JSON в Python

JSON — это строка со словарем. Она представлена в виде байтовой последовательности. Вы можете отправить ее по сети приложению, а в нём воссоздать полученную структуру в объекты языка.

Пример JSON:

```
{
   "kwarg1": "value_1",
   "kwarg2": "value_2",
   "kwarg3": "value_3",
   "additional": ["value_4", "value_5", "value_6", ]
}
```

Упаковка объектов в байтовую последовательность называется **сериализацией**. А распаковка байтов в объекты языка программирования, приведение последовательности назад к типам и структурам, — **десериализацией**.

```
>>> #импортируем библиотеку
>>> import json
>>>
>>> #объявляем переменные
>>> string = "Some test string"
>>> integer = 211
>>> array = [1, 2, 3, 4, 5]
>>>
>>> #создаем словарь
>>> mydict = {"title": string, "code": integer, "data": array}
>>>
>>> #сериализуем его в JSON-структуру, как строку
>>> x = json.dumps(mydict)
>>> x
'{"title": "Some test string", "code": 211, "data": [1, 2, 3, 4, 5]}'
>>>
>>> #проводим десериализацию JSON-объекта
```

```
>>> y = json.loads(x)
>>> y
{'title': 'Some test string', 'code': 211, 'data': [1, 2, 3, 4, 5]}
>>>
>>> y["title"]
'Some test string'
```

Функции

Dumps позволяет создать JSON-строку из переданного в нее объекта. **Loads** — преобразовать строку назад в объекты языка.

Dump и load используют, чтобы сохранить результат в файл или воссоздать объект. Работают они схожим образом, но требуют передачи специального объекта для работы с файлом — filehandler.

```
>>> import json # импортируем библиотеку
>>>
>>> # создаем filehandler с помощью контекстного менеджера
>>> with open("data.json", "w") as fh:
...     json.dump([1, 2, 3, 4, 5], fh) # записываем структуру в файл
...
>>>
>>> # открываем тот же файл, но уже на чтение
>>> with open("data.json", "r") as fh:
...     json.load(fh) # загружаем структуру из файла
...
[1, 2, 3, 4, 5]
>>>
```

Как работать с пользовательскими объектами

Пользовательские классы не относятся к JSON-сериализуемым. Это значит, что просто применить к ним функции dumps, loads или dump и load не получится:

```
>>> # создаем пользовательский класс
```

```
>>> class Test:
...     def __init__(self, title, body):
...         self.title = title
...         self.body = body
...
>>> # создаем экземпляр класса
>>> t = Test("Some string", "Here is a bit more text, but still isn't enough")
>>>
>>> # пытаемся сериализовать его в JSON, но...
>>> json.dumps(t)
>>> # получаем ошибку TypeError, что класс несериализуем
>>>
```

Написать функцию

Чтобы сериализовать пользовательский объект в JSON-структуру данных, нужен аргумент default. Указывайте вызываемый объект, то есть функцию или статический метод.

Чтобы получить аргументы класса с их значениями, нужна встроенная функция __dict__, потому что любой класс — это словарь со ссылками на значения по ключу.

Чтобы сериализовать аргументы класса и их значения в JSON, напишите функцию:

```
>>> # используем анонимную функцию (лямбду), которая
>>> # в качестве сериализуемых данных указывает полученный __dict__ объекта
>>> json.dumps(t, default=lambda x: x.__dict__)
'{"title": "Some string", "body": "Here is a bit more text, but still isn\'t
enough"}'
>>>
```