

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 №20
дисциплины «Основы программной инженерии»

Выполнила:
Ламская Ксения Вячеславовна
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и
сопровождение программного
обеспечения», очная форма обучения

(подпись)

Доцент кафедры инфокоммуникаций
Воронкин Роман Александрович

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

Тема: Лабораторная работа 2.17. Разработка приложений с интерфейсом командной строки (CLI) в Python3.

Цель работы: приобретение навыков по работе данными формата JSON при написании программ с помощью языка программирования Python версии 3.x.

Порядок выполнения работы


1. Создание репозитория GitHub.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*



Owner * / Repository name *

 ksenia-lamskaya / 20laba

✓ 20laba is available.

Great repository names are short and memorable. Need inspiration? How about [bug-free-couscous](#) ?

Description (optional)

- ☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.
- ☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

- ☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: Python ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: MIT License ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  main as the default branch. Change the default name in your [settings](#).

 You are creating a public repository in your personal account.

Create repository

Рисунок 1.1 – Создание репозитория

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

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

import argparse
import json
import sys
from datetime import datetime

from jsonschema import validate
from jsonschema.exceptions import ValidationError

def validation(instance):
    schema = {
        "type": "array",
        "items": {
            "type": "object",
            "properties": {
                "name1": {"type": "string"},
                "name2": {"type": "string"},
                "number": {"type": "number"},
            },
        },
        "required": ["name1", "name2", "number"],
    }

    try:
        validate(instance, schema=schema)
        return True
    except ValidationError as err:
        print(err.message)
        return False

def help():
    """
    Функция для вывода списка команд
    """
    # Вывести справку о работе с программой.
    print("Список команд:\n")
    print("add - добавить маршрут;")
    print("list - вывести список маршрутов;")
    print("select <тип> - вывод на экран пунктов маршрута, используя номер маршрута;")
    print("help - отобразить справку;")
    print("exit - завершить работу с программой.")
    print("load - загрузить данные из файла;")
    print("save - сохранить данные в файл;")
```

```
def load_point(file_name):
    """Загрузка списка маршрутов из файла."""
    try:
        with open(file_name, "r", encoding="utf-8") as f:
            return json.load(f)
    except FileNotFoundError:
        print(f"Файл {file_name} не найден.")
    except json.JSONDecodeError as e:
        print(f"Ошибка декодирования JSON в файле {file_name}: {e}")
    except Exception as e:
        print(f"Ошибка при загрузке данных из файла {file_name}: {e}")
    return None
```

```
def save_point(file_name, point_list):
    """Сохранение списка маршрутов в файл."""
    with open(file_name, "w", encoding="utf-8") as f:
        json.dump(point_list, f, ensure_ascii=False, indent=4)
```

```
def add(name1, name2, number):
    """Добавление маршрута в список."""
    return {
        'name1': name1,
        'name2': name2,
        'number': number
    }
```

```
def error(command):
    """
    функция для неопознанных команд
    """
    print(f"Неизвестная команда {command}")
```

```
def list(point):
    """
    Функция для вывода списка добавленных маршрутов
    """
    # Заголовок таблицы.
    line = '+-{}-+-{}-+-{}-+-{}-+'.format(
        '-' * 4,
        '-' * 30,
        '-' * 20,
        '-' * 8
    )
    print(line)
    print(
        '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(

```

```

        "№",
        "Начальный пункт.",
        "Конечный пункт",
        "№ маршрута"
    )
)
print(line)

# Вывести данные о всех маршрутах.
for idx, i in enumerate(point, 1):
    print(
        '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
            idx,
            i.get('name', ''),
            i.get('name2', ''),
            i.get('number', '')
        )
    )
print(line)

def select(point):
    """
    Функция для получения маршрута по его номеру
    """
    # Разбить команду на части для выделения номера маршрута.
    parts = input("Введите значение: ")
    # Проверить сведения работников из списка.

    # Проверить сведения.
    flag = True
    for i in point:
        if i['number'] == int(parts):
            print("Начальный пункт маршрута - ", i["name"])
            print("Конечный пункт маршрута - ", i["name2"])
            flag = False
            break
    if flag:
        print("Маршрут с таким номером не найден")

def parse_args():
    parser = argparse.ArgumentParser(description="Управление маршрутами")
    parser.add_argument('-l', '--load', type=str, help='Загрузить данные из файла')
    parser.add_argument('-s', '--save', type=str, help='Сохранить данные в файл')
    parser.add_argument('-a', '--add', action='store_true', help='Добавить новый маршрут')
    parser.add_argument('-d', '--display', action='store_true', help='Вывести список маршрутов')

```

