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»

Выполнил:

Пустяков Андрей Сергеевич

2 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и вычислительная техника (профиль) «Программное обеспечение средств вычислительной
техники и автоматизированных систем», очная форма обучения

_____

(подпись)

Руководитель практики:

Воронкин Р. А. кандидат технических наук, доцент, доцент кафедры инфокоммуникаций

_____

(подпись)

Ставрополь, 2024 г.

Тема: Разработка приложений с интерфейсом командной строки (CLI) в Python 3.

Цель: приобрести навыки в построении приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x.

Ход работы:

Создание общедоступного репозитория на «GitHub», клонирование репозитория, редактирование файла «.gitignore», организация репозитория согласно модели ветвления «git-flow» (рис. 1).

```
C:\Program Files\Git>cd C:\Users\Andrey\Desktop\Анализ_данных\3_лаба

C:\Users\Andrey\Desktop\Анализ_данных\3_лаба>git clone https://github.com/AndreyPust/Analys

Cloning into 'Analysis_data_laboratory_work_3'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.

C:\Users\Andrey\Desktop\Анализ_данных\3_лаба>cd C:\Users\Andrey\Desktop\Анализ_данных\3_лаб
rk_3

C:\Users\Andrey\Desktop\Анализ_данных\3_лаба\Analysis_data_laboratory_work_3>git flow init

Which branch should be used for bringing forth production releases?
   - main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]
```

Рисунок 1 – Организация модели ветвления «git-flow»

Проработка примеров лабораторной работы:

Пример 1.

Необходимо для примера лабораторной работы 2.16 разработать интерфейс командной строки. Имя JSON-файла для работы программы должно быть позиционным аргументом. Код программы для решения данной задачи (рис. 2).

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

import argparse
import json
import os.path
import sys
```

```python
from datetime import date


def add_worker(staff, name, post, year):
    """
    Добавить данные о работнике.
    """
    staff.append(
        {
            "name": name,
            "post": post,
            "year": year
        }
    )

    return staff


def display_workers(staff):
    """
    Отобразить список работников.
    """
    # Проверить, что список работников не пуст.
    if staff:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )

        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",
                "Ф.И.О.",
                "Должность",
                "Год"
            )
        )
        print(line)

        # Вывести данные о всех сотрудниках.
        for idx, worker in enumerate(staff, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx,
                    worker.get('name', ''),
                    worker.get('post', ''),
                    worker.get('year', 0)
                )
            )
            print(line)

    else:
        print("Список работников пуст.")


def select_workers(staff, period):
    """
    Выбрать работников с заданным стажем.
    """
    # Получить текущую дату.
```

```
    today = date.today()

    # Сформировать список работников.
    result = []
    for employee in staff:
        if today.year - employee.get('year', today.year) >= period:
            result.append(employee)

    # Возвратить список выбранных работников.
        return result


def save_workers(file_name, staff):
    """
    Сохранить всех работников в файл JSON.
    """
    # Открыть файл с заданным именем для записи.
    with open(file_name, "w", encoding="utf-8") as fout:
        # Выполнить сериализацию данных в формат JSON.
        # Для поддержки кирилицы установим ensure_ascii=False
        json.dump(staff, fout, ensure_ascii=False, indent=4)


def load_workers(file_name):
    """
    Загрузить всех работников из файла JSON.
    """
    # Открыть файл с заданным именем для чтения.
    with open(file_name, "r", encoding="utf-8") as fin:
        return json.load(fin)


def main(command_line=None):
    # 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("workers")
    parser.add_argument(
        "--version",
        action="version",
        version="%(prog)s 0.1.0"
    )

    subparsers = parser.add_subparsers(dest="command")

    # Создать субпарсер для добавления работника.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new worker"
    )
    add.add_argument(
        "-n",
        "--name",
        action="store",
        required=True,
        help="The worker's name"
    )
```

```
    add.add_argument(
        "-p",
        "--post",
        action="store",
        help="The worker's post"
    )
    add.add_argument(
        "-y",
        "--year",
        action="store",
        type=int,
        required=True,
        help="The year of hiring"
    )

