Руководство по HTML/CSS/JavaScript

[JavaScript](https://habrahabr.ru/hub/javascript/)\*, [HTML](https://habrahabr.ru/hub/html5/)\*, [CSS](https://habrahabr.ru/hub/css/)\*

**Frontend Guidelines**

Правила и руководства оформления, форматирования HTML, СSS и JavaScript кода. Его цель — повысить качество кода и облегчить совместную работу и поддержку инфраструктуры.

**От переводчика**

Приветствую всех снова, меня зовут Максим Иванов, и сегодня я подготовил перевод, который, возможно, окажется для вас полезным. [Бенджамин Де Кук](https://dribbble.com/bdc)(Benjamin De Cock), разработчик из Бельгии, собрал некоторые указания по оформлению кода, которые позиционируют себя как [лучшие практики](https://github.com/bendc/frontend-guidelines) по написанию HTML, CSS, JS. Конечно, существует множество рекомендаций, например, есть хороший [гайдлайн от Google](https://google.github.io/styleguide/htmlcssguide.xml), наверное, есть еще что-то, однако, если следовать хотя бы некоторым из них, то можно надеяться, что ваш код станет лучше. В отдельных случаях следование этим гайдлайнам не полезно, а совсем наоборот. В общем и целом, все зависит от вашего опыта и виденья дела, если вы новичок, то скорее вам будет полезно оценить то, что пишут другие и в обществе считается верным, если вы гуру, то наверное вам и не нужны гайдлайны, которые написаны непонятно кем на ваш взгляд. Итак, приступим.

**HTML**

1. [Семантика](https://habrahabr.ru/post/275729/#one)
2. [Лаконичность](https://habrahabr.ru/post/275729/#two)
3. [Читабельность](https://habrahabr.ru/post/275729/#three)
4. [Язык документа](https://habrahabr.ru/post/275729/#four)
5. [Эффективность](https://habrahabr.ru/post/275729/#five)

**CSS**

1. [Точка с запятой](https://habrahabr.ru/post/275729/#six)
2. [Блочная модель](https://habrahabr.ru/post/275729/#seven)
3. [Потоки](https://habrahabr.ru/post/275729/#eight)
4. [Позиционирование](https://habrahabr.ru/post/275729/#night)
5. [Селекторы](https://habrahabr.ru/post/275729/#ten)
6. [Специфичность](https://habrahabr.ru/post/275729/#eleven)
7. [Переопределения](https://habrahabr.ru/post/275729/#twelve)
8. [Наследование](https://habrahabr.ru/post/275729/#thirteen)
9. [Лаконичность](https://habrahabr.ru/post/275729/#fourteen)
10. [Англоязычные обозначения](https://habrahabr.ru/post/275729/#fifteen)
11. [Вендорные префиксы](https://habrahabr.ru/post/275729/#sixteen)
12. [Анимация](https://habrahabr.ru/post/275729/#seventeen)
13. [Единицы измерения](https://habrahabr.ru/post/275729/#eighteen)
14. [Цветовая модель](https://habrahabr.ru/post/275729/#nineteen)
15. [Отрисовка](https://habrahabr.ru/post/275729/#twenty)
16. [Хаки](https://habrahabr.ru/post/275729/#twenty-one)

**JavaScript**

1. [Производительность](https://habrahabr.ru/post/275729/#twenty-two)
2. [Независимость от внешнего состояния](https://habrahabr.ru/post/275729/#twenty-three)
3. [Нативность](https://habrahabr.ru/post/275729/#twenty-four)
4. [Принудительное сравнение](https://habrahabr.ru/post/275729/#twenty-five)
5. [Циклы](https://habrahabr.ru/post/275729/#twenty-six)
6. [Аргументы](https://habrahabr.ru/post/275729/#twenty-seven)
7. [Использование метода apply](https://habrahabr.ru/post/275729/#twenty-eight)
8. [Использование метода bind](https://habrahabr.ru/post/275729/#twenty-nine)
9. [Методы высшего порядка](https://habrahabr.ru/post/275729/#thirty)
10. [Композиции](https://habrahabr.ru/post/275729/#thirty-one)
11. [Кэширование](https://habrahabr.ru/post/275729/#thirty-two)
12. [Переменные](https://habrahabr.ru/post/275729/#thirty-three)
13. [Условия](https://habrahabr.ru/post/275729/#thirty-four)
14. [Итератор](https://habrahabr.ru/post/275729/#thirty-five)
15. [Объект как ассоциативный массив](https://habrahabr.ru/post/275729/#thirty-six)
16. [Каррирование](https://habrahabr.ru/post/275729/#thirty-seven)
17. [Читабельность](https://habrahabr.ru/post/275729/#thirty-eight)
18. [Повторное использование кода](https://habrahabr.ru/post/275729/#thirty-nine)
19. [Зависимости](https://habrahabr.ru/post/275729/#forty)

**HTML**

**Семантика**

HTML5 предоставляет нам достаточное количество элементов, направленных на точное семантическое описание документа или содержимого. Поэтому, убедитесь в том, что у вас все хорошо с лексикой.

<!-- плохо -->

<div id="main">

<div class="article">

<div class="header">

<h1>Blog post</h1>

<p>Published: <span>21st Feb, 2015</span></p>

</div>

<p>…</p>

</div>

</div>

<!-- хорошо -->

<main>

<article>

<header>

<h1>Blog post</h1>

<p>Published: <time datetime="2015-02-21">21st Feb, 2015</time></p>

</header>

<p>…</p>

</article>

</main>

Убедитесь, что вы поняли семантику элементов, которые вы используете в своем коде. Очень плохо использовать семантические элементы не по назначению или не в том месте, если бы вы просто оставались нейтральными к этим этим семантическим элементам.

