Что такое JSON

* [Тестирование IT-систем](https://habr.com/ru/hub/it_testing/),
* [API](https://habr.com/ru/hub/api/),
* [Тестирование веб-сервисов](https://habr.com/ru/hub/web_testing/),
* [Тестирование мобильных приложений](https://habr.com/ru/hub/mobile_testing/)

Если вы тестируете API, то должны знать про два основных формата передачи данных:

* **XML** — используется в SOAP *(всегда)* и REST-запросах *(реже)*;
* **JSON** — используется в REST-запросах.

Сегодня я расскажу вам про JSON. И расскажу в основном с точки зрения «послать запрос в Postman или прочитать ответ», потому что статья рассчитана на студентов, впервые работающих с Postman.

**JSON**(англ. *JavaScript Object Notation*) — текстовый формат обмена данными, основанный на *JavaScript*. Но при этом формат независим от JS и может использоваться в любом языке программирования.

JSON используется в *REST API*. По крайней мере, тестировщик скорее всего столкнется с ним именно там.

**См также:**

[Что такое API](https://habr.com/ru/post/464261/) — общее знакомство с API

[Что такое XML](https://habr.com/ru/post/524288/) — второй популярный формат

[Введение в SOAP и REST: что это и с чем едят](https://www.youtube.com/watch?v=2YWfJHDNQy0&lc=Ugw5ZVWFvbbnToab-oF4AaABAg&pbjreload=101&ab_channel=okiseleva) — видео про разницу между SOAP и REST

В*SOAP API* возможен только формат *XML*, а вот *REST API* поддерживает как *XML*, так и *JSON*. Разработчики предпочитают *JSON*— он легче читается человеком и меньше весит. Так что давайте разберемся, как он выглядит, как его читать, и как ломать!

Содержание

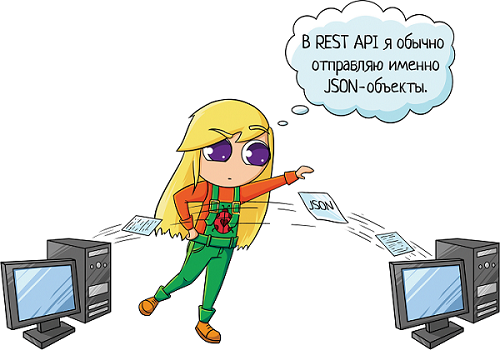
* [Как устроен JSON](https://habr.com/ru/post/554274/#json_elems)
  + [JSON-объект](https://habr.com/ru/post/554274/#json_object)
    - [Как устроен](https://habr.com/ru/post/554274/#json_object_detail)
    - [Ключ или свойство?](https://habr.com/ru/post/554274/#key_or_property)
    - [Итого](https://habr.com/ru/post/554274/#itogo_json_object)
  + [JSON-массив](https://habr.com/ru/post/554274/#json_array)
    - [Как устроен](https://habr.com/ru/post/554274/#json_array_detail)
    - [Значения внутри](https://habr.com/ru/post/554274/#json_array_value)
    - [Итого](https://habr.com/ru/post/554274/#itogo_json_array)
* [JSON vs XML](https://habr.com/ru/post/554274/#json_vs_xml)
* [Well Formed JSON](https://habr.com/ru/post/554274/#well_formed_json)
  + [Данные написаны в виде пар ключ:значение](https://habr.com/ru/post/554274/#key_value)
  + [Данные разделены запятыми](https://habr.com/ru/post/554274/#separate)
  + [Объект находится внутри фигурных скобок {}](https://habr.com/ru/post/554274/#object_bracket)
  + [Массив — внутри квадратных []](https://habr.com/ru/post/554274/#array_bracket)
* [Итого](https://habr.com/ru/post/554274/#itogo)

Как устроен JSON

В качестве значений в JSON могут быть использованы:

* JSON-объект
* Массив
* Число (целое или вещественное)
* Литералы *true*(логическое значение «истина»), *false*(логическое значение «ложь») и null
* Строка

Я думаю, с простыми значениями вопросов не возникнет, поэтому разберем массивы и объекты. Ведь если говорить про REST API, то обычно вы будете отправлять / получать именно json-объекты.



