# Соглашения по стилю кодирования

Во время работы над проектом JavaScript всегда используйте одни и те же соглашения по стилю кодирования скриптов.

Соглашения по кодированию являются своего рода руководством по стилю написания кода программы. Обычно это:

* Правила присвоения имен и декларации переменных и функций.
* Правила использования пробелов, отступов и комментариев.
* Практики и принципы программирования.

Благодаря соглашениям по кодированию:

* улучшается читаемость кода
* облегчается процесс поддержки программы.

В качестве соглашений по кодированию могут выступать как задокументированные правила, так и ваши личные привычки написания кода.

Имена переменных

Обычно для имен идентификаторов (переменных и функций) используется так называемый "горбатый регистр", начинающийся с маленькой буквы. Все имена начинаются с буквы.

firstName = "John";

lastName = "Doe";

price = 19.90;

tax = 0.20;

fullPrice = price + (price \* tax);

Пробелы вокруг операторов

Всегда ставьте пробелы вокруг операторов ( = + - \* / ) и после запятых:

var x = y + z;

var values = ["Volvo", "Saab", "Fiat"];

Отступы

Всегда используйте 4 пробела для отступа блоков кода:

function toCelsius(fahrenheit) {

return (5 / 9) \* (fahrenheit - 32);

}

**Внимание!** Не используйте для отступов табуляцию. Разные редакторы интерпретируют табуляцию по разному.

Правила для выражений

Общие правила для простых выражений:

* Всегда заканчивайте простые выражения точкой с запятой.

var values = ["Volvo", "Saab", "Fiat"];

var person = {

firstName: "John",

lastName: "Doe",

age: 50,

eyeColor: "blue"

};

Общие правила для сложных (составных) выражений:

* Открывающую фигурную скобку ставьте в конце первой строки.
* Перед открывающей фигурной скобкой ставьте пробел.
* Закрывающую фигурную скобку ставьте на новой строке без пробелов перед ней.
* Не заканчивайте сложные выражения точкой с запятой.

Функции:

function toCelsius(fahrenheit) {

return (5 / 9) \* (fahrenheit - 32);

}

Циклы:

for (i = 0; i < 5; i++) {

x += i;

}

Условия:

if (time < 20) {

greeting = "Добрый день";

} else {

greeting = "Добрый вечер";

}

Правила для объектов

Общие правила для объявления объектов:

* Ставьте открывающую фигурную скобку на той же строчке, что и имя объекта.
* Между именем свойства и его значением ставьте двоеточие с пробелом.
* Строковые значения заключайте в кавычки, а числовые нет.
* Не добавляйте запятую после последней пары "свойство-значение".
* Закрывающую фигурную скобку ставьте на новой строке без пробелов перед ней.
* Всегда после определения объекта ставьте точку с запятой.

var person = {

firstName: "John",

lastName: "Doe",

age: 50,

eyeColor: "blue"

};

Короткие определения объектов можно записывать в сжатой форме, на одной строке, используя пробелы только между свойствами, как в следующем примере:

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

Длина строки меньше 80 символов

Для улучшения читаемости строка кода не должна превышать длину в 80 символов.

Если выражение JavaScript не помещается на одной строке, то лучшее место для переноса, это после оператора или запятой:

document.getElementById("demo").innerHTML =

"Привет, мир!";

Соглашение по именам

При написании кода всегда используйте одни и те же соглашения по именам. Например:

* Имена переменных и функций пишутся в "горбатом регистре", начинающимся с маленькой буквы - имяПеременной
* Имена глобальных переменных пишутся большими буквами - ИМЯПЕРЕМЕННОЙ (Мы не советуем это делать, но это вполне обычная практика)
* Имена констант (вроде PI) пишутся большими буквами

Что лучше использовать в именах переменных — дефис (имя-переменной), "горбатый регистр" (имяПеременной) или подчеркивание (имя\_переменной)?

Этот вопрос часто обсуждается программистами. И ответ на него зависит от того, кого вы спрашиваете:

**Дефис в HTML и CSS**  
Атрибуты в HTML5 могут начинаться с сочетания data- (data-quantity, data-price).

В CSS дефис используется в именах свойств (font-size).

В JavaScript использование дефиса в именах запрещено, так как он может быть ошибочно принят за оператор вычитания.

**Подчеркивание**  
Многие программисты, особенно работающие с базами данных SQL, предпочитают в именах использовать символ подчеркивания (date\_of\_birth).

Часто подчеркивание используется в документации по PHP.

**Верхний "горбатый регистр" (PascalCase)**  
PascalCase часто используют программисты на языке C.

**Нижний "горбатый регистр" (camelCase)**  
camelCase используется самим JavaScript, jQuery и другими библиотеками JavaScript.

