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

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЁТ

по лабораторной работе №2.17

Дисциплина: «Анализ данных»
Тема: «Разработка приложений с интерфейсом командной строки (CLI) в Python3»

Выполнил:
Епифанов Алексей Александрович
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 Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

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

Порядок выполнения работы:

1. Создал новый репозиторий, клонировал его, в нем создал ветку developer и перешел на нее.

2. Проработал пример лабораторной работы:

```
❯ program/prim.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
❯ program/prim.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
❯ program/prim.py add -n Михаил -y 2012 data.json
❯ program/prim.py add -n Алексей -p Студент -y 2022 data.json
❯ program/prim.py display data.json
+------+----------------------------+----------------------+----------+
|  No  |            Ф.И.О.          |      Должность       |   Год    |
+------+----------------------------+----------------------+----------+
|    1 | Михаил                     | None                 |     2012 |
+------+----------------------------+----------------------+----------+
|    2 | Алексей                    | Студент              |     2022 |
+------+----------------------------+----------------------+----------+
❯ program/prim.py select -h
usage: workers select [-h] -P PERIOD filename

positional arguments:
  filename              The data file name

options:
  -h, --help            show this help message and exit
  -P PERIOD, --period PERIOD
                        The required period
❯ program/prim.py select -P 5
usage: workers select [-h] -P PERIOD filename
workers select: error: the following arguments are required: filename
❯ program/prim.py select -P 5 data.json
+------+----------------------------+----------------------+----------+
|  No  |            Ф.И.О.          |      Должность       |   Год    |
+------+----------------------------+----------------------+----------+
|    1 | Михаил                     | None                 |     2012 |
```

Рисунок 1. Ввод, вывод и выбор работников в консоли

3. Выполнил индивидуальное задание: для своего варианта лабораторной работы 2.16 необходимо дополнительно реализовать интерфейс командной строки (CLI).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import argparse
import bisect
import json
import os
from jsonschema import ValidationError, validate
def add_route(routes, start, end, count):
    """
    Добавить данные о маршруте.
    """
    route = {
        "начальный пункт": start.lower(),
        "конечный пункт": end.lower(),
        "номер маршрута": count,
    }
    if route not in routes:
        bisect.insort(
            routes,
            route,
            key=lambda item: item.get("номер маршрута"),
        )
    else:
        print("Данный маршрут уже добавлен.")
    return routes
def display_routes(routes):
    """
    Отобразить список маршрутов.
    """
    if routes:
        line = "+-{}-+-{}-+-{}-+".format("-" * 30, "-" * 20, "-" * 8)
        print(line)
        print("| {:^30} | {:^20} | {:^8} |".format("Начало", "Конец", "Номер"))
        print(line)
        for route in routes:
            print(
                "| {:<30} | {:<20} | {:>8} |".format(
                    route.get("начальный пункт", ""),
                    route.get("конечный пункт", ""),
```

```
                route.get("номер маршрута", ""),
            )
        )
        print(line)
    else:
        print("Список маршрутов пуст.")
def select_routes(routes, name_point):
    """
    Выбрать маршруты с заданным пунктом отправления или прибытия.
    """
    selected = []
    for route in routes:
        if (
            route["начальный пункт"] == name_point
            or route["конечный пункт"] == name_point
        ):
            selected.append(route)

    return selected
def save_routes(file_name, routes):
    """
    Сохранить все маршруты 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(routes, file_out, ensure_ascii=False, indent=4)
def load_routes(file_name):
    """
    Загрузить все маршруты из файла JSON.
    """
    schema = {
        "type": "array",
        "items": {
            "type": "object",
            "properties": {
                "начальный пункт": {"type": "string"},
                "конечный пункт": {"type": "string"},
                "номер маршрута": {"type": "integer"},
            },
            "required": [
                "начальный пункт",
                "конечный пункт",
                "номер маршрута",
```

```
            ],
        },
    }
    # Открыть файл с заданным именем и прочитать его содержимое.
    with open(file_name, "r") as file_in:
        data = json.load(file_in)  # Прочитать данные из файла
    try:
        # Валидация
        validate(instance=data, schema=schema)
        print("JSON валиден по схеме.")
        return data
    except ValidationError as e:
        print(f"Ошибка валидации: {e.message}")
        return []
def main(command_line=None):
    """
    Главная функция программы.
    """
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "filename", action="store", help="The data file name"
    )
    parser = argparse.ArgumentParser("routes")
    parser.add_argument(
        "--version", action="version", version="%(prog)s 0.1.0"
    )
    subparsers = parser.add_subparsers(dest="command")
    add = subparsers.add_parser(
        "add", parents=[file_parser], help="Add a new route"
    )
    add.add_argument(
        "-s", "--start", action="store", required=True, help="The route start"
    )
    add.add_argument(
        "-e", "--end", action="store", required=True, help="The route endpoint"
    )
    add.add_argument(
        "-n",
        "--number",
        action="store",
        type=int,
        required=True,
        help="The number of route",
    )
```

