**Написание хорошего кода**

[Главная](https://webref.ru/) / [Вёрстка](https://webref.ru/layout) / [Уроки по HTML и CSS](https://webref.ru/layout/learn-html-css) / Написание хорошего кода

Для написания HTML и CSS много что следует изучить — разные элементы, атрибуты, свойства, значения и т. д. Основной целью каждого урока до этого момента было объяснение этих различных компонентов HTML и CSS, в надежде помочь вам понять базовые основы обоих языков. Данный урок делает шаг назад и смотрит на более абстрактную картину HTML и CSS.

Более точно, этот урок посвящён передовой практики написания кода на HTML и CSS. Описанные методы служат в качестве общего каркаса для написания HTML и CSS. Они применяются в каждом уроке и о них всегда следует помнить при программировании.

Когда вы просматриваете эти методы, думайте о том, что они также могут применяться в других областях или языках программирования. Например, использование комментариев для организации кода (как мы рассмотрим в этом уроке) выгодно во всех языках программирования. Сохраняйте открытость мышления и подумайте, как вы можете в полной мере воспользоваться каждым методом.

**Методы написания HTML**

Набор передовых практик предназначен для сохранения кода простым и хорошо организованным. Основные методы в HTML не различаются, их целью является написание разметки, хорошо структурированной и совместимой со стандартами. Описанные здесь рекомендации предлагают краткое введение в методы написания HTML и список ни в коем случае не исчерпывается ими.

**Разметка по стандартам**

HTML по своей природе относится к прощающему языку, который допускает плохой код для выполнения и отображения с различной степенью точности. Успешное отображение, однако, не означает, что наш код семантически правильный или гарантирует, что код совместим со стандартами. Кроме того, плохой код непредсказуем и вы не можете быть уверены, что получите при отображении. Мы должны быть внимательны при написании HTML — во вложениях закрывать все элементы правильно, использовать идентификаторы и классы надлежащим образом и всегда проверять наш код.

Следующий код содержит несколько ошибок, в том числе использует значение intro для атрибута id несколько раз, тогда как он должен быть уникальным значением, элементы <p> и <strong> закрываются в неправильном порядке в первом абзаце, а элемент <р > во втором абзаце не закрывается вообще.

Плохой код

<p id="intro">Новинки в меню сегодня включают <strong>карамельный яблочный сидр и утренние блинчики</p>.</strong>

<p id="intro">Карамельный яблочный сидр вкусный.

Хороший код

<p class="intro">Новинки в меню сегодня включают <strong>карамельный яблочный сидр и утренние блинчики</strong>.</p>

<p class="intro">Карамельный яблочный сидр вкусный.</p>

**Использование семантических элементов**

Библиотека элементов в HTML является довольно большой, состоит более чем из 100 элементов, доступных для применения. Решить, какие элементы использовать для описания разного содержимого, может быть сложно, но эти элементы являются основой семантики. Мы должны исследовать и перепроверить свой код и убедиться что используем соответствующие семантические элементы. Люди будут благодарить нас в долгосрочной перспективе за создание более доступного сайта, а ваш HTML будет несравненно легче стилизовать. Если вы не уверены в своём коде, найдите друга, который вам поможет и выполнит рутинный обзор кода.

Ниже в HTML не используются подходящие элементы заголовка и абзаца, вместо этого применяются бессмысленные элементы стилизации и группирования содержимого.

Плохой код

<span class="heading"><strong>С возвращением</span></strong>

<br><br>

Прошло некоторое время. Чем вы занимались?

<br><br>

Хороший код

<h1>С возвращением</h1>

<p>Прошло некоторое время. Чем вы занимались?</p>

**Используйте правильную структуру документа**

Как упоминалось ранее, HTML является прощающим языком, следовательно, страницы будут отображаться без использования доктайп или элементов <html>, <head> и <body>. Без доктайпа и этих структурных элементов, страницы не будут отображаться правильно в любом браузере.

Мы всегда должны быть уверены, что используем правильную структуру документа, в том числе доктайп и элементы <html>, <head> и <body>. Это делает наши страницы совместимыми со стандартами и полностью семантическими и гарантирует, что они будут отображаться как нам нужно.