JSON-объект

**Как устроен**

Возьмем пример из [документации подсказок Дадаты по ФИО](https://confluence.hflabs.ru/pages/viewpage.action?pageId=204669115):

{

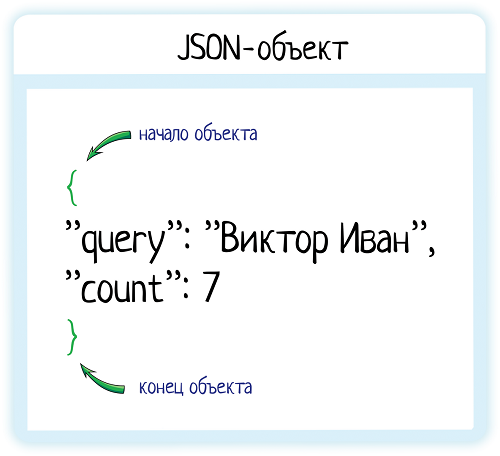
"query": "Виктор Иван",

"count": 7

}

И разберемся, что означает эта запись.

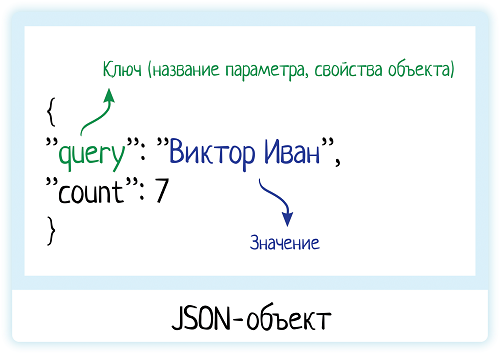
Объект заключен в фигурные скобки {}



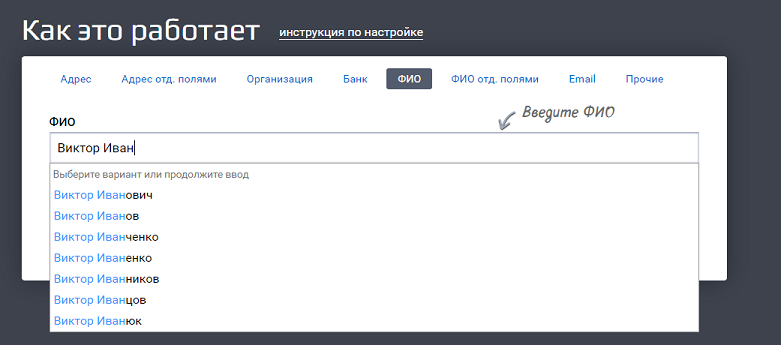
JSON-объект — это неупорядоченное множество пар *«ключ:значение»*.

Ключ — это название параметра, который мы передаем серверу. Он служит маркером для принимающей запрос системы: «смотри, здесь у меня значение такого-то параметра!». А иначе как система поймет, где что? Ей нужна подсказка!

Вот, например, «Виктор Иван» — это что? Ищем описание параметра «*query»*в документации — ага, да это же запрос для подсказок!

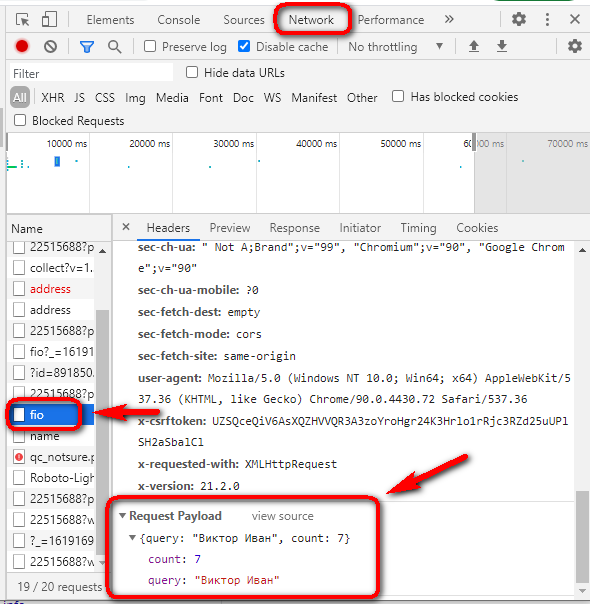


Это как если бы мы вбили строку «Виктор Иван» в GUI (графическом интерфейсе пользователя):



