

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЁТ

по лабораторной работе №2.16

Дисциплина: «Анализ данных»
Тема: «Работа с данными формата JSON в языке Python»

Выполнил:
Епифанов Алексей Александрович
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.

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

Порядок выполнения работы:

1. Создал новый репозиторий, клонировал его, в нем создал ветку developer и перешел на нее.
2. Проработал пример лабораторной работы:

```
(base) aleksejepifanov@MacBook-Pro-Aleksej gits % /Applications/anaconda3/bin/python /Users/aleksejepifanov/Desktop/пары/пара_4_сем/data_analysis/gits/Lab_2.16/program/prim.py
>>> help
Список команд:

add – добавить работника;
list – вывести список работников;
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? epifanov a. a.
Должность? programmer
Год поступления? 2022
>>> add
Фамилия и инициалы? Kozh V. A.
Должность? proger
Год поступления? 2012
>>> list
```

No	Ф.И.О.	Должность	Год
1	Kozh V. A.	proger	2012
2	epifanov a. a.	programmer	2022

```
>>> select 10
```

No	Ф.И.О.	Должность	Год
1	Kozh V. A.	proger	2012

```
>>> save file.json
>>> exit
(base) aleksejepifanov@MacBook-Pro-Aleksej gits %
```

Рисунок 1. Сохранение данных перед завершением работы программы

```
(base) aleksejepifanov@MacBook-Pro-Aleksej Lab_2.16 % /usr/local/bin/python3 /Users/aleksejepifanov/Desktop/нары/нары_4_сем/data_analysis/gits/Lab_2.16/program/prim.py
>>> add l m. n.
Неизвестная команда add l m. n.
>>> add
Фамилия и инициалы? l m. n.
Должность? df
Год поступления? 2020
>>> list
```

No	Ф.И.О.	Должность	Год
1	l m. n.	df	2020

```
>>> save t.json
>>> help
Список команд:
add - добавить работника;
list - вывести список работников;
select <стаж> - запросить работников со стажем;
help - отобразить справку;
load - загрузить данные из файла;
save - сохранить данные в файл;
exit - завершить работу с программой.
>>> load file.json
>>> list
```

No	Ф.И.О.	Должность	Год
1	Kozh V. A.	proger	2012
2	epifanov a. a.	programmer	2022

```
>>> exit
```

Рисунок 2. Загрузка данных после повторного запуска программы

3. Выполнил индивидуальное задание: 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. А также выполнил 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 bisect
import re
import sys
import json
from jsonschema import validate, ValidationError

def get_route():
    """
    Запросить данные о маршруте.
    """
    start = input("Введите начальный пункт: ")
    end = input("Введите конечный пункт: ")
    count = int(input("Введите номер маршрута: "))
    return {
        "начальный пункт": start,
        "конечный пункт": end,
        "номер маршрута": count,
    }

def display_routes(routes):
    """
    Отобразить список маршрутов.
    """
    if routes:
        line = "+-{}-+-{}-+-{}-+ ".format("-" * 30, "-" * 20, "-" * 8)
        print(line)
        print("| {:^30} | {:^20} | {:^8} | ".format("Начало", "Конец", "Номер"))
        print(line)
        for route in routes:
            print(
                "| {:<30} | {:<20} | {:>8} | ".format(
                    route.get("начальный пункт", ""),
                    route.get("конечный пункт", ""),
                    route.get("номер маршрута", ""),
                )
            )
            print(line)
    else:
        print("Список маршрутов пуст.")

def select_routes(routes, name_punct):
    """
    Выбрать маршруты с заданным пунктом отправления или прибытия.
    """
    selected = []
    for route in routes:
        if (
            route["начальный пункт"].lower() == name_punct
            or route["конечный пункт"].lower() == name_punct
        ):
            selected.append(route)
    return selected

def save_routes(file_name, routes):
    """
    Сохранить все маршруты в файл JSON.
    """
    # Открыть файл с заданным именем для записи.
    with open(file_name, "w") as file_out:
        # Записать данные из словаря в формат JSON и сохранить их
        # в открытый файл.
        json.dump(routes, file_out, ensure_ascii=False, indent=4)

def load_routes(file_name):
    """
    Загрузить все маршруты из файла JSON.
    """

```

```

schema = {
    "type": "array",
    "items": {
        "type": "object",
        "properties": {
            "начальный пункт": {"type": "string"},
            "конечный пункт": {"type": "string"},
            "номер маршрута": {"type": "integer"},
        },
        "required": ["начальный пункт", "конечный пункт", "номер маршрута"],
    },
}
# Открыть файл с заданным именем и прочитать его содержимое.
with open(file_name, "r") as file_in:
    data = json.load(file_in) # Прочитать данные из файла
try:
    # Валидация
    validate(instance=data, schema=schema)
    print("JSON валиден по схеме.")
except ValidationError as e:
    print(f"Ошибка валидации: {e.message}")
return data
def main():
    """
    Главная функция программы.
    """
    routes = []
    while True:
        command = input(">>> ").lower()
        match command:
            case "exit":
                break
            case "add":
                route = get_route()
                if route not in routes:
                    bisect.insort(
                        routes,
                        route,
                        key=lambda item: item.get("номер маршрута"),
                    )
                else:
                    print("Данный маршрут уже добавлен.")
            case "list":
                display_routes(routes)
            case _ if (m := re.match(r"(select|save|load) (.+)", command)):
                match m.group(1):
                    case "select":
                        name_punct = m.group(2)
                        selected = select_routes(routes, name_punct)
                        display_routes(selected)
                    case "save":
                        file_name = m.group(2)
                        save_routes(file_name, routes)
                    case "load":
                        file_name = m.group(2)
                        routes = load_routes(file_name)
            case "help":
                print("Список команд:\n")
                print("add - добавить маршрут;")
                print("list - вывести список маршрутов;")
                print(
                    "select <название пункта> - "
                    + "запросить маршруты, которые начинаются\n"

```

