**Подключение CSS**

**Связывание таблиц стилей с HTML-кодом:**

**HTML5:**

<link rel="stylesheet" href="css/styles.css">

HTML 4.01 похож на предыдущий, но требует дополнительного атрибута type:

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

XHTML требует в конце тега ставить закрывающий слеш:

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

rel="stylesheet" — указывает тип ссылки; в данном случае это ссылка на табли­цу стилей;

href — задает местонахождение внешнего CSS-файла на сайте. Значение этого атрибута — URL-адрес, который будет отличаться в зависимости от того, где вы храните CSS-файл. Он работает так же, как атрибут src при добавлении изображения на страницу или атрибут href гиперссылки, указывающей на другую веб-страницу.

**Прикрепление таблиц стилей с использованием CSS:**

CSS имеет встроенный способ привязки внешних таблиц стилей к коду HTML — правило @import. Его нужно добавить в HTML-тег <style>. Например:

<style type="text/css">

**@import url(css/styles.css);**

</style>

***ПРИМЕЧАНИЕ***

*Если используется тип документа HTML 4.01 или XHTML, нужно к открывающему тегу <style> до­бавить атрибут type="text/css", о чем уже упоминалось в одной из предыдущих врезок*

Чтобы выполнить привязку к внешнему файлу CSS, нужно использовать url вместо href и заключать путь к CSS-файлу в круглые скобки. Так, в рассмот­ренном выше примере css/styles.css — путь к внешнему CSS-файлу. Кавычки, в которые заключен URL, не обязательны. Таким образом, url(css/styles.css) и url("css/styles.css") будут работать одинаково.

Посредством нескольких правил @import, как и с помощью нескольких тегов <link>, можно присоединить любое количество внешних таблиц стилей:

<style type="text/css">

**@import url(css/styles.css);**

**@import url(css/forms.css);**

</style>

**Например**:

<style type="text/css">

@import url(css/styles.css);

@import url(css/forms.css);

**p { color:red; }**

</style>

Нужно поместить все правила @import перед CSS-стилями, как показано в при­мере. Веб-браузеры игнорируют любые таблицы стилей, импортируемые после CSS-правила, поэтому если изменить порядок показанного выше кода на обратный и стиль p появится первым, то браузер проигнорирует любые стили в таблицах стилей style.css или form.css.

**Аналог якоря (ankor)**

Сначала до­бавьте атрибут id в то место на странице, на кото­рое вы хотите ссылаться. Например, в указателе вы можете добавить тег <h2> с очередной буквой из алфавита. Добавьте соответствующий id к каждо­му тегу <h2>:

<h2 id="R">R</h2>. Чтобы создать ссылку в HTML, добавьте символ решетки и имя идентификатора в конец адреса URL — index.html#R. Эта ссылка указывает непосредственно на элемент с #R страницы index.html (использо­вание идентификатора таким способом оказывает действие, аналогичное применению якорного тега <a> в языке HTML: <a name="R">R</a>).

**Универсальный селектор**

Групповые селекторы можно рассматривать как подручное средство для примене­ния одинаковых свойств различных элементов. CSS предоставляет универсальный селектор \* для выборки всех тегов веб-страницы. Например, если вы хотите, чтобы все отображалось полужирным шрифтом, нужно добавить следующий код:

a, p, img, h1, h2, h3, h4, h5 ...... { font-weight: bold; }

Использование символа \* — более быстрый способ сообщить CSS о выборке всех HTML-тегов веб-страницы:

\* { font-weight: bold; }

Кроме того, вы можете задействовать универсальный селектор в составе селек­тора потомков: применяете стиль ко всем тегам-потомкам, подчиненным опреде­ленному элементу веб-страницы. Например, .banner \* выбирает все теги внутри элемента, имеющего атрибут class со значением .banner

**Селекторы Потомков**

**Предок.** Как описано в начале данного подраздела, HTML-элемент, который заключает в себе другие элементы, — это предок. Например <html> предок для <body> и <head>, в то время как <body> — предок для всех содержащихся в нем тегов: <h1>, <p>, <strong>.

**Потомок.** Элемент, расположенный внутри одного или более типов — **потомок**. <body> — потомок <html>, в то время как <p> — потомок *одновременно* и для <body>, и для <html>.

**Родительский элемент.** Он связан с другими элементами более низкого уровня и находится выше на дереве. <html> является родительским только для <head> и <body>. Тег <p> — родительский по отношению к <strong>.

**Дочерний элемент.** Элемент, непосредственно подчиненный другому элементу более высокого уровня, является дочерним. Оба тега — <h1> и <p> — дочерние по отношению к <body>, но <strong> не является дочерним для <body>, так как он расположен непосредственно внутри тега <p>, он является дочерним для этого тега.

**Создание селекторов потомков**

Селекторы потомков позволяют использовать дерево HTML, форматируя теги по-разному, в зависимости от того, где они расположены. Например, на веб-странице имеется <h1> и вы хотите выделить слово внутри этого заголовка с помощью тега <strong>. Проблема в том, что большинство браузеров отобразит все это полужирным шрифтом, поэтому всякий, кто просматривает веб-страницу, не сможет увидеть разницы между выделенным словом и другими частями заголовка. Создание селектора тега — не очень хорошее решение: так вы измените цвет всех вхождений тега <strong> веб-страницы. Селектор же потомков позволит вам изменить цвет тега только в том случае, если он расположен внутри заголовка первого уровня.

Решение проблемы выглядит следующим образом:

h1 strong { color: red; }  
В данном случае *любой* текст тега <strong>, находящегося внутри тега <h1>, будет выделен красным цветом, но на другие экземпляры веб-страницы этот стиль не повлияет.

.intro a { color: yellow; } – Применяет данный стиль ко всем ссылкам к находящиеся внутри элемента к которому применек класс .intro

**Создание модулей**

Если вставить атрибут class в открывающий <div>-тег — <div class="news">, то вы можете создать селекторы потомков, по-разному форматирующие HTML-теги внутри раздела новостей. Например,

.news h2 { color: red; }  
.news p { color: blue; }

Вы можете заключить каждую контактную информацию в ее собственный div-контейнер, а затем улучшить внешний вид элементов внутри этого div-контейнера с помощью атрибута class:

<div class="contact">  
 <p class="name">John Smith</p>  
 <p class="phone">555-555-1234</p>  
 <p class="address">1234 Elem St</p>  
</div>

Затем можно создать несколько селекторов потомков для придания стиля только этим элементам контактной информации:

.contact .name { font-weight: bold; }  
.contact .phone { color: blue ;}  
.contact .address { color: red; }

**ВАЖНО**:

Иногда в таблице стилей можно увидеть следующий код: p.intro Может показаться, что это похоже на селектор потомка, поскольку в нем присутствует как название HTML-тега, так и имя класса, но это не так. Между p и .intro нет пробела, значит, чтобы этот стиль работал, селектор класса intro должен быть применен конкретно к тегу <p> (<p class="intro">).  
Если добавить пробел, вы получите другой эффект:

p .intro { color: yellow; }

Это несколько иной вариант, выбирающий любой тег, стилизованный под класс .intro, который, в свою очередь, является потомком тега <p>. Иными словами, он не выбирает абзац, он выбирает другой тег внутри абзаца. В общем, чтобы максимально сохранить гибкость ваших стилей класса, лучше оставить в покое HTML-тег (иными словами, использовать только .intro вместо p.intro).

Если говорить подробнее, тогда:

p.intro em {color: blue;}

В данном случае применит атрибут только лишь в том случае, когда тегу p присвоен класс intro, если же класс присвоить к другому тегу (например, div или li), тогда стиль не примениться. То есть мы создаем обязательное условие для применения стиля к определенному тегу.

em {color: coral;} /\* применит стиль к любому em у которого класс не .intro \*/

p.intro em {color: blue;} /\* Применит стиль к em только в том случае если класс .intro присвоен тегу p \*/

.intro em {color: gold;} /\* применит стиль к любому тегу с классом .intro /

**Псевдоклассы и псевдоэлементы**

Иногда требуется выбрать фрагмент веб-страницы, в котором вообще нет тегов, но в то же время его достаточно просто идентифицировать. Это может быть первая строка абзаца или ссылка, на которую наведен указатель мыши. CSS предоставляет для этих целей *псевдоклассы* и *псевдоэлементы*.

**Стилизация ссылок**

a:link обозначает любую ссылку, по которой посетитель веб-страницы еще не переходил, даже если на нее не наведен указатель мыши. Это обычный стиль не посещенных гиперссылок.

a:visited является ссылкой, по которой посетитель веб-страницы уже переходил. Она сохраняется в истории браузера. Можно разработать для этого типа стиль, отличный от обычного, чтобы сказать посетителю, что он уже посещал эту страницу.

a:hover позволяет изменять вид ссылки, на которую посетитель навел указатель мыши. Вы можете добавить визуальные эффекты трансформации (переходов), которые служат для улучшения визуального восприятия, например, кнопки панели навигации.

a:active позволяет определить, как будет выглядеть ссылка во время выбора ее посетителем веб-страницы. Другими словами, это стиль во время кратковременного щелчка кнопкой мыши.

Кроме того, можно использовать псевдокласс :hover для применения стилей к элементам веб-страниц, отличным от ссылок. Например, вы можете использовать его для выделения фрагмента текста, заключенного в теги <p> или <div>, каким-либо стилевым эффектом в тот момент, когда посетитель веб-страницы перемещает указатель мыши над этим фрагментом. В этом случае вместо применения a:hover для добавления эффекта наведения указателя на ссылку вы можете создать стиль под названием p:hover, добавляющий эффект наведения указателя на абзац. А если вы хотите добавить стиль к тегам, применяя к ним специальный класс highlight, создайте стиль под названием .highlight:hover

**Стилизация фрагментов абзаца**

На этапе становления Интернета вопрос дизайна страниц и сайтов не стоял, никто и не думал о том, что они должны выглядеть красиво. Теперь же это важно. В CSS имеется два псевдоэлемента — :first-letter и :first-line. Их использование обеспечит вашим веб-страницам изящное оформление, которым печатные издания обладают уже на протяжении многих столетий.  
Псевдоэлемент :first-letter позволяет создавать буквицу — начальный символ абзаца, который выделяется из остального текста, как в начале книжной главы.  
Стилизация первой строки с помощью псевдоэлемента :first-line отличным от основного абзаца цветом притягивает посетителей веб-страницы изяществом.

**Примечание:**

В CSS3, чтобы отличить такие псевдоклассы, как :hover, от псевдоэлементов, добавлено еще одно двоеточие. Поэтому :fist-letter и :fist-line стали теперь выглядеть как ::fist-letter и ::fist-line. К счастью, чтобы сохранить поддержку более старых сайтов, браузер будет продолжать поддерживать версию псевдоэлементов с одинарным двоеточием. Это хорошо, потому что Internet Explorer 8 не понимает синтаксис с двумя двоеточиями, поэтому пока можно остановиться на одинарном двоеточии, поскольку все остальные браузеры используют их, как и раньше.

**Дополнительные псевдоклассы и псевдоэлементы**

**:focus.** Псевдокласс :focus функционирует подобно :hover с той лишь разницей, что :hover применяется, когда посетитель перемещает указатель мыши над ссылкой, а :focus — когда нажимают клавишу табуляции или щелкают кнопкой мыши на текстовом поле

Селектор :focus полезен в основном для обеспечения обратной связи c посетителями сайта. Например, для смены цвета фона заполняемого поля формы, чтобы указать, где именно нужно вводить данные (однострочные текстовые поля, поля ввода пароля, многострочные поля <textarea> — везде можно использовать :focus).

Этот стиль задает светло-желтый цвет любому текстовому полю, на котором посетитель щелкает кнопкой мыши или в которое переходит с помощью табуляции:

**input:focus { background-color: #FFFFCC; }**  
Селектор :focus задает стилевой эффект только на время, пока элемент находится в фокусе. Когда посетитель переходит к другому полю или в другое место веб-страницы, он, как и весь стилевой эффект CSS, исчезает.

**:before** Псевдоэлемент :before выполняет такую функцию, которая не присуща ни одному селектору: он позволяет добавлять сообщение, предшествующее определенному элементу веб-страницы.

Для работы псевдоэлемента нужно создать класс (скажем, .tip) и применить его к абзацам, которым должно предшествовать данное сообщение, например <p class="tip">. Добавьте текст сообщения в таблицу стилей:

p.tip:before {content: "ПОДСКАЗКА!" }

Всякий раз, когда браузеру встречается класс .tip внутри тега <p>, он будет добавлять перед абзацем сообщение ПОДСКАЗКА!

**:after.** Псевдоэлемент-селектор :before добавляет сгенерированное содержимое перед определенным элементом, а :after — после. Например, вы можете пользоваться им для добавления заключительных кавычек после процитированного материала.

**::selection.** Этот селектор, появившийся в CSS3, ссылается на элементы, которые посетитель выбрал на странице. Например, когда посетитель щелкает на тексте и проводит над ним указатель мыши, браузер выделяет этот текст и посетитель может затем скопировать текст. Обычно браузеры добавляют за текстом синий фон. Internet Explorer изменяет цвет текста на белый. Но вы можете управлять цветом фона и текста, определив этот селектор.

**Важно**: этот псевдоэлемент используется ТОЛЬКО с двоеточием ::selection и внутри него можно использовать лишь два атрибута: color: – задает цвет текста и background-color: - задает цвет выделения

p::selection {  
 color: red;  
 background-color: pink;  
 }

задает цвет выделения всем абзацам <p> если мы хотим, чтобы все элементы на странице выделялись одинаково, тогда нужно использовать вот такой код:

body \*::selection {

color: red;

background-color: pink;

}

**Селекторы атрибутов**

С помощью *селекторов атрибутов* вы можете выбрать теги с конкретными элементами. Вот пример, в котором выделены все теги <img> с атрибутами title:  
img [title]  
Первая часть селектора — название (**img**); атрибут идет далее в квадратных скобках — [title].  
CSS не ограничивает селекторы атрибутов названиями тегов: вы можете также комбинировать их с классами. Например, селектор .photo[title] выбирает все элементы стилевого класса .photo с HTML-атрибутом title.  
Если необходима более детальная выборка, можно найти элементы, которые имеют не только определенный атрибут, но и точное значение. Например, если вы хотите подсветить ссылки, указывающие на специфический URL, создайте селектор атрибута с броским стилем:

**a[href="http://www.cosmofarmer.com"] {  
color:red;  
font-weight:bold; }**

Например, чтобы выбрать только текстовые поля в форме веб-страницы, используют следующее выражение:  
**input[type="text"]**

Селектор атрибута — очень разносторонний элемент. Он не только позволяет находить теги с определенным значением атрибута (например, все поля формы с типом checkbox), но и выбирать элементы со значением атрибута, начинающимся с какого-либо значения, заканчивающимся им или содержащим его.

