https://tproger.ru/articles/chto-takoe-json-vvedenie/

1. **JSON или JavaScript Object Notation** — текстовый формат обмена данными, основанный на JavaScript. Но при этом он может использоваться в любом языке программирования. Формат был разработан [Дугласом Крокфордом](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D0%BA%D1%84%D0%BE%D1%80%D0%B4,_%D0%94%D1%83%D0%B3%D0%BB%D0%B0%D1%81).

JSON используется в REST API. Также в качестве альтернативы можно использовать XML, но разработчики больше предпочитают именно JSON, так как он более читабельный и меньше весит.

**Как устроен JSON**

В качестве значений в JSON могут быть использоваться:

* числа;
* строки;
* массивы;
* JSON-объекты;
* литералы (логические значения true, false и null).

С простыми значениями не возникнет никаких трудностей. Разберём массивы и JSON-объекты, ведь, по сути, придётся работать именно с ними.

**JSON-объект**

**JSON-объект** — это неупорядоченное множество пар «ключ:значение», заключённых в фигурные скобки { }  и взаимодействие с ним проходит, как со словарем.

**Ключ** — это название параметра (свойства), который мы передаём серверу. Он служит маркером для принимающей запрос системы, чтобы она поняла, что мы ей отправили.

Давайте рассмотрим простой пример:

{

"name": "John",

"age": 26

}

Мы передали значение "John" с ключом "name", чтобы получить это свойство, нужно обратиться по его ключу, без которого сделать это было бы невозможно.

Значение может быть не только строкой или числом. Это может быть и другой объект! Или массив, или объект в массиве, массив в объекте… Абсолютно любое количество уровней вложенности!

Если мы говорим о параметрах, то в кавычки помещаются только строки, все остальное — нет.

Переносы строк делать необязательно. Вообще пробелы и переносы нужны только человеку для читабельности, система поймёт и без них.

Например, мы можем записать JSON-объект таким образом, и это тоже будет правильно:

{"name": "John","age": 26}

**Ключ**— ВСЕГДА строка, и мы в любом случае берём его в кавычки.

Ключи могут быть записаны в любом порядке, ведь, JSON-объект — это неупорядоченное множество пар «ключ:значение».

Если я хочу обратиться к "name", как мне это сказать? В Python есть два варианта это сделать:

1. Получить значение с помощью метода get().
2. Получить значение по ключу.

**get() vs dict[key]**

Разница между этими вариантами заключается в том, что если вы обратитесь к ключу, которого нет, то получите ошибку KeyError.

person = {

"name": "John",

"age": 26

}

print(person[“last\_name”])

>>> KeyError Traceback (most recent call last)

>>> KeyError: 'last\_name'

Но если воспользуетесь методом get(), то получите пустое значение:

last\_name = person.get("last\_name")

print(type("last\_name"))

>>> NoneType

Также у этого метода есть второй передаваемый аргумент, который будет предаваться если ключа нет:

last\_name = person.get("last\_name", "There is no last\_name")

print(last\_name)

>>> 'There is no last\_name'

**JSON-массив**

Массив заключен в квадратные скобки [ ].

["MALE", "FEMALE"]

Внутри квадратных скобок идет набор значений, разделённых запятыми. Здесь нет ключей, как в объекте, поэтому обращаться к массиву можно только по номеру элемента. И поэтому в случае массива менять местами данные внутри нельзя. Это упорядоченное множество значений, так что порядок важен.

**Значения**

Внутри массива могут быть любые значения:

* Цифры;

[1, 5, 10, 33]

* Строки;

["MALE", "FEMALE"]

* Литералы;

[true, false]

* Массивы;

["MALE", "FEMALE", [1, 5, 10, 33]]

* Объекты;

[1, {a:1, b:2}, "John"]

* Смесь.

[1, "John", true, [1, 5, 10, 33], {"a": 1, "b": 2}]

**Well Formed JSON**

JSON должен быть well formed, то есть синтаксически правильный.

Правила well formed JSON:

1. Данные написаны в виде пар «ключ:значение»
2. Данные разделены запятыми
3. Объект находится внутри фигурных скобок { }
4. Массив — внутри квадратных [ ]

Чтобы проверить JSON на синтаксис, можно использовать любой JSON Validator. Я использую [JSON Formatter](https://jsonformatter.org/), он не только проверяет корректность синтаксиса, но и форматирует JSON в читабельный визуальный формат!

Из этого:

{"name": "John", "age": 26, "is\_married": false, "children": { "name": "Julia", "age": null},"skills": ["Python", "Git", "Django", "Linux"]}

Он сделает вот это:

{

"name": "John",

"age": 26,

"is\_married": false,

"children": {

"name": "Julia",

"age": null

},

"skills": [

"Python",

"Git",

"Django",

"Linux"

]

}

**Итого**

**JSON (JavaScript Object Notation)** — текстовый формат обмена данными, основанный на JavaScript, но может использоваться в любом языке программирования. Легко читается человеком и машиной. Часто используется в REST API (чаще, чем XML из-за лучшей читабельности и меньшего веса).