Когда пользователь начинает вводить данные в формочку, то сразу видит результат — появляется список подсказок. Это значит, что разработчик прописал в коде условие — делать некое действие на каждый ввод символа в это поле. Какое действие? Можно увидеть через f12.

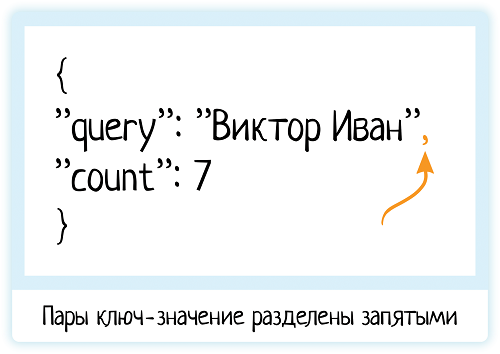
Открываем вкладку Network, вбиваем «Виктор Иван» и находим запрос, который при этом уходит на сервер. Ого, да это тот самый пример, что мы разбираем!



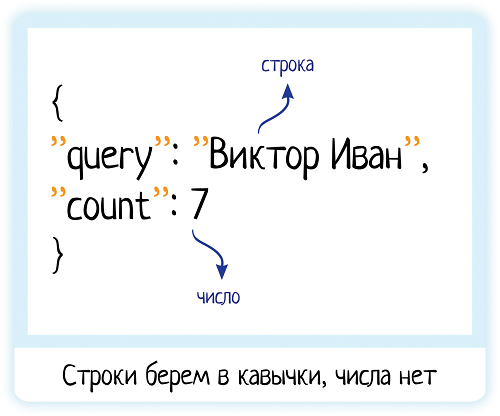
Клиент передает серверу запрос в JSON-формате. Внутри два параметра, две пары «ключ-значение»:

* *query*— строка, по которой ищем (то, что пользователь вбил в GUI);
* *count*— количество подсказок в ответе (в Дадате этот параметр зашит в форму, всегда возвращается 7 подсказок. Но если дергать подсказки напрямую, значение можно менять!)

Пары «ключ-значение» разделены запятыми:



Строки берем в кавычки, числа нет:



Конечно, внутри может быть не только строка или число. Это может быть и другой объект! Или массив... Или объект в массиве, массив в объекте... Любое количество уровней вложенности =))

Объект, массив, число, булево значение *(true / false)* — если у нас НЕ строка, кавычки не нужны. Но в любом случае это будет значение какого-то ключа:

|  |  |
| --- | --- |
| **НЕТ** | **ДА** |
| *{*  *"a": 1,*  { x:1, y:2 }  *}* | *{*  *"a": 1,*  *"inner\_object"*: *{ "x":1, "y":2 }*  } |
| *{*  *"a": 1,*  [2, 3, 4]  *}* | *{*  *"a": 1,*  *"inner\_array"*: [2, 3, 4]  *}* |

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

|  |  |
| --- | --- |
| **Так правильно** | **Так тоже правильно** |
| *{*  *"query": "Виктор Иван",*  *"count": 7*  *}* | *{  "query":"Виктор Иван", "count":7}* |

Ключ — ВСЕГДА строка, но мы все равно берем его в кавычки. В JavaScript этого можно не делать, в JSON нельзя.

|  |  |
| --- | --- |
| **Так правильно** | **Так правильно в JS, но неправильно в JSON** |
| *{*  *"query": "Виктор Иван",*  *"count": 7*  *}* | *{*  *query: "Виктор Иван",*  *count: 7*  *}* |

По крайней мере, если вы работаете с простыми значениями ключей, а несколько слов записываете в верблюжьемРегистре или в змеином\_регистре. Если вы хотите написать в ключе несколько слов через пробел, ключ нужно взять в кавычки.

|  |  |
| --- | --- |
| **НЕТ** | **ДА** |
| *{*  *my query: "Виктор Иван"*  *}* | *{*  *"my query": "Виктор Иван"*  *}* |

И все же я рекомендую использовать простые названия ключей, или использовать *snake\_case*.

