Одна из лучших особенностей объектов – это то, что мы можем хранить функцию как одно из свойств объекта. Но из-за этого объекты «тяжелее» примитивов. Они нуждаются в дополнительных ресурсах для поддержания внутренней структуры.

Вот парадокс, с которым столкнулся создатель JavaScript:

* Есть много всего, что хотелось бы сделать с примитивами, такими как строка или число. Было бы замечательно, если бы мы могли обращаться к ним при помощи методов.
* Примитивы должны быть лёгкими и быстрыми насколько это возможно.

Выбранное решение:

1. Примитивы остаются примитивами. Одно значение, как и хотелось.
2. Язык позволяет осуществлять доступ к методам и свойствам строк, чисел, булевых значений и символов.
3. Чтобы это работало, при таком доступе создаётся специальный «объект-обёртка», который предоставляет нужную функциональность, а после удаляется.

Каждый примитив имеет свой собственный «объект-обёртку», которые называются: String, Number, Boolean, Symbol и BigInt. Таким образом, они имеют разный набор методов.

Как работают метды примитивов?

Например:

let str = "Привет";

alert( str.toUpperCase() ); // ПРИВЕТ

1. Строка str – примитив. В момент обращения к его свойству, создаётся специальный объект, который знает значение строки и имеет такие полезные методы, как toUpperCase().
2. Этот метод запускается и возвращает новую строку (показывается в alert).
3. Специальный объект удаляется, оставляя только примитив str.

Формально эти методы работают с помощью временных объектов, но движки JavaScript внутренне очень хорошо оптимизируют этот процесс, так что их вызов не требует много ресурсов.

Получается, что примитивы могут предоставлять методы, и в то же время оставаться «лёгкими».

Движок JavaScript сильно оптимизирует этот процесс. Он даже может пропустить создание специального объекта. Однако, он всё же должен придерживаться спецификаций и работать так, как будто он его создаёт.

**Конструкторы String/Number/Boolean предназначены только для внутреннего пользования**

Некоторые языки, такие как Java, позволяют явное создание «объектов-обёрток» для примитивов при помощи такого синтаксиса как new Number(1) или new Boolean(false).

В JavaScript, это тоже возможно по историческим причинам, но очень **не рекомендуется**. В некоторых местах последствия могут быть катастрофическими.

Например:

alert( typeof 0 ); // "число"

alert( typeof new Number(0) ); // "object"!

Объекты в if всегда дают true, так что в нижеприведённом примере будет показан alert:

let zero = new Number(0);

if (zero) {

// zero возвращает "true", так как является объектом

alert( "zero имеет «истинное» значение?!?" );

}

С другой стороны, использование функций String/Number/Boolean без оператора new – вполне разумно и полезно. Они превращают значение в соответствующий примитивный тип: в строку, в число, в булевый тип.

К примеру, следующее вполне допустимо:

let num = Number("123"); // превращает строку в число

**null/undefined не имеют методов**

Особенные примитивы null и undefined являются исключениями. У них нет соответствующих «объектов-обёрток», и они не имеют никаких методов. В некотором смысле, они «самые примитивные».

Попытка доступа к свойствам такого значения возвратит ошибку:

alert(null.test); // ошибка

**[Можно ли добавить свойство к примитиву?](https://learn.javascript.ru/primitives-methods" \l "mozhno-li-dobavit-svoystvo-stroke)**

let str = "Привет";

str.test = 5; // (\*)

alert(str.test);

В зависимости от того, используется строгий режим (use strict) или нет, результат может быть:

1. Ошибка (strict mode)
2. undefined (без strict)

что происходит в строке кода, отмеченной (\*):

1. В момент обращения к свойству str создаётся «объект-обёртка».
2. В строгом режиме, попытка изменения этого объекта выдаёт ошибку.
3. Без строгого режима, операция продолжается, объект получает свойство test, но после этого он удаляется, так что на последней линии str больше не имеет свойства test.

**Данный пример наглядно показывает, что примитивы не являются объектами.**

Они не могут хранить дополнительные данные.