<!-- плохо -->

<h1>

<figure>

<img alt=Company src=logo.png>

</figure>

</h1>

<!-- хорошо -->

<h1>

<img alt=Company src=logo.png>

</h1>

**Лаконичность**

Старайтесь кратко писать код. Забудьте о своих старых привычках, перетекающих из XHTML.

<!-- плохо -->

<!doctype html>

<html lang=en>

<head>

<meta http-equiv=Content-Type content="text/html; charset=utf-8" />

<title>Contact</title>

<link rel=stylesheet href=style.css type=text/css />

</head>

<body>

<h1>Contact me</h1>

<label>

Email address:

<input type=email placeholder=you@email.com required=required />

</label>

<script src=main.js type=text/javascript></script>

</body>

</html>

<!-- хорошо -->

<!doctype html>

<html lang=en>

<meta charset=utf-8>

<title>Contact</title>

<link rel=stylesheet href=style.css>

<h1>Contact me</h1>

<label>

Email address:

<input type=email placeholder=you@email.com required>

</label>

<script src=main.js></script>

</html>

**Читабельность**

Читабельность вашего кода не должна иметь второстепенный характер. Конечно, вы не обязаны быть экспертом по оптимизации (WCAG) для того, чтобы наверняка улучшить свой сайт, однако, уже прямо сейчас вы можете начать немедленно исправлять мелочи, которые в итоге помогут сделать ваш код лучше, например:  
  
— использовать атрибут *alt* правильно  
— связывайте имена (links) и элементы (buttons) и называйте их по смыслу (*<div class=button>* — это зверство)  
— не полагайтесь на название цветов, в качестве информации  
— явно связывайте определенной меткой и элемент формы

<!-- плохо -->

<h1><img alt="Logo" src="logo.png"></h1>

<!-- хорошо -->

<h1><img alt="My Company, Inc." src="logo.png"></h1>

**Язык документа**

Определении языка и кодировки символов не является обязательным, однако рекомендуется всегда объявлять их на уровне документа, даже если они указаны в заголовках HTTP. Приоритетной кодировкой символов является UTF-8.

<!-- плохо -->

<!doctype html>

<title>Привет, мир.</title>

<!-- хорошо -->

<!doctype html>

<html lang=ru>

<meta charset=utf-8>

<title>Привет, мир.</title>

</html>

**Эффективность**

Если нет уважительных причин добавлять загрузку скриптов до контента, старайтесь прописывать скрипты в конце документа, тем самым ваши скрипты не будут блокировать рендеринг страницы. Если у вас очень тяжелые таблицы стилей, старайтесь разбить их на несколько и загружать по отдельности, так чтобы первичный контент был сразу отрисован. Очевидно, чем больше файлов, тем больше HTTP запросов, два запроса, конечно, медленнее, чем один, но в нашем случае, важным фактором будет являться скорость отображения нужного контента. 

<!-- плохо -->

<!doctype html>

<meta charset=utf-8>

<script src=analytics.js></script>

<title>Hello, world.</title>

<p>...</p>

<!-- хорошо -->

<!doctype html>

<meta charset=utf-8>

<title>Hello, world.</title>

<p>...</p>

<script src=analytics.js></script>

**CSS**

**Точка с запятой**

В то время как точка с запятой является чисто технически разделителем правил, CSS всегда относится к точке запятой, как к оператору завершения.

/\* плохо \*/

div {

color: red

}

/\* хорошо \*/

div {

color: red;

}

**Box model**

Блочная модель, в идеале, должно быть одинаковой для всего документа. Глобальное правило *\* { box-sizing: border-box; }* это нормально, однако нужно помнить, что вы не можете изменить блочную модель у специфичных элементов.

/\* плохо \*/

div {

width: 100%;

padding: 10px;

box-sizing: border-box;

}

/\* хорошо \*/

div {

padding: 10px;

}

**Потоки**

Не меняйте стандартное поведение элемента, если вы можете обойтись без этого. Сохраняйте элементы в их классическом отображении в потоке документа. Например, удаление пробелов под изображением, не должно влиять на его отображение.

/\* плохо \*/

img {

display: block;

}

/\* хорошо \*/

img {

vertical-align: middle;

}

Аналогично, не вытаскивайте из основного потока элементы, если вы можете обойтись без этого.

/\* плохо \*/

div {

width: 100px;

position: absolute;

right: 0;

}

/\* хорошо \*/

div {

width: 100px;

margin-left: auto;

}

**Позиционирование**

Существует множество способов позиционирования элементов в CSS, однако старайтесь ограничивать себя свойствами представленными ниже. В порядке предпочтения:

display: block;

display: flex;

position: relative;

position: sticky;

position: absolute;

position: fixed;

**Селекторы**

Минимизируйте сочетания селекторов связанных с DOM. Иногда, лучше добавить лишний класс элементу, который вы хотите найти, чем искать его через три структурных селектора: псевдо-класс, потомок, комбинатор.

/\* плохо \*/

div:first-of-type :last-child > p ~ \*

/\* хорошо \*/

div:first-of-type .info

Избегайте перегрузку селекторов.

