# JavaScript — Lurkmore

**Внимание! Статья-детектор!**

Одним из побочных эффектов от прочтения этой статьи является так называемый butthurt.
Если вы начнёте ощущать боль в нижней части спины, следует немедленно прекратить дальнейшее чтение и смириться с фактом, что вы — **мечтатель, и всё ещё верите в сказочки про доброго Дядю, который возьмёт вас на работу программистом**.

**I see what you did there.**

Информация в данной статье приведена по состоянию на эпоху популярности jQuery. Возможно, она уже безнадёжно устарела и заинтересует только слоупоков.

**В эту статью нужно добавить как можно больше информации о состоянии современного веба.**

Также сюда можно добавить интересные факты, картинки и прочие кошерные вещи.

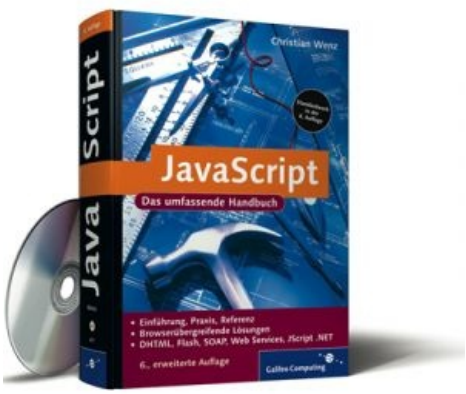
«Если бы строители строили дома так же, как программисты пишут программы, первый залетевший дятел разрушил бы всю цивилизацию. »

— *Gerald M. Weinberg*

«We are utterly spoiled by programming every day in such a joyful language as Python. JavaScript is our punishment. »

— *Harry J.W. Percival. Test-Driven Development with Python*

**JavaScript** (он же *JS, Яваскрипт, LiveScript, ECMAScript, Жабаскрипт*) — поделие Брендана Айха, чуть более чем повсеместно использующееся для написания костылей к тому, что не сделать с помощью HTML и CSS[1]. Если где-либо на веб-странице что-либо выдвигается, скукоживается, моргает, перекрашивается, перекашивается, вырвиглазно всплывает, икает, гыгыкает, хрюкает, пукает и весело ржот за кадром — и всё это при отсутствии Flash или CSS, — это и есть JavaScript.




Коробочный продукт

Своим появлением обрушил порог входа в профессию «программиста» практически до нуля, причём ноль этот абсолютно круглый и плоский. Собственно, человек, практикующий JavaScript, иногда даже не зовется программистом, а приравнивается к верстальщикам. Толпы народу, не имеющие ни малейшего представления об указателях и рекурсии, не знающие даже основных алгоритмов, называющие переменные русской латинкой — спокойно пишут на языке, где правил оформления кода нет как не было.

Язык поддерживается всеми браузерами, способен выполняться и на серверах, на нём уже пишут складской учёт, воруют данные с Яндекс-маркета, зачисляют платежи, следят за таксопарком, сдают блядей в аренду — он стал де-факто стандартом в отрасли. Как и PHP, занимает нишу Фортрана.

Также является первопричиной жырного троллинга про светлое коммунистическое будущее Web 2.0, когда наступит вендекапец и весь мир наконец-то переедет во Всемирную Сеть. И только гринпис против него, так как он может стать причиной преждевременного глобального потепления из-за кривых скриптов, прожигающих впустую такты процессора на компьютерах хомяков.

## ИЧСХ

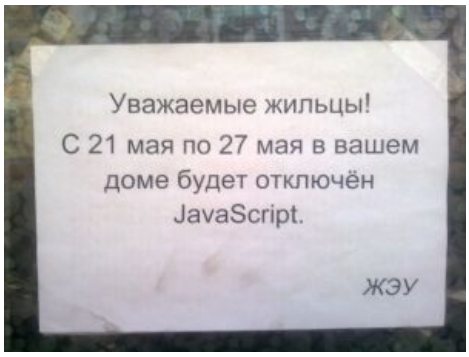
«Сто лет бы мне не видеть этих строчек —

За каждой вижу чью-нибудь судьбу, И радуюсь, когда статья не очень — Ведь всё же повезёт кому-нибудь!

»

— *Владимир Высоцкий*

При том, что JavaScript позволяет писать жуткий говнокод, не задумываясь о правилах видимости переменных, количестве аргументов и прочих тонкостях строгих языков, *он же* знаменателен тем, что позволяет создавать очень и даже очень красивые вещи, (см. всяких призеров FWA, three.js, Chrome Experiments, HighCharts, Perfect Widgets). Так происходит потому, что *он подстраивается под того, кто на нём пишет*.



Ахтунг!

Если это, к примеру, похапэшник, пришедший подхалтурить — он всегда готов помочь и спешит на помощь. И наш герой впадает в ступор, узнав, что точки в яваскрипте служат не для склеивания строк, а для склеивания непонятно чего непонятно с чем. И вот уже его лапы тянутся к готовым библиотекам вроде jQuery, Ext JS, и тогда…

Красивые вещи, таким образом, создаются совсем другими людьми.

# Мифы

## Это игрушечный язык

Устоявшийся миф. И да, на нём таки пишут игрушки и даже играют в них. Просто появление HTML5 позволило выводить на экран элементы через общий холст `Canvas()`, а не ебаться с передвижением элементов. И то, что их пишут — никакой не миф: достаточно взглянуть на портированную в браузер зачётную игру Command & Conquer. А ещё множество аркадных игрушек. И даже Майнкрафт.

А ещё — эмулятор Qemu и по крайней мере одна Java-машина.

## Он работает только в браузерах

Не только. JavaScript — скриптовый язык. Будет интерпретатор — будет и выполнение творчества, написанного на нем. Можно хоть в консоли, но не нужно рядовому пользователю (в то же время — очень даже нужно спецам для некоторых задач — консольные утилиты на Node.js, некоторые макросы для Sublime Text и др.). По крайней мере, две операционные системы для мобильных устройств, HP webOS и Firefox OS, используют JavaScript как язык системного программирования: на нём написаны их оболочки. Скрипты транслируются в байт-код, и он выполняется.