```
    _ = subparsers.add_parser(
        "list", parents=[file_parser], help="Display all routes"
    )
    select = subparsers.add_parser(
        "select", parents=[file_parser], help="Select the routes"
    )
    select.add_argument(
        "-p",
        "--point",
        action="store",
        required=True,
        help="Routes starting or ending at this point",
    )
    args = parser.parse_args(command_line)
    # Загрузить всех работников из файла, если файл существует.
    is_dirty = False
    if os.path.exists(args.filename):
        routes = load_routes(args.filename)
    else:
        routes = []
    match args.command.lower():
        case "add":
            routes = add_route(routes, args.start, args.end, args.number)
            is_dirty = True

        case "list":
            display_routes(routes)
        case "select":
            name_point = args.point.lower()
            selected = select_routes(routes, name_point)
            display_routes(selected)
    if is_dirty:
        save_routes(args.filename, routes)
if __name__ == "__main__":
    main()
```

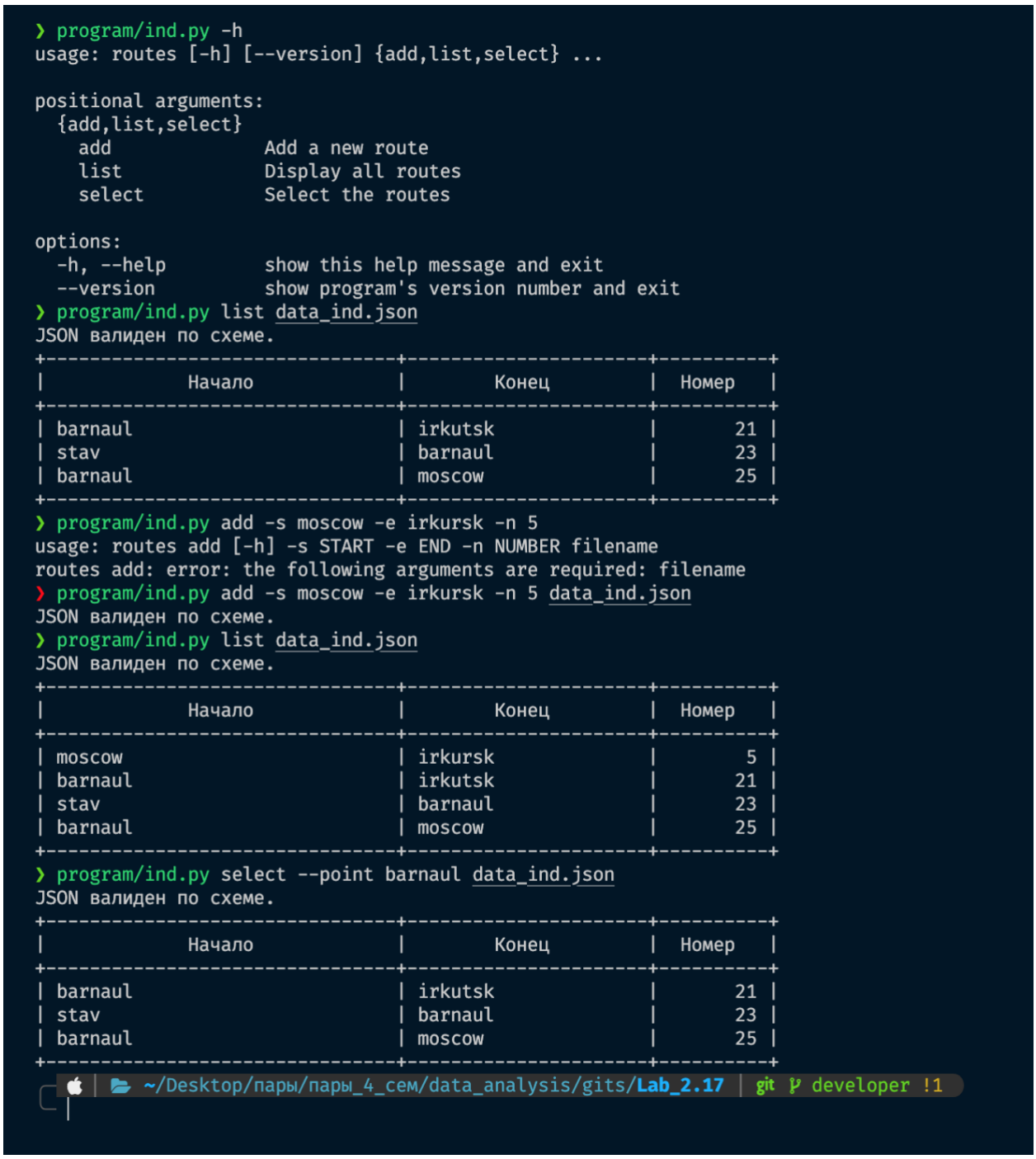

Рисунок 3. Ввод, вывод и выборка маршрутов

4. Выполнил задание повышенной сложности: Самостоятельно изучите работу с пакетом click для построения интерфейса командной строки (CLI). Для своего варианта лабораторной работы 2.16 необходимо реализовать интерфейс командной строки с использованием пакета click .

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

