

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 №19
дисциплины «Основы программной инженерии»

Выполнил:

Джараян Арег Александрович
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и
сопровождение программного
обеспечения», очная форма обучения

(подпись)

Проверил Воронкин Роман Александрович

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

Тема: Лабораторная работа 2.16 Работа с данными формата JSON языке Python.

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

Ход работы.

1. Создание нового репозитория с лицензией MIT.

Import a repository.

Required fields are marked with an asterisk (*).

Owner * / Repository name *

aregdz / l19

l19 is available.

Great repository names are short and memorable. Need inspiration? How about [improved-journey](#) ?

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](#).

i You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание репозитория

2. Клонировал репозиторий на рабочий ПК.

```
aregd@DESKTOP-5KV9QA9 MINGW64 ~  
$ cd "D:\Рабочий стол\4 семестр\опи\l19"  
  
aregd@DESKTOP-5KV9QA9 MINGW64 /d/Рабочий стол/4 семестр/опи/l19  
$ git clone https://github.com/aregdz/l19.git  
Cloning into 'l19'...  
remote: Enumerating objects: 5, done.  
remote: Counting objects: 100% (5/5), done.  
remote: Compressing objects: 100% (4/4), done.  
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0  
Receiving objects: 100% (5/5), done.  
  
aregd@DESKTOP-5KV9QA9 MINGW64 /d/Рабочий стол/4 семестр/опи/l19  
$ |
```

Рисунок 2 – Клонирование репозитория

3. Дополнил файл .gitignore необходимыми инструкциями.

```

1  # Byte-compiled / optimized / DLL files
2  __pycache__/
3  *.py[cod]
4  *$py.class
5
6  # C extensions
7  *.so
8
9  # Distribution / packaging
10 .Python
11 build/
12 develop-eggs/
13 dist/
14 downloads/
15 eggs/
16 .eggs/
17 lib/
18 lib64/
19 parts/
20 sdist/
21 var/
22 wheels/
23 share/python-wheels/

```

Рисунок 3 – Файл .gitignore

```

aregd@DESKTOP-5KV9QA9 MINGW64 /d/Рабочий стол/4 семестр/опи/119/119 (main)
$ git checkout -b develop
Switched to a new branch 'develop'

aregd@DESKTOP-5KV9QA9 MINGW64 /d/Рабочий стол/4 семестр/опи/119/119 (develop)
$

```

Рисунок 4 – организация ветки

4. Создание виртуального окружения.

```
PS D:\Рабочий стол\4 семестр\опи\l19\l19> python -m venv venv
PS D:\Рабочий стол\4 семестр\опи\l19\l19> venv\scripts\activate
(venv) PS D:\Рабочий стол\4 семестр\опи\l19\l19> 
```