**См также:**

[CamelCase, snake\_case и другие регистры](https://okiseleva.blogspot.com/2020/01/camelcase-snakecase.html) — подробнее о разных регистрах

Писать ключи можно в любом порядке. Ведь JSON-объект — это неупорядоченное множество пар «ключ:значение».

|  |  |
| --- | --- |
| **Так правильно** | **Так тоже правильно** |
| *{*  *query: "Виктор Иван",*  *count: 7*  *}* | *{*  *count: 7,*  *query: "Виктор Иван"*  *}* |

Очень важно это понимать, и тестировать! Принимающая запрос система должна ориентировать на название ключей в запросе, а не на порядок их следования. Ключевое слово «должна» )) Хотя знаю примеры, когда от перестановки ключей местами всё ломалось, ведь «первым должен идти запрос, а не count!».

**Ключ или свойство?**

Вот у нас есть JSON-объект:

{

"query": "Виктор Иван",

"count": 7

}

Что такое *«query»*? Если я хочу к нему обратиться, как мне это сказать? Есть 2 варианта, и оба правильные:

— Обратиться к свойству объекта;

— Получить значение по ключу.



То есть «*query»* можно назвать как ключом, так и свойством. А как правильно то?



Правильно и так, и так! Просто есть разные определения объекта:

**Объект**

В JS объект — это именно объект. У которого есть набор свойств и методов:

* Свойства — описывают, ЧТО мы создаем.
* Методы — что объект умеет ДЕЛАТЬ.

То есть если мы хотим создать машину, есть два пути:

1. Перечислить 10 разных переменных — модель, номер, цвет, пробег...
2. Создать один объект, где будут все эти свойства.

Аналогично с кошечкой, собачкой, другом из записной книжки...



Объектно-ориентированное программирование (ООП) предлагает мыслить не набором переменных, а объектом. Хотя бы потому, что это логичнее. Переменных в коде будет много, как понять, какие из них взаимосвязаны?

Вот если я создаю машину, сколько переменных мне надо заполнить? А если меняю данные? А если удаляю? Когда переменные разбросаны по коду, можно забыть про какую-то и получить ошибку в интерфейсе. А если у нас есть цельный объект, всегда можно посмотреть, какие у него есть свойства и методы.

Например, создадим кошечку:

var cat = {

name: “Pussy”,

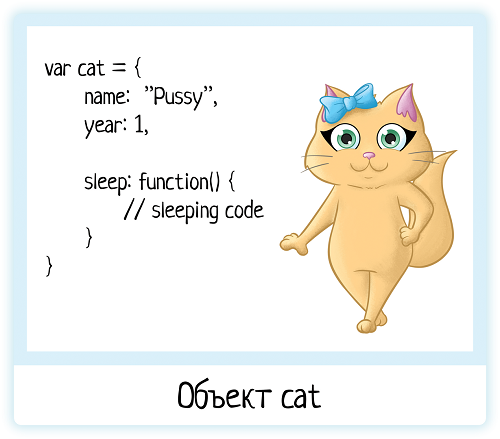
year: 1,

sleep: function() {

*// sleeping code*

}

}



В объекте cat есть:

* Свойства — *name, year* (что это за кошечка)
* Функции — *sleep*(что она умеет делать, описание поведения)

По коду сразу видно, что у кошечки есть имя и возраст, она умеет спать. Если разработчик решит добавить новые свойства или методы, он дополнит этот объект, и снова всё в одном месте.

Если потом нужно будет получить информацию по кошечке, разработчик сделает REST-метод *getByID*, *searchKitty*, или какой-то другой. А в нем будет возвращать свойства объекта.