Представьте, что вы хотите создать стиль, который выделял бы внешние ссылки (ведущие за пределы вашего сайта), говоря пользователю: «Ты покинешь этот сайт, если выберешь ссылку». Принимая во внимание то, что абсолютные ссылки внутри собственного сайта не используются, мы устанавливаем, что любая внешняя ссылка будет начинаться с **http://** — первой части любой абсолютной ссылки.  
Тогда селектор будет выглядеть следующим образом:

**a[href^="http://"]**

Символы **^=** означают «начинается с», так что вы можете использовать этот селектор для форматирования любой ссылки, начинающейся с **http://**. С его помощью вы легко стилизуете ссылку, указывающую на **http://www.google.com** либо **http://www.sawmac.com** — любую внешнюю ссылку.

***ПРИМЕЧАНИЕ***  
Этот селектор не будет работать в случае защищенного SSL-соединения, то есть когда ссылка начинается **с https://**. Чтобы создать стиль, учитывающий данную проблему, вам понадобится групповой селектор:

**a[href^="http://"], a[href^="https://"]**

Встречаются также ситуации, когда необходимо выбрать элемент с атрибутом, заканчивающимся определенным значением.

**a[href$=".pdf"]**

А стиль целиком будет выглядеть так:

**a[href$=".pdf"] {  
background-image: url(doc\_icon.png) no-repeat;  
padding-left: 15px;  
};**

Символы **$=** означают «заканчивается на»

Например, вам нужно выделить фотографии сотрудников на сайте. Вы хотите, чтобы у всех фотографий был общий стиль, допустим зеленая рамка и серый фон. Один из способов сделать это — создать класс стиля, например .**headshot**, и добавлять вручную атрибут класса к соответствующим тегам **<img>**. Однако, если вы задали для фотографий последовательные названия, это не самый быстрый метод. Например, каждую фотографию вы называли, используя при этом слово **headshot**, — **mcfarland\_headshot**, **mccord\_headshot** и т. д. В каждом файле встречается слово **headshot**, поэтому и атрибут **src** тега **<img>** содержит это слово. Вы можете создать селектор специально для этих изображений:

**img[src\*="headshot"]**

Выражение переводится как «выберите все изображения, атрибут **src** которых содержит в любом месте слово **headshot**

Символы **\*=** означают «включают в себя»

**Селекторы дочерних элементов**

Подобно применению селекторов потомков, описанных ранее, в CSS можно форматировать вложенные элементы других тегов с помощью *селектора дочерних элементов*, который использует дополнительный символ — угловую скобку (>) — для указания отношения между двумя элементами. Например**, body > h1** выбирает любой тег <h1>, дочерний по отношению к **<body>**.

**:first-child.** Возвращаясь к аналогии с семейным деревом HTML, вспомним, что такое дочерний тег: это любой тег, непосредственно заключенный в другой тег. Псевдоэлемент **:first-child** позволяет выбирать и форматировать только первый из них, независимо от того, сколько их вообще может быть.

**h1:first-child**

Этот селектор применяется к любому тегу **<h1>**, являющемуся первым дочерним тегом. Но **:first-child** может вызвать путаницу. Например, если у вас имеется тег **<h1>** внутри тега **<div>**, то **h1:first-child** выберет оба **<h1>**, тот, который непосредственно находится внутри тега **<body>**, и тот тег **<h1>**, который находится внутри div-контейнера (поскольку этот <h1> является первым дочерним элементом div).

**:last-child.** Этот псевдоэлемент похож на рассмотренный ранее **:first-child**, но выбирает последний дочерний элемент. Например, для придания стиля последнему элементу списка нужно воспользоваться селектором **ul:last-child**

**:nth-child.** Этот групповой псевдоэлемент селектора очень полезен. С его помощью можно легко и просто стилизовать каждую вторую строку в таблице, каждый третий элемент в списке или придать свой стиль любому сочетанию чередующихся дочерних элементов. Для определения, какой из дочерних элементов нужно выбирать, этот псевдоэлемент селектора требует значения.  
Самым простым выбором является ключевое слово — либо **odd** (**нечетный**), либо **even** (**четный**), — позволяющее выбрать чередующиеся нечетные или четные дочерние элементы. Например, если нужно предоставить один фоновый цвет для каждой четной строки таблицы и другой фоновый цвет для каждой строки с нечетным номером, можно создать два следующих стиля:

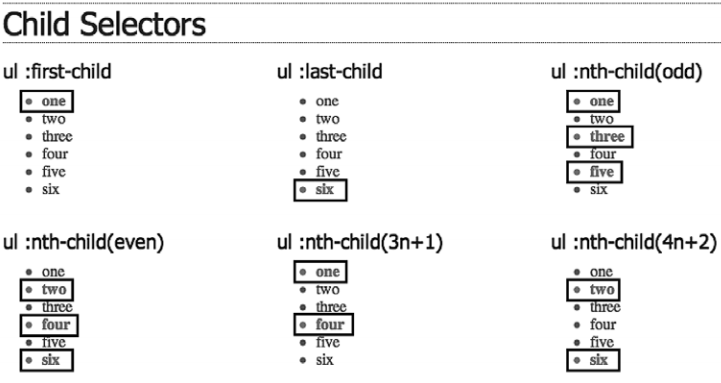
**tr:nth-child(odd) { background-color: #D9F0FF; }**

**tr:nth-child(even) { background-color: #FFFFFF; }**

Можно также выбрать, скажем, каждый третий дочерний элемент в серии, начиная со второго дочернего элемента. Представим, к примеру, что вам нужно выделить каждую третью ячейку таблицы (тег <td>) внутри строки, начиная со второй ячейки таблицы. Для этого необходимо применить следующий стиль:

**tr td:nth-child(3n+2) { background-color: #900; }**

Обычно число перед n (в данном случае **3**) представляет тот дочерний элемент, который будет выделен следующим по счету. Следовательно, 3n означает каждый третий элемент, а **4n** — каждый четвертый элемент. Знак «плюс» и следующее за ним число (**+2** в данном примере) показывают, с какого элемента нужно начинать выделение, таким образом, **+2** означает, что нужно начинать со второго дочернего элемента, а **+5** означает, что нужно начинать выделение с пятого дочернего элемента. Следовательно, использование псевдоэлемента **:nth-child(5n + 4)**



**Селекторы типов дочерних элементов**

Предположим, к примеру, что вам нужно определенным образом отформатировать первый абзац на боковой панели, но только на тех страницах, где эта боковая панель начинается с тега <h2>, и на других страницах, где она начинается с тега <p>. Псевдоэлемент :first-child для выбора этого абзаца применять нельзя, потому что в некоторых случаях абзац является вторым дочерним элементом (который следует за <h2>). Но тем не менее он всегда является первым абзацем (тегом <p>) на этой боковой панели, даже если перед ним следуют какие-нибудь другие теги, следовательно, его можно выбрать с помощью селектора типа, который называется **:first-of-type**.  
***ПРИМЕЧАНИЕ***  
*:last-child, :fist-of-type и :nth-child() поддерживаются всеми современными браузерами, включая Internet Explorer 9 и выше. Но в Internet Explorer 8 они не работают*.

**:first-of-type.** Работает так же, как и **:first-child**, но применяется к дочернему элементу, имеющему определенный тег. Предположим, например, что у вас есть боковая панель с классом **sidebar**. Для придания стилевого оформления первому абзацу этой боковой панели используется следующий селектор:

.**sidebar p:first-of-type**

Обратите внимание на **p** в **p:first-of-type**. Этот элемент обозначает тег, который вы собираетесь отформатировать.

**last-of-type.** Работает так же, как и **:last-child**, но применяется к последнему экземпляру тега определенного типа. Например, если нужно на боковой панели **div** определенным образом отформатировать последний абзац, но вы не уверены в отсутствии каких-нибудь других тегов, следующих за абзацем (например, списка, заголовка или рисунка). Этот стиль имеет следующий вид:

**sidebar p:last-of-type**

***ПРИМЕЧАНИЕ***  
следует помнить, что эти селекторы типа также должны быть дочерними по отношению к конкретному тегу. Следовательно, **p:fist-of-type** означает «первый дочерний элемент, являющийся тегом абзаца».

**:nth-of-type.** Работает так же, как и **:nth-child()**, но применяется к чередующимся дочерним элементам, имеющим определенный тег. Этот селектор может пригодиться при наличии больших абзацев текста с большим количеством фотографий. Тег **<img>** является встраиваемым тегом, поэтому у вас может быть **<p>**-тег с несколькими **<img>**- тегами внутри. И если вам захочется чередовать появление изображений, то слева, то справа, то это можно сделать с помощью двух стилей:

**img:nth-of-type(odd) { float: left; }  
img:nth-of-type(even) { float: right; }**

Как видите, для **:nth-of-type()** используются такие же ключевые слова (**odd** или **even**) и формулы (например, **2n +1**), как и для **:nth-child()**.

**Селектор :not()**

Селектор **:not()**, также известный как псевдокласс отрицания, позволяет выбрать что-нибудь, отличное от другого. Например, можно применить класс к абзацу — **<p class="classy">** — и создать селектор класса CSS, чтобы отформатировать этот абзац:

**.classy { color: red; }**

А что делать, если понадобится выбрать все абзацы, за исключением тех, что имеют класс **classy**? Именно здесь и пригодится селектор **:not()**. Чтобы показать, что не нужно выбирать, CSS-селектор помещается в круглые скобки. Например:

**p:not(.classy) { color: blue; }**

Этот стиль задает тексту синий цвет во всех абзацах, к которым не применялся класс classy. Селектор **:not()** может быть полезен при использовании селекторов атрибутов. Например, ранее было показано, что селектор атрибутов можно использовать, что-бы подобрать все ссылки, указывающие за пределы вашего веб-сайта:

**a[href^="http://"]**

Для выбора ссылки, которая указывает за пределы вашего сайта, нужно выбрать все абсолютные ссылки, не указывающие на домен mysite.com. Вот как это можно сделать:

**a[href^="http://"]:not([href^="http://mysite.com"])**

Если перевести на нормальный язык, этот селектор говорит: «Нужно выбрать все ссылки, чей атрибут **href** начинается с **http://**, но не те, которые начинаются с  [http://mysite.com»](http://mysite.com). То же самое можно написать еще короче:

**a[href^="http://"]:not([href\*="mysite.com"])**

В селекторе атрибута **\*=** означает «содержит», тем самым любой абсолютный URL, содержащий mysite.com, будет исключен.

***В отношении селектора* :not() *действуют несколько ограничений.***

- С селектором **:not()** можно использовать только простые селекторы. Другими словами, можно применять селекторы элементов (такие как **html** или **p**), универсальный селектор **(\*)**, классы (например .**footer**), идентификаторы ID (например, **#banner**) или псевдоклассы **(:hover, :checked, :first-child** и т. д.). Таким образом, все следующие селекторы можно считать правильными:

**footnote:not(div)  
img:not(.portrait)  
div:not(#banner)  
li:not(:first-child)**

- Нельзя применять к селекторам-потомкам (таким как **div p a**), псевдоэлементам (таким как **::first-line**), групповым селекторам или комбинациям (такиим как смежный селектор одного уровня **h2 + p**).  
- Нельзя в одной строке применять несколько **:not()**-селекторов. Например, следующий код будет неправильным:  
**a[href^="http://"]:not([href\*="google.com"]):not([href="yahoo.com])**Другими словами, **:not( )** с селектором можно использовать в одной строке только один раз.

**Селекторы смежных элементов одного уровня**

Предположим, вы хотите удалить отступ, который автоматически появляется перед тегом **<p>**, чтобы между заголовком и тегом не было никакого промежутка. Или желаете придать абзацу как небольшому вводному описанию другой цвет и размер шрифта.  
Селектор смежных элементов использует знак + для соединения одного элемента с другим. Поэтому, чтобы выбрать все первые абзацы, следующие за любым **<h2>**, используйте селектор **h2 + p** (пробелы необязательны, так что **h2+p** также будет прекрасно работать). Последний элемент в селекторе (в данном случае — **p**) — тег, который нужно отформатировать, но только при условии, что он следует сразу же за смежным для него тегом **<h2>**.   
Есть и другой селектор смежных элементов, который называется *общим сборным селектором смежных элементов*. Он обозначается знаком тильды **(~)** и означает следующее: «Нужно выбрать все смежные элементы этого типа». Например, если селектор **h2 + p** задает выбор отдельного **<p>**-тега, который следует сразу же за тегом **<h2>**, то селектор **h2 ~ p** определяет выбор всех смежных **<p>**-тегов, являющихся смежными элементами (то есть находящихся на одном и том же уровне) по отношению к заголовку.

Честно говоря, вы можете так и не найти этому селектору достойного применения, но в CSS, в принципе, весьма разнообразный синтаксис.

**Механизм наследования стилей**

Наследование — это прием, с помощью которого CSS-свойства, относящиеся к одному тегу веб-страницы, распространяются и на вложенные теги. Например, **<p>** всегда находится внутри **<body>**. Так, атрибуты, применяемые к **<body>**, наследуются **<p>**.

Механизм наследования — многоуровневый, то есть его эффект не только распространяется на прямых потомков (дочерние элементы), но и переносится на всевложенные элементы. Если, например, **<em>** и **<strong>** расположены внутри **<p>**, то  
также унаследуют атрибуты любого стиля, применяемого к **<body>**.

**Ограничения наследования**

Ниже описаны конкретные случаи, когда наследование точно не применяется

- Как правило, свойства, которые затрагивают размещение элементов на странице (отступы (поля), границы (рамки) элементов), не наследуются.

- Браузеры используют собственные встроенные стили для форматирования различных тегов. Заголовки обычно отображаются крупным полужирным шрифтом, ссылки — синим цветом и т. д. Даже если определен конкретный размер кегля для текстового содержимого веб-страницы и применен к **<body>**, заголовки будут отображены большим по размеру шрифтом. Теги **<h1>** будут крупнее **<h2>**. Точно так же, когда вы устанавливаете цвет применительно к  **<body>**, гиперссылки веб-страницы все равно будут показаны синим цветом с подчеркиванием.

