Обучающее пособие по специализации: **«Front-end»**

Введение в веб-разработку

Что такое сайт?

Это несколько связанных между собой страниц с текстом, картинками и всем остальным, что бывает в интернете. Ваш браузер скачивает web-сайт в виде страницы на языке HTML (именно на нем пишутся, а точнее — размечаются веб-страницы) и считал специальные разметки — наборы букв и символов, преобразовав их в нечто более симпатичное, чем просто текст.

Бывают статические и динамические сайты. Статические — это готовые HTML-страницы: например, сайты-визитки с контактами, прайс-листом и другой информацией, которую не нужно часто менять. Чтобы упростить процесс обновления информации, были придуманы динамические сайты. В их основе — веб-сервер, специальная программа, которая принимает запрос пользователя и каждый раз генерирует для него новый файл с той страницей, запрос к которой он отправил.

**Фронтенд** — все, что браузер может читать, выводить на экран и / или запускать. То есть это HTML, CSS и JavaScript.

HTML (HyperText Markup Language) говорит браузеру, каково содержание страницы, например, «заголовок», «параграф», «список», «элемент списка».

CSS (Cascading Style Sheets) говорит браузеру, как отображать элементы, например, «после первого параграфа отступ в 20 пикселей» или «весь текст в элементе body должен быть темно-серым и написан шрифтом Verdana».

JavaScript говорит браузеру, как реагировать на некоторые взаимодействия, используя легкий язык программирования. Большинство сайтов на самом деле не используют много JavaScript, но если вы нажмете на что-то и содержимое страницы поменяется без белого мигания экрана, значит, где-то использовался JavaScript.

**Бэкенд** — все, что работает на сервере, то есть «не в браузере» или «на компьютере, подсоединенном к сети (обычно к Интернету), который отвечает на сообщения от других компьютеров».

Для бэкенда вы можете использовать любые инструменты, доступные на вашем сервере (который, по сути, является просто компьютером, настроенным для ответов на сообщения). Это означает, что вы можете использовать любой универсальный язык программирования: Ruby, PHP, Python, Java, JavaScript / Node, bash. Это также означает, что вы можете использовать системы управления базами данных, такие как MySQL, PostgreSQL, MongoDB, Cassandra, Redis, Memcached.

**Верстка** – это процесс написания html и css — кода для веб-страницы. Но кода можно написать очень много, какой именно это код? На самом деле, все очень просто, этот код отвечает за то, чтобы все элементы страницы размещались там, где нужно.

Процесс создания макетов, в которых еще нет информации, но общая структура веб-страницы уже определена и называется **версткой сайта**.

Веб-страница состоит из трёх частей:

1. Информация о версии HTML,
2. Шапка веб-страницы, в которой содержится техническая информация (<head>),
3. Тело веб-страницы (<body>).

Информация об используемой версии HTML представляется в виде первой строки в исходном коде веб-страницы.

Шапка веб-страницы содержит различную техническую информацию о веб-странице (например, название, ключевые слова, метаданные) и представляет собой набор элементов, которые не входят в графическое представление веб-страницы.

Тело веб-страницы содержит графическое и информационное представление веб-страницы

Скачиваем и устанавливаем Sublime Text 3

Идем на [сайт Sublime Text](http://www.sublimetext.com/3) в раздел Download, и скачиваем последнюю версию ST3, не забудьте соблюсти разрядность 64 бит, или обычную (x86) для вашей версии операционной системы. Скачали, устанавливаем. После установки запускаем — проверяем что он работает.

### Установим горячие клавиши для реиндентации

Открываем Preferences → Key Bindings — Default  
И вверху файла добавляем строку:

{ "keys": ["alt+shift+f"], "command": "reindent" },

Здесь указано что горячие клавиши будут alt+shift+f, и по нажатию будет происходить команда реиндент.

**Возможные проблемы  с настройками в Sublime Text**

Если возникает проблема с сохранением или редактированием файла, то необходимо создать файл — который указан в ошибке. В моем случае это файл: C:UsersAdminAppDataRoamingSublime Text 3PackagesDefaultDefault (Windows).sublime-keymap Я создал его и сохранил по указанном пути. Теперь Preferences → Key Bindings — Default можно редактировать.

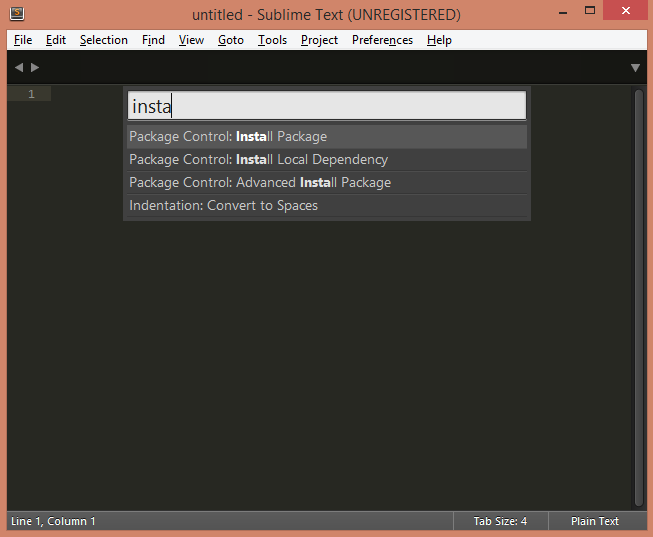
Есть аналогичная проблема с командой Preferences → Settings — Default, при открытии настроек их невозможно редактировать. Необходимо открыть, Нажать Ctrl+S, затем закрыть вкладку с настройками и открыть заново. После этого в C:UsersAdminAppDataRoamingSublime Text 3PackagesDefault должен появится файл с настройками — Preferences.sublime-settings — и настройки можно будет редактировать и сохранять.

### Устанавливаем Package Control

Идем на [сайт Package Control](https://packagecontrol.io/installation) для ST3, и устанавливаем его. Package Control (PC) необходим для того чтобы расширять функциональность нашего редактора различными плагинами.

На сайте PC копируем код для ST3, в ST3 запускаем консоль (Ctrl + ~) и вставляем код с сайта, жмем Enter и ждем успешной установки. После обязательно перезапускам ST3.

Проверим работу PC. Жмем Ctrl+Shift+P и  набираем Package Control — Install Package. Уже набрав первые буквы — нужная строка будет автоматически подсвечена.

[](http://rightblog.ru/wp-content/uploads/2015/09/pc-install-package1.png)

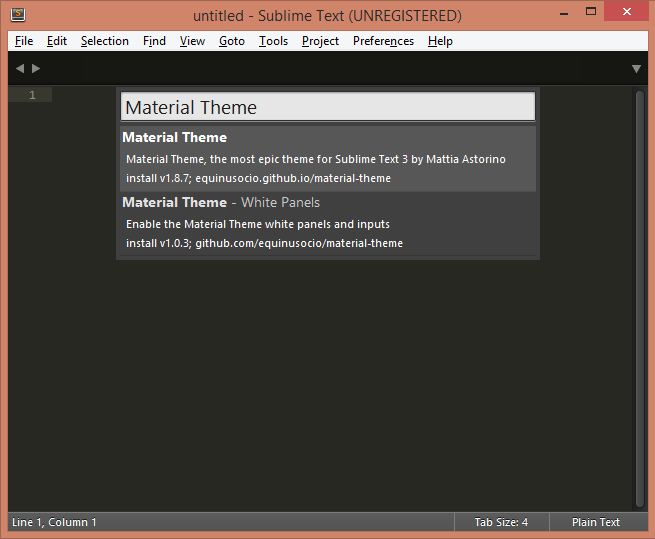
Sublime Text 3 — Package Control — установкаплагинов

## Часть 2. Установка плагинов и расширений для Sublime Text 3

## Установка темы Material Design

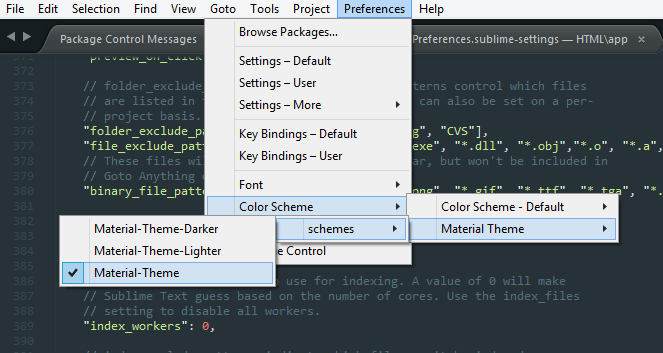
Далее я сменю стандартную тему оформления и подсветки кода на Material Design. Она тоже относится к темным темам, но имеет более яркую подсветку кода, и с ней более комфртно и приятно работать при длительном написании кода.

Идем на [сайт темы Material Design](http://equinusocio.github.io/material-theme/) и жмем кнопкe install. Происходит переход на сайт PC для ST3, понимаем что тема доступна как плагин. В ST3 запускаем Ctrl+Shift+P, выбираем install package, после ищем нужную нам тему, я набираю Material Theme и нахожу ее. Жму Enter и устанавливаю.

[](http://rightblog.ru/wp-content/uploads/2015/09/pc-material1.png)

Sublime Text 3 — Package Control — установкатемы Material Design

После этого иду в меню Preferences, и выбираю установленную тему.

[](http://rightblog.ru/wp-content/uploads/2015/09/pc-material-chose1.png)

Sublime Text 3 — Package Control — установкатемы Material Design

Также после этого, рекомендуется зайти в  настройки ST3, и дописать следующие конфиги:

"theme": "Material-Theme.sublime-theme",

"color\_scheme": "Packages/Material Theme/schemes/Material-Theme.tmTheme",

"overlay\_scroll\_bars": "enabled",

"line\_padding\_top": 3,

"line\_padding\_bottom": 3,

// On retina Mac

"font\_options": [ "gray\_antialias" ],

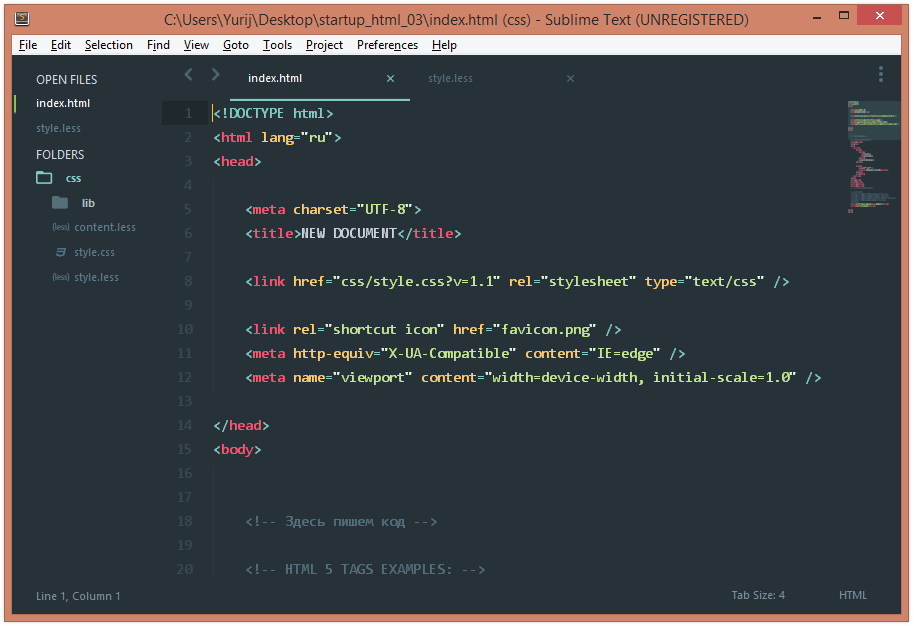
"always\_show\_minimap\_viewport": true,

"bold\_folder\_labels": true,

// Highlight active indent

"indent\_guide\_options": [ "draw\_normal", "draw\_active" ]

Это позволит отображать тему так как это было задумано ее создателями.  
В итоге у меня получился вот такой внешний вид ST3.

[](http://rightblog.ru/wp-content/uploads/2015/09/st3_material_preview1.png)

Sublime Text 3 — тема Material Design

### Установкаплагинов

Я установил следующие основные плагины:

* Emmet (ускорение написания html и css)
* Hayaku (ускорение написания css)
* LESS (syntax highlighting — подсветкакода less)
* Prefixr (префиксы для CSS)
* Placeholders (Lorem Ipsum наполнители для html файлов)

Дополнительные:

* SFTP — плагин для работы с FTP
* SideBarEnhancement расширяет возможность сайт бара
* SyncedSideBar синхронизирует SideBar с открытыми файлами по клику на них
* AdvancedNewFile создаём новые файлы и папки при помощи горячих клавиш
* BracketHighlighter подсвечивает открытие/закрытие любого фрагмента в коде
* jQuery набор сниппетов для jquery
* AutoFileName автозаполнение путей к подключаемым файлам
* CSSсomb делает код красивым
* Gist сохранение отдельных участков кода на github прямо во время редактировани

### лементы

Элементы указывают, как определять структуру и содержимое объектов на странице. Некоторые из часто используемых элементов включают в себя несколько уровней заголовков (определены как элементы с <h1> до <h6>) и абзацев (определены как <p>); в список можно включить элементы <a>, <div>, <span>, <strong> и <em> и многие другие.

Элементы идентифицируются с помощью угловых скобок <>, окружающих имя элемента. Таким образом, элемент будет выглядеть так:

<a>

### Теги

Добавление угловые скобок < и > вокруг элемента создаёт то, что известно как тег. Теги наиболее часто встречаются в парах открывающих и закрывающих тегов.

Открывающий тег отмечает начало элемента. Он состоит из символа <, затем идёт имя элемента и завершается символом >; например, <div>.

Закрывающий тег отмечает конец элемента. Он состоит из символа < с последующей косой чертой и именем элемента и завершается символом >; например, </div>.

Содержимое, которое находится между открывающим и закрывающим тегами, является содержимым этого элемента. Ссылка, к примеру, будет иметь открывающий тег <a> и закрывающий тег </a>. Что находится между этими двумя тегами будет содержимым ссылки.

Так, теги ссылок будут выглядеть примерно так:

<a>...</a>

### Атрибуты

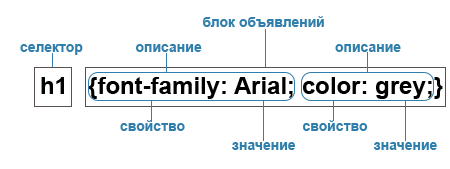
Атрибуты являются свойствами, применяемыми для предоставления дополнительной информации об элементе. Наиболее распространённые атрибуты включают в себя атрибут id, который идентифицирует элемент; атрибут class, который классифицирует элемент; атрибут src, который определяет источник встраиваемого содержимого; и атрибут href, который указывает ссылку на связанный ресурс.

Атрибуты определяются в открывающем теге после имени элемента. В общем, атрибуты включают в себя имя и значение. Формат для этих атрибутов состоит из имени атрибута со знаком равенства за ним, а затем в кавычках идёт значение атрибута. Например, элемент <a> с атрибутом href будет выглядеть следующим образом:

<ahref="http://shayhowe.com">Shay Howe</a>

Каскадные таблицы стилей описывают правила форматирования элементов с помощью свойств и допустимых значений этих свойств. Для каждого элемента можно использовать ограниченный набор свойств, остальные свойства не будут оказывать на него никакого влияния.

Объявление стиля состоит из двух частей: селектора и объявления. В HTML имена элементов нечувствительны к регистру, поэтому «h1» работает так же, как и «H1». Объявление состоит из двух частей: имя свойства (например, color) и значение свойства (grey). Селектор сообщает браузеру, какой именно элемент форматировать, а в блоке объявления (код в фигурных скобках) перечисляются форматирующие команды — свойства и их значения.



Язык HTML

HTML (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *HyperTextMarkupLanguage* — «язык [гипертекстовой](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%B5%D1%80%D1%82%D0%B5%D0%BA%D1%81%D1%82) разметки») — стандартизированный язык разметки документов во [Всемирной паутине](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D0%B5%D0%BC%D0%B8%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D1%83%D1%82%D0%B8%D0%BD%D0%B0). Большинство [веб-страниц](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0) содержат описание разметки на языке HTML (или [XHTML](https://ru.wikipedia.org/wiki/XHTML)). Язык HTML интерпретируется [браузерами](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80); полученный в результате интерпретации форматированный текст отображается на экране монитора компьютера или мобильного устройства.

Если открыть любую веб-страницу, то она будет содержать в себе типичные элементы, которые не меняются от вида и направленности сайта. В примере 4.1 показан код простого документа, содержащего основные теги.

<!DOCTYPEhtml>

<html>

<head>

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

<title>Пример веб-страницы</title>

</head>

<body>

<h1>Заголовок</h1>

<!-- Комментарий -->

<p>Первый абзац.</p>

<p>Второй абзац.</p>

</body>

</html>

Тег <html> определяет начало HTML-файла, внутри него хранится заголовок (<head>) и тело документа (<body>).

Заголовок документа, как еще называют блок <head>, может содержать текст и теги, но содержимое этого раздела не показывается напрямую на странице, за исключением контейнера <title>.

Тег <meta> является универсальным и добавляет целый класс возможностей, в частности, с помощью метатегов, как обобщенно называют этот тег, можно изменять кодировку страницы, добавлять ключевые слова, описание документа и многое другое. Чтобы браузер понимал, что имеет дело с кодировкой UTF-8 (Unicode transformation format, формат преобразования Юникод) и добавляется данная строка.

Тег <title> определяет заголовок веб-страницы, это один из важных элементов предназначенный для решения множества задач. В операционной системе Windows текст заголовка отображается в левом верхнем углу окна браузера.

Обязательно следует добавлять закрывающий тег </head>, чтобы показать, что блок заголовка документа завершен.

Тело документа <body> предназначено для размещения тегов и содержательной части веб-страницы.

HTML предлагает шесть текстовых заголовков разного уровня, которые показывают относительную важность секции, расположенной после заголовка. Так, тег <h1> представляет собой наиболее важный заголовок первого уровня, а тег <h6> служит для обозначения заголовка шестого уровня и является наименее значительным. По умолчанию, заголовок первого уровня отображается самым крупным шрифтом жирного начертания, заголовки последующего уровня по размеру меньше. Теги <h1>...<h6> относятся к блочным элементам, они всегда начинаются с новой строки, а после них другие элементы отображаются на следующей строке. Кроме того, перед заголовком и после него добавляется пустое пространство.

Некоторый текст можно спрятать от показа в браузере, сделав его комментарием. Хотя такой текст пользователь не увидит, он все равно будет передаваться в документе, так что, посмотрев исходный код, можно обнаружить скрытые заметки.

Комментарии нужны для внесения в код своих записей, не влияющих на вид страницы. Начинаются они тегом <!-- и заканчиваются тегом -->. Все, что находится между этими тегами, отображаться на веб-странице не будет.

Тег <p> определяет абзац (параграф) текста. Если закрывающего тега нет, считается, что конец абзаца совпадает с началом следующего блочного элемента.

Тег <p> является блочным элементом, поэтому текст всегда начинается с новой строки, абзацы идущие друг за другом разделяются между собой отбивкой (так называется пустое пространство между ними). Это хорошо видно на рис.

Следует добавить закрывающий тег </body>, чтобы показать, что тело документа завершено.

Последним элементом в коде всегда идет закрывающий тег </html>.

Тег **<input>** является одним из разносторонних элементов формы и позволяет создавать разные элементы интерфейса и обеспечить взаимодействие с пользователем. Главным образом **<input>**предназначен для создания текстовых полей, различных кнопок, переключателей и флажков. Хотя элемент **<input>** не требуется помещать внутрь контейнера **<form>**, определяющего форму, но если введенные пользователем данные должны быть отправлены на сервер, где их обрабатывает серверная программа, то указывать **<form>** обязательно. То же самое обстоит и в случае обработки данных с помощью клиентских приложений, например, скриптов на языке JavaScript.

Основной атрибут тега **<input>**, определяющий вид элемента — type. Он позволяет задавать следующие элементы формы: текстовое поле (text), поле с паролем (password), переключатель (radio), флажок (checkbox), скрытое поле (hidden), кнопка (button), кнопка для отправки формы (submit), кнопка для очистки формы (reset), поле для отправки файла (file) и кнопка с изображением (image). Для каждого элемента существует свой список атрибутов, которые определяют его вид и характеристики. Кроме того, в HTML5 добавлено еще более десятка новых элементов.

Тег **<button>** создает на веб-странице кнопки и по своему действию напоминает результат, получаемый с помощью тега **<input>** (с атрибутом type="button | reset | submit"). В отличие от этого тега, **<button>** предлагает расширенные возможности по созданию кнопок. Например, на подобной кнопке можно размещать любые элементы HTML, в том числе изображения. Используя стили можно определить вид кнопки путем изменения шрифта, цвета фона, размеров и других параметров.

Теоретически, тег **<button>** должен располагаться внутри формы, устанавливаемой элементом **<form>**. Тем не менее, браузеры не выводят сообщение об ошибке и корректно работают с тегом **<button>**, если он встречается самостоятельно. Однако, если необходимо результат нажатия на кнопку отправить на сервер, помещать **<button>** в контейнер **<form>** обязательно.

Валидация — это проверка значений, указанных пользователем, и отображение найденных ошибок.

Самый простой путь валидации — это определить тип input поля и расставить атрибуты required которые отвечают за обязательность заполнения.

<formaction="#"class="form">

<inputtype="text"name="name"placeholder="Имя"required />

<inputtype="email"name="email"placeholder="E-Mail" />

<inputtype="tel"name="phone"placeholder="Телефон" />

<inputtype="url"name="url"placeholder="Вашсайт" />

<inputtype="number"name="growth"placeholder="Вашрост" />

<buttontype="submit">Отправить</button>

</form>

Применение этих двух атрибутов позволит гораздо эффективнее валидировать вводимую информацию нативными методами. Ну и конечно же поддержка этих свойств браузерами наиболее широка.  
  
Отдельно хотелось бы сказать про тип поля *tel*. Ожидается что браузер будет валидировать телефонные номера, но нет, поле с типом *tel* используется сейчас только для автозаполнения. Дело в том, что валидация телефонных номеров очень неоднозначная задача из-за слишком большого количества различных форматов телефонных номеров в разных странах, которые просто никак не получится унифицировать и записать в одно правило.  
  
Однако, нам на помощь приходит атрибут *pattern*. Этот атрибут принимает в себя значение регулярного выражения. В нашем случае рассмотрим вариант паттерна для ввода мобильного телефона в РФ: *+7 (123) 456-78-91*. Для этого добавим простое регулярное выражение в наше поле с телефоном, а также ограничим минимальное и максимальное количество символов:

<inputtype="tel"name="phone"placeholder="Телефон"pattern="[\+]\d{1}\s[\(]\d{3}[\)]\s\d{3}[\-]\d{2}[\-]\d{2}"minlength="18"maxlength="18" />

Элемент **<table>** служит контейнером для элементов, определяющих содержимое таблицы. Любая таблица состоит из строк и ячеек, которые задаются с помощью тегов **<tr>** и **<td>**. Внутри **<table>**допустимо использовать следующие элементы: **<caption>**, **<col>**, **<colgroup>**, **<tbody>**, **<td>**, **<tfoot>**, **<th>**, **<thead>** и **<tr>**.

Таблицы с невидимой границей долгое время использовались для верстки веб-страниц, позволяя разделять документ на модульные блоки. Подобный способ применения таблиц нашел воплощение на многих сайтах, пока ему на смену не пришел более современный способ верстки с помощью слоев.

аблица состоит из строк и столбцов ячеек, которые могут содержать текст и рисунки. Обычно таблицы используются для упорядочения и представления данных, однако возможности таблиц этим не ограничиваются. C помощью таблиц удобно верстать макеты страниц, расположив нужным образом фрагменты текста и изображений.

Для добавления таблицы на веб-страницу используется тег **<table>**. Этот элемент служит контейнером для элементов, определяющих содержимое таблицы. Любая таблица состоит из строк и ячеек, которые задаются соответственно с помощью тегов **<tr>** и **<td>**. Таблица должна содержать хотя бы одну ячейку (пример 12.1). Допускается вместо тега **<td>** использовать тег **<th>**. Текст в ячейке, оформленной с помощью тега **<th>**, отображается браузером шрифтом жирного начертания и выравнивается по центру ячейки. В остальном, разницы между ячейками, созданными через теги **<td>** и **<th>** нет.

**Пример 12.1. Создание таблицы**

<!DOCTYPEHTMLPUBLIC "-//W3C//DTDHTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

**<html>**

**<head>**

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

**<title>**Тег TABLE**</title>**

**</head>**

**<body>**

**<table** border="1" width="100%" cellpadding="5"**>**

**<tr>**

**<th>**Ячейка 1**</th>**

**<th>**Ячейка 2**</th>**

**</tr>**

**<tr>**

**<td>**Ячейка 3**</td>**

**<td>**Ячейка 4**</td>**

**</tr>**

**</table>**

**</body>**

**</html>**

**Семантические элементы HTML5** доступно описывают свой смысл или назначение как для браузеров, так и для веб-разработчиков.  
До появления стандарта HTML5 вся разметка страниц осуществлялась преимущественно с помощью элементов <div>, которым присваивали классы class или идентификаторы id для наглядности разметки (например, <div id="header">). С их помощью в HTML-документе размещали верхние и нижние колонтитулы, боковые панели, навигацию и многое другое.

Стандарт HTML5 предоставил новые элементы для структурирования, группировки контента и разметки текстового содержимого. Новые семантические элементы позволили улучшить структуру веб-страницы, добавив смысловое значение заключенному в них содержимому (было <div id="header">, стало <header>). Для отображения внешнего вида элементов не задано никаких правил, поэтому элементы можно стилизовать по своему усмотрению. Для всех элементов доступны [‎глобальныеатрибуты](https://html5book.ru/html-attributes/).

Согласно спецификации HTML5 каждый элемент принадлежит к определенной (ноль или более) категории. Каждая из них группирует элементы со схожими характеристиками. Выделяют следующие общие категории:

* Мета содержимое
* Потоковое содержимое
* Секционное содержимое
* Заголовочное содержимое
* Текстовое содержимое
* Встроенное содержимое
* Интерактивное содержимое

Сверстанная страница не должна содержать ошибок в HTML (проверяется онлайн-валидатором - validator.w3.org)

Расположение блоков в сверстанной странице должно быть "один к одному" по сравнению с исходным макетом, за исключением случаев описанных ниже.

Поддержка веб-браузов:

* Google Chrome – последняя версия;
* Opera – последняя версия;
* Mozilla Firefox – последняя версия;
* Safari – последняя версия;
* Internet Explorer – последняя версия;
* Microsoft Edge – последняя версия.

Поддержка мобильных платформ:

* Apple iPhone 4 и выше, в последних версиях бразуеров Safari, Google Chrome;
* Apple iPad 2 и выше, в последних версиях бразуеров Safari, Google Chrome;
* мобильные устройства на базе ОС Android версии 4.2 и выше, в последних версиях бразуеров Android, Google Chrome;
* мобильные устройства на базе ОС Windows Phone версии 8 и выше, в последних версиях бразуеров Internet Explorer, Google Chrome.

Поддержка браузеров и мобильных платформ более ранних версий, а также поддержка иных браузеров и мобильных платформ производится в отдельном порядке.

Сверстанная страница должна быть оптимизирована для максимальной скорости загрузки

**Общие требования к HTML разметке**

* Форматирование HTML кода должно быть иерархическим, отступ должен составлять 4 пробела.
* Верстка HTML страниц должна быть реализована без использования JavaScript (там где это возможно).
* Необходимо следовать семантики HTML5. Использовать существующие тэги HTML5 (header, footer, aside, nav, section, article и т.д). 2. Каждый тег должен использоваться по назначению (div для блоков, ul для списков и т.д.).
* Необходимо правильно организовать структуру заголовков H1-H6. На странице не должно быть двух заголовков H1.
* Кодировка для HTML страниц должна быть определена как "UTF-8".
* Значения всех атрибутов HTML тегов должны быть заклюены в двойные кавычки (там где это возможно).
* Классы которые используются в JavaScript должны начинаться с префикса "js-" и не должны использоваться для оформления стилей.
* Для адаптивности необходимо использовать подход от меньшего к большему (порядок расположения стилей). Запрещено в одном проекте использовать сразу два подхода (от меньшего к большему и от большего к меньшему).
* Все внешние ресурсы необходимо хранить в папках с названием блока или компонента (например, /images/news/bg.jpg).
* Для всех картинок необходимо определять аттрибут "alt".
* Необходимо определять цвет фона для тега "body". Если цвет не указан в макете, необходимо использовать белый.
* Запрещается использовать размеры больше 100% (за исключением случаев, когда на этом построена логика работы, например галереи и т.д.).
* Верстка контента, редактируемого в WYSIWYG-редакторе, не должна содержать классов (там где это возможно).
* На HTML странице должны присутствовать "favicon" (с вариациями 32x32, 48x48 и 64x64) и "apple-touch-icon". "favicon" и "apple-touch-icon" должны быть предоставлены дизайнером. "favicon" и "apple-touch-icon" должны быть в формате ".png".
* Для ссылок на e-mail необходимо задавать атрибут "mailto", для ссылок на телефон - атрибут "tel".
* Запрещается использовать атрибут "id" с целью оформления стилей.

**Основы CSS**

**CSS (Cascading Style Sheets)** — язык таблиц стилей, который позволяет прикреплять стиль (например, шрифты и цвет) к структурированным документам (например, документам HTML и приложениям XML). Обычно CSS-стили используются для создания и изменения стиля элементов веб-страниц и пользовательских интерфейсов, написанных на языках HTML и XHTML, но также могут быть применены к любому виду XML-документа, в том числе XML, SVG и XUL. Отделяя стиль представления документов от содержимого документов, CSS упрощает создание веб-страниц и обслуживание сайтов.

Связанные стили

При использовании связанных стилей описание селекторов и их значений располагается в отдельном файле, как правило, с расширением css, а для связывания документа с этим файлом применяется тег **<link>**. Данный тег помещается в контейнер **<head>**

<!DOCTYPEHTML>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Стили**</title>**

**<link** rel="stylesheet" href="http://htmlbook.ru/mysite.css"**>**

**<link** rel="stylesheet" href="http://www.htmlbook.ru/main.css"**>**

**</head>**

**<body>**

**<h1>**Заголовок**</h1>**

**<p>**Текст**</p>**

**</body>**

**</html>**

Значение атрибута тега **<link>** — rel остаётся неизменным независимо от кода, как приведено в данном примере. Значение href задаёт путь к CSS-файлу, он может быть задан как относительно, так и абсолютно. Заметьте, что таким образом можно подключать таблицу стилей, которая находится на другом сайте.

Содержимое файла mysite.css подключаемого посредством тега **<link>** .

**H1**{

color:#000080;

font-size:200%;

font-family: Arial, Verdana, sans-serif;

text-align: center;

}

**P**{

padding-left:20px;

}

Как видно из данного примера, файл со стилем не хранит никаких данных, кроме синтаксиса CSS. В свою очередь и HTML-документ содержит только ссылку на файл со стилем, т. е. таким способом в полной мере реализуется принцип разделения кода и оформления сайта. Поэтому использование связанных стилей является наиболее универсальным и удобным методом добавления стиля на сайт. Ведь стили хранятся в одном файле, а в HTML-документах указывается только ссылка на него.

Глобальные стили

При использовании глобальных стилей свойства CSS описываются в самом документе и располагаются в заголовке веб-страницы. По своей гибкости и возможностям этот способ добавления стиля уступает предыдущему, но также позволяет хранить стили в одном месте, в данном случае прямо на той же странице с помощью контейнера **<style>**

<!DOCTYPE HTML>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Глобальныестили**</title>**

**<style>**

**H1**{

font-size:120%;

font-family: Verdana, Arial, Helvetica, sans-serif;

color:#333366;

}

**</style>**

**</head>**

**<body>**

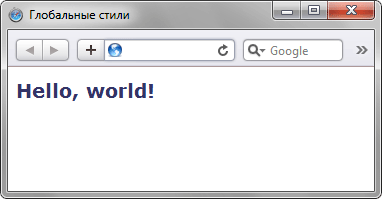
**<h1>**Hello, world!**</h1>**

**</body>**

**</html>**

Посмотреть пример  
Посмотреть пример

В данном примере задан стиль тега **<h1>**, который затем можно повсеместно использовать на данной веб-странице



Внутренние стили

Внутренний или встроенный стиль является по существу расширением для одиночного тега используемого на текущей веб-странице. Для определения стиля используется атрибут style, а его значением выступает набор стилевых правил.