Плохой код

<html>

<h1>Привет, мир!</h1>

<p>Это веб-страница.</p>

</html>

Хороший код

<!DOCTYPE html>

<html>

<head>

<title>Привет, мир</title>

</head>

<body>

<h1>Привет, мир!</h1>

<p>Это веб-страница.</p>

</body>

</html>

**Сохраняйте синтаксис организованным**

При росте страниц управлять HTML становится непростой задачей. К счастью, есть несколько простых правил, которые могут помочь сохранить наш синтаксис чистым и организованным. К ним относятся следующие:

* используйте строчные буквы в именах элементов, атрибутах и значений; делайте отступ вложенных элементов;
* используйте строго двойные кавычки, а не одинарные или вообще без кавычек;
* уберите слэш в конце самозакрывающихся элементов; опустите значения для логических атрибутов.

Соблюдение этих правил поможет сохранить наш код аккуратным и разборчивым. Взгляните на два набора HTML ниже, хороший код легче воспринимается и понятнее.

Плохой код

<Aside>

<h3>Чикаго</h3>

<H5 HIDDEN='HIDDEN'>Город в Иллинойсе</H5>

<img src=chicago.jpg alt="Чикаго, третий по числу жителей город в Соединенных Штатах" />

<ul>

<li>234 квадратных миль</li>

<li>2.715 миллионов человек</li>

</ul>

</ASIDE>

Хороший код

<aside>

<h3>Чикаго</h3>

<h5 hidden>Город в Иллинойсе</h5>

<img src="chicago.jpg" alt="Чикаго, третий по числу жителей город в Соединенных Штатах">

<ul>

<li>234 квадратных миль</li>

<li>2.715 миллионов человек</li>

</ul>

</aside>

**Используйте практичные значения идентификаторов и классов**

Создание значений идентификаторов и классов может быть одной из наиболее сложных частей написания HTML. Эти значения должны быть практичными, относящиеся к самому содержимому, а не к его стилю. Использование значения red для описания красного текста не является идеальным, так как описывает внешний вид содержимого. Если когда-нибудь понадобится стиль текста сменить на синий, то должен поменяться не только CSS, но также и HTML в каждом случае, где встречается класс red.

Ниже в HTML предполагается, что предупреждение будет красным. Однако, когда стиль предупреждения изменится на оранжевый, имя класса red потеряет смысл и, скорее всего, приведёт к путанице.

Плохой код

<p class="red">Ошибка! Попробуйте ещё раз.</p>

Хороший код

<p class="alert">Ошибка! Попробуйте ещё раз.</p>

**Используйте альтернативный текст для изображений**

Изображения всегда должны включать атрибут alt. Экранные ридеры и другое программное обеспечение полагаются на атрибут alt, чтобы обеспечить контекст для изображений.

Значение атрибута alt должно описывать, что содержит изображение. Если изображение не содержит ничего актуального, атрибут alt по прежнему должен быть включён, но при этом его значение должно быть пустым, чтобы экранные ридеры его игнорировали, а не читали имя графического файла.

Кроме того, если изображение не несёт важного смысла, возможно, к примеру, как часть пользовательского интерфейса, оно при возможности должно быть включено в CSS как фоновое изображение, а не как элемент <img>.

Плохой код

<img src="puppy.jpg">

Хороший код

<img src="puppy.jpg" alt="Красивый щенок дворняги">

**Отделяйте содержимое от стиля**

Никогда, никогда не используйте встроенные стили в HTML. Это создаёт страницы, которые слишком долго загружаются, трудно поддерживаются и вызывают головную боль у дизайнеров и разработчиков. Вместо этого используйте внешние таблицы стилей с помощью классов для целевых элементов и применяйте стили как надо.

Здесь в плохом коде любые желаемые изменения в стилях должны вноситься в HTML. Следовательно, эти стили не могут быть повторно использованы и скорее всего страдает согласованность стилей.

Плохой код

<p style="color: #393; font-size: 24px;">Спасибо!</p>

Хороший код

<p class="alert-success">Спасибо!</p>

**Избегайте лишних <div>**

При написании HTML легко увлечься, добавив элемент <div> здесь, элемент <div> там, для построения любых необходимых стилей. Хотя это работает и добавляет совсем немного наворотов на страницу, в итоге мы не уверены, что каждый элемент <div> делает.

Нам следует сделать всё, чтобы сохранить код небольшим и сократить разметку, связывая несколько стилей в одном элементе, при возможности. Кроме того, мы должны использовать подходящие структурные элементы HTML5.

Плохой код

<div class="container">

<div class="article">

<div class="headline">Заголовки по всему миру</div>

</div>

</div>

Хороший код

<div class="container">

<article>

<h1>Заголовки по всему миру</h1>

</article>

</div>

**Постоянно реорганизуйте код**

Со временем сайты и базовый код продолжают развиваться и расти, оставляя за собой совсем немного хлама. Не забудьте удалить старый код и стили как надо при редактировании страницы. Также выделите время, чтобы оценить и реорганизовать код после его написания, ищите способы, чтобы сократить его и сделать более управляемым.

