

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 №2
Дисциплины «Анализ данных»

Выполнил:

Пустяков Андрей Сергеевич

2 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и
вычислительная техника (профиль)
«Программное обеспечение средств
вычислительной
техники и автоматизированных
систем», очная форма обучения

(подпись)

Руководитель практики:

Воронкин Р. А. кандидат технических
наук, доцент, доцент кафедры
инфокоммуникаций

(подпись)

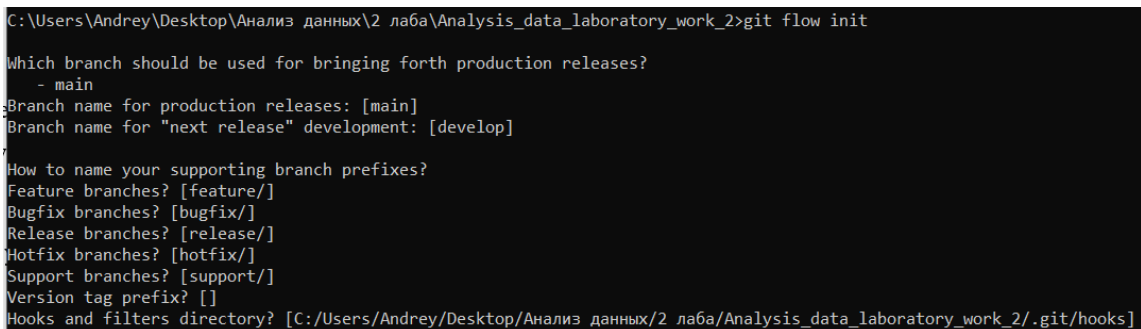
Ставрополь, 2024 г.

Тема: Работа с данными формата JSON в языке Python.

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

Ход работы:

Создание общедоступного репозитория на «GitHub», клонирование репозитория, редактирование файла «.gitignore», организация репозитория согласно модели ветвления «git-flow» (рис. 1).



```
C:\Users\Andrey\Desktop\Анализ данных\2 лаба\Analysis_data_laboratory_work_2>git flow init
Which branch should be used for bringing forth production releases?
- main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]
How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [C:/Users/Andrey/Desktop/Анализ данных/2 лаба/Analysis_data_laboratory_work_2/.git/hooks]
```

Рисунок 1 – Организация модели ветвления «git-flow».

Проработка примеров лабораторной работы:

Пример 1.

Необходимо создать программу, в которой использовать словарь, содержащий информацию о фамилии и инициалах работника, название занимаемой должности, год поступления на работу. В программе должен быть организован ввод данных в список, состоящий из словарей заданной структуры, записи должны быть упорядочены по алфавиту. Должен быть организован вывод на дисплей фамилии работников, чей стаж работы в организации превышает значение, введенное с клавиатуры, причем, если таких работников нет, то должно быть выведено соответствующее сообщение. Необходимо добавить возможность сохранения списка в файл формата JSON и чтения данных из JSON.

Код программы решения данной задачи и результаты работы программы с различными исходными данными (рис. 2, 3, 4).

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

import json
import sys
from datetime import date

def get_worker()::
    """
    Запросить данные о работнике.
    """
    name = input("Фамилия и инициалы? ")
    post = input("Должность? ")
    year = int(input("Год поступления? "))
    # Создать словарь.
    return {
        'name': name,
        'post': post,
        'year': year,
    }

def display_workers(staff):
    """
    Отобразить список работников.
    """
    # Проверить, что список работников не пуст.
    if staff:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",
                "Ф.И.О.",
                "Должность",
                "Год")
        )
        print(line)

        # Вывести данные о всех сотрудниках.
        for idx, worker in enumerate(staff, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx, worker.get('name', ''),
                    worker.get('post', ''),
                    worker.get('year', 0))
            )
            print(line)
    else:
        print("Список работников пуст.")

def select_workers(staff, period):
    """
    Выбрать работников с заданным стажем.
    """
    # Получить текущую дату.

```

```

today = date.today()

# Сформировать список работников.
result = []
for employee in staff:
    if today.year - employee.get('year', today.year) >= period:
        result.append(employee)
# Возвратить список выбранных работников.
return result

