# JavaScript

## Объекты

**+ Дескрипторы свойств объектов: что такое и что позволяют делать?**

Объекты содержут свойства. И кроме пары ключ-значения свойства могут быть гораздо мощнее и гибче.

Флаги свойств:

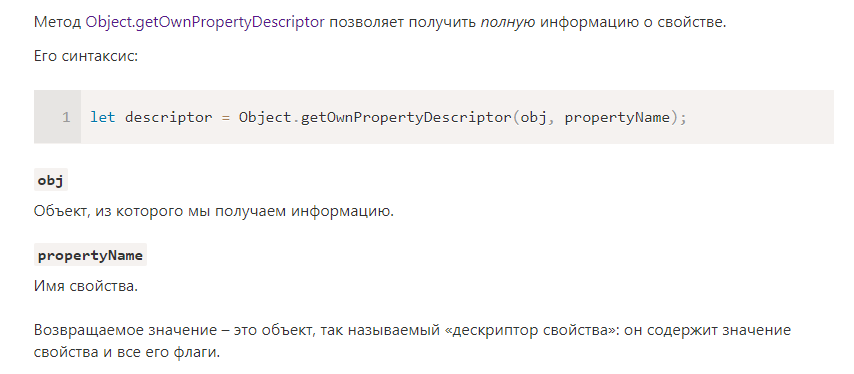
Помимо значения value, свойства объекта имеют три специальных атрибута (так называемые «флаги»).

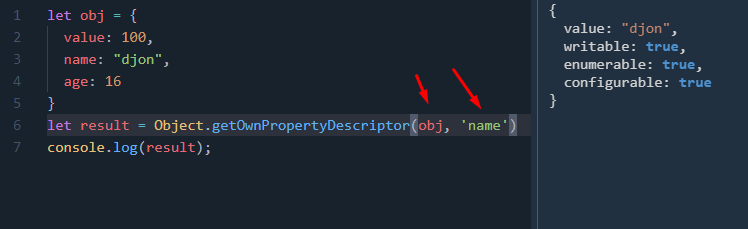
**writable** – если true, свойство можно изменить, иначе оно только для чтения.

**enumerable** – если true, свойство перечисляется в циклах, в противном случае циклы его игнорируют.

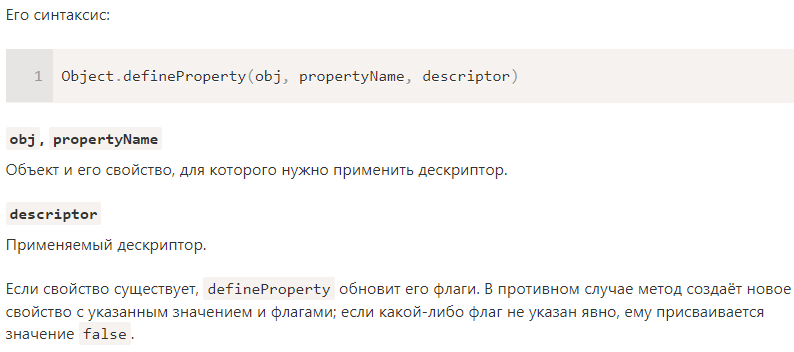
**configurable** – если true, свойство можно удалить, а эти атрибуты можно изменять, иначе этого делать нельзя.

Эти атрибуты обычно скрыты. Когда мы создаем свойство обычным способом, все они имеют значение true. Но мы можем их изменить по своему желанию.