<!DOCTYPE HTML>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Внутренниестили**</title>**

**</head>**

**<body>**

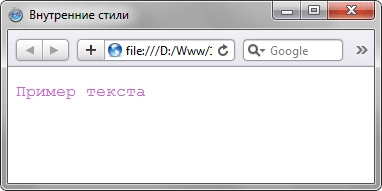
**<p** style="font-size: 120%; font-family: monospace; color: #cd66cc"**>**Примертекста**</p>**

**</body>**

**</html>**

Посмотреть пример  
Посмотреть пример

В данном примере стиль тега **<p>** задаётся с помощью атрибута style, в котором через точку с запятой перечисляются стилевые свойства (рис. 3.2).



Внутренние стили рекомендуется применять на сайте ограниченно или вообще отказаться от их использования. Дело в том, что добавление таких стилей увеличивает общий объём файлов, что ведет к повышению времени их загрузки в браузере, и усложняет редактирование документов для разработчиков.

Все описанные методы использования CSS могут применяться как самостоятельно, так и в сочетании друг с другом. В этом случае необходимо помнить об их иерархии. Первым имеет приоритет внутренний стиль, затем глобальный стиль и в последнюю очередь связанный стиль.

<!DOCTYPEHTML>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Подключениестиля**</title>**

**<style>**

**H1**{

font-size:120%;

font-family: Arial, Helvetica, sans-serif;

color: green;

}

**</style>**

**</head>**

**<body>**

**<h1** style="font-size: 36px; font-family: Times, serif; color: red"**>**Заголовок 1**</h1>**

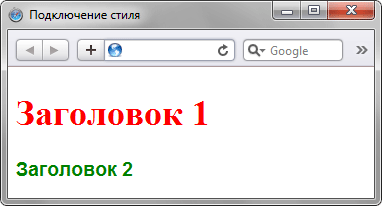
**<h1>**Заголовок 2**</h1>**

**</body>**

**</html>**

Посмотреть пример  
Посмотреть пример

В данном примере первый заголовок задаётся красным цветом размером 36 пикселов с помощью внутреннего стиля, а следующий — зелёным цветом через таблицу глобальных стилей.



Импорт CSS

В текущую стилевую таблицу можно импортировать содержимое CSS-файла с помощью команды @import. Этот метод допускается использовать совместно со связанными или глобальными стилями, но никак не с внутренними стилями. Общий синтаксис следующий.

@import url("имя файла") типы носителей;

@import "имя файла" типы носителей;

После ключевого слова @import указывается путь к стилевому файлу одним из двух приведенных способов — с помощью url или без него.

<!DOCTYPE HTML>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Импорт**</title>**

**<style>**

@import url("**style**/**header**.css");

**H1**{

font-size:120%;

font-family: Arial, Helvetica, sans-serif;

color: green;

}

**</style>**

**</head>**

**<body>**

**<h1>**Заголовок 1**</h1>**

**<h2>**Заголовок 2**</h2>**

**</body>**

**</html>**

Посмотреть пример  
Посмотреть пример

В данном примере показано подключение файла header.css, который расположен в папке style.

Аналогично происходит импорт и в файле со стилем, который затем подключается к документу.

@import "/**style**/print.css";

@import "/**style**/palm.css";

**BODY**{

font-family: Arial, Verdana, Helvetica, sans-serif;

font-size:90%;

background: white;

color: black;

}

Свойство padding задает внутренние отступы элемента — отступы от внешней границы элемента до его содержания. Эти отступы еще иногда называют полями.

Существует несколько способов записи свойства padding.

padding: 10px;

padding: 5px 10px;

padding: 5px 10px 15px;

padding: 5px 10px 15px 20px;

1. Одинаковые отступы со всех сторон.
2. Отступы сверху и снизу 5px, справа и слева 10px.
3. Отступ сверху 5px, слева и справа 10px, снизу 15px.
4. Разные отступы со всех сторон, в порядке верхний, правый, нижний, левый.

Также можно задавать отступы для разных сторон с помощью свойств padding-left, padding-right, padding-top, padding-bottom.

Для строчных элементов лучше не задавать вертикальных отступов, так как они ведут себя непредсказуемо.

margin

— oпределяет ширину поля за пределами границы элемента:

margin: margin-top | margin-right | margin-bottom | margin-left

margin: 25px 30px 40px 80px   →   *указаны ширины внешних полей (по часовой стрелке:*

*верхнее, правое, нижнее, левое)*;

margin: 0 auto   →   *позволяет расположить блок по центру.*

margin-top

— верхнее внешнее поле элемента:

margin-top: 20px   →   *величина может быть указана в пикселях или в процентах.*

margin-right

— определяет ширину правого внешнего поля:

margin-right: 40px

margin-bottom | margin-left

— определяют соответственно нижнее и левое внешнее поле.

CSS поддерживает множество единиц измерений, включая абсолютные единицы измерения, такие как дюймы, сантиметры, пункты и т.д., а также относительные единицы измерения, такие как проценты, em и т.д. Эти значения необходимо использовать при указании различных измерений в ваших правилах стиля, например border = "5px solid blue".

Ниже перечислены все единицы измерения CSS вместе с соответствующими примерами:

|  |  |  |
| --- | --- | --- |
| **Единица измерения** | **Описание** | **Пример** |
| % | Устанавливает измерение в процентах относительно другого значения, как правило, охватывающего элемента. | div {width: 50%;} |
| cm | Определяет измерение в сантиметрах. | p {padding-top: 1cm;} |
| em | Относительное измерение высоты шрифта в em пространстве. Поскольку единица em эквивалентна размеру данного шрифта, то, если Вы укажите шрифт 12pt, каждый элемент «em» будет 12pt. Таким образом, 2em будет 24pt. | h1 {letter-spacing: 3em;} |
| ex | Это значение устанавливает измерение относительно x-высоты шрифта. x-высота определяется высотой строчной буквы x шрифта. | p {letter-spacing: 7ex;} |
| in | Определяет измерение в дюймах. | h2 {word-spacing: .10in;} |
| mm | Устанавливает измерение в миллиметрах. | div {margin: .15mm;} |
| pc | Определяет измерение в пиках. Пика эквивалентна 12 пунктам (12pt). | h3 {font-size: 10pc;} |
| pt | Определяет измерение в типографских пунктах (твип), введённых компанией Adobe. Пункт равен 1/72 дюйма или 25,4/72 мм = 0,3528 мм. В странах СНГ, входящих в Таможенный союз, 1 пункт равен 0,376 мм. | table {font-size: 20pt;} |
| px | Устанавливает измерение в пикселях экрана. | p {margin: 30px;} |
| vh | Равен 1% высоты окна просмотра. | h1 {font-size: 2.0vh;} |
| vw | Равен 1% ширины окна просмотра. | h2 {font-size: 2.0vw;} |
| vmin | Равен 1vw или 1vh, в зависимости от того, что меньше. | div {font-size: 5vmin;} |

# Все значения свойства "display"

Свойство display имеет много разных значений. Обычно, используются только три из них: none, inline и block, потому что когда-то браузеры другие не поддерживали.

Но после ухода IE7-, стало возможным использовать и другие значения тоже. Рассмотрим здесь весь список.

## [Значение none](https://learn.javascript.ru/display" \l "znachenie-none)

Самое простое значение. Элемент не показывается, вообще. Как будто его и нет.

<div style="border:1px solid black">

Невидимый div (

<divstyle="display:none">Я - невидим!</div>

) Стоит внутри скобок

</div>

## [Значение block](https://learn.javascript.ru/display" \l "znachenie-block)

* Блочные элементы располагаются один над другим, вертикально (если нет особых свойств позиционирования, например float).
* Блок стремится расшириться на всю доступную ширину. Можно указать ширину и высоту явно.

Это значение display многие элементы имеют по умолчанию: <div>, заголовок <h1>, параграф <p>.

<div style="border:1px solid black">

<div style="border:1px solid blue;width:50%">Первый</div>

<div style="border:1px solid red">Второй</div>

</div>

Блоки прилегают друг к другу вплотную, если у них нет margin.

## [Значение inline](https://learn.javascript.ru/display" \l "znachenie-inline)

* Элементы располагаются на той же строке, последовательно.
* Ширина и высота элемента определяются по содержимому. Поменять их нельзя.

Например, инлайновые элементы по умолчанию: <span>, <a>.

<span style="border:1px solid black">

<span style="border:1px solid blue;width:50%">Ширина</span>

<a style="border:1px solid red">Игнорируется</a>

</span>

Если вы присмотритесь внимательно к примеру выше, то увидите, что между внутренними <span> и <a> есть пробел. Это потому, что он есть в HTML.

Если расположить элементы вплотную – его не будет:

<span style="border:1px solid black">

<span style="border:1px solid blue;width:50%">Без</span><a style="border:1px solid red">Пробела</a>

</span>

Содержимое инлайн-элемента может переноситься на другую строку.

При этом каждая строка в смысле отображения является отдельным прямоугольником («line box»). Так что инлайн-элемент состоит из объединения прямоугольников, но в целом, в отличие от блока, прямоугольником не является.

Это проявляется, например, при назначении фона.

Например, три прямоугольника подряд:

<divstyle="width:400px">

...<spanstyle="background:lightgreen">

ЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛя

ЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛяЛя

</span>...

</div>

Если инлайн-элемент граничит с блоком, то между ними обязательно будет перенос строки:

<div style="border:1px solid black">

<span style="border:1px solid red">Инлайн</span>

<div style="border:1px solid blue;width:50%">Блок</div>

<span style="border:1px solid red">Инлайн</span>

</div>

## [Значение inline-block](https://learn.javascript.ru/display" \l "znachenie-inline-block)

Это значение – означает элемент, который продолжает находиться в строке (inline), но при этом может иметь важные свойства блока.

Как и инлайн-элемент:

* Располагается в строке.
* Размер устанавливается по содержимому.

Во всём остальном – это блок, то есть:

* Элемент всегда прямоугольный.
* Работают свойства width/height.

Это значение display используют, чтобы отобразить в одну строку блочные элементы, в том числе разных размеров.

Например:

<style>

li{

display: inline-block;

list-style: none;

border:1px solid red;

}

</style>

<ul style="border:1px solid black;padding:0">

<li>Инлайн Блок<br>3 строки<br>высота/ширина явно не заданы</li>

<li style="width:100px;height:100px">Инлайн<br>Блок 100x100</li>

<li style="width:60px;height:60px">Инлайн<br>Блок 60x60</li>

<li style="width:100px;height:60px">Инлайн<br>Блок 100x60</li>

<li style="width:60px;height:100px">Инлайн<br>Блок 60x100</li>

</ul>

Свойство vertical-align позволяет выровнять такие элементы внутри внешнего блока:

<style>

li{

display: inline-block;

list-style: none;

border:1px solid red;

vertical-align: middle;

}

</style>

<ul style="border:1px solid black;padding:0">

<li>Инлайн Блок<br>3 строки<br>высота/ширина явно не заданы</li>

<li style="width:100px;height:100px">Инлайн<br>Блок 100x100</li>

<li style="width:60px;height:60px">Инлайн<br>Блок 60x60</li>

<li style="width:100px;height:60px">Инлайн<br>Блок 100x60</li>

<li style="width:60px;height:100px">Инлайн<br>Блок 60x100</li>

</ul>

Как и в случае с инлайн-элементами, пробелы между блоками появляются из-за пробелов в HTML. Если элементы списка идут вплотную, например, генерируются в JavaScript – их не будет.

## [Значения table-\*](https://learn.javascript.ru/display" \l "znacheniya-table)

Современные браузеры (IE8+) позволяют описывать таблицу любыми элементами, если поставить им соответствующие значения display.

Для таблицы целиком table, для строки – table-row, для ячейки – table-cell и т.д.

Примериспользования:

<form style="display: table">

<div style="display: table-row">

<label style="display: table-cell">Имя:</label>

<input style="display: table-cell">

</div>

<div style="display: table-row">

<label style="display: table-cell">Фамилия:</label>

<input style="display: table-cell">

</div>

</form>

Важно то, что это действительно полноценная таблица. Используются табличные алгоритмы вычисления ширины и высоты элемента, [описанные в стандарте](http://www.w3.org/TR/CSS2/tables.html#width-layout).

**Это хорошо для семантической вёрстки и позволяет избавиться от лишних тегов.**

С точки зрения современного CSS, обычные <table>, <tr>, <td> и т.д. – это просто элементы с предопределёнными значениями display:

table{display: table }

tr{display: table-row }

thead{display: table-header-group }

tbody{display: table-row-group }

tfoot{display: table-footer-group }

col{display: table-column }

colgroup{display: table-column-group }

td, th{display: table-cell }

caption{display: table-caption }

Очень подробно об алгоритмах вычисления размеров и отображении таблиц рассказывает стандарт [CSS 2.1 – Tables](http://www.w3.org/TR/CSS2/tables.html).

### [Вертикальное центрирование с table-cell](https://learn.javascript.ru/display" \l "vertikalnoe-tsentrirovanie-s-table-cell)

Внутри ячеек свойство [vertical-align](http://www.w3.org/TR/CSS2/visudet.html#propdef-vertical-align) выравнивает содержимое по вертикали.

Это можно использовать для центрирования:

<style>

div{border:1px solid black }

</style>

<div style="height:100px;display: table-cell;vertical-align: middle">

<div>Элемент<br>С неизвестной<br>Высотой</div>

</div>

CSS не требует, чтобы вокруг table-cell была структура таблицы: table-row и т.п. Может быть просто такой одинокий DIV, это допустимо.

При этом он ведёт себя как ячейка TD, то есть подстраивается под размер содержимого и умеет вертикально центрировать его при помощи vertical-align.

## [Значения list-item, run-in и flex](https://learn.javascript.ru/display" \l "znacheniya-list-item-run-in-i-flex)

У свойства display есть и другие значения. Они используются реже, поэтому посмотрим на них кратко:

**list-item**

Этот display по умолчанию используется для элементов списка. Он добавляет к блоку содержимым ещё и блок с номером(значком) списка, который стилизуется стандартными списочными свойствами:

<div style="display: list-item;list-style:inside square">Пункт 1</div>

**run-in**

Если после run-in идёт block, то run-in становится его первым инлайн-элементом, то есть отображается в начале block.

Если ваш браузер поддерживает это значение, то в примере ниже h3, благодаря display:run-in, окажется визуально внутри div:

<h3 style="display: run-in;border:2px solid red">Пропчёл.</h3>

<div style="border:2px solid black">Пчёлы - отличные создания, они делают мёд.</div>

Если же вы видите две строки, то ваш браузер НЕ поддерживает run-in.

Вот, для примера, правильный вариант отображения run-in, оформленный другим кодом:

<div style="border:2px solid black">

<h3 style="display: inline;border:2px solid red">Пропчёл.</h3>Пчёлы - отличныесоздания, ониделаютмёд.

</div>

Если этот вариант отличается от того, что вы видите выше – ваш браузер не поддерживает run-in. На момент написания этой статьи только IE поддерживал display:run-in.

**flex-box**

Flexbox позволяет удобно управлять дочерними и родительскими элементами на странице, располагая их в необходимом порядке

Для начала нужно понять, что css не создает блоки – он оформляет их. Сами по себе они формируются в HTML – языке разметки гипертекста. По умолчанию блок по высоте будет занимать место, равное высоте его содержимого. По ширине же он займет все доступное пространство, которое есть в родителе.

Для примера: если ширина родительского контейнера составляет 300 пикселей и в нем создан блок, то он тоже будет занимать такую ширину (только если у родителя не заданы отступы). Все остальные элементы, которые будут создаваться в контейнере, не могут стоять на одной линии с первым блоком и будут перенесены дальше.

Обычный пустой блочный элемент записывается с помощью парного тега — div. Сам по себе он ничего не делает. Например, давайте вспомним, что многие теги имеют определенный смысл: абзац является частью какого-то текста, подзаголовки нужны для улучшения навигации и восприятия и т.д. А вот тег div выступает просто контейнером.

Давайте сначала разберемся с размерами. В css размеры элементам можно записывать с помощью свойств width и height.

Color

Свойство задаёт цвет шрифта с помощью различных систем цветопередачи. Свойство описывает цвет текстового содержимого элемента. Кроме того, оно используется для предоставления потенциального косвенного значения (currentColor) для любых других свойств, которые принимают значения цвета.

Свойство наследуется.

Список основных ключевых слов включает в себя следующие значения:

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **HEX** | **RGB** | **Цвет** |
| black | #000000 | 0,0,0 |  |
| silver | #C0C0C0 | 192,192,192 |  |
| gray | #808080 | 128,128,128 |  |
| white | #FFFFFF | 255,255,255 |  |
| maroon | #800000 | 128,0,0 |  |
| red | #FF0000 | 255,0,0 |  |
| purple | #800080 | 128,0,128 |  |
| fuchsia | #FF00FF | 255,0,255 |  |
| green | #008000 | 0,128,0 |  |
| lime | #00FF00 | 0,255,0 |  |
| olive | #808000 | 128,128,0 |  |
| yellow | #FFFF00 | 255,255,0 |  |
| navy | #000080 | 0,0,128 |  |
| blue | #0000FF | 0,0,255 |  |
| teal | #008080 | 0,128,128 |  |
| aqua | #00FFFF | 0,255,255 |  |

RGB

Формат значения RGB в шестнадцатеричном формате — это знак #, за которым сразу следуют три или шесть шестнадцатеричных символов. Трехзначная запись RGB #rgbпреобразуется в шестизначную форму #rrggbb путем копирования цифр, а не путем добавления нулей. Например, #fb0 расширяется до #ffbb00. Это гарантирует, что белый #ffffff может быть указан в короткой записи #fff, и удаляет любые зависимости от глубины цвета дисплея.

Формат значения RGB в функциональной нотации — rgb(, за которым следует разделенный запятыми список из трех числовых значений (либо трех целочисленных значений, либо трех процентных значений), за которыми следует символ ). Целочисленное значение 255 соответствует 100% и F или FF в шестнадцатеричной записи:

rgb (255,255,255) = rgb (100%, 100%, 100%) = #FFF

Символы пробела допускаются вокруг числовых значений.

RGBA

Цветовая модель RGB расширена в этой спецификации, чтобы включить alpha, которая управляющая непрозрачностью цвета. В отличие от значений RGB, для значения RGBA нет шестнадцатеричной записи.

Формат значения [RGBA](https://html5book.ru/transparency/#rgba) в функциональной нотации — rgba( за которым следует разделенный запятыми список из трех числовых значений (либо трех целочисленных значений, либо трех процентных значений), за которыми следует значение непрозрачности, а затем ). Целочисленное значение 255 соответствует 100%, rgba (255,255,255,0.8) = rgba (100%,100%,100%,0.8). Символы пробела допускаются вокруг числовых значений.

Параметр непрозрачности применяется ко всему объекту. Любые значения за пределами диапазона от 0.0 (полностью прозрачный) до 1.0 (полностью непрозрачный) будут ограничены этим диапазоном.

**История и стандарты, базовые элементы.**

Служба Интернет World Wide Web (Всемирная Паутина) в настоящее время используемая наиболее интенсивно, появилась в 1995 г., в Европейском Центре Ядерных Исследований в Швеции. Основу WWW составляют hypermedia-документы или Web-страницы. Эти документы могут содержать как текстовые, так и не текстовые данные (например, изображения, звук), а также ссылки. Ссылки – это специальные элементы, позволяющие автоматизировать процедуру открытия и загрузки документа, на который указывает данная ссылка. Эта ссылка может указывать на документ, который расположен на жестком диске компьютера, находящегося на другом континенте.

С появлением службы WWW стало легко находить нужную информацию с помощью простого указания и щелчка мышью. Поэтому служба WWW приобрела всемирную популярность и получила большое распространение. Каждый день в сети Интернет появляются новые Web–серверы и публикуются тысячи документов.

Для построения Web-страницы в WWW используется специальный язык HTML, что означает Hyper Text Markup Language – язык гипертекстовой разметки, язык форматирования данных. Язык HTML определяет форматирование и организацию данных в Web-страницах. Документ на языке HTML – это файл в ASCII-формате, состоящий из специальных дескрипторов, называемых теги. Эти теги и определяют форматирование данных в любом Web-документе.

Когда говорят об "Интернет-стандарте", в большинстве случаев имеется в виду техническая спецификация протокола, программного интерфейса, схемы базы данных и тому подобных вещей. Стандарт - это своего рода "строительный блок", призванный в совокупности с другими элементами создать систему или решение. Для этого наряду со стандартами существуют также информационные документы с рекомендациями по применению стандарта или технологии для решения определенных задач. Обычно организации, занимающиеся стандартизацией, разрабатывают оба типа спецификаций.

Для правильного представления, как и какие стандарты определяют функционирование Интернета, полезно кратко остановиться на архитектурной модели Сети.

С точки зрения протоколов модель Интернета, основанная на протоколах TCP/IP, состоит из четырех уровней: канальный, сетевой, транспортный и уровень приложений. Ниже я кратко приведу их основные характеристики:

* **Канальный уровень** включает технологии и протоколы передачи данных в физической и локальной сети. Этому уровню принадлежат такие технологии как Ethernet, Frame Relay, ATM, MPLS. В модели TCP/IP в этот уровень также включены стандарты кодирования и передачи сигналов в физической сети - оптическое волокно, радиосигнал и т.п.
* **Сетевой уровень** определяет передачу данных между локальными сетями, обеспечивая создание интер-сетей, или собственно Интернета. Этот уровень является глобальным и универсальным - каждое устройство, непосредственно подключенное к Интернету, взаимодействует с другими устройствами на этом уровне. Сетевая технология различных сетей, составляющих Интернет, может быть различна, также как и приложения и услуги, предоставляемые в этих сетях. Однако протокол IP - основной протокол сетевого уровня - является общим знаменателем, определяющим Интернет, по крайней мере сегодня.
* Услугами протоколов **транспортного уровня** пользуются приложения, расположенные на различных хостах. Эти протоколы обеспечивают сквозную связность между хостами, а также дополнительные функции, такие как мультиплексирование виртуальных каналов, гарантированную безошибочную передачу данных, контроль пропускной способности и т.п. Основными протоколами этого уровня являются TCP и UDP. Первый из них обеспечивает обмен данными между приложениями с созданием виртуального соединения, а UDP - обмен "дейтаграммами" без создания соединения.
* **Уровень приложений, или прикладной уровень,** содержит протоколы обмена данными между приложениями, или процессами. Наиболее значительными приложениями, использующими протоколы этого уровня являются (приведу лишь некоторые протоколы):
  + Электронная почта: SMTP, POP, IMAP
  + Передача файлов: FTP, TFTP
  + Коллаборативные вэб-платформы: HTTP, WebDAV
  + Голосовая связь: SIP
  + Обмен сообщениями: XMPP
  + Инфраструктурные приложения: DNS, DHCP, TLS/SSL

Каждый протокол выполняет наиболее универсальные функции, необходимые для взаимодействия между устройствами на конкретном уровне. Например, Ethernet (IEEE 802.3) обеспечивает обмен данными между сетевыми интерфейсами локальной сети. Он поддерживает различные типы среды передачи (от коаксиального кабеля до оптоволокна) и скорости (от 10 Мбит/с до 100 Гбит/с). Однако, хотя Ethernet и обеспечивает обнаружение ошибочных данных (фреймов), исправление ошибок (например путем повторной передачи) производится протоколами верхних уровней.

Такой подход обладает поистине неограниченным инновационным потенциалом, поскольку изменения протокола одного уровня не затрагивают протоколы других уровней, при условии, что интерфейсы взаимодействия между протоколами неизменны. Так, эволюция того же Ethernet происходила абсолютно независимо от протокола следующего уровня - IP. А для создания нового приложения (или протокола прикладного уровня) нет необходимости требовать каких-либо изменений от Сети.

**XHTML** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *e****x****tensible****h****yper****t****ext****m****arkup****l****anguage* — *расширяемый язык гипертекстовой разметки*) — семейство [языков разметки](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D1%80%D0%B0%D0%B7%D0%BC%D0%B5%D1%82%D0%BA%D0%B8) [веб-страниц](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0) на основе [XML](https://ru.wikipedia.org/wiki/XML), повторяющих и расширяющих возможности [HTML 4](https://ru.wikipedia.org/wiki/HTML_4). Спецификации XHTML 1.0 и XHTML 1.1 являются рекомендациями [консорциума Всемирной паутины](https://ru.wikipedia.org/wiki/W3C). Развитие XHTML остановлено; новые версии XHTML не выпускаются; рекомендуется использовать [HTML](https://ru.wikipedia.org/wiki/HTML).

Главное отличие XHTML от HTML заключается в обработке документа. Документы XHTML обрабатываются своим модулем ([парсером](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D1%81%D0%B5%D1%80)) аналогично документам XML. В процессе этой обработки ошибки, допущенные разработчиками, не исправляются.

XHTML соответствует спецификации [SGML](https://ru.wikipedia.org/wiki/SGML), поскольку [XML](https://ru.wikipedia.org/wiki/XML) является её подмножеством. HTML обладает множеством особенностей в процессе обработки и фактически перестал относиться к семейству SGML, что и закреплено в черновике спецификации HTML 5.

Браузер выбирает [парсер](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%82%D0%BE%D1%80) для обработки документа на основании заголовка *content-type*, полученного от сервера:

* HTML — text/html
* XHTML — application/xhtml+xml
* Для локального просмотра на клиенте выбор основывается на [расширении файла](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%88%D0%B8%D1%80%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B8%D0%BC%D0%B5%D0%BD%D0%B8_%D1%84%D0%B0%D0%B9%D0%BB%D0%B0)[[*уточнить*](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%BA%D0%B8%D0%BF%D0%B5%D0%B4%D0%B8%D1%8F:%D0%90%D0%B2%D1%82%D0%BE%D1%80%D0%B8%D1%82%D0%B5%D1%82%D0%BD%D1%8B%D0%B5_%D0%B8%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8)].
* В [Internet Explorer](https://ru.wikipedia.org/wiki/Internet_Explorer) вплоть до 8-й версии парсер обработки XHTML-документов отсутствует.

Декларация DOCTYPE может не оказывать никакого влияния на определение парсера для обработки (зависит от используемого браузера).

Блочные элементы занимают всю доступную ширину и располагаются один под другим. Перенос слов осуществляется в границах блочного элемента.

Как пример блочных элементов можно привести абзацы. Абзацы выстраиваются один под другим, занимают всю ширину страницы. Текст в абзацах переносится по словам.

Абзацы <p>

Абзацы — основной структурный элемент Web-страницы.

Абзацы создаются с помощью элемента <p>. Его имя происходит от английского слова «paragraph» — «абзац».

Между тегами <p> и </p>распологается текст абзаца.

Заголовки бывают шести уровней.

Заголовок первого уровня является названием страницы. Он создаётся с помощью элемента <h1>. Заголовок второго уровня разбивает страницу на логические разделы. Он создаётся с помощью элемента <h2>. Заголовок третьего уровня разбивает разделы на смысловые части. Он создаётся с помощью элемента <h3>и т.д.

Первым элементом на странице всегда должен быть элемент <h1>.  
Заголовки являются блочными элементами.

Заголовки нельзя включать в абзацы, а абзацы – в заголовки.

Элементы <h1>, <h2>, <h3>, <h4>, <h5> и <h6> включаются в элемент <body>.

Размеченный текст <pre>

В XHTML пробелы и начала строк игнорируются. Если нужно обойти такое стандартное поведение обозревателя, применяют элемент <pre>. Текст внутри элемента <pre> выводится в том же виде, в каком он был сохранён, с соблюдением всех пробелов и новых строк.

Линейные элементы занимают ровно столько места, сколько им необходимо. При необходимости, линейный элемент может разорваться и продолжиться на следующей строке. Линейные элементы располагаются слева направо и могут свободно включаться в текст.

К линейным элементам относятся элементы логического форматирования: <strong> и <em> и ссылки — элемент <a>.

Элементы «усиление» и «эмфазис»

Линейные элементы <strong> и <em> используются для расстановки логических ударений в тексте. Поскольку эти элементы линейны, они позволяют выделить букву, слово или фразу.

Обычно <strong> отображается полужирным, а <em> — курсивом. Но это стандартное поведение легко может быть переопределено в стилях.

Сосуществование блочных и линейных элементов

Линейные элементы могут включаться как в блочные, так и в линейные элементы. Блочные элементы включаются только в блочные элементы. В абзацы никакие блочные элементы включаться не могут.

В линейные элементы нельзя включать блочные, так как при этом нарушаются сами определения блочности и линейности.

**Cсылки и пути.**

С помощью элемента <a> создаются ссылки с Web-страницы на различные ресурсы.  
Для элемента <a> определён обязательный атрибут href, задающий подключаемый ресурс.

Ссылки являются линейными элементами. Элементы <a> включаются в элемент <body> и в другие блочные и линейные элементы.

Ссылки не могут включаться одна в другую, то есть в элемент <a> нельзя включать другие элементы <a>.

Атрибут href

Атрибут href задаёт ресурс, на который указывает ссылка.

Если файл, на который производится ссылка, находится в той же папке, что Web-страница, как значение атрибута href указывается просто имя этого файла (обязательно с расширением) с соблюдением регистра. Например, в этой папке есть файл archive.zip. Ссылку на него можно создать таким образом:

<body>

  <h1>ЗАКАЧКА ФАЙЛА</h1>

  <p>

    Здесь находится <a href="archive.zip">архив</a> программы.

  </p>

</body>

Атрибут title

Атрибут title позволяет определить всплывающую подсказку. Значение атрибута title и будет её текстом.

Атрибут title определён для всех элементов, то есть любому элементу может быть назначена всплывающая подсказка.

Адрес ссылки может быть как абсолютным, так и относительным. Абсолютные адреса должны начинаться с указания протокола (обычно http://) и содержать имя сайта. Относительные ссылки ведут отсчет от корня сайта или текущего документа.

Абсолютные ссылки обычно применяются для указания документа на другом сетевом ресурсе, впрочем, допустимо делать абсолютные ссылки и внутри текущего сайта. Однако подобное практикуется нечасто, поскольку такие ссылки достаточно длинные и громоздкие. Поэтому внутри сайта преимущественно используются относительные ссылки.

При создании относительных ссылок надо понимать, какое значение для атрибута href следует указывать, поскольку оно зависит от исходного расположения файлов. Рассмотрим несколько типичных вариантов.

1. Файлы располагаются в одной папке

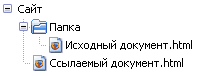
Рис. 8.4

Необходимо сделать ссылку из исходного документа на ссылаемый. В таком случае код будет следующий.

**<a** href="Ссылаемый документ.html"**>**Ссылка**</a>**

Подобное имя файла взято только для образца, на сайте в именах файлов не следует использовать русские символы с пробелами, да еще и в разном регистре.

2. Файлы размещаются в разных папках

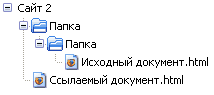


Когда исходный документ хранится в одной папке, а ссылаемый в корне сайта, то перед именем файла в адресе ссылки следует поставить две точки и слэш (/), как показано ниже.

**<a** href="../Ссылаемый документ.html"**>**Ссылка**</a>**

Две точки в данном случае означают выйти из текущей папки на уровень выше.

3. Файлы размещаются в разных папках

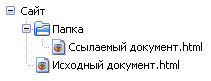


Теперь исходный файл находится в двух вложенных папках, и чтобы сослаться на документ в корне сайта, требуется повторить написание предыдущего примера два раза.

**<a** href="../../Ссылаемый документ.html"**>**Ссылка**</a>**

Аналогично обстоит дело с любым числом вложенных папок.

4. Файлы размещаются в разных папках



Теперь ситуация меняется, исходный файл располагается в корне сайта, а файл, на который необходимо сделать ссылку - в папке. В этом случае путь к файлу будет следующий.

**<a** href="Папка/Ссылаемый документ.html"**>**Ссылка**</a>**

Заметьте, что никаких дополнительных точек и слэшей перед именем папки нет. Если файл находится внутри не одной, а двух папок, то путь к нему записывается так.