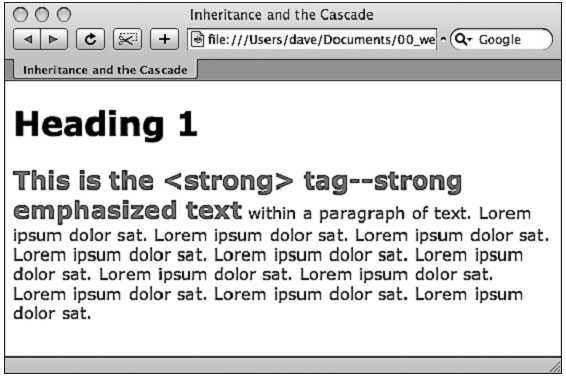
**Каскадность стилей**

***Каскадность*** — ряд правил, определяющих, какие именно стилевые свойства необходимы элементам веб-страницы, то есть задающих последовательность применения многократно определенных стилей. Другими словами, каскадность определяет, каким образом браузер должен обработать сложную структуру, относящуюся к одному и тому же тегу, и что делать, если возникает конфликт свойств. Это происходит в двух случаях: из-за механизма наследования (одинаковое свойство наследуется от нескольких родительских элементов-предков) и когда один или более стилей применяются к одному элементу веб-страницы (например, вы применили к абзацу стиль с помощью атрибута **class**, а также создали стиль для тега **<p>**, и оба стиля применяются к этому абзацу).

**Объединение унаследованных стилей**

Поскольку тег может унаследовать свойства *любого* предка (например, ссылка, наследующая шрифт родительского **<p>**), определить, почему конкретный тег отформатирован именно так, может быть сложной задачей. Предположим такую ситуацию: к **<body>** применен определенный шрифт, к **<p>** — размер шрифта, а для **<a>** — цвет. Таким образом, любой тег **<a>** абзаца унаследует свойства **<body>** и  **<p>**. Другими словами, унаследованные стили будут объединены, сформировав один сложный.  
На рисунке, имеется три стиля: один для **<body>**, второй для **<p>** и третий для **<strong>**. CSS-код выглядит следующим образом:

**body { font-family: Verdana, Arial, Helvetica, sans-serif; }  
p { color: #F30;}  
strong { font-size: 24px; }**



Тег **<strong>** вложен в абзац, который, в свою очередь, является вложенным в  **<body>**. Стилевые свойства наследует **<strong>** у всех элементов-предков, получая начертания шрифта **font-family** от **<body>** и цвет **color** от родительского абзаца. Кроме того, **<strong>** имеет собственное CSS-свойство, устанавливающее размер шрифта 24 пиксела. Конечный внешний вид тега определяется сочетанием всех трех стилей. Другими словами, **<strong>** выглядит так, будто для него создали следующий стиль:

**strong {  
font-family: Verdana, Arial, Helvetica, sans-serif;  
color: color: #F30;  
font-size: 24px;  
}**

**Превосходство близкого родительского элемента-предка**

Как видно из предыдущего примера, комбинирование стилей различных тегов с применением наследования создает полный стилевой пакет. Но что произойдет в случае конфликта унаследованных свойств CSS? Представьте себе страницу, где вы установили красный цвет шрифта для тега **body** и зеленый цвет шрифта для тега абзаца.  
На рис. выше **<strong>** отформатирован шрифтом определенного начертания, размера и цвета, несмотря на то что для него в стиле явно определено только одно свойство. Два других унаследованы от тегов-предков **<body>** и **<p>.**  
Теперь предположим, что внутри абзаца имеется тег **<strong>**. Тег **<strong>** наследует стиль как тега **body**, так и тега абзаца. Так какой текст внутри тега **<strong>** — красный или зеленый? Итак, у нас есть победитель: зеленый цвет от стиля абзаца.

**Преимущества непосредственно примененного стиля**

Единственный стиль, обладающий наивысшим приоритетом, является самым близким предком в «генеалогическом» дереве CSS. Это тот стиль, который напрямую применен к тегу. Предположим, что цвет шрифта устанавливается для **<body>**, **<p>** и **<strong>**. Стиль абзаца **<p>** является определенным по отношению к **<body>**, но **<strong>** — еще более конкретный, чем все остальные. Он форматирует только теги **<strong>**, отменяя любые конфликтующие свойства, унаследованные от других тегов. ***Другими словами, свойства стиля, явно определенного для тега, отменяют любые унаследованные.***

**Множество стилей для одного тега**

К тегу одновременно применен стиль с селектором и стилевой класс. Например, для тега **<h2>** это **.leadHeadline**. HTML-код имеет вид: **<h2 class="leadHeadline">Your Future Revealed!</h2>**. К **<h2>** ***будут применены оба стиля*.**

**Одинаковое название стиля встречается несколько раз в таблице**. Это можетбыть групповой селектор. Например, **.leadHeadline**, **.secondaryHeadline**, **.newsHeadline** и стилевой класс **.leadHeadline** в той же таблице. Форматирование элемента **leadHeadline** ***будет определяться обоими стилями*.**

**К тегу одновременно применены стилевой класс и ID-стиль**. Это может быть ID-селектор **#banner** и класс **.news**. HTML-код имеет вид: **<div id="banner" class="news">**. К **<div>** ***будут применены оба стиля***.

**С веб-страницей связано несколько таблиц, и в каждой из них содержится одинаковое название стиля**. Стили с одинаковыми названиями могут появляться во внешней таблице стилей и во внутренней таблице стилей. Или же один и тот же стиль может появиться в нескольких внешних таблицах стилей, каждая из которых связана с одной и той же страницей.

**Единственный тег веб-страницы может быть объектом воздействия сложных селекторов.** Это обычная ситуация, когда вы используете производные селекторы. Допустим, на веб-странице имеется тег **<div>** (например: **<div id="mainContent">**) и внутри него заключен абзац со стилевым классом: **<p class="byline">**. Здесь будут применены следующие селекторы:

* **#mainContent p;**
* **#mainContent .byline;**
* **p.byline;**
* **.byline.**

Если к конкретному элементу веб-страницы применено несколько стилей, то ***браузер объединяет их свойства*** ***при условии, что они не конфликтуют между собой***. Приведенный ниже пример разъясняет этот принцип. Предположим, есть абзац, в котором указаны имя автора веб-страницы и ссылка на адрес его электронной почты. HTML-код может выглядеть следующим образом:

**<p class="byline">Written by <a href="*mailto:jean@cosmofarmer.com*">Jean  
Graine de Pomme</a></p>**

Между тем в таблице стилей веб-страницы есть три стиля для форматирования гиперссылки:

**a { color: #6378df; }  
p a { font-weight: bold }  
.byline a { text-decoration: none; }**

Первый стиль окрашивает элемент тега **<a>** в зеленовато-голубой цвет; второй стиль форматирует все **<a>**, находящиеся в **<p>**, полужирным шрифтом; а третий стиль убирает подчеркивание ссылок, вложенных в элементы, которые принадлежат стилевому классу **byline**.

**ПРИМЕЧАНИЕ**  
Имейте в виду, что на стиль форматирования этой ссылки также могут влиять переданные свойства.  
Например, может быть унаследовано начертание шрифта абзаца.

**Особенности механизма каскадности: какие стили имеют преимущество**

Предыдущий пример слишком прост. Но что получится, если каждый из трех стилей, приведенных выше, имеет свое определение начертания в свойстве **font-family**? Какой из них выберет браузер? Если вы внимательно читали книгу, то помните, что механизм каскадности устанавливает несколько правил. ***Побеждают (имеют преимущество) свойства самого близкого по отношению к стилизуемому элементу, самого явно определенного стиля****.* Однако, как и в примере со стилями, не совсем понятно, какой из них является наиболее определенным. К счастью, CSS предлагает метод определения ***приоритетов***.  
Он основан на присвоении значений в условных единицах (пунктах) каждому типу стилевых селекторов: тегов, классов, ID-селекторам и т. д. Система работает так.

* Селектор тегов имеет значимость 1 пункт.
* Селектор классов — 10 пунктов.
* ID-селектор — 100.
* Встроенный (**inline**) стиль — 1000.

***ПРИМЕЧАНИЕ***

Математические расчеты, используемые для определения приоритетов, на самом деле немного более сложные. Но эта формула работает во всех случаях, кроме самых странных и запутанных.

Чем больше числовое значение, тем выше значимость данного типа селектора. Предположим, вы создали три стиля:

* теговый стиль для **<img>** (значимость = 1);
* стилевой класс **.highlight** (значимость = 10);
* ID-стиль **#logo** (значимость = 100).

Веб-страница содержит следующий HTML-код:

**<img id="logo" class="highlight" src="logo.gif" />**. Если определить одинаковый атрибут во всех трех стилях (например, рамка **border**), то будет применено значение атрибута **ID**-стиля (**#logo**), как наиболее значимого.  
***ПРИМЕЧАНИ***Е  
***Псевдоэлемент*** (например, **:fist-child**) обрабатывается браузером как теговый селектор и имеет значимость 1 пункт. ***Псевдокласс*** (например, **:link**) рассматривается как класс и имеет значимость 10 пунктов.

Поскольку производные селекторы состоят из нескольких простых — например, **content p** или

**h2 strong**, — определить их значимость сложнее: необходимо найти суммарное значение их приоритетов



***ПРИМЕЧАНИЕ***  
Наследуемые свойства вообще лишены такого показателя, как значимость. Так, даже если тег унаследует, например, селектор **#banner**, то эти свойства в любом случае будут заменены теми, что непосредственно относятся к этому тегу.

**Разрешение конфликтов: побеждает последний стиль.** Два стиля могут иметь одинаковый приоритет. Конфликт значимостей может произойти в случае двойного определения одинаковых селекторов. У вас может быть селектор тега **<p>** во внутренней таблице и такой же во внешней. Или два различных стиля могут иметь одинаковый приоритет. В таком случае более значимым будет последний определенный стиль таблицы. Вот пример HTML-кода:

**<p class="byline">Written by <a class="email"  
href="*mailto:jean@cosmofarmer.com*">Jean Graine de Pomme</a></p>**

В таблице для веб-страницы, содержащей вышеприведенные абзац и гиперссылку, у вас будет два стиля:

**p .email { color: blue; }  
.byline a { color: red; }**

Оба стиля имеют значимость, равную 11 (10 — для названия класса и 1 — для селектора тега), и относятся к **<a>**. Конфликт этих стилей очевиден. Какой цвет выберет браузер для окрашивания гиперссылки в приведенном абзаце? Красный,

**Разрешение конфликтов: побеждает последний стиль.** Два стиля могут иметь одинаковый приоритет. Конфликт значимостей может произойти в случае двойного определения одинаковых селекторов. У вас может быть селектор тега **<p>** во внутренней таблице и такой же во внешней. Или два различных стиля могут иметь одинаковый приоритет. В таком случае более значимым будет последний определенный стиль таблицы. Вот пример HTML-кода:  
**<p class="byline">Written by <a class="email" href="*mailto:jean@cosmofarmer.com*">Jean Graine de Pomme</a></p>**  
В таблице для веб-страницы, содержащей вышеприведенные абзац и гиперссылку, у вас будет два стиля:

**p .email { color: blue; }  
.byline a { color: red; }**

Оба стиля имеют значимость, равную 11 (10 — для названия класса и 1 — для селектора тега), и относятся к **<a>**. Конфликт этих стилей очевиден. Какой цвет выберет браузер для окрашивания гиперссылки в приведенном абзаце? ***Красный, так как он последний в таблице стилей***.

Теперь представьте, что таблица стилей имеет следующий вид:

**.byline a { color: red; }  
p .email { color: blue; }**

В данном случае гиперссылка была бы красного цвета. Cтиль с селектором **.byline a** расположен в таблице после **p .email** , поэтому его свойства имеют преимущество. Рассмотрим, что произойдет, если имеются конфликтующие стили (или их свойства) во внешней и во внутренней таблицах стилей. В этом случае важна последовательность размещения на веб-странице (в HTML-файле). Если вы сначала добавляете внутреннюю таблицу, используя **<style>**, а только затем присоединяете внешнюю далее по тексту HTML, используя **<link>**, то будет применен стиль последней (в сущности, это принцип, который только что был описан: значение имеет последний из конфликтующих стилей). Вывод: будьте последовательны в размещении на веб-странице внешней таблицы стилей. Сначала ее нужно присоединить, а только затем включать внутренние таблицы, когда вам абсолютно невозможно обойтись без одного или нескольких стилей, применяемых к одной странице.

***СОВЕТ***  
Любые внешние таблицы стилей, присоединяемые директивой **@import**, должны находиться довнутренних, заключенных в тег **<style>**, и до любых стилей во внешней таблице стилей. Для получения дополнительной информации о внешних и внутренних таблицах стилей.

**Отмена правил значимости**  
CSS предоставляет возможность полностью отме­нить значимость стилей. Вы можете использовать этот прием, чтобы никакой другой более значимый стиль не отменил конкретное свойство элемента веб­страницы. Просто вставьте после него значение **!important**. Рассмотрим пример. У нас есть два стиля:

**.nav a { color: red; }  
a { color: teal !important; }**

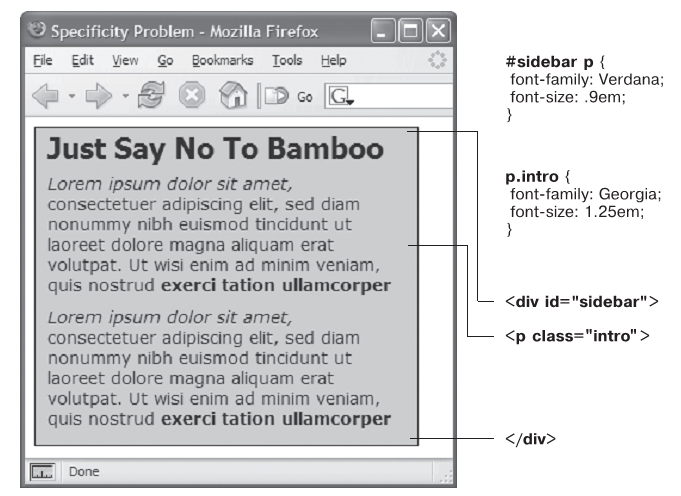
При обычном раскладе ссылка, вложенная в элемент с классом **.nav**, была бы окрашена в красный цвет, так как стиль, определенный селектором **.nav а**, более значимый, чем **<a>**. Однако добавление **!important** подразумевает, что данное свойство всегда будет иметь больший приоритет. Так, в вышеприведенном приме­ре все ссылки веб­страницы, в том числе вложенные с классом **nav**, будут отображены зеленовато­голубым цветом.  
Обратите внимание, что вы применяете **!important** к отдельному свойству, а не ко всему стилю, поэтому нужно добавить **!important** в конце каждого свой­ ства, которое не должно быть отменено. В заключение нужно сказать: когда для двух одинаковых свойств различных стилей указано **!important**,

**Управление каскадностью**

Как вы могли заметить, чем больше CSS-стилей создано, тем выше вероятность запутаться в них. Например, можно создать стилевой класс, устанавливающий для шрифта определенное начертание и размер, но применение его к абзацу ни к чему не приведет. Эта проблема обычно связана с механизмом каскадности. Даже когда вы абсолютно уверены в конечном результате, все равно может существовать стиль с большей значимостью. Есть несколько вариантов решения этой проблемы. Можно использовать **!important** (как описано в выделенном блоке выше), чтобы гарантировать применение конкретного свойства. Этот способ не совсем удобен, так как трудно.