То есть метод вернет

{

name: “Pussy”,

year: 1,

}

И при использовании имени вполне уместно говорить «обратиться к свойству объекта». Это ведь объект (кошечка), и его свойства!

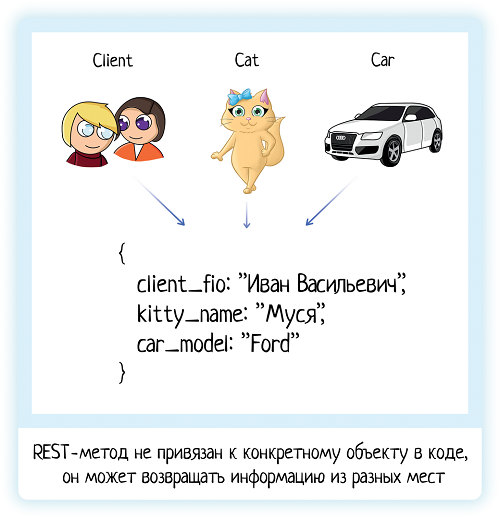
**Набор пар «ключ:значение»**

Второе определение объекта — неупорядоченное множество пар ключ:значение, заключенное в фигурные скобки {}.

Оно применимо тогда, когда внутри фигурных скобок приходит не конкретный целостный объект, а просто набор полей. Они могут быть связаны между собой, а могут относится к совершенно разным объектам внутри кода:

* *client\_fio (в коде это свойство fio объекта client)*
* *kitty\_name (в коде это свойство name объекта cat)*
* *car\_model (в коде это свойство model объекта car)*
* *…*

В таком случае логично называть эти параметры именно ключами — мы хотим получить значение по ключу.



Но в любом случае, и «ключ», и «свойство» будет правильно. Не пугайтесь, если в одной книге / статье / видео увидели одно, в другой другое... Это просто разные трактовки ¯\\_(ツ)\_/¯

**Итого**

Json-объект — 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.

Значения ключа могут быть любыми:

* число
* строка
* массив
* другой объект
* ...

И только строку мы берем в кавычки!

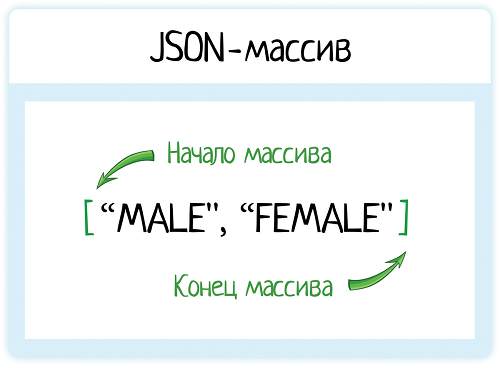
JSON-массив

**Как устроен**

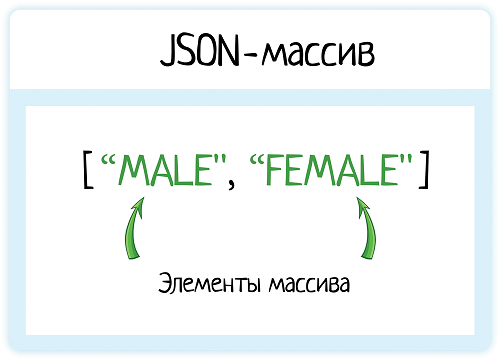
Давайте снова начнем с примера. Это массив:

[ "MALE", "FEMALE" ]

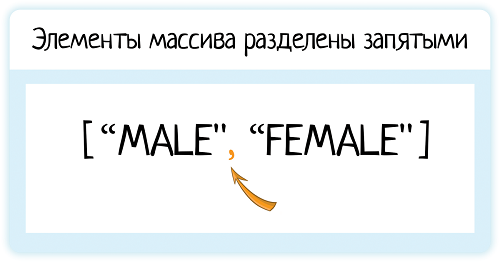
Массив заключен в квадратные скобки []



Внутри квадратных скобок идет набор значений. Тут 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.