**<a** href="Папка 1/Папка 2/Ссылаемый документ.html"**>**Ссылка**</a>**

### Ссылки относительно корня сайта

Иногда можно встретить путь к файлу относительно корня сайта, он выглядит как "/Папка/Имя файла" со слэшем вначале. Так, запись <a href="/course/">Курсы</a> означает, что ссылка ведет в папку с именем course, которая располагается в корне сайта, а в ней необходимо загрузить индексный файл.

Учтите, что такая форма записи не работает на локальном компьютере, а только под управлением веб-сервера.

## Что такое абсолютный путь к файлу

Абсолютный адрес ссылки на файл включает в себя протокол (например, **http://**), имя сайта в Сети, подкаталог (или несколько подкаталогов), название файла. Пример того, как выглядит абсолютный URL:

**https://www.google.com/images/branding/googlelogo.png**

Абсолютный путь можно использовать в CSS. Например, чтобы задать рисунок в качестве фона, вы можете взять абсолютный URL к файлу с изображением И указать его в таблице стилей:

background-image: url(http://www.example.com/img/bg.png);

Абсолютная ссылка всегда будет указывать на одно и то же место, независимо от того, где расположен содержащий ее документ (в нашем случае таблица стилей). Чего не скажешь об относительной ссылке.

## Что такое относительный путь к файлу

С относительным адресом всё намного интереснее — он может отсчитываться как от корня сайта, так и от текущего документа. **Корневой относительный путь** — это путь, который указывает на расположение файла относительно корневого каталога сайта. В этом случае адрес не содержит ни протокола, ни имени домена, и начинается со знака слэша **/**, который указывает на корневую папку. Выглядит этот адрес так:

background-image: url(/img/bg.png);

## Какой путь лучше использовать

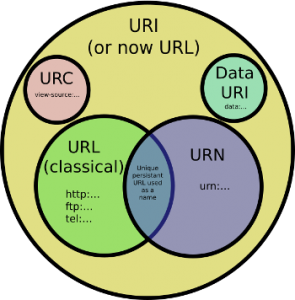
**Абсолютный адрес ссылки** понадобится использовать, если файл, на который вы ссылаетесь, находится на другом сайте. В пределах одного сайта абсолютный путь практически не используется (хоть он и будет работать). Это связано с несколькими моментами: во-первых, такие ссылки могут быть весьма громоздкими, а во-вторых, если имя домена изменится, то возникнет проблема — все ссылки будут вести на старый домен, что может повлечь за собой множество ошибок и битых ссылок.

**Корневой относительный путь** будет работать на веб-сервере, но он бесполезен при разработке на локальном компьютере. Но преимущество этого типа ссылок в том, что вы можете перемещать свою таблицу стилей в любую папку сайта и не бояться, что адреса фоновых изображений станут нерабочими.

**Относительный путь от документа** — наилучший вариант при веб-разработке на локальной машине без использования сервера. В браузере вы сможете нормально просматривать веб-страницы, хранящиеся на вашем компьютере, и адреса файлов будут работать. Они будут работать и на живом сайте, но с одним условием — если не перемещать таблицу стилей в другое место (иначе придется корректировать пути).

Попробуйте потренироваться в использовании относительных путей как на локальном компьютере, так и на веб-сервере. Понимание того, как это работает, непременно пригодится вам в любом проекте.

**URL** (или **URL адрес**) – это форма уникального адреса конкретного веб-ресурса в сети Интернет. Он может ссылаться на веб-сайт, какой-то индивидуальный документ или изображение. Пользователю Интернета нужно вставить этот код в поле поиска, чтобы найти нужный сайт, документ, папку или изображение. На простом языке это означает следующее: благодаря URL адресу пользователь узнает информацию о том, где находятся нужные ему данные.



Якорем называется закладка с уникальным именем на определенном месте веб-страницы, предназначенная для создания перехода к ней по ссылке. Якоря удобно применять в документах большого объема, чтобы можно было быстро переходить к нужному разделу.

Для создания якоря следует вначале сделать закладку в соответствующем месте и дать ей имя при помощи атрибута name тега **<a>**. В качестве значения href для перехода к этому якорю используется имя закладки с символом решетки (#) впереди.

 <a name=: "mark\_name"></a>

<a href="#mark\_name">Я – закладка</a>

Впервые появившиеся в программе Netscape Navigator 2.0 кадры HTML помогают разбить страницу на независимые (даже с прокруткой) окна, выводящие информацию различных типов. Кадры во многом похожи на таблицы HTML, но, в отличие от последних, не только организовывают данные, но и размещают их на экране дисплея. Они делят окно браузера на отдельные независимые панели, каждая их которых выводит свой файл. Другими словами, каждая такая панель представляет собой “мини-браузер”.

Кадр должен быть расположен в контейнере FRAMESET, который занимает место контейнера BODY. HTML документ, имеющий структуру FRAMESET, не имеет структуры BODY, и наоборот.

Парный тэг <FRAMESET></FRAMESET> используется при определении каждого кадра. Внутри этого контейнера могут располагаться только тэг <FRAME> или другие контейнеры FRAMESET.

Для определения имени кадра служит атрибут NAME. Например, строка

<FRAME SRC= “info.htm” NAME= “Joe”>

создает кадр по имени “Joe”, на который можно сделать гипертекстовую ссылку следующим образом:

<A HREF= “moreinfo.htm” TARGET= “Joe”> Click here to jump to Joe </A>

Списки, символы, кодировка, метаданные.

Списком называется взаимосвязанный набор отдельных фраз или предложений, которые начинаются с маркера или цифры. Списки предоставляют возможность упорядочить и систематизировать разные данные и представить их в наглядном и удобном для пользователя виде.

Любой список представляет собой контейнер **<ul>**, который устанавливает маркированный список, или **<ol>**, определяющий список нумерованный. Каждый элемент списка должен начинаться с тега **<li>**.

Маркированный список определяется тем, что перед каждым элементом списка добавляется небольшой маркер, обычно в виде закрашенного кружка. Сам список формируется с помощью контейнера **<ul>**, а каждый пункт списка начинается с тега **<li>**, как показано ниже.

**<ul>**

**<li>**Первый пункт**</li>**

**<li>**Второй пункт**</li>**

**<li>**Третий пункт**</li>**

**</ul>**

Нумерованные списки представляют собой набор элементов с их порядковыми номерами. Вид и тип нумерации зависит от атрибутов тега **<ol>**, который и применяется для создания списка. Каждый пункт нумерованного списка обозначается тегом **<li>**, как показано ниже.

**<ol>**

**<li>**Первый пункт**</li>**

**<li>**Второй пункт**</li>**

**<li>**Третий пункт**</li>**

**</ol>**

**Юнико́д** (чаще всего) или Унико́д (англ. **Unicode**) — стандарт кодирования символов, включающий в себя знаки почти всех письменных языков мира. Юникод это возможность корректно отображать буквы и знаки пунктуации из различных языков на одной страницы.

Почему вам стоит использовать Юникод?

Для поддержки UTF-8 в HTML5 добавьте <meta charset=utf-8> (при отсутствии доступа к настройкам сервера стоит добавить также <meta http-equiv= "Content-Type" content="text/html; charset=utf-8">). При старом доктайпе используется (<meta http-equiv="content-type" content="text/html; charset=UTF-8" />).

Вторая причина использования Юникода это наличие большого количества полезных символов, которые можно использовать в качестве иконок. Например, ▶, ≡ и ♥.

Их очевидный плюс в том, что вам не надо никаких дополнительных файлов, чтобы добавить их на страницу, а, значит, ваш сайт будет быстрее. Вы также можете изменить их цвет или добавить тень с помощью CSS. А добавив переходы (css transition) вы сможете плавно менять цвет иконки при наведении на нее без каких-либо дополнительных изображений.

Символьные подстановки представляют собой специальную последовательность, преобразуемую браузерами в заданный символ. Подстановку можно осуществить несколькими способами:

&мнемокод; - вставка символа по его "мнемокоду" (имени).

&#КОД10; - вставка символа по его десятичному коду.

&#xКОД16; - вставка символа по его шеснадцатиричному коду.

Ниже приводится таблица наиболее популярных символьных подстановок:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Символ | Мнемокод | 10 Код | 16 Код | Пояснение |  |
| < | &lt; | &#60; | &#x3C; |  |  |
| > | &gt; | &#62; | &#x3E; |  |  |
| & | &amp; | &#38; | &#x26; | Амперсанд |  |
| " | &quot; | &#34; | &#x22; |  |  |
|  | &nbsp; | &#160; | &#A0; | Символ "неразрывного пробела". Слова, разделяемые им не переносятся по отдельности. |  |
|  | &shy; | &#173; | &#AD; | Символ "мягкого переноса". Части слова, соединяемые им, переносятся, при необходимости. На месте переноса отображается символ "-" |  |
| © | &copy; | &#169; | &#A9; |  |  |
| ® | &reg; | &#174; | &#AE; |  |  |
| ™ | &trade; | &#8482; | &#2122; |  |  |
| ¤ | &curren; |  |  |  |  |
| ¦ | &brvbar; |  |  |  |  |
| § | &sect; |  |  |  |  |
| ª | &ordf; |  |  |  |  |
| « | &laquo; |  |  |  |  |
| ° | &deg; |  |  |  |  |
| ± | &plusmn; |  |  |  |  |
| µ | &micro; |  |  |  |  |
| ¶ | &para; |  |  |  |  |
| · | &middot; |  |  |  |  |
| ¹ | &sup1; |  |  |  |  |
| € | &euro; |  |  |  |  |
| ʼ ˈˌːˑ ˘ ‖‿ |  | &#700; &#712; &#716; &#720; &#721; &#728; &#8214; &#8255; |  |  |  |

**Пакет** — это определённым образом оформленный блок [данных](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5), передаваемый по сети в пакетном режиме.

Пакет состоит из двух типов данных: управляющей информации и данных пользователя (называемых также полезной нагрузкой). Управляющая информация содержит данные, необходимые для доставки данных пользователя: адреса отправителя и получателя, коды обнаружения ошибок (типа контрольных сумм) и информацию об очерёдности. Как правило, управляющая информация содержится в заголовке и хвосте пакета, а между ними размещаются пользовательские данные.

Заголовок-сущность **Content-Type** используется для того, чтобы определить MIME тип ресурса.

В ответах сервера заголовок Content-Type сообщает клиенту, какой будет тип передаваемого контента. В некоторых случаях браузеры пытаются сами определить MIME тип передаваемого контента, но их реакция может быть неадекватной. Чтобы предотвратить такие ситуации, Вы можете установить в заголовке [X-Content-Type-Options](https://developer.mozilla.org/ru/docs/Web/HTTP/Headers/X-Content-Type-Options) значение nosniff.

Тип MIME - это метка, используемая для идентификации типа данных. Он используется, поэтому программное обеспечение может знать, как обрабатывать данные. Он служит той же цели в Интернете, что расширения файлов делают в Microsoft Windows.

Метаданные - это информация о данных.

Тег <meta> обеспечивает метаданные о HTML документе. Метаданные не будут отображаться на странице, но будут обрабатываться машиной.

Элементы meta обычно используются, чтобы указать описание страницы, ключевые слова, автора документа, дату последней модификации страницы, и другие метаданные.

Тег <meta> всегда идет внутри элемента head.

Метаданные могут быть использованы браузерами (как отображать содержание или перезагружать страницу), поисковыми системами (ключевые слова), и другими веб службами.

### META-теги имеют два возможных атрибута

— <META HTTP-EQUIV="имя" CONTENT="содержимое">  
— <META NAME="имя" CONTENT="содержимое">  
META-теги должны находиться в заголовке HTML-документа между <HEAD> и </HEAD> (особенно это важно для документов, использующих фреймы).

META-теги с атрибутом HTTP-EQUIV эквивалентны HTTP-заголовкам. Обычно они управляют действиями броузеров и могут быть использованы для совершенствования информации, выдаваемой обычными заголовками. Теги такой формы могут дать такой же эффект, что и HTTP-заголовки, и на некоторых серверах автоматически могут быть переведены в настоящие HTTP-заголовки.   
Некоторые генерируемые сервером поля заголовков не могут быть подменены значениями из META-тегов (в частности Date), другие подменяются только при ненормальном статус-коде (<>200). Когда заголовок не понятен, то значение HTTP-заголовка превалирует над значением META-тегов.

Meta Author и Copyright. Этими тегами можно прописать авторство графического и текстового содержимого страницы.

<meta name="Author" content="Авторстраницы">

<meta name="Copyright" content="Владелец авторских прав">

<metaname="Address" content="адрес автора/собственника ">

**Как работает Google Поиск?**

В наше время поисковые системы, в частности Google, напоминают «витрину» Интернета и являются наиболее важным каналом распространения информации в цифровом маркетинге.

**Сканирование**

Сканирование может быть описано, как автоматизированный процесс систематического изучения общедоступных страниц в Интернете. Проще говоря, во время этого процесса Google обнаруживает новые или обновленные страницы и добавляет их в свою базу. Для облегчения работы он использует специальную программу. «Googlebots» (можно встретить альтернативные названия: «боты» или «роботы») посещают список URL-адресов, полученных в процессе прошлого сканирования и дополненных данными карты сайта, которую предоставляют веб-мастера и анализируют их содержание. При обнаружении ссылок на другие страницы во время посещения сайта, боты также добавляют их в свой список и устанавливают систематические связи. Процесс сканирования происходит на регулярной основе в целях выявления изменений, изъятия «мертвых» ссылок и установления новых взаимосвязей. И это при том, что только по данным на сентябрь 2014 года насчитывается около миллиарда веб-сайтов. Можете себе представить сложность такой задачи? Тем ни менее, боты не посещают абсолютно каждый сайт. Чтобы попасть в список проверяемых, веб-ресурс должен быть рассмотрен, как достаточно важный. 

**Индексация**

Индексация — процесс сохранения полученной информации в базе данных в соответствии с различными факторами для последующего извлечения информации. Ключевые слова на странице, их расположение, мета-теги и ссылки представляют особый интерес для индексации Google.  
  
Для того чтобы эффективно хранить информацию о миллиардах страниц в базе данных поисковой системы, Google использует крупные центры обработки данных в Европе, Азии, Северной и Южной Америке. В этих центрах, как было подсчитано, на основе энергопотребления Google в 2010 году, работает около 900,000 серверов.   
  
Основная цель процесса индексации: быстро реагировать на поисковой запрос пользователя. Его как раз мы и будем обсуждать на следующей стадии.

**Обработка**

Когда пользователь вводит запрос, Google производит в базе данных поиск, подходящий под условия и алгоритмически определяет актуальность содержания, что выводит к определенному рейтингу среди найденных сайтов. Логично, что результаты, которые считаются более релевантными для пользователя поисковой системы, намеренно получают более высокий ранг, чем результаты, которые имеют меньше шансов обеспечить адекватный ответ.

Description

Большинство поисковых серверов отображают содержимое поля description при выводе результатов поиска. Если этого тега нет на странице, то поисковый движок просто перечислит первые встречающиеся слова на странице, которые, как правило, оказываются не очень-то и в тему.

Keywords

Этот метатег был предназначен для описания ключевых слов, встречающихся на странице. Но в результате действия людей, желающих попасть в верхние строчки поисковых систем любыми средствами, теперь дискредитирован. Поэтому многие поисковики пропускают этот параметр.

Введение в СSS. Представление блочных элементов.

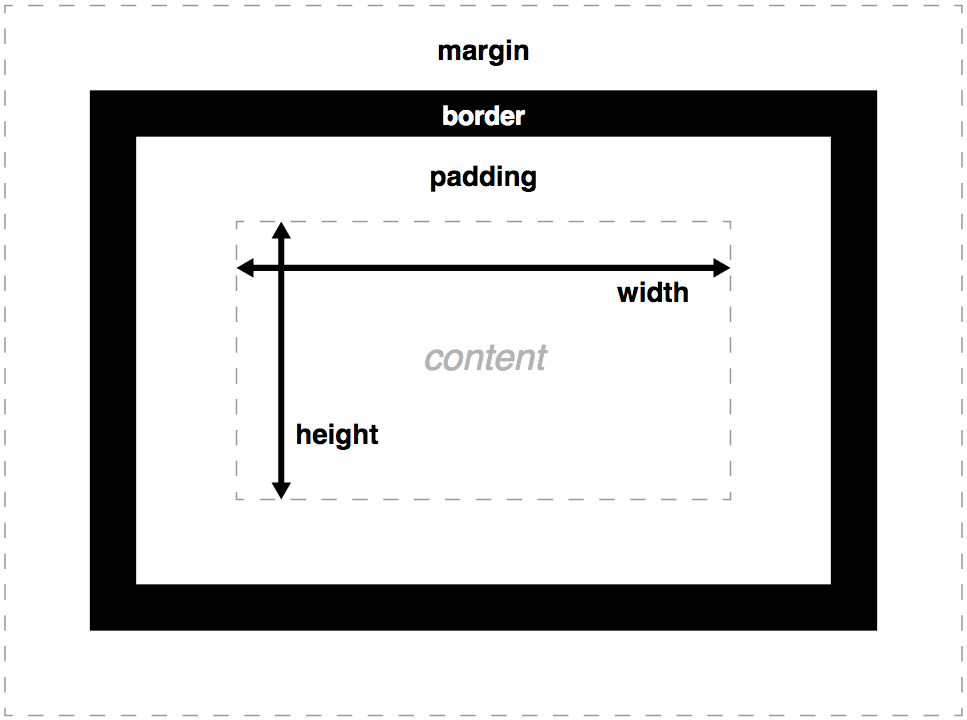
CSS используется для определения стилей ваших документов, в том числе дизайна, верстки и вариаций макета для различных устройств и размеров экрана. Вы можете разместить стили CSS внутри тега <HEAD> документа с встроенной таблицей стилей, или приложить отдельный CSS-файл, который будет определять ваши стили извне. Чтобы привязать внешнюю таблицу стилей к документу, просто добавьте ссылку на таблицу стилей в заголовке <HEAD> документа.

У внешней таблицы стилей есть множество преимуществ. Сохранение стилей отдельно от содержания HTML:

* Помогает избежать дублирования
* Облегчает обслуживание
* Позволяет делать изменения для всего сайта в одном месте

Блочная модель CSS является основой структурирования Веб страниц - каждый элемент представляет собой прямоугольный блок, у которого есть сам контент, внутренний отступ, граница и внешний отступ которые наслаиваются друг на друга, подобно луковице. Поскольку браузер отображает макет веб-страницы, он определяет, какие стили применяются к содержимому каждого окна, насколько велик каждый слой данной луковицы, и как блоки находятся по отношению к друг другу. Прежде чем понять как создавать CSS слои, вы должны понять блочную модель - именно это мы сейчас и рассмотрим.

Каждый элемент внутри документа структурирован как прямоугольник, данный прямоугольник имеет размер и так называемые "слои лука", которые могут быть изменены с использованием определенных свойств CSS. Соответствующие свойства следующие:



**width and height**

width и height свойства задающие длину и высоту **окна содержимого**, это содержимое включает в себя как текстовое содержимое, находящееся внутри блока, так и другие поля, представляющие вложенные дочерние элементы.

**padding**

**Padding** относится к внутреннему отступу блока - между внешним краем окна содержимого и внутренним краем границы. Размер этого слоя можно установить со всех четырех сторон сразу с помощью [padding](https://developer.mozilla.org/ru/docs/Web/CSS/padding) свойства, или отдельно по одной стороне [padding-top](https://developer.mozilla.org/ru/docs/Web/CSS/padding-top), [padding-right](https://developer.mozilla.org/ru/docs/Web/CSS/padding-right), [padding-bottom](https://developer.mozilla.org/ru/docs/Web/CSS/padding-bottom) и [padding-left](https://developer.mozilla.org/ru/docs/Web/CSS/padding-left).

**border**

Слой **border** (граница) располагается между внешним краем padding и внутренним краем margin. По умолчанию размер границы (border) равен 0, что делает ее невидимой, но чтобы она появилась вы можете установить ширину, стиль и цвет границы. Короткая запись свойства [border](https://developer.mozilla.org/ru/docs/Web/CSS/border) позволяет вам установить все основные параметры для всех сторон, например, border: 1px solid black. Эту запись можно разделить на несколько различных свойств, если нужны более специфические стили:

* [border-top](https://developer.mozilla.org/ru/docs/Web/CSS/border-top), [border-right](https://developer.mozilla.org/ru/docs/Web/CSS/border-right), [border-bottom](https://developer.mozilla.org/ru/docs/Web/CSS/border-bottom), [border-left](https://developer.mozilla.org/ru/docs/Web/CSS/border-left): Устанавливает ширину, стиль и цвет для одной стороны границы.
* border-width, border-style, border-color: Устанавливает только ширину, стиль или цвет отдельно, для всех сторон границы.
* Вы также можете установить одно из трех свойств для одной стороны отдельно, используйте для этого [border-top-width](https://developer.mozilla.org/ru/docs/Web/CSS/border-top-width), [border-top-style](https://developer.mozilla.org/ru/docs/Web/CSS/border-top-style), [border-top-color](https://developer.mozilla.org/ru/docs/Web/CSS/border-top-color), и т.д.

**margin**

Margin окружает блок и отталкивает другие блоки внутри разметки страницы. Это похоже на поведение padding; есть короткая запись свойства [margin](https://developer.mozilla.org/ru/docs/Web/CSS/margin), а также специальные свойства для разных сторон [margin-top](https://developer.mozilla.org/ru/docs/Web/CSS/margin-top), [margin-right](https://developer.mozilla.org/ru/docs/Web/CSS/margin-right), [margin-bottom](https://developer.mozilla.org/ru/docs/Web/CSS/margin-bottom) и [margin-left](https://developer.mozilla.org/ru/docs/Web/CSS/margin-left).

Существует 16 стандартных цветов, которые можно записать в значении свойства **color**:

|  |  |  |
| --- | --- | --- |
| **Ключевое слово цвета** | **HEX-код** | **RGB** |
| red | #FF0000 | 255, 0, 0 |
| maroon | #800000 | 128, 0, 0 |
| yellow | #FFFF00 | 255, 255, 0 |
| olive | #808000 | 128, 128, 0 |
| lime | #00FF00 | 0, 255, 0 |
| green | #008000 | 0, 128, 0 |
| aqua | #00FFFF | 0, 255, 255 |
| teal | #008080 | 0, 128, 128 |
| blue | #0000FF | 0, 0, 255 |
| navy | #000080 | 0, 0, 128 |
| fuchsia | #FF00FF | 255, 0, 255 |
| purple | #800080 | 128, 0, 128 |
| white | #FFFFFF | 255, 255, 255 |
| silver | #C0C0C0 | 192, 192, 192 |
| gray | #808080 | 128, 128, 128 |
| black | #000000 | 0, 0, 0 |

Свойство overflow добавляет полосы прокрутки, если содержимое элемента является слишком большим, чтобы поместиться в заданном районе.

Свойство overflow имеет следующие значения:

* visible видимый - По умолчанию. Переполнение не обрезается. Оно отображается за элементом блока
* hidden скрытый - Переполнение обрезается и остальной контент не будет видим
* scroll прокручиваемый - Переполнение обрезается и добавляются полоса прокрутки, чтобы увидеть остальное содержимое

auto автоматический - Если переполнение обрезается, полоса прокрутки должна быть добавлены, чтобы увидеть остальное содержимое

**Обтекание.**

Float

Определяет, по какой стороне будет выравниваться элемент, при этом остальные элементы будут обтекать его с других сторон. Когда значение свойства float равно none, элемент выводится на странице как обычно, при этом допускается, что одна строка обтекающего текста может быть на той же линии, что и сам элемент.

float: left | right | none | inherit

left

Выравнивает элемент по левому краю, а все остальные элементы, вроде текста, обтекают его по правой стороне.

right

Выравнивает элемент по правому краю, а все остальные элементы обтекают его по левой стороне.

none

Обтекание элемента не задается.

inherit

Наследует значение родителя

Цвет текста и фона. Изображения.

Цвет, как таковой, не существует в природе. Он является продуктом обработки человеческим мозгом информации, которая поступает к нам через глаза в виде световой волны. Световые волны имеют различную длину, в зависимости от которой и формируется цвет. В 1676 году Исаак Ньютон провёл эксперимент с расщеплением светового луча при помощи призмы, в результате которого он смог получить 7 чётко выраженных цветов: красный (длина волны 800-650 н/м), оранжевый (640-590 н/м), желтый (580-550 н/м), зелёный (530-490 н/м), голубой (480-460 н/м), синий (450-440 н/м), фиолетовый (430-390 н/м).

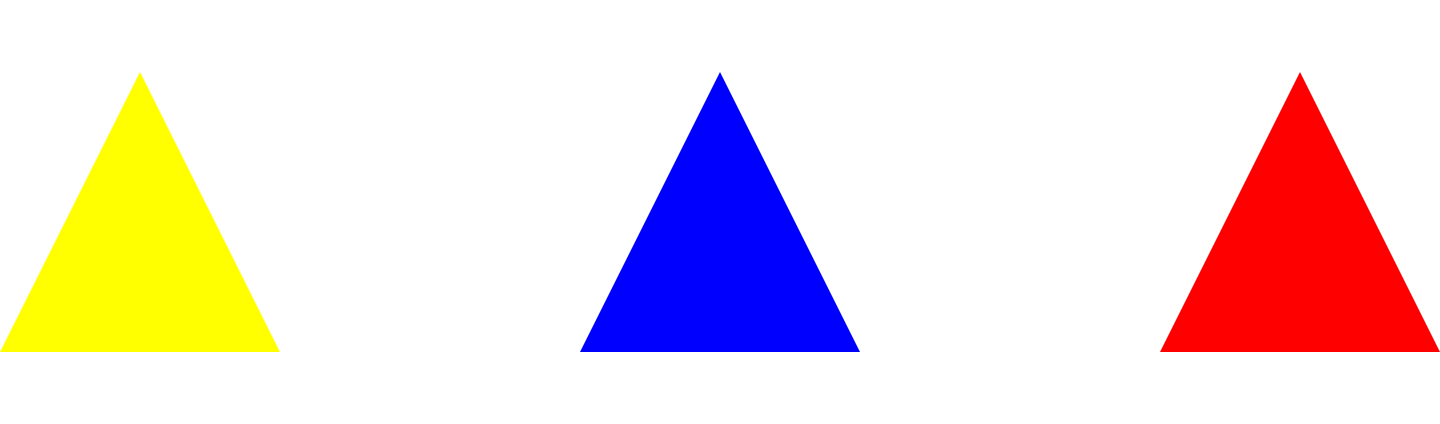
После этого учёный пропустил полученный спектр через собирательную линзу и получил белый цвет.

Своим экспериментом сер Ньютон смог доказать, что белый цвет состоит из всех цветов, отсутствие света ведёт к отсутствию цвета, а чёрный цвет – это полное отсутствие цвета.

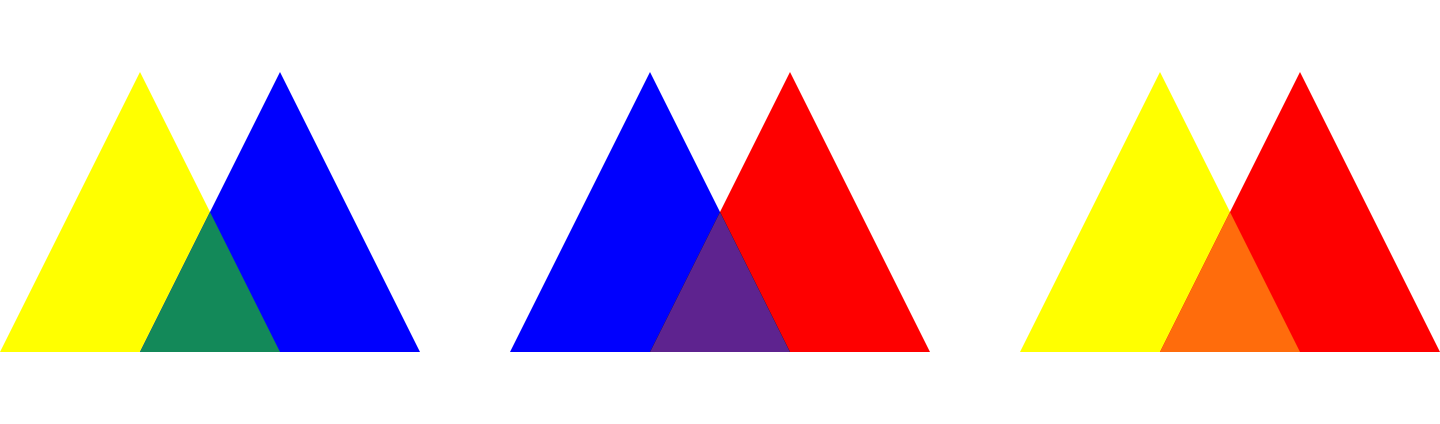
Также он выяснил, что у предметов нет цвета. Когда предмет освещается светом, он поглощает часть световых волн, а остальные отражает (в зависимости от своих физических свойств). Отраженные световые волны являются цветом предмета.

На этом базовый курс физики света окончен.

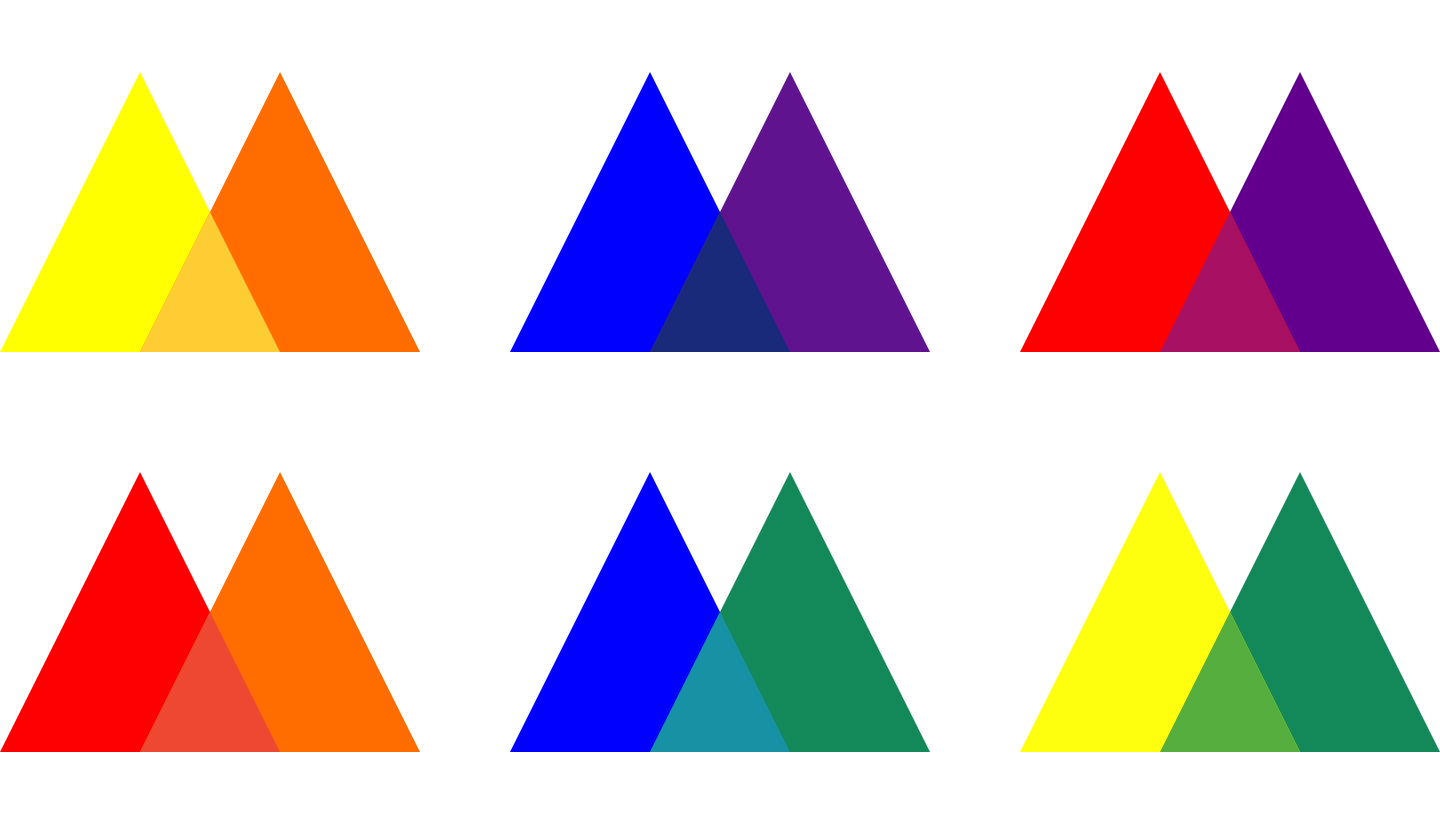
Основу теории цвета Иоханнеса Иттена составляет цветовой круг. Его построение начинается с трёх основных цветов: желтого, синего и красного. Любой цвет можно получить путём смешивания этих трёх основных цветов. Эти цвета также называются первичными.

