

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ

ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития

Кафедра информационных систем и технологий

Отчет по лабораторной работе №17.

Дисциплина: «Основы программной инженерии»

Выполнил:

Студент группы ПИЖ-б-о-22-1,

направление подготовки: 09.03.04

«Программная инженерия»

ФИО: Рядская Мария Александровна

Проверил:

Воронкин Р. А.

Ставрополь 2024

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.

### 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 \*

 mryadskaya

/ lab19

lab19 is available.

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

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.](#)

Рисунок 1 – Создание репозитория

## 2. Клонировала репозиторий на рабочий ПК.

Рисунок 2 – Клонирование репозитория

```
C:\git1>git clone https://github.com/mryadskaya/lab19.git
Cloning into 'lab19'...
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.

C:\git1>
```

3. Дополнила 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     ----/
```

Рисунок 3 – Файл .gitignore

```
C:\git1\lab19>git checkout -b develop
Switched to a new branch 'develop'
```

```
C:\git1\lab19>
```

Рисунок 4 – организация ветки

#### 4. Создание виртуального окружения.

```
(base) PS C:\git1\lab19> python -m venv venv
(base) PS C:\git1\lab19> venv\scripts\activate
(venv) (base) PS C:\git1\lab19>
```

Рисунок 5 – Терминал

#### 5. Установка пакетов.

```
(venv) (base) PS C:\git1\lab19> pip install flake8, black, pre-commit Collecting flag8
Collecting flake8
  Obtaining dependency information for flake8 from https://files.pythonhosted.org/packages/e3/01/cc8cdec7b61db0315c2ab62d80677a138ef0
6832ec17f04d87e6ef858f7f/flake8-7.0.0-py2.py3-none-any.whl.metadata
  Downloading 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/0f/11/fa05ac9429d971d0fc10da85f24dafc3fa578
8733fb0dd1c186b7577cefdb/black-24.4.0-cp311-cp311-win_amd64.whl.metadata
  Downloading black-24.4.0-cp311-cp311-win_amd64.whl.metadata (76 kB)
----- 76.4/76.4 kB 1.4 MB/s eta 0:00:00
Collecting pre-commit
  Obtaining dependency information for pre-commit from https://files.pythonhosted.org/packages/53/1a/1870d52d0d86d534d4d5f7a58e65d40a
4610b57e13309917faa9e62818fe/pre_commit-3.7.0-py2.py3-none-any.whl.metadata
  Downloading pre_commit-3.7.0-py2.py3-none-any.whl.metadata (1.3 kB)
ERROR: Could not find a version that satisfies the requirement Collecting (from versions: none)
ERROR: No matching distribution found for Collecting
```

Рисунок 7 – Установка

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

## arm / pre-commit-config.yaml

skaya Add files via upload

lame 23 lines (21 loc) · 541 Bytes  Code 55% faster with GitHub

```
repos:
  - repo: local
    hooks:
      - id: black
        name: black
        entry: black --config ./pyproject.toml .
        language: system
        types: [ python ]
        pass_filenames: false

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

      - id: isort
        name: isort
        entry: isort --settings-file ./pyproject.toml .
        language: system
        types: [ python ]
```

Рисунок 8 – создание файла

7. Выполнила 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:")
    print("add - добавить информацию;")
    print("list - вывести список ;")
    print(
        "select <тип> - вывод на экран фамилия, имя; знак Зодиака; дата рождения ")
    print("help - отобразить справку;")
    print("exit - завершить работу с программой.")
# Список работников.
workers = []

def add1():
    """
Функция для добавления информации о новых рейсах
    """
    # Запросить данные о работнике.

    name = input("Фамилия и инициалы? ")
    post = input("знак зодиака? ")
    """
```

Рисунок 9 – выполнение задания

```
Список команд:
```

```
add - добавить рейс;
list - вывести список рейсов;
select <тип> - вывод на экран пунктов назначения и номеров рейсов для данного
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
Фамилия и инициалы? ryadsksya
знак зодиака? rak
Год рождения? 2005
>>> list
+-----+-----+-----+
|    1 | ryadsksya           | rak          | 2005 |
+-----+-----+-----+
>>> save mar.json
>>> load ram.json
```

Рисунок 10 – пример выполнения программы

```
+ FullyQualifiedErrorId : CommandNotFoundException

(venv) (base) PS C:\git1\lab19> git add .
(venv) (base) PS C:\git1\lab19> git commit -m"one"
On branch develop
nothing to commit, working tree clean
(venv) (base) PS C:\git1\lab19>
```

Рисунок 11 – закомитила изменения

```
(venv) (base) PS C:\git1\lab19> git push origin develop
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:     https://github.com/mryadskaya/lab19/pull/new/develop
remote:
To https://github.com/mryadskaya/lab19.git
 * [new branch]      develop -> develop
(venv) (base) PS C:\git1\lab19>
```

Рисунок 12 – отправка изменений

9. Слила 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)

-Булевые 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.