


[🏠](#) → [Язык JavaScript](#) → [Объекты: основы](#) 20 июня 2022 г.

Объекты

Как мы знаем из главы [Типы данных](#), в JavaScript существует 8 типов данных. Семь из них называются «примитивными», так как содержат только одно значение (будь то строка, число или что-то другое).

Объекты же используются для хранения коллекций различных значений и более сложных сущностей. В JavaScript объекты используются очень часто, это одна из основ языка. Поэтому мы должны понять их, прежде чем углубляться куда-либо ещё.

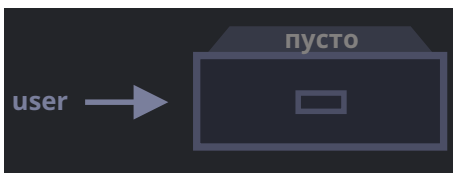
Объект может быть создан с помощью фигурных скобок `{...}` с необязательным списком *свойств*. Свойство – это пара «ключ: значение», где **ключ** – это строка (также называемая «именем свойства»), а **значение** может быть чем угодно.

Мы можем представить объект в виде ящика с подписанными папками. Каждый элемент данных хранится в своей папке, на которой написан ключ. По ключу папку легко найти, удалить или добавить в неё что-либо.



Пустой объект («пустой ящик») можно создать, используя один из двух вариантов синтаксиса:

```
1 let user = new Object(); // синтаксис "конструктор объекта"
2 let user = {}; // синтаксис "литерал объекта"
```



Обычно используют вариант с фигурными скобками `{...}`. Такое объявление называют *литералом объекта* или *литеральной нотацией*.

Литералы и свойства

При использовании литерального синтаксиса `{...}` мы сразу можем поместить в объект несколько свойств в виде пар «ключ: значение»:

```

1 let user = {      // объект
2   name: "John",   // под ключом "name" хранится значение "John"
3   age: 30         // под ключом "age" хранится значение 30
4 };

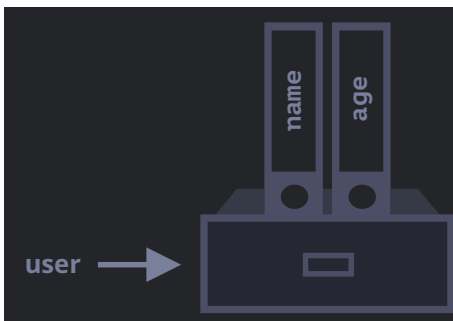
```

У каждого свойства есть ключ (также называемый «имя» или «идентификатор»). После имени свойства следует двоеточие ":", и затем указывается значение свойства. Если в объекте несколько свойств, то они перечисляются через запятую.

В объекте `user` сейчас находятся два свойства:

1. Первое свойство с именем `"name"` и значением `"John"`.
2. Второе свойство с именем `"age"` и значением `30`.

Можно сказать, что наш объект `user` – это ящик с двумя папками, подписанными «name» и «age».



Мы можем в любой момент добавить в него новые папки, удалить папки или прочитать содержимое любой папки.

Для обращения к свойствам используется запись «через точку»:

```

1 // получаем свойства объекта:
2 alert( user.name ); // John
3 alert( user.age );  // 30

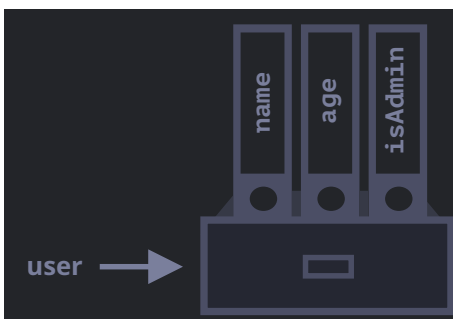
```

Значение может быть любого типа. Давайте добавим свойство с логическим значением:

```

1 user.isAdmin = true;

