**Кодстайл**

**Что такое кодстайл и зачем он?**

Кодстайлом называют то, как разработчики организуют код в проекте. Это могут быть личные предпочтения в написании кода или контракт между программистами.

Когда вы создаёте проект в одиночку, вы сами устанавливаете удобные правила и придерживаетесь их. Тем не менее даже в таких ситуациях могут появиться проблемы, связанные с оформлением.

Например, вы можете создать и запустить проект, забыть о нём, а через пару месяцев вернуться к нему, чтобы что-то поменять. Почти всегда в таких случаях вы будете удивлены своему коду. Он будет гораздо хуже, чем тот, что вы писали вчера. Но аккуратность подхода и организации кода позволят быстрее вспомнить былое и решить задачу.

В этой статье мы поговорим про контракт между программистами. Они договариваются, как писать проект, чтобы тратить меньше сил на понимание кода друг-друга.

В проектах, где идёт коллективная разработка, кодстайл невероятно важен. Единообразие оформления, конвенции об использовании или неиспользовании отдельных приёмов написания кода прямо связаны с эффективностью. К написанию единообразного кода легче подключать новых людей, в коде других разработчиков проще разбираться.

Бывает, что договорённости о кодстайле выполняют продуктовые функции. Если для продукта важна поддержка устаревших браузеров, у вас есть ограничения на использование некоторых инструментов CSS.

В целом подход к описанию кодстайла можно разделить на три составляющие:

* как оформлять код,
* как организовывать код,
* какие инструменты использовать или не использовать для определённых случаев.

Договорённости могут отличаться в разных командах, но существуют общие принципы, хотя и дискуссионные. О них и поговорим, но сосредоточимся только на HTML и CSS — основных инструментах вёрстки.

**Общие принципы оформления HTML и CSS**

**Вложенность**

Все программисты сходятся в том, что блоки кода удобнее читать, когда они вложены друг в друга. Это означает, что родительские элементы визуально отделены от их потомков, а определяемые конструкции языка визуально отделены от способа их определения.

HTML и CSS — это языки, где отсутствие вложенности не влияет на работу кода.

Этот код:

Скопировать код

HTML

<ul><li>Пункт</li></ul>

Равносилен такому:

Скопировать код

HTML

<ul>

<li>Пункт</li>

</ul>

И даже такому:

Скопировать код

HTML

<ul>

<li>Пункт</li>

</ul>

Или такому:

Скопировать код

HTML

<ul>

<li>

Пункт

</li>

</ul>

С CSS то же самое:

Скопировать код

CSS

.aside,.block{background-color:blue;}

Аналогично:

Скопировать код

CSS

.aside,

.block {

background-color: blue;

}

В HTML принято писать так, чтобы дочерние элементы были отделены табуляцией внутри родительских блоков.

Скопировать код

HTML

<ul>

<li>Пункт</li>

</ul>

Скопировать код

HTML

<ul>

<li>

Пункт

</li>

</ul>

Оба приведённых выше варианта возможны. В обсуждении с командой вы поймёте, какой стиль написания оставить.

Обычно в работающих проектах уже есть решения. В таком случае вам придётся принять эти правила.

В случае с CSS есть консенсус относительно того, что:

* свойства должны быть отделены табуляцией внутри фигурных скобок,
* после селектора ставится пробел,
* каждое свойство пишут с новой строки.

Правила CSS отделяют друг от друга пустой строкой. Так заведено в большинстве команд.

Вот приемлемый вариант:

Скопировать код

CSS

.block-inverted {

color: #fff;

background-color: #000;

}

.block-common {

color: #000;

background-color: #fff;

}

На практике мы сталкивались с мнением, что в случае, когда у нас есть группа однотипных правил, в которой меняется только одно свойство, удобно это подчёркивать таким образом:

Скопировать код

CSS

.tooltip {

width: 500px;

height: 30px;

padding: 20px 40px;

*/\* любые другие общие правила \*/*

}

*/\* красим тултипы в зависимости от их функции \*/*

.tooltip.error { background-color: red; }

.tooltip.success { background-color: green; }

Мы находим такой подход неудобным. Лучше соблюдать единообразие визуального образа кода. Но если вы всё же договорились о подобных допущениях в команде, это ок.