/\* плохо \*/

img[src$=svg], ul > li:first-child {

opacity: 0;

}

/\* хорошо \*/

[src$=svg], ul > :first-child {

opacity: 0;

}

**Специфичность**

Не используйте значения и селекторы, которые нельзя переопределить. Минимизируйте использование *id* (идентификаторов) и избегайте правило *!important*

/\* плохо \*/

.bar {

color: green !important;

}

.foo {

color: red;

}

/\* хорошо \*/

.foo {

color: red;

}

.foo.bar {

color: green;

}

**Переопределения**

Переопределение стилей селекторов затрудняет отладку. Избегайте этого, если можете.

/\* плохо \*/

li {

visibility: hidden;

}

li:first-child {

visibility: visible;

}

/\* хорошо \*/

li + li {

visibility: hidden;

}

**Наследование**

Не дублируйте определения селекторов, стили которых и так могут передаваться по наследству.

/\* плохо \*/

div h1, div p {

text-shadow: 0 1px 0 #fff;

}

/\* хорошо \*/

div {

text-shadow: 0 1px 0 #fff;

}

**Лаконичность**

Пусть ваш код будет немногословным. Старайтесь избегать использование несколько свойств в том месте, где это вовсе не нужно.

/\* плохо \*/

div {

transition: all 1s;

top: 50%;

margin-top: -10px;

padding-top: 5px;

padding-right: 10px;

padding-bottom: 20px;

padding-left: 10px;

}

/\* хорошо \*/

div {

transition: 1s;

top: calc(50% - 10px);

padding: 5px 10px 20px;

}

**Англоязычные обозначения**

Предпочтительно использовать английскую математику.

/\* плохо \*/

:nth-child(2n + 1) {

transform: rotate(360deg);

}

/\* хорошо \*/

:nth-child(odd) {

transform: rotate(1turn);

}

**Вендорные префиксы**

Не используйте устаревшие префиксы. Если вам действительно необходимо использовать их, в этом случае, добавляйте их перед стандартными.

/\* плохо \*/

div {

transform: scale(2);

-webkit-transform: scale(2);

-moz-transform: scale(2);

-ms-transform: scale(2);

transition: 1s;

-webkit-transition: 1s;

-moz-transition: 1s;

-ms-transition: 1s;

}

/\* хорошо \*/

div {

-webkit-transform: scale(2);

transform: scale(2);

transition: 1s;

}

**Анимация**

Иногда предпочтительнее использовать переходы (transitions), ежели анимацию (animations). Избегайте анимирование свойств в пользу *opacity* и *transform*. 

/\* плохо \*/

div:hover {

animation: move 1s forwards;

}

@keyframes move {

100% {

margin-left: 100px;

}

}

/\* хорошо \*/

div:hover {

transition: 1s;

transform: translateX(100px);

}

**Единицы измерения**

Используйте безразмерные значения. Старайтесь использовать *rem*, если вы рассчитываете все в относительных единицах. Предпочтительно использовать секунды, а не миллисекунды.

/\* плохо \*/

div {

margin: 0px;

font-size: .9em;

line-height: 22px;

transition: 500ms;

}

/\* хорошо \*/

div {

margin: 0;

font-size: .9rem;

line-height: 1.5;

transition: .5s;

}

**Цветовая модель**

Если нужна прозрачность, использовать *rgba*. В противном случае, всегда используйте шестнадцатеричный формат.

/\* плохо \*/

div {

color: hsl(103, 54%, 43%);

}

/\* хорошо \*/

div {

color: #5a3;

}

**Отрисовка**

Избегайте http-запросов, если ресурс легко воспроизводится с помощью CSS.

/\* плохо \*/

div::before {

content: url(white-circle.svg);

}

/\* хорошо \*/

div::before {

content: "";

display: block;

width: 20px;

height: 20px;

border-radius: 50%;

background: #fff;

}

**Хаки**

Не используйте их.

/\* плохо \*/

div {

// position: relative;

transform: translateZ(0);

}

/\* хорошо \*/

div {

/\* position: relative; \*/

will-change: transform;

}

**JavaScript**

**Производительность**

Легкочитаемый, выразительный и правильный код важнее производительности. На самом деле, JavaScript никогда не будет вашим узким местом в расчете на производительность. Лучше уделять внимание таким вещам, как сжатие изображений, оптимизация http-запросов и доступа к сети, перерасчет DOM-дерева. Если вы запомните хотя бы одну из рекомендаций в данной статье, опирайтесь на нее.

/\* плохо (хотя и намного быстрее) \*/

const arr = [1, 2, 3, 4];

const len = arr.length;

var i = -1;

var result = [];

while (++i < len) {

var n = arr[i];

if (n % 2 > 0) continue;

result.push(n \* n);

}

/\* хорошо \*/

const arr = [1, 2, 3, 4];

const isEven = n => n % 2 == 0; // инициализация анонимной стрелочной функции

const square = n => n \* n; // синтаксис: (param1, param2, …, paramN) => expression

const result = arr.filter(isEven).map(square);

**К прочтению**:  
1. Babeljs (пример в ES5): [плохо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20arr%20%3D%20%5B1%2C%202%2C%203%2C%204%5D%3B%0Aconst%20len%20%3D%20arr.length%3B%0Avar%20i%20%3D%20-1%3B%0Avar%20result%20%3D%20%5B%5D%3B%0Awhile%20(%2B%2Bi%20%3C%20len)%20%7B%0A%20%20var%20n%20%3D%20arr%5Bi%5D%3B%0A%20%20if%20(n%20%25%202%20%3E%200)%20continue%3B%0A%20%20result.push(n%20*%20n)%3B%20%0A%7D%0A%0A%2F%2F%20result%20%3D%20%5B4%2C%2016%5D%3B%0A) vs [хорошо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20arr%20%3D%20%5B1%2C%202%2C%203%2C%204%5D%3B%0Aconst%20isEven%20%3D%20b%20%3D%3E%20(n%20%25%202%20%3D%3D%200)%3B%0Aconst%20square%20%3D%20b%20%3D%3E%20n%20*%20n%3B%0A%0Aconst%20result%20%3D%20arr.filter(isEven).map(square)%3B)  
2. [Стрелочные функции](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Functions#Cтрелочные_функции) и [перебирающие методы массивов](https://learn.javascript.ru/array-iteration)