```

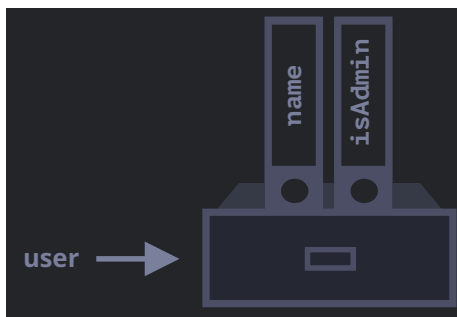


Для удаления свойства мы можем использовать оператор `delete`:

```

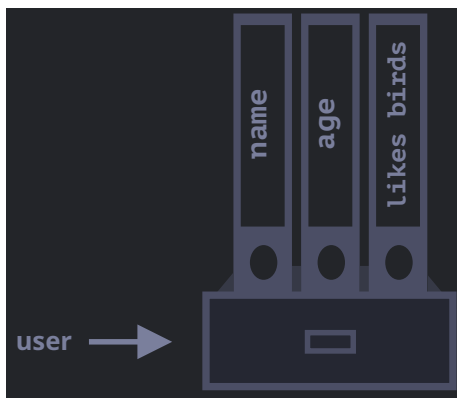
1 delete user.age;

```



Имя свойства может состоять из нескольких слов, но тогда оно должно быть заключено в кавычки:

```
1 let user = {  
2   name: "John",  
3   age: 30,  
4   "likes birds": true // имя свойства из нескольких слов должно быть в кавычка  
5 };
```



Последнее свойство объекта может заканчиваться запятой:

```
1 let user = {  
2   name: "John",  
3   age: 30,  
4   }
```

Это называется «висячая запятая». Такой подход упрощает добавление, удаление и перемещение свойств, так как все строки объекта становятся одинаковыми.

i Объект, объявленный как константа, может быть изменён

Объект, объявленный через `const`, может быть изменён.

Например:

```
1  const user = {  
2    name: "John"  
3  };  
4  
5  user.name = "Pete"; // (*)  
6  
7  alert(user.name); // Pete
```

Может показаться, что строка `(*)` должна вызвать ошибку, но нет, здесь всё в порядке. Дело в том, что объявление `const` защищает от изменений только саму переменную `user`, а не её содержимое.

Определение `const` выдаст ошибку только если мы присвоим переменной другое значение: `user=...`.

Есть ещё один способ сделать константами свойства объекта, который мы рассмотрим в главе [Флаги и дескрипторы свойств](#).

Квадратные скобки

Для свойств, имена которых состоят из нескольких слов, доступ к значению «через точку» не работает:

```
1  // это вызовет синтаксическую ошибку  
2  user.likes birds = true
```

JavaScript видит, что мы обращаемся к свойству `user.likes`, а затем идёт непонятное слово `birds`. В итоге синтаксическая ошибка.

Точка требует, чтобы ключ был именован по правилам именования переменных. То есть не имел пробелов, не начинался с цифры и не содержал специальные символы, кроме `$` и `_`.

Для таких случаев существует альтернативный способ доступа к свойствам через квадратные скобки. Такой способ сработает с любым именем свойства:

```
1  let user = {};  
2  
3  // присваивание значения свойству  
4  user["likes birds"] = true;  
5  
6  // получение значения свойства  
7  alert(user["likes birds"]); // true  
8  
9  // удаление свойства  
10 delete user["likes birds"];
```

Сейчас всё в порядке. Обратите внимание, что строка в квадратных скобках заключена в кавычки (подойдёт любой тип кавычек).

Квадратные скобки также позволяют обратиться к свойству, имя которого может быть результатом выражения. Например, имя свойства может храниться в переменной:

```
1 let key = "likes birds";
2
3 // то же самое, что и user["likes birds"] = true;
4 user[key] = true;
```

Здесь переменная `key` может быть вычислена во время выполнения кода или зависеть от пользовательского ввода. После этого мы используем её для доступа к свойству. Это даёт нам большую гибкость.

Пример:

```
1 let user = {
2   name: "John",
3   age: 30
4 };
5
6 let key = prompt("Что вы хотите узнать о пользователе?", "name");
7
8 // доступ к свойству через переменную
9 alert( user[key] ); // John (если ввели "name")
```

Запись «через точку» такого не позволяет:

```
1 let user = {
2   name: "John",
3   age: 30
4 };
5
6 let key = "name";
7 alert( user.key ); // undefined
```

Вычисляемые свойства

Мы можем использовать квадратные скобки в литеральной нотации для создания *вычисляемого свойства*.

