Лабораторная работа 2.16 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.*

Ход работы

JSON (англ. *JavaScript Object Notation*, обычно произносится как /ˈdʒeɪsən/ 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. Это информативное руководство поможет вам быстрее разобраться с данными, которые вы можете использовать с JSON и основной структурой с синтаксисом этого же формата.

Синтаксис и структура данных формата JSON

Объект JSON это формат данных — *ключ-значение*, который обычно рендерится в фигурных скобках. Когда вы работаете с JSON, то вы скорее всего видите JSON объекты в `.json` файле, но они также могут быть и как JSON объект или строка уже в контексте самой программы.

Вот так выглядит JSON объект:

```
{
    "first_name" : "Sammy",
    "last_name" : "Shark",
    "location" : "Ocean",
    "online" : true,
    "followers" : 987
}
```

Хоть это и короткий пример, и JSON мог бы быть гораздо больше, он показывает то, что этот формат указывается двумя фигурными скобками, которые выглядят так `{ }`, а данные в формате ключ-значние уже находятся между ними. Большинство используемых данных в JSON, заключаются в JSON объекты.

Пары ключ-значние разделены двоеточием, как например тут `"key" : "value"`. Каждая пара значений разделена двоеточием, таким образом середина JSON выглядит так: `"key" : "value"`, `"key" : "value"`, `"key" : "value"`. В нашем примере выше, первая пара ключевых значений это `"first_name" : "Sammy"`.

Ключи в JSON находятся с левой стороны от двоеточия. Их нужно оборачивать в скобки, как с `"key"` и это может быть любая строка. В каждом объекте, ключи должны быть уникальными. Такие ключевые строки могут содержать пробелы, как в `"first_name"`, но такой подход может усложнить получение доступа к ним во время процесса разработки, так что лучшим вариантом в таких случаях будет использование нижнего подчеркивания, как сделано тут `"first_name"`.

JSON значения находятся с правой стороны от двоеточия. Если быть точным, то им нужно быть одним из шести типов данных: строкой, числом, объектом, массивом, булевым значением или `null`.

Каждый тип данных, который передается как значения в JSON будет поддерживать свой собственный синтаксис, так что строки будут в кавычках, а цифры нет.

Хоть в `.json` файлах мы обычно видим формат нескольких строк, JSON также может быть написан в одну сплошную строку.

```
{ "first_name" : "Sammy", "last_name": "Shark",  "online" : true, }
```

Такой подход наиболее распространен в файлах других форматов или при работе с JSON строкой.

Работа с JSON в многострочном формате зачастую делает его более читабельным, особенно когда вы пытаетесь справиться с большим набором данных. Так как JSON игнорирует пробелы между своими элементами, вы можете разделить их теми же самыми пробелами, чтобы сделать данные более читабельными:

```
{
    "first_name"  :  "Sammy",
    "last_name"   :  "Shark",
    "online"      :  true
}
```

Очень важно помнить то, что хоть они и похожи визуально, объекты JSON не имеют тот же формат, как объекты JavaScript, так образом, хоть вы и можете использовать функции внутри JavaScript объектов, вы не можете использовать их как значения в JSON. Самым важным свойством JSON является то, что он может без труда передаваться между языками программирования в формате, который понимают практически все из них. JavaScript объекты могут работать только напрямую через JavaScript.

Типы значений JSON

Как было показано ранее JSON-текст представляет собой (в закодированном виде) одну из двух структур:

- Набор пар *ключ: значение*. В различных языках это реализовано как запись, структура, словарь, хеш-таблица, список с ключом или ассоциативный массив. Ключом может быть только строка (регистрозависимость не регулируется стандартом, это остаётся на усмотрение программного обеспечения. Как правило, регистр учитывается программами — имена с буквами в разных регистрах считаются разными, значением — любая форма. Повторяющиеся имена ключей допустимы, но не рекомендуются стандартом; обработка таких ситуаций происходит на усмотрение программного обеспечения, возможные варианты — учитывать только первый такой ключ, учитывать только последний такой ключ, генерировать ошибку.

- Упорядоченный набор *значений*. Во многих языках это реализовано как массив, вектор, список или последовательность.

Структуры данных, используемые JSON, поддерживаются любым современным языком программирования, что и позволяет применять JSON для обмена данными между различными языками программирования и программными системами.

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.

Строка очень похожа на литерал одноимённого типа данных в языке JavaScript. *Число* тоже очень похоже на JavaScript-число, за исключением того, что используется только десятичный формат (с точкой в качестве разделителя). Пробелы могут быть вставлены между любыми двумя синтаксическими элементами.