```
import os
import click
from jsonschema import ValidationError, validate
def add_route(routes, start, end, number):
    """
    Добавить данные о маршруте.
    """
    is_dirty = False
    route = {
        "начальный пункт": start.lower(),
        "конечный пункт": end.lower(),
        "номер маршрута": number,
    }
    if route not in routes:
        bisect.insort(
            routes,
            route,
            key=lambda item: item.get("номер маршрута"),
        )
        is_dirty = True
    else:
        print("Данный маршрут уже добавлен.")
    return routes, is_dirty
def display_routes(routes):
    """
    Отобразить список маршрутов.
    """
    if routes:
        line = "+-{}-+-{}-+-{}-+".format("-" * 30, "-" * 20, "-" * 8)
        print(line)
        print("| {:^30} | {:^20} | {:^8} |".format("Начало", "Конец", "Номер"))
        print(line)
        for route in routes:
            print(
                "| {:<30} | {:<20} | {:>8} |".format(
                    route.get("начальный пункт", ""),
                    route.get("конечный пункт", ""),
                    route.get("номер маршрута", ""),
                )
            )
        print(line)
    else:
        print("Список маршрутов пуст.")
def select_routes(routes, name_point):
    """
```

```
    Выбрать маршруты с заданным пунктом отправления или прибытия.
    """
    selected = []
    for route in routes:
        if (
            route["начальный пункт"] == name_point
            or route["конечный пункт"] == name_point
        ):
            selected.append(route)

    return selected
def save_routes(file_name, routes):
    """
    Сохранить все маршруты 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(routes, file_out, ensure_ascii=False, indent=4)
def load_routes(file_name):
    """
    Загрузить все маршруты из файла JSON.
    """
    schema = {
        "type": "array",
        "items": {
            "type": "object",
            "properties": {
                "начальный пункт": {"type": "string"},
                "конечный пункт": {"type": "string"},
                "номер маршрута": {"type": "integer"},
            },
            "required": [
                "начальный пункт",
                "конечный пункт",
                "номер маршрута",
            ],
        },
    }
    if not os.path.exists(file_name):
        return []
    # Открыть файл с заданным именем и прочитать его содержимое.
    with open(file_name, "r") as file_in:
        data = json.load(file_in)  # Прочитать данные из файла
```

```
    try:
        # Валидация
        validate(instance=data, schema=schema)
        print("JSON валиден по схеме.")
        return data
    except ValidationError as e:
        print(f"Ошибка валидации: {e.message}")
        return []
@click.group()
def command():
    pass
@command.command()
@click.argument("filename")
@click.option("-s", "--start", required=True, help="The route start")
@click.option("-e", "--end", required=True, help="The route endpoint")
@click.option(
    "-n", "--number", required=True, type=int, help="The number of route"
)
def add(filename, start, end, number):
    """
    Add a new route.
    """
    routes = load_routes(filename)

    routes, is_dirty = add_route(routes, start.lower(), end.lower(), number)
    if is_dirty:
        save_routes(filename, routes)
@command.command()
@click.argument("filename")
@click.option(
    "-p",
    "--point",
    required=True,
    help="Routes starting or ending at this point",
)
def select(filename, point):
    """
    Select the routes
    """
    point = point.lower()
    routes = load_routes(filename)
    selected_routes = select_routes(routes, point)
    display_routes(selected_routes)
@command.command()
```