Пример:

```
1 let fruit = prompt("Какой фрукт купить?", "apple");
2
3 let bag = {
4   [fruit]: 5, // имя свойства будет взято из переменной fruit
5 };
6
7 alert( bag.apple ); // 5, если fruit="apple"
```

Смысл вычисляемого свойства прост: запись `[fruit]` означает, что имя свойства необходимо взять из переменной `fruit`.

И если посетитель введёт слово `"apple"`, то в объекте `bag` теперь будет лежать свойство `{apple: 5}`.

По сути, пример выше работает так же, как и следующий пример:



```
1 let fruit = prompt("Какой фрукт купить?", "apple");
2 let bag = {};
3
4 // имя свойства будет взято из переменной fruit
5 bag[fruit] = 5;
```

...Но первый пример выглядит лаконичнее.

Мы можем использовать и более сложные выражения в квадратных скобках:

```
1 let fruit = 'apple';
2 let bag = {
3   [fruit + 'Computers']: 5 // bag.appleComputers = 5
4 };;
```

Квадратные скобки дают намного больше возможностей, чем запись через точку. Они позволяют использовать любые имена свойств и переменные, хотя и требуют более громоздких конструкций кода.

Подведём итог: в большинстве случаев, когда имена свойств известны и просты, используется запись через точку. Если же нам нужно что-то более сложное, то мы используем квадратные скобки.

Свойство из переменной

В реальном коде часто нам необходимо использовать существующие переменные как значения для свойств с тем же именем.

Например:



```
1 function makeUser(name, age) {
2   return {
3     name: name,
4     age: age
5     // ...другие свойства
6   };
7 }
8
9 let user = makeUser("John", 30);
10 alert(user.name); // John
```

В примере выше название свойств `name` и `age` совпадают с названиями переменных, которые мы подставляем в качестве значений этих свойств. Такой подход настолько распространён, что существуют специальные *короткие свойства* для упрощения этой записи.

Вместо `name:name` мы можем написать просто `name` :

```
1 function makeUser(name, age) {
2   return {
3     name, // то же самое, что и name: name
4     age   // то же самое, что и age: age
5     // ...
6   };
7 }
```

Мы можем использовать как обычные свойства, так и короткие в одном и том же объекте:

```
1 let user = {
2   name, // тоже самое, что и name:name
3   age: 30
4 };
```

Ограничения на имена свойств

Как мы уже знаем, имя переменной не может совпадать с зарезервированными словами, такими как «for», «let», «return» и т.д.

Но для свойств объекта такого ограничения нет:

```
1 // эти имена свойств допустимы
2 let obj = {
3   for: 1,
4   let: 2,
5   return: 3
6 };
7
8 alert( obj.for + obj.let + obj.return ); // 6
```



Иными словами, нет никаких ограничений к именам свойств. Они могут быть в виде строк или символов (специальный тип для идентификаторов, который будет рассмотрен позже).

Все другие типы данных будут автоматически преобразованы к строке.

Например, если использовать число 0 в качестве ключа, то оно превратится в строку "0" :

```
1 let obj = {
2   0: "Тест" // то же самое что и "0": "Тест"
3 };
4
5 // обе функции alert выведут одно и то же свойство (число 0 преобразуется в стр
6 alert( obj["0"] ); // Тест
7 alert( obj[0] ); // Тест (то же свойство)
```



Есть небольшой подводный камень, связанный со специальным свойством `__proto__`. Мы не можем установить его в необъектное значение:



```
1 let obj = {};  
2 obj.__proto__ = 5; // присвоим число  
3 alert(obj.__proto__); // [object Object], значение - это объект, т.е. не то, чт
```

Как мы видим, присвоение примитивного значения 5 игнорируется.

Мы более подробно исследуем особенности свойства `__proto__` в следующих главах [Прототипное наследование](#), а также предложим [способы исправления](#) такого поведения.

Проверка существования свойства, оператор «in»

В отличие от многих других языков, особенность JavaScript-объектов в том, что можно получить доступ к любому свойству. Даже если свойства не существует – ошибки не будет!

При обращении к свойству, которого нет, возвращается `undefined`. Это позволяет просто проверить существование свойства:



```
1 let user = {};  
2  
3 alert( user.noSuchProperty === undefined ); // true означает "свойства нет"
```

Также существует специальный оператор `"in"` для проверки существования свойства в объекте.