**Независимость от внешнего состояния**

Старайтесь создавать свободные функции. Все функции и методы в идеале не должны нести никакого побочного характера, зависеть от внешних данных, и в итоге, они должны возвращать новые объекты, а не вносить изменения в существующие.

/\* плохо \*/

const merge = (target, ...sources) => Object.assign(target, ...sources);

merge({ foo: "foo" }, { bar: "bar" }); // => { foo: "foo", bar: "bar" }

/\* хорошо \*/

const merge = (...sources) => Object.assign({}, ...sources);

merge({ foo: "foo" }, { bar: "bar" }); // => { foo: "foo", bar: "bar" }

**К прочтению**:  
1. Babeljs (пример в ES5): [плохо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20merge%20%3D%20(target%2C%20...sources)%20%3D%3E%20Object.assign(target%2C%20...sources)%3B%0Amerge(%7B%20foo%3A%20%22foo%22%20%7D%2C%20%7B%20bar%3A%20%22bar%22%20%7D)%3B) vs [хорошо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20merge%20%3D%20(...sources)%20%3D%3E%20Object.assign(%7B%7D%2C%20...sources)%3B%0Amerge(%7B%20foo%3A%20%22foo%22%20%7D%2C%20%7B%20bar%3A%20%22bar%22%20%7D)%3B)  
2. [Остаточные параметры](http://frontender.info/es6-in-depth-rest-parameters-and-defaults/#ostatochneparametr)

**Нативность**

Полагайтесь на стандартные методы, как можно больше.

/\* плохо \*/

const toArray = obj => [].slice.call(obj);

/\* хорошо \*/

const toArray = (() =>

Array.from ? Array.from : obj => [].slice.call(obj)

)();

**К прочтению**:  
1. Babeljs (пример в ES5): [плохо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20toArray%20%3D%20obj%20%3D%3E%20%5B%5D.slice.call(obj)%3B) vs [хорошо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20toArray%20%3D%20(()%20%3D%3E%0A%20%20Array.from%20%3F%20Array.from%20%3A%20obj%20%3D%3E%20%5B%5D.slice.call(obj)%0A)()%3B)  
2. [Массивоподобные объекты](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/slice#Array-like)

**Принудительное сравнение**

Лишние сравнения не имеют смысл, избегайте их.

/\* плохо \*/

if (x === undefined || x === null) { /\* ... \*/ }

/\* хорошо \*/

if (x == undefined) { /\* ... \*/ }

**К прочтению**:  
1. [Сравнение с null и undefined](https://learn.javascript.ru/comparison#сравнение-с-null-и-undefined)

**Циклы**

Не используйте циклы, так как они заставляют вас использовать изменяемые объекты. Полагайтесь на методы *Array.prototype*.

/\* плохо \*/

const sum = arr => {

var sum = 0;

var i = -1;

for (;arr[++i];) {

sum += arr[i];

}

return sum;

};

sum([1, 2, 3]); // => 6

/\* хорошо \*/

const sum = arr => arr.reduce((x, y) => x + y);

sum([1, 2, 3]); // => 6

**К прочтению**:  
1. Babeljs (пример в ES5): [плохо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20sum%20%3D%20arr%20%3D%3E%20%7B%0A%20%20var%20sum%20%3D%200%3B%0A%20%20var%20i%20%3D%20-1%3B%0A%20%20for%20(%3Barr%5B%2B%2Bi%5D%3B)%20%7B%0A%20%20%20%20sum%20%2B%3D%20arr%5Bi%5D%3B%0A%20%20%7D%0A%20%20return%20sum%3B%0A%7D%3B%0A%0Asum(%5B1%2C%202%2C%203%5D)%3B%20%2F%2F%20%3D%3E%206) vs [хорошо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20sum%20%3D%20arr%20%3D%3E%20arr.reduce((x%2C%20y)%20%3D%3E%20x%20%2B%20y)%3B%0A%0Asum(%5B1%2C%202%2C%203%5D)%3B%20%2F%2F%20%3D%3E%206)  
2. [Суммирование всех значений в массиве](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce#Example:_Sum_up_all_values_within_an_array)  
  
Если вы не можете использовать методы *Array.prototype*, используйте рекурсию.

/\* плохо \*/

const createDivs = howMany => {

while (howMany--) {

document.body.insertAdjacentHTML("beforeend", "<div></div>");

}

};

createDivs(5);

/\* плохо \*/

const createDivs = howMany =>

[...Array(howMany)].forEach(() =>

document.body.insertAdjacentHTML("beforeend", "<div></div>")

);

createDivs(5);

/\* хорошо \*/

const createDivs = howMany => {

if (!howMany) return;

document.body.insertAdjacentHTML("beforeend", "<div></div>");

return createDivs(howMany - 1);

};

createDivs(5);

Вот общая функция [Loop](https://gist.github.com/bendc/6cb2db4a44ec30208e86), делающая рекурсию проще в использовании.  
  