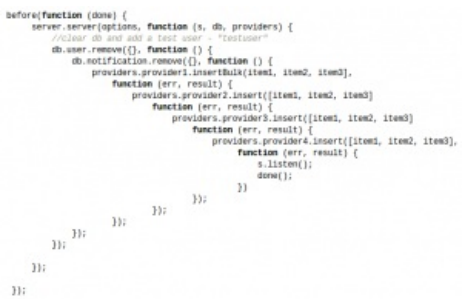
В Wiindows JavaScript можно использовать для написания скриптов (работа с системой осуществляется через ActiveX), юные хацкеры могут вам насоздавать на жестком диске 100500 файлов (поэтому не запускайте файлы с расширением .js). Можно создавать hta-приложений, а также почти полноценные приложения в версиях Wiindows >= 8. Для рендеринга страниц в них используется движок Trident, который используется IE. Ишак же у всех верстальщиков вызывает ненависть (на сайте Вконтакте долгое время при заходе с IE было перенаправление на страницу /badbrowser.php).

Как особое извращение существуют интерпретаторы, написанные на JavaScript. Например, CoffeeScript, который помимо того что позволяет экономить на скобках, еще позволяет использовать полноценные классы. Быдлокодеры, пришедшие из пыхепе, так и не смогли понять всю мощь прототипного наследования и рискуют сдохнуть от сахарного диабета и собственного пафоса. Другие же быдлокодеры не смогли смириться с динамической типизацией и запилили TypeScript. Гугол решил не отставать от мелкософта и создал хуету под названием Dart. Кроме перечисленных языков, существуют также HaXe и C++/C.

Многие продукты Adobe, например, тот же Photoshop, имеют JavaScript API, которое позволяет работать не только с документами, но также и с файловой системой и сетью. Подробнее об этом можно почитать на сайте адобы.

JavaScript встроен как скриптовый язык в Unity3D (там он больше напоминает ActionScript), использовать его можно и вместе с фреймворком Qt от расовых норвежских троллей из Trolltech. Что касается последнего, то там он существует сразу в нескольких вариантах: QScript, QML и собственно JavaScript. QScript мало отличается от своего прародителя в плане синтаксиса, а вот QML представляет собой какую-

то смесь из CSS/JSON-нотации и собс-но JavaScript, но, тем не менее, очень перспективный и мощный язык.



Ехал }); через });

И, например, такая абсолютно лютая серверная вещь как Node.js, как утверждает рекламщики, даст фору всяким разным PHP и ему подобным. Данное утверждение довольно спорное, потому как все бенчмарки проводятся с базовым примером, что на сайте ноды. Как нетрудно догадаться, функционал Апача во много раз превосходит данное говноподелие, но велосипедостроителям нравится асинхронность и макаронный код, когда callback callback'а callback'ом ебет *(спойлер:* где второй callback, в свою очередь, ебет товарища, который до сих пор не изволил покушать Promise, приготовленный на генераторах*)*. Чтобы представить более наглядно, можно привести такой пример: есть два одинаковых человека, одинакового роста, веса и т.д. И они соревнуются в забеге на короткую дистанцию. Только вот один из них в каске, противогазе и бронежилете *(спойлер:* Apache*)*, а другой налегке. Тот же Go делает в бенчмарках ноду всухую *(спойлер:* правда, чуть хуже, чем Си делает Джаву; да и в Java же типизация! OH-SHI--*)*.

### JavaScript — это Java

## Java ? JavaScript / 0 : goto begin;

begin: Непосвящённые считают сабж языком, произошедшим от великой и ужасной Жаббы, в которой последовательно портили синтаксис. Это объясняется нежеланием не слишком пытливого ума почитать хотя бы педивикию, а вместо этого делать умозаключения, основываясь на слуховом и зрительном восприятии названий этих двух языков. На самом деле, вот так выглядит «Hello world» в этих языках:

**Java:**

```
public class HelloWorld {
    public static void main (String[] args) {
        System.out.println ("Hello, World!");     /* ООП: класс и метод в нём */
    }
}
```

**JavaScript:**

```
window.onload = function () {
    with (document) {
        writeln ("Hello, World!");                /* да, это замыкание; дальше соображай сама */
    }
}
```

Или вот так:

```
document.write('Hello, World!');
```

Если Жаба простая — большая жирная гипножаба, в веб-приложениях в основном не вылазит с сервера и выполняет там кучу работы, то жаба скриптовая — совсем маленькая, зелёненькая, почти лягушка, обитает, как правило, на стороне клиента, то есть на 90% в браузерах, сидит там и может поднасрать Анонимусу. Авторы книг по яваскрипту в первых же главах предупреждают читателя, что, мол, дорогой читатель, изволь уяснить себе, что перед тобой два разных языка. Сходства, конечно, есть, но они носят сугубо внешний характер. Некоторые фреймворки рассчитаны на сотрудничество этих двух языков. См. ZK, например.

И это, в замыкание над безымянной функцией в примере выше попадает переменная «document», если чо.

### JavaScript — лёгкий язык

А вот и нет. Программиста, работающего с классическими классовыми языками программирования, вырубает его прототипно-ориентированность, конструкторы, своеобразное наследование и ещё куча вещей, рвущих шаблоны. Ну а что ты хочешь от языка, на разработку которого было выделено 10 дней?

### JavaScript — это JQuery

JQuery — самая популярная и в большинстве случаев не нужная библиотека для JS. Большинство JS-быдлокодеров легко отличить по следующему признаку: если вы предложите им написать что-либо без использования JQuery, то они сразу откажутся.

Сам JQuery представляет собой тяжёлую коллекцию тормозного синтаксического сахара и очевидных даже для начинающего конструкций. Особенно доставляет функция $(), которая объединяет в себе все возможные случаи поиска элемента по различным идентификаторам, признакам, а также просто создание элемента из HTML-кода.


Особую драму составляют так называемые «плагины для JQuery». По сути это просто библиотеки, создающие свойство в объекте $ и использующие функциональность JQuery. Из-за того, что авторам этих плагинов религия запрещает инжектировать их в объект window и использовать стандартизированное DOM API, дописав некоторые функции несколькими строчками кода, пользователям этих говнобиблиотек приходится либо их чистить, либо тащить громадный JQuery, что порождает тонны ненависти и срач «Vanilla JS vs JQuery vs говнофреймворки».

## Вся голая Правда

> «Я не знаю, как у вас.
>
> А у нас в Японии. Семь врачей пизду смотрели. Нихуя не поняли.
>
> »

— *RSDN*

Жабаскрипт 1.1 был написан в Netscape за 15 дней и никто не переживал из-за глобальных переменных или тем более из-за того, что данные в скриптах представлены как строки. Достаточно было и того, что на страницах выполнялись какие-то действия при нажатии мышкой на какие-нибудь кнопки. В июне бородатого 1997 года он был отправлен на стандартизацию в комитет ISO под псевдонимом ECMAScript, и комитет принял стандарт.