**Внимание!** Не начинайте имена с символа доллара ($). Это приведет к конфликту с многими библиотеками JavaScript.

Загрузка JavaScript в HTML

Для загрузки внешних скриптов используйте простой синтаксис (атрибут ***type*** не нужен):

<script src="//msiter.ru/myscript.js"></script>

Доступ к элементам HTML

Следствием "неопрятного" стиля в HTML могут стать ошибки в JavaScript.

Следующие два выражения JavaScript дадут разные результаты:

var obj = getElementById("Demo")

var obj = getElementById("demo")

По возможности в HTML используйте те же соглашения по именам (как в JavaScript).

Расширения файлов

Файлы HTML должны иметь расширение .html (не .htm).

Файлы CSS должны иметь расширение .css.

Файлы JavaScript должны иметь расширение .js.

В названиях файлов используйте только маленькие буквы

Большинство веб-серверов (Apache, Unix) регистрозависимы в отношении имен файлов: нельзя получить доступ к файлу с именем london.jpg как London.jpg.

Другие же веб-сервера (Microsoft, IIS) регистронезависимы: обратиться к файлу с именем london.jpg можно как London.jpg или london.jpg.

Если вы смешиваете верхний и нижний регистры, то вы должны быть чрезвычайно последовательным.

Если вы смените регистронезависимый сервер на регистрозависимый, то даже малейшая ошибка может нарушить работу вашего сайта.

Чтобы избежать этой проблемы, всегда в названиях файлов используйте нижний регистр.

Быстродействие

Соглашения по кодированию не предназначены и не учитываются компьютерами. Большинство правил написания кода мало влияют на процесс выполнения программы.

Отступы и дополнительные пробелы не имеют особого значения в небольших скриптах.

Во время разработки скрипта читаемость кода имеет большее значение. Но большие скрипты на действующем сайте должны быть минифицированы.

# Хорошая практика программирования на Javascript

Избегать **глобальных переменных**, избегать оператора **new**, избегать оператора **==**, избегать функции **eval()**.

Избегайте глобальных переменных

Старайтесь сводить к минимуму использование глобальных переменных.

Сюда относятся все типы данных, объекты и функции.

Дело в том, что глобальные переменные и функции могут переписываться другими скриптами.

Вместо этого используйте локальные переменные и учитесь использовать *замыкание*.

Всегда декларируйте локальные переменные

Все переменные, используемые в функции, должны декларироваться как **локальные переменные**.

Локальные переменные должны декларироваться при помощи ключевого слова **var**, иначе они станут глобальными переменными.

**Внимание!** Строгий режим не допускает использование недекларированных переменных.

Декларирование в верху области видимости

Хорошей практикой программирования на JavaScript является практика размещения всех деклараций в верху скрипта или функции.

Это позволит:

* получить более чистый код
* обеспечить единое место для отслеживания локальных переменных
* избегать нежелательных (неявных) глобальных переменных
* снизить вероятность появления нежелательных повторных деклараций

// Декларируем в начале

var firstName, lastName, price, discount, fullPrice;

// Используем потом

firstName = "John";

lastName = "Doe";

price = 19.90;

discount = 0.10;

fullPrice = price \* 100 / discount;

Это также относится и к переменным цикла:

// Декларируем в начале

var i;

// Используем потом

for (i = 0; i < 5; i++) {

**Внимание!** По умолчанию, JavaScript перемещает все декларации в верх области видимости ([Поднятие в Javascript](https://msiter.ru/tutorials/javascript/js_hoisting)).

Инициализируйте переменные

По хорошему всегда следует инициализировать переменные во время их декларации.

Это позволит:

* получить более чистый код
* обеспечить единое место для инициализации переменных
* избежать неопределенных значений

// Декларируем и инициализируем в начале

var firstName = "",

lastName = "",

price = 0,

discount = 0,

fullPrice = 0,

myArray = [],

myObject = {};

Никогда не декларируйте объекты Number, String, Boolean

С числами, строками и логическими значениями всегда обращайтесь как к примитивам, а не объектам.

Декларирование этих типов как объектов замедляет скорость выполнения скрипта и приводит к неприятным побочным эффектам:

var x = "John";

var y = new String("John");

(x === y) // возвращает false, так как x — строка, а y — объект

Или даже хуже:

var x = new String("John");

var y = new String("John");

(x == y) // возвращает false, так как нельзя сравнивать объекты

Не используйте команду new Object()

* Используйте { } вместо new Object()
* Используйте "" вместо new String()
* Используйте 0 вместо new Number()
* Используйте false вместо new Boolean()
* Используйте [ ] вместо new Array()
* Используйте /()/ вместо new RegExp()
* Используйте function (){ } вместо new Function()

var x1 = {}; // новый объект

var x2 = ""; // новая строка

var x3 = 0; // новое число

var x4 = false; // новое значение boolean

var x5 = []; // новый объект массива

var x6 = /()/; // новое регулярное выражение

var x7 = function(){}; // новый объект функции

Помните об автоматическом преобразовании типов

Помните о том, что числа могут быть случайно преобразованы в строки или значение **NaN** (Не число).