**К прочтению**:  
1. Babeljs (пример в ES5): [плохо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=%2F*%20%D0%BF%D0%BB%D0%BE%D1%85%D0%BE%20*%2F%0Aconst%20createDivs%20%3D%20howMany%20%3D%3E%20%7B%0A%20%20while%20(howMany--)%20%7B%0A%20%20%20%20document.body.insertAdjacentHTML(%22beforeend%22%2C%20%22%3Cdiv%3E%3C%2Fdiv%3E%22)%3B%0A%20%20%7D%0A%7D%3B%0A%0AcreateDivs(5)%3B) ([или](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20createDivs%20%3D%20howMany%20%3D%3E%0A%20%20%5B...Array(howMany)%5D.forEach(()%20%3D%3E%0A%20%20%20%20document.body.insertAdjacentHTML(%22beforeend%22%2C%20%22%3Cdiv%3E%3C%2Fdiv%3E%22)%0A%20%20)%3B%0A%0AcreateDivs(5)%3B)) vs [хорошо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20createDivs%20%3D%20howMany%20%3D%3E%20%7B%0A%20%20if%20(!howMany)%20return%3B%0A%20%20document.body.insertAdjacentHTML(%22beforeend%22%2C%20%22%3Cdiv%3E%3C%2Fdiv%3E%22)%3B%0A%20%20return%20createDivs(howMany%20-%201)%3B%0A%7D%3B%0A%0AcreateDivs(5)%3B)  
2. [insertAdjacentHTML() работает быстрее, чем манипуляции с innerHTML()](https://developer.mozilla.org/ru/docs/Web/API/Element/insertAdjacentHTML)

**Аргументы**

Забудьте об *arguments* как об объекте. Остаточные параметры это всегда лучший вариант, потому что:  
1. Передаваемое имя дает лучшее представление о том, что ожидает функция на входе  
2. Остаточный параметр — это массив и он легок в использовании

/\* плохо \*/

const sortNumbers = () =>

Array.prototype.slice.call(arguments).sort();

/\* хорошо \*/

const sortNumbers = (...numbers) => numbers.sort();

**К прочтению**:  
1. Babeljs (пример в ES5): [плохо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20sortNumbers%20%3D%20()%20%3D%3E%0A%20%20Array.prototype.slice.call(arguments).sort()%3B) vs [хорошо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20sortNumbers%20%3D%20(...numbers)%20%3D%3E%20numbers.sort()%3B)  
2. [Доступ к аргументам](https://learn.javascript.ru/arguments-pseudoarray#доступ-к-лишним-аргументам)

**Использование метода apply**

Забудьте метода apply(). Вместо этого используйте оператор распространения (вызова функции).

const greet = (first, last) => `Hi ${first} ${last}`;

const person = ["John", "Doe"];

/\* плохо \*/

greet.apply(null, person);

/\* хорошо \*/

greet(...person);

**К прочтению**:  
1. Babeljs (пример в ES5): [плохо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20greet%20%3D%20(first%2C%20last)%20%3D%3E%20%60Hi%20%24%7Bfirst%7D%20%24%7Blast%7D%60%3B%0Aconst%20person%20%3D%20%5B%22John%22%2C%20%22Doe%22%5D%3B%0A%0A%2F%2F%20%D0%BF%D0%BB%D0%BE%D1%85%D0%BE%0Agreet.apply(null%2C%20person)%3B) vs [хорошо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20greet%20%3D%20(first%2C%20last)%20%3D%3E%20%60Hi%20%24%7Bfirst%7D%20%24%7Blast%7D%60%3B%0Aconst%20person%20%3D%20%5B%22John%22%2C%20%22Doe%22%5D%3B%0A%0A%2F%2F%20%D1%85%D0%BE%D1%80%D0%BE%D1%88%D0%BE%0Agreet(...person)%3B)  
2. [Оператор распространения](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Spread_operator)

**Использование метода bind**

Не используйте bind(), когда есть более идиоматические подходы.

/\* плохо \*/

["foo", "bar"].forEach(func.bind(this));

/\* хорошо \*/

["foo", "bar"].forEach(func, this);

/\* плохо \*/

const person = {

first: "John",

last: "Doe",

greet() {

const full = function() {

return `${this.first} ${this.last}`;

}.bind(this);

return `Hello ${full()}`;

}

}

/\* хорошо \*/

const person = {

first: "John",

last: "Doe",

greet() {

const full = () => `${this.first} ${this.last}`;

return `Hello ${full()}`;

}

}

**К прочтению**:  
1. Babeljs (пример в ES5): [плохо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20person%20%3D%20%7B%0A%20%20first%3A%20%22John%22%2C%0A%20%20last%3A%20%22Doe%22%2C%0A%20%20greet()%20%7B%0A%20%20%20%20const%20full%20%3D%20function()%20%7B%0A%20%20%20%20%20%20return%20%60%24%7Bthis.first%7D%20%24%7Bthis.last%7D%60%3B%0A%20%20%20%20%7D.bind(this)%3B%0A%20%20%20%20return%20%60Hello%20%24%7Bfull()%7D%60%3B%0A%20%20%7D%0A%7D) vs [хорошо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20person%20%3D%20%7B%0A%20%20first%3A%20%22John%22%2C%0A%20%20last%3A%20%22Doe%22%2C%0A%20%20greet()%20%7B%0A%20%20%20%20const%20full%20%3D%20()%20%3D%3E%20%60%24%7Bthis.first%7D%20%24%7Bthis.last%7D%60%3B%0A%20%20%20%20return%20%60Hello%20%24%7Bfull()%7D%60%3B%0A%20%20%7D%0A%7D)  
2. [bind()](https://habrahabr.ru/post/199456/), а также [итераторы](http://alljs.ru/articles/array/iterators) в JavaScript

**Методы высшего порядка**

Избегайте вложенных функций там где это не нужно. 