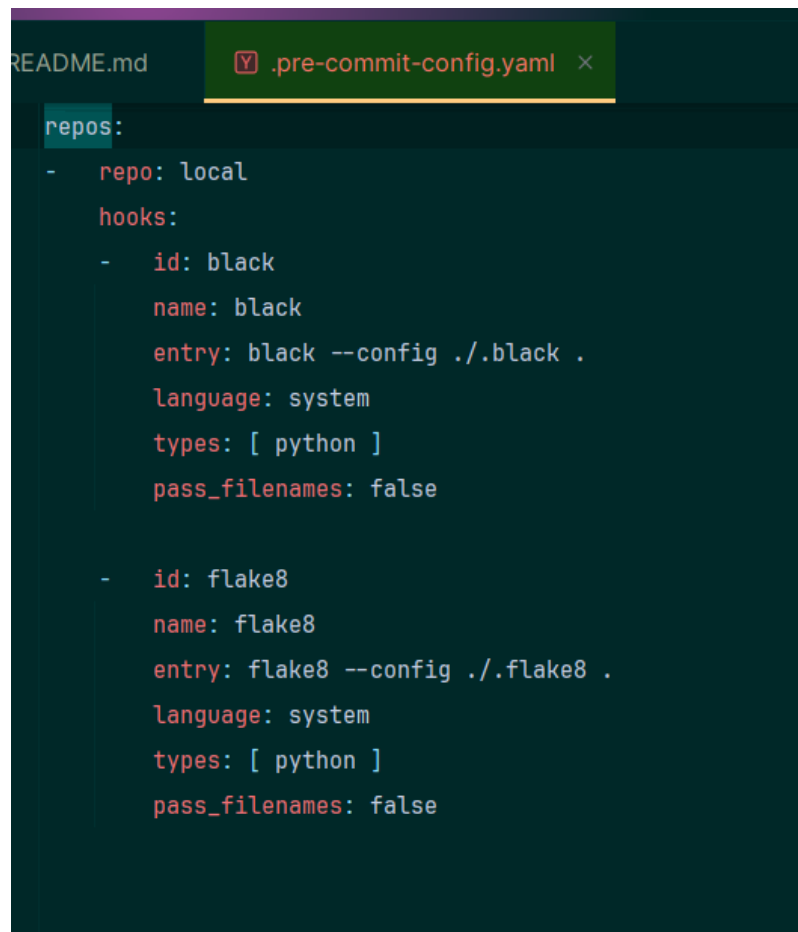
Рисунок 5 – Терминал

5. Установка пакетов.

```
(venv) PS D:\Рабочий стол\4 семестр\опи\l19\l19> pip install flake8, black, pre-commit
Collecting flake8
  Obtaining dependency information for flake8 from https://files.pythonhosted.org/packages/e3/01/cc8cdec7b61db031
  Using cached flake8-7.0.0-py2.py3-none-any.whl.metadata (3.8 kB)
Collecting black
  Obtaining dependency information for black from https://files.pythonhosted.org/packages/3e/58/89e5f5a1c4c5b66dc
  Using cached black-24.2.0-cp312-cp312-win_amd64.whl.metadata (74 kB)
Collecting pre-commit
  Obtaining dependency information for pre-commit from https://files.pythonhosted.org/packages/f8/7c/f7a50d07ae9f
```

Рисунок 7 – Установка

6. Создание файла .pre-commit-config.yaml.



```
README.md .pre-commit-config.yaml x
repos:
- repo: local
  hooks:
  - id: black
    name: black
    entry: black --config ./black .
    language: system
    types: [ python ]
    pass_filenames: false

  - id: flake8
    name: flake8
    entry: flake8 --config ./flake8 .
    language: system
    types: [ python ]
    pass_filenames: false
```

Рисунок 8 – создание файла

7. Выполнил примеры, приведённые в работе.

8. Для своего варианта лабораторной работы 2.8 необходимо дополнительно реализовать сохранение и чтение данных из файла формата JSON. Необходимо также проследить за тем, чтобы файлы генерируемый этой программой не попадали в репозиторий лабораторной работы.

Задание повышенной сложности

Очевидно, что программа в примере 1 и в индивидуальном задании никак не проверяет правильность загружаемых данных формата JSON. В следствие чего, необходимо после загрузки из файла JSON выполнять валидацию загруженных данных. Валидацию данных необходимо производить с использованием спецификации JSON Schema, описанной на сайте <https://json-schema.org/>. Одним из возможных вариантов работы с JSON Schema является использование пакета `jsonschema`, который не является частью стандартной библиотеки Python. Таким образом, необходимо реализовать валидацию загруженных данных с помощью спецификации JSON Schema.