    # Создать субпарсер для отображения всех работников.
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all workers"
    )

    # Создать субпарсер для выбора работников.
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the workers"
    )
    select.add_argument(
        "-P",
        "--period",
        action="store",
        type=int,
        required=True,
        help="The required period"
    )

    # Выполнить разбор аргументов командной строки.
    args = parser.parse_args(command_line)

    # Загрузить всех работников из файла, если файл существует.
    is_dirty = False
    if os.path.exists(args.filename):
        workers = load_workers(args.filename)
    else:
        workers = []

    # Добавить работника.
    if args.command == "add":
        workers = add_worker(
            workers,
            args.name,
            args.post,
            args.year
        )
        is_dirty = True

    # Отобразить всех работников.
    elif args.command == "display":
        display_workers(workers)

    # Выбрать требуемых рааботников.
    elif args.command == "select":
        selected = select_workers(workers, args.period)
```

```
        display_workers(selected)

    # Сохранить данные в файл, если список работников был изменен.
    if is_dirty:
        save_workers(args.filename, workers)


if __name__ == "__main__":
    main()
```

Рисунок 2 – Код программы примера №1

Для работы программы необходимо создать соответствующее виртуальное окружение (рис. 3).

```
(base) C:\Users\Andrey\Desktop\Анализ_данных\3_лаба\Analysis_data_laboratory_work_3>conda create --name lab_3 python=3.9

Retrieving notices: ...working... done
Collecting package metadata (current_repodata.json): done
```

Рисунок 3 – Создание виртуального окружения для работы программы примера

Результаты работы данной программы с различными исходными данными и для разных команд (рис. 4).

```
C:\Users\Andrey\Desktop\Анализ_данных\3_лаба\Analysis_data_laboratory_work_3\examples>python example_1.py add data.json --name="Сидоров Сидор" --post="Главный инженер" --year=2012

C:\Users\Andrey\Desktop\Анализ_данных\3_лаба\Analysis_data_laboratory_work_3\examples>python example_1.py add data.json --name="Иванов Иван" --post="Директор" --year=2007

C:\Users\Andrey\Desktop\Анализ_данных\3_лаба\Analysis_data_laboratory_work_3\examples>python example_1.py add data.json --name="Петров Петр" --post="Бухгалтер" --year=2010

C:\Users\Andrey\Desktop\Анализ_данных\3_лаба\Analysis_data_laboratory_work_3\examples>python example_1.py display data.json
+-----+--------------------------+--------------------+----------+
|  №  |          Ф.И.О.          |      Должность     |   Год    |
+-----+--------------------------+--------------------+----------+
|   1 | Сидоров Сидор            | Главный инженер    |     2012 |
+-----+--------------------------+--------------------+----------+
|   2 | Иванов Иван              | Директор           |     2007 |
+-----+--------------------------+--------------------+----------+
|   3 | Петров Петр              | Бухгалтер          |     2010 |
+-----+--------------------------+--------------------+----------+