Корректные значения JSON:

1. JSON-объект — неупорядоченное множество пар «ключ:значение», заключённое в фигурные скобки { };
2. массив — упорядоченный набор значений, разделённых запятыми. Находится внутри квадратных скобок [ ];
3. число (целое или вещественное);
4. литералы true (логическое значение «истина»), false (логическое значение «ложь») и null;
5. строка.

## Введение

## https://www.c-sharpcorner.com/article/working-with-json-string-in-C-Sharp/

JSON (нотация объектов JavaScript) — это облегченный формат обмена данными. Он не зависит от языка, прост для понимания и дает самоописание. Он используется как альтернатива XML. JSON сейчас очень популярен.

JSON представляет объекты в формате структурированного текста и данные, хранящиеся в парах ключ-значение. Многие сторонние элементы управления, такие как сетка пользовательского интерфейса Kendo, предоставляют данные из размера клиента на сторону сервера в формате строки JSON, поэтому необходимо привести нашу строку JSON к соответствующему объекту для доступа к данным. Существует множество способов работы с JSON в коде [C#](http://www.c-sharpcorner.com/technologies/csharp-programming) . .

## Использование библиотек Newtonsoft

## https://www.newtonsoft.com/json

Newtonsoft также известен как Json.NET. Это высокопроизводительная платформа JSON для .NET. Он очень прост в использовании и намного быстрее, чем встроенная сериализация JSON в .NET.

### Использование JsonConverter

Класс JsonConvert имеет метод для преобразования в строку JSON и из нее, SerializeObject() и DeserializeObject() соответственно. Его можно использовать там, где мы не будем преобразовывать в строку JSON и из нее.  
  
В следующем примере я использовал метод «JsonConvert.DeserializeObject», чтобы преобразовать объект JSON в объект пользовательского класса. Здесь имя ключа JSON должно совпадать с именем свойства класса, и совпадение не зависит от регистра.

static void Main(string[] args)

{

stringjsonData = @ "{

'FirstName': 'Jignesh', 'LastName': 'Trivedi'

}

";

var myDetails = JsonConvert.DeserializeObject < MyDetail > (jsonData);

Console.WriteLine(string.Concat("Hi ", myDetails.FirstName, " " + myDetails.LastName));

Console.ReadLine();

}

public class MyDetail

{

public string FirstName {

get;

set;

}

public string LastName {

get;

set;

}

}

С#

Копировать

### Использование JObject.Parse

Класс JObject имеет метод разбора; он анализирует строку JSON и преобразует ее в объект словаря "ключ-значение". В следующем примере я использовал метод «JObject.Parse» и извлек данные с помощью ключа.

string jsonData = @"{

'FirstName':'Jignesh',

'LastName':'Trivedi'

}";

var details = JObject.Parse(jsonData);

Console.WriteLine(string.Concat("Hi ", details["FirstName"], " " + details["LastName"]));

Console.ReadLine();

С#

Копировать

В следующем фрагменте кода мы просто назначаем вывод метода JObject.Parse динамическому объекту и получаем доступ к значению как к свойствам динамического объекта.

string jsonData = @"{

'FirstName':'Jignesh',

'LastName':'Trivedi'

}";

dynamic data = JObject.Parse(jsonData);

Console.WriteLine(string.Concat("Hi ", data.FirstName, " " + data.LastName));

Console.ReadLine();

С#

Копировать

Обратите внимание, что в описанном выше методе имя свойства или имя ключа чувствительно к регистру; т.е. имя свойства класса должно соответствовать ключу данных JSON.

### Использование класса Data Contract Json Serializer

Платформа .NET также предоставляет классы для сериализации и десериализации в JSON. Одним из них является DataContractJsonSerializer. Используя следующий код, мы можем десериализовать объект JSON. Чтобы использовать этот метод, необходимо помнить следующие моменты:

* + В проекте должна быть эталонная библиотека System.Runtime.Serialization.
  + Класс должен украшаться DataContract, а свойства украшаться атрибутами DataMember.
  + Используйте метод WriteObject для сериализации объекта и используйте метод ReadObject для десериализации объекта JSON.

static void Main(string[] args)

{

string jsonData = "{ \"FirstName\":\"Jignesh\",\"LastName\":\"Trivedi\" }";

DataContractJsonSerializerjsonSerializer = newDataContractJsonSerializer(typeof(MyDetail));

MemoryStream stream = newMemoryStream(Encoding.UTF8.GetBytes(jsonData));

stream.Position = 0;

MyDetaildataContractDetail = (MyDetail) jsonSerializer.ReadObject(stream);

Console.WriteLine(string.Concat("Hi ", dataContractDetail.FirstName, " " + dataContractDetail.LastName));

Console.ReadLine();

}

[DataContract]

public class MyDetail

{

[DataMember]

public string FirstName {

get;

set;

}

[DataMember]

public string LastName {

get;

set;

}

}

С#

Копировать

**Вывод**

вывод

## Резюме

Используя описанный выше метод, мы можем разобрать JSON в коде C# и извлечь значение из этого объекта.