/\* плохо \*/

const result = [1, 2, 3].map(num => String(num)); // => ["1", "2", "3"]

/\* хорошо \*/

const result = [1, 2, 3].map(String); // Приведение всех элементов к типу string

**К прочтению**:  
1. Babeljs (пример в ES5): [плохо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20result%20%3D%20%5B1%2C%202%2C%203%5D.map(num%20%3D%3E%20String(num))%3B) vs [хорошо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20result%20%3D%20%5B1%2C%202%2C%203%5D.map(String)%3B%20)  
2. [Отображение массива чисел с использованием функции](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/map#Example:_Mapping_an_array_of_numbers_using_a_function_containing_an_argument)

**Композиции**

Избегайте несколько вложенных вызовов функций. Используйте вместо них композицию.

const plus1 = a => a + 1;

const mult2 = a => a \* 2;

/\* плохо \*/

mult2(plus1(5)); // => 12

/\* хорошо \*/

const pipeline = (...funcs) => val => funcs.reduce((a, b) => b(a), val);

const addThenMult = pipeline(plus1, mult2);

addThenMult(5); // => 12

**К прочтению**:  
1. Babeljs (пример в ES5): [плохо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20plus1%20%3D%20a%20%3D%3E%20a%20%2B%201%3B%0Aconst%20mult2%20%3D%20a%20%3D%3E%20a%20*%202%3B%0A%0A%2F*%20%D0%BF%D0%BB%D0%BE%D1%85%D0%BE%20*%2F%0Amult2(plus1(5))%3B%20%2F%2F%20%3D%3E%2012) vs [хорошо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20plus1%20%3D%20a%20%3D%3E%20a%20%2B%201%3B%0Aconst%20mult2%20%3D%20a%20%3D%3E%20a%20*%202%3B%0A%0A%2F*%20%D1%85%D0%BE%D1%80%D0%BE%D1%88%D0%BE%20*%2F%0Aconst%20pipeline%20%3D%20(...funcs)%20%3D%3E%20val%20%3D%3E%20funcs.reduce((a%2C%20b)%20%3D%3E%20b(a)%2C%20val)%3B%0Aconst%20addThenMult%20%3D%20pipeline(plus1%2C%20mult2)%3B%0A%0AaddThenMult(5)%3B%20%2F%2F%20%3D%3E%2012)

**Кеширование**

Дорогостоящие операции с большими структурами данных непростительны, если вы не кешируете их.

/\* плохо \*/

const contains = (arr, value) =>

Array.prototype.includes

? arr.includes(value)

: arr.some(el => el === value);

contains(["foo", "bar"], "baz"); // => false

/\* хорошо \*/

const contains = (() =>

Array.prototype.includes

? (arr, value) => arr.includes(value)

: (arr, value) => arr.some(el => el === value)

)();

contains(["foo", "bar"], "baz"); // => false

**К прочтению**:  
1. Babeljs (пример в ES5): [плохо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20contains%20%3D%20(arr%2C%20value)%20%3D%3E%0A%20%20Array.prototype.includes%0A%20%20%20%20%3F%20arr.includes(value)%0A%20%20%20%20%3A%20arr.some(el%20%3D%3E%20el%20%3D%3D%3D%20value)%3B%0Acontains(%5B%22foo%22%2C%20%22bar%22%5D%2C%20%22baz%22)%3B%20%2F%2F%20%3D%3E%20false) vs [хорошо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20contains%20%3D%20(()%20%3D%3E%0A%20%20Array.prototype.includes%0A%20%20%20%20%3F%20(arr%2C%20value)%20%3D%3E%20arr.includes(value)%0A%20%20%20%20%3A%20(arr%2C%20value)%20%3D%3E%20arr.some(el%20%3D%3E%20el%20%3D%3D%3D%20value)%0A)()%3B%0Acontains(%5B%22foo%22%2C%20%22bar%22%5D%2C%20%22baz%22)%3B%20%2F%2F%20%3D%3E%20false)  
2. Определяем, [находится ли одна строка внутри другой](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/String/contains), а также [делаем проверку значений элементов массива](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/some#Example:_Testing_size_of_all_array_elements)

**Переменные**

Используйте *const* вместо *let* и *let* вместо *var*.

/\* плохо \*/

var me = new Map();

me.set("name", "Ben").set("country", "Belgium");

/\* хорошо \*/

const me = new Map();

me.set("name", "Ben").set("country", "Belgium");

**К прочтению**:  
1. Babeljs (пример в ES5): [плохо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=var%20me%20%3D%20new%20Map()%3B%0Ame.set(%22name%22%2C%20%22Ben%22).set(%22country%22%2C%20%22Belgium%22)%3B) vs [хорошо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20me%20%3D%20new%20Map()%3B%0Ame.set(%22name%22%2C%20%22Ben%22).set(%22country%22%2C%20%22Belgium%22)%3B)  
2. [Константы](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/const#Examples), [оператор let](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/let#let_vs_var), и немного об [ассоциативном массиве](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Map#Использование_объекта_Map)

**Условия**

Будьте привержены концепции IIFE's (немедленно выполняемый функтор), используйте возвращаемые значения в условиях (if, else if, else и switch).

