

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»

Вариант №18

Выполнил:
Кулешов Олег Иванович
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:
Воронкин Р. А.

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

Тема: “Работа с данными формата JSON в языке Python”

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

Пример 1. Для примера 1 лабораторной работы 2.8 добавьте возможность сохранения списка в файл формата JSON и чтения данных из файла JSON.

Листинг программы:

```
#!/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(
                "No", "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 соответствии 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()

```

```

>>> list
+-----+-----+-----+-----+
|  No  |          Ф.И.О.          |      Должность      |      Год      |
+-----+-----+-----+-----+
|   1  | Кулешов О.И.             | Рядовой              | 2022 |
|   2  | Микаелян В.Р.            | Мэр города           | 2022 |
|   3  | Тимошенко А.Н.           | Охранник сада        | 2012 |
+-----+-----+-----+-----+

```

Рисунок 1. Создал таблицу с данными

```
example_1.py  data.json x
1  [
2      {
3          "name": "Кулешов О.И.",
4          "post": "Рядовой",
5          "year": 2022
6      },
7      {
8          "name": "Микаелян В.Р.",
9          "post": "Мэр города",
10         "year": 2022
11     },
12     {
13         "name": "Тимошенко А.Н.",
14         "post": "Охранник сада",
15         "year": 2012
16     }
17 ]
```

Рисунок 2. С помощью команды “**save data.json**” преобразовал данные в формат JSON

```
C:\Users\User\.conda\envs\Data_Analysis_2\python.exe C:\Users\User\PycharmProjects\Data_Analysis_2\example_1.py
>>> list
Список работников пуст.
>>> load data.json
>>> list
+-----+-----+-----+-----+
| No |      Ф.И.О.      |      Должность      |      Год      |
+-----+-----+-----+-----+
|  1 | Кулешов О.И.    |      Рядовой      |      2022     |
|  2 | Микаелян В.Р.   |      Мэр города    |      2022     |
|  3 | Тимошенко А.Н.  |      Охранник сада |      2012     |
+-----+-----+-----+-----+
>>>
```

Рисунок 3. С помощью команды “**load data.json**” выгрузил данные из файла data.json

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.

Листинг программы:

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

import sys
import json

# Путь к файлу для сохранения данных о поездах.
```

```

file_path = 'data/trains.json'

def save_data(data):
    with open(file_path, 'w') as file:
        json.dump(data, file, indent=4)

def load_data():
    try:
        with open(file_path, 'r') as file:
            data = json.load(file)
            return data
    except FileNotFoundError:
        return []

if __name__ == '__main__':
    # Загрузить данные из файла при запуске программы.
    trains = load_data()

    # Организация бесконечного цикла запроса команд.
    while True:
        # Запросить команду из терминала.
        command = input('>>> ').lower()

        # Выполнить действие в соответствии с командой.
        if command == 'exit':
            # Сохранить данные перед выходом.
            save_data(trains)
            break
        elif command == 'add':
            # Запросить данные о поезде.
            destination = input('Название пункта назначения? ')
            number = input('Номер поезда? ')
            departure_time = input('Время отправления? ')

            # Создать словарь.
            train = {
                'destination': destination,
                'number': number,
                'departure_time': departure_time
            }

            # Добавить словарь в список.
            trains.append(train)

            # Отсортировать список по времени отправления поезда.
            trains.sort(key=lambda item: item.get('departure_time', ''))

        elif command == 'list':
            # Вывести информацию о поездах.
            for idx, train in enumerate(trains, 1):
                print(f'{idx}. Пункт назначения: {train["destination"]},
Номер поезда: {train["number"]}, Время отправления:
{train["departure_time"]}')