Изначально JS был императивным языком, в котором все действия выполнялись построчно, и, несмотря на его название, не имел ничего общего ни с ООП, ни с Java.

Одно время находился под влиянием Delphi, откуда позаимствовал ключевое слово «var» (оно уходит корнями в Паскаль, Алгол и Кобол), а сейчас единственно верной считается версия ECMA-262 Edition 5.1, где начался закос под Scheme.

| Название | Дата | Описание | Ссылка |
|---|---|---|---|
| ECMA-262 | июнь 1997 | Оригинал стандарта основан на жабаскрипте 1.1 | PDF |
| ECMA-262 Edition 2 | август 1998 | Вторая ревизия | PDF |
| ECMA-262 Edition 3 | декабрь 1999 | Третья ревизия на жабаскрипте 1.5 | PDF |
| ECMA-262 Edition 5 | декабрь 2009 | Пятая ревизия эрата мечта кулхацкера | PDF |
| ECMA-262 Edition 5.1 | июнь 2011 | Кошерная ревизия | PDF |
| ECMA-357 | июнь 2004 | ECMAScript для XML она же E4X (уст., к сож.) | PDF |

В последний раз JavaScript оказался в центре всеобщего внимания после появления Google Maps, когда каждый получил возможность посмотреть на свой дом и двор со спутника. Вдруг все узнали, что помимо всяких там двигающихся текстов, можно использовать JS для взаимодействия с сервером, обновляя не всю страницу, а только её небольшую часть. Только представьте себе те адские мучения, которые настигли бы несчастного хомячка, если бы ему нужно было обновлять страничку для просмотра каждого нового участка карты или смены её масштаба!

С тех пор технологию AJAX (*Asynchronous JavaScript and XML*) юзают все кому не лень, и везде, где только можно. И нельзя, ибо какой ты без Ajax, блеать, Web 2.0-кун? А модные ребята, которым особенно не лень, юзают WebSocket, «use asm», прочий «use strict» и всякие мутки хроноса, типа WebGL, а то и вовсе WebCL.

## Особенности языка

> «Тяжела и неказиста жизнь простого программиста. »

— *Народное*

Гибкость его необычайна. Функции являются объектами, объекты — функциями, строки — переменными, переменные — строками, типы — объектами и т.д. и т.п. Всё это, как уже говорилось выше, имеет две стороны.

Она ебётся, а ты нет.

Скрипты компилируются не *до* исполнения, а в процессе, и не целиком, а только те функции, что были вызваны — так что ошибки вроде «Bad command or file name» и «No ROM Basic» стали обычным делом. Гуглонетскейпам этого показалось мало, и вот уже в языке появилась возможность менять объекты прямо во время выполнения, добавляя к ним любые функции, вне зависимости от исходного класса объекта. Кстати, классов тоже нет: вместо них — прототипы будущих объектов, и их тоже можно менять.

Ну и наконец, каждая переменная может быть равна *true*, *false*, *null*, *undefined*, содержать ноль или пустую строку, а также указывать на объект, массив или список… Следующий код вначале скажет «false», поскольку первая переменная равна «false», — а затем «WTF»:

```
var simpleType = false,
    objectType = new Boolean( false )

if ( simpleType)
    alert( 'Этот код никогда не выполнится' )
else
    alert( 'Да, false. А вы ожидали чего-то другого?' )


if ( objectType )
    alert( 'O_o WTF? Быдлокодер-кун негодує!' )
else
    alert( 'Этот код никогда не выполнится…' )
```

Переменная objectType — объект, рождённый конструктором Boolean, любой объёкт при приведении типа превратится в true.

А разгадка проста...
Конструкторы String/Number/Boolean – только для внутреннего использования.

Тащемта, можно невозбранно создавать объекты для примитивов и вручную, например new Number. Но в ряде случаев получится дико бредовое поведение, например.

```
alert( typeof 1 ); // "number"
alert( typeof new Number(1) ); // "object" ?!?
```

Или, еще фимознее:

```
var zero = new Number(0);
if (zero) { // объект - true, так что alert выполнится
  $alert( "Шта? число ноль -- true?!?" );
}
```

Собсна, поэтому в явном виде new String, new Number и new Boolean никогда не вызываются, ибо нехуй!

Это привело к появлению надстроек над JavaScript'ом вроде HaXe и TypeScript, и теперь уже языки со строгой типизацией транслируются в JavaScript, а оттуда уже в байт-код — и тогда, по крайней мере, таких ошибок не возникает. Так что да, многое зависит от того, *кто* пишет на JavaScript и *как* он пишет — и мало что зависит от самого языка.

При сравнении объектов стоит забыть про оператор нестрогого сравнения == и везде использовать строгое ===. Почему? — А вот примерчик:

```
[42] == '\t42'           // true
'\t42' == new Number(42) // true
new Number('\t42') == 42 // true
42 == ['\t42']           // true
```

На практике, всё упирается в условия работы и уровень заработной платы:

> — Есть фирмы, есть решения, есть люди работающие на них…
>
> — Именно так. Есть загнивающие быдловские конторишки вроде того гадюшника, где тебя дрючат манагеры, в которых работает никчемное быдло (и у дверей стоит очередь такого же быдла, желающего поработать вместо тебя), а есть настоящие, крутые конторы, в которых пользуются правильными, серьёзными подходами к программированию. Эти конторы торгуются на NASDAQ, а никак не твои быдлятники.

— *Луговский об экономике*

# JavaScript-библиотеки

## «Доширак, велосипед, зеркалка и Интересные Проекты — вот анатомия программистишки! »

— ***О программистишках***/33248

В целях упрощения процесса генерирования очень нужных и важных графических хреновин и прочего, для простых смертных были написаны туевы хучи библиотек типа jQuery, Mootools и т. д. Позволяют в три строчки понаписать всякие там исчезалки/растворялки/скрывалки/раскрывалки части страничек. В тех же трёх строчках можно уместить и какие-нибудь мудрёные Ajax-запросы с обработкой полученного ответа.


Mario негодує

JS-библиотеки, таким образом, позволяют сочетать несовместимое: писать скрипты *быстро*, *качественно*, и одновременно с тем ещё и *недорого*.

### Быстро

Это значит быстро. Это значит затратить десять минут на то, с чем советский НИИ Кибернетики трахался бы полгода.

### Качественно

Это значит… это… это…

Доступность быдлокодерам и прочим отбросам от программирования уже сделала своё дело: каждый третий напичканный сабжем сайт перманентно жрёт 100% процессора. Хочется взять и уебать всем этим мудакам с падающими снежинками и прочим говном, тормозящим не только браузер, но и MP3-проигрыватели, IRC-чаты и вообще всё!!!