[](http://miloskiy.com/wp-content/uploads/2016/01/Color-wheel-1-1.png)

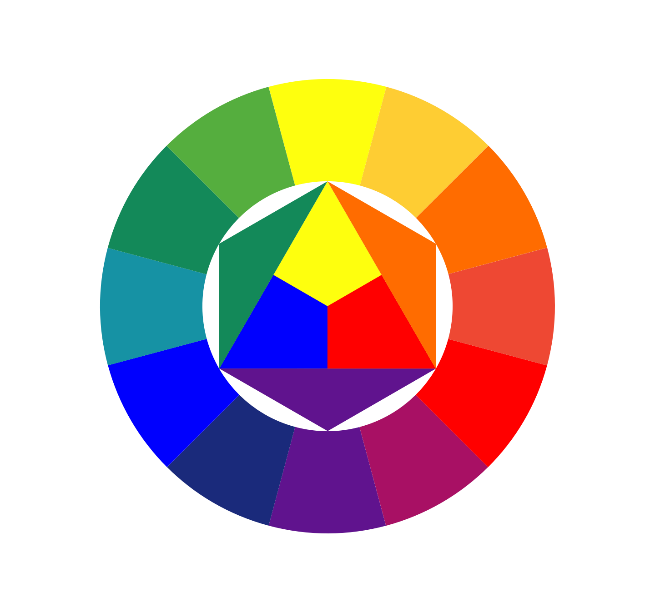
Если равномерно смешать первичные цвета между собой, то мы можем получить зелёный, фиолетовый и оранжевый. Эти цвета называются вторичными. Они размещаются во второй ряд цветового круга.

[](http://miloskiy.com/wp-content/uploads/2016/01/Color-wheel-2-1.png)

Далее, смешивая основные цвета с вторичными, мы получаем третичные цвета: желто-зелёный, желто-оранжевый, красно-фиолетовый, сине-фиолетовый, сине-зелёный, красно-оранжевый.

[](http://miloskiy.com/wp-content/uploads/2016/01/Color-wheel-2-Copy.png)

Первичные и вторичные цвета соприкасаются с внешним цветовым кольцом, в который включаются промежуточные третичные цвета. В результате получаем завершенный цветовой круг.

[](http://miloskiy.com/wp-content/uploads/2016/01/Color-Wheel-4.png)

Как задать цвет шрифта в css?

Для этого вы можете воспользоваться специальным CSS-свойством  ****color****

color: black;

Все цвета шрифта вы можете задавать в различных форматах. Вот наиболее распространённые:

*При помощи кодового значения цвета html.*

Например:

color: black;

*Задание цвета в шестнадцатеричном коде:*

Например:

color: #000000;

Где #000000; — это код черного цвета.

*Задание цвета в формате rgb*

Данный формат представляет собой набор трёх числовых значений от 0 до 255.

Он основывается на использовании трёх цветов, путём смешивания которых получаются все остальные  оттенки

R – (red) – числовое значение красного цвета  
G – (green) – числовое зелёного  
B – (blue) — числовое значение синего

В CSS это выглядит так:

сolor: rgb (0, 155, 0);

Где 0, 155, 0 – это код зелёного цвета.

*Задание цвета при помощи формата rgba*

Здесь принцип использования абсолютно такой же как и в предыдущем случае, но добавляется ещё один параметр – прозрачность. То есть вы можете регулировать не только оттенок, но и его прозрачность

Например:

color: rgba (89, 107, 108, 0.5);

Где 89, 107, 108 – это rgb  код серого цвета, а 0,5 – это уровень прозрачности.

Прозрачность задаётся в виде десятичного значения от 0  до 1, где 0 – цвет совсем не виден, а 1 – цвет максимально непрозрачный.

**.JPEG (JPG)**

Это, пожалуй, самый распространенный графический формат. Обычно он используется для публикации в интернете фотографий и изображений с текстом. JPEG является TrueColor-форматом, то есть может хранить изображения с глубиной цвета 24 бит/пиксель. Данный формат может отображать более 16 млн цветов.

**.GIF**

Формат GIF (Graphics Interchange Format) не радует глубиной цвета (8 бит). Он может хранить сжатые без потери данных изображения в формате не более 256 цветов. Одной из особенностей GIF является поддержка анимации.

**.PNG**

Данный формат был разработан в качестве замены GIF. Расшифровывается PNG как Portable Network Graphics. В отличии от GIF, у PNG есть поддержка градаций прозрачности за счет дополнительного альфа-канала. Внешне файлы в формате PNG практически не отличаются от JPG-изображений. PNG сжимает данные без потерь. Если для вас важна прозрачность, лучше выбирать именно этот формат.

**Изображения на странице**

Вставка изображения на WEB-страницу выполняется одиночным тегом <img src="адрес изображения">. Атрибут src является обязательным в теге <img>. Изображения, вставляемые на WEB-страницу, должны иметь формат поддерживаемый браузером. Стандартные форматы WEB-графики - JPG, GIF и PNG.

Например, вы хотите вставить картинку image.jpg на HTML-страницу. Для этого сохраните файл с изображением в определенной папке. Если сохранить файл в папке, где находиться сам HTML-документ, тогда атрибут src будет иметь вид: src="image.jpg". Также, при вставке изображений надо следить, чтобы имя файла в теге указывалось с соблюдением регистра.

**Работа с фоном**

При помощи CSS фон можно задать не только странице, но и заголовку, и абзацу, и любому другому элементу.

Рассмотрим группу свойств *background*:

* *background-color* - задаетцветфона. По умолчанию не наследуется, но его можно сделать наследуемым, если в качестве значения указать значение *inherit*.
* *background-image* - задает фоновое изображение. Значением свойства является URL графического файла. Формат задания следующий: сначала идет обозначение функции url, а затем в круглых скобках указывается путь к файлу. Путь к файлу указывается относительно таблицы стилей.

Шрифты и текст.

Шрифт и гарнитура. В чем разница?

Многие люди считают термины «шрифт» и «гарнитура» синонимами. На самом деле это два разных понятия. Перед тем, как начать разговор о типографике, давайте разберемся с терминологией. Гарнитуры – это набор типографических знаков и символов. Это буквы, цифры и другие символы, при помощи которых мы составляем слова на бумаге (или экране). Шрифты, с другой стороны, это то, что мы привыкли называть полным набором знаков внутри гарнитуры, часто определенного стиля и размера. Шрифтами также часто называют специальные компьютерные файлы, которые содержат все символы и глифы одной гарнитуры. Когда большинство из нас говорит «шрифты», на самом деле мы подразумеваем гарнитуры, семейства начертаний шрифта (группы шрифтов, имеющих общие стилевые особенности).

Существует несколько классификаций гарнитур. Самая общая – классификация по техническому стилю: антиква, гротески, рукописные, акцидентные и так далее.



С помощью CSS можно определять стиль и вид текста. Аналогично тому, что используется тег <font>, задающий свойства шрифта, но стили обладают большими возможностями и позволяют сократить код HTML.

Свойства шрифта

Изменение начертания шрифта и его размера происходит через свойства CSS, указанные в таблице ниже.

|  |  |  |  |
| --- | --- | --- | --- |
| **Свойство** | **Значение** | **Описание** | **Пример** |
| font-family | имя шрифта | Задает список шрифтов | P {font-family: Arial, serif} |
| font-style | normal  italic  oblique | Нормальный шрифт Курсив Наклонный шрифт | P {font-style: italic} |
| font-variant | normal small-caps | Капитель (особые прописные буквы) | P {font-variant: small-caps} |
| font-weight | normal lighter bold bolder 100–900 | Нормальная жирность Светлое начертание Полужирный Жирный 100 — светлый шрифт, 900 — самый жирный | P {font-weight: bold} |
| font-size | normal pt px  % | нормальный размер  пункты  пикселы  проценты | font-size: normal font-size: 12pt font-size: 12px font-size: 120% |

Свойства текста

Кроме изменения параметров шрифтов, можно управлять и свойствами всего текста.

|  |  |  |  |
| --- | --- | --- | --- |
| **Свойство** | **Значение** | **Описание** | **Пример** |
| line-height | normal  множитель  значение  % | Интерлиньяж (межстрочный интервал) | line-height: normal  line-height: 1.5  line-height: 12px  line-height: 120% |
| text-decoration | none  underline  overline  line-through  blink | Убрать все оформление  Подчеркивание  Линия над текстом  Перечеркивание  Мигание текста | text-decoration: none |
| text-transform | none  capitalize  uppercase  lowercase | Убрать все эффекты  Начинать С Прописных  ВСЕ ПРОПИСНЫЕ  все строчные | text-transform: capitalize |
| text-align | left  right  center  justify | Выравнивание текста | text-align: justify |
| text-indent | значение % | Отступ первой строки | text-indent: 15px;  text-indent: 10% |

Расположение элементов. Каскадные таблицы стилей.

CSS рассматривает макет html-документа как дерево элементов. Уникальный элемент, у которого нет родительского элемента, называется **корневым** элементом. Модуль CSS-позиционирование описывает, как любой из элементов может быть размещен независимо от порядка документа (т.е. извлечен из «потока»).

В CSS блок элемента может быть расположен в соответствии с тремя схемами позиционирования:

#### **1. Нормальный поток**

Нормальный поток включает блочный контекст форматирования (элементы с displayblock, list-item или table), строчный (встроенный) контекст форматирования (элементы с display inline, inline-block или inline-table), и относительное и «липкое» позиционирование элементов уровня блока и строки.

#### **2. Обтекание**

В обтекающей модели блок удаляется из нормального потока и позиционируется влево или вправо. Содержимое обтекает правую сторону элемента с float: left и левую сторону элемента с float: right.

#### **3. Абсолютное позиционирование**

В модели абсолютного позиционирования блок полностью удаляется из нормального потока и ему присваивается позиция относительно содержащего блока. Абсолютное позиционирование реализуется с помощью значений position: absolute; и position: fixed;.

Элементом «вне потока» может быть плавающий, абсолютно позиционированный или корневой элемент.

Свойство position определяет, какой из алгоритмов позиционирования используется для вычисления положения блока.

|  |  |
| --- | --- |
| **position** | |
| Значение: |  |
| static | Блок располагается в соответствии с нормальным потоком. Свойства top, right, bottom и left не применяются. Значение по умолчанию. |
| relative | Положение блока рассчитывается в соответствии с нормальным потоком. Затем блок смещается относительно его нормального положения и во всех случаях, включая элементы таблицы, не влияет на положение любых следующих блоков. Тем не менее, такое смещение может привести к перекрытию блоков, а также к появлению полосы прокрутки в случае переполнения.  Относительно позиционированный блок сохраняет свои размеры, включая разрывы строк и пространство, первоначально зарезервированное для него.  Относительно позиционированный блок создает новый содержащий блок для абсолютно позиционированных потомков.  Влияние position: relative; на элементы таблицы определяется следующим образом: Элементы с table-row-group, table-header-group, table-footer-group иd table-row смещаются относительно их обычной позиции в таблице. Если ячейки таблицы занимают несколько строк, смещаются только ячейки начальной строки. table-column-group, table-column не смещает соответствующий столбец и не оказывает визуального влияния. table-caption and table-cell смещаются относительно своего нормального положения в таблице. Если ячейка таблицы охватывает несколько столбцов или строк, то она смещается целиком. |
| absolute | Положение блока (и, возможно, размер) задается с помощью свойств top, right, bottom и left. Эти свойства определяют явное смещение относительно его содержащего блока. Абсолютно позиционированные блоки полностью удаляется из нормального потока, не влияя на расположение сестринских элементов.  Отступы margin абсолютно позиционированных блоков не схлопываются.  Абсолютно позиционированный блок создает новый содержащий блок для дочерних элементов нормального потока и потомков с position: absolute;.  Содержимое абсолютно позиционированного элемента не может обтекать другие блоки. Абсолютно позиционированный блок могут скрывать содержимое другого блока (или сами могут быть скрыты), в зависимости от значения z-index перекрывающихся блоков. |
| sticky | Положение блока рассчитывается в соответствии с нормальным потоком. Затем блок смещается относительно своего ближайшего предка с прокруткой или окна просмотра, если ни у одного из предков нет прокрутки.  «Липкий» блок может перекрывать другие блоки, а также создавать полосы прокрутки в случае переполнения.  «Липкий» блок сохраняет свои размеры, включая разрывы строк и пространство, первоначально зарезервированное для него.  «Липкий» блок создает новый содержащий блок для абсолютно и относительно позиционированных потомков. |
| fixed | Фиксированное позиционирование аналогично абсолютному позиционированию, с отличием в том, что для содержащим блоком устанавливается окно просмотра. Такой блок полностью удаляется из потока документа и не имеет позиции относительно какой-либо части документа. Фиксированные блоки не перемещаются при прокрутке документа. В этом отношении они похожи на фиксированные фоновые изображения.  При печати фиксированные блоки повторяются на каждой странице, содержащим блоком для них устанавливается область страницы. Блоки с фиксированным положением, которые больше области страницы, обрезаются. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Таблицы.

1. Как создать таблицу

Таблица создаётся при помощи парного тега <table></table>. Данный тег является контейнером для элементов таблицы и все элементы должны находиться внутри него. Например, с помощью данной разметки можно создать таблицу, состоящую из двух столбцов и двух строк:

<table>

<tr><th>текст заголовка</th><th>текст заголовка</th></tr><!--ряд с ячейками заголовков-->

<tr><td>данные</td><td>данные</td></tr><!--ряд с ячейками тела таблицы-->

</table>

HTML

|  |  |
| --- | --- |
| текст заголовка | текст заголовка |
| данные | данные |

РИС.1. ВНЕШНИЙ ВИД ТАБЛИЦЫ БЕЗ ФОРМАТИРОВАНИЯ CSS-СВОЙСТВАМИ

По умолчанию таблица и ячейки не имеют видимых границ. Границы задаются с помощью свойства border:

/\* внешние границы таблицы серого цвета толщиной 1px \*/

table{border: 1px solid grey;}

/\* границы ячеек первого ряда таблицы \*/

th{border: 1px solid grey;}

/\* границыячеектелатаблицы \*/

td{border: 1px solid grey;}

CSS

Промежуткимеждуячейкамитаблицы убираютсяспомощьюсвойства table {border-collapse: collapse;}.

Ширина таблицы по умолчанию равна ширине её внутреннего содержимого. Чтобы установить ширину, нужно задать значение для свойства width:

/\* сделает ширину таблицы равной ширине блока контейнера, в котором она находится \*/

table{width: 100%;}

/\* задаст фиксированную ширину для таблицы \*/

table{width: 600px;}

CSS

Если для ячеек таблицы заданы внутренние отступы и границы, то ширина таблицы будет включать в себя следующие значения:  
padding-left и padding-right, ширина border-left плюс ширина border-rightпоследней ячейки в ряду. Если заданы ширина и границы ячеек, то ширина таблицы будет складываться из ширины ячеек плюс ширина border-left и ширина border-rightпоследней ячейки в ряду.

2. Как создать строки (ряды) таблицы

Строки или ряды таблицы создаются с помощью тега <tr>. Количество горизонтальных строк таблицы определяется количеством парных тегов <tr></tr>.

3. Как сделать ячейку заголовка столбца таблицы

Элемент <th> создаёт заголовок столбца — специальную ячейку, текст в которой выделяется полужирным. Количество ячеек заголовка определяется количеством пар тегов <th></th>. Для элемента доступны атрибуты colspan, rowspan, headers.

<table>

<tr><th>ячейказаголовка</th><th>ячейказаголовка</th></tr>

</table>

HTML

4. Как сделать ячейку тела таблицы

Элемент <td> создаёт ячейки таблицы, внутрь которых помещаются данные таблицы. Парные теги <td></td>, расположенные в одном ряду, определяют количество ячеек в строке таблицы. Количество пар ячеек <td> должно быть равно количеству пар ячеек <th>. Для элемента доступны атрибуты colspan, rowspan, headers.

<table>

<tr><th>ячейка заголовка</th><th>ячейка заголовка</th></tr>

<tr><td>ячейка тела таблицы</td><td>ячейка тела таблицы</td></tr>

</table>

HTML

5. Как добавить подпись (заголовок) к таблице

Элемент <caption> создает подпись таблицы. Добавляется непосредственно после тега <table>, вне строки или ячейки.

<table>

<caption>Перечень продуктов</caption>

<tr>

<th>№ п/п</th>

<th>Наименованиетовара</th>

<th>Ед. изм.</th>

<th>Количество</th>

<th>Цена за ед. изм., руб.</th>

<th>Стоимость, руб.</th>

</tr>

<tr>

<td>1.</td>

<td>Томаты свежие</td><td>кг</td><td>15,20</td><td>69,00</td><td>1048,80</td>

</tr>

<tr>

<td>2.</td>

<td>Огурцы свежие</td><td>кг</td><td>2,50</td><td>48,00</td><td>120,00</td>

</tr>

<tr>

<td colspan="5" style="text-align:right">ИТОГО:</td><td>1168,80</td>

</tr>

</table>

HTML

6. Группирование строк и столбцов таблицы

Элемент <colgroup> создает структурную группу столбцов, выделяя логически однородные ячейки. Группирует один или более столбцов для единого форматирования, позволяя применить стили к столбцам вместо того, чтобы повторять стили для каждой ячейки и для каждой строки. Добавляется непосредственно после тегов <table> и <caption>.

Элемент <col> формирует группы столбцов, которые делят таблицу на разделы, не относящиеся к общей структуре, т.е. не содержащие информацию одного типа. Позволяет задавать свойства столбцов для каждого столбца в пределах элемента <colgroup>. С помощью атрибута style можно изменить основной цвет фона ячеек. Для элемента <col> доступен атрибут span, задающий количество столбцов для объединения.

<table>

<colgroup>

<col span="2" style="background:Khaki"><!-- С помощью этой конструкции задаем цвет фона для первых двух столбцов таблицы-->

<col style="background-color:LightCyan"><!-- Задаем цвет фона для следующего (одного) столбца таблицы-->

</colgroup>

<tr>

<th>№ п/п</th>

<th>Наименование</th>

<th>Цена, руб.</th>

</tr>

<tr>

<td>1</td>

<td>Карандаш цветной</td>

<td>20,00</td>

</tr>

<tr>

<td>2</td>

<td>Линейка 20 см</td>

<td>30,00</td>

</tr>

</table>

HTML

РИС. 2. ВЫДЕЛЕНИЕ СТОЛБЦОВ ТАБЛИЦЫ ДРУГИМ ЦВЕТОМ С ИСПОЛЬЗОВАНИЕМ ТЕГОВ <COL> И <COLGROUP>

7. Группировка разделов таблицы

Элемент <thead> создает группу заголовков для строк таблицы с целью задания единого оформления. Используется в сочетании с элементами <tbody> и <tfoot> для указания каждой части таблицы.

Элемент должен быть использован в следующем порядке: как дочерний элемент <table>, после <caption> и <colgroup>, и перед <tbody>, <tfoot> и <tr>элементами. В пределах одной таблицы можно использовать один раз.

Элемент <tbody> группирует основное содержимое таблицы. Используется в сочетании с элементами <thead> и <tfoot>.

Элемент <tfoot> создает группу строк для представления информации о суммах или итогах, расположенную в нижней части таблицы. Используется в таблице один раз. Располагается после тега <thead>, перед тегами <tbody> и <tr>.

<table>

<thead>

<tr>

<th>№ п/п</th>

<th>Наименование товара</th>

<th>Ед. изм.</th>

<th>Количество</th>

<th>Цена за ед. изм., руб.</th>

<th>Стоимость, руб.</th>

</tr>

</thead>

<tfoot>

<tr>

<td colspan="5" style="text-align:right">ИТОГО:</td><td>1168,80</td>

</tr>

</tfoot>

<tbody>

<tr>

<td>1.</td>

<td>Томатысвежие</td><td>кг</td><td>15,20</td><td>69,00</td><td>1048,80</td>

</tr>

<tr>

<td>2.</td>

<td>Огурцысвежие</td><td>кг</td><td>2,50</td><td>48,00</td><td>120,00</td>

</tr>

</tbody>

</table>

HTML

*Такая группировка строк была заложена в стандарте в расчете на то, что обозреватели при отображении длинных таблиц обеспечат прокрутку строк данных при сохранении надзаголовка и подзаголовка неподвижными, а при их выводе на принтер смогут использовать надзаголовок и подзаголовок в качестве колонтитулов страницы. Однако, современные обозреватели этого не делают и либо просто отображают <thead> и <tfoot> как строки данных, либо, в лучшем случае, просто помещают соответствующие строки в начало и конец таблицы.*

8. Как объединить ячейки таблицы

Атрибуты colspan и rowspan объединяют ячейки таблицы. Атрибут colspan задает количество ячеек, объединенных по горизонтали, а rowspan — по вертикали.

<table>

<tr>

<th>№ п/п</th>

<th>Наименование товара</th>

<th>Ед. изм.</th>

<th>Количество</th>

<th>Цена за ед. изм., руб.</th>

<th>Стоимость, руб.</th>

</tr>

<tr>

<td>1.</td>

<td>Томаты свежие</td><td>кг</td><td>15,20</td><td>69,00</td><td>1048,80</td>

</tr>

<tr>

<td>2.</td>

<td>Огурцы свежие</td><td>кг</td><td>2,50</td><td>48,00</td><td>120,00</td>

</tr>

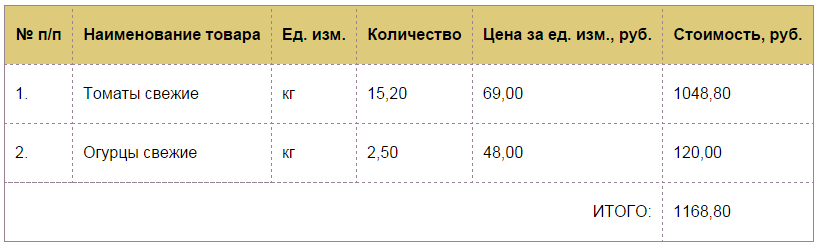
<tr>

<td colspan="5" style="text-align:right">ИТОГО:</td><td>1168,80</td><!-- Задаем количество ячеек по горизонтали для объединения-->

</tr>

</table>

HTML

РИС. 3. ПРИМЕР ОБЪЕДИНЕНИЯ ЯЧЕЕК ТАБЛИЦЫ ПО ГОРИЗОНТАЛИ ПРИ ПОМОЩИ АТРИБУТА COLSPAN

9. Атрибуты элементов таблицы

|  |  |
| --- | --- |
| ТАБЛИЦА 1. АТРИБУТЫ ЭЛЕМЕНТОВ ТАБЛИЦЫ | |
| Атрибут | Описание, принимаемое значение |
| colspan | Количество ячеек в строке для объединения по горизонтали. <td colspan="3"> Возможные значения: число от 1 до 999. |
| headers | Задает список ячеек заголовка, содержащих информацию о заголовке текущей ячейки данных. Предназначен для речевых браузеров. <th id="идентификатор">...</th> <th headers="идентификатор">...</th> Принимаемые значения: список имен ячеек, разделенных пробелами; эти имена должны быть присвоены ячейкам через их атрибут id. |
| rowspan | Количество ячеек в столбце для объединения по вертикали. <td rowspan="2"> Возможные значения: число от 1 до 999. |
| span | Количество колонок, объединяемых для задания единого стиля, по умолчанию равно 1. <col span="2"> Принимаемые значения: любое целое положительное число. |

Оформление таблиц.

### 1. Границы таблицы border

Таблица и ячейки внутри неё по-умолчанию отображаются в браузере без видимых границ. Границы таблицы задаются свойством border:

table{

border-collapse: collapse;/\*убираем пустые промежутки между ячейками\*/

border: 1px solid grey;/\*устанавливаем для таблицы внешнюю границу серого цвета толщиной 1px\*/

}

CSS

Границы ячеек заголовка каждого столбца задаются для элемента th:

th{border: 1px solid grey;}

CSS

Границы ячеек тела таблицы задаются для элемента td:

td{border: 1px solid grey;}

CSS

Толщина рамок соседних ячеек не удваивается, поэтому задать границы для всей таблицы можно следующим способом:

th, td{border: 1px solid grey;}

CSS

Внешнюю границу таблицы можно выделить, задав ей увеличенную ширину:

table{border: 3px solid grey;}

CSS

Границы можно задавать частично:

/\* устанавливаем для таблицы внешнюю границу серого цвета толщиной 3px \*/

table{border-top: 3px solid grey;}

/\* задаём для ячейки тела таблицы границу серого цвета толщиной 1px \*/

td{border-bottom: 1px solid grey;}

CSS

Подробнее о свойстве border вы можете прочитать [здесь](https://html5book.ru/css-border/).

### 2. Как задать ширину и высоту таблицы

По умолчанию ширина и высота таблицы определяется содержимым её ячеек. Если ширина не задана, то она будет равна ширине самого широкого ряда (строки).

Ширина таблицы и столбцов задаётся с помощью свойства width. Если для таблицы задано table {width: 100%;}, то ширина таблицы будет равна ширине блока-контейнера, в котором она находится.

Ширину таблицы и столбцов обычно задают в px или %, например:

table{width: 600px;}

th{width: 20%;}

td:first-child{width: 30%;}

CSS

Высотатаблицы незадается. Высотой рядов таблицы можно управлять, добавив верхний и нижний padding для элементов <td> и <th>. Фиксировать высоту с помощью свойства height не рекомендуется.

th, td{padding: 10px 15px;}

CSS

### 3. Как задать фон таблицы

По умолчанию фон таблицы и ячеек прозрачный. Если страница или блок, содержащие таблицу, имеют фон, то он будет просвечиваться сквозь таблицу. Если фон задан и для таблицы и для ячеек, то в местах наложения фона таблицы и ячеек будет виден фон только ячеек. В качестве фона для таблицы в целом и её ячеек могут выступать:  
заливка [сплошным цветом](https://html5book.ru/css-background/#part1),  
[градиентная заливка](https://html5book.ru/css3-gradient/),  
[фоновое изображение](https://html5book.ru/css-background/#part2).

### 4. Столбцы таблицы

Модель CSS таблиц ориентирована в основном на строки (ряды), формируемые с помощью тега <tr>. На практике бывают случаи, когда необходимо специальное форматирование столбцов, которое возможно следующими способами:

с помощью тега <col> можно задать фон для любого количества столбцов;

с помощью селектора table td:first-child, table td:last-child можно задать стили для первого или последнего столбца таблицы (за исключением первой ячейки заголовка таблицы);

с помощью селектора table td:nth-child(правило отбора столбцов) можно задать стили для любых столбцов таблицы.

Подробнее про тег <col> вы можете прочитать [здесь](https://html5book.ru/html-table/#part5).

Подробнее про CSS-селекторы вы сможете прочитать [здесь](https://html5book.ru/osnovy-css/#part5).

### 5. Как добавить таблице заголовок

Добавить заголовок в таблицу можно с помощью тега <caption>, а с помощью свойства caption-side его можно поместить перед таблицей или под ней. Для горизонтального выравнивания текста заголовка применяется свойство text-align. Наследуется.

|  |  |
| --- | --- |
| caption-side | |
| Значения: |  |
| top | Заголовок таблицы располагается над таблицей. Значение по умолчанию. |
| bottom | Располагает заголовок под таблицей. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

<table>

<caption>Таблица № 1</caption>

<tr>

<th>Company</th>

<th>Q1</th>

<th>Q2</th>

<th>Q3</th>

<th>Q4</th>

</tr>

...

</table>

HTML

caption{

caption-side: bottom;

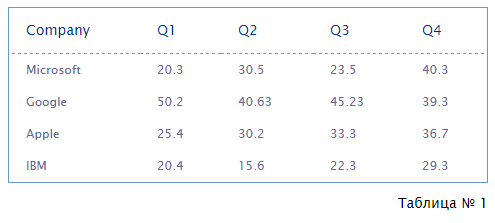
text-align: right;

padding: 10px 0;

font-size: 14px;

}

CSS

РИС. 2. ПРИМЕР ОТОБРАЖЕНИЯ ЗАГОЛОВКА ПОД ТАБЛИЦЕЙ

### 6. Как убрать промежуток между рамками ячеек

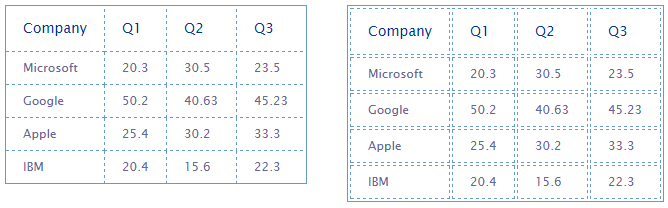
Рамки ячеек таблицы по умолчанию разделены небольшим промежутком. Если задать для таблицы border-collapse: collapse, то промежуток уберётся. Свойство наследуется.

|  |  |
| --- | --- |
| border-collapse | |
| Значения: |  |
| separate | Рамки ячеек располагаются раздельно. |
| collapse | Рамки ячеек сливаются в одну, а промежутки между рамками убираются. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Синтаксис

table{border-collapse: collapse;}

CSS

РИС. 3. ПРИМЕР ТАБЛИЦ СО СЛИВАЮЩИМИСЯ И РАЗДЕЛЬНЫМИ РАМКАМИ ЯЧЕЕК

### 7. Как увеличить промежуток между рамками ячеек

С помощью свойства border-spacing можно менять расстояние между рамками ячеек. Данное свойство применяется к таблице в целом. Наследуется.

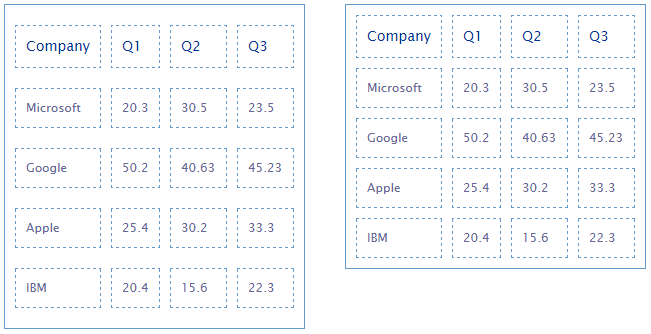
|  |  |
| --- | --- |
| border-spacing | |
| Значения: |  |
| <длина><длина> | Добавляет промежутки между рамками как по вертикали, так и по горизонтали. Если заданы две длины, то первая всегда определяет горизонтальный промежуток, а вторая — вертикальный. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Синтаксис

table{border-collapse: separate;border-spacing: 10px 20px;}

table{border-collapse: separate;border-spacing: 10px;}

CSS

РИС. 4. ПРИМЕР ТАБЛИЦ С УВЕЛИЧЕННЫМИ ПРОМЕЖУТКАМИ МЕЖДУ РАМКАМИ ЯЧЕЕК

### 8. Как скрыть пустые ячейки таблицы

Свойство empty-cells скрывает или показывает пустые ячейки. Действует только на ячейки, которые не содержат какой-либо контент. Если для ячейки задан фон, а для таблицы задано table {border-collapse: collapse;}, то ячейка не будет скрыта. Наследуется.

|  |  |
| --- | --- |
| empty-cells | |
| Значения: |  |
| show | Рамка и фон пустой ячейки будут отрисовываться так же, как для ячейки таблицы, имеющей содержимое. |
| hide | Если все ячейки строки пусты, то вся строка отображается так, если бы было задано значение display: none. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

<table>

<tr>

<th>Company</th>

<th>Q1</th>

<th>Q2</th>

<th>Q3</th>

</tr>

<tr>

<td>Microsoft</td>

<td>20.3</td>

<td>30.5</td>

<td></td>

</tr>

<tr>

<td>Google</td>

<td>50.2</td>

<td>40.63</td>

<td>45.23</td>

</tr>

</table>

HTML

table{

border: 1px solid #69c;

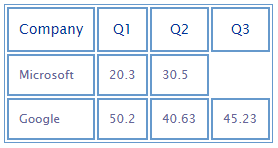
border-collapse: separate;

empty-cells: hide;

}

th, td{border: 2px solid #69c;}

CSS

РИС. 5. ПРИМЕР СКРЫТИЯ ПУСТОЙ ЯЧЕЙКИ ТАБЛИЦЫ

### 9. Компоновка макета таблицы с помощью свойства table-layout

Компоновка макета таблицы определяется одним из двух подходов: фиксированный макет или автоматический макет. Под компоновкой в данном случае подразумевается как распределяется ширина таблицы между шириной ячеек. Свойство не наследуется.

|  |  |
| --- | --- |
| table-layout | |
| Значения: |  |
| auto | Значение по умолчанию. Ширина макета таблицы определяется шириной её содержимого с учетом значений свойств padding-left, padding-right, border-left width плюс одна ширина border-right последней ячейки в ряду, или заданной шириной ячеек и толщиной рамки. Если ширина ячеек не задана явно, они могут быть разной ширины. |
| fixed | Свойство сработает только в том случае, если для таблицы задана ширина. Ширина ячеек будет одинаковой, а содержимое ячеек, которое не помещается в ячейку, будет наползать под содержимое соседней ячейки. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Псевдоклассы. Указатель мыши. Оформление списков. Устаревшие методы оформления.