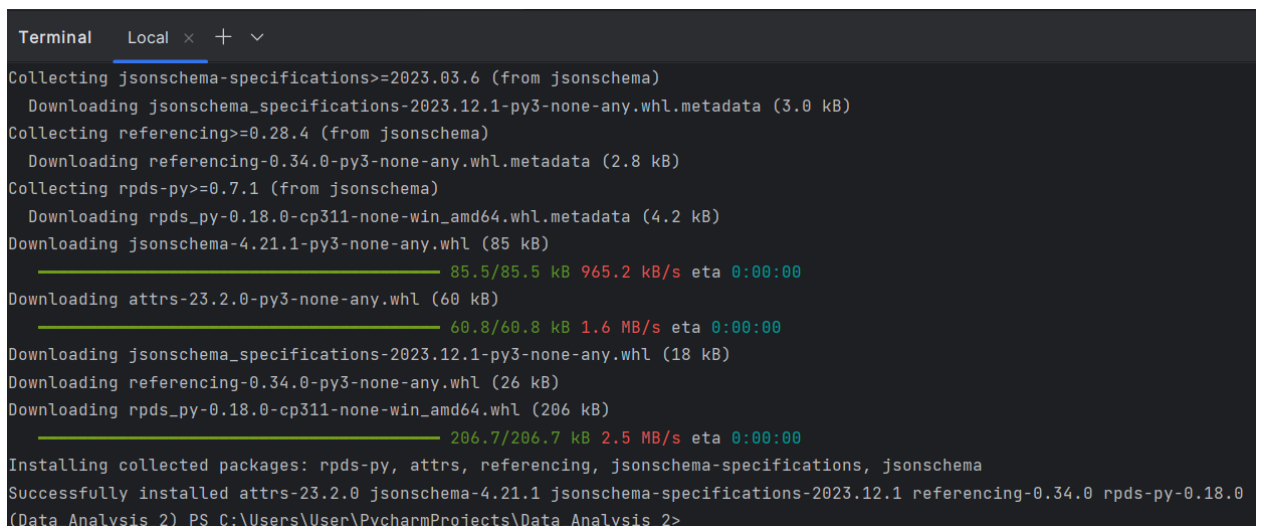
            # Другие команды остаются без изменений.

        elif command == 'help':
            # Вывести справку о работе с программой.
            print('Список команд:\n')
            print('add - добавить информацию о поезде;')
            print('list - вывести список всех поездов;')
            print('select <пункт_назначения> - запросить информацию о поездах
в заданном пункте назначения;')
            print('exit - завершить работу с программой.')

```

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

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.



```
Terminal Local x + v
Collecting jsonschema-specifications>=2023.03.6 (from jsonschema)
  Downloading jsonschema_specifications-2023.12.1-py3-none-any.whl.metadata (3.0 kB)
Collecting referencing>=0.28.4 (from jsonschema)
  Downloading referencing-0.34.0-py3-none-any.whl.metadata (2.8 kB)
Collecting rpds-py>=0.7.1 (from jsonschema)
  Downloading rpds_py-0.18.0-cp311-none-win_amd64.whl.metadata (4.2 kB)
Downloading jsonschema-4.21.1-py3-none-any.whl (85 kB)
 85.5/85.5 kB 965.2 kB/s eta 0:00:00
Downloading attrs-23.2.0-py3-none-any.whl (60 kB)
 60.8/60.8 kB 1.6 MB/s eta 0:00:00
Downloading jsonschema_specifications-2023.12.1-py3-none-any.whl (18 kB)
Downloading referencing-0.34.0-py3-none-any.whl (26 kB)
Downloading rpds_py-0.18.0-cp311-none-win_amd64.whl (206 kB)
 206.7/206.7 kB 2.5 MB/s eta 0:00:00
Installing collected packages: rpds-py, attrs, referencing, jsonschema-specifications, jsonschema
Successfully installed attrs-23.2.0 jsonschema-4.21.1 jsonschema-specifications-2023.12.1 referencing-0.34.0 rpds-py-0.18.0
(Data_Analysis_2) PS C:\Users\User\PycharmProjects\Data_Analysis_2>
```

Рисунок 3. Установил библиотеку `jsonschema` (`pip install jsonschema`)

Листинг программы:

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

import sys
import json
import jsonschema
from jsonschema import validate

def save_data(filename):
    with open(filename, 'w') as file:
        json.dump(trains, file, indent=4)
    print(f'Данные сохранены в файл {filename}')

def load_data(filename):
    global trains
```

```

try:
    with open(filename, 'r') as file:
        data = json.load(file)
        # Проверка на валидность загруженных данных
        for item in data:
            validate(instance=item, schema=train_schema)
        trains = data
        print(f'Данные успешно загружены из файла {filename}')
except Exception as e:
    print(f'Ошибка загрузки данных из файла: {e}', file=sys.stderr)

# JSON схема для данных о поездах
train_schema = {
    "type": "object",
    "properties": {
        "destination": {"type": "string", "pattern": "^[A-Za-z ]+$"},
        "number": {"type": "string", "pattern": "^[0-9]+$"},
        "departure_time": {"type": "string", "pattern": "^(([01]?[0-9]|2[0-3]):[0-5][0-9])$"}
    },
    "required": ["destination", "number", "departure_time"]
}

if __name__ == '__main__':
    trains = []
    # Организация бесконечного цикла запроса команд.
    while True:
        # Запросить команду из терминала.
        command = input('>>> ').lower()

        # Выполнить действие в соответствии с командой.
        if command == 'exit':
            break
        elif command == 'add':
            # Запросить данные о поезде.
            destination = input('Название пункта назначения? ')
            number = input('Номер поезда? ')
            departure_time = input('Время отправления? ')

            # Создать словарь.
            train = {
                'destination': destination,
                'number': number,
                'departure_time': departure_time
            }

            # Валидация данных с использованием JSON Schema.
            try:
                jsonschema.validate(train, train_schema)
            except jsonschema.exceptions.ValidationError as e:
                print(f'Ошибка валидации данных: {e}')
                continue

            # Добавить словарь в список.
            trains.append(train)

            # Отсортировать список по времени отправления поезда.
            trains.sort(key=lambda item: item.get('departure_time', ''))

        elif command == 'list':
            # Заголовок таблицы.
            line = '+-{}-+{}-+{}-+'.format(