Значения разделены запятыми:



**Значения внутри**

Внутри массива может быть все, что угодно:

**Цифры**

[ 1, 5, 10, 33 ]

**Строки**

[ "MALE", "FEMALE" ]

**Смесь**

[ 1, "Андрюшка",  10, 33 ]

**Объекты**

Да, а почему бы и нет:

[1, {a:1, b:2}, "такой вот массивчик"]

Или даже что-то более сложное. Вот пример ответа подсказок из Дадаты:

[

{

"value": "Иванов Виктор",

"unrestricted\_value": "Иванов Виктор",

"data": {

"surname": "Иванов",

"name": "Виктор",

"patronymic": null,

"gender": "MALE"

}

},

{

"value": "Иванченко Виктор",

"unrestricted\_value": "Иванченко Виктор",

"data": {

"surname": "Иванченко",

"name": "Виктор",

"patronymic": null,

"gender": "MALE"

}

},

{

"value": "Виктор Иванович",

"unrestricted\_value": "Виктор Иванович",

"data": {

"surname": null,

"name": "Виктор",

"patronymic": "Иванович",

"gender": "MALE"

}

}

]

Система возвращает массив подсказок. Сколько запросили в параметре *count*, столько и получили. Каждая подсказка — объект, внутри которого еще один объект. И это далеко не сама сложная структура! Уровней вложенности может быть сколько угодно — массив в массиве, который внутри объекта, который внутри массива, который внутри объекта...

Ну и, конечно, можно и наоборот, передать массив в объекте. Вот пример запроса в подсказки:

{

"query": "Виктор Иван",

"count": 7,

"parts": ["NAME", "SURNAME"]

}

Это объект (так как в фигурных скобках и внутри набор пар «ключ:значение»). А значение ключа *"parts"* — это массив элементов!

**Итого**

Массив — это просто набор значений, разделенных запятыми. Находится внутри квадратных скобок [].

А вот внутри него может быть все, что угодно:

* числа
* строки
* другие массивы
* объекты
* смесь из всего вышеназванного

JSON vs XML

В SOAP можно применять только XML, там без вариантов.

В REST можно применять как XML, так и JSON. Разработчики отдают предпочтение json-формату, потому что он проще воспринимается и меньше весит. В XML есть лишняя обвязка, название полей повторяется дважды (открывающий и закрывающий тег).

Сравните один и тот же запрос на обновление данных в карточке пользователя:

**XML**

<req>

<surname>Иванов</surname>

<name>Иван</name>

<patronymic>Иванович</patronymic>

<birthdate>01.01.1990</birthdate>

<birthplace>Москва</birthplace>

<phone>8 926 766 48 48</phone>

</req>

**JSON**

{

"surname": "Иванов",

"name": "Иван",

"patronymic": "Иванович",

"birthdate": "01.01.1990",

"birthplace": "Москва",

"phone": "8 926 766 48 48"

}

За счет того, что мы не дублируем название поля каждый раз *«surname – surname»*, читать JSON проще. И за счет этого же запрос меньше весит, что при плохом интернете бывает важно. Или при большой нагрузке.

**См также:**

[Инфографика REST vs SOAP](https://nordicapis.com/rest-vs-soap-nordic-apis-infographic-comparison/)

Well Formed JSON

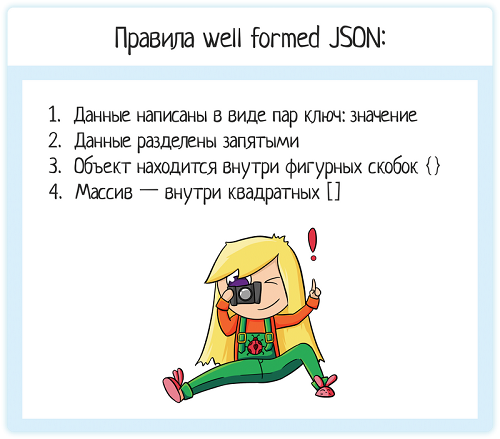
Разработчик сам решает, какой JSON будет считаться правильным, а какой нет. Но есть общие правила, которые нельзя нарушать. Наш JSON должен быть *well formed*, то есть синтаксически корректный.