**Изменение значимости**

На рисунке приведен пример, когда определенный стиль тега проигрывает в каскадной игре. К счастью, в большинстве случаев можно запросто изменить значимость одного из конфликтующих стилей и приберечь **!important** для понастоящему безысходных случаев. Здесь два стиля форматируют первый абзац. Стилевой класс **.intro** не столь значимый, как **#sidebar p**. Таким образом, свойства **.intro** не будут применены к абзацу. Чтобы увеличить значимость, добавьте к стилю идентификатор **#sidebar .intro**



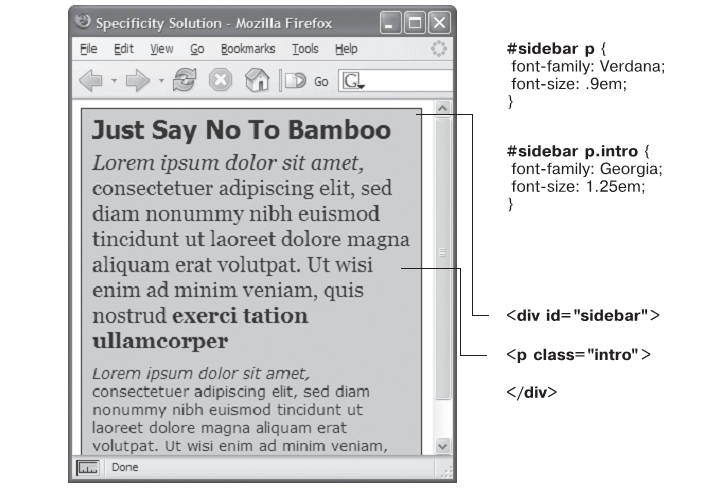
На рисунке  абзац размещен внутри тега **<div>** с идентификатором **#sidebar**. Таким образом, производный селектор **#sidebar p** является более значимым по сравнению со стилевым классом **.intro** (соотношение значимостей — 101 к 10). Вывод: необходимо придать **.intro** большую значимость, добавив перед ним идентификатор, как на рисунке который представлен ниже:

**#sidebar p.intro**.

Но простое добавление дополнительных селекторов с целью «победить» свойства стиля может привести к так называемым войнам значимости, когда у вас в конечном итоге получаются таблицы стилей, содержащие очень длинные и запутанные имена стилей вроде: **#home #main #story h1**. Нужно избегать стилей такого типа и стараться, чтобы селекторы были как можно короче.

**Выборочная отмена значимости**

Можно точно проработать дизайн веб-страниц, выборочно отменяя стили. Предположим, вы создали внешнюю таблицу и назвали ее **styles.css**, связав со всеми страницами сайта. Этот CSS-файл содержит общие определения стилей дизайна страниц: шрифт и цвет для тегов заголовков **<h1>**, стиль элементов форм и т. д. Возможно, вы хотите, чтобы на главной (домашней) странице **<h1>** выглядел отлично от того, как он отображен на остальных. Например, был выделен крупным полужирным шрифтом или шрифтом меньшего размера, чтобы вместить больший объем информации. Другими словами, вы все еще хотите использовать ***большинство***стилей файла **styles.css**, но необходимо отменить несколько атрибутов отдельных тегов (**<h1>,** , **<p>** и т. д.).



Один из способов сделать так заключается в простом создании внутренней таблицы, содержащей стили, которые вы хотите отменить и переопределить. Предположим, в файле **styles.css** имеется следующий стиль:

**h1 {  
font-family: Arial, Helvetica, sans-serif;  
font-size: 24px;  
color: #000;  
}**

Вы хотите, чтобы заголовок **<h1>** главной веб-страницы был отображен крупным шрифтом красного цвета. Просто добавьте во внутреннюю таблицу следующий стиль:

**h1 {  
font-size: 36px;  
color: red;  
}**

В данном случае к тегу **<h1>** главной страницы сайта будет применен шрифт Arial (из внешней таблицы стилей), но в то же время он будет окрашен в красный цвет и будет иметь размер 36 пикселов (эти параметры определены во внутренней таблице стилей).

***СОВЕТ***  
Убедитесь, что вы присоединяете внешнюю таблицу стилей перед внутренней в разделе HTML-заголовка **<head>**. Это гарантирует, что нужные стили будут иметь преимущество в тех случаях, где значимость одинаковая.

Другой метод заключается в создании еще одной внешней таблицы. Например, таблицы home.css, которую нужно будет присоединить к главной веб-странице в дополнение к **styles.css**. Файл **home.css** будет содержать стили **styles.css**, которые вы хотите переопределить. Для правильной работы файл **home.css** должен быть присоединен ***после*****styles.css** в HTML-коде главной веб-страницы:

**<link rel="stylesheet" href="css/ styles.css">  
<link rel="stylesheet" href="css/home.css">**

***СОВЕТ***  
Еще один способ выполнить точную постраничную стилизацию веб-страниц основан на использовании различных имен класса для тега **<body>** веб-страниц разного содержания. Например, чтобы изменить дизайн отдельных веб-страниц, применяются идентификаторы **.review**, **.story**, **.home**, а затем создаются производные селекторы.

**Как избежать войн значимости**

Как уже упоминалось, теперь многие веб-дизайнеры предпочитают вместо ID-селекторов использовать классы. Одна из причин состоит в том, что ID-селекторы обладают очень большой значимостью, в силу чего для их отмены требуется более высокая значимость. Зачастую это приводит к войнам значимости, при которых таблицы стилей загружаются с излишне пространными и сложными селекторами. Суть этой проблемы проще объяснить на примере. Предположим, что в вашей странице есть следующий фрагмент кода HTML:

**<div id="article">  
<p>A paragraph</p>  
<p>Another paragraph</p>  
<p class="special">A special paragraph</p>  
</div>**

Вы решили, что нужно сделать текст абзаца внутри **div**-контейнера **article** красным, и создали следующий селектор потомка:

**#article p { color: red; }**

Но затем вам захотелось, чтобы текст одного абзаца с классом **special** был синим. Если просто создать селектор класса, вы не получите желаемого результата.

**.special { color: blue; }**

Как уже говорилось, когда определяется, какое из свойств нужно применить к тегу, веб-браузер использует для разрешения стилевых конфликтов простую математическую формулу: браузеры дают ID-селектору значение 100, селектору класса — значение 10, а селектору тега  —значение 1. Поскольку селектор **#article p** составлен из одного идентификатора и одного элемента (суммарный показатель значимости  —101), он заменяет собой простой стиль класса, заставляя вас изменить селектор:  
**#article .special {color: blue; }**  
К сожалению, это изменение является причиной возникновения еще двух проблем. Во-первых, селектор становится длиннее, и во-вторых, теперь этот синий цвет применяется, только когда класс **special** появляется внутри какого-нибудь элемента с идентификатором **article**. Другими словами, если вы скопируете HTML-код **<p class="special">A special paragraph</p>** и вставите его в какое-нибудь другое место страницы, текст уже не будет синим. То есть использование идентификаторов делает селекторы не только длиннее, но и бесполезнее. А теперь посмотрим, что получится, если просто заменить все идентификаторы классами. Предыдущий код HTML приобретет следующий вид:

**<div class="article">  
<p>A paragraph</p>  
<p>Another paragraph</p>  
<p class="special">A special paragraph</p>  
</div>**

**И код CSS можно заменить следующим:**

**.article p { color: red; }  
p.special { color: blue; }**

Первый стиль — **.article p** — является селектором потомка с показателем значимости 11. Второй стиль **p.special** также имеет показатель 11 (один тег и один класс) и означает «применить следующие свойства к любому абзацу с классом **special** ». Теперь, если вырезать этот код HTML и вставить его в какое-нибудь другое место страницы, вы получите синий текст, обусловленный стилем, к чему, собственно, мы и стремились. Это только один из примеров, но не составит особого труда найти таблицы стилей с абсурдно длинными селекторами вроде

**#home #article #sidebar #legal p** и **#home #article #sidebar #legal p.special** .  
По сути, использовать идентификаторы нет никакого смысла. Они не дают ни чего, что нельзя было бы получить с применением простого селектора класса или селектора тега, а их высокая значимость может только привести к ненужному усложнению таблиц стилей.

**Начинаем с чистого листа**

Для предотвращения кросс-браузерного несоответствия лучше всего начинать таблицу стилей с чистого листа. Другими словами, удалить встроенное в браузер форматирование и добавить собственное. Концепция устранения стилей браузера называется *сбросом стандартных стилей (CSS-сбросом)*

Рассмотрим шаблон сброса стандартных стилей:

html, body, div, span, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre, a, abbr, acronym, address, big, cite, code, del, dfn, em, img, ins, kbd, q, s, samp, small, strike, strong, sub, sup, tt, var, b, u, i, center, dl, dt, dd, ol, ul, li, fieldset, form, label, legend, table, caption, tbody, tfoot, thead, tr, th, td, article, aside, canvas, details, embed, figure, figcaption, footer, header, hgroup, menu, nav, output, ruby, section, summary, time, mark, audio, video {  
margin: 0;  
padding: 0;  
border: 0;  
font-size: 100%;  
vertical-align: baseline;  
}  
article, aside, details, figcaption, figure, footer, header, hgroup, menu, nav, section {  
display: block;  
}  
body {  
line-height: 1.2;  
}  
ol {  
padding-left: 1.4em;  
list-style: decimal;  
}  
ul {  
padding-left: 1.4em  
list-style: square;  
}  
table {  
border-collapse: collapse;  
border-spacing: 0;  
}

**Первый стиль** - очень длинный групповой селектор, затрагивающий наиболее распространенные теги и «обнуляющий» их. Он удаляет поля и отступы, устанавливая 100%-ный размер шрифта и убирая выделения полужирным. Благодаря этому шагу ваши теги смотрятся практически одинаково. Но так и нужно — ведь вы хотите начать с чистого листа, а затем добавить собственное форматирование, чтобы все браузеры согласованно отображали ваш HTML-код.

**Второй селектор** (**article, aside, details**…) является еще одним групповым селектором, помогающим устаревшим браузерам правильно отображать новые теги HTML5.

**Третий селектор** стиля (**body**) устанавливает однообразное значение свойства **line-height** (пространства между строками в абзаце)

**Четвертый и пятый** стили (для тегов **<ol>** и **<ul>**) устанавливают согласованные отступы от левого края и определенное форматирование (далее в книге будет представлена стилизация списков), а последний стиль упрощает добавление рамок к ячейкам таблицы (польза от установки этого стиля будет рассмотрена далее).

**ПРИМЕНЕНИЕ CSS**

**Текст.**

Подключение шрифта:

**@font-face {  
font-family: "League Gothic";  
src: url('fonts/League\_Gothic-webfont.ttf');  
}**

При использовании внутри директивы **@font-face** свойство **font-family** присваивает шрифту имя. Затем имя этого шрифта задействуется, когда данный шрифт нужно применить к стилю.

Пример.

**@font-face {  
font-family: "max mega font"; /\*здесь можно писать абсолютно любое название\*/  
src: url('fonts/max\_font.ttf'); /\*путь по которому лежит шрифт\*/**

**p {font-family: “max mega font”}**

**-------------------------------------------------------------------------------------------------------------------------------------------------------**

**Правильный импорт шрифта в CSS**

**@font-face {  
font-family: 'League Gothic';  
src: url('fonts/League\_Gothic-webfont.eot');  
src: url('fonts/League\_Gothic-webfont.eot?#iefix') format('embedded-opentype'),  
 url('fonts/League\_Gothic-webfont.woof') format('woff'),  
 url('fonts/League\_Gothic-webfont.ttf') format('truetype'),  
 url('fonts/League\_Gothic-webfont.svg') format('svg');  
}**

Это выглядит как какое-то нагромождение, и, к сожалению, из-за недостатка браузера Internet Explorer код неоправданно усложнен. Давайте в нем разберемся.

- **Строка 2** сохранила свой прежний вид. Свойство **font-family** предоставляет имя шрифта, это то же самое имя, которое вы будете использовать, применяя шрифт к своим CSS-стилям.

