Формат json

Мы уже знаем, что в файле можно хранить текстовую информацию, информацию в виде байтов и даже некоторым образом структурированную, как это делается в файлах с разделителями. Но для работы с сохраненной информацией средствами языка Python ее нужно преобразовать из того вида, в котором она хранится в файле, в более сложные объекты: списки, словари и другие. Конечно, у нас есть мощный метод `split()` и специальные библиотеки, но хорошо бы иметь возможность сразу хранить готовые объекты, а не заниматься преобразованием каждый раз при чтении файла. И такую возможность нам предоставляет формат json.

JSON

JSON (JavaScript Object Notation) — текстовый формат для хранения и обмена данными. Этот формат очень похож на словарь в Python и предназначен для хранения объектов языка.

Он очень удобен и поэтому широко используется для работы с API (Application Programming Interface, программный интерфейс приложения, с помощью которого одна программа/приложение может взаимодействовать с другой).

Например, погодный сервис в интернете по ссылке

https://samples.openweathermap.org/data/2.5/weather?id=2172797&appid=439d4b804bc8187953eb36d2a8c26a02

возвращает ответ в виде json:


```
{
  "coord": {
    "lon": 145.77,
    "lat": -16.92
  },
  "weather": [
    {
      "id": 802,
      "main": "Clouds",
      "description": "scattered clouds",
      "icon": "03n"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 300.15,
    "pressure": 1007,
    "humidity": 74,
    "temp_min": 300.15,
    "temp_max": 300.15
```

```
  },
  "visibility": 10000,
  "wind": {
    "speed": 3.6,
    "deg": 160
  },
  "clouds": {
    "all": 40
  },
  "dt": 1485790200,
  "sys": {
    "type": 1,
    "id": 8166,
    "message": 0.2064,
    "country": "AU",
    "sunrise": 1485720272,
    "sunset": 1485766550
  },
  "id": 2172797,
  "name": "Cairns",
  "cod": 200
}
```

Теперь у нас есть готовый объект языка, и с ним можно работать из программы как с объектом, в данном случае — со словарем, но это могут быть и список, кортеж или некоторые другие объекты.

Модуль json

Так же, как и в случае csv-документов, в Python есть встроенный модуль для работы с json. Чтобы им воспользоваться, в первой строке программы нужно написать

```
import json
```

Чтение json

Откроем в IDE файл cats_json.json. Мы видим, что там в виде словаря записана некоторая информация о питомце:

```
{
  "name": "Barsik",
  "age": 7,
  "meals": [
    "Wiskas",
    "Royal Canin",
    "Purina",
    "Hills",
```

```
    "Brit Care"
  ]
}
```

Обратите внимание: в формате JSON используются только двойные кавычки.

Для чтения данных в модуле json есть два метода:

- `json.load()` — считывает целиком файл в формате JSON и возвращает объекты Python
- `json.loads()` — считывает строку в формате JSON и возвращает объекты Python

Напишем код, читающий файл и выводящий содержимое полученного словаря:

```
import json

with open('cats_json.json') as cat_file:
    data = json.load(cat_file)
for key, value in data.items():
    if type(value) == list:
        print(f'{key}: {", ".join(value)}')
    else:
        print(f'{key}: {value}')
name: Barsik
age: 7
meals: Wiskas, Royal Canin, Purina, Hills, Brit Care
```

В отличие от метода `json.load(filename)`, метод `json.loads(string)` считывает строку и возвращает объект json, если его возможно получить из переданной строки.

Создайте в IDE новый файл cats_2.json, в который кроме Барсика добавьте еще одного кота по вашему усмотрению. Поскольку теперь в файле у нас уже два словаря, их надо объединить в одну структуру, пусть это будет список. Получим что-то вроде такого содержимого:

```
[
  {
    "name": "Barsik",
    "age": 7,
    "meals": [
      "Wiskas",
      "Royal Canin",
      "Purina",
      "Hills",
      "Brit Care"
    ]
```

```
  },
  {
    "name": "Mursik",
    "age": 3,
    "meals": [
      "Purina",
      "Hills",
      "Brit Care"
    ]
  }
]
```

Чтобы воспользоваться методом `loads()`, нужно сначала считать весь файл в строку, а затем передать ее методу для преобразования в json-объект.

```
import json

with open('cats_2.json') as cat_file:
    f = cat_file.read()
    data = json.loads(f)
    for item in data:
        for key, value in item.items():
            if type(value) == list:
                print(f'{key}: {", ".join(value)}')
            else:
                print(f'{key}: {value}')
        print()
```

И в том, и в другом случае мы получили объект языка Python, словарь или список, с которым можно сразу работать средствами языка.

Сериализация и десериализация

Мы производили преобразования между объектами языка Python и json-объектами. Такие преобразования называются **сериализацией** (кодирование в json-формат, то есть в поток байт) и **десериализацией** (декодирование в объект языка).

В таблице представлено соответствие между данными в Python и в JSON:

Python	JSON
dict	object
list, tuple	array
str	string
int, float	number
True	true
False	false

None	null

При обратном преобразовании массив array преобразуется в список.

Запись в файл

Для записи информации в файл в json также есть два метода:

- `json.dump()` — метод записывает объект Python в файл в формате JSON
- `json.dumps()` — метод преобразует объект Python в строку в формате JSON

Давайте создадим словарь, в котором коту добавим хозяев, а затем полученную информацию сохраним в файле:

```
import json

cats_dict = {
    'name': 'Pushin',
    'age': 1,
    'meals': [
        'Purina', 'Cat Chow', 'Hills'
    ],
    'owners': [
        {
            'first_name': 'Bill',
            'last_name': 'Gates'
        },
        {
            'first_name': 'Melinda',
            'last_name': 'Gates'
        }
    ]
}

with open('cats_3.json', 'w') as cat_file:
    json.dump(cats_dict, cat_file)
```

Метод `dumps()` используется, когда надо просто преобразовать объект в json-формат, необязательно для записи в файл. Это нужно, например, при сборке url-адреса в строку.