C:\Users\Andrey\Desktop\Анализ_данных\3_лаба\Analysis_data_laboratory_work_3\examples>python example_1.py select data.json --period=12
+-----+--------------------------+--------------------+----------+
|  №  |          Ф.И.О.          |      Должность     |   Год    |
+-----+--------------------------+--------------------+----------+
|   1 | Сидоров Сидор            | Главный инженер    |     2012 |
+-----+--------------------------+--------------------+----------+
```

Рисунок 4 – Результаты работы программы примера №1

Содержимое файла «data.json» (рис. 5).

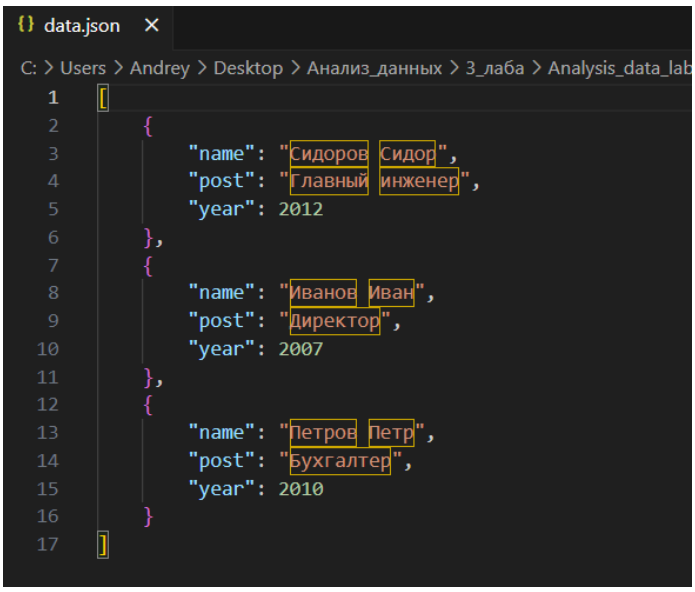

Рисунок 5 – Файл «data.json»

Выполнение индивидуальных заданий:

Необходимо для своего варианта лабораторной работы 2.16 дополнительно реализовать интерфейс командной строки (CLI).

Необходимо использовать словарь, содержащий следующие ключи: название пункта назначения; номер поезда; время отправления. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в список, состоящий из словарей заданной структуры; записи должны быть упорядочены по времени отправления поезда; вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры; если таких поездов нет, выдать на дисплей соответствующее сообщение (Вариант 26 (7), работа 2.8). Код программы индивидуального задания (рис. 6).

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

import argparse
import json
import os.path
import sys


def add_train(trains, departure_point, number_train, time_departure,
destination):
    """
    Добавить данные о поезде.
    """
    trains.append(
        {
```

```
            "departure_point": departure_point,
            "number_train": number_train,
            "time_departure": time_departure,
            "destination": destination
        }
    )

    return trains


def display_trains(trains):
    """
    Отобразить список поездов со станциями.
    """
    # Проверить, что список поездов не пуст.
    if trains:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+--{}--+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 13,
            '-' * 18,
            '-' * 14
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^13} | {:^18} | {:^14} |'.format(
                "№",
                "Пункт отправления",
                "Номер поезда",
                "Время отправления",
                "Пункт назначения"
            )
        )
        print(line)

        # Вывести данные о всех поездах со станциями.
        for idx, train in enumerate(trains, 1):
            print(
                '| {:>4} | {:<30} | {:<13} | {:>18} | {:^16} |'.format(
                    idx, train.get('departure_point', ''),
                    train.get('number_train', ''),
                    train.get('time_departure', ''),
                    train.get('destination', '')
                )
            )
            print(line)

    else:
        print("Список поездов пуст.")


def select_trains(trains, point_user):
    """
    Выбрать поезда по пункту назначения.
    """
    # Сформировать список поездов.
    result = []
    for train in trains:
        if point_user == str.lower(train['destination']):
            result.append(train)

    # Возвратить список выбранных поездов, направляющихся в пункт.
    return result
```

```python
def save_trains(file_name, trains):
    """
    Сохранить все поезда со станциями в файл JSON.
    """
    # Открыть файл с заданным именем для записи.
    with open(file_name, "w", encoding="utf-8") as fout:
        # Выполнить сериализацию данных в формат JSON.
        # Для поддержки кирилицы установим ensure_ascii=False
        json.dump(trains, fout, ensure_ascii=False, indent=4)


def load_trains(file_name):
    """
    Загрузить все поезда со станциями из файла JSON.
    """
    # Открыть файл с заданным именем для чтения.
    with open(file_name, "r", encoding="utf-8") as fin:
        return json.load(fin)


def main(command_line=None):
    # 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("trains")
    parser.add_argument(
        "--version",
        action="version",
        version="%(prog)s 0.1.0"
    )

    subparsers = parser.add_subparsers(dest="command")

    # Создать субпарсер для добавления поезда.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new train"
    )
    add.add_argument(
        "-dep",
        "--departure_point",
        action="store",
        required=True,
        help="The train's departure point"
    )
    add.add_argument(
        "-n",
        "--number_train",
        action="store",
        required=True,
        help="The train's number"
    )
    add.add_argument(
        "-t",
        "--time_departure",
```

```
        action="store",
        required=True,
        help="The time departure of train"
    )
    add.add_argument(
        "-des",
        "--destination",
        action="store",
        required=True,
        help="The destination of train"
    )

    # Создать субпарсер для отображения всех поездов.
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all trains"
    )

    # Создать субпарсер для выбора поездов по пунктам назначения.
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the trains"
    )
    select.add_argument(
        "-P",
        "--point_user",
        action="store",
        required=True,
        help="The required point"
    )

    # Выполнить разбор аргументов командной строки.
    args = parser.parse_args(command_line)

    # Загрузить все поезда со станциями из файла, если файл существует.
    is_dirty = False
    if os.path.exists(args.filename):
        trains = load_trains(args.filename)
    else:
        trains = []

    # Добавить поезд со станциями.
    if args.command == "add":
        trains = add_train(
            trains,
            args.departure_point,
            args.number_train,
            args.time_departure,
            args.destination
        )
        is_dirty = True

    # Отобразить все поезда со станциями.
    elif args.command == "display":
        display_trains(trains)

    # Выбрать требуемые поезда.
    elif args.command == "select":
        selected = select_trains(trains, args.point_user)
        display_trains(selected)

    # Сохранить данные в файл, если список поездов был изменен.
```

```
    if is_dirty:
        save_trains(args.filename, trains)


if __name__ == "__main__":
    main()
```