Синтаксис оператора:

```
1 "key" in object
```

Пример:



```
1 let user = { name: "John", age: 30 };  
2  
3 alert( "age" in user ); // true, user.age существует  
4 alert( "blabla" in user ); // false, user.blabla не существует
```

Обратите внимание, что слева от оператора `in` должно быть *имя свойства*. Обычно это строка в кавычках.

Если мы опускаем кавычки, это значит, что мы указываем переменную, в которой находится имя свойства. Например:



```
1 let user = { age: 30 };  
2  
3 let key = "age";  
4 alert( key in user ); // true, имя свойства было взято из переменной key
```

Для чего вообще нужен оператор `in`? Разве недостаточно сравнения с `undefined`?

В большинстве случаев прекрасно сработает сравнение с `undefined`. Но есть особый случай, когда оно не подходит, и нужно использовать `"in"`.

Это когда свойство существует, но содержит значение `undefined`:



```
1 let obj = {
2   test: undefined
3 };
4
5 alert( obj.test ); // выведет undefined, значит свойство не существует?
6 alert( "test" in obj ); // true, свойство существует!
```

В примере выше свойство `obj.test` технически существует в объекте. Оператор `in` сработал правильно.

Подобные ситуации случаются очень редко, так как `undefined` обычно явно не присваивается. Для «неизвестных» или «пустых» свойств мы используем значение `null`. Таким образом, оператор `in` является экзотическим гостем в коде.

Цикл "for..in"

Для перебора всех свойств объекта используется цикл `for..in`. Этот цикл отличается от изученного ранее цикла `for(;;)`.

Синтаксис:

```
1 for (key in object) {
2   // тело цикла выполняется для каждого свойства объекта
3 }
```

К примеру, давайте выведем все свойства объекта `user`:



```
1 let user = {
2   name: "John",
3   age: 30,
4   isAdmin: true
5 };
6
7 for (let key in user) {
8   // ключи
9   alert( key ); // name, age, isAdmin
10  // значения ключей
11  alert( user[key] ); // John, 30, true
12 }
```

Обратите внимание, что все конструкции «for» позволяют нам объявлять переменную внутри цикла, как, например, `let key` здесь.

Кроме того, мы могли бы использовать другое имя переменной. Например, часто используется вариант `"for (let prop in obj)"`.

Упорядочение свойств объекта

Упорядочены ли свойства объекта? Другими словами, если мы будем в цикле перебирать все свойства объекта, получим ли мы их в том же порядке, в котором мы их добавляли? Можем ли мы на это рассчитывать?

Короткий ответ: свойства упорядочены особым образом: свойства с целочисленными ключами сортируются по возрастанию, остальные располагаются в порядке создания. Разберёмся подробнее.

В качестве примера рассмотрим объект с телефонными кодами:



```
1 let codes = {
2   "49": "Германия",
3   "41": "Швейцария",
4   "44": "Великобритания",
5   // ..,
6   "1": "США"
7 };
8
9 for (let code in codes) {
10  alert(code); // 1, 41, 44, 49
11 }
```

Если мы делаем сайт для немецкой аудитории, то, вероятно, мы хотим, чтобы код 49 был первым.

Но если мы запустим код, мы увидим совершенно другую картину:

- США (1) идёт первым
- затем Швейцария (41) и так далее.

Телефонные коды идут в порядке возрастания, потому что они являются целыми числами: 1, 41, 44, 49.

i Целочисленные свойства? Это что?

Термин «целочисленное свойство» означает строку, которая может быть преобразована в целое число и обратно без изменений.