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

import json
import jsonschema

def help1():
    """
    Функция для вывода списка команд
    """
    # Вывести справку о работе с программой.
    print("Список команд:\n")
    print("add - добавить рейс;")
    print("list - вывести список рейсов;")
    print("select <тип> - вывод на экран пунктов назначения и номеров рейсов для
данного типа самолёта")
    print("select <стаж> - запросить работников со стажем;")
    print("help - отобразить справку;")
    print("load - загрузить данные из файла;")
    print("save - сохранить данные в файл;")
    print("exit - завершить работу с программой.")

def add1():
    """
    Функция для добавления информации о новых рейсах
    """
    # Запросить данные о работнике.
    name = input("Название пункта назначения рейса? ")
    number = int(input("Номер рейса? "))
    tip = input("Тип самолета? ")
    # Создать словарь.
    i = {
        'name': name,
        'number': number,
        'tip': tip,
    }

    return i

def error1(command):
    """
    функция для неопознанных команд
    """
    print(f"Неизвестная команда {command}")

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

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

```

```

    print(line)

def validate_json_data(data):

    schema = {
        "title": "Aircrafts",
        "type": "array",
        "items": {
            "type": "object",
            "properties": {
                "name": {"type": "string"},
                "number": {"type": "integer"},
                "tip": {"type": "string"}
            },
            "additionalProperties": False,
            "required": ["name", "number", "tip"]
        }
    }

    try:
        jsonschema.validate(data, schema)
        print("Данные прошли валидацию.")
        return True
    except jsonschema.exceptions.ValidationError as e:
        print("Ошибка валидации данных:", e)
        return False

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 кириллицы 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:
        n = json.load(fin)
        if validate_json_data(n):
            return n
        else:
            return False

def select(command, aircrafts):
    """
    Функция для получения номера рейса и пункта назначения по заданному типу самолёта.
    """
    # Разбить команду на части для выделения номера года.
    parts = command.split(' ', maxsplit=1)
    # Проверить сведения работников из списка.
    count = 0

    for i in aircrafts:
        for k, v in i.items():

            if v == parts[1]:
                print("Пункт назначения - ", i["name"])
                print("Номер рейса - ", i["number"])
                count += 1
    # Если счетчик равен 0, то работники не найдены.

    if count == 0:
        print("Рейс с заданным типом самолёта не найден.")

```



```

def main1():
    """
    Главная функция программы.
    """
    print("Список команд:\n")
    print("add - добавить рейс;")
    print("list - вывести список рейсов;")
    print("select <тип> - вывод на экран пунктов назначения и номеров рейсов для
данного типа самолёта")
    print("select <стаж> - запросить работников со стажем;")
    print("help - отобразить справку;")
    print("load - загрузить данные из файла;")
    print("save - сохранить данные в файл;")
    print("exit - завершить работу с программой.")
    # Список работников.
    aircrafts = []
    # Организовать бесконечный цикл запроса команд.
    while True:
        # Запросить команду из терминала.
        command = input(">>> ").lower()
        # Выполнить действие в соответствии с командой.

        if command == 'exit':
            break

        elif command == 'add':
            # Добавить словарь в список.
            i = add1()
            aircrafts.append(i)
            # Отсортировать список в случае необходимости.
            if len(aircrafts) > 1:
                aircrafts.sort(key=lambda item: item.get('name', ''))

        elif command.startswith("save "):
            # Разбить команду на части для выделения имени файла.
            parts = command.split(maxsplit=1)
            # Получить имя файла.
            file_name = parts[1]

            # Сохранить данные в файл с заданным именем.
            save_workers(file_name, aircrafts)

        elif command.startswith("load "):
            # Разбить команду на части для выделения имени файла.
            parts = command.split(maxsplit=1)
            # Получить имя файла.
            file_name = parts[1]