```

    parser.add_argument('-n', '--number', type=int, help='Вывести маршрут по номеру')
    return parser.parse_args()

def main():
    parser = argparse.ArgumentParser(description="Управление маршрутами")
    subparsers = parser.add_subparsers(dest='command', help='Доступные команды')

    list_parser = subparsers.add_parser('list', help='Отобразить список маршрутов')
    list_parser.add_argument('filename', type=str, help='Имя файла с маршрутами для отображения')
    add_parser = subparsers.add_parser('add', help='Добавить новый маршрут')
    add_parser.add_argument('filename', type=str, help='Имя файла для сохранения маршрута')
    add_parser.add_argument('-s', '--start', required=True, help='Название начального пункта маршрута', metavar='START')
    add_parser.add_argument('-n', '--end', required=True, help='Название конечного пункта маршрута', metavar='END')
    add_parser.add_argument('-z', '--number', type=int, required=True, help='Номер маршрута', metavar='NUMBER')

    args = parser.parse_args()
    points = load_point(args.filename) if args.command in ['add', 'list', 'select'] else []
    is_dirty = False

    match args.command:
        case 'list':
            if points:
                list(points)
            else:
                print(f"Не удалось загрузить данные из файла {args.filename}.")

        case 'add':
            if args.start and args.end and args.number:
                new_point = add(args.start, args.end, args.number)
                points.append(new_point)
                save_point(args.filename, points)
                print("Маршрут успешно добавлен.")
                is_dirty = True

        case 'select':
            selected_point = select(points, args.number)
            if selected_point:
                print("Выбранный маршрут:")
                print(selected_point)
            else:
                print(f"Маршрут с номером {args.number} не найден.")

```

```
case _:  
    parser.print_help()  
  
if is_dirty:  
    save_point(args.filename, points)  
  
if __name__ == '__main__':  
    main()
```

Рисунок 2.1 – Код программы

```
python ind1.py add data.json -s "Москва" -n "Санкт-Петербург" -z 100
```

Рисунок 2.2 – Результат программы

1. В чем отличие терминала и консоли?

Термин "консоль" обычно относится к физическому устройству или его программному представлению, через которое происходит взаимодействие с компьютером. В прошлом это были специальные устройства с монитором и клавиатурой, подключаемые напрямую к компьютеру. Сейчас термин часто используется как синоним командной строки в операционной системе.

Термин "терминал" исходит из времен, когда компьютерами управляли посредством удаленных терминальных станций. Сегодня этот термин чаще всего применяется к программе эмулятору терминала, такой как Terminal в macOS, GNOME Terminal в Linux и Command Prompt в Windows, которая имитирует ввод/вывод текстовой консоли в графической среде.

По сути, в современном понимании разница между терминами стала размыта, но первоначально консоль представляла собой оборудование, а терминал — точку доступа к консоли.

2. Что такое консольное приложение?

Консольное приложение — это программа, которая взаимодействует с пользователем через текстовый интерфейс. Такие приложения получают входные данные через стандартный ввод (обычно с клавиатуры) и выводят результаты обработки на стандартный вывод (текстовая консоль или терминал). Консольные приложения широко используются для автоматизации задач, скриптинга и администрирования системы.

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

Python предлагает несколько стандартных модулей для создания CLI приложений:

- `sys` — предоставляет доступ к некоторым переменным и функциям, взаимодействующим с интерпретатором Python, включая аргументы командной строки.

- `argparse` — мощный модуль для обработки аргументов командной строки. Позволяет создавать пользовательские CLI с использованием опций и подкоманд.

- `getopt` — модуль, вдохновленный функцией C ``getopt()``. Он служит для парсинга аргументов командной строки, но его возможности несколько ограничены по сравнению с ``argparse``.

4. Какие особенности построение CLI с использованием модуля `sys`?

Модуль ``sys`` предоставляет доступ к списку аргументов командной строки через ``sys.argv``, который является списком строк. Основное преимущество ``sys`` в его простоте, но он также ограничен:

- Не поддерживает непосредственно опции (ключи) командной строки.
- Требуется ручная обработка аргументов и конвертации типов.
- Не предоставляет средства для вывода справки по использованию программы.

5. Какие особенности построения CLI с использованием модуля `getopt`?

``getopt`` схож с инструментами обработки аргументов в языке C и поддерживает стили опций как в UNIX, так и в GNU. Он позволяет определять короткие и длинные опции командной строки и автоматически генерировать сообщения об ошибках для нераспознанных опций. Недостатки:

- Интерфейс может показаться менее интуитивным по сравнению с ``argparse``.
- Менее мощный и гибкий в настройке, чем ``argparse``.

6. Какие особенности построения CLI с использованием модуля `argparse`?

``argparse`` — это рекомендуемый модуль для создания CLI в Python. Основные особенности:

- Поддерживает создание сложных CLI с опциями и подкомандами.
- Автоматически генерирует сообщения о помощи и использовании программы.
- Позволяет выполнять проверку и конвертацию типов аргументов.
- Обеспечивает гибкость при настройке поведения и вывода программы.

В целом, ``argparse`` предлагает наиболее полный набор функциональности для построения удобных и мощных командных интерфейсов.