
        "-f",
        "--find",
        action="store",
        help="find flights served by this type of plane"
    )
    # Выполнить разбор аргументов командной строки.
    args = parser.parse_args(command_line)

    # Загрузить все рейсы из файла, если файл существует.
    is_dirty = False

    if os.path.exists(args.filename):
        flights = load_flights(args.filename)
    else:
        flights = []

    # Добавить рейс.
    if args.command == "add":
        flights = add_flight(
            flights,
            args.destination,
            args.number,
            args.type
        )
        is_dirty = True

    # Отобразить всех работников.
    elif args.command == "display":
        list_flights(flights)

    # Выбрать требуемых раработников.
    elif args.command == "find":
        find_flights(flights, args.type)

    # Сохранить данные в файл, если список работников был изменен.
    if is_dirty:
        save_flights(args.filename, flights)
# => => => => => => => => => => => => => => => => =>
```

Рисунок 6. Функция main()

```
(venv) (base) elenamiheeva@MacBook-Pro-Elena Data_Analysis_2.17 % python3 program/fly.py -h
usage: flights [-h] [--version] {add,display,find} ...

positional arguments:
  {add,display,find}
    add               Add a new flight
    display           Display all flights
    find              Find the flights

options:
  -h, --help          show this help message and exit
  --version           show program's version number and exit
(venv) (base) elenamiheeva@MacBook-Pro-Elena Data_Analysis_2.17 % python3 program/fly.py add -h
usage: flights add [-h] [-d DESTINATION] [-n NUMBER] [-t TYPE] filename

positional arguments:
  filename              The data file name

options:
  -h, --help            show this help message and exit
  -d DESTINATION, --destination DESTINATION
                        The departure point
  -n NUMBER, --number NUMBER
                        The plane's number
  -t TYPE, --type TYPE  The type of plane
(venv) (base) elenamiheeva@MacBook-Pro-Elena Data_Analysis_2.17 % python3 program/fly.py add -d  tallin -n 342 -t plane data1.json
>>> Data is obtained!
(venv) (base) elenamiheeva@MacBook-Pro-Elena Data_Analysis_2.17 % python3 program/fly.py display data1.json
>>> Data is obtained!
+-----+-------------------------------+--------------------+----------------------+
|  No |        Пункт назначения       |     Номер рейса    |     Тип самолета     |
+-----+-------------------------------+--------------------+----------------------+
|   1 | sochi                         | 678                |               plane1 |
|   2 | moscow                        | 341                |                plane |
|   3 | tallin                        | 342                |                plane |
```

Рисунок 7. Результат работы программы

Для данной задачи был реализован интерфейс командной строки с использованием пакета click.



```python
import click
import json
import jsonschema


@click.group()
def cli():
    pass


@cli.command()
@click.argument('filename', type=click.Path())
@click.option('-d', '--destination', prompt='Destination',
              help='The destination of the flight')
@click.option('-n', '--number', prompt='Flight number',
              help='The flight number')
@click.option('-t', '--plane_type', prompt='Type of plane',
              help='The type of plane')
def add(filename, destination, number, plane_type):
    """Add a new flight"""
    flights = load_flights(filename)
    flights.append(
        {
            'destination': destination,
            'flight_number': number,
            'plane_type': plane_type
        }
    )
    save_flights(filename, flights)
    click.echo('Flight added successfully.')


@cli.command()
@click.argument('filename', type=click.Path())
def display(filename):
    """Display all flights"""
    flights = load_flights(filename)
    if flights:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 20
        )
        print(line)

        print(
            '| {:^4} | {:^30} | {:^20} | {:^20} |'.format(
                "No",
                "Пункт назначения",
                "Номер рейса",
                "Тип самолета"
            )
        )

        print(line)

        for idx, flight in enumerate(flights, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>20} |'.format(
                    idx,
                    flight.get('destination', ''),
                    flight.get('flight_number', ''),
                    flight.get('plane_type', '')
                )
            )

        print(line)
    else:
        click.echo("No flights available.")


@cli.command()
@click.argument('filename', type=click.Path())
@click.option('-f', '--plane_type', prompt='Type of plane',
              help='The type of plane to search for')
def find(filename, plane_type):
    """Find flights by plane type"""
    flights = load_flights(filename)
    found_flights = [
        flight for flight in flights if flight['plane_type'] == plane_type
    ]
    if found_flights:
        for flight in found_flights:
            click.echo(
                f"Destination: {flight['destination']}, "
                f"Flight number: {flight['flight_number']}, "
                f"Plane type: {flight['plane_type']}"
            )
    else:
        click.echo(f"No flights found for plane type '{plane_type}'.")