- **Строка 3** предназначена для **Internet Explorer 9**, но только при его работе в режиме совместимости (Compatibility mode), то есть в режиме, когда Internet Explorer 9  
работает наподобие Internet Explorer 8. Это странное свойство было добавлено в Internet Explorer 9, чтобы веб-сайты, приспособленные под недостатки Internet Explorer 8 и более ранних версий, неплохо выглядели в Internet Explorer 9. Пользователь должен преднамеренно переключиться в режим совместимости в Internet Explorer 9, поэтому лучше, наверное, оставить эту настройку в покое.  
- **Строка 4** начинается со второго свойства src, которое в соответствии с правилами директивы **@font-face** может иметь указания на несколько шрифтов. Первым опять указан шрифт EOT, но на этот раз вы видите, что к окончанию расширения файла **.eot** нужно добавить строку **?#iefix**. Это сделано, чтобы приспособиться к дополнительным недостаткам **Internet Explorer**, на этот раз дело касается **Internet Explorer 6–8**. Если после **.eot** не будет этой небольшой добавки к URL, то шрифт может неправильно отобразиться в Internet Explorer 8 или в более ранних версиях. После URL можно также заметить новый фрагмент кода:

**format('embedded-opentype')**

Он показывает формат шрифта и добавляется после каждого URL для различных форматов шрифтов.  
- **Строки 5–7** просто определяют дополнительные форматы шрифтов. Указано всего одно свойство — **src**, разбитое на несколько строк, чтобы его было легче читать. Для каждого типа шрифта, заданного в свойстве **src**, добавляется URL, формат и запятая (для всех, за исключением последнего шрифта): **url('fonts/League\_Gothic-webfont.woof') format('woff'),**

ПРИМЕЧАНИЕ  
В конце списка файлов для свойства **src** добавляется точка с запятой, чтобы обозначить конец свойства **src** (строка 7 в показанном выше примере). Об этой завершающей точке с запятой забывать нельзя, иначе директива **@font-face** не будет работать.

Даже если браузер способен понимать различные типы шрифтов **(например, Chrome может использовать шрифты WOFF, TrueType и SVG)**, он не станет загружать все файлы шрифтов. Вместо этого по мере чтения списка типов шрифтов браузер загружает **только первый шрифт**, который ему понятен. Иначе говоря, если браузеру Chrome попадется показанный выше код, он пропустит файл с расширением **EOT**, поскольку этот формат шрифта ему непонятен, а загрузит

файл с расширением WOOF. Затем он целиком пропустит файлы с форматами TrueType и SVG. Из этого следует, что порядок, в котором перечислены шрифты, играет важную роль. Формат WOOF в большинстве случаев более предпочтителен, поскольку его файлы меньше по размеру и загружаются быстрее, а формат SVG намного больше по размеру, поэтому нужно убедиться, что шрифты перечислены в следующем порядке: EOT, WOOF, TTF и SVG.

**Создание стилей с использованием веб-шрифтов**

h1 {  
font-family: 'League Gothic';  
font-weight: normal;  
}

Обратите внимание на употребленное здесь новое свойство font-weight. Обычно браузеры отображают содержимое тегов <h1> полужирным шрифтом. Большинство браузеров искусственно сделают шрифт полужирным, когда требуется такая версия шрифта. В результате получается уродливое полужирное начертание. Установка для свойства **font-weight** значения **normal** заставляет браузер просто использовать шрифт **League Gothic** таким, какой он есть, и избежать попыток сделать его полужирным.

СОВЕТ  
На веб-странице можно также использовать шрифты, содержащие различные символы и значки. То есть вместо создания, например, графического изображения предупреждающего значка и помещения его в тексте абзаца можно воспользоваться директивой @font-face для загрузки шрифта, содержащего значок предупреждающего знака, и применить обыкновенную букву (отображенную в этом шрифте как значок). Для использования шрифтов, содержащих различные значки, можно обратиться к нескольким ресурсам: http://css-tricks.com/flt-icons-icon-fonts/, http://css-tricks.com/html-for-icon-font-usage/ и <http://css-ricks.com/examples/IconFont/>

**Работа с полужирными и курсивными вариантами**

в случае с веб-шрифтами для каждого варианта шрифта вам понадобится отдельный файл. Итак, для основного текста веб-страницы нужны как минимум обычная версия шрифта, полужирная версия, курсивная версия и комбинированная  
полужирная и курсивная версия. Кроме того, для каждого варианта шрифта вы должны создать отдельную директиву @font-face.

**Простой способ добавления полужирного и курсивного вариантов**

Самый простой способ добавления полужирного и курсивного вариантов шрифтов заключается в добавлении к директиве @font-face свойств font-weight и font-style. Обычно CSS-свойство font-weight требует от браузера показать шрифт в полужирном, (bold), обычном (normal ) варианте или в одном из нескольких других вариантах плотности, а свойство font-style требует от браузера отобразить шрифт в курсивном(italic) или обычном (normal ) варианте. Но при использовании в директиве @fontface свойство font-style требует от браузера применить шрифт, когда стиль запрашивает конкретный вариант шрифта.

|  |  |
| --- | --- |
| @font-face { **font-family: 'PTSans';** src: url('PTSansRegular.eot'); src: url('PTSansRegular.eot?#iefix') format('embedded-opentype'), url('PTSansRegular.woof') format('woff'), url('PTSansRegular.ttf') format('truetype'), url('PTSansRegular.svg') format('svg'); **font-weight: normal;** **font-style: normal;** } | @font-face { font-family: 'PTSans'; src: url('PTSansItalic.eot'); src: url('PTSansItalic.eot?#iefix') format('embedded-opentype'), url('PTSansItalic.woof') format('woff'), url('PTSansItalic.ttf') format('truetype'), url('PTSansItalic.svg') format('svg'); font-weight: normal; **font-style: italic;** |
| @font-face { font-family: 'PTSans'; src: url('PTSansBold.eot'); src: url('PTSansBold.eot?#iefix') format('embedded-opentype'), url('PTSansBold.woof') format('woff'), url('PTSansBold.ttf') format('truetype'), url('PTSansBold.svg') format('svg'); **font-weight: bold;** font-style: normal; } | @font-face { font-family: 'PTSans'; src: url('PTSansBoldItalic.eot'); src: url('PTSansBoldItalic.eot?#iefix') format('embedded-opentype'), url('PTSansBoldItalic.woof') format('woff'), url('PTSansBoldItalic.ttf') format('truetype'), url('PTSansBoldItalic.svg') format('svg'); **font-weight: bold;** **font-style: italic;** } |

Иначе говоря, для охвата всех вариантов полужирного, курсивного и обычного текста вам нужны четыре директивы @font-face: обратите внимание на то, что для всех случаев используется одно и то же имя font-family, а изменяются только свойства src (чтобы указывать на разные файлы) и свойства font-weight и font-style. Преимущество этого метода состоит в том, что к тексту можно назначить обычный шрифт, применить к коду HTML теги <em> и <strong> и позволить браузеру самому выбирать файл шрифта для загрузки и использования. В данном примере, если нужно назначить шрифт PTSans всем абзацам, достаточно просто добавить этот стиль к своей таблице стилей:

p {  
 font-family: PTSans;

}

Эти директивы работают даже для заголовков. Если вы создаете стиль для форматирования всех тегов h1 с использованием шрифта PTSans, то он может иметь следующий вид:

h1 {  
font-family: PTSans;  
}

При наличии данного стиля веб-браузер будет фактически применять полужирную версию шрифта PTSans, поскольку заголовки обычно отображаются полужирным шрифтом. (При использовании данной технологии с привлечением нескольких вариантов шрифтов уже не нужно, как раньше, добавлять фрагмент font-weight: normal; .) К сожалению, Internet Explorer 8 и более ранние версии не понимают этого метода и будут использовать для всего текста шрифт PTSansRegular. Для тегов <em> и <strong> Internet Explorer будет создавать искусственный курсивный и искусственный полужирный варианты шрифтов, то есть он будет наклонять шрифт PTSansRegular на экране для получения курсива и делать шрифт PTSansRegular толще для получения полужирного варианта. Получающиеся в результате этого сгенерированные компьютером полужирный и курсивный варианты обычно выглядят неважно.

**Добавление полужирного и курсивного вариантов и поддержка Internet Explorer 8**

Для начала нужно по-прежнему создать четыре директивы @font-face, по одной для каждого варианта шрифта. Но вместо задания для них одного и того же имени семейства font-family (например, PTSans), каждому из них дается свое уникальное имя (PTSansRegular, PTSansItalic и т. д.). Иными словами, четыре директивы @font-face нужно переписать следующим образом:

|  |  |
| --- | --- |
| @font-face { **font-family: 'PTSansRegular';** src: url('PTSansRegular.eot'); src: url('PTSansRegular.eot?#iefix') format('embedded-opentype'), url('PTSansRegular.woof') format('woff'), url('PTSansRegular.ttf') format('truetype'),  url('PTSansRegular.svg') format('svg'); } | @font-face { **font-family: 'PTSansItalic';** src: url('PTSansItalic.eot'); src: url('PTSansItalic.eot?#iefix') format('embedded-opentype'), url('PTSansItalic.woof') format('woff'), url('PTSansItalic.ttf') format('truetype'), url('PTSansItalic.svg') format('svg'); } |
| @font-face { **font-family: 'PTSansBold';** src: url('PTSansBold.eot'); src: url('PTSansBold.eot?#iefix') format('embedded-opentype'), url('PTSansBold.woof') format('woff'), url('PTSansBold.ttf') format('truetype'), url('PTSansBold.svg') format('svg'); } | @font-face { **font-family: 'PTSansBoldItalic';** src: url('PTSansBoldItalic.eot'); src: url('PTSansBoldItalic.eot?#iefix') format('embedded-opentype'), url('PTSansBoldItalic.woof') format('woff'), url('PTSansBoldItalic.ttf') format('truetype'), url('PTSansBoldItalic.svg') format('svg'); } |

Заметьте, что у каждой директивы @font-face имеется собственное название семейства, которое соответствует варианту шрифта: PTSansRegular, PTSansItalic, PTSansBold и PTSansBoldItalic.

А теперь, к сожалению, вам придется применять разные имена шрифтов для различных тегов: для p — имя обычного шрифта, для em — имя курсивного шрифта, для strong — имя полужирного шрифта. А чтобы справиться со случаем применения полужирного курсива придется составлять селектор потомка. То есть, чтобы могли работать различные варианты шрифта PTSans, нужно создать четыре стиля, содержащие довольно много строк кода:

|  |  |
| --- | --- |
| p { font-family: PTSansRegular; font-size: 18px; font-style: normal; font-weight: normal; } | p em { font-family: PTSansItalic; font-style: normal; font-weight: normal; } |
| p strong { font-family: PTSansBold; font-style: normal; font-weight: normal; } | p strong em, p em strong { font-family: PTSansBoldItalic; font-weight: normal; font-style: normal; } |

Если применяются теги <b> и <i> тогда и для них нужно сделать те же манипуляции.

Нужно также обратить внимание на необходимость установки во всех этих стилях значения normal для свойств font-weight и font-style. Если этого не сделать, многие браузеры (не только Internet Explorer) будут пытаться сделать жирнее полужирный шрифт и применить курсив к курсивной версии шрифта (особенно уродливо смотрится искусственно усиленный жирный шрифт в Firefox).

**Шрифты от Google.**

**Standard.** Предоставляет тег <link>, указывающий на внешнюю таблицу стилей (это то же самое, что и создание ссылки на любую внешнюю таблицу стилей). Но на самом деле это особая ссылка, указывающая на веб-сервер Google и предоставляющая информацию, которая необходима Google для доставки нужных шрифтов. Например:

**<link href='http://fonts.googleapis.com/css?family=Codystar: 300,400|Gentium+Book+Basic:400,400italic'**

**rel='stylesheet' type='text/css'>**

Обратите внимание, что в конце атрибута href перечисляются шрифты и их стили. В данном примере используются шрифты **Codystar** и **Gentium**. А Google загрузит несколько стилей: **300** и **400** для Codystar,  **400** и  **400italic** для **Gentium**. Эти числа являются способом обозначения плотности (или толщины)

**@import.** Чтобы увидеть код, необходимый для использования директивы @import, нужно щелкнуть на вкладке @import. Преимущество данной директивы заключается в том, что ее можно добавить в начало другой таблицы стилей. Предположим, например, что у вас есть одна таблица стилей, которая предназначена для вашего сайта и на которую ссылаются все его страницы. Стандартный метод <link> требует добавления своего кода к каждой странице сайта. А используя директиву @import, можно добавить код к единственной внешней таблице стилей и закрыть вопрос.

После того как шрифты загружены, их можно использовать практически так же, как и любой другой шрифт. Нужно просто создать стиль, добавить свойство font-family и указать шрифт. Имя шрифта показано в нижней части страницы использования сайта веб-шрифтов Google

Если используется несколько стилей шрифта, нужно также добавить к стилю свойства font-weight и font-style. В Google обычные ключевые слова normal или bold для обозначения плотности шрифтов не используются. Вместо них применяется числовая шкала от **100** до **900**. Значение **700** соответствует варианту bold, 400 — варианту normal , а остальные числа обозначают другие варианты толщины. Предположим, например, что вам нужно применить обычную курсивную версию шрифта Gentium Book Basic к тегу <em>. Для этого можно создать следующий стиль:

**em {  
 font-family: "Gentium Book Basic", serif;  
 font-weight: 400;  
 font-style: italic;}**

**Придание тексту цветового оформления**

**color: #3E8988;** шестнадцатиричная система

**color: rgb(100%,100%,100%);**   
или RGB система   
**color: rgb(255,255,255);**

**color: rgba(255, 100, 50, .5);** RGBA последнее значение .5 означает альфа канал - прозрачтонсть

**color: hsl(0, 100%, 50%);**  HSL

**color: hsla(0, 100%, 50%, .5);** HDLA с альфапрозрачностью

**Внимание – IE 8 и ниже не поддерживают альфа прозрачность и заливают все черным !!!**

**Установка размера шрифта**

В большинстве случаев обычный текст, находящийся вне тегов заголовков, отобразится высотой 16 пикселов — это ***основной (базовый) размер шрифта текста***.

**em** размер буквы m, напечатанной шрифтом Cicero

font-size: 1em;

**Пикселы**

**font-size: 36px;**

**Ключевые слова.** CSS предлагает семь ключевых слов, которые позволяют назначить размер шрифта относительно базового: **xx-small , x-small , small , medium, large, x-large** и**xx-large**. CSS-код будет выглядеть следующим образом:

**font-size: large**;

Обычно **xx-small** является эквивалентом **9** пикселов (принимая во внимание, что вы не изменяли базовый размер шрифта в своем браузере), x**-small - 10** пикселам, **small  - 13**, **medium -16,** **large - 18**, **x-large- 24**, а **xx-large - 32** пикселам.

**Процентные значения.**

**font-size: 200%;** шррифт будет равен **32** пикселам, умножаем **16** на **200%**(или просто на 2) и получаем **32**.

Размер шрифта наследуется, поэтому нужно быть внимательным.