            # Сохранить данные в файл с заданным именем.
            aircrafts = load_workers(file_name)

        elif command == 'list':
            list(aircrafts)

        elif command.startswith('select '):
            select(command, aircrafts)

        elif command == 'help':
            help1()

        else:
            error1("command")

if __name__ == '__main__':
    main1()

```

Рисунок 9 – выполнение задания

```

add - добавить рейс;
list - вывести список рейсов;
select <тип> - вывод на экран пунктов назначения и номеров рейсов для данного типа самолёта
select <стаж> - запросить работников со стажем;
help - отобразить справку;
load - загрузить данные из файла;
save - сохранить данные в файл;
exit - завершить работу с программой.
>>> add
Название пункта назначения рейса? Минеральные воды
Номер рейса? 123
Тип самолета? va
>>> list
+-----+-----+-----+
|  1  | Минеральные воды          | 123          |    va    |
+-----+-----+-----+
>>> save asd.json
>>> load zxc.json
Данные прошли валидацию.
>>> list
+-----+-----+-----+
|  1  | sdf                        | 345          |    dfg   |
|  2  | sdfg                      | 456          |    sdf   |
+-----+-----+-----+
>>>

```

Рисунок 10 – пример выполнения программы

```

nothing to commit, working tree clean
(venv) PS D:\Рабочий стол\4 семестр\опи\l19\l19> pre-commit install
pre-commit installed at .git\hooks\pre-commit
(venv) PS D:\Рабочий стол\4 семестр\опи\l19\l19> git add .
(venv) PS D:\Рабочий стол\4 семестр\опи\l19\l19> git commit -m"one"
black.....(no files to check)Skipped
flake8.....(no files to check)Skipped
On branch develop
nothing to commit, working tree clean
(venv) PS D:\Рабочий стол\4 семестр\опи\l19\l19> git status
On branch develop
nothing to commit, working tree clean
(venv) PS D:\Рабочий стол\4 семестр\опи\l19\l19>

```

Рисунок 11 – закомитил изменения

```
nothing to commit, working tree clean
(venv) PS D:\Рабочий стол\4 семестр\опи\л19\л19> git push origin develop
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 8 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (17/17), 4.83 KiB | 1.61 MiB/s, done.
Total 17 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:   https://github.com/aregdz/l19/pull/new/develop
remote:
To https://github.com/aregdz/l19.git
 * [new branch]      develop -> develop
(venv) PS D:\Рабочий стол\4 семестр\опи\л19\л19>
```

Рисунок 12 – отправка изменений

9. Слил ветки.

Контрольные вопросы:

1. JSON (JavaScript Object Notation) используется для обмена данными между сервером и клиентом в веб-приложениях, а также для хранения и передачи структурированных данных в различных приложениях.

2. В JSON могут использоваться следующие типы значений:

- Строки (String)
- Числа (Number)
- Булевы значения (Boolean)
- Массивы (Array)
- Объекты (Object)
- Null

3. Для работы со сложными данными в JSON используются массивы и объекты. Массивы могут содержать любые типы значений, включая другие массивы и объекты. Объекты представляют собой набор пар "ключ:

значение", где ключи являются строками, а значения могут быть любого типа.

4. Формат данных JSON5 является расширением формата JSON и добавляет дополнительные функции, такие как поддержка комментариев, необязательные запятые в конце массивов и объектов, однострочные строки и многое другое.

5. Для работы с данными в формате JSON5 на языке программирования Python можно использовать сторонние библиотеки, такие как `json5`, которая предоставляет функции для чтения и записи данных в формате JSON5.

6. Язык программирования Python предоставляет модуль `json` для сериализации данных в формат JSON. Этот модуль включает функции `json.dump()` и `json.dumps()`.

7. Отличие между функциями `json.dump()` и `json.dumps()` заключается в том, что `json.dump()` используется для записи данных в файл, а `json.dumps()` возвращает строку JSON.

8. Для десериализации данных из формата JSON в языке Python также используется модуль `json`. В этом модуле есть функции `json.load()` для чтения из файла и `json.loads()` для чтения из строки.

9. Для работы с данными формата JSON, содержащими кириллицу, необходимо использовать правильную кодировку при чтении и записи файлов, например, `utf-8`.

10. JSON Schema - это спецификация, которая определяет формат и структуру данных в формате JSON. Она используется для описания и валидации данных. Схема данных JSON определяет ожидаемую структуру объекта JSON, включая его типы данных, ограничения и допустимые значения.