Сериализует obj в JSON-подобный формат записывая его в fp (который поддерживает .write())

Если skipkeys=True (по умолчанию: False), тогда ключи словаря не базового типа (str, int, float, bool, None) будут пропущены, вместо того, чтобы вызывать исключение TypeError.

Модуль json всегда создает объекты str, не bytes. Следовательно, fp.write() должен поддерживать ввод str.

Когда 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.

Когда check\_circular=False (по умолчанию: True), тогда проверка циклических ссылок для типов контейнера будет пропущена, а такие ссылки будут вызывать OverflowError (или ошибку серьёзнее).

Если allow\_nan=False (по умолчанию: True), при каждой попытке сериализировать значение float, выходящее за допустимые пределы (nan, inf, -inf), будет возникать ValueError, в соответствии с сертификацией JSON. В случае если allow\_nan=True, будут использованы JavaScript аналоги (NaN, Infinity, -Infinity).

Когда indent является неотрицательным целым числом или строкой, то объекты и массивы JSON будут выводиться с этим количеством отступов. Если уровень отступа равен 0, отрицательный или "", будут использоваться новые строки без отступов. None (по умолчанию) отражает наиболее компактное представление. Если indent строка (например, "\t"), эта строка используется в качестве отступа.

Изменения в версии 3.2: Допускаются строки для отступа в дополнение к целым числам.

Separators должны быть tuple (item\_separator, key\_separator). По умолчанию используется значение (', ', ': ') если indent=None и (',', ': ') при другом значении. Чтобы получить наиболее компактное представление JSON, вы должны указать (',', ':').

Изменения в версии 3.4: Используйте(',', ': ') при отступе None.

Значение default должно быть функцией. Он вызывается для объектов, которые не могут быть сериализованы. Функция должна вернуть кодируемую версию объекта JSON или вызывать TypeError. Если default не указано, возникает ошибка TypeError.

Если sort\_keys=True (по умолчанию: False), ключи выводимого словаря будут отсортированы.

**Метод json dumps**

**json.dumps**(*obj, \*, skipkeys=False, ensure\_ascii=True, check\_circular=True, allow\_nan=True, cls=None, indent=None, separators=None, default=None, sort\_keys=False, \*\*kw*)

Сериализирует obj в строку str формата JSON с помощью [**таблицы конвертации**](https://pythonru.com/osnovy/modul-json-v-python#jsonencoder). Аргументы имеют то же значение, что и для dump().

Ключи в парах ключ/значение всегда являются строками. Когда словарь конвертируется в JSON, все ключи словаря преобразовываются в строки. Если в результате, сначала конвертировать его в JSON, а потом обратно, новый в словарь может отличаться от, то можно получить словарь идентичный исходному. Другими словами, loads(dumps(x)) != x если x имеет не строковые ключи.

### Метод json load

js**on.load**(fp, \*, cls=None, object\_hook=None, parse\_float=None, parse\_int=None, parse\_constant=None, object\_pairs\_hook=None, \*\*kw)

Десериализует из fp (текстовый или бинарный файл, который поддерживает метод .read() и содержит JSON документ) в объект Python используя эту таблицу конвертации.

object\_hook — опциональная функция, которая применяется к результату декодирования объекта. Использоваться будет значение, возвращаемое этой функцией, а не полученный словарь dict. Эта функция используется для реализации пользовательских декодеров (например [**JSON-RPC**](https://www.jsonrpc.org/)).

object\_pair\_shook — опциональная функция, которая применяется к результату декодирования объекта с определенной последовательностью пар ключ/значение. Вместо исходного словаря dict будет использоваться результат, возвращаемый функцией. Эта функция используется для реализации пользовательских декодеров. Если задан object\_hook, object\_pairs\_hook будет в приоритете.

В случае определения parse\_float, он будет вызван для каждого значения JSON с плавающей точкой. По умолчанию, это эквивалентно float(num\_str). Можно использовать другой тип данных или парсер для этого значения (например decimal.Decimal)

В случае определения parse\_int, он будет вызван для декодирования строк JSON int. По умолчанию, эквивалентен int(num\_str). Можно использовать другой тип данных или парсер для этого значения (например float).

В случае определения parse\_constant, он будет вызван для строк: -Infinity, Infinit, NaN. Может быть использован для вызова исключений при обнаружении недопустимых чисел JSON. parse\_constant больше не вызывается при null, true, fasle.

Чтобы использовать собственный подкласс JSONDecoder, укажите его с помощью аргумента cls; в противном случае используется JSONDecoder. Дополнительные аргументы ключевого слова будут переданы конструктору класса.

Если десериализованные данные не являются допустимым документом JSON, возникнет JSONDecodeError.

### Метод json loads

**json.loads**(s, \*, cls=None, object\_hook=None, parse\_float=None, parse\_int=None, parse\_constant=None, object\_pairs\_hook=None, \*\*kw)

Десериализует s (экземпляр str, bytes или bytearray, содержащий JSON документ) в объект Python используя таблицу конвертации.

Остальные аргументы аналогичны аргументам в load(), кроме кодировки, которая устарела либо игнорируется.

Если десериализованные данные не являются допустимым документом JSON, возникнет ошибка JSONDecodeError.

## 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:

### 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: