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:
Кожуховский Виктор Андреевич
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.

Тема: Разработка приложений с интерфейсом командной строки (CLI) в Python3

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

Порядок выполнения работы:

1. Изучил теоретический материал работы.

2. Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT и язык программирования Python.

3. Выполнил клонирование созданного репозитория.

4. Дополнил файл .gitignore необходимыми правилами для работы с IDE PyCharm.

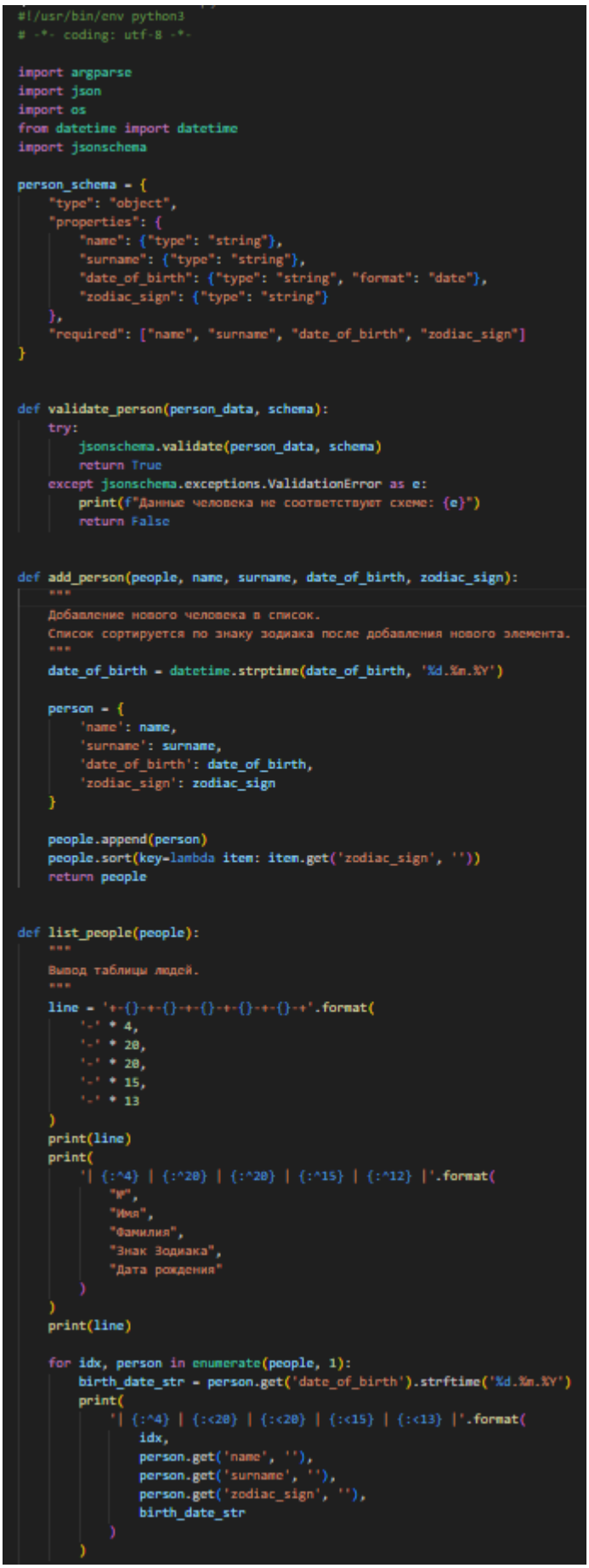
5. Организовал свой репозиторий в соответствие с моделью ветвления git-flow.

6. Создал проект в папке репозитория.

7. Проработал пример лабораторной работы.

8. Выполнил индивидуальное задание и задание повышенной сложности.

Для своего варианта лабораторной работы 2.16 необходимо дополнительно реализовать интерфейс командной строки (CLI).



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

import argparse
import json
import os
from datetime import datetime
import jsonschema

person_schema = {
    "type": "object",
    "properties": {
        "name": {"type": "string"},
        "surname": {"type": "string"},
        "date_of_birth": {"type": "string", "format": "date"},
        "zodiac_sign": {"type": "string"}
    },
    "required": ["name", "surname", "date_of_birth", "zodiac_sign"]
}


def validate_person(person_data, schema):
    try:
        jsonschema.validate(person_data, schema)
        return True
    except jsonschema.exceptions.ValidationError as e:
        print(f"Данные человека не соответствуют схеме: {e}")
        return False


def add_person(people, name, surname, date_of_birth, zodiac_sign):
    """
    Добавление нового человека в список.
    Список сортируется по знаку зодиака после добавления нового элемента.
    """
    date_of_birth = datetime.strptime(date_of_birth, '%d.%m.%Y')