**Методы написания CSS**

Подобно HTML, методы написания CSS сосредоточены на сохранении кода простым и хорошо организованным. CSS также содержит дополнительные принципы, связанные с некоторыми тонкостями языка.

**Организация кода через комментарии**

CSS-файлы могут стать весьма обширными, занимая сотни строк. Такие большие файлы делают поиск и редактирование наших стилей почти невозможным. Так что организуйте стили в логические группы. Затем перед каждой группой давайте комментарий, отмечая к чему относятся следующие стили.

При желании мы также можем использовать комментарии для построения оглавления в верхней части нашего файла. Это напомнит нам и другим, что именно содержится в файле со стилями.

Плохой код

header { ... }

article { ... }

.btn { ... }

Хороший код

/\* Основной заголовок \*/

header { ... }

/\* Избранная статья \*/

article { ... }

/\* Кнопки \*/

.btn { ... }

**Пишите CSS с помощью нескольких строк и пробелов**

При написании CSS важно поместить каждый селектор и описание правил на новой строке. Затем внутри каждого селектора мы хотим сделать отступ описаний.

После селектора и до первого описания идёт открывающая фигурная скобка, которая должна содержать перед ней пробел. В описании мы должны поставить пробел после двоеточия, затем идёт свойство и завершается точкой с запятой.

Такой код легко читать, а также редактировать. Когда весь код свален в одну строку без пробелов, его трудно просматривать и вносить изменения.

Плохой код

a,.btn{background:#aaa;color:#f60;font-size:18px;padding:6px;}

Хороший код

a,

.btn {

background: #aaa;

color: #f60;

font-size: 18px;

padding: 6px;

}

**Комментарии и пробелы**

Эти две рекомендации — организация кода через комментарии и использование нескольких строк и пробелов, относится не только к CSS, но и к HTML или любому другому языку. В целом, мы должны держать наш код организованным и хорошо документированным. Если определённая часть кода является более сложной, объясните, как она работает и что к ней относится в комментариях. Это помогает другим, работающим с тем же базовым кодом, а также самому себе, когда мы пересмотрим наш собственный код в будущем.

**Используйте подходящие имена классов**

Имена классов (или значения) должны быть модульными и относиться к содержимому внутри элемента, а не к его внешнему виду, насколько это возможно. Эти значения должны быть записаны таким образом, что они походили на синтаксис языка CSS. Соответственно, имена классов должны быть в нижнем регистре и использовать дефис в качестве разделителя.

Плохой код

.Red\_Box { ... }

Хороший код

.alert-message { ... }

**Выстраиваете правильные селекторы**

Селекторы CSS могут выйти из-под контроля, если они не тщательно поддерживаются. Селекторы легко могут стать слишком длинными и чересчур специфичными.

Чем длиннее селектор и больше включает уточнений, тем выше его специфичность. А чем выше специфичность, тем больше вероятность что селектор прервёт каскад CSS и вызовет нежелательные проблемы.

Кроме того, сохраняя специфичность наших селекторов как можно ниже, давайте не будем применять идентификаторы в наших селекторах. Идентификаторы излишне конкретны, быстро поднимают специфичность селектора и довольно часто нарушают каскад в CSS-файлах. Минусы идентификаторов далеко перевешивают их плюсы и разумнее будет избегать их.

Давайте использовать более короткие и в основном прямые селекторы. Указывайте только два-три уровня вложения и удалите столько чересчур длинных селекторов, насколько это возможно.

Плохой код

#aside #featured ul.news li a { ... }

#aside #featured ul.news li a em.special { ... }

Хороший код

.news a { ... }

.news .special { ... }

**Используйте конкретные классы при необходимости**

Бывают времена, когда селектор CSS настолько длинный и специфичный, что он больше не несёт смысла. Это создаёт задержки производительности и напряжение при управлении. В этом случае рекомендуется использовать одиночный класс. Применение класса к целевому элементу может создать больше кода в HTML, но позволит коду отображаться быстрее и уберёт любые препятствия для управления.

Например, если элемент <em> вложен в <h1> внутри элемента <aside> и всё это вложено в элемент <section>, то селектор может выглядеть как aside h1 em. Как только элемент <em> перемещается из <h1>, стили больше не будут применяться. Более гибким селектором будет использование класса, такого как text-offset, для элемента <em>.