### Недорого

А ещё одно направление открыли и показали нам авторы ExtJS: потрясающе красивой библиотеки, позволяющей херачить диалоговые окошки в точности так, как это было в MS Access и Delphi в 1994—1995 году. Далёкий отголосок истерии «Windows'95 всех пиписьками закидает, смело покупайте Windows'95!». Зарплаты упали соответственно. И упали они навсегда: «web-программист» — это не профессия, это диагноз.

Механизм рыночной экономики безжалостен к её участникам, маленьким людишкам. Много народу владеет инструментом — зарплаты снижаются, мало кто знает — повышаются. «Вот тебе рубль за то, что гайку закрутил, — и ещё трёшка за то, что знал, какую гайку крутить». Никогда не будут разработчики диалоговых окошек получать столько же, сколько получают разработчики бизнес-логики, СУБД и операционных систем… никогда! Язык может быть прекрасен, но он встроен во все браузеры — и этим всё сказано.

Эта тенденция имеет и неочевидную сторону: в то время как ~~космические корабли бороздят просторы~~ *количество* людей, владеющих языком растёт (и, соответственно, зарплата падает), *качество* же погромистов (и их ~~гов~~продукта) развивается по совсем другому, но все же известному принципу: хороших — немного.

По этому если аутсорсинговая контора набирает джунов[2] за $300, то синьйор/техлид легко просит в 13 раз больше. ИЧСХ, получает их, так как за время приготовления 1 (одной) чашки кофе он легко объяснит джуну, почему тот — джун *(спойлер:* и почему после его кода люди сливают воду и моют руки*)*.

И да, бизнес-логика таки пишется на JS (не только для proof of concept) — Node.js, ентерпрайз на Angular, ExtJS и других монстрах.

## jQuery

Лютая, бешеная библиотека для JavaScript, на 95% состоящая из самопильной функции $(), которая найдёт тебе что угодно, а если не найдёт, то вернёт ещё больше методов и функций. Данное творение стоит у каждого быдлокодера, позволяя ему прожить денёк-второй на «заработанные» деньги.

Как и любая библиотека, создана, чтобы писать коротко и понятно то, что обычно пишут длинно и запутанно. Но настоящего криворукого программиста это не остановит: он и на jQuery напишет так, что только браузер прочитает.

Да, на jQ реализуется многое из того, что ~~не может обычный JavaScript~~ слишком долго и лень пилить на чистенькой лягушке. А причина одна: особое, упоротое нахождение элементов:

```
$(document).ready(function() {     // (0)
    $('#IDEбучегоDIV').draggable({ // (1)
        containment:'html',        // (2)
        handle: '#head'
    });
```

Да, короче кода не нашлось. Что же тут произошло?

0. Проверяем таким черезжопным способом, вместо DOMContentLoaded-события на всё окно, что намного удобнее. По загрузке DOM функция продолжится.
1. находим элемент с ID «IDEбучегоDIV». Стоит сказать: фича нахождения — годнейшая: ищет в DOM как через getElement*, так и по querySelector, но… То, что нашли — заворачиваем в дохуя ненужных методов и свойств. То, что методы, например, плавного перемещения по окну и пропадания в бездну, нахуй могут быть не нужны для одного мигания кнопочки или получения текста без HTML-тегов, разработчики, конечно, не подумали. Да и зачем? И вообще, это задумка такая, чтобы очистить мир от притормаживающих браузеров.
2. Все параметры передаются в объекте. Зачем? Иногда так и правда удобнее, но жутко бесит тех, кто хотел юзать обектную ориентированность Лягушки на минимум.

Алсо, про мир я не так просто сказал. Сейчас jQuery стоит не то что у каждого третьего, а у каждого второго разработчика. Даже Микромягкие, моззиловцы, даже Википедия себе на сайт подключили, что уж говорить о быдлокодерах, желающих быстро заработать? Каждый третий сайт из результатов любого запроса в Гугл имеет подключение к jQuery. Что же будет, если в один прекрасный день сайт библиотеки упадёт? *(спойлер:* Соснут все, кто не умеют в использование локальной копии библиотеки. Кстати, из Китая гугл никак не доступен, так что там все сайты, тянущие JQuery с гуглохранилища, как раз таки и не работают нормально.*)* .

Это не хорошо и не плохо, но будь готов к тому, что в нагрузку к языку придётся разобраться и с этой библиотекой.

С введением Selectors API JQuery несколько потерял свою актуальность. Для обращения к элементам, используя CSS-селекторы, можно применять такой вот самопальный сниппет:

```
function $$(selector, node) {
  // NodeList имеет только метод item и свойство length. Array предоставляет гораздо более широкий набор функционал.
  return [].slice.call((node || document).querySelectorAll(selector));
}

// Та же функция на ES6 выглядит намного элегантнее
const $$ = (selector, node = document) => [...node.querySelectorAll(selector)];
```

Тут важно отметить тот факт, что без JQuery методы Element.querySelector и Element.querySelectorAll не появились бы. JQuery произвел огромное влияние на развитие языка JavaScript. По состоянию на 2017 год часть функционала JQuery реализуется на CSS3, часть на EcmaScript 2015, и лишь малая часть функционала как-то еще востребована. В наши дни поддерживать унылое говно в виде IE 9 нигде не требуется, а нормальные браузеры типа Chrome и Огнелиса умеют обновляться автоматически. На долю же IE приходится малая доля рынка, такая, что про их существование можно забыть (тем более сейчас Microsoft заменяет ослика на Edge на движке того же Хрома). И JQuery с его с кроссбраузерностью уже никому не нужен, потому как и FireFox и производные от Chrome (браузеры на основе Chromium типа ненавистного Amigo) следуют стандартам в отличие от IE (тут ради справедливости нужно сказать, что в 2006, 2007, 2008 и т.д., когда IE еще имел какую-то популярность никаких стандартов и не было).

## Достоинства? Недостатки?

> «Там, значит, как на бугор подымешься, да, там пашня. Правда, там, сейчас, пожалуй, размокло. Да, однако, обратно же, не потопнешь. Вспахано-то неглыбко, я сам под зябь пахал, да. В прежние-то времена, конечно, пахали поглыбже. Поскольку земля своя. Теперича всё колхозное. Теперича хочь так паши, хоть эдак… плата одна — ВО!!! »
>
> *— Жизнь и необычайные приключения солдата Ивана Чонкина*

Единственный недостаток — серые толпы, наводнившие отрасль после краха 2008 года, когда офисный планктон повыбрасывали со всех так называемых работ, а новых не появилось. Это офисный JS-погромист, а есть ещё погромист подвальный: гибрид макаки и павиана, бегающий с голой задницей по саванне, страдающий от постоянных оскорблений, унижений и обвинений, страха смерти, зависти и подавленной

злобы. Такой вот неудачник, юродивый, склонный к предательству мелкий пакостник, а в заголовке большинства скриптов можно спокойно ставить «вы ещё поебётесь».