Рисунок 6 – Код программы индивидуального задания

Результаты работы программы с готовым файлом, содержащим список словарей поездов со станциями (рис. 7).

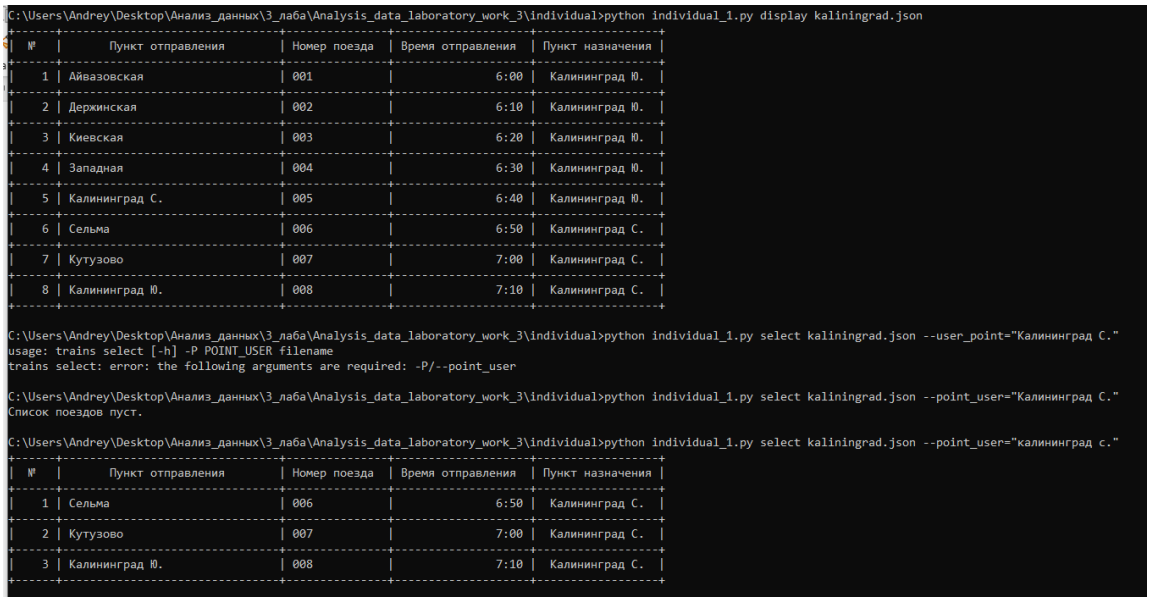

Рисунок 7 – Результаты работы программы индивидуального задания

Содержимое готового файла «kaliningrag.json» (рис. 8).