Строка — заданная последовательность из нуля и больше символов Юникода, заключенная в две двойные кавычки. Этот пример показывает, что **Том** обозначает строку, поскольку является набором символов внутри двойных кавычек.

```
{"firstName": "Том"}
```

Число в JSON должно быть **целым** или с **плавающей запятой**, например:

```
{"age": 30}
```

Для данных в формате JSON допустим и **булев** тип. Вы можете использовать **true** или **false** в качестве значения, как показано ниже:

```
{"married": false}
```

Значение **null** показывает отсутствие информации.

```
{"bloodType": null}
```

Массив — это упорядоченная коллекция значений. Он заключен в **квадратные скобки []**, а каждое значение внутри разделено запятой. Значение массива может содержать объекты JSON, что означает, что он использует ту же концепцию пар ключей/значений. Например:

```
{
    "students":[
        {"firstName":"Tom", "lastName":"Jackson"},
        {"firstName":"Linda", "lastName":"Garner"},
        {"firstName":"Adam", "lastName":"Cooper"}
    ]
}
```

Информация в квадратных скобках — это массив, в котором есть три объекта.

Объект содержит ключ и значение. После каждого ключа стоит двоеточие, а после каждого значения — запятая, которая также различает каждый объект. Оба они находятся внутри кавычек. Объект как значение должен подчиняться тому же правилу, что и объект. Например:

```
{
    “employees”: {"firstName": "Tom", "lastName": "Jackson"}
}
```

Здесь **employees** — ключ, а всё, что находится внутри фигурных скобок — объект.

Следующий пример показывает JSON-представление данных об объекте, описывающем человека. В данных присутствуют *строковые* поля имени и фамилии, информация об адресе и массив, содержащий список телефонов. Как видно из примера, *значение* может представлять собой вложенную структуру.

```
{
    "firstName": "Иван",
    "lastName": "Иванов",
    "address": {
        "streetAddress": "Московское ш., 101, кв.101",
        "city": "Ленинград",
        "postalCode": 101101
    },
    "phoneNumbers": [
        "812 123-1234",
        "916 123-4567"
    ]
}
```

Обратите внимание на пару `"postalCode": 101101`. В качестве значений в JSON могут быть использованы как число, так и строка. Поэтому запись `"postalCode": "101101"` содержит строку, а `"postalCode": 101101` - уже числовое значение.

Работа с комплексными типами 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.

Вложенные объекты

В файле `users.json`, для каждого из четырех пользователей (`"sammy"`, `"jesse"`, `"drew"`, `"jamie"`) есть вложенный JSON объект, передающий значения для каждого из пользователей, со своими собственными вложенными ключами `"username"` и `"location"`. Первый вложенный JSON объект подсвечен ниже:

```
{
    "sammy" : {
        "username"  : "SammyShark",
        "location"  : "Indian Ocean",
        "online"    : true,
        "followers" : 987
    },
    "jesse" : {
        "username"  : "JesseOctopus",
        "location"  : "Pacific Ocean",
        "online"    : false,
        "followers" : 432
    },
    "drew" : {
        "username"  : "DrewSquid",
        "location"  : "Atlantic Ocean",
        "online"    : false,
        "followers" : 321
    },
    "jamie" : {
        "username"  : "JamieMantisShrimp",
        "location"  : "Pacific Ocean",
        "online"    : true,
        "followers" : 654
    }
}
```

В примере выше, фигурные скобки везде используются для формирования вложенного JSON объекта с ассоциированными именами пользователей и данными локаций для каждого из них. Как и с любым другим значением, используя объекты, двоеточие используется для разделения элементов.

Вложенные массивы

Данные также могут быть вложены в формате JSON, используя JavaScript массивы, которые передаются как значения. JavaScript использует квадратные скобки `[ ]` для формирования массива. Массивы по своей сути — это упорядоченные коллекции и могут включать в себя значения совершенно разных типов данных.

Мы можем использовать массив при работе с большим количеством данных, которые могут быть легко сгруппированны вместе, как например, если есть несколько разных сайтов и профайлов в социальных сетях ассоциированных с одним пользователем.

В этом примере, первый вложенный массив подсвечен:

```
{
    "first_name" : "Sammy",
    "last_name" : "Shark",
    "location" : "Ocean",
```

```
    "websites" : [
        {
            "description" : "work",
            "URL" : "https://www.digitalocean.com/"
        },
        {
            "desciption" : "tutorials",
            "URL" : "https://www.digitalocean.com/community/tutorials"
        }
    ],
    "social_media" : [
        {
            "description" : "twitter",
            "link" : "https://twitter.com/digitalocean"
        },
        {
            "description" : "facebook",
            "link" : "https://www.facebook.com/DigitalOceanCloudHosting"
        },
        {
            "description" : "github",
            "link" : "https://github.com/digitalocean"
        }
    ]
}
```

Ключи `"websites"` и `"social_media"` используют массив для вложения информации о сайтах пользователя и профайлов в социальных сетях. Мы знаем, что это массивы — из-за квадратных скобок.