Псевдоклассы определяют динамическое состояние элементов, которое изменяется с помощью действий пользователя, а также положение в дереве документа. Примером такого состояния служит текстовая ссылка, которая меняет свой цвет при наведении на неё курсора мыши. При использовании псевдоклассов браузер не перегружает текущий документ, поэтому с помощью псевдоклассов можно получить разные динамические эффекты на странице.

## Псевдоклассы, определяющие состояние элементов

К этой группе относятся псевдоклассы, которые распознают текущее состояние элемента и применяют стиль только для этого состояния.

:active

Происходит при активации пользователем элемента. Например, ссылка становится активной, если навести на неё курсор и щёлкнуть мышкой. Несмотря на то, что активным может стать практически любой элемент веб-страницы, псевдокласс :active используется преимущественно для ссылок.

:link

Применяется к непосещенным ссылкам, т. е. таким ссылкам, на которые пользователь ещё не нажимал. Браузер некоторое время сохраняет историю посещений, поэтому ссылка может быть помечена как посещенная хотя бы потому, что по ней был зафиксирован переход ранее.

:focus

Применяется к элементу при получении им фокуса. Например, для текстового поля формы получение фокуса означает, что курсор установлен в поле, и с помощью клавиатуры можно вводить в него текст

:hover

Псевдокласс :hover активизируется, когда курсор мыши находится в пределах элемента, но щелчка по нему не происходит.

:visited

Данный псевдокласс применяется к посещённым ссылкам. Обычно такая ссылка меняет свой цвет по умолчанию на фиолетовый, но с помощью стилей цвет и другие параметры можно задать самостоятельно

**Оформление списков.**

1. Тип маркера списка list-style-type

Свойство изменяет типа маркера или удаляет маркер для маркированного и нумерованного списков. Наследуется.

|  |  |
| --- | --- |
| list-style-type | |
| Значения: |  |
| disc | Значение по умолчанию. В качестве маркера элементов списка выступает закрашенный кружок. |
| armenian | Традиционная армянская нумерация. |
| circle | В качестве маркера выступает незакрашенный кружок. |
| cjk-ideographic | Идеографическая нумерация. |
| decimal | 1, 2, 3, 4, 5, … |
| decimal-leading-zero | 01, 02, 03, 04, 05, … |
| georgian | Традиционная грузинская нумерация. |
| hebrew | Традиционная еврейская нумерация. |
| hiragana | Японская нумерация: a, i, u, e, o, … |
| hiragana-iroha | Японская нумерация: i, ro, ha, ni, ho, … |
| katakana | Японская нумерация: A, I, U, E, O, … |
| katakana-iroha | Японская нумерация: I, RO, HA, NI, HO, … |
| lower-alpha | a, b, c, d, e, … |
| lower-greek | Строчные символы греческого алфавита. |
| lower-latin | a, b, c, d, e, … |
| lower-roman | i, ii, iii, iv, v, … |
| none | Маркер отсутствует. |
| square | В качестве маркера выступает закрашенный или незакрашенный квадрат. |
| upper-alpha | A, B, C, D, E, … |
| upper-latin | A, B, C, D, E, … |
| upper-roman | I, II, III, IV, V, … |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Синтаксис

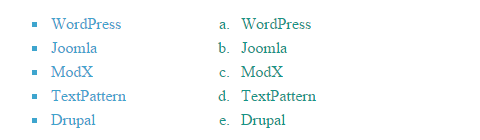
ul{list-style-type: none;}

ul{list-style-type: square;}

ol{list-style-type: none;}

ol{list-style-type: lower-alpha;}

CSS

РИС. 1. ПРИМЕР ОФОРМЛЕНИЯ МАРКИРОВАННОГО И НУМЕРОВАННОГО СПИСКОВ

2. Изображения для элементов списка list-style-image

В качестве маркера элементов списка можно использовать изображения и градиентые заливки. Наследуется.

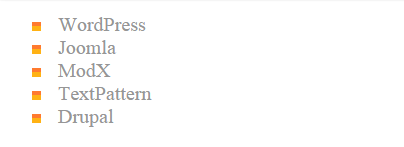
|  |  |
| --- | --- |
| list-style-image | |
| Значения: |  |
| url(url) | Путь к изображению. |
| none | значение по умолчанию, означает отсутствие изображения. Также убирает изображение для элемента из группы элементов с установленным изображением-маркером. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Синтаксис

ul{list-style-image:url("images/romb.png");}

ul{list-style-image:linear-gradient(#FF7A2F 0, #FF7A2F 50%, #FFB214 50%);}

CSS

РИС. 2. ОФОРМЛЕНИЕ МАРКИРОВАННОГО СПИСКА С ПОМОЩЬЮ ИЗОБРАЖЕНИЯРИС. 3. ОФОРМЛЕНИЕ МАРКИРОВАННОГО СПИСКА С ПОМОЩЬЮ ГРАДИЕНТА

3. Местоположение маркера списка list-style-position

Данное свойство предоставляет возможность располагать маркер вне или внутри содержимого элемента списка. Наследуется.

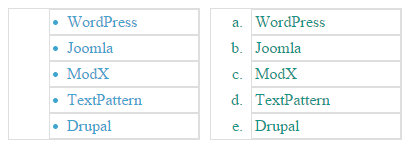
|  |  |
| --- | --- |
| list-style-position | |
| Значения: |  |
| outside | Значение по умолчанию. Маркер располагается вне блока с текстом. |
| inside | Маркер списка изображается в одном блоке с текстом. Последующие строки текста будут располагаться под значком маркера, т.е. маркер будет обтекаться текстом. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Синтаксис

ul{list-style-position: inside;}

ol{list-style-position: outside;}

CSS

РИС. 4. ПРИМЕР РАСПОЛОЖЕНИЯ МАРКЕРА ВНУТРИ И СНАРУЖИ БЛОКА СПИСКА

4. Краткая форма задания стилей списка

Можно объединить все три свойства форматирования списка в одно с помощью list-style. Значения свойств могут быть расположены в произвольном порядке, а часть значений может быть опущена. Если присутствует одно значение, то другие свойства примут значения браузера по умолчанию.

Основы JavaScript.

Что такое JavaScript?

Изначально JavaScript был создан, чтобы «сделать веб-страницы живыми».

Программы на этом языке называются скриптами. Они могут встраиваться в HTML и выполняться автоматически при загрузке веб-страницы.

Скрипты распространяются и выполняются, как простой текст. Им не нужна специальная подготовка или компиляция для запуска.

Это отличает JavaScript от другого языка – Java.

**Как подключить JavaScript?**

Итак, рассматриваем два варианта подключения скрипта.

**Вариант 1. Подключение внутри html-файла.**

Делается это с помощью конструкции:

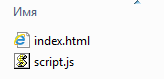
<script type="text/javascript"> … Код javascript … </script>

**Вариант 2. Подключение внешнего файла скрипта с кодом.**

Если javascript-код находится во внешнем файле, то способ его подключение будет несколько другим.

Пусть такой файл называется script.js.

В итоге, имеем два файла index.html и script.js, которые находятся в одной папке.

[](https://webgyry.info/wp-content/uploads/2014/10/21-10-2014-13-04-21.png)

Как теперь подключить скрипт к html файлу?

В этом случае нужно воспользоваться тэгом

<script src="путь\_к\_файлу\_скрипта/script.js"></script>

Об адресах и как их правильно выставлять информация здесь.

Так как файл скрипта script.js находится в той же папке, что и файл index.html, то html-код может быть следующим:

<!doctype html><html><head><meta charset="utf-8"><title>Документ без названия</title><script src="script.js"></script></head><body> Содержимое html-файла </body></html>

Как правило, скрипты подключаются в области <head> документа.

Способ 1. Скачать в папку и подключить файл

Наиболее распространённый способ, если сайт могут просматривать локально, без подключения сети Интернет (неужели такие еще есть?).

Идем [по адресу](http://api.jquery.com/) и жмем большую желто-оранжевую кнопку **Download jQuery**. Переходим на страницу где выбираем версию jQuery. Желательно выбирать последнюю из доступных. Обозначение *compressed* – означает, что библиотека минимизирована, т.е. занимаем минимум места, но, к сожалению, читать исходных код трудно. Качаем! После сохранения подключаем файл библиотеки.

*<!DOCTYPEhtml>*

<htmllang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<!--Подключаембиблиотеку-->

<script src="js/jquery-2.2.3.min.js"></script>

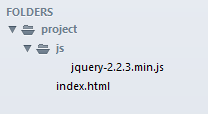
</head>

<body>

</body>

</html>

Так выглядит структура проекта, библиотеку jQuery скачали в папку js



библиотека лежит в папке js

В некоторых случаях, библиотеку подключают перед закрывающимся тегом body, что связанно с порядком обработки html страницы браузером. Поскольку браузер вычитывает строки последовательно, то при подключении jQuery в конце файла вначале браузер отобразит сайт, а затем уже подключит динамику. При медленном соединении, такой подход обеспечивает увеличение скорости загрузки сайта, и уже потом работу слайдеров и остального. Код при таком подключении выглядит:

*<!DOCTYPEhtml>*

<htmllang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

</head>

<body>

<!--Подключаембиблиотеку-->

<script src="js/jquery-2.2.3.min.js"></script>

</body>

</html>

Способ 2. Подключаем напрямую с CDN

Данный способ хорош тем, что библиотека подключается с сайта и не валяется на жестком диске. Особенно это актуально при большом количестве маленьких проектов и для обучения.

Для подключения рекомендую использоваться подключение от Google Developers, обычно я нахожу данную страницу по ссылке [jquery google](https://developers.google.com/speed/libraries/). Наша задача скопировать строку:

<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.2.0/jquery.min.js"></script>

и подключить ее в файл. Выглядит это так:

*<!DOCTYPEhtml>*

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<!--Подключаембиблиотеку-->

<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.2.0/jquery.min.js"></script>

</head>

<body>

</body>

</html>

Второй способ подключения, перед закрывающимся тегом body:

*<!DOCTYPE html>*

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

</head>

<body>

<!--Подключаембиблиотеку-->

<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.2.0/jquery.min.js"></script>

</body>

</html>

Изменение класса является одним из наиболее часто используемых действий в скриптах.

Когда-то давно в JavaScript существовало ограничение: зарезервированное слово типа "class" не могло быть свойством объекта. Это ограничение сейчас отсутствует, но в то время было невозможно иметь свойство elem.class.

Поэтому для классов было введено схожее свойство "className": elem.className соответствует атрибуту "class".

Например:

<body class="main page">

<script>

alert(document.body.className);// main page

</script>

</body>

Если мы присваиваем что-то elem.className, то это заменяет всю строку с классами. Иногда это то, что нам нужно, но часто мы хотим добавить/удалить один класс.

Для этого есть другое свойство: elem.classList.

elem.classList – это специальный объект с методами для добавления/удаления одного класса.

Например:

<body class="main page">

<script>

// добавлениекласса

document.body.classList.add('article');

alert(document.body.className);// main page article

</script>

</body>

Так что мы можем работать как со строкой полного класса, используя className, так и с отдельными классами, используя classList. Выбираем тот вариант, который нам удобнее.

Методы classList:

* elem.classList.add/remove("class") – добавить/удалить класс.
* elem.classList.toggle("class") – добавить класс, если его нет, иначе удалить.
* elem.classList.contains("class") – проверка наличия класса, возвращает true/false.

Кроме того, classList является перебираемым, поэтому можно перечислить все классы при помощи for..of:

<body class="main page">

<script>

for(let name of document.body.classList){

alert(name);// main, затем page

}

</script>

</body>

[Element style](https://learn.javascript.ru/styles-and-classes#element-style)

Свойство elem.style – это объект, который соответствует тому, что написано в атрибуте "style". Установка стиля elem.style.width="100px" работает так же, как наличие в атрибуте style строки width:100px.

Для свойства из нескольких слов используется camelCase:

background-color => elem.style.backgroundColor

z-index => elem.style.zIndex

border-left-width => elem.style.borderLeftWidth

document.body.style.backgroundColor = prompt('background color?','green');

Сброс стилей

Иногда нам нужно добавить свойство стиля, а потом, позже, убрать его.

Например, чтобы скрыть элемент, мы можем задать elem.style.display = "none".

Затем мы можем удалить свойство style.display, чтобы вернуться к первоначальному состоянию. Вместо delete elem.style.display мы должны присвоить ему пустую строку: elem.style.display = "".

// если мы запустим этот код, <body> "мигнёт"

document.body.style.display ="none";// скрыть

setTimeout(()=> document.body.style.display ="",1000);// возвраткнормальномусостоянию

Если мы установим в style.display пустую строку, то браузер применит CSS-классы и встроенные стили, как если бы такого свойства style.display вообще не было.

Обычно мы используем style.\* для присвоения индивидуальных свойств стиля. Нельзя установить список стилей как, например, div.style="color: red; width: 100px", потому что div.style – это объект, и он доступен только для чтения.

Для задания нескольких стилей в одной строке используется специальное свойство style.cssText:

<div id="div">Button</div>

<script>

// можем даже устанавливать специальные флаги для стилей, например, "important"

div.style.cssText=`color: red !important;

background-color: yellow;

width: 100px;

text-align: center;

`;

alert(div.style.cssText);

</script>

Это свойство редко используется, потому что такое присваивание удаляет все существующие стили: оно не добавляет, а заменяет их. Можно ненароком удалить что-то нужное. Но его можно использовать, к примеру, для новых элементов, когда мы точно знаем, что не удалим существующий стиль.

То же самое можно сделать установкой атрибута: div.setAttribute('style', 'color: red...').

Следите за единицами измерения

Не забудьте добавить к значениям единицы измерения.

Например, мы должны устанавливать 10px, а не просто 10 в свойство elem.style.top. Иначе это не сработает:

<body>

<script>

// неработает!

document.body.style.margin =20;

alert(document.body.style.margin);// '' (пустаястрока, присваиваниеигнорируется)

// сейчас добавим единицу измерения (px) - и заработает

document.body.style.margin ='20px';

alert(document.body.style.margin);// 20px

alert(document.body.style.marginTop);// 20px

alert(document.body.style.marginLeft);// 20px

</script>

</body>

Пожалуйста, обратите внимание, браузер «распаковывает» свойство style.margin в последних строках и выводит style.marginLeft и style.marginTop из него.

JavaScript показать и скрыть элемент с текстом

При разработке любых сайтов или интернет магазинов, требуется **динамически скрывать и показывать элементы**. Современные сайты должны выглядеть динамично, иначе цена разработки сайта будет очень низкой, т.к. ни кто много за статичный сайт платить не будет!

Обычно показываются, и скрываются **html тег div**, но не обязательно, можно манипулировать любыми элементами.   
**Изменение видимости элемента** достигается за счет изменения css-свойства display. Если display = ‘block’, то элемент будет видим, если ‘none’, то скрыт.

Сейчас напишем небольшой примерчик, в нем при нажатии на ссылку мы будем показывать блок, а при повторном нажатии скрывать.

Вот исходный код примера, с подробным описанием:

<html>

    <head>

        <title>JavaScript показать и скрыть элемент с текстом</head>

    <body>

        <script type="text/javascript">

            /\*\*

            \* Функция Скрывает/Показывает блок

            \*\*/

            function showHide(element\_id) {

                //Если элемент с id-шником element\_id существует

                if (document.getElementById(element\_id)) {

                    //Записываем ссылку на элемент в переменную obj

                    var obj = document.getElementById(element\_id);

                    //Если css-свойство display не block, то:

                    if (obj.style.display != "block") {

                        obj.style.display = "block"; //Показываем элемент

                    }

                    else obj.style.display = "none"; //Скрываем элемент

                }

                //Если элемент с id-шником element\_id не найден, то выводим сообщение

                else alert("Элемент с id: " + element\_id + " не найден!");

            }

        </script>

<!-- При клике запускаем функцию showHide, и передаем параметр

        id-шник элемента который нужно показать/скрыть -->

        <a href="javascript:void(0)" onclick="showHide('block\_id')">Скрыть/Показатьэлемент</a><br/><br/>

        <div id="block\_id" style="display: none;">

            Этот текст будет скрыт.

        </div>

    </body>

</html>

Во время обычной отправки формы <form> браузер собирает значения её полей, делает из них строку и составляет тело GET/POST-запроса для посылки на сервер.

При отправке данных через XMLHttpRequest, это нужно делать самим, в JS-коде. Большинство проблем и вопросов здесь связано с непониманием, где и какое кодирование нужно осуществлять.

## [Кодировка urlencoded](https://learn.javascript.ru/xhr-forms" \l "kodirovka-urlencoded)

Основной способ кодировки запросов – это urlencoded, то есть – стандартное кодирование URL.

Например, форма:

<form action="/submit"method="GET">

<input name="name"value="Ivan">

<input name="surname"value="Ivanov">

</form>

Здесь есть два поля: name=Ivan и surname=Ivanov.

Браузер перечисляет такие пары «имя=значение» через символ амперсанда & и, так как метод GET, итоговый запрос выглядит как /submit?name=Ivan&surname=Ivanov.

Все символы, кроме английских букв, цифр и - \_ . ! ~ \* ' ( ) заменяются на их цифровой код в UTF-8 со знаком %.

Например, пробел заменяется на %20, символ / на %2F, русские буквы кодируются двумя байтами в UTF-8, поэтому, к примеру, Ц заменится на %D0%A6.

Например, форма:

<form action="/submit"method="GET">

<input name="name"value="Виктор">

<input name="surname"value="Цой">

</form>

Будетотправленатак: /submit?name=%D0%92%D0%B8%D0%BA%D1%82%D0%BE%D1%80&surname=%D0%A6%D0%BE%D0%B9.

в JavaScript есть функция [encodeURIComponent](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/encodeURIComponent) для получения такой кодировки «вручную»:

Эта кодировка используется в основном для метода GET, то есть для передачи параметра в строке запроса. По стандарту строка запроса не может содержать произвольные Unicode-символы, поэтому они кодируются как показано выше.

GET-запрос

Формируя XMLHttpRequest, мы должны формировать запрос «руками», кодируя поля функцией encodeURIComponent.

Например, для посылки GET-запроса с параметрами name и surname, аналогично форме выше, их необходимо закодировать так:

// Передаём name и surname в параметрах запроса

var xhr =new XMLHttpRequest();

var params ='name='+ encodeURIComponent(name)+

'&surname='+ encodeURIComponent(surname);

xhr.open("GET",'/submit?'+ params,true);

xhr.onreadystatechange =...;

xhr.send();

POST с urlencoded

В методе POST параметры передаются не в URL, а в теле запроса. Оно указывается в вызове send(body).

В стандартных HTTP-формах для метода POST доступны [три кодировки](https://html.spec.whatwg.org/multipage/forms.html#submit-body), задаваемые через атрибут enctype:

* application/x-www-form-urlencoded
* multipart/form-data
* text-plain

В зависимости от enctype браузер кодирует данные соответствующим способом перед отправкой на сервер.

В случае с XMLHttpRequest мы, вообще говоря, не обязаны использовать ни один из этих способов. Главное, чтобы сервер наш запрос понял. Но обычно проще всего выбрать какой-то из стандартных.

В частности, при POST обязателен заголовок Content-Type, содержащий кодировку. Это указание для сервера – как обрабатывать (раскодировать) пришедший запрос.

Для примера отправим запрос в кодировке application/x-www-form-urlencoded:

var xhr =new XMLHttpRequest();

var body ='name='+ encodeURIComponent(name)+

'&surname='+ encodeURIComponent(surname);

xhr.open("POST",'/submit',true);

xhr.setRequestHeader('Content-Type','application/x-www-form-urlencoded');

xhr.onreadystatechange =...;

xhr.send(body);

## [Кодировка multipart/form-data](https://learn.javascript.ru/xhr-forms" \l "kodirovka-multipart-form-data)

Кодировка urlencoded за счёт замены символов на %код может сильно «раздуть» общий объём пересылаемых данных. Поэтому для пересылки файлов используется другая кодировка: [multipart/form-data](http://ru.wikipedia.org/wiki/Multipart_form-data).

В этой кодировке поля пересылаются одно за другим, через строку-разделитель.

Чтобы использовать этот способ, нужно указать его в атрибуте enctype и метод должен быть POST:

<form action="/submit"method="POST"enctype="multipart/form-data">

<input name="name"value="Виктор">

<input name="surname"value="Цой">

</form>

Форма при такой кодировке будет выглядеть примерно так:

...Заголовки...

Content-Type: multipart/form-data; boundary=RaNdOmDeLiMiTeR

--RaNdOmDeLiMiTeR

Content-Disposition: form-data; name="name"

Виктор

--RaNdOmDeLiMiTeR

Content-Disposition: form-data; name="surname"

Цой

--RaNdOmDeLiMiTeR--

…То есть, поля передаются одно за другим, значения не кодируются, а чтобы было чётко понятно, какое значение где – поля разделены случайно сгенерированной строкой, которую называют «boundary» (англ. граница), в примере выше это RaNdOmDeLiMiTeR:

Сервер видит заголовок Content-Type: multipart/form-data, читает из него границу и раскодирует поля формы.

Такой способ используется в первую очередь при пересылке файлов, так перекодировка мегабайтов через urlencoded существенно загрузила бы браузер. Да и объём данных после неё сильно вырос бы.

Однако, никто не мешает использовать эту кодировку всегда для POST запросов. ДляGETдоступнатолькоurlencoded.

POSTсmultipart/form-data

Сделать POST-запрос в кодировке multipart/form-data можно и через XMLHttpRequest.

Достаточно указать в заголовке Content-Type кодировку и границу, и далее сформировать тело запроса, удовлетворяющее требованиям кодировки.

Пример кода для того же запроса, что и раньше, теперь в кодировке multipart/form-data:

var data ={

name:'Виктор',

surname:'Цой'

};

var boundary = String(Math.random()).slice(2);

var boundaryMiddle ='--'+ boundary +'\r\n';

var boundaryLast ='--'+ boundary +'--\r\n'

var body =['\r\n'];

for(var key in data){

// добавлениеполя

body.push('Content-Disposition: form-data; name="'+ key +'"\r\n\r\n'+ data[key]+'\r\n');

}

body = body.join(boundaryMiddle)+ boundaryLast;

// Тело запроса готово, отправляем

var xhr =new XMLHttpRequest();

xhr.open('POST','/submit',true);

xhr.setRequestHeader('Content-Type','multipart/form-data; boundary='+ boundary);

xhr.onreadystatechange =function(){

if(this.readyState !=4)return;

alert(this.responseText );

}

xhr.send(body);

Тело запроса будет иметь вид, описанный выше, то есть поля через разделитель.

FormData

Современные браузеры, исключая IE9- (впрочем, есть полифил), поддерживают встроенный объект [FormData](https://developer.mozilla.org/en-US/docs/DOM/XMLHttpRequest/FormData/Using_FormData_Objects), который кодирует формы для отправки на сервер.

Это очень удобно. Например:

<formname="person">

<input name="name"value="Виктор">

<input name="surname"value="Цой">

</form>

<script>

// создать объект для формы

var formData =new FormData(document.forms.person);

// добавить к пересылке ещё пару ключ - значение

formData.append("patronym","Робертович");

// отослать

var xhr =new XMLHttpRequest();

xhr.open("POST","/url");

xhr.send(formData);

</script>

Этот код отправит на сервер форму с полями name, surname и patronym.

Интерфейс:

* Конструктор new FormData([form]) вызывается либо без аргументов, либо с DOM-элементом формы.
* Метод formData.append(name, value) добавляет данные к форме.

Объект formData можно сразу отсылать, интеграция FormData с XMLHttpRequest встроена в браузер. Кодировка при этом будет multipart/form-data.

Что такое JS-framework и зачем он нужен?

JS-framework’и — это инструменты для построения динамических веб/мобильных/настольных приложений на языке Javascript. Как и к любым другим инструментам,  
разработчики прибегают к использованию js-фреймворков там, где невозможно/очень сложно/очень долго выполнять задачу обычными средствами. В подавляющем большинстве случаев, фреймворки используются для написания, так называемых, Single Page Applications. Т.е. все, что проиходит на сайте, проиходит на одной страничке, без прямого перехода с нее.

С их помощью можно разрабатывать как полноценные сайты, так и функциональные модули (различные онлайн-инструменты). Конечно, полноценные фреймворки лучше подходят для первой задачи, а для второй рекомендуется использовать более легковесные фреймворки или библиотеки.

Популярные JSфреймворки

1. React

Эффективная и гибкая декларативная библиотека JavaScript для сборки UI от команды Facebook. Она позволяет без усилий создавать интерактивный пользовательский интерфейс.

2. Vue

Vue.js, прогрессивный фреймворк для сборки пользовательских интерфейсов,включает доступную корневую библиотеку, которая в первую очередь решает задачи уровня представления, и экосистему дополнительных библиотек, позволяющую создавать сложные и объемные одностраничные приложения.

3.Angular

Angular — фреймворк от компании Google, сочетаются декларативные шаблоны, внедрение зависимости, двустороннее связывание данных и лучшие практики решения проблем разработки. Эта платформа позволяет собирать приложения для веб, мобильных устройств и настольных ПК. В ней предлагается самый удобный и понятный для начинающих интерфейс командной строки (CLI) и даже консоль (Console) — клиент с графическим интерфейсом.

Адаптивность и кроссбраузерность.

Для начала разберем что же такое гибкий сайт (адаптивный web-дизайн). Это дизайн страниц, который обеспечивает корректное и удобное для мобильного юзера представление сайта. Адаптивный web-сайт автоматически подгоняется под размеры окна браузера. Цель веб-дизайна, гибкого под электронные гаджеты – универсальность представления сайта. Это означает, что нет необходимости настраивать сайт под отдельный вид гаджета, адаптивный веб-ресурс правильно отображается на любых мобильных девайсах, лэптопах и даже на телевизоре. При этом серфинг ресурса будет равносильно удобен на любом устройстве и пользователю не нужно увеличивать выбранные блоки, чтобы не кликнуть не на ту кнопку.

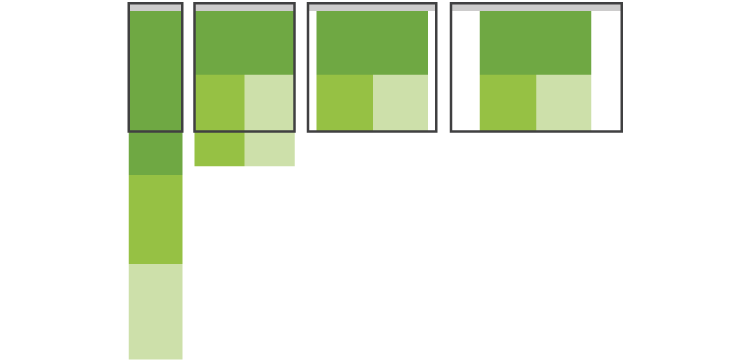
В отличие от ресурса, который адаптирован и имеет один URL, мобильная модификация создается на поддомене. Мобильная версия сайта подразумевает максимальное упрощение страницы, избавление части контента и функциональных возможностей. Наряду с более быстрой загрузкой (адаптивная версия веб-страниц в этом проигрывает и требует дополнительной оптимизации скорости загрузки), создание мобильной версии ресурса дорогостоящее и стоит столько же, как и разработка мобильного приложения. Все недостатки гибкого сайта являются достоинствами мобильного типа сайта, а его недостатки – преимуществами адаптива.

Так или иначе, это принципиально различные версии веб-ресурса, они требуют разных технических действий, поэтому сама страница по-разному выглядит при загрузке с мобильного устройства.

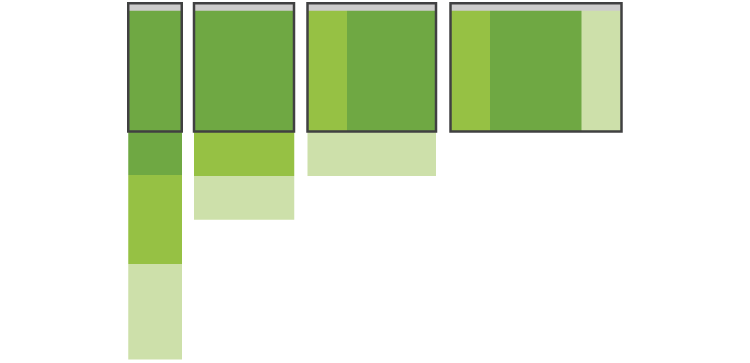
Типы адаптивного дизайна

Формирование адаптивности выполняется на стадии разработки дизайна. Поэтому важно выбрать оптимальный вид адаптивного дизайна из пяти основных:

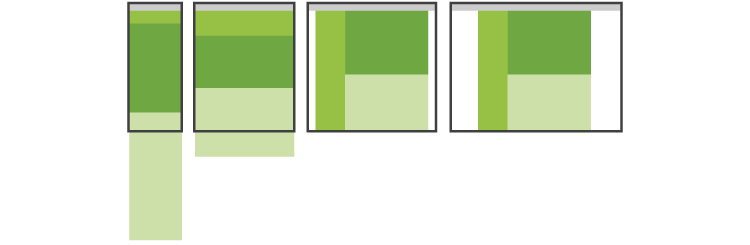
1. Резиновый макет. Самый распространенный и простой вариант адаптировать дизайн сайта. Он подразумевает сжатие блоков по ширине до размера мобильного гаджета. А те блоки, которые невозможно сжать, настраиваются друг под другом;



2. Выстраивание блоков. Данный тип оптимален для ресурсов, обладающих многоколончатой структурой. При сужении страницы, блоки переносятся вниз макета;



3. Переключение макетов. Тип заключен в разработке макетов под различные разрешения экрана. Не самый популярный и очень трудоемкий способ из-за сложности стандартизации дизайна за счет разнообразия девайсов;



4. Масштабирование изображений и текстов. Самый простой тип адаптивности для простых ресурсов. Масштабируется не весь сайт, а его отдельные блоки/части;



Перечисленные типы макетов не считаются оптимальным выбором. Для  каждого веб-проекта важно подобрать более подходящий под возможности сайта вариант.

**Медиазапросы**

Наряду с типами носителей в CSS3 включена поддержка различных технических параметров устройств, на основе которых требуется загружать те или иные стили. К примеру, можно определить смартфон с максимальным разрешением 640 пикселов и для него установить одни стилевые свойства, а для остальных устройств другие. Также можно выявить различные характеристики вроде наличия монохромного экрана, ориентации (портретная или альбомная) и др. Все характеристики легко комбинируются, поэтому допустимо задать стиль только для устройств в альбомной ориентации с заданным разрешением экрана.