В результате наш герой сполна получает в компаниях, далёких от производства программного обеспечения, где он и шароёбится полный день, с утра отмечаясь на доске опозданий. Выхода, кстати, из тупика нет и не будет. Он-то думал, что распространённость языка обеспечит ему занятость и высокий социальный статус. Ага, отчасти так и есть: чем больше народу «знает язык» (или делает вид, что знает), тем сильнее становится у роботодятелов желание их всех нанять на работу. В этом случае исполнитель становится идеальным конторским служащим: его в любой момент можно выбросить за дверь и на его место взять другого.

Это если о недостатках. Достоинств же в этом случае вовсе нет.

## Примеры кода

### «Есть многое на свете, друг Горацио! Что и не снилось нашим мудрецам! »

— *Гамлет*

Создаём класс и два объекта этого класса:

```
function Obj () {
    this.id = 5;           // У нас будет поле "id", по умолчанию оно равно "5"
}
Obj.prototype.id_pr = 3;   // У всех объектов этого класса будет поле "id_pr", равное "3"

var obj1 = new Obj ();
var obj2 = new Obj ();

obj1.id = 3;               // В первом объекте поле "id" пусть будет равно "3"
Obj.prototype.id_pr = 4;   // И пусть теперь у всех объектов поле "id_pr" будет равно "4"

alert ([
        'obj1.id = ' + obj1.id,
        'obj2.id = ' + obj2.id,
        'obj1.id_pr = ' + obj1.id_pr,
        'obj2.id_pr = ' + obj2.id_pr
        ].join('\n'));
```

На что браузер скажет:

«obj1.id = 3
obj2.id = 5
obj1.id_pr = 4
obj2.id_pr = 4 »

— *Косая Горящая Лиса*

А разгадка одна - id_pr не является полем объектов obj1/ obj2, а полем объекта-прототипа, к которому ссылаются объекты класса Obj. Курите прототипное наследование.

К любому из существующих объектов можно добавить новые свойства:

```
// Добавляем метод reverse для строк
String.prototype.reverse = function () {
  return this.split('').reverse().join('');
}

console.log('Hello, world!'.reverse()); // !dlrow ,olleH
```

Можно прицепить их и к классам объектов, хотя в JavaScript нет таких же точно классов, как во всём остальном ООП: были сделаны *прототипы будущих объектов*, например, глобальный объект **String** и тому подобное. И здесь метод reverse, возвращает «перевернутую» строку. Данный метод будет добавлен к каждой строке.

Стоит оговориться, что добавление новых свойств к встроенным объектам считается дурным тоном, если только это не делается ради обратной совместимости с другими бр[о|а]узерам. Например, добавление метода **trim** к **String**, который является частью текущего стандарта, но не реализован, скажем, в 6,7,8,... ишаке.

А вот пример замыкания, когда во время выполнения над функцией создаётся объект-обёртка, и теперь функция может работать с переменными, которые там оказались — и так и будет с ними работать, вне зависимости от того, когда и где и кем она была вызвана:

```
function test (i) {
    return function () {
        return i * i;
    }
}

var a = test (2);          // Теперь в объекте "a" поле "i" будет равно "2"
var b = test (3);          // А в объекте "b" точно такое же поле "i" будет равно "3"

alert ('a() = ' + a() + '\n' + 'b() = ' + b());

onunload = function () {   // А это против утечек памяти
    a = null;
    b = null;
}
```

На что браузер скажет:

«a() = 4
b() = 9 »

— *Лучший в мире отображатель страниц*

Замыкания впервые появились в языке Лисп. В данном случае замыкание — то же, что объект с единственной точкой входа. Внутри него остаются данные: в примере выше — то, что было ему передано в переменной «i».

Пример иллюстрирующий неожиданное поведение языка*(спойлер:* мало кто осиливает почитать спецификацию EcmaScript*)*:

```
> 4 + '2'
"42"
> 4 * '2'
8
```

# Разное

«И сердце бьётся раненою птицей

Когда начну свою статью читать, И кровь в висках так ломится, стучится, Как мусора, когда приходят брать.

»

— *Владимир Высоцкий*

### Здесь всё работает

Здесь всё работает

- В JavaScript есть значения *true*, *false*, *null*, *undefined*, *NaN*, *Infinity*, что очень весело и удобно — см. рис. Следует различать функции isNaN и Number.isNaN и не путать *Infinity* с *Number.NEGATIVE_INFINITY*. Далее - кто во что горазд. Например, авторы новомодной библиотечки AngularJS решили, что просто необходимо дополнить список логических значений строчками "*no*", "*f*" и даже "*n*".
- Благодаря наличию более, чем одной виртуальной машины получаются вот такие таблички совместимости. Энтуазиазм и находчивость вендоров при реализации спецификации (написании VM) выливаются в мелкие несовместимости, кои ни в одну табличку не поместятся. Сугубо абстрактно, JS сам по себе никогда не умел ни DOM, ни в файловую систему, однако (тут сугубо реально) используется... или в контексте браузеров, что разращивает таблички совместимости аж до сайтов с десятками табличек совместимости; или в контексте "сервера" - что порождает спецификации для, например, подгрузки кода в трех несовместимых вариантах, которые, разумеется,

так же подвержены исключительно благородному влиянию вендоров.

- Скрипты исполняются в одной кодировке, а данные в Ajax передаются в другой — что совершенно естественно, все привыкли.
- В JavaScript можно невозбранно делить на ноль. ИЧСХ будет не NaN, null, false и т.п. Вернётся самая настоящая бесконечность.
- JavaScript — язык современных, практически мыслящих людей. Всякие там C, Lisp, OS/2 и BSD давно уже устарели.
- Наряду с PHP, именно он, наконец, изменит мир к лучшему.