    person = {
        'name': name,
        'surname': surname,
        'date_of_birth': date_of_birth,
        'zodiac_sign': zodiac_sign
    }

    people.append(person)
    people.sort(key=lambda item: item.get('zodiac_sign', ''))
    return people


def list_people(people):
    """
    Вывод таблицы людей.
    """
    line = '+-{}-+-{}-+-{}-+-{}-+-{}-+'.format(
        '-' * 4,
        '-' * 20,
        '-' * 20,
        '-' * 15,
        '-' * 13
    )
    print(line)
    print(
        '| {:^4} | {:^20} | {:^20} | {:^15} | {:^12} |'.format(
            "№",
            "Имя",
            "Фамилия",
            "Знак Зодиака",
            "Дата рождения"
        )
    )
    print(line)

    for idx, person in enumerate(people, 1):
        birth_date_str = person.get('date_of_birth').strftime('%d.%m.%Y')
        print(
            '| {:^4} | {:<20} | {:<20} | {:<15} | {:<13} |'.format(
                idx,
                person.get('name', ''),
                person.get('surname', ''),
                person.get('zodiac_sign', ''),
                birth_date_str
            )
        )
```

Рисунок 1. Код решения индивидуального задания 1

```
def list_people(people):

    print(line)


def select_people(people, month):
    """
    Вывести список людей, родившихся в заданном месяце.
    """
    count = 0
    for person in people:
        if person.get('date_of_birth').month == month:
            count += 1
            print('{:>4}: {} {}'.format(count, person.get(
                'name', ''), person.get('surname', '')))

    if count == 0:
        print("Люди, родившиеся в указанном месяце, не найдены.")


def save_people(file_name, staff):
    """
    Сохранить всех работников в файл JSON.
    """
    staff_formatted = [{**person, 'date_of_birth': person.get(
        'date_of_birth').strftime('%d.%m.%Y')} for person in staff]
    # Открыть файл с заданным именем для записи.
    with open(file_name, "w", encoding="utf-8") as fout:
        # Выполнить сериализацию данных в формат JSON.
        json.dump(staff_formatted, fout, ensure_ascii=False, indent=4)


def load_people(file_name):
    """
    Загрузить всех людей из файла JSON.
    """
    # Открыть файл с заданным именем для чтения.
    with open(file_name, "r", encoding="utf-8") as fin:
        staff_loaded = json.load(fin)
        result_people = []
        cnt = 0
        for person in staff_loaded:
            cnt += 1
            if validate_person(person, person_schema):
                try:
                    person['date_of_birth'] = datetime.strptime(
                        person['date_of_birth'], '%d.%m.%Y')
                    result_people.append(person)
                except:
                    print(
                        f"Ошибка при разборе даты в записи, пропуск записи {cnt}.")
            else:
                print("Неверные данные человека, пропуск записи.")
        return result_people


def main():
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "filename",
        action="store",
        help="The data file name"
    )
    # Создание основного парсера.
    parser = argparse.ArgumentParser(description="Управление списком людей")
    subparsers = parser.add_subparsers(dest="command")

    # Создание парсера для добавления человека.
    parser_add = subparsers.add_parser(
        'add', jsonfile=[file_parser], help="Добавить человека")
    parser_add.add_argument("-n", "--name", help="Имя человека")
    parser_add.add_argument("-s", "--surname", help="Фамилия человека")
    parser_add.add_argument(
        "-d", "--date_of_birth", help="Дата рождения (формат ДД.ММ.ГГГГ)")
    parser_add.add_argument("-z", "--zodiac_sign", help="Знак зодиака")

    # Создание парсера для вывода списка людей.
    _ = subparsers.add_parser(
        'list', jsonfile=[file_parser], help="Вывести список людей")

    # Создание парсера для выбора человека по месяцу рождения.
    parser_select = subparsers.add_parser(
        'select', jsonfile=[file_parser], help="Выбрать людей по месяцу рождения")
    parser_select.add_argument(
```

Рисунок 2. Код решения индивидуального задания 1

```python
def main():
        "-m", "--month", type=int, help="Месяц рождения")

    # Разбираем аргументы командной строки.
    args = parser.parse_args()

    is_dirty = False

    filename = os.path.join("data", args.filename)

    if os.path.exists(filename):
        people = load_people(filename)
    else:
        people = []

    # Определяем, какую команду нужно выполнить.
    if args.command == 'add':
        people = add_person(people, args.name, args.surname,
                            args.date_of_birth, args.zodiac_sign)
        is_dirty = True

    elif args.command == 'list':
        list_people(people)

    elif args.command == 'select':
        select_people(people, args.month)

    if is_dirty:
        save_people(filename, people)