Чтобы проверить JSON на синтаксис, можно использовать любой JSON Validator (так и гуглите). Я рекомендую сайт [w3schools](https://www.w3schools.com/js/js_json_syntax.asp). Там есть сам валидатор + описание типичных ошибок с примерами.

Но учтите, что парсеры внутри кода работают не по википедии или w3schools, а по RFC, стандарту. Так что если хотите изучить «каким должен быть JSON», то правильнее открывать RFC и искать там [JSON Grammar](https://tools.ietf.org/html/rfc7159#page-4). Однако простому тестировщику хватит набора типовых правил с w3schools, их и разберем.

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 []



1. Данные написаны в виде пар «ключ:значение»

Например, так:

"name":"Ольга"

В JSON название ключа нужно брать в кавычки, в JavaScript не обязательно — он и так знает, что это строка. Если мы тестируем API, то там будет именно JSON, так что кавычки обычно нужны.

Но учтите, что это правило касается JSON-объекта. Потому что json может быть и числом, и строкой. То есть:

123

Или

"Ольга"

Это тоже корректный json, хоть и не в виде пар «ключ:значение».

И вот если у вас по ТЗ именно json-объект на входе, попробуйте его сломать, не передав ключ. Ещё можно не передать значение, но это не совсем негативный тест — система может воспринимать это нормально, как пустой ввод.

2. Данные разделены запятыми

Пары «ключ:значение» в объекте разделяются запятыми. После последней пары запятая не нужна!

**Типичная ошибка:** поставили запятую в конце объекта:

{

"query": "Виктор Иван",

"count": 7,

}



Это последствия копипасты. Взяли пример из документации, подставили в постман (ну или разработчик API подставил в код, который будет вызывать систему), а потом решили поменять поля местами.

В итоге было так:

{

"count": 7,

"query": "Виктор Иван"

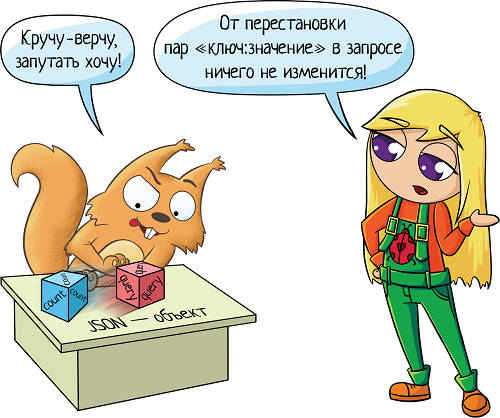
}

Смотрим на запрос — ну, *query*то важнее чем *count*, надо поменять их местами! Копипастим всю строку «*"count": 7,»,*вставляем ниже. Перед ней запятую добавляем, а «лишнюю» убрать забываем. По крайней мере у меня это частая ошибка, когда я «кручу-верчу, местами поменять хочу».

Другой пример — когда мы добавляем в запрос новое поле. Примерный сценарий:

1. У меня уже есть работающий запрос в Postman-е. Но в нем минимум полей.
2. Я его клонирую
3. Копирую из документации нужное мне поле. Оно в примере не последнее, так что идёт с запятой на конце.
4. Вставляю себе в конце запроса — в текущий конец добавляю запятую, потом вставляю новую строку.
5. Отправляю запрос — ой, ошибка! Из копипасты то запятую не убрала!

Я на этот сценарий постоянно напарываюсь при тестировании перестановки полей. А ведь это нужно проверять! Хороший запрос должен быть как в математической присказке: «от перемены мест слагаемых сумма не меняется».



Не зря же определение json-объекта гласит, что «это неупорядоченное множество пар ключ:значение». Раз неупорядоченное — я могу передавать ключи в любом порядке. И сервер должен искать по запросу название ключа, а не обращаться к индексу элемента.