## Не дови на меня

Сценарии на JavaScript можно запускать, набрав в адресной строке браузера *javascript:<некий код>*, это даёт возможность проверять значения некоторых глобальных переменных и вызвать определённые функции, а также использовать в коде web-страницы ущербные ссылки типа

```
<a href="javascript: alert( 'НЕ ДОВИ НА МЕНЯ!' );">ссылка</a>
```

вместо ~~кошерных~~ тоже ущербных

```
<a href="#" onclick="alert( 'НЕ ДОВИ НА МЕНЯ!' );">ссылка</a>
```

Объяснение: вариант (a href="javascript:...") для типовых применений ущербен по определению, ибо при открытии в новом окне браузер не сможет выполнить код. Откроется пустое окно с недействительным адресом.

Вариант (a href="#« onclick=»...") тоже сливает, ибо нажатие на ссылку средней кнопкой вызовет открытие бесполезного дубликата страницы, а при отключённых скриптах возможности выполнить действие нет. Совсем нет.

Правильный вариант: ссылка должна иметь реальный href на реальную страницу, либо onclick нужно дать не ссылке, а другому элементу, например, картинке. Подробнее читаем у Тёмы на сайте.

## Длиннющие названия

Длинные названия методов JavaScript позаимствовал у своего дальнего родственника — языка Java. Методы типа `getElementById()` (получить элемент по `id`) или `getElementsByTagName()` (получить элементы по тегу) прямо как в 1С с их

```
 КомпоновщикМакета = Новый КомпоновщикМакетаКомпоновкиДанных;
```

И это только некоторая часть ебаного стыда. Настоящий пиздец начинается, когда нужно проникнуть в некоторую узкую часть функции. Из-за изначальной кривоватости ООП в сабже может получиться так:

```
window['document'].getElementById('xyita').style.color = 'blue';
```

Приведённый пример меняет в элементе с `id` **xyita** цвет текста через CSS на синий.

В общем-то, много где нужно писать то, что в нормальном языке можно было бы и не писать.

## Быстрый доступ к DOM

Все современные браузеры позволяют обращаться к элементам DOM без вызова `document.getElementById()` просто обращаясь к элементам как к переменным. Имя переменной совпадает с атрибутом `id` элемента (переменная будет создана автоматически, если ее имя не зарезервированно). Например, на этом сайте в консоле (F12) можно обратиться к переменной `globalWrapper`, что короче чем вызов `document.getElementById("globalWrapper")`, но делать так в реальных проектах настоятельно не рекомендуется. Это сделано для совместимости с IE, где когда-то давным-давно не долго думая ввели эту сомнительную фичу.

## У вас this отклеился

Объект `this` ведет себя довольно странно.

```
var obj = {
  meth: function () {
    console.log(this);
  }
};
```

```
obj.meth(); // Object {}
var fn = obj.meth;
// И тут произойдет нежданчик
fn(); // Window {top: Window, location: Location, document: document, window: Window, external: Object…}
```

У данной проблемы есть тривиальное решение:

```
var fn = obj.meth.bind(obj);
```

Либо можно запомнить `this`:

```
function Stuff() {
  // запоминаем this в переменную
  var that = this;
  // и в дальнейшем к ней обращаемся
  that.meth = function () {
    console.log(that);
  };
}

var obj = new Stuff();
var fn = obj.meth;
fn(); // Stuff {meth: function}
```

Из-за чего так происходит? – Значение this вычисляется из контекста вызова.

- Про сабж

## Callback Hell

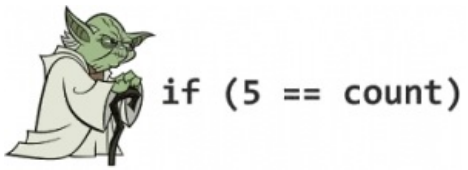
Типичный такой код на фреймворке Node.js. Является разновидностью макаронного кода. Promises теоретически могут решить эту проблему, но на деле их использование только позволяет убрать полосу прокрутки в рабочей области IDE. Это цена асинхронности. С этим можно смириться либо забыть о программировании на js.

Callback Hell

## Yoda Conditions

Болезнь в терминальной стадии, которой подвержены фронтенд-девелоперы. Пример:

```
if ('someshit' === somevar){
  // ...
}
```




Тревожный симптом, который говорит о том, что автор данных строк нуждается в немедленной эвтаназии

За такое советуют бить по рукам, но тем не менее распространенная практика среди математиков и пользователей всех остальных языков программирования, не ведущих свою родословную от C/C++. Потому что только в уродском синтаксисе C-подобных языков символ равенства является оператором присваивания, а сравнение производится двойным «==», или даже тройным «===», что является причиной массы трудно находимых ошибок. Если программист, привыкший к человеческому оператору сравнения, напишет "if (42 = foo) { ... }", то сразу вывалится ошибка компиляции.

Нужно также добавить, что для борьбы с этой напастью, в современных языках программирования пришлось специально запретить использование оператора присваивания в булевых выражениях. Чтобы возомнившие себя кулхацкерами JS-макаки не портили статистику по багам в корпоративном ПО.

## Точка с запятой

Есть мнение, что она нахуй не нужна в исходниках. Если ранее ее использование объяснялось тем, что обфускаторы и минификаторы не могли самостоятельно точки с запятой расставить, то в настоящее время – это уже вопрос вкуса. На самом деле точка с запятой в Джаваскрипт не более чем синтаксический сахар, но все же ее использование бывает оправданным, например, в таком коде:

```
function foo(){}
foo()
[1,2,3].map(x => x*x) // Uncaught TypeError: Cannot read property '3' of undefined
```

Тут интерпретатор, встретив [, решит что функция foo возвращает массив и это оператор взятия элемента массива по индексу, что приведет в итоге к ошибке. Тогда код нам придется переписать так:

```
function foo(){}
foo()
;[1,2,3].map(x => x*x)
```

А еще лучше наш массив засунуть в какую-нибудь переменную, и тогда необходимость в использовании точки с запятой исчезнет. Хуйня как описано выше может случится в самых разных случаях. Если Вы решили придерживаться подобного стиля, то не следует начинать новую строку с "(", "[" и "`". Если же такая необходимость возникла, поставьте перед ними точку с запятой.

# EcmaScript 2015 и последующие

JavaScript долгое время служил верой и правдой (20 лет), не претерпевая существенных изменений. Ситуация изменилась в 0x7df году. В этот год вышел стандарт ES-2015 (EcmaScript 6). Многие фичи из стандарта были реализованы в нормальных (т.е. всех, кроме IE) браузерах давно.