То есть, "49" – это целочисленное имя свойства, потому что если его преобразовать в целое число, а затем обратно в строку, то оно не изменится. А вот свойства "+49" или "1.2" таковыми не являются:



```
1 // Math.trunc - встроенная функция, которая удаляет десятичную часть
2 alert( String(Math.trunc(Number("49"))) ); // "49", то же самое ⇒ свойство
3 alert( String(Math.trunc(Number("+49"))) ); // "49", не то же самое, что "+4
4 alert( String(Math.trunc(Number("1.2"))) ); // "1", не то же самое, что "1.2"
```

...С другой стороны, если ключи не целочисленные, то они перебираются в порядке создания, например:



```
1 let user = {
2   name: "John",
3   surname: "Smith"
4 };
5 user.age = 25; // добавим ещё одно свойство
6
7 // не целочисленные свойства перечислены в порядке создания
8 for (let prop in user) {
9   alert( prop ); // name, surname, age
10 }
```

Таким образом, чтобы решить нашу проблему с телефонными кодами, мы можем схитрить, сделав коды не целочисленными свойствами. Добавления знака "+" перед каждым кодом будет достаточно.

Пример:

```
1  let codes = {
2    "+49": "Германия",
3    "+41": "Швейцария",
4    "+44": "Великобритания",
5    // ...,
6    "+1": "США"
7  };
8
9  for (let code in codes) {
10    alert( +code ); // 49, 41, 44, 1
11  }
```



Теперь код работает так, как мы задумывали.

Итого

Объекты – это ассоциативные массивы с рядом дополнительных возможностей.

Они хранят свойства (пары ключ-значение), где:

- Ключи свойств должны быть строками или символами (обычно строками).
- Значения могут быть любого типа.

Чтобы получить доступ к свойству, мы можем использовать:

- Запись через точку: `obj.property`.
- Квадратные скобки `obj["property"]`. Квадратные скобки позволяют взять ключ из переменной, например, `obj[varWithKey]`.

Дополнительные операторы:

- Удаление свойства: `delete obj.prop`.
- Проверка существования свойства: `"key" in obj`.
- Перебор свойств объекта: цикл `for` `for (let key in obj)`.

То, что мы изучали в этой главе, называется «простым объектом» («plain object») или просто `Object`.

В JavaScript есть много других типов объектов:

- `Array` для хранения упорядоченных коллекций данных,
- `Date` для хранения информации о дате и времени,
- `Error` для хранения информации об ошибке.
- ... и так далее.

У них есть свои особенности, которые мы изучим позже. Иногда люди говорят что-то вроде «тип данных `Array`» или «тип данных `Date`», но формально они не являются отдельными типами, а относятся к типу данных `Object`. Они лишь расширяют его различными способами.

Объекты в JavaScript очень мощные. Здесь мы только немного углубились в действительно огромную тему. Мы будем плотно работать с объектами и узнаем о них больше в следующих частях учебника.

✓ Задачи

Привет, object

важность: 5

Напишите код, выполнив задание из каждого пункта отдельной строкой:

1. Создайте пустой объект `user`.
2. Добавьте свойство `name` со значением `John`.
3. Добавьте свойство `surname` со значением `Smith`.
4. Измените значение свойства `name` на `Pete`.
5. Удалите свойство `name` из объекта.

решение

```
1 let user = {};  
2 user.name = "John";  
3 user.surname = "Smith";  
4 user.name = "Pete";  
5 delete user.name;
```

Проверка на пустоту

важность: 5

Напишите функцию `isEmpty(obj)`, которая возвращает `true`, если у объекта нет свойств, иначе `false`.

Должно работать так:

```
1 let schedule = {};  
2  
3 alert( isEmpty(schedule) ); // true  
4  
5 schedule["8:30"] = "get up";  
6  
7 alert( isEmpty(schedule) ); // false
```

Открыть песочницу с тестами для задачи.

решение

Просто в цикле перебираем свойства объекта и возвращаем `false`, как только встречаем свойство.

```
1 function isEmpty(obj) {  
2   for (let key in obj) {
```

```
3     // если тело цикла начнет выполняться - значит в объекте есть свойст
4     return false;
5 }
6 return true;
7 }
```

[Открыть решение с тестами в песочнице.](#)

Объекты-константы?

важность: 5

Можно ли изменить объект, объявленный с помощью `const` ? Как вы думаете?

```
1 const user = {
2   name: "John"
3 };
4
5 // это будет работать?
6 user.name = "Pete";
```

решение

Конечно, это сработает без проблем.

Объявление `const` защищает только саму переменную от изменений.