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

Рисунок 3. Код решения индивидуального задания 1

```
PS C:\Users\viktor\Desktop\скфу\DataAnalysis\Data3> python 2.py -h
usage: 2.py [-h] {add,list,select} ...

Управление списком людей

positional arguments:
  {add,list,select}
    add             Добавить человека
    list            Вывести список людей
    select          Выбрать людей по месяцу рождения

options:
  -h, --help        show this help message and exit
PS C:\Users\viktor\Desktop\скфу\DataAnalysis\Data3> python 2.py list bruh.json
+------+----------------------+----------------------+----------------+---------------+
|  №   |         Имя          |       Фамилия        |  Знак Зодиака  | Дата рождения |
+------+----------------------+----------------------+----------------+---------------+
|  1   | asdhasdf             | sdfsdfg              | asdasf         | 20.01.2000    |
+------+----------------------+----------------------+----------------+---------------+
PS C:\Users\viktor\Desktop\скфу\DataAnalysis\Data3> python 2.py add -n br -s dasdasdasd -d 23.12.2000 -z asdh bruh.json
PS C:\Users\viktor\Desktop\скфу\DataAnalysis\Data3> python 2.py add -n br -s dasdasdasd -d 23.01.2000 -z asdh bruh.json
PS C:\Users\viktor\Desktop\скфу\DataAnalysis\Data3> python 2.py select -m 01 bruh.json
   1: asdhasdf sdfsdfg
   2: br dasdasdasd
PS C:\Users\viktor\Desktop\скфу\DataAnalysis\Data3> python 2.py list bruh.json
+------+----------------------+----------------------+----------------+---------------+
|  №   |         Имя          |       Фамилия        |  Знак Зодиака  | Дата рождения |
+------+----------------------+----------------------+----------------+---------------+
|  1   | asdhasdf             | sdfsdfg              | asdasf         | 20.01.2000    |
|  2   | br                   | dasdasdasd           | asdh           | 23.12.2000    |
|  3   | br                   | dasdasdasd           | asdh           | 23.01.2000    |
+------+----------------------+----------------------+----------------+---------------+
```

Рисунок 4. Выполнение кода решения индивидуального задания 1

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.

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

import os
import click
import json
from datetime import datetime
import jsonschema
from operator import itemgetter

person_schema = {
    "type": "object",
    "properties": {
        "name": {"type": "string"},
        "surname": {"type": "string"},
        "date_of_birth": {"type": "string", "format": "date"},
        "zodiac_sign": {"type": "string"}
    },
    "required": ["name", "surname", "date_of_birth", "zodiac_sign"]
}


def validate_person(person_data, schema):
    try:
        jsonschema.validate(person_data, schema)
        return True
    except jsonschema.exceptions.ValidationError as e:
        print(f"Данные человека не соответствуют схеме: {e}")
        return False


def add_person(people, name, surname, date_of_birth, zodiac_sign):
    """
    Добавление нового человека в список.
    Список сортируется по знаку зодиака после добавления нового элемента.
    """
    date_of_birth = datetime.strptime(date_of_birth, '%d.%m.%Y')

    person = {
        'name': name,
        'surname': surname,
        'date_of_birth': date_of_birth,
        'zodiac_sign': zodiac_sign
    }

    people.append(person)
    people.sort(key=lambda item: item.get('zodiac_sign', ''))
    return people


def list_people(people):
    """
    Вывод таблицы людей.
    """
    line = '+-{}-+-{}-+-{}-+-{}-+-{}-+'.format(
        '-' * 4,
        '-' * 20,
        '-' * 20,
        '-' * 15,
        '-' * 13
    )
    print(line)
```

Рисунок 5. Код решения индивидуального задания 2

```
    print(
        '| {:^4} | {:^20} | {:^20} | {:^15} | {:^12} |'.format(
            "№",
            "Имя",
            "Фамилия",
            "Знак Зодиака",
            "Дата рождения"
        )
    )
    print(line)

    for idx, person in enumerate(people, 1):
        birth_date_str = person.get('date_of_birth').strftime('%d.%m.%Y')
        print(
            '| {:^4} | {:<20} | {:<20} | {:<15} | {:<13} |'.format(
                idx,
                person.get('name', ''),
                person.get('surname', ''),
                person.get('zodiac_sign', ''),
                birth_date_str
            )
        )

    print(line)


def select_people(people, month):
    """
    Вывести список людей, родившихся в заданном месяце.
    """
    count = 0
    for person in people:
        if person.get('date_of_birth').month == month:
            count += 1
            print('{:>4}: {} {}'.format(count, person.get(
                'name', ''), person.get('surname', '')))

    if count == 0:
        print("Люди, родившиеся в указанном месяце, не найдены.")