Отдельный вопрос вызывает оформление групп селекторов в CSS. Вот два варианта:

Скопировать код

CSS

h1, h2, h3, h4, h5, h6, p {

margin: 0;

}

или

Скопировать код

CSS

h1,

h2,

h3,

h4,

h5,

h6,

p {

margin: 0;

}

Несмотря на визуальную сложность, кажется, что второй вариант проще воспринимать. Каждый селектор на отдельной строке легче отслеживается, а значит, такой код проще исправлять.

**Табы против пробелов**

Вложенность порождает новый вопрос — как эту самую вложенность создавать. Среди программистов есть две группировки. Одни голосуют за отделение кода пробелами, другие — за использование табуляции. И вроде бы нажать одну клавишу Tab всегда проще, чем несколько раз пробел, привычки у всех свои.

Про табы и пробелы очень много шуток. Этот спор стар, как само программирование.

Редактор кода, которым вы пользуетесь, скорее всего, умеет преобразовывать табуляцию в символы пробелов. Не переживайте, что это разные символы. Мы не хотим поддерживать эту войну, как и любую другую. Пользуйтесь, чем хотите.

Однако появляется второй вопрос: сколько пробелов должно быть в одной табуляции? Обычно используют два или четыре. И это тоже два лагеря. Сторонники двух пробелов говорят, что так код выглядит компактнее, сторонники четырёх — что с более глубокой отбивкой вложенность легче читать. Весь этот раздел читается, наверное, невероятно занудно, как и сами эти споры.

В нашем тренажёре используются два пробела вместо табов. Мы договорились при разработке курса, что будет так. На самом деле редактор кода можно настроить так, что всё будет работать автоматически нужным образом. В [рекомендациях по оформлению кода](https://code.s3.yandex.net/web-developer/verstka/%D0%9A%D0%BE%D0%B4%D1%81%D1%82%D0%B0%D0%B9%D0%BB__%D0%BE%D0%B1%D1%89%D0%B8%D0%B5_%D0%BF%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF%D1%8B.pdf) мы приводим пример, как это сделать плагином editorconfig. На работе рекомендуем тоже спросить, есть ли готовый файл настроек.

**Длинные строки HTML**

Когда мы пишем HTML, бывает, появляется много атрибутов. Примерно вот так:

Скопировать код

HTML

<a href="https://ya.ru" target="\_blank" class="link link\_active" title="ссылка на ya.ru">Я.ру</a>

В таких случаях нормально переносить атрибуты на новые строки и организовывать код. Так будет значительно легче его читать и исправлять.

Скопировать код

HTML

<a

href="https://ya.ru"

target="\_blank"

class="link link\_active"

title="ссылка на ya.ru"

>

Я.ру

</a>

Может выглядеть странно, особенно перенос символа > на отдельную строку. Тем не менее такой код удобно воспринимать, потому что видна вложенность и связь между открывающимся и закрывающимся тегами.

**Комментирование кода**

Мы сталкивались с мнением, что хороший код не нужно комментировать, он и так понятен. Мы очень уважаем всех, от кого это слышали, но звучит неправдоподобно. Хорошие комментарии всегда помогают читать и понимать код.

В вопросах комментирования есть несколько аспектов:

* что нужно комментировать,
* как писать комментарий,
* как оформлять комментарий.

Сначала дадим пару советов о том, что нужно комментировать.

Точно нужно комментировать непонятное. Звучит многозначно, поэтому попробуем объяснить.

В вёрстке вы будете сталкиваться с кодом, который вы понимаете в момент написания, но, скорее всего, не вспомните, что он значит, вернувшись к нему. Нужно стараться избегать таких ситуаций. Но это не всегда получается.

Вот пример. Он может быть вам не до конца понятен, потому что вы ещё не знаете этих свойств. Но комментарий раскрывает его суть.

Скопировать код

CSS

.block {

*/\* рассчитана пропорция резинового элемента по высоте.*

*В макете фиксированные размеры 939x524 \*/*

padding-bottom: calc(524px \* 100% / 939px);

}

Благодаря этому комментарию вы или другой разработчик поймёте, что в макете был элемент размером 939x524, который нужно было сделать резиновым. Поэтому пришлось составить пропорцию, где 939 взято за 100%.