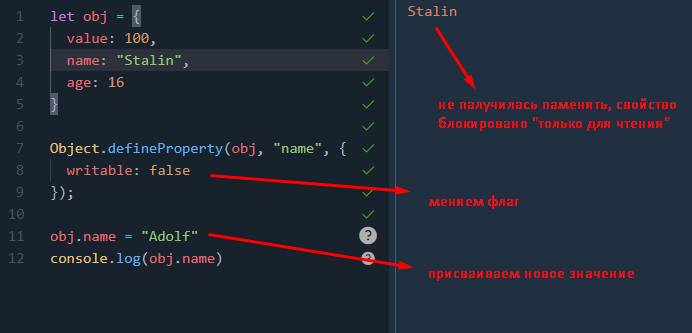


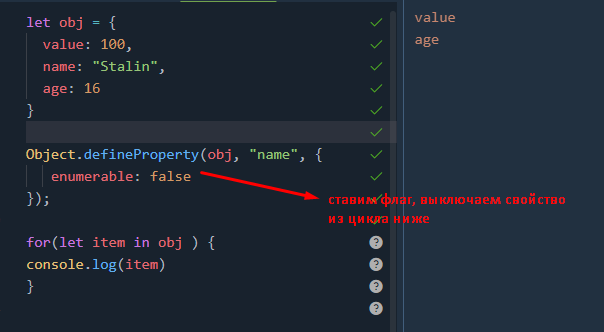
Например, вот так мы можем обратиться к свойству объекта и получить его флаги:  


Чтобы изменить эти флаги, мы можем использовать метод **Object.defineProperty**.



Зачем это нужно?

Например, чтобы **защитить свойство от перезаписи**, сделать его только «для чтения»  


Так же можно **выключить какое либо свойство из цикла**, с помощью такого флага:  


Можно намертво запечатать какое-нибудь свойство и **запретить его любые изменения**:  


Существует метод Object.defineProperties(obj, descriptors), который позволяет определять множество свойств сразу.



Чтобы получить все дескрипторы свойств сразу, можно воспользоваться методом **Object.getOwnPropertyDescriptors(obj).**

Существует еще ряд методов, которые ограничивают доступ ко всему объекту, но на практики они используются не часто.

**+ Что такое геттеры и сеттеры? Как ими пользоваться в JS нативным способом?**

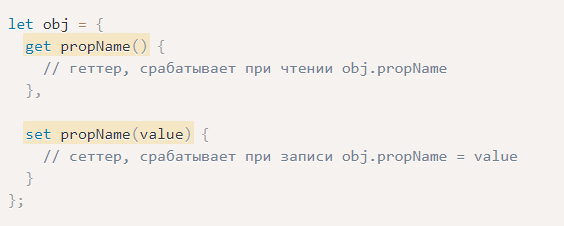
У объекта существует два типа свойств.

**Свойство-данные**

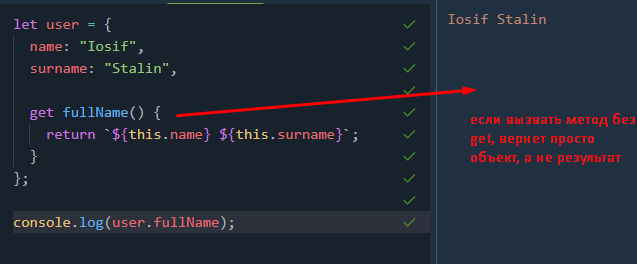
(data properties) – свойства:данные в обычном понимании.

**Геттеры и сеттеры**

Это когда свойства представлены методами. Геттер для чтения и сеттер для записи.

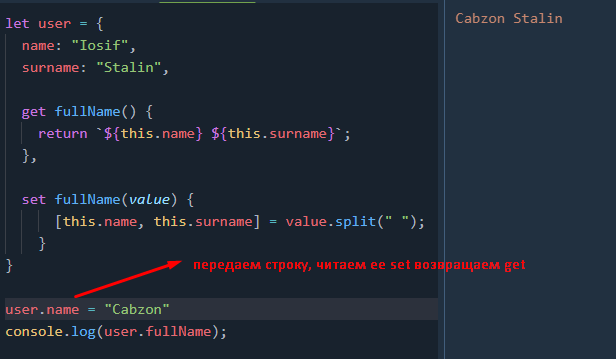


Может быть нужен, чтобы избежать дублирования информации, например:



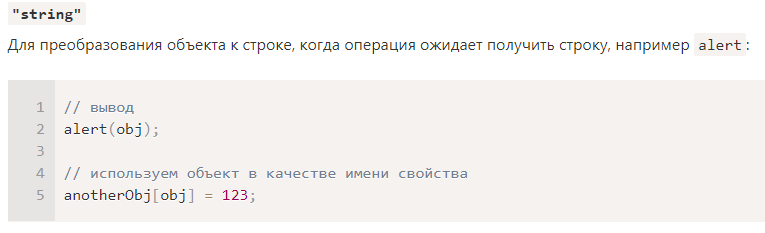
Мы не вызываем fullName как функцию, мы читаем ее как обычное СВОЙСТВО. Гетер начинает выполнять всю работу под капотом самостоятельно.