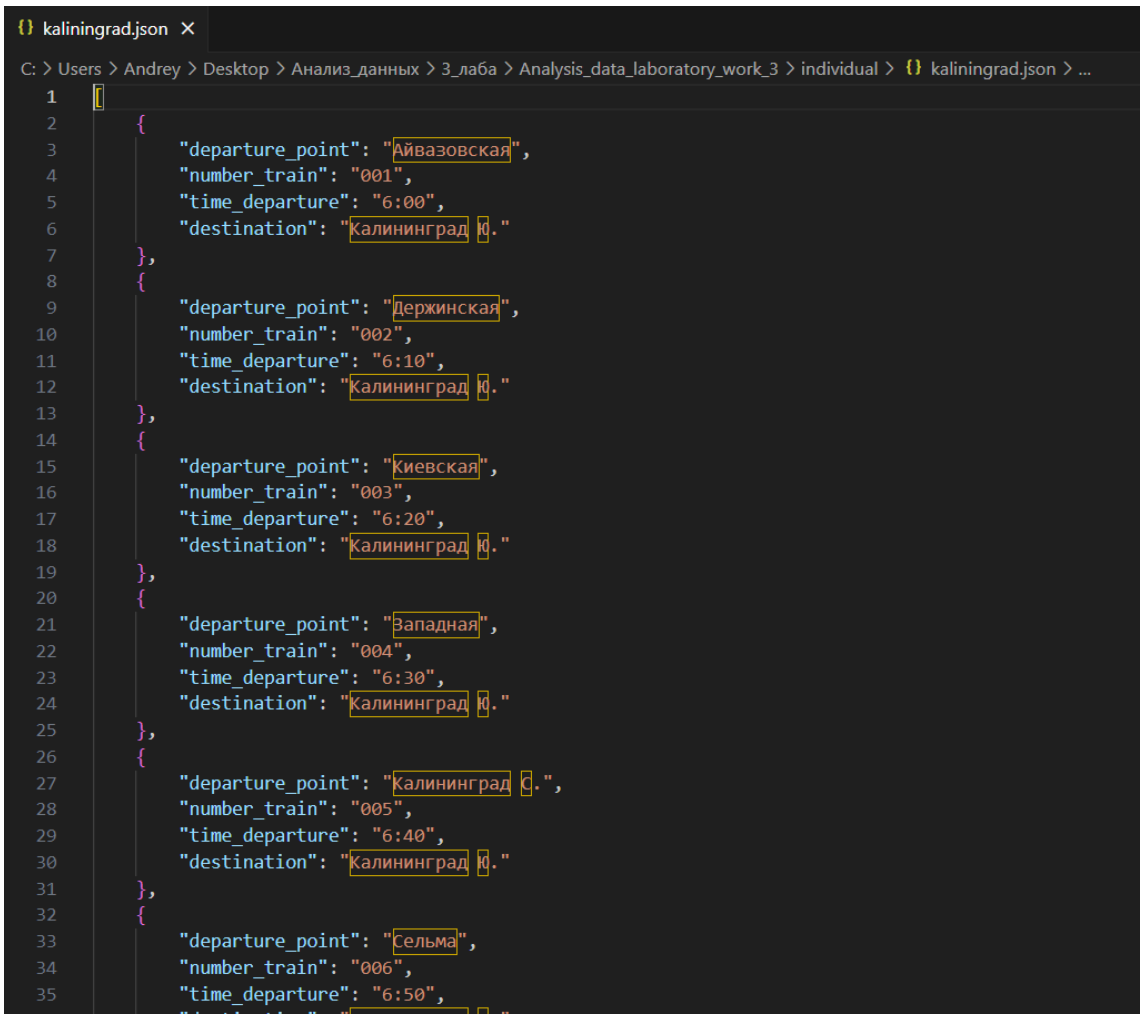

Рисунок 8 – Файл «kaliningrad.json»

Выполнение задания повышенной сложности:

Необходимо изучить работу с пакетом «click» и для своего варианта лабораторной работы 2.16 реализовать интерфейс командной строки с использованием этого пакета.

Код программы задания повышенной сложности с реализованным интерфейсом командной строки (CLI) с использованием пакета «click» (рис. 9).