Пример

**<style>**

**p {font-size: 150%}**

**blockquote {50%}**

**</style>**

**<body>**

**<blockquote><p>test</p></blockquote>**

**</body>**

Для тега p установлен шрифт **16\*150%(**или **16\*1.5)=24** пиксела, но он заключен в тег blockquote у которого шрифт **16\*50%(16\*0,5)=8** пикселов который и наследует тег р и принимает эти **8** пикселов за **100%**, в следствии мы получаем отображение тега р в **8\*150%=12** пикселов

**Единица измерения еm.**

значение размера шрифта **1em** означает то же самое, что и **100 %,** как описано в предыдущем разделе. Процентное значение эквивалентно em, умноженному на **100**: .5em = 50% и т. д.

Например, этот CSS-код имеет тот же эффект, что и font-size: 200%;

**font-size: 2em;**

**Разбор вложенных конструкций**

Унаследованные значения размеров шрифтов могут вызвать проблемы для вложенных (многоуровневых) списков. Например, у вас имеется стиль ul { font-size: 75% } , а затем вы создаете вложен­ный список, то есть внутри <ul> расположены другиетеги. Получается, что для вложенного <ul> должен быть установлен размер шрифта, равный **75 %** от внешнего <ul>. Следовательно, подпункты списков будут отобра­ жены меньшим шрифтом, чем пункты и т. д. примени­тельно к подпунктам следующего подуровня. Чтобы решить эту проблему, создайте дополнительный стиль с селектором потомков: ul ul {font-size: 100%} . Этот стиль устанавливает размер шрифта любого <ul>, вло­женного в другой <ul>, равным **100 %**. Другими словами, **100 %** от размера шрифта родительского элемента ul , в который вложен другой элемент. В данном примере это обеспечивает сохранение размера шрифта вложенных подпунктов списков равным **75 %** от базового размера текста. Есть еще один способ предотвращения этого эффекта комнаты кривых зеркал, сжимающего текст. В CSS3 вклю­чена новая единица измерения под названием rem. Это название означает Root EM, то есть его значение основано на размере текста корневого (root) элемента. В боль­шинстве случаев это просто относится к базовому раз­меру текста, поэтому значение .75em, можно заменить на следующее значение:

font-size: .75rem;

Этот стиль задает размер шрифта равным **.75** от ба­зового размера текста, а не текущего размера шриф­та (как в случае применения единиц измерения em). Корневым элементом в HTML считается элемент html , который можно найти в начале веб­страницы. При ис­пользовании rem­ значений можно установить базовый размер текста элемента html , а затем использовать единицу измерения rem для установки размера тек­ста относительно этого базового размера. Например, можно установить базовый размер текста равным 20 пикселам:

**html {  
font-size: 20px;  
}**

А затем воспользоваться единицей измерения rem для создания размера шрифтов относительно этого **20** ­пиксельного базового размера текста, тогда, что­ бы задать для всех абзацев размер текста, равный **15** пикселам, нужно добавить следующий стиль:

**p {  
font-size: .75rem;  
}**

Но нужно иметь в виду, что такую единицу, как rem, понимают на данный момент все основные браузе­ры, кроме браузера Internet Explorer 8 и более ранних версий.

**Курсив и полужирный шрифт**  
Браузеры отображают текст, заключенный внутри тегов <em> и <i>, *курсивом* (шрифтом с наклонным начертанием), а текст, находящийся в тегах <strong>, <b>, <th> (**table header** — «заголовок таблицы»), тегах заголовков (<h1>, <h2> и т. д.) — **полужирным**.

Чтобы выделить текст курсивом, добавьте к стилю следующий код:

**font-style: italic;**

Или, наоборот, можете установить для текста обычный, не курсивый шрифт:

**font-style: normal;**

Чтобы сделать шрифт текста полужирным, наберите:

**font-weight: bold;** /\*Для Google fonts: font-weight: 600;(вместо 600 пишем необходимую толщину) \*/

Обычный шрифт устанавливается следующим образом:

**font-weight: normal;**

**Прописные буквы**

**text-transform: uppercase;** Все буквы станут большими (пример: ПРИВЕТ ЧУВАК, КАК ДЕЛА?)

**text-transform: lowercase;**  превратит первые буквы слов в строчные (привет чувак, как дела?)

**text-transform: capitalize;** превратит первые буквы слов в прописные (Привет Чувакб Как Дела?)

**font-variant: small-caps**; В этом стиле строчные буквы выглядят как немного уменьшенные заглавные.

**ПРИМЕЧАНИЕ**  
По какой-то странной причине браузеры Chrome и Safari не распознают свойство text-transform, когда оно используется с псевдоэлементом :fist-line. Другими словами, вы не можете применять в Chrome и Safari CSS для преобразования букв первой строки абзаца в прописные.

**Украшение текста**

**text-decoration: underline;** подчеркнутый текст

**text-decoration: overline;** надчеркнутый текст

**text-decoration: line-through;** перечеркнутый текст

**text-decoration: blink;** мерцающий текст (можно не применять практически нигде не пашет,)

можно оформлять так:

**text-decoration: underline overline;**

Отменить все элементы украшений можно путем применения ключевого слова none:

**text-decoration: none;**

Зачем вам может понадобиться отмена декорирования? Самый распространенный пример — удаление стандартной предопределенной подчеркивающей линии гиперссылки.

**Межсимвольный и межсловный интервал**

**letter-spacing: -1px;** Межсимвольный интервал, 0px - стандарт (-4px уменьшает отступ 4px увеличивает)

**word-spacing: 2px;** межсловный интервал 0px - стандарт (-4px уменьшает отступ 4px увеличивает)

С этими свойствами можно использовать любые единицы измерений, применимые к тексту: пикселы, em, проценты (с положительными или отрицательными значениями).

**Добавление текстовых теней**

**text-shadow: -4px 4px 3px #999999;**

Первое значение: **-4px** — означает «показать тень на **4** пиксела левее текста». (Положительное значение приведет к показу тени правее текста.) Второе значение — **4px** — задает отображение тени на **4** пиксела ниже текста. (Отрицательное значение приведет к показу тени над текстом.) Значение **3px** определяет степень размытости тени. Значение **0px** (без размытости) приводит к отбрасыванию четкой тени, и чем больше будет значение, тем более размытой будет тень. И наконец, И наконец, последнее значение определяет цвет отбрасываемой тени.

Для достижения более сложных эффектов можно даже создать несколько теней: нужно просто добавить запятую, а после нее дополнительные значения отбрасываемой тени:

**text-shadow: -4px 4px 3px #666, 1px -1px 2px #000, 5px 6px 3px rgba(181, 36, 36, .7);**

Количество добавляемых таким образом теней ничем не ограничивается (кроме вашего хорошего вкуса). К сожалению, этот эффект не работает в Internet Explorer 9 или более ранних версиях. Но он работает во всех других современных браузерах (даже в Internet Explorer 10). Иными словами, чтобы текст лучше читался, полагаться на этот эффект не стоит.

**Форматирование абзацев текста**

**Установка межстрочного интервала**

**ВАЖНО: Стандартный межстрочный интервал эквивалентен 120 %.**

**line-height: 150%;** можно использовать в **процентах, em, пикселах**.

Вообще, процентные значения или em лучше, нежели пикселы: размер, установленный в этих единицах измерения, напрямую зависит от параметров шрифта и автоматически корректируется пропорционально изменению свойства font-size. Так, если вы зададите межстрочный интервал **16** пиксельного шрифта равным **10** пикселам и затем измените на гораздо больший (например, на **36** пикселов), то высота останется равной **10**, а строки будут накладываться друг на друга. Однако при использовании значения размера в процентах (скажем, **150 %)** межстрочный интервал корректируется пропорционально всякий раз, когда вы изменяете значение размера шрифта соответствующего свойства font-size

**ПРИМЕЧАНИЕ**  
чтобы определить размер межстрочного интервала, браузер вычитает высоту шрифта из высоты строки. В результате получается размер межстрочного интервала (**leading**) между двумя соседними строками текста абзаца. Допустим, размер шрифта составляет **12** пикселов. Межстрочный интервал, установленный в размере **150 %**, в итоге равняется **18** пикселам. Таким образом, браузер добавляет пустой промежуток размером **6** пикселов между двумя строками текста.

**Межстрочный интервал в виде числового значения.**

**line-height: 1.5;**

После этого значения не нужно указывать никакую единицу измерения. Чтобы определить межстрочный интервал или высоту строки, браузер просто умножает это число на размер шрифта. Так, если размер шрифта текста составляет **1 em**, а высота строки установлена равной **1,5**, то расчетное значение межстрочного интервала равно **1,5 em**. В большинстве случаев эффект при указании значения **1,5** em или **150 %** ничем не отличается. Иногда множитель достаточно удобен. Особенно когда вложенные теги наследуют значения межстрочного интервала родительских тегов.  
Например, вы устанавливаете атрибут высоты строки <body> равным **150 %.** Все теги веб-страницы унаследуют это значение. Однако наследуется не процентное значение, а ***рассчитанное значение***межстрочного интервала. Допустим, основной размер шрифта задан равным **10** пикселам. **150 %** от **10** пикселов составляет **15**. Все теги унаследуют межстрочный интервал размером **15** пикселов. Так, если на веб-странице есть абзац текста со шрифтом высотой **36** пикселов, его межстрочный интервал размером **15** пикселов будет намного меньше самого текста, а строки сольются.   
В данном примере вместо высоты строки **150 %** для тега **<body>** лучше установить общий для всех тегов пропорциональный базовый межстрочный интервал в размере **1,5**. Любой тег, вместо того чтобы наследовать точное абсолютное значение высоты строки в пикселах от стиля body, просто умножает размер своего шрифта на этот  
коэффициент. Так, в вышеупомянутом примере, где абзац текста отображен размером шрифта **36** пикселов, межстрочный интервал составит **1,5  ⋅ 36 = 54** пиксела.

**Выравнивание текста**

**text-align: center;** выравнивание по центру

**text-align: right;** выравнивание по правому краю

**text-align: left;** выравнивание по левому краю

**text-align: justify;** выравнивание по ширине родительского элемента

**Метод стенографии для набора стилей, форматирования текста**

Многократный повторный набор стилевых свойств - остаточно утомительное занятие, особенно если нужно использовать несколько различных текстовых свойств сразу. К счастью, в CSS есть свойство **font**, облегчающее написание стилей. Оно позволяет объединять несколь­ко свойств в одну строку:

**font-style, font-variant,font-weight, font-size, line-height и font-family.**

Рассмотрим, например, следующее объявление:

**font: italic bold small-caps 18px/150% Arial, Helvetica, sans-serif;**

Оно приводит к созданию полужирного, курсивного шрифта с малыми прописными буквами размером **18 пикселов**, семейства Arial (или Helvetica, или sansserif ) с межстрочным интервалом **150 %**. Имейте в виду следующие правила:

• не обязательно применять все эти свойства, но нужно включить размер шрифта и начертание (се­мейства шрифтов):

**font: 1.5em Georgia, Times, serif;**

• используйте один пробел между каждым значе­нием свойства, а запятую только для того, чтобы отделить семейства шрифтов в списке:

**Arial, Helvetica, sans-serif;**

• определяя межстрочный интервал, добавляйте слеш после размера шрифта, за которым должно следовать значение самого межстрочного интервала: **1.5em/150** или **24px/37px**;

• последние два свойства должны быть комбинаци­ей: размер шрифта/межстрочный интервал (или font-size/line-height) и имени шрифта(например **Helvetica** ) ; все остальные могут быть перечислены в любом порядке. Например, оба объявления: - font: italic bold smallcaps 1.5em Arial и font: bold small-caps italic 1.5em Arial; — равнозначны и к ним применен одинаковый эффект.

Наконец, исключение значения из списка означает то же, что и установка его по умолчанию. Допустим, вы создали стиль <p>, который форматирует все абзацы полужирным курсивым шрифтом с малыми пропис­ными буквами и межстрочным интервалом **2000 %** (совсем не обязательно это повторять). Затем создали стилевой класс под названием, скажем, .specialParagraph с таким объявлением стиля шрифта: font: 1.5em Arial; — и применили его к какому­то абза­цу текста. В результате данный абзац не унаследует по­лужирное курсивое начертание с малыми прописными буквами и межстрочный интервал. Исключение этих четырех значений из стиля .specialParagraph можно приравнять к написанию следующего:

**font: normal normal normal 1.5em/normal Arial;**

**Отступ первой строки абзаца и удаление полей абзацев**

**Абзацный отступ.**

Для установки отступа первой строки абзаца можно использовать такие единицы измерения, как **пиксел** и **em**:

**text-indent: 25px;**

или

**text-indent: 5em;**

Значения в **пикселах** — абсолютные значения, точное число, в то время как **em** определяет размер отступа в количестве символов (базируется на текущем размере шрифта).

СОВЕТ  
В свойстве абзацного отступа **text-indent** вы можете использовать отрицательные значения для создания выступа, то есть абзаца с выступающей (смещенной, «свисающей») влево первой строкой. Обычно отрицательное значение абзацного отступа используется вместе с указанием значения поля, чтобы отрицательный абзацный отступ не выходил за левую сторону страницы, столбца или прямоугольной структуры.

Вы можете также применить **процентные** значения, но со свойством **text-indent** эти единицы измерения приобретают другое значение. В данном случае размер выступа, установленный в процентах, связан не со шрифтом текста, а с шириной элемента, в который заключен абзац. Например, если текстовый отступ установлен равным **50%** и абзац охватывает всю ширину окна браузера, то первая строка будет начинаться посередине экрана. Если вы меняете размеры окна, изменяется ширина абзаца и, соответственно, отступ (о значениях свойств, устанавливаемых в процентах, и о том, как они взаимодействуют с шириной элементов веб-страницы.

**Управление полями (границами) абзацев.**

Многие веб-дизайнеры не любят промежутки или дополнительные интервалы, которые любой браузер добавляет между абзацами. Чтобы полностью избавиться от верхнего и нижнего полей, введите следующее:

**margin-top: 0;  
margin-bottom: 0;**

Чтобы удалить поля между *всеми* абзацами веб-страницы, создайте такой стиль:

**p {  
 margin-top: 0;  
 margin-bottom: 0;  
}**

Для установки значений абзацных полей, как и для отступов, вы можете применять такие единицы измерения, как пикселы или **em**. Можно также использовать проценты, но, как и в случае с абзацными отступами, процентные значения относятся к ***ширине***элемента, в который заключен абзац. Во избежание путаницы, связанной с вычислением верхнего и нижнего полей, расчет которых базируется на ширине абзацев, проще применять значения в **em** или пикселах.