Да, это всё еще непросто, потому что придётся разобраться, почему тут padding-bottom и зачем вообще делать этот элемент резиновым. Но как минимум числа перестанут быть в коде «магическими». «Магические числа» — это устойчивое выражение как раз для таких случаев.

Ещё мы рекомендуем комментировать большие куски кода, которые вы точно будете копировать целиком, например, чтобы вставить в другие страницы или зоны сайта. Это сэкономит время.

Предположим, вы создали блок с карточками «Ещё статьи по теме». В этом блоке — список, который вы предлагаете к прочтению пользователям после каждого лонгрида на сайте. Такой список может быть достаточно массивным и запутанным.

Вот код и комментарий:

Скопировать код

HTML

*<!-- Блок «Ещё статьи по теме» -->*

<section class="read-more">

<h2 class="read-more\_\_title">Ещё статьи по теме</h2>

<ul class="article-list">

<li class="article-list\_\_item">

<a href="#0" class="article-list\_\_link">

<article class="article-list\_\_card">

<img src="somepic.jpg" alt="some title" class="article-list\_\_image">

<h3 class="article-list\_\_item-title">Заголовок</h3>

<p class="article-list\_\_item-description">Короткое описание</p>

</article>

</a>

</li>

<li class="article-list\_\_item">

<a href="#0" class="article-list\_\_link">

<article class="article-list\_\_card">

<img src="somepic.jpg" alt="some title" class="article-list\_\_image">

<h3 class="article-list\_\_item-title">Заголовок</h3>

<p class="article-list\_\_item-description">Короткое описание</p>

</article>

</a>

</li>

<li class="article-list\_\_item">

<a href="#0" class="article-list\_\_link">

<article class="article-list\_\_card">

<img src="somepic.jpg" alt="some title" class="article-list\_\_image">

<h3 class="article-list\_\_item-title">Заголовок</h3>

<p class="article-list\_\_item-description">Короткое описание</p>

</article>

</a>

</li>

</ul>

</section>

*<!-- Конец Блока «Ещё статьи по теме» -->*

Даже в одиночестве этот блок очень большой. В полотне остального кода его будет вообще очень сложно идентифицировать. Но комментарий поможет. Вы точно сможете понять, где его начало и конец, а значит, потратите меньше времени на копирование в другую зону сайта. Останется только исправить ссылки и тексты.

Комментируйте неочевидные решения. Если вы чувствуете, что ваш код смахивает на трюк — подскажите это другим разработчикам. Помните, что время и умственные усилия разработчику полезнее и приятнее тратить на создание чего-то нового.

Вот пример:

Скопировать код

CSS

section {

*/\* грид-контейнер создан, чтобы управлять контекстом наложения в секции \*/*

display: grid;

}

В этом примере грид-контейнер выглядит избыточно без других свойств грид-контейнера. Но если вы сделали это умышленно ради возможности управлять z-index, расскажите об этом.

Комментируйте, если знаете, как развивать ваше решение. Есть особый вид комментариев с префиксом TODO:. Такой префикс распознают многие редакторы кода и позволяют осуществлять навигацию по ним.

Вот пример:

Скопировать код

HTML

*<!-- TODO: заменить логотип на svg, когда он придёт из дизайна -->*

<img src="logo.png" alt="Логотип сайта">

Вы могли заметить, что комментарии во всех примерах мы писали на отдельной строке, а после комментария оставляли пустую строку. Это тоже элемент кодстайла. Возьмите его на вооружение. Так проще читать.

**Организация кода**

**Разделение кода на части**

Чем длиннее CSS-файл, тем сложнее в нём найти что-то конкретное. На небольших проектах делить файлы обычно не приходится. Но на больших проектах, где стилей действительно много, встаёт вопрос, как организовать всё так, чтобы вносить правки можно было быстро.

В HTML вы можете подключить много файлов стилей. Они будут последовательно применяться к страницам. Организация файлов CSS — вопрос архитектуры проекта. Скорее всего, если вы приходите работать в существующий продукт, вам придётся только изучить уже существующий подход. Но если вы стартуете с нуля, понадобится ряд архитектурных решений.

Есть достаточно жёсткие методолгии разделения кода на части. Например, [БЭМ](https://ru.bem.info/methodology/filestructure/) с его классической [nested](https://ru.bem.info/methodology/filestructure/#nested) -структурой. У нас будет отдельный лонгрид на эту тему. На начальных этапах и в небольших проектах можно идти более простым путём.