```

        + "или заканчиваются в данном пункте;"
    )
    print("help - отобразить справку;")
    print("load <filename> - загрузить данные из файла;")
    print("save <filename> - сохранить данные в файл;")
    print("exit - завершить работу с программой.")
    case _:
        print(f"Неизвестная команда {command}", file=sys.stderr)
if __name__ == "__main__":
    main()

```

```

● (lab-2-16-vJ5HNmp_py3.12) aleksejepifanov@MacBook-Pro-Aleksej Lab_2.16 % /Users/aleksejepifanov/Desktop/пары/пары_4_сем/data_analysis/gits/Lab_2.16/program/ind.py
>>> list
Список маршрутов пуст.
>>> help
Список команд:

add - добавить маршрут;
list - вывести список маршрутов;
select <название пункта> - запросить маршруты, которые начинаются
или заканчиваются в данном пункте;
help - отобразить справку;
load <filename> - загрузить данные из файла;
save <filename> - сохранить данные в файл;
exit - завершить работу с программой.
>>> add
Введите начальный пункт: stavoropol
Введите конечный пункт: novgorod
Введите номер маршрута: 21
>>> add
Введите начальный пункт: kislovodsk
Введите конечный пункт: stavoropol
Введите номер маршрута: 31
>>> add
Введите начальный пункт: armavir
Введите конечный пункт: kislovodsk
Введите номер маршрута: 12
>>> list

```

Начало	Конец	Номер
armavir	kislovodsk	12
stavoropol	novgorod	21
kislovodsk	stavoropol	31

```

>>> select stavoropol

```

Начало	Конец	Номер
stavoropol	novgorod	21
kislovodsk	stavoropol	31

```

>>> save routes.json
>>> exit

```

Рисунок 3. Результат работы программы и сохранение данных в файл