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:
        return json.load(fin)

def main():
    """
    Главная функция программы.
    """
    # Список работников.
    workers = []
    # Организовать бесконечный цикл запроса команд.
    while True:
        # Запросить команду из терминала.
        command = input(">>> ").lower()
        # Выполнить действие в соответствие с командой.
        if command == "exit":
            break

        elif command == "add":
            # Запросить данные о работнике.
            worker = get_worker()
            # Добавить словарь в список.
            workers.append(worker)
            # Отсортировать список в случае необходимости.
            if len(workers) > 1:
                workers.sort(key=lambda item: item.get('name', ''))

        elif command == "list":
            # Отобразить всех работников.
            display_workers(workers)

        elif command.startswith("select "):
            # Разбить команду на части для выделения стажа.
            parts = command.split(maxsplit=1)
            # Получить требуемый стаж.
            period = int(parts[1])
            # Выбрать работников с заданным стажем.

```

```

        selected = select_workers(workers, period)
        # Отобразить выбранных работников.
        display_workers(selected)

    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, workers)

    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.
        workers = load_workers(file_name)

    elif command == 'help':
        # Вывести справку о работе с программой.
        print("Список команд:\n")
        print("add - добавить работника;")
        print("list - вывести список работников;")
        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.")

    else:
        print(f"Неизвестная команда {command}", file=sys.stderr)

if __name__ == '__main__':
    main()

```

Рисунок 2 – Код программы примера 1

```

>>> save data
>>> select 2005
Список работников пуст.
>>> list
+-----+-----+-----+-----+
| № |          Ф.И.О.          |      Должность      |   Год   |
+-----+-----+-----+-----+
|  1 | Вторых Д.А.              | заместитель директора |   2002  |
|  2 | Железнов В.В.            | менеджер             |   2005  |
|  3 | Пустяков А.С.            | директор             |   2000  |
|  4 | Цифрова А.А.             | бухгалтер            |   2005  |
+-----+-----+-----+-----+
>>> select 2000
Список работников пуст.
>>> select 24
+-----+-----+-----+-----+
| № |          Ф.И.О.          |      Должность      |   Год   |
+-----+-----+-----+-----+
|  1 | Пустяков А.С.            | директор             |   2000  |
+-----+-----+-----+-----+
>>> select 19
+-----+-----+-----+-----+
| № |          Ф.И.О.          |      Должность      |   Год   |
+-----+-----+-----+-----+
|  1 | Вторых Д.А.              | заместитель директора |   2002  |
|  2 | Железнов В.В.            | менеджер             |   2005  |
|  3 | Пустяков А.С.            | директор             |   2000  |

```

Рисунок 3 – Результаты работы программы без подгрузки данных

```

C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe "C:\Users\Andrey\De
>>> load data
>>> select 24
+-----+-----+-----+-----+
| № |          Ф.И.О.          |      Должность      |   Год   |
+-----+-----+-----+-----+
|  1 | Пустяков А.С.            | директор             |   2000  |
+-----+-----+-----+-----+
>>> select 22
+-----+-----+-----+-----+
| № |          Ф.И.О.          |      Должность      |   Год   |
+-----+-----+-----+-----+
|  1 | Вторых Д.А.              | заместитель директора |   2002  |
|  2 | Пустяков А.С.            | директор             |   2000  |
+-----+-----+-----+-----+
>>> exit