**ПРИМЕЧАНИЕ**  
поскольку не все браузеры обрабатывают верхнее и нижнее поля заголовков и абзацев согласованно, рекомендую просто обнулить (то есть удалить) все поля в начале таблицы стилей.

**Стилизация списков**

Большинство браузеров отображают маркированные списки (теги <ul>), используя маркеры в виде окружности, а нумерованные списки (<ol>) — предваряя пункты числами.

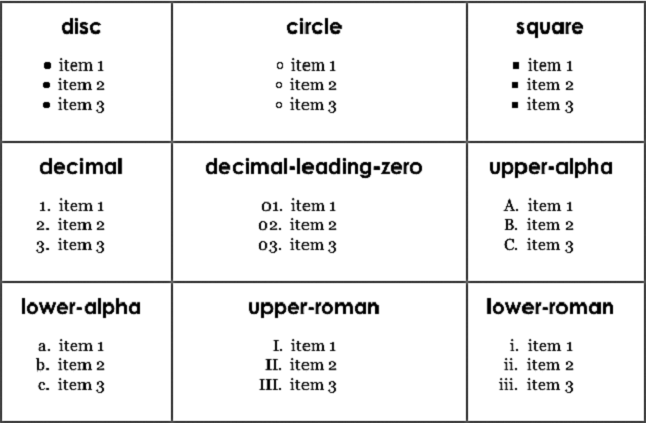
Для маркированных списков есть три типа маркеров: **dis*k***(круглый маркер с заполнением), **cirkle**(круг, полый круглый маркер), **square**(квадрат с зарисовкой). Для нумерованных списков предусмотрено шесть вариантов-схем  
нумерации: **decimal , decimal-leading-zero, upper-alpha, lower-alpha, upper-roman, lower-roman** (рис). Все эти варианты можно выбрать, используя CSS-свойство

**list-style-type:**

**list-style-type: square;**

или

**list-style-type: upper-alpha;**



Если нужно учесть чьи-нибудь особые предпочтения, числа можно заменить буквами греческого алфавита воспользовавшись настройкой lower-greek. Существует множество других схем нумерации, включая армянский, грузинский, катакана и другие региональные варианты. Информацию о них можно найти по адресу <https://developer.mozilla.org/en-US/docs/CSS/list-style-type>.

Тем не менее вы можете применить это свойство и к отдельно взятому элементу списка (<li>). Вы даже можете применить несколько стилей с различными маркерами к отдельным пунктам-элементам одного и того же списка.

Или, используя рассмотренный ранее селектор nth-of-type, можно вообще избавиться от имени класса:

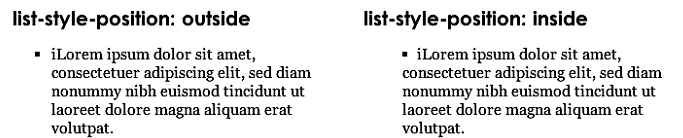
**li {list-style-type: square; }  
li:nth-of-type(odd) { list-style-type: circle; }**

Иногда может понадобиться вообще скрыть маркеры, в том числе тогда, когда вы захотите использовать собственные графические маркеры Кроме того, когда панель навигации сайта представляет собой список ссылок, вы также можете использовать список <ul>, скрыв его маркеры. Чтобы отключить отображение маркеров, используйте ключевое слово none:

**list-style-type: none;**

**Позиционирование маркеров и нумерации списков**

**list-style-position: outside;**  
 или  
**list-style-position: inside;**



СОВЕТ  
Вы можете изменить промежуток между маркером и текстом путем увеличения или уменьшения значения свойства **padding-left**. Чтобы использовать это свойство, нужно создать стиль для тега элементов списка **<li>**. Данный способ работает только в том случае, если свойство

**list-style-position** определено со значением **outside** (или вообще отсутствует)

Кроме того, если вам не нравится, что браузеры делают для списков отступ, смещая все содержимое вправо, можете переопределить стиль, установив значения свойств **margin-left** и **padding-left** равными 0. Чтобы удалить отступ, можно создать такой групповой селектор:

**ul, ol {  
 padding-left: 0;  
 margin-left: 0;  
 }**

Вы можете создать стилевой класс с такими свойствами и применить его к конкретным тегам **<ul>** или **<ol>**. Рекомендую установить собственные значения свойств **padding** и **margin** по той причине, что одни браузеры для управления отступом используют свойство **padding** (Firefox, Mozilla, Safari), а другие — **margin** (Internet Explorer).

В обычном порядке браузеры отображают пункты маркированных списков один над другим, без дополнительного промежутка. Добавить интервал между ними можно, применяя свойства

**margin-top** и **margin-bottom** к конкретным элементам списка. Они работают с интервалом элементов точно так же, как с абзацами.

**Графические маркеры**

CSS-свойство **list-style-image** позволяет определить путь к графическому символу на сайте таким же образом, как вы указываете местонахождение файла с изображением, используя атрибут

**src** HTML-тега **<img>**. Синтаксис команды  
следующий:

**list-style-image: url(images/bullet.gif);**

Термин **url** и круглые скобки обязательны. Часть, заключенная в круглые скобки, в данном примере **images/bullet.gif** - это и есть путь к графическому символу. Обратите также внимание на то, что, в отличие от HTML, не нужно заключать путь в кавычки.

**Настройка параметров маркеров и чисел в списках**

способ, экономящий объем CSS кода (не работает в IE 7 и ниже), состоит в том, чтобы использовать так называемое ***сгенерированное содержимое***. По сути, это все то, что не набирается вручную в виде кода веб-страницы, а автоматически добавляется браузером при отображе­нии страницы. Хороший пример — сами маркеры. Вы не вводите знаки маркера при создании списка — они добавляются на веб-страницу сами. С помощью CSS мож­но сообщить браузеру, чтобы он генерировал, добавлял такое содержимое и даже отформатировал должным образом все, что находится перед текстом пунктов списка — **<li>**. Если вы хотите сделать обычные маркеры рядом с элементами списка красными, добавьте к табли­це стилей следующий CSS:

**ul li {  
list-style-type: none;  
 }  
ul li:before {  
content: counter(item, disc) " ";  
color: red;  
 }**

А если нужно сделать красными элементы нумерованного списка, можно добавить следующий CSS:

**ol li {  
list-style-type: none;  
counter-increment: item;  
}  
ol li:before {  
content: counter(item) ". ";  
color: red;**

**}**

**ПРИМЕЧАНИЕ**  
Задавая путь или адрес к графическим файлам (изображениям, графическим символам) во внешней таблице стилей, имейте в виду, что **путь должен указываться относительно таблицы стилей**, а не веб-страницы.

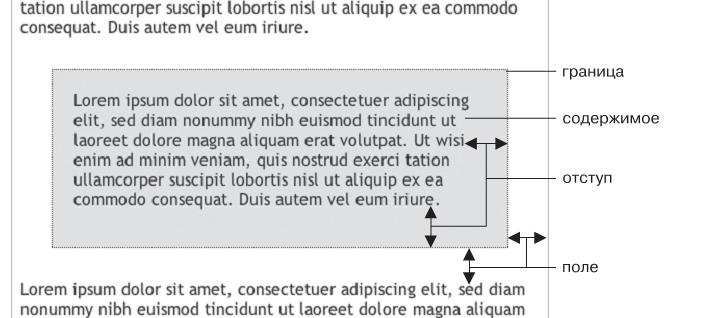
**СОВЕТ**  
Как и в случае со свойством **font**, здесь применяется метод упрощенного набора атрибутов списков. В свойстве **list-style** могут быть перечислены все параметры форматирования списков, в том числе

**list-style-image, list-style-position** и **list-style-type**. Например, стиль **ul { list-style: circle inside; }** отобразит маркированные списки с маркерами в виде окружностей без заполнения, не выделяя их из текста пунктов — элементов списка. Когда вы описываете стиль списка с одновременным указанием типа маркера и графического символа — **liststyle: circle url(images/bullet.gif) inside;** — и графический символ не может быть найден по заданному пути браузер будет использовать предопределенный маркер, в данном случае - окружность.

**Поля, отступы, границы**

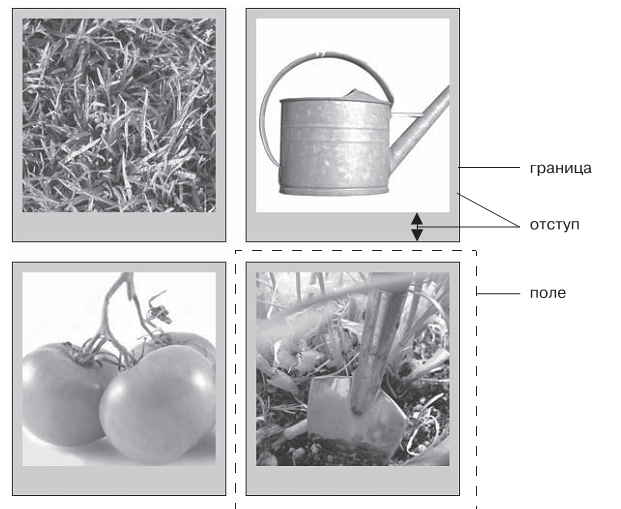
**Понятие блочной модели**

**Padding** — ***отступ*, *«заполнение»*** — промежуток между содержимым и его границей. Отступ отделяет фотографию от окаймляющей ее рамки.  
**Border** — ***граница*, *рамка* —** линия, черта, контур с любой стороны элемента. Граница может быть со всех четырех сторон или с одной стороны в любой их комбинации.  
**Background-color** — ***цвет фона*** — заполняет пространство внутри границы, включая область отступа padding.



ПРИМЕЧАНИЕ  
поскольку разные браузеры применяют разное количество отступов и полей к одним и тем же тегам, лучше всего «**обнулять**» значения этих свойств для всех тегов. Другими словами, используйте набор простых стилей, названный сбросом значений CSS, для удаления отступов и полей из тегов HTML. Потом, когда вы будете создавать дополнительные стили, добавляющие поля и отступы, вы сможете быть уверены в том, что у вас будет согласованный вид элементов в разных браузерах.

**Управление размерами полей и отступов**  
Как поля, так и отступы добавляют промежутки вокруг содержимого тегов. Свойства margin и padding используются для отделения одного элемента веб-страницы от другого. Можно использовать их, например, чтобы добавить пустое пространство между навигационным меню слева и основным содержимым главного раздела веб-страницы справа. Возможно, вы захотите отодвинуть границу от края фотографии



Свойства padding и margin производят одинаковый визуальный эффект, и, пока вы не примените границу или цвет фона, вы не сможете сказать наверняка, каким свойством определен этот промежуток. Но если элемент имеет обрамляющую границу или цветной фон (подложку), вы заметите существенное различие этих свойств. Отступ добавляет промежуток между содержимым и границей элемента и предотвращает эффект заключения содержимого элемента в рамку. Он также включает область фона, поэтому пространство, занимаемое отступом, может быть свободно от содержимого (текста или фотографии), но может быть заполнено фоновым цветом или изображением. А поля добавляют так называемые межстолбцовые промежутки, которые придают веб-странице более «воздушный» внешний вид.

Вы можете использовать любые единицы измерения CSS для определения размеров полей  
и отступов, например:

**margin-right: 20px;  
padding-top: 3em;  
margin-left: 10%;**

Пикселы и em используются и работают точно так же, как при форматировании текста. Поле 20 пикселов добавляет соответствующий пустой промежуток, отступ 3 em — промежуток, в три раза больше размера шрифта стилизуемого элемента. Вы можете также использовать процентные значения, но здесь не все так просто.

**ИНФОРМАЦИЯ ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ**

**Поля, отступы в процентах**

При использовании в качестве единиц измерения процентов браузеры вычисляют размер промежутков полей и отступов на основе *ширины самого элемента-контейнера*, в который заключены форматиру­емые элементы. Рассмотрим самый простой случай веб-страницы, когда таким элементом ­контейнером является <body>, который имеет ширину всего окна браузера. В данном случае значение в процент в каждый конкретный момент времени вычисляется на основании текущей ширины окна. Допустим, оно составляет 760 пикселов. При этом левое поле, равное 10%, добавит промежуток 76 пикселов с левого края стилизуемого элемента. Но если вы измените раз­меры окна браузера, размер промежутка левого поля тоже изменится. Уменьшение до 600 пикселов изме­нит размер на 60 пикселов (10 % от 600 пикселов). Однако элемент-­контейнер не всегда равен ширине окна браузера. В последующих главах книги, когда мы будем создавать более сложный дизайн веб-страниц, вы увидите, что для разработки комплексного дизайна придется добавлять дополнительные элементы. Возможно, вы захотите добавить в веб-страницу тег **<div>** для группировки, отделения содержимого бо­кового навигационного меню. Допустим, меню имеет ширину 300 пикселов. Тег **<div>** будет элементом ­кон­тейнером для всех остальных вложенных в него тегов. Таким образом, размер правого поля любого элемента, вложенного в **<div>** бокового меню и установленного в размере 10 %, будет равен 30 пикселам.  
При установке процентных значений верхнего и нижне го полей элементов ситуация еще более запутанная: **эти значения вычисляются на основании ширины элемента­ контейнера, а не его высоты**. Таким образом, **20%­ное верхнее поле** составит **20% от ширины** стилизуемого тега.

**Сокращенный набор свойств margin и padding**

**margin: 0 10px 10px 20px**; *верхнее значение*, *правое значение*, *нижнее* и *левое*.

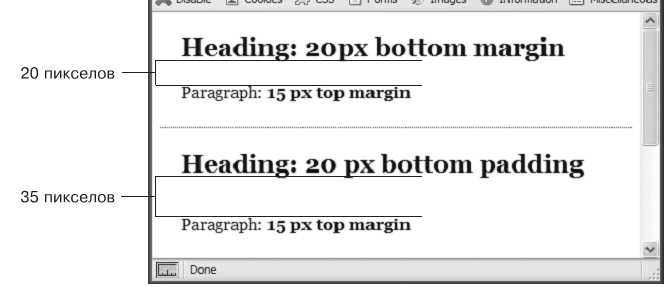
**padding: 10px 5px 5px 10px**; *верхнее значение*, *правое значение*, *нижнее* и *левое*.