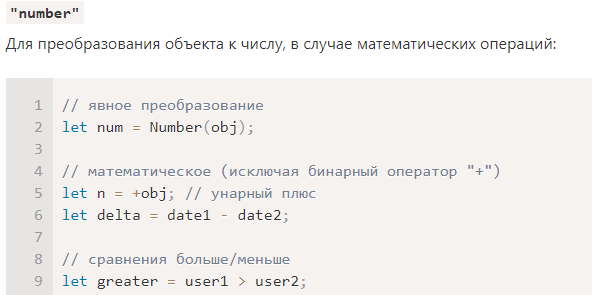
Set служит для чтения, например:

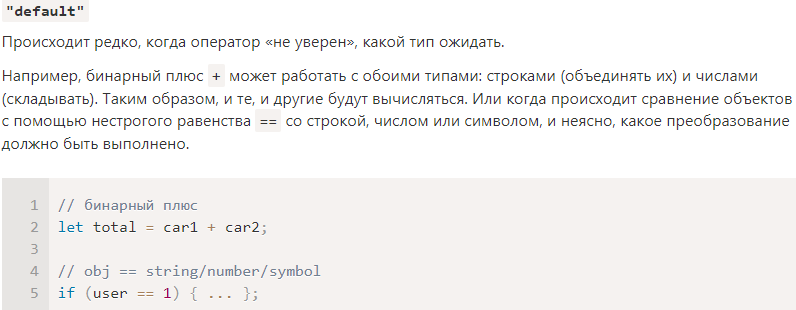


**+ Приведение объектов к примитивам: когда происходит и как работает?**

Некоторое стандартные методы преобразуют к примитивам самостоятельно. Например, alert.

Существуют три варианта преобразования: 





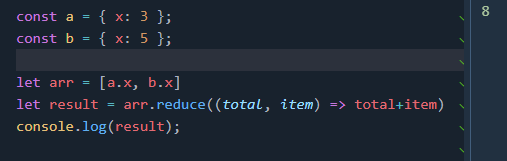
**+ Как сделать так, чтобы было возможно сложение двух объектов?**

const a = { x: 3 };

const b = { x: 5 };

console.log(a + b); // 8

Например, reduс-ом



**+ Как сделать так, чтобы следующее выражение вернуло `true`?**

**(c==1 && c==2 && c==3)**

#### Ресурсы

+ [Object to primitive conversion](https://javascript.info/object-toprimitive)

+ [ToPrimitive алгоритм в спецификации](https://www.ecma-international.org/ecma-262/9.0/index.html#sec-toprimitive)

## Массивы

+ Можно ли прервать обход элементов в методе `forEach`?

+ Можно ли с помощью метода `indexOf` определить позицию подмассива в массиве?

+ Как строку преобразовать в массив? Можно ли для строк применить методы массивов?

## Функции

+ Что такое "всплытие" объявления переменных, и в каком случае у функций тут появляются особенности?

+ Влияет ли директива `use strict` на "всплытие"? Как избежать "всплытия" переменных?

+ Что такое IIFE (immediately invoked function expression)? Какие есть способы написать IIFE и почему это так работает?

+ Как сделать так, чтобы при вызове immediately invoked function expression сохранялся родительский контекст?

+ Как использовать стандартные методы массива для ненастоящих массивов (например, вызвать `map` для `arguments`) и почему это работает?

#### Ресурсы

+ [Immediately-Invoked Function Expression (IIFE)](http://benalman.com/news/2010/11/immediately-invoked-function-expression/)

+ [How does Array.prototype.slice.call() work?](https://stackoverflow.com/questions/7056925/how-does-array-prototype-slice-call-work)