JavaScript — слабо типизированный язык программирования. Переменные могут содержать значения разных типов, при этом переменные могут менять тип, содержащихся в них данных:

var x = "Hello"; // typeof x — строка

x = 5; // изменяем тип, теперь typeof x — число

Во время математических операций JavaScript может преобразовывать числа в строки:

var x = 5 + 7; // x.valueOf() возвращает 12, typeof x — число

var x = 5 + "7"; // x.valueOf() возвращает 57, typeof x — строка

var x = "5" + 7; // x.valueOf() возвращает 57, typeof x — строка

var x = 5 - 7; // x.valueOf() возвращает -2, typeof x — число

var x = 5 - "7"; // x.valueOf() возвращает -2, typeof x — число

var x = "5" - 7; // x.valueOf() возвращает -2, typeof x — число

var x = 5 - "x"; // x.valueOf() возвращает NaN, typeof x — число

Вычитание строки из строки не генерирует ошибку, а возвращает значение **NaN** (Не число):

"Hello" - "Dolly" // возвращает NaN

Используйте сравнение ===

Оператор сравнения == всегда приводит сравниваемые значения к одинаковым типам.

Оператор сравнения === сравнивает как сами значения, так и их типы:

0 == ""; // true

1 == "1"; // true

1 == true; // true

0 === ""; // false

1 === "1"; // false

1 === true; // false

Используйте параметры по умолчанию

Если функция вызывается с пропущенным параметром, то этому параметру присваивается значение **undefined**.

Значения **undefined** могут сломать ваш код. Если взять за правило присваивать параметрам значения по умолчанию, то это будет хорошей привычкой.

function myFunction(x, y) {

if (y === undefined) {

y = 0;

}

}

Всегда заканчивайте выражение switch оператором default

Всегда заканчивайте выражение **switch** оператором **default**. Даже если вы думаете, что он совсем не нужен.

switch (new Date().getDay()) {

case 0:

day = "Воскресенье";

break;

case 1:

day = "Понедельник";

break;

case 2:

day = "Вторник";

break;

case 3:

day = "Среда";

break;

case 4:

day = "Четверг";

break;

case 5:

day = "Пятница";

break;

case 6:

day = "Суббота";

break;

default:

day = "Неизвестно";

}

Старайтесь не использовать eval()

Функция **eval()** используется, чтобы запустить текст, как код. Почти во всех случаях в этом нет необходимости.

Из-за того, что эта функция позволяет запускать самостоятельный код, она еще может являться и источником проблем безопасности.

# Быстродействие программы на Javascript

Каким образом можно ускорить работу кода JavaScript.

Уменьшайте активность в циклах

В программировании циклы используются часто.

Каждое выражение в цикле, включая само выражение for, выполняется при каждом проходе (итерации) цикла.

Выражения или присваивания, вынесенные за пределы цикла, позволят циклу выполняться быстрее.

Плохо:

var i;

for (i = 0; i < arr.length; i++) {

Лучше:

var i;

var l = arr.length;

for (i = 0; i < l; i++) {

В первом варианте код обращается к свойству массива length во время каждой итерации цикла.

Во втором варианте код обращается к свойству массива length один раз за пределами цикла, что ускоряет работу цикла.

Уменьшайте количество обращений к DOM

По сравнению с другими выражениями JavaScript обращение к HTML DOM очень медленный процесс.

Если вам необходимо обратиться к элементу DOM несколько раз, то сохраните его в локальной переменной и затем обращайтесь к ней:

var obj;

obj = document.getElementById("demo");

obj.innerHTML = "Привет";

Уменьшайте размер DOM

Старайтесь, чтобы в HTML DOM было небольшое количество элементов.

Это улучшает загрузку страницы и ускоряет ее рендеринг (отображение), особенно на медленных устройствах.

Каждая попытка поиска по DOM (вроде getElementsByTagName) даст выгоду от небольшого дерева DOM.

Избегайте ненужных переменных

Не создавайте новые переменные для значений, которые вы не планируете использовать в дальнейшем.

Часто следующий код:

var fullName = firstName + " " + lastName;

document.getElementById("demo").innerHTML = fullName;

Можно заменить на такой:

document.getElementById("demo").innerHTML = firstName + " " + lastName

Откладывайте загрузку кода JavaScript

Если разместить скрипты внизу тела веб-страницы, то браузер сначала загрузит код самой страницы.