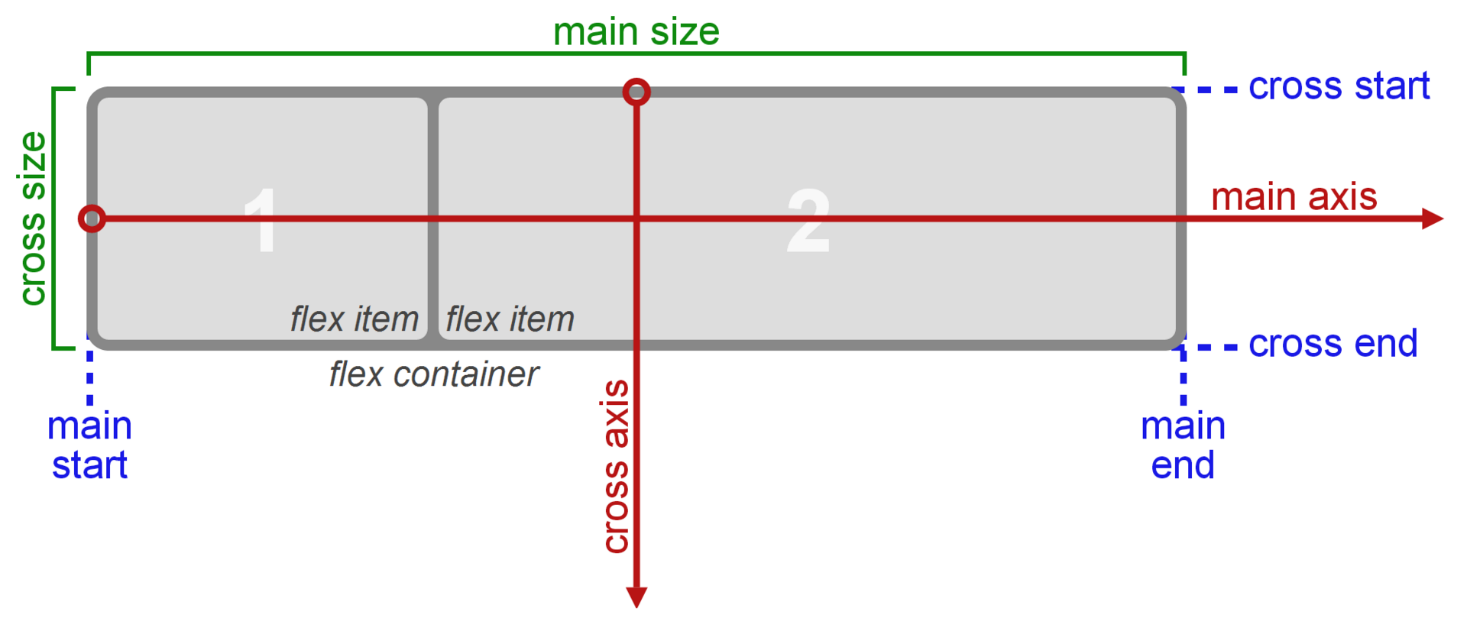
Возможности медиа-запросов не ограничиваются выявлением мобильных устройств, с их помощью можно создавать адаптивный макет. Такой макет подстраивается под разрешение монитора и окна браузера, меняя при необходимости ширину макета, число колонок, размеры изображений и текста. Медиа-запросы ограничивают ширину макета и при достижении этого значения (к примеру за счёт уменьшения окна или при просмотре на устройстве с указанным размером) уже применяется другой стиль.

Синтаксис

Все запросы начинаются с правила [@media](http://htmlbook.ru/css/media), после чего следует условие, в котором используются типы носителей, логические операторы и медиа-функции. Типы носителей перечислены в табл. 1.

|  |  |
| --- | --- |
| **Тип** | **Описание** |
| all | Все типы. Это значение используется по умолчанию. |
| braille | Устройства, основанные на системе Брайля, которые предназначены для чтения слепыми людьми. |
| embossed | Принтеры, использующие для печати систему Брайля. |
| handheld | Смартфоны и аналогичные им аппараты. |
| print | Принтеры и другие печатающие устройства. |
| projection | Проекторы. |
| screen | Экран монитора. |
| speech | Речевые синтезаторы, а также программы для воспроизведения текста вслух. Сюда, например, можно отнести речевые браузеры. |
| tty | Устройства с фиксированным размером символов (телетайпы, терминалы, устройства с ограничениями дисплея). |
| tv | Телевизоры. |

### 1. Основные понятия



Для описания модуля Flexbox используется определенный набор терминов. Значение flex-flow и режим записи определяют соответствие этих терминов физическим направлениям: верх / право / низ / лево, осям: вертикальная / горизонтальная и размерам: ширина / высота.

**Главная ось (main axis)** — ось, вдоль которой выкладываются flex-элементы. Она простирается в основном измерении.

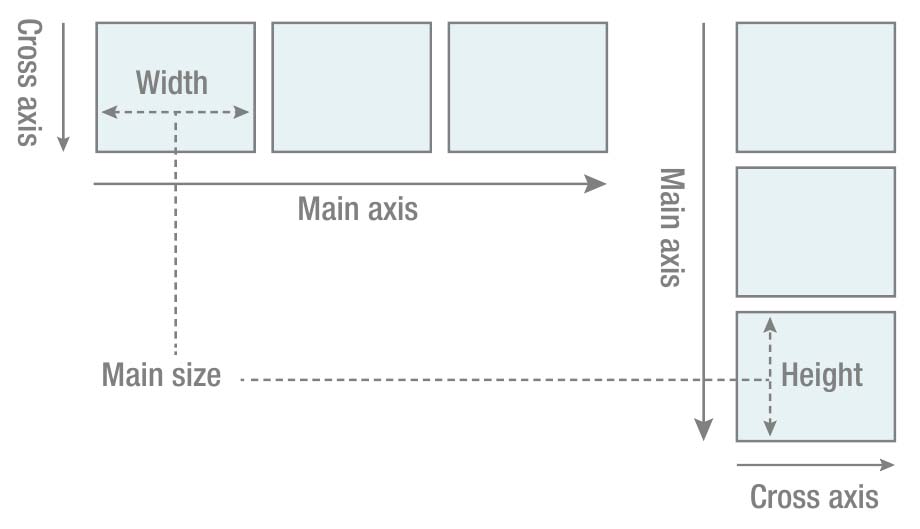
**Main start и main end** — линии, которые определяют начальную и конечную стороны flex-контейнера, относительно которых выкладываются flex-элементы (начиная с main start по направлению к main end).

**Основной размер (main size**) — ширина или высота flex-контейнера или flex-элементов, в зависимости от того, что из них находится в основном измерении, определяют основной размер flex-контейнера или flex-элемента.

**Поперечная ось (cross axis)** — ось, перпендикулярная главной оси. Она простирается в поперечном измерении.

**Cross start и cross end** — линии, которые определяют начальную и конечную стороны поперечной оси, относительно которых выкладываются flex-элементы.

**Поперечный размер (cross size)** — ширина или высота flex-контейнера или flex-элементов, в зависимости от того, что находится в поперечном измерении, являются их поперечным размером.

РИС. 2.

### 2. Flex-контейнер

Flex-контейнер устанавливает новый гибкий контекст форматирования для его содержимого. Flex-контейнер не является блочным контейнером, поэтому для дочерних элементов не работают такие CSS-свойства, как float, clear, vertical-align. Также, на flex-контейнер не оказывают влияние свойства column-\*, создающие колонки в тексте и псевдоэлементы ::first-line и ::first-letter.

Модель flexbox-разметки связана с определенным значением CSS-свойства displayродительского html-элемента, содержащего внутри себя дочерние блоки. Для управления элементами с помощью этой модели нужно установить свойство display следующим образом:

.flex-container{

/\*генерирует flex-контейнер уровня блока\*/

display: -webkit-flex;

display: flex;

}

.flex-container{

/\*генерирует flex-контейнеруровнястроки\*/

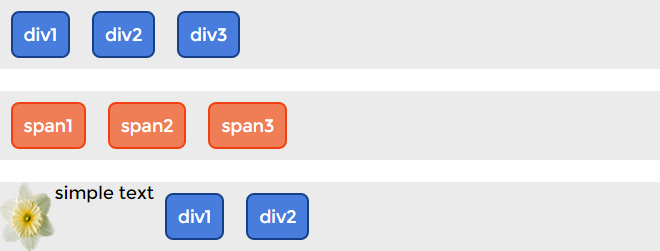
display: -webkit-inline-flex;

display: inline-flex;

}

CSS

После установки данных значений свойства каждый дочерний элемент автоматически становится flex-элементом, выстраиваясь в один ряд (вдоль главной оси). При этом блочные и строчные дочерние элементы ведут себя одинаково, т.е. ширина блоков равна ширине их содержимого с учетом внутренних полей и рамок элемента.



Если родительский блок содержит текст или изображения без оберток, они становятся анонимными flex-элементами. Текст выравнивается по верхнему краю блока-контейнера, а высота изображения становится равной высоте блока, т.е. оно деформируется.

### 3. Flex-элементы

Flex-элементы — блоки, представляющие содержимое flex-контейнера в потоке. Flex-контейнер устанавливает новый контекст форматирования для своего содержимого, который обуславливает следующие особенности:

* Для flex-элементов блокируется их значение свойства display. Значение display: inline-block; и display: table-cell; вычисляетсяв display: block;.
* Пустое пространство между элементами исчезает: оно не становится своим собственным flex-элементом, даже если межэлементный текст обернут в анонимный flex-элемент. Для содержимого анонимного flex-элемента невозможно задать собственные стили, но оно будет наследовать их (например, параметры шрифта) от flex-контейнера.
* Абсолютно позиционированный flex-элемент не участвует в компоновке гибкого макета.
* Поля margin соседних flex-элементов не схлопываются.
* Процентные значения margin и padding вычисляются от внутреннего размера содержащего их блока.
* margin: auto; расширяются, поглощая дополнительное пространство в соответствующем измерении. Их можно использовать для выравнивания или раздвигания смежных flex-элементов.
* Автоматический минимальный размер flex-элементов по умолчанию является минимальным размером его содержимого, то есть min-width: auto;. Для контейнеров с прокруткой автоматический минимальный размер обычно равен нулю.

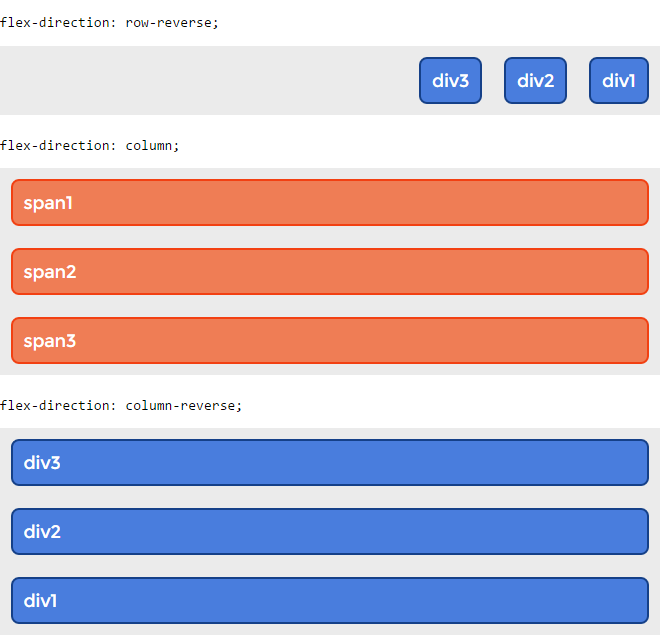
### 4. Порядок отображения flex-элементов и ориентация

Содержимое flex-контейнера можно разложить в любом направлении и в любом порядке (переупорядочение flex-элементов внутри контейнера влияет только на визуальный рендеринг).

#### **4.1. Направление главной оси: flex-direction**

Свойство относится к flex-контейнеру. Управляет направлением главной оси, вдоль которой укладываются flex-элементы, в соответствии с текущим режимом записи. Не наследуется.

|  |  |
| --- | --- |
| **flex-direction** | |
| Значения: |  |
| row | Значение по умолчанию, слева направо (в rtl справа налево). Flex-элементы выкладываются в строку. Начало (main-start) и конец (main-end) направления главной оси соответствуют началу (inline-start) и концу (inline-end) инлайн-оси (inline-axis). |
| row-reverse | Направление справа налево (в rtl слева направо). Flex-элементы выкладываются в строку относительно правого края контейнера (в rtl — левого). |
| column | Направление сверху вниз. Flex-элементы выкладываются в колонку. |
| column-reverse | Колонка с элементами в обратном порядке, снизу вверх. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |



**Синтаксис**

.flex-container{

display: -webkit-flex;

-webkit-flex-direction: row-reverse;

display: flex;

flex-direction: row-reverse;

}

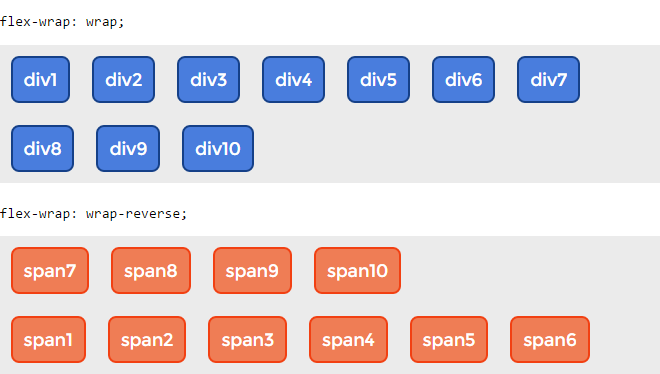
CSS

#### **4.2. Управление многострочностью flex-контейнера: flex-wrap**

Свойство определяет, будет ли flex-контейнер однострочным или многострочным, а также задает направление поперечной оси, определяющее направление укладки новых линий flex-контейнера.

По умолчанию flex-элементы укладываются в одну строку, вдоль главной оси. При переполнении они будут выходить за пределы ограничивающей рамки flex-контейнера. Свойство не наследуется.

|  |  |
| --- | --- |
| **flex-wrap** | |
| Значения: |  |
| nowrap | Значение по умолчанию. Flex-элементы не переносятся, а располагаются в одну линию слева направо (в rtl справа налево). |
| wrap | Flex-элементы переносятся, располагаясь в несколько горизонтальных рядов (если не помещаются в один ряд) в направлении слева направо (в rtl справа налево). |
| wrap-reverse | Flex-элементы переносятся на новые линии, располагаясь в обратном порядке слева-направо, при этом перенос происходит снизу вверх. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |



**Синтаксис**

.flex-container{

display: -webkit-flex;

-webkit-flex-wrap: wrap;

display: flex;

flex-wrap: wrap;

}

CSS

#### **4.3. Краткая запись направления и многострочности: flex-flow**

Свойство позволяет определить направления главной и поперечной осей, а также возможность переноса flex-элементов при необходимости на несколько строк. Представляет собой сокращённую запись свойств flex-direction и flex-wrap. Значение по умолчанию flex-flow: row nowrap;. свойство не наследуется.

|  |  |
| --- | --- |
| **flex-flow** | |
| Значения: |  |
| направление | Указывает направление главной оси. Значение по умолчанию row. |
| многострочность | Задаёт многострочность поперечной оси. Значение по умолчанию nowrap. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

**Синтаксис**

.flex-container{

display: -webkit-flex;

-webkit-flex-flow: row wrap;

display: flex;

flex-flow: row wrap;

}

CSS

#### **4.4. Порядок отображения flex-элементов: order**

Свойство определяет порядок, в котором flex-элементы отображаются и располагаются внутри flex-контейнера. Применяется к flex-элементам. Свойство не наследуется.

Первоначально все flex-элементы имеют order: 0;. При указании значения от -1 для элемента он перемещается в начало сроки, значение 1 — в конец. Если несколько flex-элементов имеют одинаковое значение order, они будут отображаться в соответствии с исходным порядком.

|  |  |
| --- | --- |
| **order** | |
| Значения: |  |
| число | Свойство задается целым числом, отвечающим за порядок отображения flex-элементов. Значение по умолчанию 0. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

**Синтаксис**

.flex-container{

display: -webkit-flex;

display: flex;

}

.flex-item{

-webkit-order: 1;

order: 1;

}

CSS



### 5. Гибкость flex-элементов

Определяющим аспектом гибкого макета является возможность «сгибать» flex-элементы, изменяя их ширину / высоту (в зависимости от того, какой размер находится на главной оси), чтобы заполнить доступное пространство в основном измерении. Это делается с помощью свойства flex. Flex-контейнер распределяет свободное пространство между своими дочерними элементами (пропорционально их коэффициенту flex-grow) для заполнения контейнера или сжимает их (пропорционально их коэффициенту flex-shrink), чтобы предотвратить переполнение.

Flex-элемент будет полностью «негибок», если его значения flex-grow и flex-shrinkравны нулю, и «гибкий» в противном случае.

#### **5.1. Задание гибких размеров одним свойством: flex**

Свойство является сокращённой записью свойств flex-grow, flex-shrink и flex-basis. Значение по умолчанию: flex: 0 1 auto;. Можно указывать как одно, так и все три значения свойств. Свойство не наследуется.

W3C рекомендует использовать сокращённую запись, так как она правильно сбрасывает любые неуказанные компоненты, чтобы подстроиться под типичное использование.

|  |  |
| --- | --- |
| **flex** | |
| Значения: |  |
| коэффициент растяжения | Коэффициент увеличения ширины flex-элемента относительно других flex-элементов. |
| коэффициент сужения | Коэффициент уменьшения ширины flex-элемента относительно других flex-элементов. |
| базовая ширина | Базовая ширина flex-элемента. |
| auto | Эквивалентно flex: 1 1 auto;. |
| none | Эквивалентно flex: 0 0 auto;. |
| initial | Устанавливает значение свойства в значение по умолчанию. Эквивалентно flex: 0 1 auto;. |
| inherit | Наследует значение свойства от родительского элемента. |

**Синтаксис**

.flex-container{

display: -webkit-flex;

display: flex;

}

.flex-item{

-webkit-flex: 3 1 100px;

-ms-flex: 3 1 100px;

flex: 3 1 100px;

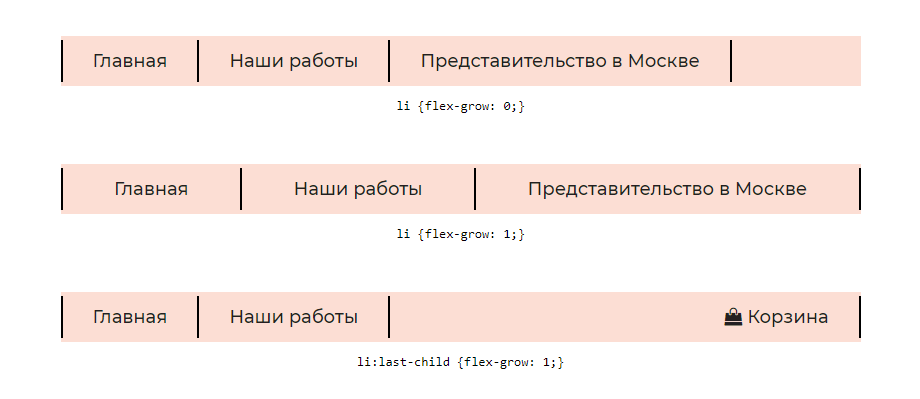
}

CSS

#### **5.2. Коэффициент роста: flex-grow**

Свойство определяет коэффициент роста одного flex-элемента относительно других flex-элементов в flex-контейнере при распределении положительного свободного пространства. Если сумма значений flex-grow flex-элементов в строке меньше 1, они занимают менее 100% свободного пространства. Свойство не наследуется.

|  |  |
| --- | --- |
| **flex-grow** | |
| Значения: |  |
| число | Положительное целое или дробное число, устанавливающее коэффициент роста flex-элемента. Значение по умолчанию 0. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

[](https://html5book.ru/goto/https:/codepen.io/html5book/pen/KEqOJB)

**Синтаксис**

.flex-container{

display: -webkit-flex;

display: flex;

}

.flex-item{

-webkit-flex-grow: 3;

flex-grow: 3;

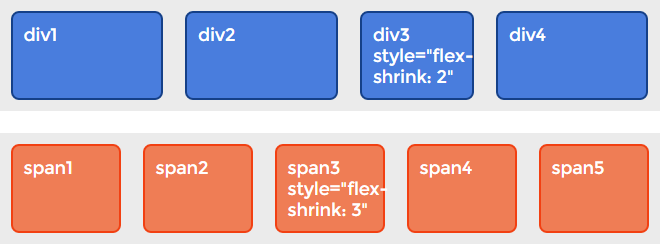
}

CSS

#### **5.3. Коэффициентсжатия: flex-shrink**

Свойство указывает коэффициент сжатия flex-элемента относительно других flex-элементов при распределении отрицательного свободного пространства. Умножается на базовый размер flex-basis. Отрицательное пространство распределяется пропорционально тому, насколько элемент может сжаться, поэтому, например, маленький flex-элемент не уменьшится до нуля, пока не будет заметно уменьшен flex-элемент большего размера. Свойство не наследуется.

|  |  |
| --- | --- |
| **flex-shrink** | |
| Значения: |  |
| число | Положительное целое или дробное число, устанавливающее коэффициент уменьшения flex-элемента. Значение по умолчанию 1. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |



**Синтаксис**

.flex-container{

display: -webkit-flex;

display: flex;

}

.flex-item{

-webkit-flex-shrink: 3;

flex-shrink: 3;

}

CSS

#### **5.4. Базовыйразмер: flex-basis**

Свойство устанавливает начальный основной размер flex-элемента до распределения свободного пространства в соответствии с коэффициентами гибкости. Для всех значений, кроме auto и content, базовый гибкий размер определяется так же, как width в горизонтальных режимах записи. Процентные значения определяются относительно размера flex-контейнера, а если размер не задан, используемым значением для flex-basis являются размеры его содержимого. Не наследуется.

|  |  |
| --- | --- |
| **flex-basis** | |
| Значения: |  |
| auto | Значение по умолчанию. Элемент получает базовый размер, соответствующий размеру его содержимого (если он не задан явно). |
| content | Определяет базовый размер в зависимости от содержимого flex-элемента. |
| длина | Базовый размер определяется так же, как для ширины и высоты. Задается в единицах длины. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

**Синтаксис**

.flex-container{

display: -webkit-flex;

display: flex;

}

.flex-item{

-webkit-flex-basis: 100px;

flex-basis: 100px;

}

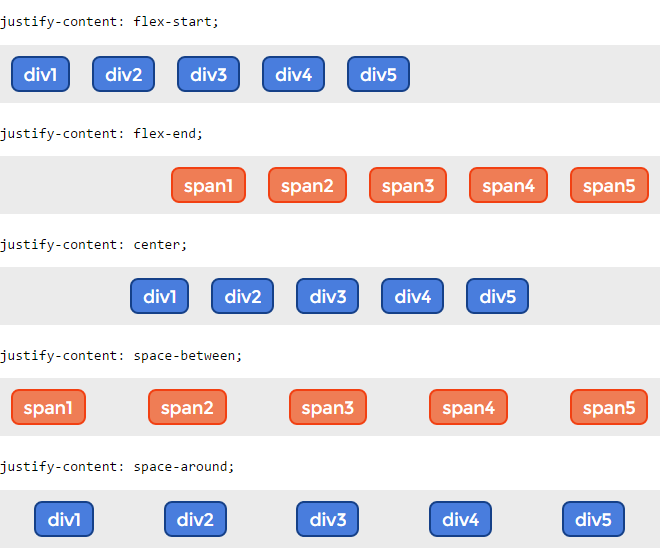
CSS

### 6. Выравнивание

#### **6.1. Выравнивание по главной оси: justify-content**

Свойство выравнивает flex-элементы по главной оси flex-контейнера, распределяя свободное пространство, незанятое flex-элементами. Когда элемент преобразуется в flex-контейнер, flex-элементы по умолчанию сгруппированы вместе (если для них не заданы поля margin). Промежутки добавляются после расчета значений margin и flex-grow. Если какие-либо элементы имеют ненулевое значение flex-grow или margin: auto;, свойство не будет оказывать влияния. Свойство не наследуется.

|  |  |
| --- | --- |
| **justify-content** | |
| Значения: |  |
| flex-start | Значение по умолчанию. Flex-элементы выкладываются в направлении, идущем от начальной линии flex-контейнера. |
| flex-end | Flex-элементы выкладываются в направлении, идущем от конечной линии flex-контейнера. |
| center | Flex-элементы выравниваются по центру flex-контейнера. |
| space-between | Flex-элементы равномерно распределяются по линии. Первый flex-элемент помещается вровень с краем начальной линии, последний flex-элемент — вровень с краем конечной линии, а остальные flex-элементы на линии распределяются так, чтобы расстояние между любыми двумя соседними элементами было одинаковым. Если оставшееся свободное пространство отрицательно или в строке присутствует только один flex-элемент, это значение идентично параметру flex-start. |
| space-around | Flex-элементы на линии распределяются так, чтобы расстояние между любыми двумя смежными flex-элементами было одинаковым, а расстояние между первым / последним flex-элементами и краями flex-контейнера составляло половину от расстояния между flex-элементами. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |



**Синтаксис**

.flex-container{

display: -webkit-flex;

-webkit-justify-content: flex-start;

display: flex;

justify-content: flex-start;

}

CSS

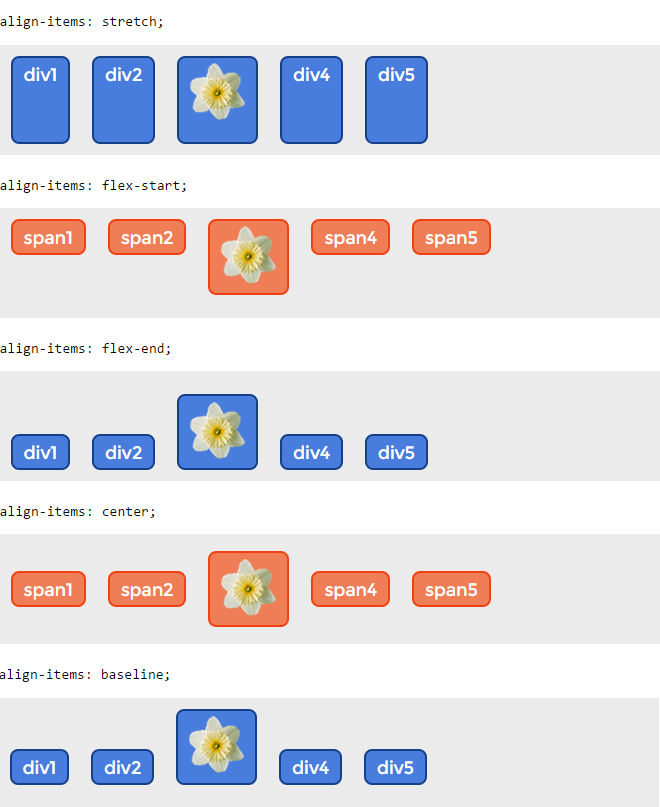
#### **6.2. Выравнивание по поперечной оси: align-items и align-self**

Flex-элементы можно выравнивать по поперечной оси текущей строки flex-контейнера. align-items устанавливает выравнивание для всех элементов flex-контейнера, включая анонимные flex-элементы. align-self позволяет переопределить это выравнивание для отдельных flex-элементов. Если любое из поперечных margin flex-элемента имеет значение auto, align-self не имеет никакого влияния.

##### **6.2.1. Align-items**

Свойство выравнивает flex-элементы, в том числе и анонимные flex-элементы по поперечной оси. Не наследуется.

|  |  |
| --- | --- |
| **align-items** | |
| Значения: |  |
| flex-start | Верхний край flex-элемента помещается вплотную с flex-линией (или на расстоянии, с учетом заданных полей margin и рамок border элемента), проходящей через начало поперечной оси. |
| flex-end | Нижний край flex-элемента помещается вплотную с flex-линией (или на расстоянии, с учетом заданных полей margin и рамок border элемента), проходящей через конец поперечной оси. |
| center | Поля flex-элемента центрируется по поперечной оси в пределах flex-линии. |
| baseline | Базовые линии всех flex-элементов, участвующих в выравнивании, совпадают. |
| stretch | Если поперечный размер flex-элемента вычисляется как auto и ни одно из поперечных значений margin не равно auto, элемент растягивается. Значение по умолчанию. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |



**Синтаксис**

.flex-container{

display: -webkit-flex;

-webkit-align-items: flex-start;

display: flex;

align-items: flex-start;

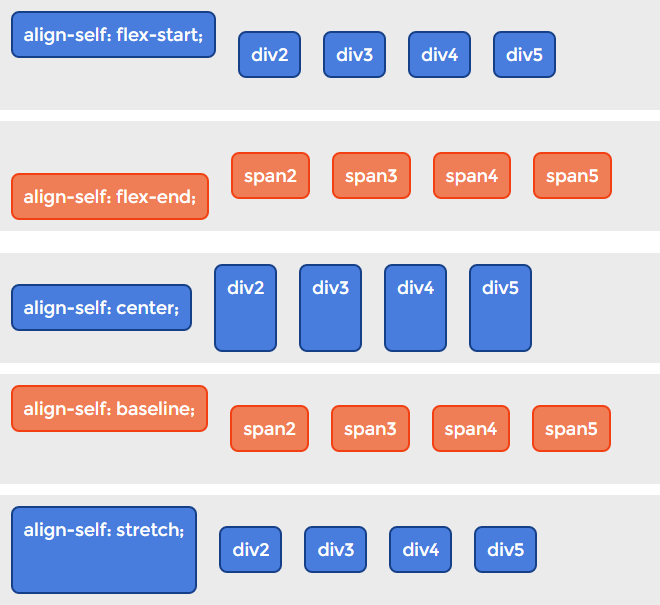
}

CSS

##### **6.2.2. Align-self**

Свойство отвечает за выравнивание отдельно взятого flex-элемента по высоте flex-контейнера. Переопределяет выравнивание, заданное align-items. Не наследуется.

|  |  |
| --- | --- |
| **align-self** | |
| Значения: |  |
| auto | Значение по умолчанию. Flex-элемент использует выравнивание, указанное в свойстве align-items flex-контейнера. |
| flex-start | Верхний край flex-элемента помещается вплотную с flex-линией (или на расстоянии, с учетом заданных полей margin и рамок border элемента), проходящей через начало поперечной оси. |
| flex-end | Нижний край flex-элемента помещается вплотную с flex-линией (или на расстоянии, с учетом заданных полей margin и рамок border элемента), проходящей через конец поперечной оси. |
| center | Поля flex-элемента центрируется по поперечной оси в пределах flex-линии. |
| baseline | Flex-элемент выравнивается по базовой линии. |
| stretch | Если поперечный размер flex-элемента вычисляется как auto и ни одно из поперечных значений margin не равно auto, элемент растягивается. Значение по умолчанию. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |



**Синтаксис**

.flex-container{

display: -webkit-flex;

display: flex;

}

.flex-item{

-webkit-align-self: center;

align-self: center;

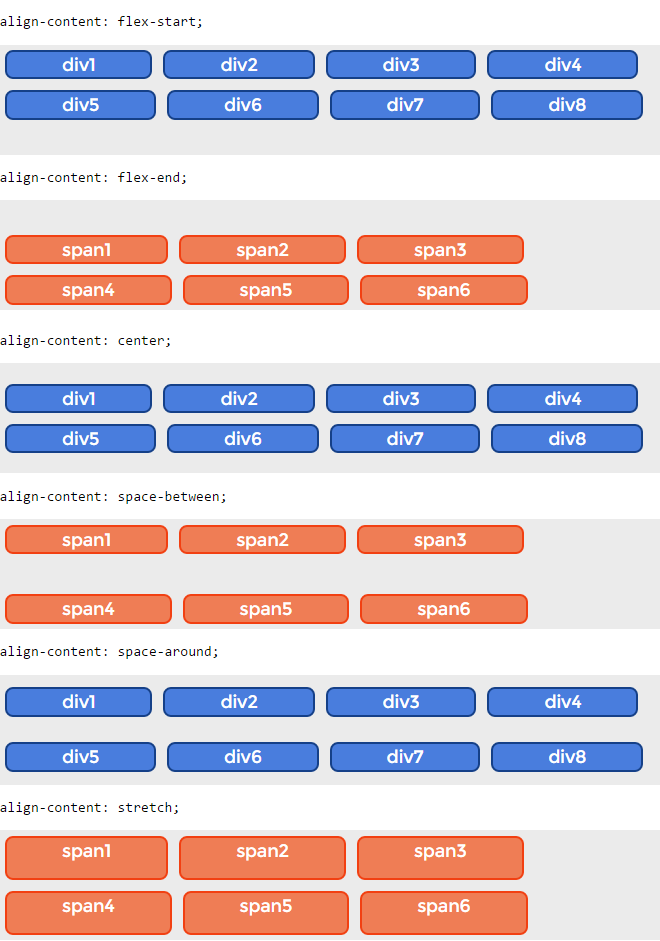
}

CSS

### 6.3. Выравниваниестрок flex-контейнера: align-content

Свойство выравнивает строки в flex-контейнере при наличии дополнительного пространства на поперечной оси, аналогично выравниванию отдельных элементов на главной оси с помощью свойства justify-content. Свойство не влияет на однострочный flex-контейнер. Не наследуется.

|  |  |
| --- | --- |
| **align-content** | |
| Значения: |  |
| flex-start | Строки укладываются по направлению к началу flex-контейнера. Край первой строки помещается вплотную к краю flex-контейнера, каждая последующая — вплотную к предыдущей строке. |
| flex-end | Строки укладываются по направлению к концу flex-контейнера. Край последней строки помещается вплотную к краю flex-контейнера, каждая предыдущая — вплотную к последующей строке. |
| center | Строки укладываются по направлению к центру flex-контейнера. Строки расположены вплотную друг к другу и выровнены по центру flex-контейнера с равным расстоянием между начальным краем содержимого flex-контейнера и первой строкой и между конечным краем содержимого flex-контейнера и последней строкой. |
| space-between | Строки равномерно распределены в flex-контейнере. Если оставшееся свободное пространство отрицательно или в flex-контейнере имеется только одна flex-линия, это значение идентично flex-start. В противном случае край первой строки помещается вплотную к начальному краю содержимого flex-контейнера, край последней строки — вплотную к конечному краю содержимого flex-контейнера. Остальные строки распределены так, чтобы расстояние между любыми двумя соседними строками было одинаковым. |
| space-around | Строки равномерно распределены в flex-контейнере с половинным пробелом на обоих концах. Если оставшееся свободное пространство отрицательно, это значение идентично центcenter. В противном случае строки распределяются таким образом, чтобы расстояние между любыми двумя соседними строками было одинаковым, а расстояние между первой / последней строками и краями содержимого flex-контейнера составляло половину от расстояния между строками. |
| stretch | Значение по умолчанию. Строки flex-элементов равномерно растягиваются, заполняя все доступное пространство. Если оставшееся свободное пространство отрицательно, это значение идентично flex-start. В противном случае свободное пространство будет разделено поровну между всеми строками, увеличивая их поперечный размер. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |



Проверка кросбраузерности и адаптивности.