Что нового появилось в языке:

- Для объявления переменных теперь можно использовать операторы const/let. Особенность этих операторов является то, что переменные объявленные с их помощью, доступны только в локальном пространстве имен (между фигурными скобками "{}"). let от const отличается тем, что переменной, объявленной с помощью const, нельзя присвоить новое значение (Uncaught TypeError: Assignment to constant variable.). Спецификация советует не использовать больше оператор var. А const нужно использовать всегда, если не планируется присваивать переменной новое значение.

```
{
  // Это локальное пространство имен
  const q = 42;
  // Напечатает 42
  console.log(q);
}
// Uncaught ReferenceError: q is not defined
console.log(q);

// Если мы q объявим с помощью var, то будет напечатано 42
{
  var q = 42;
}
console.log(q);

for (let i = 0; i < 1000; ++i) {
  // переменная i доступна только внутри цикла (и вложенных циклов, скопов,
  // ваш К.О.)
}
// Uncaught ReferenceError: i is not defined
console.log(i);
```

Объекты, объявленные с помощью const, могут изменяться, при этом их адрес в памяти остаётся неизменным. Const гарантирует лишь то, что переменной не будет присвоено новое значение. Чтобы создать неизменяемый объект, нужно использовать такой способ:

```
const freezed = Object.freeze(source);.
```

- Появился нормальный foreach (хотя лучше использовать for of):

```
{
  const fruits = ['apple', 'banana', 'orange'];
  // Мацилла почему-то против использования const в условии цикла
  for (let fruit of fruits) {
    console.log(fruit);
  }
}
```

- Строки, заключенные в гравис (`), работают как php-шные строки в двойных кавычках:

```
function hello(name) {
  console.log(`Hello ${name}!`);
}
hello('World');

// Теперь можно использовать переносы внутри строки без экранирования типа такого:
const html = `<html>
  <body>
    <h1>Хуй</h1>
  </body>
</html>`;
```

- Распаковка/упаковка объектов

Упаковка:

```
const x = 42;
const o = {x}; // то же самое, что и {x: x}
console.log(o);

// В функциях также можно использовать
function хуй({пизда, джигурда}) { console.log(пизда) }
хуй({пизда: 42, джигурда: 'пидорас', кал: 'php'})
```

Распаковка:

```
const {x} = {x: 42, y: 265};
console.log(x);
```

А есче можно сделать swap:

```
let x = 42, y = 5
[x, y] = [y, x]
console.log(x, y)
```

- Спреды (spreads)

```
var q = [2, 3, 4];
// Аналог [1].concat(q).concat([5])
console.log([1, ...q, 5]);

// Также спреды можно применять в функциях

// Аналог пистоновского:
// def f(*args):
//     print(args)
function f(...args) { // f(x, y, ...args)
  console.log(args);
}

f('foo', 42);

// Аналог Math.max.apply(null, [1, 5, 3, 4, 2])
console.log(Math.max(...[1, 5, 3, 4, 2]));
```

- Генераторы

```
function* g() {
  yield "first";
  yield "second";
  yield "third";
}

var it = g();
it.next();
// {value: "first", done: false}
it.next();
// {value: "second", done: false}
it.next();
// {value: "third", done: false}
it.next();
// {value: undefined, done: true}

// Можно реализовать аналог питоновского range

function* range(start, stop, step){
  if (arguments.length < 2) {
    [start, stop] = [0, start];
  }
  if (arguments.length < 3) {
    step = 1;
  }
  for (let i = start; i < stop; i += step) {
    yield i;
  }
}

[...range(10)];
// [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[...range(1, 9, 3)];
// [1, 4, 7]

// yield* возвращает значения итератора
function* genAlphaNum() {
```

```
  yield* range(48, 58); // 0-9
  yield* range(65, 91); // A-Z
  yield* range(97, 123); // a-z
}

var alphaNum = '';
// Обход итератора с помощью for
for (let c of genAlphaNum()) {
  alphaNum += String.fromCharCode(c);
}
console.log(alphaNum);
// 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

// Функцию g можно переписать так:
function* g() {
  yield* ["first", "second", "third"];
}

// Мы не только можем получать значения, но и передавать их

function* test() {
  const ans = yield '2 * 2 = ?';
  if (ans != 4) {
    throw Error('Incorrect');
  }
  return 'OK';
}

var t1 = test();
console.log(t1.next().value); // 2 * 2 = ?
try {
  // Получим ошибку, т.к. ответ неверный
  console.log(t1.next(5).value);
} catch (err) {
  console.log(err);
}


var t2 = test();
t2.next(); // 2 * 2 = ?
console.log(t2.next(4).value); // OK

// Мы можем так же вызывать исключения внутри функций-генераторов

function* testException() {
  try {
    let q = yield null;
  } catch (err) {
    console.log('Catch ' + err);
  }
}

var it = testException()
it.next(); // {value: null, done: false}
// Теперь можно сгенерировать ошибку
it.throw(Error('test')); // Catch Error: test
```

- Функции обзавелись аргументами по умолчанию `function f(x = 42) { ... };`
- Появились монады

```
{
  const triple = (x) => x * x * x;
  console.log(triple(2));
}
```

Нужно отметить одну очень важную особенность стрелочных функций: они захватывают контекст this, т.е. `(x) => { /* ... */ }` аналогично такому объявлению функции `function (x) { /* ... */ }.bind(this);`.

- Появились классы

Сначала вспомним, как раньше классы наследовались через прототипы:

```
function Animal(type, name, age) {
  this.type = type;
  this.name = name;
  this.age = age;
}

Animal.prototype.info = function () {
```

```
  console.log(`${this.name} is a(n) ${this.type} ${this.age} years old`);
};

function Dog(name, age) {
  Animal.call(this, 'dog', name, age);
}

Object.setPrototypeOf(Dog.prototype, Animal.prototype);
// Object.setPrototypeOf заменяет эти две строки:
// Dog.prototype = Object.create(Animal.prototype);
// Dog.prototype.constructor = Dog;

Dog.prototype.bark = function() {
  console.log(this.name + ' says wooof');
};

function Human(name, age) {
  Animal.call(this, 'human', name, age);
}

Object.setPrototypeOf(Human.prototype, Animal.prototype);

Human.prototype.say = function (what) {
  console.log(`${this.name} says ${what}`);
};

var buddy = new Dog('Buddy', 2);
buddy.info();
// Buddy is a(n) dog 2 years old
buddy.bark();
// Buddy says wooof

var john = new Human('John', 34);
john.info();
// John is a(n) human 34 years old
john.say('hello');
// John says hello
```

Теперь это делается проще и красивее:

```
class Animal{
  constructor(type, name, age) {
    this.type = type;
    this.name = name;
    this.age = age;
  }

  info() {
    console.log(`${this.name} is a(n) ${this.type} ${this.age} years old`);
  }
}

class Dog extends Animal {
  constructor(name, age) {
    super('dog', name, age);
  }

  bark() {
    console.log(this.name + ' says wooof');
  }
}

class Human extends Animal {
  constructor(name, age) {
    super('human', name, age);
  }

  say(what) {
    console.log(`${this.name} says ${what}`);
  }
}
```

- Вложенные классы НЕ запрещены

```
class Sup {
  Sub = class Sub {}; // this needs --harmony flag
}
```

и не были

```
class Sup {}
Sup.Sub = class Sub {}
```

Про прототипы документация советует вообще забыть. this по-прежнему отклеивается.