Дело в том, что когда загружается скрипт, ничего другого браузер загружать не будет. Кроме этого, любой парсинг и рендеринг также могут быть заблокированы.

Согласно спецификации HTTP браузер не может одновременно загружать больше двух компонентов.

В качестве альтернативы можно указать атрибут **defer="true"** в теге script. Этот атрибут определяет, что скрипт должен выполниться только после того, как страница будет полностью сформирована в браузере. Но работает этот атрибут только с внешними скриптами, и в использовании имеет свои подводные камни.

Можно добавить скрипт при помощи кода, после того, как страница будет загружена:

<script>

window.onload = function() {

var element = document.createElement("script");

element.src = "myScript.js";

document.body.appendChild(element);

};

</script>

Не используйте ключевое слово with

Избегайте использование ключевого слова **with**. Оно отрицательно влияет на скорость работы скрипта. Кроме этого оно загромождает области видимости JavaScript.

В строгом режиме использование ключевого слова with запрещено.

# Зарезервированные слова в Javascript

В JavaScript нельзя использовать следующие зарезервированные слова в качестве имен переменных, функций или меток:

|  |  |  |  |
| --- | --- | --- | --- |
| abstract | arguments | await\* | boolean |
| break | byte | case | catch |
| char | class\* | const | continue |
| debugger | default | delete | do |
| double | else | enum\* | eval |
| export\* | extends\* | false | final |
| finally | float | for | function |
| goto | if | implements | import\* |
| in | instanceof | int | interface |
| let\* | long | native | new |
| null | package | private | protected |
| public | return | short | static |
| super\* | switch | synchronized | this |
| throw | throws | transient | true |
| try | typeof | var | void |
| volatile | while | with | yield |

Слова, помеченные знаком \*, новые в ECMAScript 5 и 6.

Удаленные зарезервированные слова

Следующие зарезервированные слова были удалены в стандарте ECMAScript 5/6:

|  |  |  |  |
| --- | --- | --- | --- |
| abstract | boolean | byte | char |
| double | final | float | goto |
| int | long | native | short |
| synchronized | throws | transient | volatile |

Не используйте эти слова в качестве переменных, так как стандарт ECMAScript 5/6 поддерживается не всеми браузерами.

Объекты, свойства и методы в JavaScript

Также следует избегать использования не по прямому назначению имен встроенных объектов, свойств и методов JavaScript:

|  |  |  |  |
| --- | --- | --- | --- |
| Array | Date | eval | function |
| hasOwnProperty | Infinity | isFinite | isNaN |
| isPrototypeOf | length | Math | NaN |
| name | Number | Object | prototype |
| String | toString | undefined | valueOf |

Зарезервированные слова Java

JavaScript часто используется вместе с Java. Поэтому также не следует использовать в качестве идентификаторов JavaScript некоторых объектов и свойств Java:

|  |  |  |  |
| --- | --- | --- | --- |
| getClass | java | JavaArray | javaClass |
| JavaObject | JavaPackage |  |  |

Другие зарезервированные слова

JavaScript может использоваться в качестве языка программирования во многих приложениях. Вы также не должны использовать имена объектов и свойств HTML и Window:

|  |  |  |  |
| --- | --- | --- | --- |
| alert | all | anchor | anchors |
| area | assign | blur | button |
| checkbox | clearInterval | clearTimeout | clientInformation |
| close | closed | confirm | constructor |
| crypto | decodeURI | decodeURIComponent | defaultStatus |
| document | element | elements | embed |
| embeds | encodeURI | encodeURIComponent | escape |
| event | fileUpload | focus | form |
| forms | frame | innerHeight | innerWidth |
| layer | layers | link | location |
| mimeTypes | navigate | navigator | frames |
| frameRate | hidden | history | image |
| images | offscreenBuffering | open | opener |
| option | outerHeight | outerWidth | packages |
| pageXOffset | pageYOffset | parent | parseFloat |
| parseInt | password | pkcs11 | plugin |
| prompt | propertyIsEnum | radio | reset |
| screenX | screenY | scroll | secure |
| select | self | setInterval | setTimeout |
| status | submit | taint | text |
| textarea | top | unescape | untaint |
| window |  |  |  |

Обработчики событий HTML

В дополнение ко всему вышесказанному следует избегать использования имен всех обработчиков событий HTML.

Например:

|  |  |  |  |
| --- | --- | --- | --- |
| onblur | onclick | onerror | onfocus |
| onkeydown | onkeypress | onkeyup | onmouseover |
| onload | onmouseup | onmousedown | onsubmit |