Можно разделить файлы по функциональному признаку: стили для шрифтов в отдельном файле, стили анимаций — в другом, общие глобальные настройки — в третьем, а потом уже стили всех компонентов.

Скопировать код

styles

|--fonts.css

|--global.css

|--animations.css

|--style.css

Любая структура — условность, которую вы принимаете в команде. Сейчас хочется, чтобы вы просто поняли, что код можно делить, и начали думать о том, как его делить удобнее.

Вот ещё вариант:

Скопировать код

styles

|--fonts.css

|--global.css

|--animations.css

|--common-blocks.css

|--index.css

|--about.css

|--contacts.css

В этой структуре все общие компоненты стилизуются в файле common-blocks.css, а особенные стили для оформления страниц лежат в своих файлах.

Вариантов файловых структур может быть огромное количество. Главное, помните, что порядок подключения стилей важен. Те, что подключены в HTML тегом <link> ниже, переопределяют те, что были подключены выше.

**Последовательность правил в CSS**

Как бы вы ни организовывали файлы своего проекта, в них всегда будет много блоков правил. Эти правила стоит писать в понятном порядке. Старайтесь соблюдать такой же порядок, в котором селекторы представлены в HTML.

Например, в HTML у вас такая структура:

Скопировать код

HTML

<body>

<header class="header">

<nav class="menu"></nav>

</header>

<main class="main">

<section class="content">

<h2 class="content\_\_title"></h2>

</section>

</main>

</body>

В CSS стоит описывать правила для кода сверху вниз.

Скопировать код

CSS

body {

*/\* Стили body \*/*

}

.header {

*/\* Стили шапки \*/*

}

.menu {

*/\* Стили меню внутри шапки \*/*

}

.main {

*/\* Стили блока с основным содержимым \*/*

}

.content {

*/\* Стили секции \*/*

}

.content\_\_title {

*/\* Стили заголовка внутри секции \*/*

}

Если какому-то элементу требуется модификация или уточнение, описывайте этот стиль сразу под общим стилем этого элемента:

Скопировать код

CSS

button {

*/\* общие стили кнопки \*/*

}

button:hover {

*/\* цвет кнопки по наведению \*/*

}

button.button\_type\_alert {

*/\* стили кнопки с модификатором alert \*/*

}

button.button\_type\_alert:hover {

*/\* стили кнопки с модификатором alert по наведению \*/*

}

Даже этих небольших правил про последовательность свойств достаточно, чтобы код стало значительно проще читать и исправлять. Кодстайл — это про уважение к вашему времени и времени ваших коллег.

**Именование классов**

В вёрстке постоянно используют классы для стилизации. Мы рекомендуем пользоваться в основном ими, реже прибегать к стилизации селекторов тегов и никогда — идентификаторов. Раз классы так важны, значит, и называть их нужно с умом.

Сначала общие правила.

Избегайте заглавных букв. В HTML в принципе принято все называть строчными. В качестве разделителей используют дефисы или символы нижних подчёркиваний.

Так хорошо:

Скопировать код

HTML

<nav class="header-menu"></nav>

Так тоже нормально:

Скопировать код

HTML

<nav class="header\_\_menu"></nav>

Примечание: тут использованы два нижних подчёркивания. Это заморочки из именования в стиле [БЭМ](https://ru.bem.info/methodology/naming-convention/), про которые мы напишем отдельную статью.

А вот так не надо:

Скопировать код

HTML

<nav class="headerMenu"></nav>

Старайтесь придумывать названия, которые отражают смысл, а не визуальную составляющую.

Например, имя класса button-red хуже, чем button-accent. Цвет кнопки может поменяться в дизайне. Но её смысл от этого не поменяется.

Имя класса column-left хуже, чем column-start. Вдруг сайт придётся перевести на иврит, где пишут справа налево? Всё то же самое касается использования порядковых номеров в классах, указания позиции элемента. Пытайтесь описать именем класса предназначение элемента, а не его позицию или внешний вид.

Не бойтесь длинных имён классов, не жертвуйте понятностью в пользу скорости написания. Понятность важнее. Имя класса description лучше, чем desc, а button лучше, чем btn.