/\* плохо \*/

var grade, result = 50;

if (result < 50)

grade = "bad";

else if (result < 90)

grade = "good";

else

grade = "excellent";

// grade = "good";

/\* хорошо \*/

let result = 50;

const grade = (() => {

if (result < 50)

return "bad";

if (result < 90)

return "good";

return "excellent";

})(); // grade = "good";

**К прочтению**:  
1. Babeljs (пример в ES5): [плохо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=var%20grade%2C%20result%20%3D%2050%3B%0Aif%20(result%20%3C%2050)%0A%20%20grade%20%3D%20%22bad%22%3B%0Aelse%20if%20(result%20%3C%2090)%0A%20%20grade%20%3D%20%22good%22%3B%0Aelse%0A%20%20grade%20%3D%20%22excellent%22%3B%0A%20%20%0A%2F%2F%20grade%20%3D%20%22good%22%3B) vs [хорошо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=let%20result%20%3D%2050%3B%0Aconst%20grade%20%3D%20(()%20%3D%3E%20%7B%0A%20%20if%20(result%20%3C%2050)%0A%20%20%20%20return%20%22bad%22%3B%0A%20%20if%20(result%20%3C%2090)%0A%20%20%20%20return%20%22good%22%3B%0A%20%20return%20%22excellent%22%3B%0A%7D)()%3B%20%2F%2F%20grade%20%3D%20%22good%22%3B) ([или](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=let%20res%20%3D%2050%3B%0Aconst%20grade%20%3D%20((result)%20%3D%3E%20%7B%0A%20%20if%20(result%20%3C%2050)%0A%20%20%20%20return%20%22bad%22%3B%0A%20%20if%20(result%20%3C%2090)%0A%20%20%20%20return%20%22good%22%3B%0A%20%20return%20%22excellent%22%3B%0A%7D)(res)%3B%20%2F%2F%20grade%20%3D%20%22good%22%3B))  
2. [Немедленно вызываемые функции](http://getinstance.info/articles/javascript/immediately-invoked-function-expressions/)

**Итераторы**

Избегайте цикл перебора *for...in*

const shared = { foo: "foo" };

const obj = Object.create(shared, {

bar: {

value: "bar",

enumerable: true

}

});

/\* плохо \*/

for (var prop in obj) {

if (obj.hasOwnProperty(prop))

console.log(prop); // => bar

}

/\* хорошо \*/

Object.keys(obj).forEach(prop => console.log(prop));

Разница между циклом и методом в том, что цикл перечисляет свойства и из цепочки прототипов.  
  
**К прочтению**:  
1. Babeljs (пример в ES5): [плохо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20shared%20%3D%20%7B%20foo%3A%20%22foo%22%20%7D%3B%0Aconst%20obj%20%3D%20Object.create(shared%2C%20%7B%0A%20%20bar%3A%20%7B%0A%20%20%20%20value%3A%20%22bar%22%2C%0A%20%20%20%20enumerable%3A%20true%0A%20%20%7D%0A%7D)%3B%0A%0A%2F*%20%D0%BF%D0%BB%D0%BE%D1%85%D0%BE%20*%2F%0Afor%20(var%20prop%20in%20obj)%20%7B%0A%20%20if%20(obj.hasOwnProperty(prop))%0A%20%20%20%20console.log(prop)%3B%0A%7D) vs [хорошо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20shared%20%3D%20%7B%20foo%3A%20%22foo%22%20%7D%3B%0Aconst%20obj%20%3D%20Object.create(shared%2C%20%7B%0A%20%20bar%3A%20%7B%0A%20%20%20%20value%3A%20%22bar%22%2C%0A%20%20%20%20enumerable%3A%20true%0A%20%20%7D%0A%7D)%3B%0A%0A%2F*%20%D1%85%D0%BE%D1%80%D0%BE%D1%88%D0%BE%20*%2F%0AObject.keys(obj).forEach(prop%20%3D%3E%20console.log(prop))%3B)  
2. [Использование hasOwnProperty для проверки существования свойства](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Object/hasOwnProperty#Example:_Using_hasOwnProperty_to_test_for_a_property.27s_existence) и [обход свойств объекта](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Object/keys#Examples)

**Объект как ассоциативный массив**

В то время как объекты можно законно использовать, как правило, ассоциативный массив (*Map*) лучше и мощнее. Если сомневаетесь, тогда точно используйте *Map*.

/\* плохо \*/

const me = {

name: "Ben",

age: 30

};

var meSize = Object.keys(me).length;

meSize; // => 2

me.country = "Belgium"; // можно

meSize++;

meSize; // => 3

/\* хорошо \*/

const me = new Map();

me.set("name", "Ben");

me.set("age", 30);

me.size; // => 2

me.set("country", "Belgium");