Чтобы выполнить тестирование кроссбраузерности потребуется обратиться к одному из многочисленных онлайн-сервисов или локальных приложений.

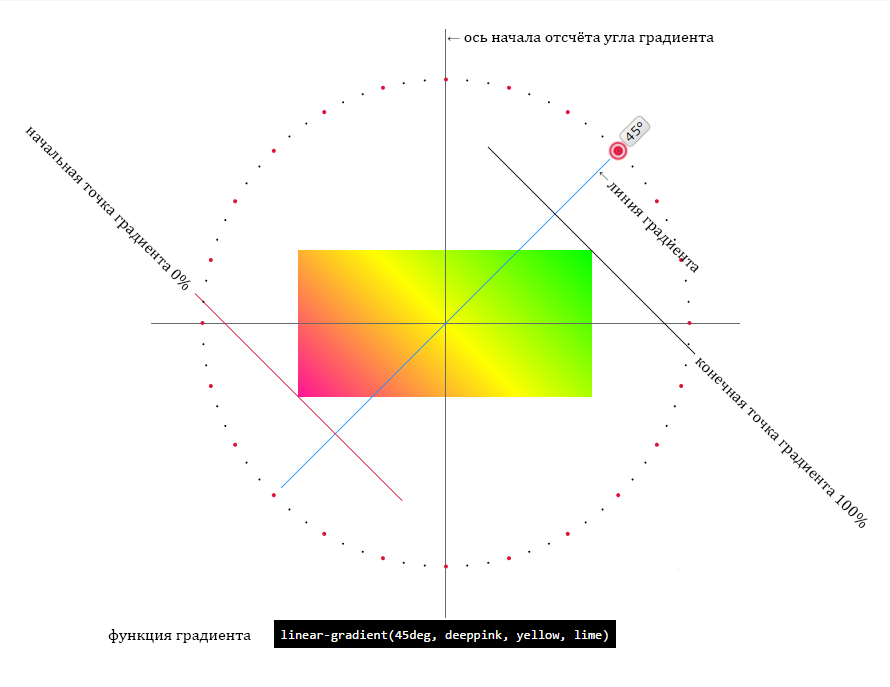
****Browsershots****  
  
Данный сервис проверки кроссбраузерности сайтов онлайн специализируется на создании скриншотов проверяемого веб-сайта из разных браузеров различных версий (в т. ч. устаревших и не очень популярных), общее количество которых превышает 200 шт.

****IENetRenderer****  
  
Полностью бесплатное решение для проверки кроссбраузерности в онлайне. Особенность его в том, что проверка доступна только для браузера Internet Explorer (с версии 5.5 до 11).

****Browser Sandbox****  
  
Онлайн сервис для тестирования кроссбраузерности, поддерживающий очень много как десктропных, так и мобильных версий браузеров.

Оформление веб-проекта.

### 1. Линейный градиент linear-gradient()

РИС. 1. ЛИНИЯ ГРАДИЕНТА, НАЧАЛЬНАЯ И КОНЕЧНАЯ ТОЧКИ И УГОЛ ГРАДИЕНТА

**Линейный градиент** создается с помощью двух и более цветов, для которых задано направление, или **линия градиента**.

Если направление не указано, используется значение по умолчанию — **сверху-вниз**.

Цвета градиента по умолчанию распределяются равномерно в направлении, перпендикулярном линии градиента.

background:linear-gradient(угол / сторона или угол наклона с помощью ключевого слова (пары ключевых слов), первый цвет, второй цвет и т.д.);

CSS

**Направление** градиента может быть задано двумя способами:  
**с помощью угла наклона** в градусах deg, значение которого определяет угол наклона линии внутри элемента.

div{

height: 200px;

background:linear-gradient(45deg, #EECFBA, #C5DDE8);

}

CSS

**с помощью ключевых слов** to top, to right, to bottom, to left, которые соответствуют углу градиента, равному 0deg, 90deg, 180deg и 270degсоответственно.

div{

height: 200px;

background:linear-gradient(to right, #F6EFD2, #CEAD78);

}

CSS

Если направление задано парой ключевых слов, например, to top left, то начальная точка градиента будет расположена в противоположном направлении, в данном случае справа внизу.

div{

height: 200px;

background:linear-gradient(to top left, powderblue, pink);

}

CSS

Для неравномерного распределения цветов указывается начальная позиция каждого цвета через точки остановки градиента, так называемые **color stops**. **Точки остановки** задаются в %, где 0% — начальная точка, 100% — конечная точка, например:

div{

height: 200px;

background:linear-gradient(to top, #E4AF9D 20%, #E4E4D8 50%, #A19887 80%);

}

CSS

Для чёткого распределения цветных полос каждый последующий цвет нужно начинать с точки остановки предыдущего цвета:

div{

height: 200px;

background:linear-gradient(to right, #FFDDD6 20%, #FFF9ED 20%, #FFF9ED 80%, #DBDBDB 80%);

}

CSS

### 2. Радиальный градиент radial-gradient()

**Радиальный градиент** отличается от линейного тем, что цвета выходят из одной точки (центра градиента) и равномерно распределяются наружу, рисуя форму круга или эллипса.

background:radial-gradient(форма градиента / размер / позиция центра, первый цвет, второй цвет и т.д.);

CSS

**Форма градиента** определяется ключевыми словами circle или ellipse. Если форма не задана, по умолчанию радиальный градиент принимает форму эллипса.

div{

height: 200px;

background:radial-gradient(white, #FFA9A1);

}

CSS

**Позиция центра** задаётся с помощью ключевых слов, используемых в свойстве background-position, с добавлением приставки at. Если позиция центра не задана, используется значение по умолчанию at center.

div{

height: 200px;

background:radial-gradient(at top, #FEFFFF, #A7CECC);

}

CSS

С помощью пары значений, указанных в единицах длины %, em или px, можно управлять размером эллипсообразного градиента. Первое значение задает ширину эллипса, второе — высоту.

div{

height: 200px;

background:radial-gradient(40% 50%, #FAECD5, #CAE4D8);

}

CSS

**Размер градиента** задаётся с помощью ключевых слов. Значение по умолчанию farthest-corner (к дальнему углу).

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| closest-side | Размер градиента рассчитывается из расстояния до любой ближней стороны блока для circle или до ближних сторон по X и по У для ellipse. |
| farthest-side | Размер рассчитывается из расстояния до дальних сторон. |
| closest-corner | Размер рассчитывается из расстояния до ближних углов. |
| farthest-corner | Размер рассчитывается из расстояния до дальних углов. |

div{

height: 200px;

background:radial-gradient(circle farthest-corner at 100px 50px, #FBF2EB, #352A3B);

}

CSS

С помощью радиального градиента можно создавать реалистичные объёмные фигуры, такие как мячи, кнопки.

#### **Мяч**

div{

width: 200px;

height: 200px;

border-radius: 50%;

margin: 0 auto;

background:radial-gradient(circle at 65% 15%, aqua, darkblue);

}

CSS

#### **Кнопка**

<div class="wrap"><div class="button"></div></div>

HTML

.wrap{

height: 200px;

padding: 50px 0;

background: #cccccc;

}

.button{

width: 100px;

height: 100px;

border-radius: 50%;

margin: 0 auto;

background:radial-gradient(farthest-side ellipse at top left, white, #aaaaaa);

box-shadow: 5px 10px 20px rgba(0,0,0,0.3), -5px -10px 20px rgba(255,255,255,0.5);

}

CSS

#### **Почтовая марка**



Используя в градиентах прозрачный цвет, можно создавать вот такие эффекты.

<div class="container">

<div class="wrap">

<img src="https://html5book.ru/wp-content/uploads/2015/10/flowers-by-barbara-florchik-2.jpg">

</div>

</div>

HTML

.container{

background: #B7D1D8;

padding: 25px;

}

.wrap{

background:radial-gradient(transparent, transparent 4px, white 4px,white);

padding: 10px;

width: 300px;

height: 220px;

background-size: 20px 20px;

background-position: -10px -10px;/\*обрезаемузорпокраю\*/

margin: 0 auto;

}

img{

width: 100%;

}

CSS

### 3. Повторградиента

В добавление к линейному и радиальному градиентам существует повтор градиента, который задается с помощью функций repeating-linear-gradient() и repeating-radial-gradient() соответственно. Фон из повторяющихся градиентов выглядит неаккуратно, поэтому рекомендуется начинать следующий цвет с точки остановки предыдущего, создавая таким образом полосатые узоры.

div{

height: 200px;

background:repeating-linear-gradient(45deg, #606dbc, #606dbc 10px, #465298 10px, #465298 20px);

}

CSS

div{

height: 200px;

background:repeating-radial-gradient(circle, #B9ECFE, #B9ECFE 10px, #82DDFF 10px, #82DDFF 20px);

}

CSS

### 4. Кроссбраузерный градиент

Для корректного отображения градиентов во всех браузерах необходимо добавить кроссбраузерную запись.

**Линейныйградиент**

-ms-filter:"progid:DXImageTransform.Microsoft.gradient (GradientType=0, startColorstr=#1471da, endColorstr=#1C85FB)";/\* IE 8-9 \*/

background:-webkit-linear-gradient(left, red, #f06d06);/\* Safari 5.1, iOS 5.0-6.1, Chrome 10-25, Android 4.0-4.3 \*/

background:-moz-linear-gradient(left, red, #f06d06);/\* Firefox 3.6-15 \*/

background:-o-linear-gradient(left, red, #f06d06);/\* Opera 11.1-12 \*/

background:linear-gradient(to right, red, #f06d06);/\* Opera 15+, Chrome 25+, IE 10+, Firefox 16+, Safari 6.1+, iOS 7+, Android 4.4+ \*/

CSS

**Повторлинейногоградиента**

background:-webkit-repeating-linear-gradient(red, yellow 10%, green 20%);/\* Safari 5.1 - 6.0 \*/

background:-o-repeating-linear-gradient(red, yellow 10%, green 20%);/\* Opera 11.1-12.0 \*/

background:-moz-repeating-linear-gradient(red, yellow 10%, green 20%);/\* Firefox 3.6-15 \*/

background:repeating-linear-gradient(red, yellow 10%, green 20%);/\* Стандартныйсинтаксис \*/

CSS

**Радиальныйградиент**

background:-webkit-radial-gradient(red, yellow, green);/\* Safari 5.1-6.0 \*/

background:-o-radial-gradient(red, yellow, green);/\* Opera 11.6-12.0 \*/

background:-moz-radial-gradient(red, yellow, green);/\* Firefox 3.6-15 \*/

background:radial-gradient(red, yellow, green);/\* Стандартныйсинтаксис \*/

background:-webkit-radial-gradient(60% 55%, farthest-side, red, yellow, black);/\* Safari 5.1-6.0 \*/

background:-o-radial-gradient(60% 55%, farthest-side, red, yellow, black);/\* Opera 11.6-12.0 \*/

background:-moz-radial-gradient(60% 55%, farthest-side, red, yellow, black);/\* Firefox 3.6-15 \*/

background:radial-gradient(farthest-side at 60% 55%, red, yellow, black);/\* Стандартныйсинтаксис \*/

CSS

**Повторрадиальногоградиента**

background:-webkit-repeating-radial-gradient(red, yellow 10%, green 15%);/\* Safari 5.1-6.0 \*/

background:-o-repeating-radial-gradient(red, yellow 10%, green 15%);/\* Opera 11.6-12.0 \*/

background:-moz-repeating-radial-gradient(red, yellow 10%, green 15%);/\* Firefox 3.6-15 \*/

background:repeating-radial-gradient(red, yellow 10%, green 15%);/\* Стандартныйсинтаксис \*/

CSS

### 5. Комбинация градиента и фонового изображения

За счёт комбинации градиента и изображения можно создавать интересные эффекты. Для градиента нужно использовать полупрозрачные цвета, чтобы картинка оставалась видимой.

div{

height: 453px;

background:linear-gradient(45deg,rgba(103, 0, 31, .8),rgba(34, 101, 163, .5)),url(https://html5book.ru/wp-content/uploads/2016/12/photo-8.jpg);

background-size: cover;

}

**Границы, аутлайныитени.**

Border

Универсальное свойство border позволяет одновременно установить толщину, стиль и цвет границы вокруг элемента. Значения могут идти в любом порядке, разделяясь пробелом, браузер сам определит, какое из них соответствует нужному свойству. Для установки границы только на определенных сторонах элемента, воспользуйтесь свойствами border-top, border-bottom, border-left,border-right.

Значение border-width определяет толщину границы. Для управления ее видом предоставляется несколько значений border-style. Их названия и результат действия представлен на рис. 1.

Стили рамок

border-color устанавливает цвет границы, значение может быть в любом допустимом для CSS формате.

inherit наследует значение родителя.

Outline

Задает стиль внешней границы элемента. В отличие от линии, задаваемой через border, линия через outline отображается вокруг элемента, не влияя на ширину блока или его положение.

Рис. 1

Добавляет тень к элементу. Допускается использовать несколько теней, указывая их параметры через запятую, при наложении теней первая тень в списке будет выше, последняя ниже. Если для элемента задается радиус скругления через свойство border-radius, то тень также получится с закругленными уголками. Добавление тени увеличивает ширину элемента, поэтому возможно появление горизонтальной полосы прокрутки в браузере.

none

Отменяет добавление тени.

inset

Тень выводится внутри элемента. Необязательный параметр.

сдвиг по x

Смещение тени по горизонтали относительно элемента. Положительное значение этого параметра задает сдвиг тени вправо, отрицательное — влево. Обязательный параметр.

сдвиг по y

Смещение тени по вертикали относительно элемента. Положительное значение задает сдвиг тени вниз, отрицательное — вверх. Обязательный параметр.

размытие

Задает радиус размытия тени. Чем больше это значение, тем сильнее тень сглаживается, становится шире и светлее. Если этот параметр не задан, по умолчанию устанавливается равным 0, тень при этом будет четкой, а не размытой.

растяжение

Положительное значение растягивает тень, отрицательное, наоборот, ее сжимает. Если этот параметр не задан, по умолчанию устанавливается 0, при этом тень будет того же размера, что и элемент.

цвет

Цвет тени в любом доступном CSS формате, по умолчанию тень черная. Необязательный параметр.

Допускается указывать несколько теней, разделяя их параметры между собой запятой. Учитывается следующий порядок: первая тень в списке размещается на самом верху, последняя в списке — в самом низу.

**Лучшие шрифты для веб:**

**Roboto**

Roboto

*Roboto*

**Open sans**

Open sans

*Open sans*

**Georgia**

Georgia

*Georgia*

**Montsserat**

Montsserat

*Montsserat*

Стили указателей.

Сursor

Устанавливает форму курсора, когда он находится в пределах элемента. Вид курсора зависит от операционной системы и установленных параметров.

Прежде чем воспользоваться возможностью переделать вид курсора, решите, а будет ли он использоваться к месту. Многих пользователей подобные изменения могут ввести в заблуждение, когда, например, вместо традиционной руки, появляющейся при наведении на ссылку, возникает нечто другое. В большинстве случаев, лучше оставить все как есть.

url

Позволяет установить свой собственный курсор, для этого нужно указать путь к файлу с курсором.

auto

Вид курсора по умолчанию для текущего элемента.

inherit

Наследует значение родителя.

|  |  |  |  |
| --- | --- | --- | --- |
| **Вид** | **Значение** | **Тест** | **Пример** |
| http://htmlbook.ru/files/images/css/css_cursor_default.png | default |  | P {cursor: default} |
| http://htmlbook.ru/files/images/css/css_cursor_crosshair.png | crosshair |  | P {cursor: crosshair} |
| http://htmlbook.ru/files/images/css/css_cursor_help.png | help |  | P {cursor: help} |
| http://htmlbook.ru/files/images/css/css_cursor_move.png | move |  | P {cursor: move} |
| http://htmlbook.ru/files/images/css/css_cursor_pointer.png | pointer |  | P {cursor: pointer} |
| http://htmlbook.ru/files/images/css/css_cursor_progress.png | progress |  | P {cursor: progress} |
| http://htmlbook.ru/files/images/css/css_cursor_text.png | text |  | P {cursor: text} |
| http://htmlbook.ru/files/images/css/css_cursor_wait.png | wait |  | P {cursor: wait} |
| http://htmlbook.ru/files/images/css/css_cursor_nresize.png | n-resize |  | P {cursor: n-resize} |
| http://htmlbook.ru/files/images/css/css_cursor_neresize.png | ne-resize |  | P {cursor: ne-resize} |
| http://htmlbook.ru/files/images/css/css_cursor_eresize.png | e-resize |  | P {cursor :e-resize} |
| / | se-resize |  | P {cursor: se-resize} |
| http://htmlbook.ru/files/images/css/css_cursor_sresize.png | s-resize |  | P {cursor: s-resize} |
| http://htmlbook.ru/files/images/css/css_cursor_swresize.png | sw-resize |  | P {cursor: sw-resize} |
| http://htmlbook.ru/files/images/css/css_cursor_wresize.png | w-resize |  | P {cursor: w-resize} |
| http://htmlbook.ru/files/images/css/css_cursor_nwresize.png | nw-resize |  | P {cursor :nw-resize} |

В зависимости от операционной системы и ее настроек вид курсора может отличаться от приведенных в таблице.

При добавлении курсора из файла синтаксис несколько видоизменится.

cursor: url('путь к курсору1'), url('путь к курсору2'), ..., <ключевое слово>

Через запятую допускается указывать несколько значений url, в этом случае браузер попытается открыть первый файл с курсором и если это по каким-либо причинам не получится, перейдет к следующему файлу. Список обязательно заканчивается ключевым словом, например, auto или pointer, допустимые значения перечислены выше.

**Фавикон** (favicon - это сокращение от Favorite icon, в переводе с анг. означает любимый значок) - иконка, также известная как ярлык, значок веб-сайта или значок закладки. Фавикон является обычным графическим изображением (картинкой), связанным с конкретной веб-страницей или веб-сайтом. Браузеры, которые поддерживают добавление фавикона, обычно отображают его в адресной строке и во вкладках рядом с названием страницы, некоторые браузеры используют его как значок для ярлыка, сохраненного на рабочем столе.

Для добавления фавикона на веб-страницу в начало HTML-документа потребуется вписать всего одну строку с [тегом <link>](https://puzzleweb.ru/html/tag_link.php), в котором нужно указать атрибут rel, указывающий браузеру что мы добавляем иконку, атрибут href, содержащий адрес нашего изображения, и атрибут type, показывающий какой формат изображения мы используем:

<html>

  <head>

    <link rel="icon" href="путь\_к\_иконке/favicon.ico" type="image/x-icon">

  </head>

  <body>

  </body>

</html>

В таблице показаны различные варианты добавления фавикона

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Google Chrome** | **Internet Explorer** | **Firefox** | **Opera** | **Safari** |
| <link rel="shortcut icon" href="httр://mysite.ru/myicon.ico"> | Да | Да | Да | Да | Да |
| <link rel="icon" href="httр://mysite.ru/image.ico"> | Да | Да (с IE 11) | Да | Да | Да |
| <link rel="icon" type="image/vnd.microsoft.icon" href="httр://mysite.ru/image.ico"> | Да | Да (с IE 9) | Да | Да | Да |
| <link rel="icon" type="image/png" href="httр://mysite.ru/image.png"> | Да | Да (с IE 11) | Да | Да | Да |
| <link rel="icon" type="image/gif" href="httр://mysite.ru/image.gif"> | Да | Да (с IE 11) | Да | Да | Да |
| <link rel="icon" type="image/x-icon" href="httр://mysite.ru/image.ico"> | Да | Да (с IE 9) | Да | Да | Да |

Advansed CSS.

**Приоритет стилей.**

!important

Играет роль в том случае, когда пользователи подключают свою собственную таблицу стилей. Если возникает противоречие, когда стиль автора страницы и пользователя для одного и того же элемента не совпадает, то !important позволяет повысить приоритет стиля.

При использовании пользовательской таблицы стилей или одновременном применении разного стиля автора и пользователя к одному и тому же селектору, браузер руководствуется следующим алгоритмом.

!important добавлен в авторский стиль — будет применяться стиль автора.

!important добавлен в пользовательский стиль — будет применяться стиль пользователя.

!important нет как в авторском стиле, так и стиле пользователя — будет применяться стиль автора.

!important содержится в авторском стиле и стиле пользователя — будет применяться стиль пользователя.

|  |  |  |
| --- | --- | --- |
| **Стиль автора** | **Стиль пользователя** | **Результат** |
| BODY { /\* Серый цвет текста \*/ color: silver;   /\* Размер текста 8 пунктов \*/ font-size: 8pt  } | BODY { /\* Черный цвет текста \*/ color: #000;   /\* Размер текста 12 пунктов \*/ font-size: 12pt  } | Lorem ipsum dolor sit amet...  Будет применяться стиль автора, а именно: установлен серый цвет текста, размер шрифта 8 пунктов. |
| BODY { /\* Серый цвет текста \*/ color: silver;  /\* Размер текста 8 пунктов \*/ font-size: 8pt  } | BODY { /\* Черный цвет текста, повышенная важность \*/ color: #000 !important;   /\* Размер текста 12 пунктов \*/ font-size: 12pt  } | Lorem ipsum dolor sit amet...  Цвет текста будет установлен пользовательским, а размер шрифта останется исходным. В итоге увидим черный текст размером 8 пунктов. |
| BODY { /\* Серый цвет текста \*/ color: silver;   /\* Размер текста 8 пунктов \*/ font-size: 8pt  } | BODY { /\* Черный цвет текста, повышенная важность \*/ color: #000 !important;  /\* Размер текста 12 пунктов, повышенная важность \*/ font-size: 12pt !important } | Lorem ipsum dolor sit amet...  Будут использоваться все стилевые свойства пользователя. Текст станет отображаться как черный, размер 12 пунктов. |
| BODY { /\* Серый цвет текста, повышенная важность \*/ color: silver !important;   /\* Размер текста 8 пунктов, повышенная важность \*/ font-size: 8pt !important } | BODY { /\* Черный цвет текста, повышенная важность \*/ color: #000 !important;  /\* Размер текста 12 пунктов, повышенная важность \*/ font-size: 12pt !important } | Lorem ipsum dolor sit amet...  При добавлении !important в ту и другую таблицу приоритет в этом случае имеет стиль пользователя. В результате текст станет черным, а размер его увеличится до 12 пунктов. |

**Псевдоклассы и псевдоэлементы.**

**Псевдокласс** в CSS — *это ключевое слово, добавленное к селектору,* которое определяет его особое состояние. Например, [:hover](https://developer.mozilla.org/ru/docs/Web/CSS/:hover) может быть использован для изменения цвета кнопки при наведении курсора на неё.

div:hover{

background-color: #F89B4D;

}

Псевдоклассы дают возможность стилизовать элемент на основе не только отношений в DOM-дереве, но и основываясь на внешних факторах, таких как история посещений (например, [:visited](https://developer.mozilla.org/ru/docs/Web/CSS/:visited)), состояние содержимого (вроде [:checked](https://developer.mozilla.org/ru/docs/Web/CSS/:checked) у некоторых элементов формы) или позиции курсора мыши (например, [:hover](https://developer.mozilla.org/ru/docs/Web/CSS/:hover) определяет, находится ли курсор мыши над элементом).

**Список стандартных псевдоклассов**

* [:active](https://developer.mozilla.org/ru/docs/Web/CSS/:active)
* [:any](https://developer.mozilla.org/ru/docs/Web/CSS/:any)
* [:any-link](https://developer.mozilla.org/ru/docs/Web/CSS/:any-link)
* [:checked](https://developer.mozilla.org/ru/docs/Web/CSS/:checked)
* [:default](https://developer.mozilla.org/ru/docs/Web/CSS/:default)
* [:defined](https://developer.mozilla.org/ru/docs/Web/CSS/:defined)
* [:dir()](https://developer.mozilla.org/ru/docs/Web/CSS/:dir)
* [:disabled](https://developer.mozilla.org/ru/docs/Web/CSS/:disabled)
* [:empty](https://developer.mozilla.org/ru/docs/Web/CSS/:empty)
* [:enabled](https://developer.mozilla.org/ru/docs/Web/CSS/:enabled)
* [:first](https://developer.mozilla.org/ru/docs/Web/CSS/:first)
* [:first-child](https://developer.mozilla.org/ru/docs/Web/CSS/:first-child)
* [:first-of-type](https://developer.mozilla.org/ru/docs/Web/CSS/:first-of-type)
* [:fullscreen](https://developer.mozilla.org/ru/docs/Web/CSS/:fullscreen)
* [:focus](https://developer.mozilla.org/ru/docs/Web/CSS/:focus)
* [:hover](https://developer.mozilla.org/ru/docs/Web/CSS/:hover)
* [:indeterminate](https://developer.mozilla.org/ru/docs/Web/CSS/:indeterminate)
* [:in-range](https://developer.mozilla.org/ru/docs/Web/CSS/:in-range)
* [:invalid](https://developer.mozilla.org/ru/docs/Web/CSS/:invalid)
* [:lang()](https://developer.mozilla.org/ru/docs/Web/CSS/:lang)
* [:last-child](https://developer.mozilla.org/ru/docs/Web/CSS/:last-child)
* [:last-of-type](https://developer.mozilla.org/ru/docs/Web/CSS/:last-of-type)
* [:left](https://developer.mozilla.org/ru/docs/Web/CSS/:left)
* [:link](https://developer.mozilla.org/ru/docs/Web/CSS/:link)
* [:not()](https://developer.mozilla.org/ru/docs/Web/CSS/:not)
* [:nth-child()](https://developer.mozilla.org/ru/docs/Web/CSS/:nth-child)
* [:nth-last-child()](https://developer.mozilla.org/ru/docs/Web/CSS/:nth-last-child)
* [:nth-last-of-type()](https://developer.mozilla.org/ru/docs/Web/CSS/:nth-last-of-type)
* [:nth-of-type()](https://developer.mozilla.org/ru/docs/Web/CSS/:nth-of-type)
* [:only-child](https://developer.mozilla.org/ru/docs/Web/CSS/:only-child)
* [:only-of-type](https://developer.mozilla.org/ru/docs/Web/CSS/:only-of-type)
* [:optional](https://developer.mozilla.org/ru/docs/Web/CSS/:optional)
* [:out-of-range](https://developer.mozilla.org/ru/docs/Web/CSS/:out-of-range)
* [:read-only](https://developer.mozilla.org/ru/docs/Web/CSS/:read-only)
* [:read-write](https://developer.mozilla.org/ru/docs/Web/CSS/:read-write)
* [:required](https://developer.mozilla.org/ru/docs/Web/CSS/:required)
* [:right](https://developer.mozilla.org/ru/docs/Web/CSS/:right)
* [:root](https://developer.mozilla.org/ru/docs/Web/CSS/:root)
* [:scope](https://developer.mozilla.org/ru/docs/Web/CSS/:scope)
* [:target](https://developer.mozilla.org/ru/docs/Web/CSS/:target)
* [:valid](https://developer.mozilla.org/ru/docs/Web/CSS/:valid)
* [:visited](https://developer.mozilla.org/ru/docs/Web/CSS/:visited)

**Псевдоэлемент** в CSS — *это ключевое слово, добавляемое к селектору,* которое позволяет стилизовать определённую часть выбранного элемента. Например, псевдоэлемент [::first-line](https://developer.mozilla.org/ru/docs/Web/CSS/::first-line) может быть использован для изменения шрифта первой строки абзаца.

/\* Первая строка каждого элемента <p>. \*/

p::first-line{

color: blue;

text-transform: uppercase;

}

**Списокстандартныхпсевдоэлементов**

[::after](https://developer.mozilla.org/ru/docs/Web/CSS/::after)

* [::before](https://developer.mozilla.org/ru/docs/Web/CSS/::before)
* [::cue](https://developer.mozilla.org/ru/docs/Web/CSS/::cue)
* [::first-letter](https://developer.mozilla.org/ru/docs/Web/CSS/::first-letter)
* [::first-line](https://developer.mozilla.org/ru/docs/Web/CSS/::first-line)
* [::selection](https://developer.mozilla.org/ru/docs/Web/CSS/::selection)
* [::slotted](https://developer.mozilla.org/ru/docs/Web/CSS/::slotted)
* [::backdrop](https://developer.mozilla.org/ru/docs/Web/CSS/::backdrop)
* [::placeholder](https://developer.mozilla.org/ru/docs/Web/CSS/::placeholder)
* [::marker](https://developer.mozilla.org/ru/docs/Web/CSS/::marker)
* [::spelling-error](https://developer.mozilla.org/ru/docs/Web/CSS/::spelling-error)

[::grammar-error](https://developer.mozilla.org/ru/docs/Web/CSS/::grammar-error)

# Переходы и анимация

Эволюцией CSS3 стала возможность задавать поведение для переходов и анимации. Фронтенд-разработчики много лет просили реализовать эти взаимодействия внутри HTML и CSS, без использования JavaScript или Flash. Теперь их мечта сбылась.

С помощью переходов CSS3 у вас есть потенциал менять внешний вид и поведение элемента всякий раз, когда происходит изменение его состояния, к примеру, когда на элемент наводится курсор, он получает фокус, становится активным или к элементу происходит переход по ссылке.

Анимация в CSS3 позволяет менять внешний вид и поведение элемента с помощью нескольких ключевых кадров. Переходы обеспечивают смену от одного состояния к другому, в то время как анимация может установить несколько точек перехода через разные ключевые кадры.

## Переходы

Как уже упоминалось, чтобы [переход](http://www.alistapart.com/articles/understanding-css3-transitions/) имел место, элемент должен получить изменение состояния и для каждого состояния должны быть определены разные стили. Самый простой способ задать стили для различных состояний — это воспользоваться псевдоклассами :hover, :focus, :active и :target.

Естьчетыресвойства, связанныхспереходамивцелом, это transition-property, transition-duration, transition-timing-function и transition-delay. Не все они обязательны для создания перехода, наиболее популярны первые три.

В приведённом ниже примере блок изменит цвет фона в течение одной секунды в линейном виде.