Process finished with exit code 0

```

Рисунок 4 – Результаты работы программы с подгрузкой данных из файла

Выполнение индивидуальных заданий:

Задание 1.

Необходимо для своего варианта лабораторной работы 2.8 дополнительно реализовать сохранение и чтение данных из файлов формата JSON (генерируемые файлы не должны находиться в репозитории лабораторной работы).

Необходимо использовать словарь, содержащий следующие ключи: название пункта назначения; номер поезда; время отправления. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в список, состоящий из словарей заданной структуры; записи должны быть упорядочены по времени отправления поезда; вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры; если таких поездов нет, выдать на дисплей соответствующее сообщение (Вариант 26 (7), работа 2.8).

Код программы для решения данной задачи, для избежания сохранения файлов в папке репозитория была осуществлена смена рабочей директории в программе (рис. 5).

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

import json
import sys
import os

def get_train():
    """
    Запросить данные о поезде и пунктах.
    """
    departure_point = input("Пункт отправления поезда? ")
    number_train = input("Номер поезда? ")
    time_departure = input("Время отправления? ")
    destination = input("Пункт назначения? ")
    # Создать словарь.
    return {
        'departure_point': departure_point,
        'number_train': number_train,
        'time_departure': time_departure,
        'destination': destination,
    }

def display_trains(staff):
    """
    Отобразить список поездов.
```

```

"""
# Проверить, что список поездов не пуст.
if staff:
    # Заголовок таблицы.
    line = '+-{}-+-{}-+-{}-+-{}-+-+'.format(
        '-' * 4,
        '-' * 30,
        '-' * 13,
        '-' * 18,
        '-' * 14
    )
    print(line)
    print(
        '| {:^4} | {:^30} | {:^13} | {:^18} | {:^14} |'.format(
            "№",
            "Пункт отправления",
            "Номер поезда",
            "Время отправления",
            "Пункт назначения"
        )
    )
    print(line)

    # Вывести данные о всех поездах.
    for idx, points in enumerate(staff, 1):
        print(
            '| {:>4} | {:<30} | {:<13} | {:>18} | {:^16} |'.format(
                idx, points.get('departure_point', ''),
                points.get('number_train', ''),
                points.get('time_departure', ''),
                points.get('destination', '')
            )
        )
        print(line)
else:
    print("Список станций пуст.")

def select_trains(staff, point_user):
    """
    Выбрать поезда, направляющиеся в указанный пункт.
    """

    # Сформировать список поездов.
    result = []
    for train in staff:
        if point_user == str.lower(train['destination']):
            result.append(train)

    # Возвратить список выбранных поездов, направляющихся в пункт.
    return result

def save_trains(file_name, staff):
    """
    Сохранить все поезда со станциями в файл JSON.
    """

    # Сменить рабочую директорию, для того, что файл не попал в репозиторий
    os.chdir("C:\\Users\\Andrey\\Desktop\\Анализ_данных\\2_лаба\\Kaliningrad")
    # 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_trains(file_name):
    """
    Загрузить все поезда со станциями из файла JSON.
    """
    # Сменить рабочую директорию, для того, что файл не попал в репозиторий
    os.chdir("C:\\Users\\Andrey\\Desktop\\Анализ_данных\\2_лаба\\Kaliningrad")
    # Открыть файл с заданным именем для чтения.
    with open(file_name, "r", encoding="utf-8") as fin:
        return json.load(fin)

def main():
    """
    Главная функция программы.
    """
    # Список поездов.
    trains = []
    # Организовать бесконечный цикл запроса команд.
    while True:
        # Запросить команду из терминала.
        command = input(">>> ").lower()
        # Выполнить действие в соответствие с командой.
        if command == "exit":
            break

        elif command == "add":
            # Запросить данные о поезде.
            train = get_train()
            # Добавить словарь в список.
            trains.append(train)
            # Отсортировать список в случае необходимости по времени
            отправления поезда.
            if len(trains) > 1:
                trains.sort(key=lambda item: item.get('time_departure', ''))

        elif command == "list":
            # Отобразить все поезда.
            display_trains(trains)

        elif command.startswith("select "):
            # Разбить команду на части для выделения станции.
            point_comand = command.split(maxsplit=1)
            point_user = point_comand[1]
            # Выбрать поезда с заданным пунктом назначения.
            selected = select_trains(trains, point_user)
            # Отобразить выбранные поезда.
            display_trains(selected)

        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_trains(file_name, trains)

        elif command.startswith("load "):
            # Разбить команду на части для выделения имени файла.
            parts = command.split(maxsplit=1)
            # Получить имя файла.
            file_name = parts[1]
            # Загрузить данные из файла с заданным именем.
            trains = load_trains(file_name)

```

```

elif command == 'help':
    # Вывести справку о работе с программой.
    print("Список команд:\n")
    print("add - добавить поезд;")
    print("list - вывести список поездов;")
    print("select <станция> - запросить поезда направляющиеся в
пункт;")

    print("help - отобразить справку;")
    print("load - загрузить данные из файла;")
    print("save - сохранить данные в файл;")
    print("exit - завершить работу с программой.")

else:
    print(f"Неизвестная команда {command}", file=sys.stderr)

if __name__ == '__main__':
    main()