Другими словами, `user` хранит ссылку на объект. И это не может быть изменено. Но содержимое объекта менять можно.

```
1 const user = {
2   name: "John"
3 };
4
5 // Работает!
6 user.name = "Pete";
7
8 // Ошибка
9 user = 123;
```



Сумма свойств объекта

важность: 5

У нас есть объект, в котором хранятся зарплаты нашей команды:

```
1 let salaries = {
2   John: 100,
3   Ann: 160,
4   Pete: 130
5 }
```

Напишите код для суммирования всех зарплат и сохраните результат в переменной `sum`. Должно получиться 390.

Если объект `salaries` пуст, то результат должен быть 0.

решение

```
1 let salaries = {
2   John: 100,
3   Ann: 160,
4   Pete: 130
5 };
6
7 let sum = 0;
8 for (let key in salaries) {
9   sum += salaries[key];
10 }
11
12 alert(sum); // 390
```

Умножаем все числовые свойства на 2

важность: 3

Создайте функцию `multiplyNumeric(obj)`, которая умножает все числовые свойства объекта `obj` на 2.

Например:

```
1 // до вызова функции
2 let menu = {
3   width: 200,
4   height: 300,
5   title: "My menu"
6 };
7
8 multiplyNumeric(menu);
9
10 // после вызова функции
11 menu = {
12   width: 400,
13   height: 600,
14   title: "My menu"
15 };
```

Обратите внимание, что `multiplyNumeric` не нужно ничего возвращать. Следует напрямую изменять объект.

P.S. Используйте `typeof` для проверки, что значение свойства числовое.

[Открыть песочницу с тестами для задачи.](#)

решение

```
1 function multiplyNumeric(obj) {
2   for (let key in obj) {
3     if (typeof obj[key] == 'number') {
4       obj[key] *= 2;
5     }
6   }
7 }
```

[Открыть решение с тестами в песочнице.](#)



Предыдущий урок

Следующий урок



Поделиться



Карта учебника

Проводим курсы по JavaScript и фреймворкам.



Комментарии

- Если вам кажется, что в статье что-то не так - вместо комментария напишите [на GitHub](#).
- Для одной строки кода используйте тег `<code>`, для нескольких строк кода — тег `<pre>`, если больше 10 строк — ссылку на песочницу ([plnkr](#), [JSBin](#), [codepen...](#))
- Если что-то непонятно в статье — пишите, что именно и с какого места.

Присоединиться к обсуждению...

ВОЙТИ С ПОМОЩЬЮ

ИЛИ ЧЕРЕЗ DISQUS ?

Имя

 93

Поделиться

Лучшие Новые Старые

K

KingNut Academy

день назад

— 

Разбор задач, тестов и всего что поможет при прохождении собеседования на должность frontend developer (html, css, js, ts, react).
Заходи к нам в ТГ @interview_masters

0 0 Ответить • Поделиться ›



Комментарий был удален.

—

V

Vad

→ Guest

5 дней назад

— 

У тебя sum = undefined, а при сложении undefined будет NaN

0 0 Ответить • Поделиться ›



Valeriy

7 дней назад

— 

Если кому-то пригодится ❤️

```
let menu = {  
  width: 200,  
  height: 300,  
  title: "My menu"  
};  
  
function multiplyNumeric(obj){  
  for(let key in obj){  
    if(typeof(obj[key]) == 'number'){  
      obj[key] *= 2  
    }  
  }  
  return obj  
}  
console.log(multiplyNumeric(menu))
```

0 0 Ответить • Поделиться ›


```
let menu = {
  width: 200,
  height: 300,
  title: "My menu"
};

console.log("Before:");
console.log(menu);
_(menu);
console.log("After:");
console.log(menu);

function _(__){for(let _ in __)if(!isNaN(+__[_]))__[_]*=2}
```

Но за такой код собраты по цеху могут и яйсы отстрелить...

1 0 Ответить • Поделиться ›

A

Ahteryakov

20 дней назад

Ну зачем во всех задачах вывод проверок через alert? Это не удобно.
Я во всех задачах правлю его на console.log.

1 0 Ответить • Поделиться ›

T

Tazor

→ Ahteryakov

19 дней назад

Кто как хочет, так и кодит...