```



```

        '-' * 20,
        '-' * 15,
        '-' * 20
    )
    print(line)
    print(
        '| {:^20} | {:^15} | {:^20} |'.format(
            "Пункт назначения",
            "Номер поезда",
            "Время отправления"
        )
    )
    print(line)

    # Вывести информацию о поездах.
    for idx, train in enumerate(trains, 1):
        print(
            '| {:<20} | {:^15} | {:^20} |'.format(
                train.get('destination', ''),
                train.get('number', ''),
                train.get('departure_time', '')
            )
        )
    print(line)

elif command.startswith('select '):
    # Получить название пункта назначения из команды.
    parts = command.split(' ', maxsplit=1)
    destination = parts[1]

    # Поиск поездов с заданным пунктом назначения.
    selected_trains = [train for train in trains if
train['destination'] == destination]

    if selected_trains:
        # Вывести информацию о найденных поездах в виде таблицы.
        line = '+-{}-+{}-+{}-+'.format(
            '-' * 20,
            '-' * 15,
            '-' * 20
        )
        print(line)
        print(
            '| {:^20} | {:^15} | {:^20} |'.format(
                "Пункт назначения",
                "Номер поезда",
                "Время отправления"
            )
        )
        print(line)
        for train in selected_trains:
            print(
                '| {:<20} | {:^15} | {:^20} |'.format(
                    train.get('destination', ''),
                    train.get('number', ''),
                    train.get('departure_time', '')
                )
            )
        print(line)
    else:
        print(f'Поездов в пункт "{destination}" не найдено')

elif command == 'help':

```

```

# Вывести справку о работе с программой.
print('Список команд:\n')
print('add - добавить информацию о поезде;')
print('list - вывести список всех поездов;')
print('select <пункт_назначения> - запросить информацию о поездах
в заданном пункте назначения;')
print('exit - завершить работу с программой.')
elif command.startswith('save'):
    filename = command.split(' ')[1]
    save_data(filename)

elif command.startswith('load'):
    filename = command.split(' ')[1]
    load_data(filename)
else:
    print(f'Неизвестная команда "{command}"!', file=sys.stderr)