```
@click.argument("filename")
def display(filename):
    """
    Display all routes
    """
    routes = load_routes(filename)
    display_routes(routes)
if __name__ == "__main__":
    command()
```

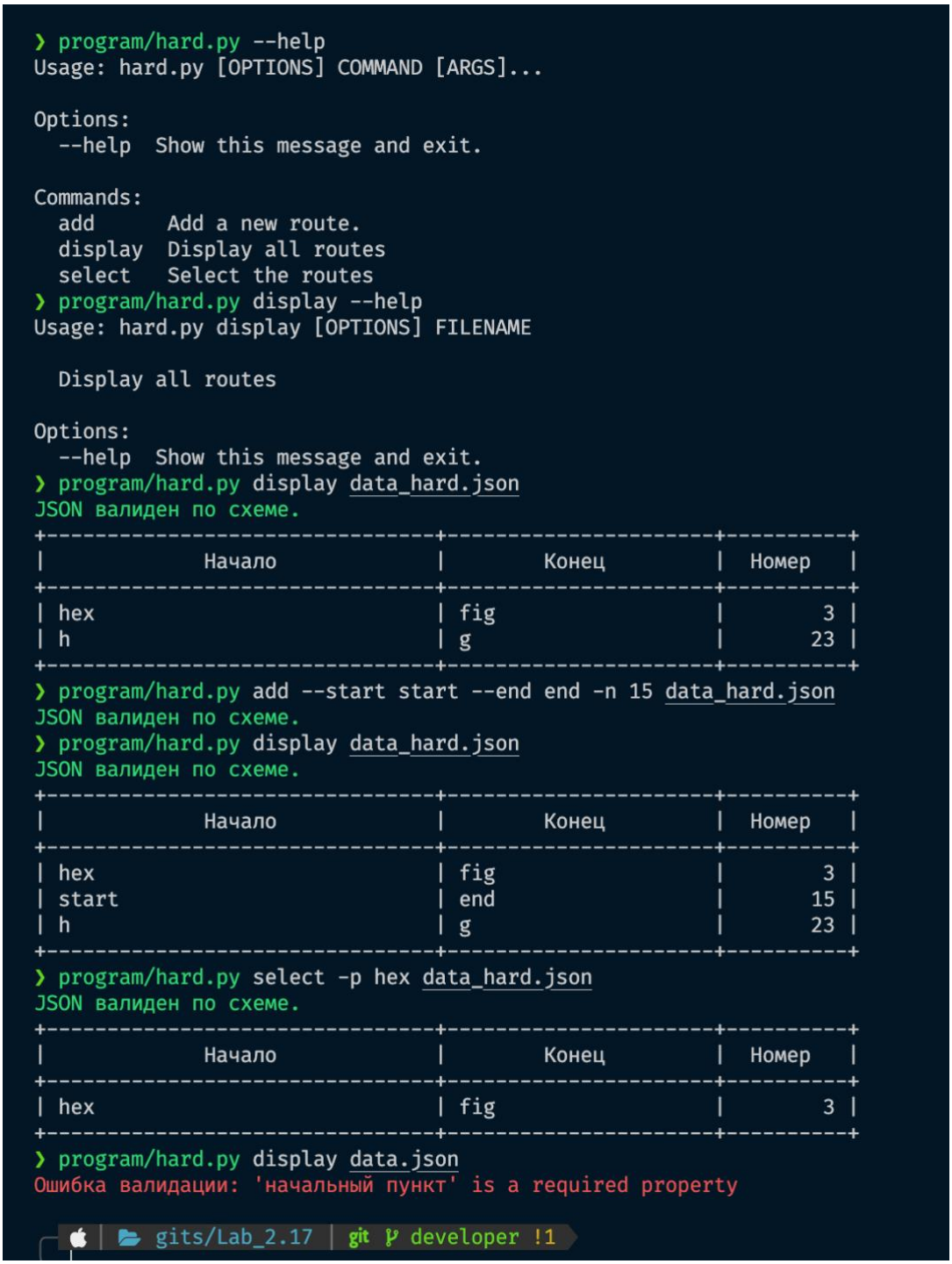

Рисунок 4. Ввод, вывод и выборка маршрутов

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 argc 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 n. 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, +rgb, /f и т.п. «внутренне противоречивыми» и «не поддерживается optpars'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.

Вывод: в результате выполнения работы были приобретены знания о построении приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x.