def save_flights(filename, flights):
    with open(filename, 'w') as f:
        json.dump(flights, f)


def load_flights(filename):
    schema = {
        "type": "array",
        "items": {
            "type": "object",
            "properties": {
                "destination": {"type": "string"},
                "flight_number": {"type": "string"},
                "plane_type": {"type": "string"}
            },
            "required": ["destination", "flight_number", "plane_type"]
        }
    }
    # Открыть файл с заданным именем для чтения.
    with open(filename, "r", encoding="utf-8") as fin:
        loaded = json.load(fin)
    try:
        jsonschema.validate(loaded, schema)
    except jsonschema.exceptions.ValidationError as e:
        print(">>> Error:")
        print(e.message)  # Ошибка валидации будет выведена на экран
    return loaded


if __name__ == '__main__':
    cli()
```

Рисунок 8. Программа fly_click.py

```
(venv) (base) elenamiheeva@MacBook-Pro-Elena Data_Analysis_2.17 % python3 program/fly_click.py display data2.json
+------+------------------------------+----------------------+----------------------+
|  No  |        Пункт назначения      |     Номер рейса      |     Тип самолета     |
+------+------------------------------+----------------------+----------------------+
|    1 | tallin                       | 342                  |                plane |
|    2 | tallin                       | 349                  |                plane |
(venv) (base) elenamiheeva@MacBook-Pro-Elena Data_Analysis_2.17 % python3 program/fly_click.py find -f plane data1.json
Destination: moscow, Flight number: 341, Plane type: plane
Destination: tallin, Flight number: 342, Plane type: plane
(venv) (base) elenamiheeva@MacBook-Pro-Elena Data_Analysis_2.17 % python3 program/fly_click.py add -d  paris -n 657 -t plane data2.json
Flight added successfully.
(venv) (base) elenamiheeva@MacBook-Pro-Elena Data_Analysis_2.17 % python3 program/fly_click.py display data2.json
+------+------------------------------+----------------------+----------------------+
|  No  |        Пункт назначения      |     Номер рейса      |     Тип самолета     |
+------+------------------------------+----------------------+----------------------+
|    1 | tallin                       | 342                  |                plane |
|    2 | tallin                       | 349                  |                plane |
|    3 | paris                        | 657                  |                plane |
(venv) (base) elenamiheeva@MacBook-Pro-Elena Data_Analysis_2.17 %
```

Рисунок 9. Результат работы программы

Ответы на контрольные вопросы:

1. В чем отличие терминала и консоли?

Терминал (от лат. terminus - граница) - устройство или ПО, выступающее посредником между человеком и вычислительной системой. Обычно данный термин используется, когда точка доступа к системе вынесена в отдельное физическое устройство и предоставляет свой пользовательский интерфейс на основе внутреннего интерфейса (например, сетевых протоколов).

Консоль console - исторически реализация терминала с клавиатурой и текстовым дисплеем. В настоящее время это слово часто используется как синоним сеанса работы или окна оболочки командной строки. В том же смысле иногда применяется и слово "терминал".

2. Что такое консольное приложение?

Консольное приложение console application - вид ПО, разработанный с расчётом на работу внутри оболочки командной строки, т.е. опирающийся на текстовый ввод-вывод.

3. Какие существуют средства языка программирования Python для построения приложений командной строки?

Python 3 поддерживает несколько различных способов обработки аргументов командной строки.

Встроенный способ - использовать модуль sys. С точки зрения имен и использования, он имеет прямое отношение к библиотеке C (libc). Второй способ — это модуль getopt, который обрабатывает как короткие, так и длинные параметры, включая оценку значений параметров.

Кроме того, существуют два других общих метода. Это модуль argparse, производный от модуля optparse, доступного до Python 2.7. Другой метод - использование модуля docopt, доступного на GitHub.

Также есть модуль click.

4. Какие особенности построение CLI с использованием модуля sys?

Это базовый модуль, который с самого начала поставлялся с Python. Он использует подход, очень похожий на библиотеку C, с использованием агдс и argv для доступа к аргументам. Модуль sys реализует аргументы командной строки в простой структуре списка с именем sys.argv.

Каждый элемент списка представляет собой единственный аргумент.

Первый элемент в списке sys.argv[0] - это имя скрипта Python. Остальные элементы списка, от sys.argv[1] до sys.argv[n], являются аргументами командной строки с 2 по п. В качестве разделителя между аргументами используется пробел. Значения аргументов, содержащие пробел, должны быть заключены в кавычки, чтобы их правильно проанализировал sys. Эквивалент argc - это просто количество элементов в списке. Чтобы получить это значение, необходимо использовать оператор len .

5. Какие особенности построение CLI с использованием модуля getopt?

Модуль sys разбивает строку командной строки только на отдельные фасеты. Модуль getopt в Python идет немного дальше и расширяет разделение входной строки проверкой параметров. Основанный на функции C getopt, он позволяет использовать как короткие, так и длинные варианты, включая присвоение значений.

6. Какие особенности построение CLI с использованием модуля argparse?

argparse предлагает:

- анализ аргументов sys.argv;

- конвертирование строковых аргументов в объекты программы и работа с ними;

- форматирование и вывод информативных подсказок.

Библиотеки getopt и optparse уступают argparse по нескольким причинам:

- обладая всей полнотой действий с обычными параметрами командной строки, они не умеют обрабатывать позиционные аргументы (positional arguments). Позиционные аргументы — это аргументы, влияющие на работу программы, в зависимости от порядка, в котором они в эту программу передаются. Простейший пример - программа cp, имеющая минимум 2 таких аргумента («cp source destination»).

- argparse дает на выходе более качественные сообщения о подсказке при минимуме затрат;

- argparse дает возможность программисту устанавливать для себя, какие символы являются параметрами, а какие нет. В отличие от него, optparse считает опции с синтаксисом наподобие "-pf, -file, trgb, /f и т.п. «внутренне противоречивыми» и «не поддерживается optars'ом и никогда не будет»;

- argparse даст возможность использовать несколько значений переменных у одного аргумента командной строки (nargs);

- argparse поддерживает субкоманды (subcommands). Это когда основной парсер отсылает к другому (субпарсеру), в зависимости от аргументов на входе.