Использование вложенности в нашем JSON формате позволяет нам работать с наиболее сложными и иерархичными данными.

Обработка данных формата JSON в языке Python

1. **Сериализация** данных в формат 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.

2. **Десериализация** данных из формата JSON:

```
json.load()  # прочитать json из файла и конвертировать в python объект
json.loads() # тоже самое, но из строки с json  (s на конце от string/строка)
```

Обе эти функции принимают следующие аргументы:

- `object_hook` - опциональная функция, которая применяется к результату декодирования объекта (`dict`). Использоваться 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.

Если не удастся десериализовать JSON, будет возбуждено исключение `ValueError`.

Запись списка или словаря в файл:

```
import json

my_list = ['foo', 'bar']


# вариант 1
contents = json.dumps(my_list)
with open("foo.txt", "w", encoding="utf-8") as f:
   f.write(contents)

# вариант 2
with open("foo.txt", "w", encoding="utf-8") as f:
   json.dump(my_list, f)
```

Чтение списка или словаря из файла:

```
import json

#вариант 1
with open("foo.txt", "r") as f:
   contents = f.read()
my_list = json.loads(contents)

#вариант 2
with open("foo.txt", "r") as f:
   my_list = json.load(f)
```

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

Решение: Введем следующие команды для работы с файлом формата JSON в интерактивном режиме:

- *load* - загрузить данные из файла, имя файла должно отделяться от команды *load* пробелом. Например:

```
>>> load data.json
```

- *save* - сохранить сделанные изменения в файл, имя файла должно отделяться от команды *save* пробелом. Например:

```
>>> save data.json
```

Напишем программу для решения поставленной задачи.

```
#!/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 кирилицы 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()
```

Аппаратура и материалы

1. Компьютерный класс общего назначения с конфигурацией ПК не хуже рекомендованной для ОС Windows 10 с подключением к глобальной сети Интернет.
2. Операционная система Windows 10.
3. Система контроля версий Git.
4. Браузер для доступа к web-сервису GitHub, рекомендован к использованию Google Chrome.
5. Дистрибутив языка программирования Python, включающий набор популярных библиотек Anaconda.
6. Интегрированная среда разработки PyCharm Community Edition.

Указания по технике безопасности

При работе на ЭВМ без разрешения руководителя занятия запрещается:

- подавать (снимать) напряжение на ПЭВМ и электрические розетки с распределительного щита;
- включать и выключать блоки питания ПЭВМ и мониторы;

- извлекать ПЭВМ из защитного кожуха;
- устранять неисправности, возникшие в ходе выполнения лабораторной работы.

Методика и порядок выполнения работы

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.
3. Выполните клонирование созданного репозитория.
4. Дополните файл `.gitignore` необходимыми правилами для работы с IDE PyCharm.
5. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.
6. Создайте проект PyCharm в папке репозитория.
7. Проработайте примеры лабораторной работы. Создайте для них отдельные модули языка Python. Зафиксируйте изменения в репозитории.
8. Приведите в отчете скриншоты результатов выполнения примера при различных исходных данных вводимых с клавиатуры.
9. Зафиксируйте сделанные изменения в репозитории.
10. Приведите в отчете скриншоты работы программ решения индивидуальных заданий.
11. Зафиксируйте сделанные изменения в репозитории.
12. Добавьте отчет по лабораторной работе в *формате PDF* в папку *doc* репозитория. Зафиксируйте изменения.
13. Выполните слияние ветки для разработки с веткой *master/main*.
14. Отправьте сделанные изменения на сервер GitHub.
15. Отправьте адрес репозитория GitHub на электронный адрес преподавателя.

Индивидуальные задания

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.*

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.

Содержание отчета и его форма

Отчет по лабораторной работе оформляется письменно в рабочей тетради, должен содержать ответы на контрольные вопросы, ссылку на репозиторий с которым выполнялась работа, скриншоты IDE PyCharm, скриншоты результатов работы программ.

Вопросы для защиты работы

1. Для чего используется JSON?
2. Какие типы значений используются в JSON?
3. Как организована работа со сложными данными в JSON?
4. Самостоятельно ознакомьтесь с форматом данных JSON5? В чем отличие этого формата от формата данных JSON?
5. Какие средства языка программирования Python могут быть использованы для работы с данными в формате JSON5?
6. Какие средства предоставляет язык Python для сериализации данных в формате JSON?
7. В чем отличие функций json.dump() и json.dumps()?
8. Какие средства предоставляет язык Python для десериализации данных из формата JSON?
9. Какие средства необходимо использовать для работы с данными формата JSON, содержащими кирилицу?
10. Самостоятельно ознакомьтесь со спецификацией JSON Schema? Что такое схема данных? Приведите схему данных для примера 1.