.box {

background: #2db34a;

transition-property: background;

transition-duration: 1s;

transition-timing-function: linear;

}

.box:hover {

background: #ff7b29;

}

### Вендорныепрефиксы

Приведённый выше код, как и остальные образцы кода в этом уроке, не используют вендорные префиксы. Это сделано намерено в интересах сохранения фрагмента кода маленьким и понятным. Для лучшей поддержки во всех браузерах используйте префиксы. Для справки, версия с префиксами для кода выше будет выглядеть следующим образом.

.box {

background: #2db34a;

-webkit-transition-property: background;

-moz-transition-property: background;

-o-transition-property: background;

transition-property: background;

-webkit-transition-duration: 1s;

-moz-transition-duration: 1s;

-o-transition-duration: 1s;

transition-duration: 1s;

-webkit-transition-timing-function: linear;

-moz-transition-timing-function: linear;

-o-transition-timing-function: linear;

transition-timing-function: linear;

}

.box:hover {

background: #ff7b29;

}

### transition-property

Свойство transition-property определяет, какие именно свойства будут меняться в сочетании с другими свойствами перехода. По умолчанию, изменяться будут все свойства в различных состояниях. Тем не менее, только свойства заданные в значении transition-property будут затронуты в любых переходах.

В приведённом выше примере, свойство background определено в значении transition-property. Здесь background — единственное свойство, которое будет изменяться в течение 1 секунды в линейном виде. Любые другие свойства, включённые когда изменяется состояние элемента, но не добавленные в значение transition-property, не получат поведение перехода, как это установлено в свойствах transition-duration или transition-timing-function.

Если требуется добавить несколько свойств в переход, они могут быть разделены запятой внутри значения transition-property. Кроме того, может быть использовано ключевое слово all для перехода всех свойств элемента.

.box {

background: #2db34a;

border-radius: 6px

transition-property: background, border-radius;

transition-duration: 1s;

transition-timing-function: linear;

}

.box:hover {

background: #ff7b29;

border-radius: 50%;

}

#### **Свойства в переходах**

Важно отметить, что **не для всех свойств может быть выполнен переход**, а только для свойств, которые имеют различимую точку на полпути. Цвета, размеры шрифтов и тому подобное могут переходить от одного значения к другому, поскольку у них есть распознаваемые значения в промежутке друг с другом. У свойства display, например, не может быть перехода, поскольку оно не имеет никакой средней точки. Набор наиболее популярных свойств, к которым применим переход следующий.

* background-color
* background-position
* border-color
* border-width
* border-spacing
* bottom
* clip
* color
* crop
* font-size
* font-weight
* height
* left
* letter-spacing
* line-height
* margin
* max-height
* max-width
* min-height
* min-width
* opacity
* outline-color
* outline-offset
* outline-width
* padding
* right
* text-indent
* text-shadow
* top
* vertical-align
* visibility
* width
* word-spacing
* z-index

### transition-duration

Длительность перехода задаётся с помощью свойства transition-duration. В качестве значения этого свойства можно использовать основные значения времени, в том числе секунды (s) и миллисекунды (ms). Эти значения времени можно также задавать и дробными — .2s, например.

При переходе нескольких свойств можно задать несколько длительностей, по одному на каждое свойство. Как и в случае значения свойства transition-property, несколько длительностей могут быть объявлены с помощью значений, разделённых запятыми. Важен порядок этих значений при определении отдельных свойств и длительностей. К примеру, первое свойство, заданное в transition-property, будет совпадать с первым значением в свойстве transition-duration и так далее.

Если на несколько свойств в переходе приходится только одно значение длительности, то оно будет задавать длительность для всех свойств перехода.

.box {

background: #2db34a;

border-radius: 6px;

transition-property: background, border-radius;

transition-duration: .2s, 1s;

transition-timing-function: linear;

}

.box:hover {

background: #ff7b29;

border-radius: 50%;

}

### Функция перехода

Свойство transition-timing-function используется для задания скорости, с которой будет двигаться переход. Зная длительность из свойства transition-duration у перехода может быть несколько скоростей в пределах одной длительности. Некоторые наиболее популярные значения ключевых слов для свойства transition-timing-function включают linear, ease-in, ease-out и ease-in-out.

Значение linear определяет переход, движущийся в постоянной скоростью из одного состояния к другому. Значение ease-inопределяет переход, который начинается медленно и ускоряется на протяжении перехода, в то время как значение ease-outопределяет переход, который начинает быстро и замедляется в течение всего перехода. Значение ease-in-out определяет переход, который начинается медленно, ускоряется в середине, а затем снова замедляется перед окончанием.

За каждой функцией времени стоит кубическая кривая Безье, которую можно указать конкретно с помощью значения cubic-bezier(x1, y1, x2, y2). Дополнительные значения включают step-start, step-stop и значение steps(<число шагов>, <направление>).

При переходе нескольких свойств, вы можете определить и несколько функций времени. Эти значения, подобно другим значениям свойств перехода, могут быть объявлены через запятую.

.box {

background: #2db34a;

border-radius: 6px;

transition-property: background, border-radius;

transition-duration: .2s, 1s;

transition-timing-function: linear, ease-in;

}

.box:hover {

background: #ff7b29;

border-radius: 50%;

}

### transition-delay

Кроме объявления свойства перехода, длительности и функции времени, вы можете также установить задержку через свойство transition-delay. Задержка задаёт значение времени, в секундах или миллисекундах, которое определяет, сколько времени переход должен ожидать перед выполнением. Как и со всеми другими свойствами перехода, чтобы задержать многочисленные переходы, каждая задержка может быть перечислена через запятую.

.box {

background: #2db34a;

border-radius: 6px

transition-property: background, border-radius;

transition-duration: .2s, 1s;

transition-timing-function: linear, ease-in;

transition-delay: 0s, 1s;

}

.box:hover {

background: #ff7b29;

border-radius: 50%;

}

## Сокращённоесвойство transition

Написание каждого свойства перехода по отдельности может напрягать, особенно с префиксами. К счастью, есть сокращённое свойство transition, способное поддерживать все эти разные свойства и значения. Используя одиночное значение transition, вы можете установить каждое значение перехода в следующем порядке: transition-property, transition-duration, transition-timing-function и, наконец, transition-delay. Не используйте запятые с этими значениями, если не желаете определить несколько переходов.

Чтобы задать несколько переходов сразу, установите каждую отдельную группу значений перехода, а затем используйте запятую, чтобы отделить каждую дополнительную группу значений.

.box {

background: #2db34a;

border-radius: 6px;

transition: background .2s linear, border-radius1s ease-in 1s;

}

.box:hover {

color: #ff7b29;

border-radius: 50%;

}

## Анимация

Переходы выполняют большую работу по созданию визуальных взаимодействий из одного состояния в другое и идеально подходят для этих видов одиночных изменений состояния. Тем не менее, когда требуется больший контроль, для переходов необходимо несколько состояний. С этого момента переходы уходят, а в дело вступает [анимация](http://coding.smashingmagazine.com/2011/09/14/the-guide-to-css-animation-principles-and-examples/).

### Ключевые кадры анимации

Чтобы установить несколько точек, в которых элемент должен совершать переход, используется правило @keyframes. Это правило включает имя анимации, любое число контрольных точек, а также свойства, предназначенные для анимации.

@keyframesslide {

0% {

left: 0;

top: 0;

}

50% {

left: 244px;

top: 100px;

}

100% {

left: 488px;

top: 0;

}

}

### Вендорные префиксы в правиле @keyframes

Правило @keyframes должно быть с вендорными префиксами, так же, как и все другие свойства transition и animation. Префиксы для @keyframes выглядят следующим образом:

* @-moz-keyframes
* @-o-keyframes
* @-webkit-keyframes

Анимация выше называется slide, состояния начинаются сразу же после открытия правила @keyframes. Разные контрольные точки ключевых кадров задаются помощью процентов, начиная с 0% и работая до 100%, с промежуточной точкой на 50%. При желании вместо 0% и 100% могут быть использованы ключевые слова from и to. Кроме 50% также могут быть указаны дополнительные контрольные точки. Свойства элементов для анимации перечислены внутри каждой контрольной точки, left и top в приведённом выше примере.

Важно отметить, как и с переходами, могут быть анимированы только отдельные свойства. Подумайте, как вы могли бы переместить элемент сверху вниз, например. Попытка анимировать от top: 0 до bottom: 0 не будет работать, потому что анимация может применять только переход в пределах одного свойства, а не от одного свойства к другому. В этом случае элемент необходимо анимировать от top: 0 до top: 100%.

### animation-name

После того, как объявлены ключевые кадры для анимации, они должны быть назначены для элемента. Для этого используется свойство animation-name с именем анимации из правила @keyframes, как значение свойства. Декларация animation-nameприменяется к элементу, для которого должна быть задана анимация.

.stage:hover.ball {

animation-name: slide;

}

Использования одного свойства animation-name при этом недостаточно. Кроме того, необходимо объявить свойство animation-duration и значение, чтобы браузер знал, как долго должна длиться анимация до завершения.

### animation-duration, функциявремении animation-delay

После того, как вы объявили свойство animation-name для элемента, анимация ведёт себя подобно переходам. Она включают в себя длительность, функцию времени и задержку при желании. Сперва анимации требуется длительность, объявленная с помощью свойства animation-duration. Как и в случае с переходами, длительность может быть задана в секундах или миллисекундах.

.stage:hover.ball {

animation-name: slide;

animation-duration: 2s;

}

Функциявремениизадержкамогутбытьобъявленыспомощьюсвойств animation-timing-function и animation-delay, соответственно. Значения этих свойств подражают и ведут себя так же, как это делается с переходами.

.stage:hover.ball {

animation-name: slide;

animation-duration: 2s;

animation-timing-function: ease-in-out;

animation-delay: .5s;

}

Анимация ниже должна вызвать отскок ball один раз при перемещении вправо, но только при наведении курсора на stage.

**HTML**

<divclass="stage">

<figureclass="ball"></figure>

</div>

**CSS**

@keyframesslide {

0% {

left: 0;

top: 0;

}

50% {

left: 244px;

top: 100px;

}

100% {

left: 488px;

top: 0;

}

}

.stage {

height: 150px;

position: relative;

}

.ball {

height: 50px;

position: absolute;

width: 50px;

}

.stage:hover.ball {

animation-name: slide;

animation-duration: 2s;

animation-timing-function: ease-in-out;

animation-delay: .5s;

}

## Настройка анимации

Анимация также предлагает возможность дальнейшей настройки поведения элемента, в том числе можно задать, сколько раз анимация выполняется, а также направление, в котором анимация завершится.

### animation-iteration-count

По умолчанию, анимация запускает свой цикл один раз от начала до конца, а затем останавливается. Чтобы анимация повторялась много раз может быть использовано свойство animation-iteration-count. Значения для него включают целое число или ключевое слово infinite. С помощью целого числа анимация будет повторяться столько раз, сколько указано, в то время как ключевое слово infinite будет повторять анимацию бесконечно и никогда не остановится.

.stage:hover.ball {

animation-name: slide;

animation-duration: 2s;

animation-timing-function: ease-in-out;

animation-delay: .5s;

animation-iteration-count: infinite;

}

### animation-direction

Кроме возможности установить, сколько раз анимация повторяется, вы можете также объявить направление, в котором анимация завершается, с помощью свойства animation-direction. Значения этого свойства включают в себя normal, reverse, alternate и alternate-reverse.

Значение normal воспроизводит анимацию, как и намеревалось, от начала до конца. Значение reverse воспроизводит анимацию точно наоборот, чем это определено в правиле @keyframes, таким образом, начиная со 100% и работая в обратном направлении до 0%.

Значение alternate воспроизведёт анимацию вперёд, а затем назад. В ключевых кадрах это включает выполнение вперёд от 0% до 100%, а затем в обратном направлении от 100% до 0%. Используя свойство animation-iteration-count можно ограничить количество раз, когда анимация работает вперёд и назад. Счёт начинается с 1, когда анимация проходит вперёд от 0% до 100%, а затем добавляется 1, когда анимация проходит в обратном порядке от 100% до 0%. Объединение в общей сложности двух итераций. Значение alternate также инвертирует любые функции времени при воспроизведении в обратном направлении. Если анимация использует значение ease-in идущее от 0% до 100%, то затем использует значение ease-outидущее от 100% до 0%.

И, наконец, значение alternate-reverse сочетает оба значения alternate и reverse, запуская анимацию назад, а затем вперёд. Значение alternate-reverse начинается со 100% и выполняется до 0%, а затем снова обратно до 100%.

.stage:hover.ball {

animation-name: slide;

animation-duration: 2s;

animation-timing-function: ease-in-out;

animation-delay: .5s;

animation-iteration-count: infinite;

animation-direction: alternate;

}

### animation-play-state

Свойство animation-play-state позволяет анимации воспроизводиться или остановиться на паузу, с помощью ключевых слов running и paused, соответственно. Когда анимация снимается с паузы, она возобновляет работу из текущего состояния, а не начинается снова с самого начала.

В приведённом ниже примере свойство animation-play-state устанавливается на паузу, когда stage активен при щелчке по нему. Обратите внимание на то, как анимация временно приостанавливается, пока вы не отпустите кнопку мыши.

.stage:hover.ball {

animation-name: slide;

animation-duration: 2s;

animation-timing-function: ease-in-out;

animation-delay: .5s;

animation-iteration-count: infinite;

animation-direction: alternate;

}

.stage:active.ball {

animation-play-state: paused;

}

### animation-fill-mode

Свойство animation-fill-mode определяет, как элемент должен быть стилизован — до, после или до и после запуска анимации. Свойство animation-fill-mode принимает четыре значения ключевых слов, включая none, forwards, backwards и both.

Значение none не будет применять любые стили к элементу до или после того, как анимация запущена.

Значение forwards сохранит стили, объявленные в последнем указанном ключевом кадре. На эти стили, однако, могут повлиять значения свойств animation-direction и animation-iteration-count, меняя, где анимация заканчивается.

Значение backwards будет применять стили первого заданного ключевого кадра, до того, как анимация будет работать. Включает применение этих стилей в течение всего времени, что может быть задано в animation-delay. Значение backwardsтакже может зависеть от значения свойства animation-direction.

Наконец, значение both применит поведение сразу от обоих значений forwards и backwards.

.stage:hover.ball {

animation-name: slide;

animation-duration: 2s;

animation-timing-function: ease-in-out;

animation-delay: .5s;

animation-fill-mode: forwards;

}

.stage:active.ball {

animation-play-state: paused;

}

Иснструменты верстальщика.

**Хостинг - что это?**

Интернет-ресурсы размещаются на серверах (компьютерах, работающих 24 часа в сутки, имеющих несколько разных интернет-провайдеров, размещенных в дата-центрах). Любой сайт представляет собой набор файлов и баз данных, которые требуют места для хранения на сервере. Хостинг – название услуги по размещению на сервере файлов Вашего сайта.**Для чего** нужен хостинг: без заказа хостинга при создании сайта не обойтись, т.к. посететелям сайта необходим круглосуточный доступ к его содержимому. Собственный сайт возможно создать и на своём компьютере, но затем его необходимо разместить на хостинге. Именно хостер следит за круглосуточной бесперебойной работой вашего сайта и работоспособностью серверного оборудования.

**Что такое домен сайта?**

**Домен – это имя сайта, его адрес в глобальной сети, то, что мы набираем в адресной строке браузера**.

Полное доменное имя (от англ. FQDN – Fully Qualified Domain Name – полностью определенное доменное имя) состоит из нескольких частей – собственно имени и всех доменов, разделенных точками.

Рассмотрим это на примере домена google.com.ua.

. (точка) – домен нулевого уровня – чаще всего точка не указывается, но подразумевается;

.ua — домен первого (верхнего) уровня;

com.ua — домен второго уровня;

google.com.ua — домен третьего уровня.

В этом контексте стоит далее рассмотреть такое понятие, как поддомен.

**Доменные зоны**

Домены верхнего уровня (от англ. TLD – top–level domains) разделяютна:

национальные (ccTLD – country code top level domains) – домены, относящиеся к определенной стране, состоят из двух букв, например, [.ua](https://hostiq.ua/domains/ua/) – Украина, [.ru](https://hostiq.ua/domains/ru/) – Россия;

тематические (gTLD – generic top level domains) – домены, определяющие тематическую принадлежность сайта, например, [.com](https://hostiq.ua/domains/com/) – коммерческий, [.travel](https://hostiq.ua/domains/travel/)– туристические услуги.

**Выбор домена**

При подборе подходящего имени для Вашего сайта стоит обратить внимание на такие моменты:

имя домена является коротким и запоминающимся;

имя домена ассоциируется с вашей деятельностью;

имя домена включает название вашего бренда или компании;

доменная зона относится к территории проживания аудитории сайта ([.com.ua](https://hostiq.ua/domains/com.ua/) для жителей Украины) или отражает тематическую направленность сайта ([.photo](https://hostiq.ua/domains/photo/) для фотографов);

старайтесь избегать сложных буквосочетаний (zh, sch), цифр и специальных символов (–, \*) в случае, если они не несут смысловой нагрузки.

**Командная разработка.**

Таск менеджер (менеджер задач) — это полезный инструмент для управления проектами.Рабочий процесс упрощается и становится более организованным. В таск менеджере (менеджере задач) вся информация разложена по полочкам. Кто над чем работает, когда начал, когда закончил. Если возникают вопросы по какой-то из задач, обсудить можно прямо в таск менеджере, чтобы не плодить бесконечные имейлы.

**Time Doctor** — это*мощная программа учета рабочего времени*, нацеленная на производительность и управление ежедневными задачами.

Подходит для профессионалов, желающих улучшить личный тайм-менеджмент при управлении проектами и увеличить продуктивность. Это полезный инструмент для команд, помогая освещать работу каждого сотрудника и отслеживать — как было потрачено время. Веб-сервис способен искоренить привычки залезать на сайты, в приложения и программы, которые отвлекают от рабочего процесса.

Система контроля версий или VCS может значительно облегчить работу разработчиков, пытающихся проанализировать изменения и вклады в общий код. Проще говоря, система контроля версий – это ключевой элемент в системе управления настройками программного обеспечения, которые отвечают потребностям проекта. VCS дают возможность назначать для определенных изменений/ревизий/обновлений буквенные или числовые значения. Также могут предоставить информацию о временных метках и идентификаторе человека внесшего изменения.

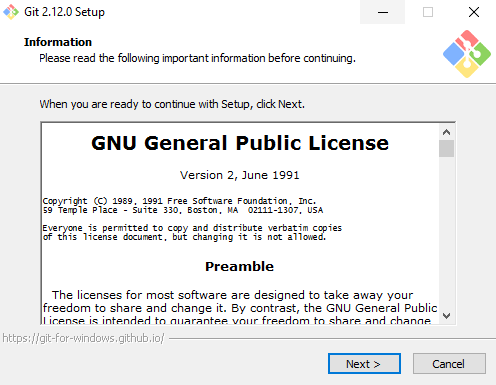
## ****Шаг 1 — Установка GIT****

В данной части руководства будут показаны простейшие способы установки GIT на разные операционные системы:

### ****Вариант 1 — Установка GIT на Windows****

Установка GIT на Windows, осуществляется также просто, как и установка любого другого приложения; загрузка установщика и его запуск.  Выполните данные шаги для установки GIT на Windows:

1. Посетите [этот сайт](https://git-for-windows.github.io/) и скачайте установщик GIT для Windows.
2. После загрузки, запустите установку двойным нажатием мыши. Следуйте инструкциям на экране, продолжайте нажимать **Next (Далее)** и наконец **Finish** **(Финиш)** для успешного завершения установки.



1. Запустите командную строку и впишите следующие команды в терминале:

git config --global user.name "John Smith"

git config --global user.email "example@email.com"

Это все что нужно для установки GIT на Windows.

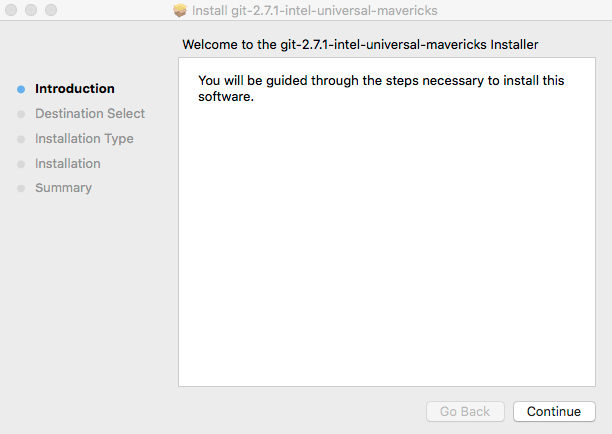
### ****Вариант 2 — Установка GIT на Mac****

Существует множество способов для установки GIT на Mac, есть даже вероятность того, что GIT уже установлен на вашем компьютере. Eсли у вас установлен XCode; выполните следующую команду в терминале для проверки:

git --version

Если ваш результат будет схож с таким git version 2.7.0 (Apple Git-66), тогда можете смело приступать к работе с GIT, если нет, то выполните следующие шаги:

1. Посетите [этот сайт](https://sourceforge.net/projects/git-osx-installer/files/) и скачайте последнюю версию установщика для Mac.
2. Следуйте инструкциям установщика и завершите установку.



1. Используйте команду git –version снова, для подтверждения того, что установка была успешна.
2. Выполните следующие команды в терминале для настройки вашего имени пользователя и адреса электронной почты, которые будут связаны с вашим GIT аккаунтом:

git config --global user.name "John Smith"

git config --global user.email "example@email.com"

### ****Вариант 3 — Установка GIT на Linux****

Если вы являетесь пользователем Linux, вы должны уметь устанавливать программы и пакеты на ваш компьютер используя простые команды apt-get или yum install; чтож, установка GIT ничем отличается:

**Для пользователей Debian/Ubuntu (apt-get):**

1. Откройте терминал и выполните следующую команду:

sudo apt-get update

sudo apt-get install git

1. Проверьте правильности установки командой git –version.
2. Выполните следующие команды в терминале для настройки вашего имени пользователя и адреса электронной почты, которые будут связаны с вашим GIT аккаунтом:

git config --global user.name "John Smith"

git config --global user.email "example@email.com"

**Fedora (yum/dnf):**

Вы можете получить пакеты GIT, используя как yum, так и dnf.

1. Откройте терминал и выполните следующие команды:

sudo dnf install git

sudo yum install git

1. Проверьте правильность установки командой git –version.
2. Выполните следующие команды в терминале для настройки вашего имени пользователя и адреса электронной почты, которые будут связаны с вашим GIT аккаунтом:

git config --global user.name "John Smith"

git config --global user.email "example@email.com"

## ****Шаг 2 — Использование GIT****

Теперь, когда GIT настроен на вашем устройстве Windows / Mac / Linux, давайте изучим основы GIT и то, как вы можете начать с ним работу.

* **Создание/настройка/клонирование репозитория:**

Репозиторий является одним из самых сильных инструментов любого проекта с контролем версий. Для превращения любого каталога в репозиторий GIT поможет простая команда git init <каталог>. После выполнения команды, в месте где была выполнена команда должен появиться каталог с названием .git.

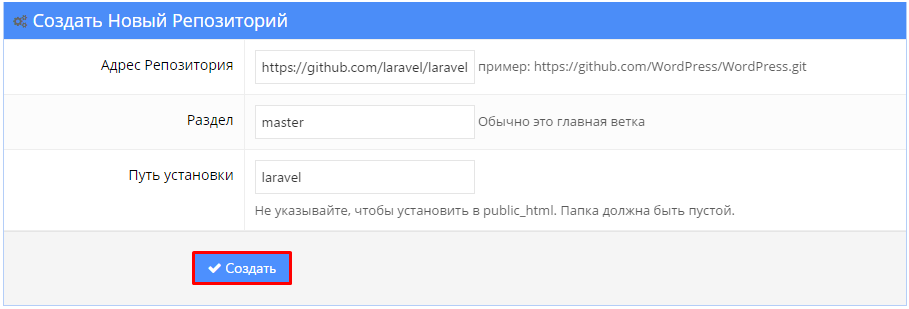
Если же у вас уже имеется каталог и вы хотите добавить (клонировать) его используйте команду git clone. Для клонирования локального репозитория используйте следующую команду:

git clone /путь/до/локального/репозитория

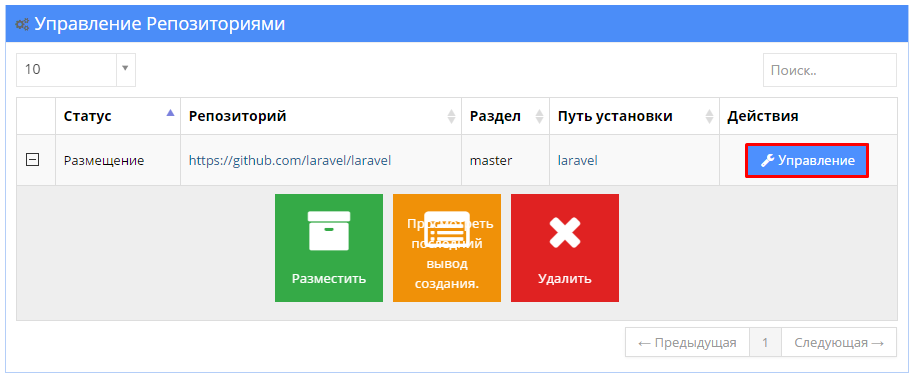
Если вы планируете клонировать удаленный репозиторий используйте:

git clone user.name@host:/путь/до/удаленного/репозитория

Если у вас есть учетная запись Hostinger, вы легко можете клонировать и управлять репозиторием через инструмент в **Панели управления** → **GIT**. К примеру, вы хотите клонировать репозиторий GIT, просто введите его адрес, выберите ветку, путь установки и нажмите кнопку **Создать**.



После завершения создания, вы сможете управлять своим репозиторием нажав кнопку **Управление**.



* **Рабочий процесс:**

Теперь, когда репозиторий настроен, давайте поговорим о его структуре. Каждый локальный репозиторий состоит из трех деревьев: **рабочего каталога**– который состоит из файлов, **индекса**– который играет роль места где проходят все изменения перед их применением и **HEAD** – являющегося указателем на последний коммит пользователя. Сам рабочий процесс можно объяснить таким образом: пользователь добавляет файл или изменение из рабочего каталога в индекс, далее после его просмотра и анализа оно применяется к **HEAD**.

* **Команды Add и Commit:**

Предлагаемые изменения или файлы добавляются в индекс используя команду add. Вот простая команда для добавления любого файла:

git add <имя\_файла>

Если вы достаточно уверены в изменении для его коммита к **HEAD**, вы можете использовать команду commit. Для этого используйте следующую команду:

git commit –m “Добавьте здесь любое сообщение описывающее коммит”

* **Дальнейшее продвижение изменений:**

После коммита ваших изменений (и если они готовы к отправке в оригинальный репозиторий), вы можете использовать команду push.

После выполнения команды git push origin master из рабочего каталога, все изменения присутствующие в **HEAD** будут отправлены на удаленный репозиторий. В вышеупомянутой команде, значение **master** должно быть изменено на название ветки для которой вы хотите произвести коммит изменений.

Однако, если существующий репозиторий еще не был клонирован и вы хотите установить соединение между вашим репозиторием и удаленным сервером, сделайте это, запустив следующую команду:

git remote add origin <сервер>

После клонирования, любые сделанные изменения будут перенесены на соответствующий сервер.

* **Ветки:**

Еще одной превосходной и продвинутой особенностью GIT является возможность разработчикам и руководителям проектов создавать несколько независимых веток в рамках одного проекта. Основная цель ветки – это разработка функций, которые при этом находятся изолированно и не влияют друг на друга. Стандартная ветвь в любом проекте всегда является главной веткой. По мере необходимости, может быть создано любое количество веток, которые в конечном итоге могут быть объединены с главной веткой.

Новая ветка может быть создана с помощью данной команды:

git checkout -b feature\_n \*

**feature\_n** это имя ветки

Если вы хотите переключиться обратно к главной ветке, используйте следующую команду:

git checkout master

При помощи данной команды может быть удалена любая ветка:

git checkout -d feature\_n

Для того, чтобы сделать ветку доступной для других пользователей, вы должны отправить ее на удаленный репозиторий. Для этого введите данную команду:

git push origin feature\_n

* **Обновление и объединение:**

В случае если вам необходимо обновить ваш локальный каталог на его последнюю версию из удаленного репозитория, просто введите команду git pull.

Для объединения активной ветки с какой-либо другой воспользуйтесь этой командой: git merge feature\_n.

Используете ли вы pull или merge, GIT всегда пытается справиться с возникающими конфликтами самостоятельно, но это не всегда у него получается. В случае неудачи выполнения команды, пользователь должен разрешить возникший конфликт вручную. После того, как вы отредактировали файлы (для устранения конфликта), пометьте их как merged с помощью команды:

git add <имя.файла>

Если вы хотите просмотреть различия до слияния веток, может быть запущена следующая команда:

git diff <имя\_ветки\_источника><имя\_целевой\_ветки>

* **Маркировка:**

Перед выпуском обновлений или ревизий в программном обеспечении, рекомендуется создать для них метки или теги. Для выполнения данного действия в GIT, выполните данную команду:

git tag 1.1.0 1c2d2d56fa

**1c2d2d56fa** в вышеприведенной команде относится к первым 10 символам commit-id, на которые ссылается тег. Идентификатор коммита или Commit ID можно найти в журнале.

* **Журнал:**

Изучить историю репозитория можно заглянув в журнал. Команда для вызова журнала git log. Для отображения коммитов сделанных конкретным пользователем, используйте:

git log --author =Имя

Сжатую версию журнала (один коммит на строке) можно просмотреть при помощи команды :

git log --pretty=oneline

Для просмотра измененных файлов:

git log --name-status

* **Замена локальных изменений:**

Если вы каким-то образом навели беспорядок и хотите отменить изменения, сделанные в файле, произведите этот процесс с помощью следующей команды:

git checkout -- <имяфайла>

Выполнив эту команду вы замените измененные данные в рабочем дереве на последние данные находящиеся в **HEAD**. Любые изменения, которые уже были добавлены в индекс не пострадают.

И наоборот, если все локальные изменения или фиксации должны быть удалены, а локальная главная ветвь должна быть привязана к последней истории с сервера, выполните следующие команды:

git fetch origin

git reset --hard origin/master

**Другие среды разработки.**

Программное обеспечение JetBrains WebStorm представляет собой инструмент для разработки web-сайтов и редактирования HTML, CSS и JavaScript кода. Решение обеспечивает быструю навигацию по файлам и генерирует уведомления о возникающих проблемах в коде в режиме реального времени. JetBrains WebStorm позволяет добавлять разметку HTML-документов или элементов SQL непосредственно в JavaScript. JetBrains WebStorm осуществляет развертывание и синхронизацию проектов через протокол FTP.   
  
Используя возможности кода HTML/XHTML и XML, WebStorm обеспечивает автоматическое завершение стилей, ссылок, атрибутов и других элементов кода. При работе с CSS осуществляется завершение кода классов, HTML-номеров, ключевых слов и т. д. WebStorm предлагает автоматическое решение таких проблем, как выбор формата, свойств, классов, ссылок на файлы и других атрибутов CSS. Решение позволяет использовать мощность инструмента Zen coding для верстки HTML, отображает действия тэга на web-странице. Продукт WebStorm осуществляет завершение кода JavaScript для ключевых слов, лейблов, переменных, параметров и функций DOM и поддерживает специфические особенности популярных браузеров. Реализованные в решении функции рефакторинга JavaScript позволяют преобразовывать структуру кода и файлов и .js.   
  
WebStorm обеспечивает отладку кода JavaScript и предоставляет широкий диапазон возможностей: нахождение точки останова в HTML и JavaScript, настройка параметров точки останова, тестирование синтаксиса кода в режиме реального времени и т. д. Продукт поддерживает платформы JQuery, YUI, Prototype, DoJo, MooTools, Qooxdoo и Bindows. WebStorm предусматривает интегрированную проверку текста на тэги, последовательность кода, ошибки в написании и т. д. WebStorm позволяет редактировать файлы и автоматически синхронизировать их по требованию при удаленной работе или хранении.   
  
Продукт поддерживает функцию контроля версий и предварительных вариантов кода и фиксирует все произведенные действия и изменения. Благодаря созданию истории в WebStorm можно восстанавливать кодовые выражения, блоки и даже целые файлы.