```

В этом примере я добавил JSON Schema для валидации данных о поездах и использовал библиотеку `jsonschema` для выполнения валидации. После запроса данных о поезде программа проверяет их на соответствие схеме JSON Schema. Если данные не проходят валидацию, выводится сообщение об ошибке.

```

>>> list
+-----+-----+-----+
| Пункт назначения | Номер поезда | Время отправления |
+-----+-----+-----+
| Moscow          | 777          | 12:30              |
| Stavropol       | 126          | 14:50              |
| Spb              | 123          | 9:50               |
+-----+-----+-----+
>>> |

```

Рисунок 4. Ввёл данные в таблицу

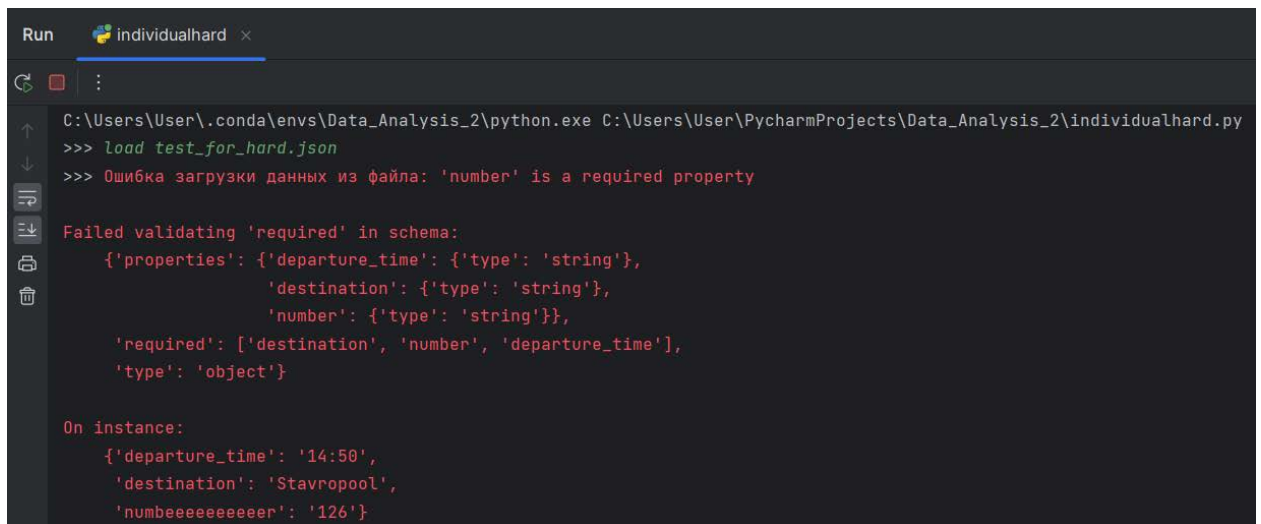
```
[
  {
    "destination": "Moscow",
    "number": "777",
    "departure_time": "12:30"
  },
  {
    "destination": "Stavropool",
    "number": "126",
    "departure_time": "14:50"
  },
  {
    "destination": "Spb",
    "number": "123",
    "departure_time": "9:50"
  }
]
```

Рисунок 5. Получил файл json



```
1 [
2   {
3     "destination": "Moscow",
4     "number": "777",
5     "departure_time": "12:30"
6   },
7   {
8     "destination": "Stavropool",
9     "numbeeeeeeeeeer": "126",
10    "departure_time": "14:50"
11  },
12  {
13    "destination": "Spb",
14    "number": "123",
15    "departure_time": "9:50"
16  }
17 ]
```

