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).

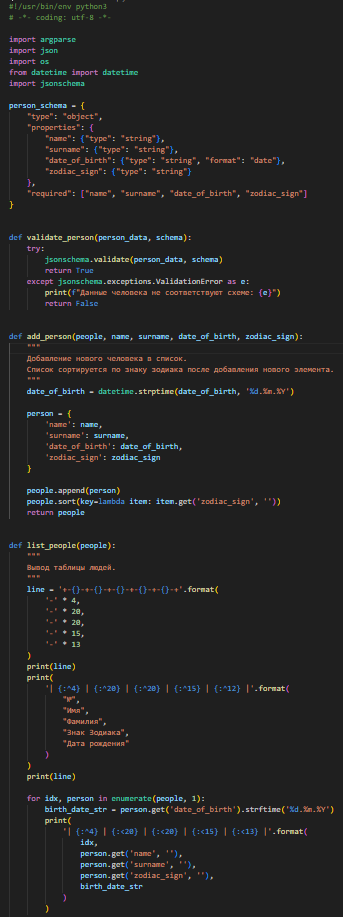


Рисунок 1. Код решения индивидуального задания 1

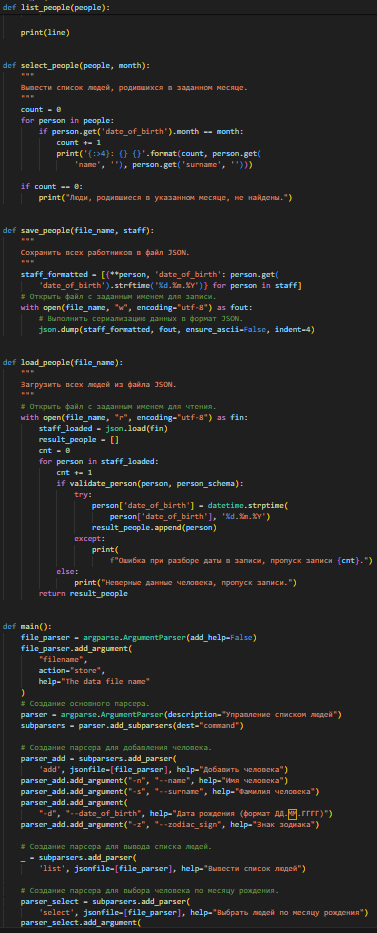


Рисунок 2. Код решения индивидуального задания 1

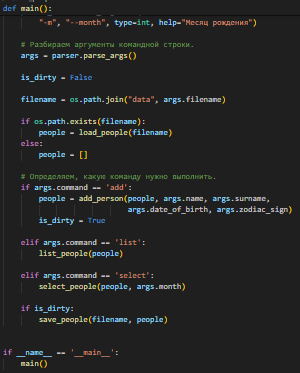


Рисунок 3. Код решения индивидуального задания 1

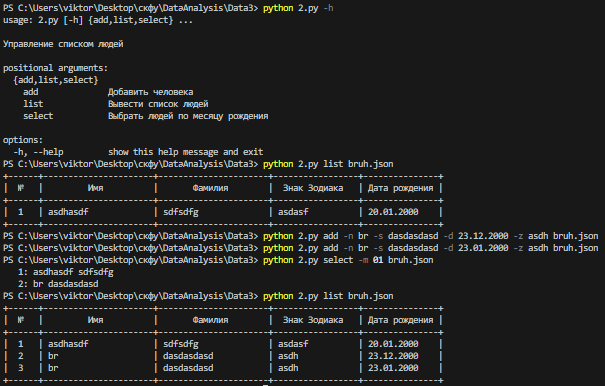


Рисунок 4. Выполнение кода решения индивидуального задания 1

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.

#!/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

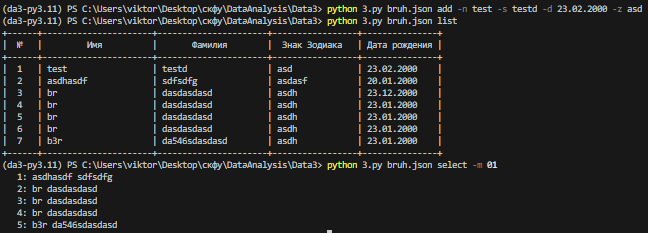


Рисунок 8. Выполнение кода решения индивидуального задания 2

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.