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

import bisect
import json
import os
import sys
import click


def add_train(trains, departure_point, number_train, time_departure,
destination):
    """
```

```
    Добавить данные о поездах со станциями.
    """
    is_dirty = False
    train = {
        "departure_point": departure_point,
        "number_train": number_train,
        "time_departure": time_departure,
        "destination": destination
    }
    if train not in trains:
        bisect.insort(
            trains,
            train,
            key=lambda item: item.get("time_departure"),
        )
        is_dirty = True
    else:
        click.echo("Данный поезд уже добавлен.")
    return trains, is_dirty


def display_trains(trains):
    """
    Отобразить список поездов со станциями.
    """
    if trains:
        line = '+-{}-+-{}-+-{}-+-{}-+--{}--+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 13,
            '-' * 18,
            '-' * 14
        )
        click.echo(line)
        click.echo(
            '| {:^4} | {:^30} | {:^13} | {:^18} | {:^14} |'.format(
                "№",
                "Пункт отправления",
                "Номер поезда",
                "Время отправления",
                "Пункт назначения"
            )
        )
        click.echo(line)
        for idx, train in trains:
            click.echo(
                '| {:>4} | {:<30} | {:<13} | {:>18} | {:^16} |'.format(
                    idx, train.get('departure_point', ''),
                    train.get('number_train', ''),
                    train.get('time_departure', ''),
                    train.get('destination', '')
                )
            )
        click.echo(line)
    else:
        click.echo("Список поездов пуст.")


def select_trains(trains, point_user):
    """
    Выбрать поезда по пункту назначения.
    """
    selected = []
    for train in trains:
```

```
        if point_user == str.lower(train['destination']):
            selected.append(train)

    # Возвратить список выбранных поездов, направляющихся в пункт.
    return selected


def save_trains(file_name, trains):
    """
    Сохранить все поезда в файл JSON.
    """
    # Открыть файл с заданным именем для записи.
    with open(file_name, "w") as file_out:
        # Записать данные из словаря в формат JSON и сохранить их
        # в открытый файл.
        json.dump(trains, file_out, ensure_ascii=False, indent=4)


def load_trains(file_name):
    """
    Загрузить все поезда из файла JSON.
    """
    # Открыть файл с заданным именем и прочитать его содержимое.
    with open(file_name, "r", encoding="utf-8") as fin:
        return json.load(fin)


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


@command.command()
@click.argument("filename")
@click.option("-dep", "--departure_point", required=True, help="The departure
point train")
@click.option("-n", "--number_train", required=True, help="The number train")
@click.option("-t", "--time_departure", required=True, help="The time
departure of train")
@click.option("-des", "--destination", required=True, help="The destination
of train")
def add(filename, departure_point, number_train, time_departure,
destination):
    """
    Add a new train.
    """

    filename = os.path.join("data", filename)
    trains = load_trains(filename)

    routes, is_dirty = add_train(trains, departure_point.lower(),
number_train.lower(), time_departure.lower(), destination.lower())
    if is_dirty:
        save_trains(filename, trains)


@command.command()
@click.argument("filename")
@click.option("-p", "--point_user", required=True, help="Destination train")
def select(filename, point_user):
    """
    Select the trains
    """
    filename = os.path.join("data", filename)
```

```
    point_user = point_user.lower()
    trains = load_trains(filename)
    selected_trains = select_trains(trains, point_user)
    display_trains(selected_trains)


@command.command()
@click.argument("filename")
def display(filename):
    """
    Display all trains
    """
    filename = os.path.join("data", filename)
    trains = load_trains(filename)
    display_trains(trains)


if __name__ == "__main__":
    command()
```

Рисунок 9 – Код программы усложненного задания

Результаты работы усложненного задания (рис. 10).