1 0 Ответить • Поделиться ›

B

Виктор Денисевич

21 день назад

Можно не преобразовывать свойство к Number
Math.trunc - делает это и так

```
console.log(String(Math.trunc('+1.2'))); // результат '1'
```

0 0 Ответить • Поделиться ›



Player_50B

24 дня назад edited

Остальные располагаются в порядке создания

Может я не правильно понял, но почему то все идет по "алфавитном
порядку" а не в порядке создания.

```
const myNewObject = {
  "k": "k"
```

```
    0: 0
    "1": "1",
    a: "a",
  };
  console.log(myNewObject);
  /*
  1: "1"
  a: "a"
  b: "b"
  */
```

0 0 Ответить • Поделиться ›

Е

Ернар Муратов

→ Player_50B

— 🚩

24 дня назад

for..in когда используешь тогда будет этот порядок который был расписан, там же про переборку данных в объекте рассказывали. Как понял так и объяснил)

1 0 Ответить • Поделиться ›



Player_50B

→ Ернар Муратов

— 🚩

24 дня назад

Да, спасибо.

0 0 Ответить • Поделиться › [🐦](#) [f](#) [🔗](#)

I

iconic

месяц назад

— 🚩

что значит вот это плюс?

```
sum += salaries[key];
```

0 0 Ответить • Поделиться ›

В

Вано

→ iconic

— 🚩

месяц назад

это сокращенная запись `sum = sum + salaries[key];`

7 0 Ответить • Поделиться ›

Н

Human

месяц назад

— 🚩

Объясните плиз что тут произошло в последней задаче

Есть объект и в нем данные

width: 200,

height: 300...

Мы создали функцию которая перебирает каждое значения, проверяет на число и потом умножает значения на два, но как происходит присвоения? то есть они сделали так:

```
for (let key in obj) {
  obj[key] *= 2
}
```

Это работает но как это возможно?

```
console.log(key) = width, height...
console.log(obj[key]) = 200, 300...
```

То есть `obj[key] *= 2`

[показать больше](#)

1 0 Ответить • Поделиться ›

E

EvilYou

→ Human

месяц назад edited

— 🚩

Так же, как и работает любое изменение переменной:

```
let width = 200;
width *= 2; // здесь берется старое значение переменной width и
умножается на 2, после чего в переменную записывается новое
значение.
```

Похожие выражения уже встречались ранее, никакой магии здесь нет:

```
let a = 5;
--a; // эквивалентно a = a - 1
```

3 0 Ответить • Поделиться ›

S

slawkir

месяц назад

— 🚩

Функция должна что-то возвращать. Поэтому в последней задаче после выхода из `for ... in` нужно добавить `"return obj"` или `"alert(obj)"` или `"console.log(obj)"`:

```
function multiplyNumeric(obj) {
  for (let key in obj) {
    if (typeof (obj[key]) == 'number') {
      obj[key] *= 2;
    }
  }
  return obj;
}
```

4 2 Ответить • Поделиться ›



Dima Kim

→ slawkir

месяц назад

— 🚩

функция может ничего не возвращать

```
let multiplyNumeric = (menu) => {
  for (const key in menu) {
    typeof menu[key] == "number" ? (menu[key] *= 2) : menu[key];
  }
};
multiplyNumeric(menu);
console.log(menu);
```

1 1 Ответить • Поделиться ›

■ ■

Илья

— 🚩

И

2 месяца назад

Всем привет. Очень нравится учебник, но мне хочется больше практики (ведь только так можно научиться кодить). Может ли кто ни будь посоветовать задачник или что то вроде такого. Надеюсь тут есть заинтересованные люди помимо спамщиков, заранее спасибо.

2 0 Ответить • Поделиться ›

Н**Hick Hickert**

→ Илья

2 месяца назад

— 🚩

codewars самый известный

4 0 Ответить • Поделиться ›

А**Алексей Данилюк**

2 месяца назад

— 🚩

```
let menu = {  
width: 200,  
height: 300,  
title: "My menu"  
};
```

```
function multiplyNumeric(obj) {  
for (let key in obj) {  
if (typeof obj[key] == 'number') {  
obj[key] *= 2;  
}  
alert(obj[key])  
}  
}
```

```
multiplyNumeric(menu);
```

0 0 Ответить • Поделиться ›

Загрузить ещё комментарии[Подписаться](#)[Privacy](#)[Не продавайте мои данные](#)