Рисунок 6. Создал заранее неправильную JSON-схему



```
Run individualhard x
C:\Users\User\.conda\envs\Data_Analysis_2\python.exe C:\Users\User\PycharmProjects\Data_Analysis_2\individualhard.py
>>> load test_for_hard.json
>>> Ошибка загрузки данных из файла: 'number' is a required property

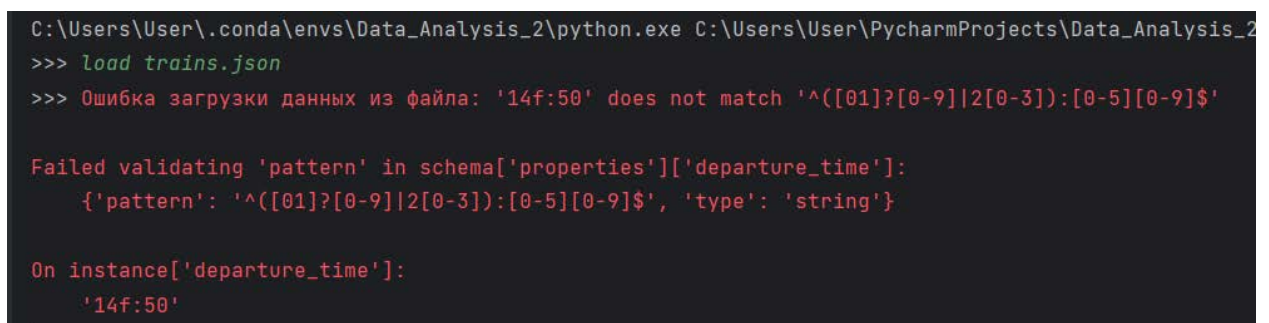
Failed validating 'required' in schema:
  {'properties': {'departure_time': {'type': 'string'},
                  'destination': {'type': 'string'},
                  'number': {'type': 'string'}},
   'required': ['destination', 'number', 'departure_time'],
   'type': 'object'}

On instance:
  {'departure_time': '14:50',
   'destination': 'Stavropool',
   'numbeeeeeeeeeer': '126'}
```

Рисунок 7. Как следствие, получил ошибку валидации

Также, посчитал полезным, чтобы в проверке валидации учитывалось содержимое:

- "destination" должно содержать только буквы и пробелы (используется регулярное выражение $^[A-Za-z]+\$$),
- "number" должно содержать только цифры (используется регулярное выражение $^[0-9]+\$$),
- "departure_time" должно быть временем в 24-часовом формате (используется регулярное выражение $^([01]?[0-9]|2[0-3]):[0-5][0-9]\$$).



```
C:\Users\User\.conda\envs\Data_Analysis_2\python.exe C:\Users\User\PycharmProjects\Data_Analysis_2
>>> load trains.json
>>> Ошибка загрузки данных из файла: '14f:50' does not match '^([01]?[0-9]|2[0-3]):[0-5][0-9]$\

Failed validating 'pattern' in schema['properties']['departure_time']:
  {'pattern': '^([01]?[0-9]|2[0-3]):[0-5][0-9]$', 'type': 'string'}

On instance['departure_time']:
  '14f:50'
```

Рисунок 8. Можно заметить, что в расписании времени не соблюдается форма записи “часы:минуты”, так как в часы добавлена буква “f”.

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

Ответы на контрольные вопросы

1. JSON (JavaScript Object Notation) используется для обмена данными между приложениями. Он широко используется в веб-разработке для передачи структурированных данных между клиентом и сервером.
2. В JSON используются следующие типы значений: строки, числа, логические значения (true/false), массивы, объекты, null.
3. Для работы со сложными данными в JSON можно использовать вложенные объекты и массивы. Это позволяет представлять структурированные данные различных уровней сложности.
4. Формат данных JSON5 является расширением стандартного JSON и добавляет некоторые удобные возможности, такие как поддержка комментариев и различные способы записи чисел. Отличие от стандартного JSON заключается в дополнительных возможностях и удобствах при работе с данными.
5. Для работы с данными в формате JSON5 в Python можно использовать сторонние библиотеки, такие как json5 или json5decoder.
6. В языке Python для сериализации данных в формате JSON используется модуль json. С помощью функции json.dumps() можно преобразовать данные Python в формат JSON.
7. Функция json.dump() используется для записи данных в файл в формате JSON, а функция json.dumps() преобразует данные в формат JSON в строку.
8. Для десериализации данных из формата JSON в Python используется функция json.loads(), которая преобразует строку JSON в объект Python.
9. Для работы с данными формата JSON, содержащими кириллицу, необходимо учитывать кодировку. При чтении и записи данных в файлы следует указывать соответствующую кодировку (например, "utf-8"). При работе с данными в памяти Python обычно автоматически обрабатывает Unicode строки, поэтому проблем с кириллицей не должно возникать.