me.size; // => 3

**К прочтению**:  
1. Babeljs (пример в ES5): [плохо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20me%20%3D%20%7B%0A%20%20name%3A%20%22Ben%22%2C%0A%20%20age%3A%2030%0A%7D%3B%0A%0A%2F%2F%20me%20%3D%201%3B%20%2F%2F%20error!%20%D1%82%D0%B0%D0%BA%20%D0%BA%D0%B0%D0%BA%20%D0%BD%D0%B5%D0%BB%D1%8C%D0%B7%D1%8F%20%D0%B8%D0%B7%D0%BC%D0%B5%D0%BD%D0%B8%D1%82%D1%8C%20%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%BA%D1%81%D1%82%20%D0%BA%D0%BE%D0%BD%D1%81%D1%82%D0%B0%D0%BD%D1%82%D1%8B%0A%2F%2F%20%D0%BC%D0%BE%D0%B6%D0%BD%D0%BE%20%D0%B4%D0%BE%D0%B1%D0%B0%D0%B2%D0%BB%D1%8F%D1%82%D1%8C%20%D0%BA%D0%BB%D1%8E%D1%87%D0%B8%20%D0%B8%20%D0%B8%D0%B7%D0%BC%D0%B5%D0%BD%D1%8F%D1%82%D1%8C%20%D0%B8%D1%85%0A%0Avar%20meSize%20%3D%20Object.keys(me).length%3B%0AmeSize%3B%20%2F%2F%20%3D%3E%202%0Ame.country%20%3D%20%22Belgium%22%3B%0AmeSize%2B%2B%3B%0AmeSize%3B%20%2F%2F%20%3D%3E%203) vs [хорошо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20me%20%3D%20new%20Map()%3B%0Ame.set(%22name%22%2C%20%22Ben%22)%3B%0Ame.set(%22age%22%2C%2030)%3B%0Ame.size%3B%20%2F%2F%20%3D%3E%202%0Ame.set(%22country%22%2C%20%22Belgium%22)%3B%0Ame.size%3B%20%2F%2F%20%3D%3E%203)  
2. [Коллекция для хранения записей вида ключ: значение](https://learn.javascript.ru/set-map#map)

**Каррирование**

Каррирование является мощной, но в свою очередь чужеродной парадигмой для многих разработчиков. Не злоупотребляйте.

/\* плохо \*/

const sum = a => b => a + b;

sum(5)(3); // => 8

/\* хорошо \*/

const sum = (a, b) => a + b;

sum(5, 3); // => 8

**К прочтению**:  
1. Babeljs (пример в ES5): [плохо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20sum%20%3D%20a%20%3D%3E%20b%20%3D%3E%20a%20%2B%20b%3B%0Asum(5)(3)%3B%20%2F%2F%20%3D%3E%208) vs [хорошо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20sum%20%3D%20(a%2C%20b)%20%3D%3E%20a%20%2B%20b%3B%0Asum(5%2C%203)%3B%20%2F%2F%20%3D%3E%208)  
2. [Каррирование](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%80%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [примеры](http://true-coder.ru/javascript/karrirovanie-karring-v-javascript.html)

**Читабельность**

Не скрывайте намерения вашего кода, используя умные трюки.

/\* плохо \*/

foo || doSomething();

/\* плохо \*/

void function() { /\* IIFE \*/ }();

/\* плохо \*/

const n = ~~3.14;

/\* хорошо \*/

if (!foo) doSomething();

/\* хорошо \*/

(function() { /\* IIFE \*/ }());

/\* хорошо \*/

const n = Math.floor(3.14);

**К прочтению**:  
1. [Об идеальном коде и суровой реальности](https://habrahabr.ru/post/167877/)

**Повторное использование кода**

Не бойтесь создавать много мелких и многоразовых функций, они очень полезны.

/\* плохо \*/

let last = arr[arr.length - 1];

/\* плохо \*/

const product = (a, b) => a \* b;

const triple = n => n \* 3;

/\* хорошо \*/

const first = arr => arr[0]; // => first(arr);

const last = arr => first(arr.slice(-1)); // => last(arr);

/\* хорошо \*/

const product = (a, b) => a \* b;

const triple = product.bind(null, 3);  
**К прочтению**:  
1. Babeljs (пример в ES5): [плохо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=let%20last%20%3D%20arr%5Barr.length%20-%201%5D%3B%20%2F*%20%D0%BF%D0%BB%D0%BE%D1%85%D0%BE%20*%2F%0A%0A%2F*%20%D0%BF%D0%BB%D0%BE%D1%85%D0%BE%20*%2F%0Aconst%20product%20%3D%20(a%2C%20b)%20%3D%3E%20a%20*%20b%3B%0Aconst%20triple%20%3D%20n%20%3D%3E%20n%20*%203%3B) vs [хорошо](https://babeljs.io/repl/#?experimental=false&evaluate=true&loose=false&spec=false&code=const%20first%20%3D%20arr%20%3D%3E%20arr%5B0%5D%3B%20%2F%2F%20%3D%3E%20first(arr)%3B%0Aconst%20last%20%3D%20arr%20%3D%3E%20first(arr.slice(-1))%3B%20%2F%2F%20%3D%3E%20last(arr)%3B%0A%0A%2F*%20%D1%85%D0%BE%D1%80%D0%BE%D1%88%D0%BE%20*%2F%0Aconst%20product%20%3D%20(a%2C%20b)%20%3D%3E%20a%20*%20b%3B%0Aconst%20triple%20%3D%20product.bind(null%2C%203)%3B)

**Зависимости**

Минимизируйте зависимости стороннего кода, если вы не в курсе как он работает. Не нужно загружать всю библиотеку ради нескольких методов, которые легко воспроизводимы:

/\* плохо \*/

var \_ = require("underscore");

\_.compact(["foo", 0]));

\_.unique(["foo", "foo"]);

\_.union(["foo"], ["bar"], ["foo"]);

/\* хорошо \*/

const compact = arr => arr.filter(el => el);

const unique = arr => [...Set(arr)];

const union = (...arr) => unique([].concat(...arr));

compact(["foo", 0]);

unique(["foo", "foo"]);

union(["foo"], ["bar"], ["foo"]);