def save_people(file_name, staff):
    """
    Сохранить всех работников в файл JSON.
    """
    file_path = os.path.join('data', file_name)
    staff_formatted = [{**person, 'date_of_birth': person.get(
        'date_of_birth').strftime('%d.%m.%Y')} for person in staff]
    # Открыть файл с заданным именем для записи.
    with open(file_path, "w", encoding="utf-8") as fout:
        # Выполнить сериализацию данных в формат JSON.
        json.dump(staff_formatted, fout, ensure_ascii=False, indent=4)


def load_people(file_name):
    """
    Загрузить всех людей из файла JSON.
    """
    file_path = os.path.join('data', file_name)
    # Открыть файл с заданным именем для чтения.
    with open(file_path, "r", encoding="utf-8") as fin:
        staff_loaded = json.load(fin)
        result_people = []
```

Рисунок 6. Код решения индивидуального задания 2

```
        cnt = 0
        for person in staff_loaded:
            cnt += 1
            if validate_person(person, person_schema):
                try:
                    person['date_of_birth'] = datetime.strptime(
                        person['date_of_birth'], '%d.%m.%Y')
                    result_people.append(person)
                except:
                    print(
                        f"Ошибка при разборе даты в записи, пропуск записи {cnt}.")
            else:
                print("Неверные данные человека, пропуск записи.")
        return result_people


@click.group()
@click.argument('filename')
@click.pass_context
def cli(ctx, filename):
    """Управление списком людей."""
    ctx.ensure_object(dict)
    ctx.obj['FILENAME'] = filename

@cli.command()
@click.option('-n', '--name', required=True, help="Имя человека")
@click.option('-s', '--surname', required=True, help="Фамилия человека")
@click.option('-d', '--date_of_birth', required=True, help="Дата рождения (формат ДД.ММ.ГГГГ)")
@click.option('-z', '--zodiac_sign', required=True, help="Знак зодиака")
@click.pass_context
def add(ctx, name, surname, date_of_birth, zodiac_sign):
    """Добавить человека."""
    people = load_people(ctx.obj['FILENAME'])
    people = add_person(people, name, surname, date_of_birth, zodiac_sign)
    save_people(ctx.obj['FILENAME'], people)

@cli.command()
@click.pass_context
def list(ctx):
    """Вывести список людей."""
    people = load_people(ctx.obj['FILENAME'])
    list_people(people)

@cli.command()
@click.option('-m', '--month', required=True, type=int, help="Месяц рождения")
@click.pass_context
def select(ctx, month):
    """Выбрать людей по месяцу рождения."""
    people = load_people(ctx.obj['FILENAME'])
    select_people(people, month)

if __name__ == '__main__':
    cli(obj={})
```

Рисунок 7. Код решения индивидуального задания 2

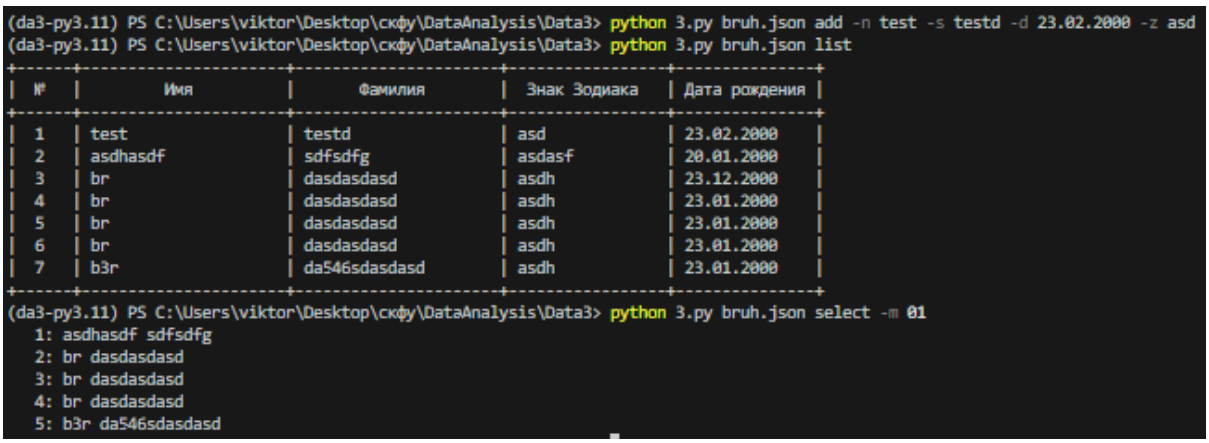


Рисунок 8. Выполнение кода решения индивидуального задания 2

9. Зафиксировал сделанные изменения в репозитории.

10. Добавил отчет по лабораторной работе в формате PDF в папку doc репозитория. Зафиксировал изменения.

11. Выполнил слияние ветки для разработки с веткой master/main.

12. Отправил сделанные изменения на сервер GitHub.

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.