Посмотрим на примере:

```
import json

weekdays = {i: day
            for i, day in enumerate(['Sunday',
                                     'Monday',
                                     'Tuesday',
                                     'Wednesday',
```

```
                                        'Thursday',
                                        'Friday',
                                        'Saturday'
                                        ])}
data = json.dumps(weekdays)
print(data)
print(type(data))
```

Получим строку в формате json:

```
{"0": "Sunday", "1": "Monday", "2": "Tuesday", "3": "Wednesday",
"4": "Thursday", "5": "Friday", "6": "Saturday"}
<class 'str'>
```

Пояснение

Встроенная функция `enumerate(sequence[, start=0])` применяется для итерируемых коллекций (списки, кортежи, словари и т. д.), чтобы получать не только само значение элемента коллекции, но и его индекс. У функции есть необязательный параметр start, позволяющий изменять начальное значение счетчика.

Дополнительные параметры методов записи

Методы записи имеют несколько необязательных параметров, которые можно менять для более удобного чтения человеком. Рассмотрим некоторые из них.

ensure_ascii. В случае, если `ensure_ascii=True` (по умолчанию), все не-ASCII символы в выводе будут экранированы последовательностями \uXXXX. Если `ensure_ascii=False`, эти символы будут записаны как есть. Это важно, если в содержимом есть русские буквы.

Запишем в файл cats.json содержимое списка без флага:

```
with open('cats.json', 'w') as file:
    json.dump(data, file)
```

В файле:

```
[
    {
        "name": "\u0411\u0430\u0440\u0441\u0438\u043a",
        "age": 7,
        "toys": [
            "\u041c\u044b\u0448\u043a\u0430",
            "\u041f\u0440\u0443\u0442\u0438\u043a",
            "\u0411\u0430\u043d\u0442\u0438\u043a",
            "\u0421\u0432\u043e\u0439
```

```
\u0445\u0432\u043e\u0441\u0442"
        ]
    },
    {
        "name": "\u041c\u0443\u0440\u0437\u0438\u043a",
        "age": 3,
        "toys": [
            "\u0420\u0443\u043a\u0430
\u0445\u043e\u0437\u044f\u0439\u043a\u0438",
            "\u0428\u043d\u0443\u0440 \u043e\u0442
\u0442\u0435\u043b\u0435\u0432\u0438\u0437\u043e\u0440\u0430",
            "\u041e\u0431\u043e\u0438 \u043d\u0430
\u0441\u0442\u0435\u043d\u0435"
        ]
    }
]
```

А теперь — с флагом:

```
with open('cats.json', 'w') as file:
    json.dump(data, file, ensure_ascii=False)
```

В файле:

```
[
    {
        "name": "Барсик",
        "age": 7,
        "toys": [
            "Мышка",
            "Прутик",
            "Бантик",
            "Свой хвост"
        ]
    },
    {
        "name": "Мурзик",
        "age": 3,
        "toys": [
            "Рука хозяйки",
            "Шнур от телевизора",
            "Обои на стене"
        ]
    }
]
```

indent. Отступ indent нужен для удобного для чтения человеком представления информации в объекте JSON. По умолчанию имеет значение None для более компактного представления, то есть без отступов. Также отступов не будет, если значение indent равно 0, отрицательному числу или пустой строке. Если indent — строка (например, \t), эта строка используется в качестве отступа.

Пример со значением `indent=2`:

```
with open('cats.json', 'w') as file:
    json.dump(data, file, ensure_ascii=False, indent=2)
```

Результат:

```
[
  {
    "name": "Барсик",
    "age": 7,
    "toys": [
      "Мышка",
      "Прутик",
      "Бантик",
      "Свой хвост"
    ]
  },
  {
    "name": "Мурзик",
    "age": 3,
    "toys": [
      "Рука хозяйки",
      "Шнур от телевизора",
      "Обои на стене"
    ]
  }
]
```

sort_keys. Если `sort_keys=True` (по умолчанию False), ключи выводимого словаря будут отсортированы. Это удобно, если ключей много.

Пример со значением `sort_keys=True`:

```
with open('cats.json', 'w') as file:
    json.dump(data, file, ensure_ascii=False,
              indent=2, sort_keys=True)
```

Результат:

```
[
  {
    "age": 7,
    "name": "Барсик",
    "toys": [
      "Мышка",
      "Прутик",
      "Бантик",
      "Свой хвост"
    ]
  },
  {
    "age": 3,
    "name": "Мурзик",
    "toys": [
      "Рука хозяйки",
      "Шнур от телевизора",
      "Обои на стене"
    ]
  }
]
```

Замечания

- Чтобы не возникали проблемы с кодировкой, если в файл передаются данные с русскими буквами, как и во всех других случая работы с файлом, при открытии нужно принудительно устанавливать кодировку:`with open('cats_3.json', 'w', encoding='utf8') as cat_file:`
  `cat_file.write(json.dumps(cats_dict))`

- При создании json-файла «вручную» нужно помнить, что в нем нельзя использовать одинарные кавычки. При создании программными средствами нужные кавычки ставятся автоматически.
- Ключами словаря в json не могут быть кортежи и числа. Но ключ-число не вызовет ошибку при сериализации, он будет просто преобразован в строку.
- Помните, что при преобразовании данные будут не всегда того же типа, что были в Python, то есть:

```
print(json.loads(json.dumps(weekdays)) == weekdays)  # False
```