Не бойтесь присваивать несколько классов элементам. Это нормально. Например, у одного элемента могут быть сразу три класса:

1. с общими стилями;
2. с описанием состояния в конкретный момент;
3. с указанием позиции элемента или его темы.

Скопировать код

HTML

<a

href="#0"

class="link header-link link-theme-dark link-active"

>

Ссылка в шапке

</a>

Всё это в порядке вещей.

**Настройки IDE**

Текстовый редактор помогает с форматированием. Но это не значит, что можно не стараться писать код красиво. Во время обучения соблюдайте базовые правила оформления кода. Это прививает вкус и развивает внимательность.

Изучая новые технологии, вы будете всё больше автоматизировать работу со своим кодом. На начальном этапе вам будут помогать плагины, разработанные для редактора, а в будущем — специальные программы, которые запускаются внутри папки с кодом и помогают в форматировании и исправлениях.

**Подсказки в IDE**

Редактор создан, чтобы упрощать работу с кодом. Он больше рассказывает не про кодстайл, а про предупреждение ошибок. Чем более серьёзный код вы пишете, тем больше ошибок способен отловить редактор кода и подсказать, где вы опечатались или неправы.

Вот пара примеров из HTML и CSS.

*Стоит вам пропустить скобку, IDE сразу покрасит в красный следующий тег*

*Пропущенный символ точки с запятой изменит цвет следующего свойства и добавит красных подчеркиваний в ваш код*

Текстовый редактор не может видеть все ошибки. Но если что-то выглядит не так, как обычно, присмотритесь. Это может помочь быстрее обнаружить проблему.

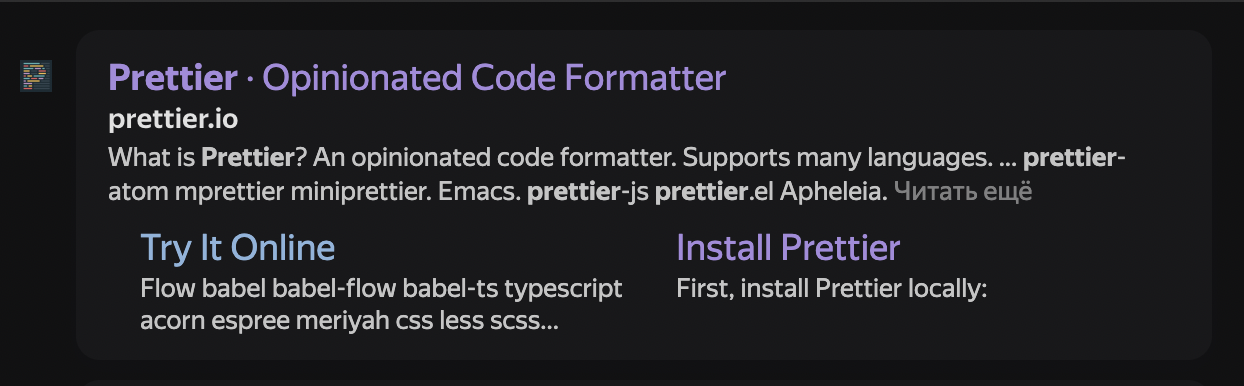
**Prettier**

В основной части курса мы рекомендуем автоматическое форматирование плагином [Prettier — Code Formatter](https://marketplace.visualstudio.com/items?itemName=esbenp.prettier-vscode) для VSCode. Он хорош, прост в использовании и популярен. Вы устанавливаете его и выбираете “Format document with” в выпадающем меню ваших HTML-файлов. Ваш код становится аккуратнее.

*Prettier в HTML*

*Работа Prettier с CSS*

Про этот плагин нужно знать важное. Его настройки — это мнение его разработчиков о том, как должен выглядеть код. Они позиционируют свою разработку как “Opinionated code formatter”.



Opinionated — обусловленный личными предпочтениями

В команде разработки курса по вёрстке мнения о Prettier разные, как и в сообществе. Некоторые решения, которые диктует это плагин, спорные. Например, он закрывает одиночные теги, хотя их можно не закрывать.

Делает так:

Скопировать код

HTML

<input />

<img src="" alt="" />

<meta />

Ещё он привязывается к максимально допустимой длине строки, которую установил сам. Из-за этого переносы в коде порой кажутся необычными.