```
• (lab-2-16-vJ5HNmp_-py3.12) aleksejepifanov@MacBook-Pro-Aleksej Lab_2.16 % /Users/aleksejepifanov/Desktop/пары/пары_4_сем/data_analysis/gits/Lab_2.16/program/ind.py
>>> lisr
Неизвестная команда lisr
>>> list
Список маршрутов пуст.
>>> load file.json
Ошибка валидации: 'начальный пункт' is a required property
>>> load routes.json
JSON валиден по схеме.
>>> list
+-----+-----+-----+
|          Начало          |          Конец          |          Номер          |
+-----+-----+-----+
| armavir                  | kislodvorsk             |          12             |
| stavropol                | novgorod                |          21             |
| kislodvorsk              | stavropol               |          31             |
+-----+-----+-----+
>>> exit
```

Рисунок 4. Загрузка данных и валидация

Ответы на контрольные вопросы:

1. Для чего используется JSON?

JSON (англ. JavaScript Object Notation, обычно произносится как /'dʒeɪsən/ JAY-sən) – текстовый формат обмена данными, основанный на JavaScript. Как и многие другие текстовые форматы, JSON легко читается людьми. Формат JSON был разработан Дугласом Крокфордом.

Несмотря на происхождение от JavaScript (точнее, от подмножества языка стандарта ECMA-262 1999 года), формат считается независимым от языка и может использоваться практически с любым языком программирования. Для многих языков существует готовый код для создания и обработки данных в формате JSON.

За счёт своей лаконичности по сравнению с XML формат JSON может быть более подходящим для сериализации сложных структур. Применяется в веб-приложениях как для обмена данными между браузером и сервером (AJAX), так и между серверами (программные HTTP-сопряжения).

Легкочитаемый и компактный, JSON представляет собой хорошую альтернативу XML и требует куда меньше форматирования контента.

2. Какие типы значений используются в JSON?

В качестве значений в JSON могут быть использованы:

запись — это неупорядоченное множество пар ключ: значение, заключённое в фигурные скобки «{ }». Ключ описывается строкой, между ним

и значением стоит символ «:». Пары ключ-значение отделяются друг от друга запятыми.

массив (одномерный) — это упорядоченное множество значений. Массив заключается в квадратные скобки «[]». Значения разделяются запятыми. Массив может быть пустым, т.е. не содержать ни одного значения. Значения в пределах одного массива могут иметь разный тип.

число (целое или вещественное).

литералы `true` (логическое значение «истина»), `false` (логическое значение «ложь») и `null`.

строка — это упорядоченное множество из нуля или более символов юникода, заключённое в двойные кавычки. Символы могут быть указаны с использованием escape-последовательностей, начинающихся с обратной косой черты «\» (поддерживаются варианты `'`, `"`, `\`, `\\`, `\t`, `\n`, `\r`, `\f` и `\b`), или записаны шестнадцатеричным кодом в кодировке Unicode в виде `\uFFFF`.

3. Как организована работа со сложными данными в JSON?

JSON может содержать другие вложенные объекты в JSON, в дополнение к вложенным массивам.

Такие объекты и массивы будут передаваться, как значения назначенные ключам и будут представлять собой связку ключ-значение.

4. Самостоятельно ознакомьтесь с форматом данных JSON5? В чем отличие этого формата от формата данных JSON?

Формат данных JSON5 является расширенной версией формата JSON, который добавляет несколько улучшений для удобства пользователя. Основные отличия включают поддержку комментариев, однострочные и многострочные, а также разрешенные одинарные кавычки для строковых значений. JSON5 также позволяет использовать запятые в конце списков и свойств, что облегчает редактирование и управление данными.

5. Какие средства языка программирования Python могут быть использованы для работы с данными в формате JSON5?

Для работы с JSON5 в Python можно использовать стороннюю библиотеку `json5`.

6. Какие средства предоставляет язык Python для сериализации данных в формате JSON?

`json.dump()` - конвертировать python объект в json и записать в файл.

`json.dumps()` - тоже самое, но в строку.

Обе эти функции принимают следующие необязательные аргументы: Если `skipkeys = True`, то ключи словаря не базового типа (`str`, `int`, `float`, `bool`, `None`) будут проигнорированы, вместо того, чтобы вызывать исключение `TypeError`.

Если `ensure_ascii = True`, все не-ASCII символы в выводе будут экранированы последовательностями `\uXXXX`, и результатом будет строка, содержащая только ASCII символы. Если `ensure_ascii = False`, строки запишутся как есть.

Если `check_circular = False`, то проверка циклических ссылок будет пропущена, а такие ссылки будут вызывать `OverflowError`.

Если `allow_nan = False`, при попытке сериализовать значение с запятой, выходящее за допустимые пределы, будет вызываться `ValueError` (`nan`, `inf`, `-inf`) в строгом соответствии со спецификацией JSON, вместо того чтобы использовать эквиваленты из JavaScript (`NaN`, `Infinity`, `-Infinity`).

Если `indent` является неотрицательным числом, то массивы и объекты в JSON будут выводиться с этим уровнем отступа. Если уровень отступа 0, отрицательный или `""`, то вместо этого будут просто использоваться новые строки. Значение по умолчанию `None` отражает наиболее компактное представление. Если `indent` - строка, то она и будет использоваться в качестве отступа.

Если `sort_keys = True`, то ключи выводимого словаря будут отсортированы.

7. В чем отличие функций `json.dump()` и `json.dumps()`?

`json.dump()` - конвертировать python объект в json и записать в файл.

`json.dumps()` - тоже самое, но в строку.

8. Какие средства предоставляет язык Python для десериализации данных из формата JSON?

`json.load()` # прочитать json из файла и конвертировать в python объект.

`json.loads()` # тоже самое, но из строки с json.

Обе эти функции принимают следующие аргументы:

`object_hook` - опциональная функция, которая применяется к результату декодирования объекта (dict). Используется будет значение, возвращаемое этой функцией, а не полученный словарь.

`object_pairs_hook` - опциональная функция, которая применяется к результату декодирования объекта с определённой последовательностью пар ключ/значение. Будет использован результат, возвращаемый функцией, вместо исходного словаря. Если задан так же `object_hook`, то приоритет отдаётся `object_pairs_hook`.

`parse_float`, если определён, будет вызван для каждого значения JSON с плавающей точкой. По умолчанию, это эквивалентно `float(num_str)`.

`parse_int`, если определён, будет вызван для строки JSON с числовым значением. По умолчанию эквивалентно `int(num_str)`.

`parse_constant`, если определён, будет вызван для следующих строк: "-Infinity", "Infinity", "NaN". Может быть использовано для возбуждения исключений при обнаружении ошибочных чисел JSON.

9. Какие средства необходимо использовать для работы с данными формата JSON, содержащими кириллицу?

`ensure_ascii=False`.

10. Самостоятельно ознакомьтесь со спецификацией JSON Schema? Что такое схема данных? Приведите схему данных для примера 1.

JSON Schema - это спецификация, позволяющая определять формат JSON данных, их структуру, типы данных, ограничения и правила валидации. С помощью JSON Schema можно создавать схемы данных для JSON, что облегчает валидацию и документацию формата JSON.

```
{
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "name": {"type": "string"},
      "post": {"type": "string"},
      "year": {
        "type": "integer",
        "minimum": 2000,
        "maximum": 2024
      }
    },
    "required": ["name", "post", "year"]
  }
}
```

Вывод: в результате выполнения работы были приобретены навыки по работе с данными формата JSON с помощью языка программирования Python версии 3.x.