Разработчик, который будет взаимодействовать с API, тоже человек, который может ошибиться. И если система будет выдавать невразумительное сообщение об ошибке, можно долго думать, где конкретно ты налажал. Поэтому ошибки тоже тестируем.

Чтобы протестировать, как система обрабатывает «плохой json», замените запятую на точку с запятой:

{

"count": 7;

"query": "Виктор Иван"

}

Или добавьте лишнюю запятую в конце запроса — эта ошибка будет встречаться чаще!

{

"count": 7,

"query": "Виктор Иван",

}

Или пропустите запятую там, где она нужна:

{

"count": 7

"query": "Виктор Иван"

}

Аналогично с массивом. Данные внутри разделяются через запятую. Хотите попробовать сломать? Замените запятую на точку с запятой! Тогда система будет считать, что у вас не 5 значений, а 1 большое:

[1, 2, 3, 4, 5] <!-- корректный массив на 5 элементов\* -->

[1; 2; 3; 4; 5] <!-- некорректный массив, так как такого разделителя быть не должно. Это может быть простой строкой, но тогда нужны кавычки -->!

 \*Я добавила комментарии внутри блока кода. Но учтите, что в JSON комментариев нет. Вообще. Так что если вы делаете запрос в Postman, не получится расставить комментарии у разных строчек в JSON-формате.

3. Объект находится внутри фигурных скобок {}

Это объект:

{a: 1, b: 2}

Чтобы сломать это условие, уберите одну фигурную скобку:

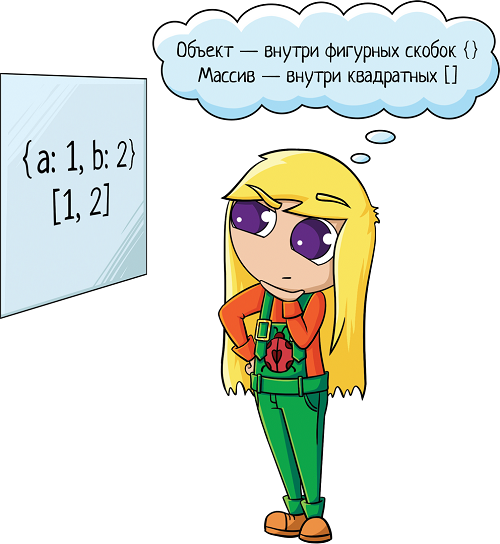
{a: 1, b: 2

a: 1, b: 2}

Или попробуйте передать объект как массив:

[ a: 1, b: 2 ]

Ведь если система ждет от вас в запросе объект, то она будет искать фигурные скобки.



4. Массив — внутри квадратных []

Это массив:

[1, 2]

Чтобы сломать это условие, уберите одну квадратную скобку:

[1, 2

1, 2]

Или попробуйте передать массив как объект, в фигурных скобках:

{ 1, 2 }

Ведь если система ждет от вас в запросе массив, то она будет искать квадратные скобки.

Итого

**JSON** *(JavaScript Object Notation)*— текстовый формат обмена данными, основанный на *JavaScript*. Легко читается человеком и машиной. Часто используется в REST API (чаще, чем XML).



Корректные значения JSON:

* JSON-объект — неупорядоченное множество пар «ключ:значение», заключённое в фигурные скобки «{ }».
* Массив — упорядоченный набор значений, разделенных запятыми. Находится внутри квадратных скобок [].
* Число (целое или вещественное).
* Литералы *true*(логическое значение «истина»), *false*(логическое значение «ложь») и *null*.
* Строка

При тестировании *REST API* чаще всего мы будем работать именно с объектами, что в запросе, что в ответе. Массивы тоже будут, но обычно внутри объектов.

Комментариев в JSON, увы, нет.

Правила *well formed JSON*:

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

**См также:**

[Introducing JSON](https://www.json.org/json-en.html)

[RFC (стандарт)](https://tools.ietf.org/html/rfc7159)

[Что такое XML](https://habr.com/ru/post/524288/)

*+комментарии к этой статье =)*

*PS — больше полезных*