- Промисы и метод fetch как замена XMLHttpRequest, возвращающая промис;
- Async/await

Сбылась мечта всех быдлокодеров: теперь асинхронный код можно писать как синхронный (ну почти). До появления async/await для этих целей использовали генераторы, теперь макаронного кода будет поменьше...

```
// Без async бросит ошибку Uncaught SyntaxError: Unexpected identifier
// т.е. все функции, где вызывается оператор await должны быть объявлены как
// async.
async function run(username, token) {
  // https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch
  const headers = new Headers();
  headers.append('Authorization', 'Basic ' + btoa(`${username}:${token}`));
  const result = await fetch('https://api.github.com/user', {headers})
    .then(response => response.json());
  console.log(JSON.stringify(result, true, 2));
  // ...
}

run('tz4678', '064...4c2');

// Вернет

{
  "login": "tz4678",
  "id": 12753171,
  "avatar_url": "https://avatars.githubusercontent.com/u/12753171?v=3",
  "gravatar_id": "",
  "url": "https://api.github.com/users/tz4678",
  "html_url": "https://github.com/tz4678",
  // ...
}

// Отлов ошибок
function f() {
  return new Promise((resolve, reject) => setTimeout(reject, 0, 'some error'));
}

// Можно отлавливать через try/catch
(async function run() {
  try {
    await f();
  } catch (e) {
    console.log(e);
  }
})();

// А можно с помощью метода catch, т.к. await-функции возвращают Promise
(async function run() {
  await f();
})().catch(e => console.log(e));
```

Подробнее про async/await можно почитать здесь.

- Новые типы данных типа (Symbol, Map и пр.).

Как нетрудно догадаться, новые фичи поддерживаются только нормальные браузеры (как обычно, IE сосет). Ну а на сервере (Node.js) все эти новшества можно опробовать уже сейчас.

## ТруЪ JavaScript-программисты

Настоящие мужики:

- Аксель Раушмайер — автор нескольких годных книг по JS и активный участник разработки стандарта языка.
- Дэвид Флэнаган — автор святого писания по JavaScript, под названием «JavaScript — Подробное руководство». Книга чуть менее, чем полностью состоит из перекрёстных ссылок, что значительно затрудняет и без того нелёгкое чтение этого произведения простыми смертными.
- Джон Ресиг — JavaScript-ниндзя 80-уровня и автор рафинированного сахара под названием jQuery. Да, это он добавил «доллар» к операторам языка.
- Питер-Поль Кох — автор сайта quirksmode.org и книги, написанной так давно, что о ней уже никто и

не помнит.
- Дуглас Крокфорд — суровый дядька из Yahoo, автор формата JSON, а также программы для чистки кода под названием JSLint и книги «JavaScript — The Good Parts».
- Николас Закас — ещё один (уже бывший) программист из Yahoo. Принимал участие в разработке фреймворка YUI (пхп-программистам не путать с YII).
- Дима Барановский — автор JS-библиотеки Raphaël, облегчающий нелёгкий труд рисования в браузерах с помощью SVG. Живет в Австралии.
- Юра (aka Kangax) Зайцев — разработчик ядра библиотеки prototype. Живёт в Нью-Йорке.
- Дэн Абрамов — пришел к успеху пиаря существующие вещи как новые под именем Redux.

## Алсо

- JScript — То же самое, что и JavaScript, тоже реализация языка ECMAScript, но микрософтом. Назван так, чтобы Sun (владелец торговой марки Java) не заметил совпадения названий.
- Ajax — «технология» отправки запросов серверу и получения ответов без перезагрузки страницы, обычно улучшает веб-приложения и портит обычные сайты.
- Node.js — теперь можно писать на JavaScript и на стороне сервера. Вообще-то, на стороне сервера на ЖС можно было невозбранно писать под тем ещё ASP, однако по умолчанию был выставлен Visual Basic, как более доступный простому народу, плывущему по течению, время от времени подгребая под себя.
- UnityScript — переделанный JS. Начиная с Unity 2018.3, выпилен [1].
- QtScript — ещё один JS, но уже внутри библиотеки Qt.
- Asm.js — подмножество JS, позволяющее ускорить выполнение засчёт Ahead-of-Time компиляции.

## См. также

- Стандарт, например
- Письмомер скорости жабаскрипта для браузеров, версия Гугла
- Письмомер скорости жабаскрипта для браузеров, яблочная версия
- JavaScript: фрактал отсоса
- Сайт, который поможет превратить нормальный JS-код в натуральный BrainFuck
- Актуальный онлайн учебник Javascript
- Принципы написания консистентного, идиоматического кода на JavaScript
- Рекомендации по написанию кода на языке JavaScript
- JavaScript Garden na russkom
- Про JS-разработку тоже есть порно
- Говорим на ECMAScript 5
- обзор новых фич ECMAScript 6
- Описание всех возможностей ES6 на одном сайте (англ.)
- Совместимость браузеров с ES6

## Примечания

1. ↑ В особо запущенных случаях требует костылей для самого себя, например, отсутствие nextElementSibling и т. п.
2. ↑ нубов, черпаков


Языки программирования
++i + ++i 1C AJAX BrainFuck C Sharp C++ Dummy mode Erlang Forth FUBAR
God is real, unless explicitly declared as integer GOTO Haskell Ifconfig Java JavaScript LISP
My other car Oracle Pascal Perl PHP Prolog Pure C Python RegExp Reverse Engineering
Ruby SAP SICP Tcl TeX Xyzzy Анти-паттерн Ассемблер Быдлокодер
Выстрелить себе в ногу Грязный хак Дискета ЕГГОГ Индусский код Инжалид дежице
Капча КОИ-8 Костыль Лог Метод научного тыка Очередь Помолясь Проблема 2000
Программист Процент эс Рекурсия Свистелки и перделки Спортивное программирование
СУБД Тестировщик Умение разбираться в чужом коде Фаза Луны Фортран Хакер
Языки программирования

w:JavaScript en.w:JavaScript ae:JavaScript