```

Рисунок 5 – Код программы решения данной задачи

Результаты работы программы с загрузкой списка из файла формата JSON (рис. 6).

```

C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe C:\Users\Andrey\Desktop\Анализ_данных\
>>> help
Список команд:

add - добавить поезд;
list - вывести список поездов;
select <станция> - запросить поезда направляющиеся в пункт;
help - отобразить справку;
load - загрузить данные из файла;
save - сохранить данные в файл;
exit - завершить работу с программой.
>>> load Kaliningrad
>>> list
+-----+-----+-----+-----+
| № | Пункт отправления | Номер поезда | Время отправления | Пункт назначения |
+-----+-----+-----+-----+
| 1 | Айвазовская | 001 | 6:00 | Калининград Ю. |
| 2 | Держинская | 002 | 6:10 | Калининград Ю. |
| 3 | Киевская | 003 | 6:20 | Калининград Ю. |
| 4 | Западная | 004 | 6:30 | Калининград Ю. |
| 5 | Калининград С. | 005 | 6:40 | Калининград Ю. |
| 6 | Сельма | 006 | 6:50 | Калининград С. |
| 7 | Кутузово | 007 | 7:00 | Калининград С. |
| 8 | Калининград Ю. | 008 | 7:10 | Калининград С. |
+-----+-----+-----+-----+
>>> select Калининград Ю.
+-----+-----+-----+-----+
| № | Пункт отправления | Номер поезда | Время отправления | Пункт назначения |
+-----+-----+-----+-----+

```

```

+-----+-----+-----+-----+
| 1 | Айвазовская | 001 | 6:00 | Калининград Ю. |
| 2 | Держинская | 002 | 6:10 | Калининград Ю. |
| 3 | Киевская | 003 | 6:20 | Калининград Ю. |
| 4 | Западная | 004 | 6:30 | Калининград Ю. |
| 5 | Калининград С. | 005 | 6:40 | Калининград Ю. |
+-----+-----+-----+-----+

>>> select Калининград С.
Список станций пуст.
>>> select Калининград С.
+-----+-----+-----+-----+
| № | Пункт отправления | Номер поезда | Время отправления | Пункт назначения |
+-----+-----+-----+-----+
| 1 | Сельма | 006 | 6:50 | Калининград С. |
| 2 | Кутузово | 007 | 7:00 | Калининград С. |
| 3 | Калининград Ю. | 008 | 7:10 | Калининград С. |
+-----+-----+-----+-----+

>>> exit

Process finished with exit code 0

```

Рисунок 6 – Результаты работы программы

Содержимое файла «Kaliningrad.json» (рис. 7).

```

kaliningrad
1  [
2    {
3      "departure_point": "Айвазовская",
4      "number_train": "001",
5      "time_departure": "6:00",
6      "destination": "Калининград Ю."
7    },
8    {
9      "departure_point": "Держинская",
10     "number_train": "002",
11     "time_departure": "6:10",
12     "destination": "Калининград Ю."
13   },
14   {
15     "departure_point": "Киевская",
16     "number_train": "003",
17     "time_departure": "6:20",
18     "destination": "Калининград Ю."
19   },
20   {
21     "departure_point": "Западная",
22     "number_train": "004",
23     "time_departure": "6:30",
24     "destination": "Калининград Ю."
25   },
26   {
27     "departure_point": "Калининград С.",
28     "number_train": "005",
29     "time_departure": "6:40",
30     "destination": "Калининград Ю."
31   },
32   {
33     "departure_point": "Сельма",
34     "number_train": "006",
35     "time_departure": "6:50",
36     "destination": "Калининград С."
37   },
38 ]

```

Рисунок 7 – Файл «Kaliningrad.json»

Задание повышенной сложности:

Необходимо организовать валидацию загружаемых данных из файла формата JSON с помощью спецификации JSON Schema.

Для того, чтобы производить валидацию файлов необходимо установить виртуальное окружение «jsonschema» (рис. 8).

```
(base) C:\Users\Andrey>conda create --name lab_2
Retrieving notices: ...working... done
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 23.7.4
  latest version: 24.3.0

Please update conda by running

  $ conda update -n base -c defaults conda

Or to minimize the number of packages updated during conda update use

  conda install conda=24.3.0

## Package Plan ##

  environment location: C:\Users\Andrey\anaconda3\envs\lab_2

Proceed ([y]/n)? y

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate lab_2
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

Рисунок 8 – Создание нового виртуального окружения

Установка необходимых пакетов для виртуального окружения (рис. 9).