Тем не менее, простота Prettier сделала его настолько популярным, что в современной разработке им пользуются в значительной части проектов. В нашем курсе мы не везде используем Prettier, а вам рекомендуем. Потому что он достаточно хорош, хотя и не идеален.

В рабочих проектах вы можете столкнуться с тем, что разработчики предпочтут инструменты форматирования с более тонкой настройкой, но и более высоким порогом входа в технологию настроек.

**Порядок свойств CSS**

Внутри одного CSS-правила можно по-разному группировать свойства. На этот счёт нет общей конвенции, но есть много мнений. Есть, например, такое, что свойства стоит писать в алфавитном порядке. Но кажется, на практике это очень неудобно. Есть ещё разные [рассуждения от опытных разработчиков](https://github.com/necolas/idiomatic-css#declaration-order). Но всё же о порядке свойств CSS в проекте удобнее договариваться при старте работы, чем стандартизировать.

Авторы этого курса обсудили и сошлись примерно на такой группировке свойств:

* сначала те, что описывают лейаут и видимость элемента;
* потом — свойства блочной модели и поведения блока;
* затем — цвет, шрифт, текст, фон, анимации и всё остальное.

У нас нет единого ответа на вопрос, как делать правильно. Вам стоит обсудить это с командой, в которой вы будете работать.

Мы подготовили для вас настройку в VSCode, которую используем для написания кода курса. Она определяет порядок свойств, о котором мы договорились с авторами. Вот как её настроить:

1. Установите плагин для редактора кода <https://marketplace.visualstudio.com/items?itemName=mrmlnc.vscode-postcss-sorting>
2. Откройте файл по ссылке <https://gist.github.com/verstka-passion/76c76cd844026512f71ccefd725d7c00> и скопируйте этот код.
3. Зайдите в настройки VSCode и найдите там ссылку на settings.json PostCSS Sorting. Это легко делать через поиск по слову “Sorting”.
4. Вставьте в этот файл код, который мы выложили на GitHub, и сохраните изменения.
5. Теперь в любом файле CSS вы можете нажать клавишу F1 и выбрать в меню PostCSS Sorting: Run.
6. Порядок свойств в правилах поменяется:

Вы можете менять порядок и дополнять свойства, обсуждая лучшее решение с командой. Главное, что у вас теперь есть инструмент для диалога о том, как сделать CSS единообразным в проекте.

**О важности обсуждений в команде**

Мы написали эту статью, чтобы обозначить вопросы кодстайла. В каждой команде этот вопрос проявляется по-своему. Поэтому основной инструмент настройки вашего кодстайла — диалог, а не технические инструменты.

В работе вам придётся обсуждать, ставить ли точки в десятичных обозначениях CSS, использовать старый или новый синтаксис функций rgb(), можете ли вы уже писать свойства вроде aspect-ratio или браузеры ваших пользователей их ещё не поддерживают. Это придётся обсудить голосом или текстом, и лишь потом искать инструменты, которые будут помогать предупреждать или исправлять ошибки.

**Взгляд вперёд**

Сейчас вы только начинаете свой путь в веб-разработке, поэтому в этой статье только общие концепции и самые простые и полезные программы. И Prettier, и PostCSS Sorting — это лишь частные инструменты, которые воплощены в плагинах для IDE.

В будущем мы разберём настоящую автоматизацию преобразований кода. Например, посмотрим, как перед отправкой кода на GitHub автоматически проверить его на ошибки синтаксиса или правил, валидировать через W3C Validator, форматировать и сортировать свойства. Всё это часть большой инфраструктуры, которая в современном мире сопутствует написанию кода приложений.

Мы считаем, что на первых этапах лучше сосредоточиться на основах, а заниматься настройками лучше вторым шагом. Дозировано вы получите всю эту информацию и знания. Настройки окружения бывают настолько сложными и сами по себе интересными, что легко забыть о самом важном — создании чего-то нового и необычного через код.

Сейчас пора заняться кодингом, а не инфраструктурой, и, конечно, приучить себя к виду хорошего кода. Мы надеемся, что эта статья обозначила направления, о чём думать и за чем следить при написании очередной строчки кода веб-страницы.