Плохой код

section aside h1 em { ... }

Хороший код

.text-offset { ... }

**Используйте сокращённые свойства и значения**

Одной из особенностей CSS является возможность использовать сокращённые свойства и значения. Большинство свойств и значения допускают сокращённый вариант написания. В качестве примера, вместо написания четырёх разных свойств на основе margin- и их значений для задания отступов вокруг всех четырёх сторон элемента, используйте единственное свойство margin, которое установит значения для всех четырёх сторон одновременно. Использование сокращённого варианта позволяет быстро установить и определить стили.

При этом, когда вам требуется установить одно значение, не следует применять сокращённый вариант. Если для блока нужен только нижний margin, используйте одно свойство margin-bottom. Это гарантирует, что другие значения margin не будут перезаписаны и мы можем легко определить, к какой стороне применяется margin без особых когнитивных усилий.

Плохой код

img {

margin-top: 5px;

margin-right: 10px;

margin-bottom: 5px;

margin-left: 10px;

}

button {

padding: 0 0 0 20px;

}

Хороший код

img {

margin: 5px 10px;

}

button {

padding-left: 20px;

}

**Используйте сокращённые шестнадцатеричные значения цвета**

При возможности используйте трёхсимвольное сокращённое шестнадцатеричное значение цвета и всегда пишите строчные символы в любом шестнадцатеричном значении цвета. Идея, опять же, в том, чтобы оставаться последовательным во избежание путаницы и принять синтаксис написания языка кода.

Плохой код

.module {

background: #DDDDDD;

color: #FF6600;

}

Хороший код

.module {

background: #ddd;

color: #f60;

}

**Отбросьте единицы у нулевых значений**

Один из способов для простого сокращения количества CSS при написании — это удаление единиц у нулевых значений. Независимо от того, какая единица длины применяется — пиксели, проценты, em и др. — ноль всегда ноль. Добавление единицы не нужно и не несёт никакой дополнительной ценности.

Плохой код

div {

margin: 20px 0px;

letter-spacing: 0%;

padding: 0px 5px;

}

Хороший код

div {

margin: 20px 0;

letter-spacing: 0;

padding: 0 5px;

}

**Группирование и выравнивание вендорных префиксов**

С CSS3 префиксы получили некоторую популярность, добавив немного кода в CSS-файлы. Работа по добавлению префиксов нередко имеет смысл при генерации стилей, однако, они должны храниться организованно. Соответственно, с целью написания кода, который легко читать и изменять, лучше группировать и сдвигать отдельные префиксы так, чтобы имена свойств располагались вертикально, как и их значения.

В зависимости от того, где расположен вендорный префикс — в свойстве или в значении, выравнивание может изменяться. Например, следующий хороший код сохраняет свойства background выровненными влево, в то время как функции linear-gradient() с префиксами сдвинуты так, чтобы их значения подгонялись по вертикали. Затем свойства box-sizing с префиксами сдвинуты так, чтобы все они подгонялись по вертикали.

Как всегда, цель — сделать стили проще для чтения и редактирования.

Плохой код

div {

background: -webkit-linear-gradient(#a1d3b0, #f6f1d3);

background: -moz-linear-gradient(#a1d3b0, #f6f1d3);

background: linear-gradient(#a1d3b0, #f6f1d3);

-webkit-box-sizing: border-box;

-moz-box-sizing: border-box;

box-sizing: border-box;

}

Хороший код

div {

background: -webkit-linear-gradient(#a1d3b0, #f6f1d3);

background: -moz-linear-gradient(#a1d3b0, #f6f1d3);

background: linear-gradient(#a1d3b0, #f6f1d3);

-webkit-box-sizing: border-box;

-moz-box-sizing: border-box;

box-sizing: border-box;

}

**Вендорные префиксы**

При использовании префиксов мы должны убедиться, что поместили версию свойства или значения без префикса в конец, после всех версий с префиксом. Это гарантирует, что браузеры, поддерживающие версию без префикса отобразят этот стиль в соответствии с его размещением в каскаде, читая стили с верхней части файла до низа.

Хорошей новостью является то, что браузеры в основном отходят от использования префиксов. Со временем это станет меньше беспокоить, однако, в настоящее время мы советуем перепроверить, какие стили требуют префикса и держать эти префиксы организованно.

**Модульность стилей для повторного использования**

CSS построен на том, что позволяет повторно использовать стили, в частности, с помощью классов. По этой причине, стили, назначенные классу, должны быть модульными и доступны для обмена между элементами при необходимости.