```
(base) C:\Users\Andrey>conda activate lab_2
(lab_2) C:\Users\Andrey>conda install jsonschema
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

Рисунок 9 – Установка «jsonschema»

Добавление валидации подгружаемых файлов в функцию подгрузки файлов (рис. 10).

```
def load_trains(file_name):  
    """  
    Загрузить все поезда со станциями из файла JSON и валидировать его.  
    """  
    schema = {  
        "type": "array",  
        "items": {  
            "type": "object",  
            "properties": {  
                "departure_point": {"type": "string"},  
                "number_train": {"type": "string"},  
                "time_departure": {"type": "string"},  
                "destination": {"type": "string"},  
            },  
            "required": [  
                "departure_point",  
                "number_train",  
                "time_departure",  
                "destination",  
            ],  
        },  
    }  
  
    # Сменить рабочую директорию, для того, что файл не попал в репозиторий  
    os.chdir("C:\\Users\\Andrey\\Desktop\\Анализ_данных\\2_лаба\\Kaliningrad")  
  
    # Открыть файл с заданным именем для чтения.  
    with open(file_name, "r", encoding="utf-8") as fin:  
        data = json.load(fin) # чтение данных из файла  
  
    try:  
        # Валидация файла  
        validate(instance=data, schema=schema)  
        print("JSON-файл прошел валидацию по заданной схеме.")  
    except ValidationError as e:  
        print(f"Произошла ошибка валидации! {e.message}")  
    return data
```

Рисунок 10 – Функция загрузки JSON файлов

Результаты работы данной функции для подходящего и неподходящего файлов (рис. 11).

```
C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe C:\Users\Andrey\  
>>> load kaliningrad_2  
Произошла ошибка валидации! 'departure_point' is a required property  
>>> load kaliningrad  
JSON-файл прошел валидацию по заданной схеме.  
>>>
```

Рисунок 11 – Валидация загружаемых файлов

Ответы на контрольные вопросы:

1. Для чего используется JSON?

JSON — это стандарт обмена данными. Он позволяет легко сериализовать и десериализовать объекты. Стандарт часто применяют, когда разрабатывают API и веб-приложения.

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. Список (одномерный) — это упорядоченное множество значений. Список заключается в квадратные скобки «[]». Значения разделяются запятыми. 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.

3. Как организована работа со сложными данными в JSON?

JSON позволяет организовать сложные структуры данных, такие как списки и вложенные словари (объекты). В JSON можно хранить разные типы данных, включая числа, строки, логические значения, массивы и объекты. Для организации сложных данных в JSON используются вложенные объекты и списки, позволяя создавать структуры данных любой сложности.

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

JSON5 — это расширение формата данных JSON, разработанное для улучшения читаемости и удобства записи JSON-данных. Отличие JSON5 от обычного JSON включает в себя дополнительные возможности, такие как использование комментариев, разделителей ключей и значений, а также возможность использования одиночных кавычек вместо двойных. JSON5

является более гибким и читаемым форматом для записи данных, но не является стандартом и не поддерживается всеми JSON-парсерами.

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

Для работы с данными в формате JSON5 на Python, можно использовать парсеры, поддерживающие JSON5, такие как `demjson`. Однако, JSON5 не является стандартом, поэтому поддержка может быть ограничена.

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

Для сериализации данных в формат JSON в Python можно использовать модуль `json`. Он предоставляет функции `json.dump()` и `json.dumps()`, а также класс `json.JSONEncoder`, который может быть настроен для сериализации данных в формат JSON.

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

«`json.dump()`» записывает данные в файл. Вы используете его, когда хотите сохранить данные в файле. `json.dumps()` превращает данные в строку. Вы используете его, когда хотите получить данные в виде строки для дальнейшей обработки, но не сохранять их в файле.

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

Для десериализации данных из формата JSON в Python используется модуль `json`, предоставляющий функции `json.load()` и `json.loads()`.

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

Для работы с данными JSON, содержащими кириллицу, важно убедиться, что данные правильно кодируются и декодируются. Обычно это не вызывает проблем, поскольку JSON поддерживает Unicode, включая кириллические символы. Однако, при чтении и записи JSON-файлов, убедитесь, что правильно установлена кодировка (например, `utf-8`) для текстовых данных.

10. Самостоятельно ознакомьтесь со спецификацией JSON Schema? Что такое схема данных?

JSON Schema - это спецификация, которая описывает формат данных JSON и правила их валидации. С помощью JSON Schema можно определить структуру, типы данных и ограничения для JSON-данных. JSON Schema используется для проверки соответствия данных определенным правилам. Это полезно, например, при валидации данных, получаемых из внешних источников. JSON Schema не является частью стандартной библиотеки Python, но существуют библиотеки и инструменты, поддерживающие JSON Schema, которые могут использоваться в Python.

Вывод: в ходе выполнения лабораторной работы были приобретены навыки по работе с JSON-файлами, а также была изучена спецификация «JSON Schema», позволяющая проводить валидацию подгружаемых данных.