```
C:\Users\Andrey\Desktop\Анализ_данных\3_лаба\Analysis_data_laboratory_work_3\individual>python individual_1.py display kaliningrad.json
+------+-----------------------------+--------------+-------------------+-----------------+
|  №   |      Пункт отправления      | Номер поезда | Время отправления | Пункт назначения |
+------+-----------------------------+--------------+-------------------+-----------------+
|    1 | Айвазовская                 | 001          |             6:00 | Калининград Ю.  |
+------+-----------------------------+--------------+-------------------+-----------------+
|    2 | Держинская                  | 002          |             6:10 | Калининград Ю.  |
+------+-----------------------------+--------------+-------------------+-----------------+
|    3 | Киевская                    | 003          |             6:20 | Калининград Ю.  |
+------+-----------------------------+--------------+-------------------+-----------------+
|    4 | Западная                    | 004          |             6:30 | Калининград Ю.  |
+------+-----------------------------+--------------+-------------------+-----------------+
|    5 | Калининград С.              | 005          |             6:40 | Калининград Ю.  |
+------+-----------------------------+--------------+-------------------+-----------------+
|    6 | Сельма                      | 006          |             6:50 | Калининград С.  |
+------+-----------------------------+--------------+-------------------+-----------------+
|    7 | Кутузово                    | 007          |             7:00 | Калининград С.  |
+------+-----------------------------+--------------+-------------------+-----------------+
|    8 | Калининград Ю.              | 008          |             7:10 | Калининград С.  |
+------+-----------------------------+--------------+-------------------+-----------------+

C:\Users\Andrey\Desktop\Анализ_данных\3_лаба\Analysis_data_laboratory_work_3\individual>python individual_1.py select kaliningrad.json --user_point="Калининград С."
usage: trains select [-h] -P POINT_USER filename
trains select: error: the following arguments are required: -P/--point_user

C:\Users\Andrey\Desktop\Анализ_данных\3_лаба\Analysis_data_laboratory_work_3\individual>python individual_1.py select kaliningrad.json --point_user="Калининград С."
Список поездов пуст.

C:\Users\Andrey\Desktop\Анализ_данных\3_лаба\Analysis_data_laboratory_work_3\individual>python individual_1.py select kaliningrad.json --point_user="калининград с."
+------+-----------------------------+--------------+-------------------+-----------------+
|  №   |      Пункт отправления      | Номер поезда | Время отправления | Пункт назначения |
+------+-----------------------------+--------------+-------------------+-----------------+
|    1 | Сельма                      | 006          |             6:50 | Калининград С.  |
+------+-----------------------------+--------------+-------------------+-----------------+
|    2 | Кутузово                    | 007          |             7:00 | Калининград С.  |
+------+-----------------------------+--------------+-------------------+-----------------+
|    3 | Калининград Ю.              | 008          |             7:10 | Калининград С.  |
+------+-----------------------------+--------------+-------------------+-----------------+
```

Рисунок 10 – Результаты работы усложненного задания

Ответы на контрольные вопросы:

1. В чем отличие терминала и консоли?

Терминал – это программное обеспечение или устройство, позволяющее пользователю взаимодействовать с операционной системой. Он предоставляет текстовый интерфейс для ввода команд и получения вывода. Обычно, терминал используется для запуска команд и управления системой. Консоль –

это окно, где пользователь может вводить команды, запускать приложения и видеть вывод этих программ. Это графическое представление терминала.

2. Что такое консольное приложение?

Консольное приложение – это программа, предназначенная для выполнения в командной строке (консоли). Такие приложения обрабатывают ввод пользователя и выводят результат на консоль без графического интерфейса.

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

В Python существует несколько встроенных модулей для создания CLIприложений: sys, argparse, getopt и другие.

4. Какие особенности построение CLI с использованием модуля sys?

Модуль sys предоставляет доступ к некоторым переменным и функциям, связанным с интерпретатором Python. Он обеспечивает доступ к аргументам командной строки через sys.argv, что позволяет обрабатывать аргументы при запуске скрипта.

5. Какие особенности построение CLI с использованием модуля getopt?

Модуль getopt предоставляет функции для парсинга аргументов командной строки. Он позволяет более гибко управлять аргументами командной строки и их опциями.

6. Какие особенности построение CLI с использованием модуля argparse?

Модуль argparse является более мощным и гибким инструментом для создания интерфейса командной строки в Python. Он позволяет определять аргументы, их типы, флаги и даже создавать справочную информацию для пользователей. argparse автоматически генерирует справку о том, как использовать ваше CLI-приложение.

Вывод: приобрел навыки построения приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x.