Если раздел новостей представлен в блоке, который включает границы, цвет фона и другие стили, класс news может показаться хорошим вариантом. Однако, те же самые стили могут также применяться к разделу предстоящих событий. Класс news не подходит в данном случае. Класс feat-box имел бы больше смысла и может быть широко использован по всему сайту.

Плохой код

.news {

background: #eee;

border: 1px solid #ccc;

border-radius: 6px;

}

.events {

background: #eee;

border: 1px solid #ccc;

border-radius: 6px;

}

Хороший код

.feat-box {

background: #eee;

border: 1px solid #ccc;

border-radius: 6px;

}

**Резюме**

Надеюсь, принципы написания красивых HTML и CSS становятся понятнее. Хотя каждый язык содержит свои собственные сложности, большинство из этих методов может быть поделено между двумя языками как и со многими другими языками программирования.

Лично мы должны сделать всё возможное, чтобы поддержать эти принципы, а при работе в команде мы также должны сделать всё возможное, чтобы обучить команду этим техникам. В свою очередь, у команды могут быть ценные предложения и методы, которым мы должны следовать.

Чтобы выделить некоторые главные темы этого урока, наши HTML и CSS всегда должны:

* быть хорошо организованными, так что их легко читать, редактировать и поддерживать;
* быть модульными и гибкими, что позволяет нам повторно использовать код и шаблоны по мере необходимости;
* выглядеть так, словно их писал один человек, даже если свой вклад внесли несколько людей.

Эти методы являются лишь началом и поскольку языки развиваются и мы пишем всё больше и больше HTML и CSS, мы разработаем новые методы. Это всё часть прекрасных знаний HTML и CSS.

Сейчас вы вооружены некоторыми мощными знаниями того, как создавать веб-сайты на HTML и CSS и я рад видеть, что вы с ними делаете. Держите меня в курсе о том, как это происходит и удачного вам созидания!

**Ресурсы и ссылки**

Каждый урок идёт с набором ресурсов для дополнительного обучения и находок. Ниже изложен длинный список ресурсов, а также полезных ссылок.

**HTML и CSS**

* [Mozilla Developer Network](https://developer.mozilla.org/en-US/) на Mozilla
* [Opera.Dev](http://dev.opera.com/) на Opera
* [HTML and CSS Tutorials](http://www.htmldog.com/) на HTML Dog
* [DocHub](http://dochub.io/) — Мгновенный поиск документации
* [Pears](http://pea.rs/) — Основные шаблоны разметки и стиля

**Дизайн для вдохновения**

* [Dribbble](http://dribbble.com/)
* [Pattern Tap](http://patterntap.com/)
* [Premium Pixels](http://www.premiumpixels.com/)

**Фреймворки и руководства по стилям**

* [Web Style Guide](http://webstyleguide.com/wsg3/index.html)
* [Bootstrap](http://twitter.github.com/bootstrap/)
* [Foundation](http://foundation.zurb.com/)
* [Skeleton Framework](http://getskeleton.com/)
* [Google HTML/CSS Style Guide](https://google-styleguide.googlecode.com/svn/trunk/htmlcssguide.xml)
* [GitHub Styleguide](https://github.com/styleguide/)

**Иконки**

* [Helveticons](http://hlvticons.ch/) на Goodbye Horses
* [Ion Icons](http://ionicons.com/) от Ben Sperry
* [Social Network Icon Pack](http://www.komodomedia.com/blog/2009/06/social-network-icon-pack/) на Rogie King
* [Fugue Icons](http://p.yusukekamiyamane.com/) на Yusuke Kamiyamane
* [famfamfam Icons](http://www.famfamfam.com/lab/icons/silk/) от Mark James
* [Pictos](http://pictos.cc/) от Drew Wilson
* [Picons](http://picons.me/)
* [The Noun Project](http://thenounproject.com/)

**Разное**

* [COLOURlovers](http://www.colourlovers.com/) — Тенденции цвета и палитры
* [ColorHexa](http://www.colorhexa.com/) — Энциклопедия цвета
* [Modernizr](http://modernizr.com/) — Библиотека обнаружения фич на JavaScript
* [jQuery](http://jquery.com/) — Навороченная JavaScript-библиотека
* [Google Hosted Libraries](https://developers.google.com/speed/libraries/devguide) — Сеть распределения контента для библиотек JavaScript
* [Copy Paste Character](http://copypastecharacter.com/) — Копирование «скрытых» символов