**margin: 0 2em;** удаляет верхнее и нижнее поля, а левое и правое поля делает равными **2 em**.

**margin: 0 2em 1em;** установит размер верхнего поля в 0, левого и правого полей - 2 ems, а нижнего поля - 1 em.

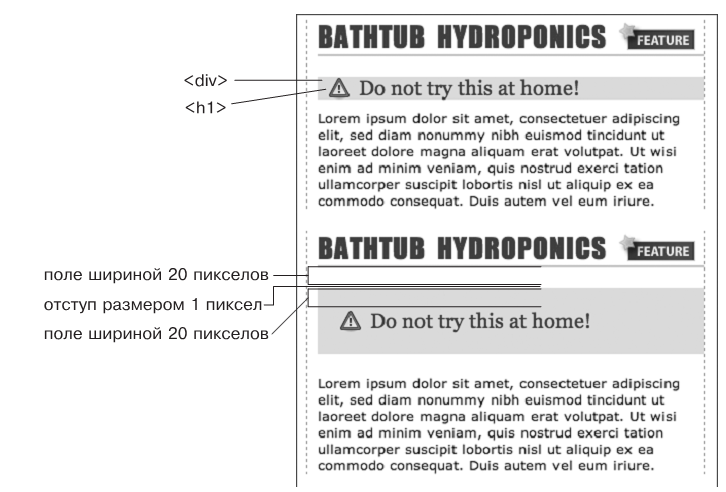
**Конфликты полей**

В CSS не всегда справедливы математические расчеты. Вы сами можете в этом убедиться, когда **нижнее** **поле** (**margin-bottom**) одного элемента веб-страницы касается верхнего поля другого элемента. Вместо того чтобы объединить эти поля, **браузер использует большее из них** (рис. , *вверху*). Предположим, значение нижнего поля(**margin-bottom**) маркированного списка установлено равным **30 пикселам**, а значение **верхнего поля**(**margin-top**) следующего за ним абзаца составляет **20 пикселов**. Вместо того чтобы сложить два значения, получив общий промежуток в размере **50 пикселов** между списком и абзацем, браузер применяет ***наибольшее из двух***значений — в данном случае 30 пикселов. **Если вас это не устраивает, используйте вместо полей верхний или нижний отступ** **(padding)**



несмотря на то что и верхний заголовок имеет нижнее поле в размере **20** **пикселов**, а у расположенного ниже абзаца текста верхнее составляет **15 пикселов**, браузер добавляет промежуток между ними, равный всего **20 пикселам**. Чтобы получить тот промежуток, который вы хотели (**35 пикселов**), используйте вместо полей отступы, как показано в нижней части рисунка. Здесь для заголовка установлен нижний отступ, равный **20 пикселам**. Он складывается с верхним полем абзаца, равным **15 пикселам**, и получается общий промежуток в размере **35 пикселов**.

Ситуация еще более усугубляется, когда один элемент веб-страницы ***вложен***в другой. Это может привести к затиранию отдельных частей. Допустим, вы добавляете на веб-страницу «предупреждение» (заключенное в тег **<div>**). Верхнее и нижнее поля устанавливаются в размере **20 пикселов**, чтобы отделить сообщение от заголовка сверху и от абзаца текста снизу. Пока все выглядит неплохо. Но, предположим, вы вставляете заголовок в сообщение с предупреждением и, чтобы добавить небольшие промежутки между сообщением и верхним и нижним краем блока **<div>**, устанавливаете для заголовка поле в размере **10 пикселов**. Вы, наверное, думаете, что добавили 10-пиксельный промежуток между заголовком и верхним и нижним краем блока **<div>**, но вы не правы (рис. ниже). Вместо этого поле появляется ***над***блоком **<div>**. В данном случае не имеет значения, какого размера поле применяется к заголовку, — поле все равно окажется над **<div>**.



Чтобы решить эту проблему, добавьте небольшой отступ или границу вокруг элемента-контейнера (в данном случае **1 пиксел** нижнего отступа для тега **<div>**).

Есть два пути решения этой проблемы: добавить либо небольшой отступ, либо границу вокруг **<div>**. Поскольку ***между***этими двумя полями располагаются граница и отступ, поля больше не соприкасаются и заголовок имеет небольшой отделяющий промежуток.

**Удаление пустых полей с помощью отрицательных значений**

CSS допускает использование отрицательных значений для создания определенных визуальных эффектов. Отрицательные поля вместо добавления пустого пространства между тегом и соседними элементами, наоборот, вызывают ***удаление***этих промежутков. В результате может получиться абзац, накладывающийся на заголовок, выступающий из своего элемента-контейнера (с боковой панели или из другого элемента **<div>**) или даже совсем исчезающий за пределами окна браузера.

**СОВЕТ**Можете использовать либо верхнее поле абзаца с отрицательным значением, либо отрицательное нижнее поле заголовка. Оба варианта приведут к одному и тому же визуальному эффекту поднятия абзаца выше, ближе к тексту заголовка.

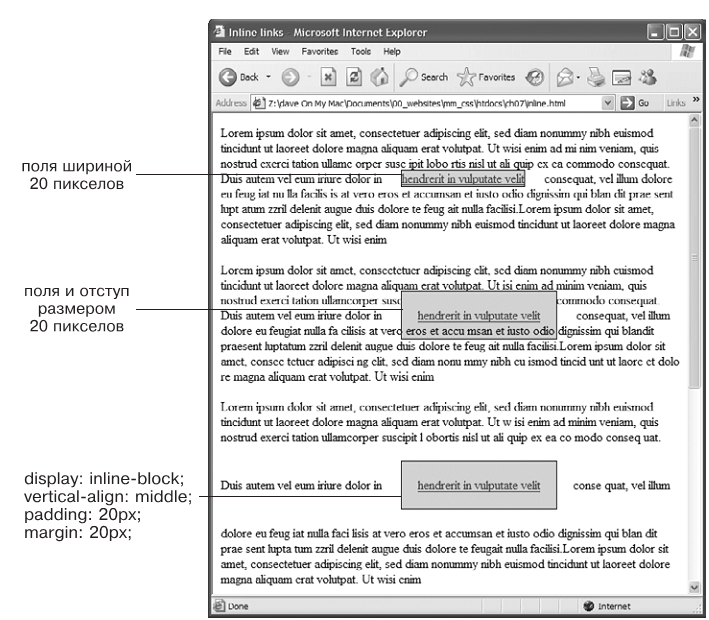
**Отображение линейных и блочных элементов**

В CSS есть два различных типа элементов: ***блочные (прямоугольные)***и ***линейные (inline)***, которым соответствуют два различных типа тегов.

В ***блочных (block)***элементах создается разрыв строки перед тегом и после него. Например, тег **<p>** создает блок, отделенный от тегов, расположенных выше и ниже его. Другими примерами являются заголовки, теги **<div>**, таблицы, списки и элементы списков.

***Линейные* (inline)** элементы не создают разрывов строк ни до, ни после самих тегов. Они отображаются на одной строке с содержимым рядом стоящих тегов. Тег **<strong>** — линейный. Слово, отформатированное с его помощью, будет расположено на одной строке с текстом, который заключен в другие линейные теги, например **<em>.** Было бы очень странно, если бы отдельное слово в середине абзаца, выделенное **<strong>**, вдруг появилось на отдельной строке.  
Другими примерами линейных тегов являются **<img>** — для добавления изображений, **<a>** — для создания ссылок, различные теги для полей форм и т. д.

В большинстве случаев CSS работает с линейными и блочными элементами одинаково. Можно применять шрифты, цвет, фоновые параметры, границы к обоим типам элементов. Однако **поля** и **отступы** линейных элементов браузеры обрабатывают по-другому. Если добавить поля или отступы слева или справа линейного элемента, то посредством установки верхнего или нижнего отступа или поля увеличить высоту линейного элемента не удастся. В верхнем абзаце на рис.  линейному элементу назначено форматирование с применением границ, фонового цвета и полей по 20 пикселов со всех четырех сторон. Но браузер добавляет пустые промежутки **только с левой и правой** сторон элемента.



На рис. выше, в среднем абзаце фон и границы ссылки накладываются на текст, находящийся выше и ниже стилизуемого. Цвет фона линейного элемента отображается поверх вышестоящей строки текста, но под следующей. Браузер обрабатывает каждую строку так, как будто она расположена в стеке, наверху по отношению к предыдущей. Как правило, это не представляет проблемы, так как строки текста обычно не накладываются. Если вы хотите, чтобы верхние и нижние поля работали для линейного элемента, используйте инструкцию **display: inline-block** (см рис ***внизу***). Она оставит элемент линейным, но он будет восприниматься как блочный, поэтому отступы, поля, границы, ширина и высота будут к нему применяться. Это работает даже в таких браузерах, как Internet Explorer 7, но только для нормальных линейных элементов, например ссылок, тегов **<strong>,** **<em>** и **<span>**. Кроме того, вам следует добавить инструкцию

**vertical-align:middle** чтобы **Internet Explorer 7** отображал линейный блок таким же образом, как и остальные браузеры.

**ПРИМЕЧАНИЕ**  
Есть одно исключение из этого правила: если поля или отступы применяются к линейным элементам **<img>**, элементы не изменяют своей высоты. Браузеры корректно изменяют высоту контейнера элемента-изображения, чтобы подогнать ваши отступы и поля.

Иногда требуется, чтобы линейные элементы вели себя так же, как блочные ,или наоборот. Маркированные списки рассматривают элемент в виде отдельного блока, то есть каждый из списка располагается в стеке поверх следующего. Но что делать, если вы хотите изменить поведение пунктов списка таким образом, чтобы все они располагались рядом друг с другом, на одной строке? Или, возможно, вы захотите, чтобы линейный элемент обрабатывался как блочный, например, изображение, встроенное в абзац текста, было расположено на отдельной строке, с верхним и нижним промежутками-интервалами. К счастью, в CSS есть команда, которая позволяет вам это сделать, — это свойство **display**. С его помощью можно заставить блочный элемент работать как линейный:

**display: inline;**

Или, наоборот, вы можете сделать так, чтобы линейные элементы, например изображение или ссылка, вели себя как блочные:

**display: block;**

**ПРИМЕЧАНИЕ**  
У свойства **display** есть большое количество параметров, многие из которых не работают во всех браузерах. Значение **inline-block** работает во всех современных браузерах. Другое значение — **none** — обрабатывается в большинстве браузеров и имеет множество вариантов использования. Это значение выполняет одну простую функцию — полностью скрывает стилизуемый элемент, чтобы он не отображался в окне браузера. Используя программный код **JavaScript**, вы можете скрыть элементы, чтобы они стали видимыми после изменения значения свойства **display** на **inline** или **block**. Сделать их видимыми можно и средствами **CSS**.

**Добавление границ**

Вы можете управлять тремя различными свойствами любой из границ: **color** (цвет), **width** (толщина) и **style** (стиль)

**width**— толщина линии, используемой для очерчивания. Вы можете использовать любые единицы измерения CSS (кроме процентов) или ключевые слова **thin, medium** и **thick** (толстая).

**color** может быть представлен шестнадцатеричным числом, ключевым словом или RGB (RGBA)

**style** - **solid, dotted, dashed, double, groove, ridge, inset, outset, none** и **hidden**

**Сокращенный набор свойства border**

border: 4px solid #FF0000;

ПРИМЕЧАНИЕ  
Последовательность указания параметров свойства не имеет значения: **border: 4px solid #FF0000;** работает так же, как

**border: #FF0000 solid 4px;**.

**Форматирование отдельных границ**

**border- top: 2px solid black;  
border-left: 2px solid black;  
border-right: 2px solid black;  
border-bottom: 4px dashed #333;**

Такого же эффекта можно достигнуть всего двумя строками CSS-кода:

**border: 2px solid black;  
border-bottom: 4px dashed #333;**

Предположим, вы хотите установить границы только с трех сторон элемента (сверху, слева, снизу). Всего две строки кода обеспечат такое форматирование:

**border: 2px inset #FFCC33;  
border-right: none;**

Например, можно переписать определение **border: 2px double #FFCC33;** в следующем виде:

**border-width: 2px;  
border-style: double;  
border-color: #FFCC33;**

В этом варианте используются три строки кода вместо одной, поэтому вы, наверное, будете избегать такого способа. Однако каждая сторона имеет свой собственный набор из трех свойств, которые удобно использовать для отмены одного. Правая граница: **border-right-width**, **border-right-style** и **border-right-color**. Левая, верхняя и нижняя границы имеют похожие свойства: **border-left-width, border-leftstyle** и т. д. Вы можете изменить ширину единственной стороны границы так:

**border-right-width: 4px;**

При таком подходе хорошо то, что, когда вы позже решите изменить границу на сплошную, нужно будет отредактировать только групповое свойство границы, изменив **dashed** на **solid**. Кроме того, вы можете задать собственные значения для каждой стороны границы, используя свойства **border-width, border-style** и **border-color**.

Например, **border-width: 10px 5px 15px 13px;** применит четыре различных значения ширины для каждой из сторон (**верхней, правой, нижней и левой**). Допустим, вы хотите установить все четыре границы элемента в виде пунктирной линии толщиной 2 пиксела, но при этом нужно, чтобы каждая граница имела свой цвет (возможно, вы создаете сайт для детей). Вот способ быстро сделать это:

**border: 2px dashed green;  
border-color: green yellow red blue;**

Этот набор правил создает границы в виде двухпиксельной пунктирной линии со всех четырех сторон элемента, при этом верхняя граница будет иметь зеленый цвет, правая — желтый, нижняя — красный, а левая — синий.

СОВЕТ  
Как правило, при использовании границ требуется добавлять отступы. Они обеспечивают промежутки между границами и содержимым элементов: текстом, изображениями, прочими тегами. Обычно границы отображаются слишком близко к содержимому элементов, только если вы не захотите разместить их вокруг изображения.

**Установка цвета фона.**

**background-color: rgb(109,218,63);**

**body { background-color: rgb(109,218,63); }**

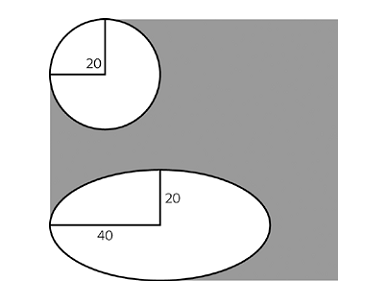
Или можете создать стилевой класс, например .review, со свойством желаемого цвета фона, а затем применить его к тегу **<body>** HTML-кода таким образом: **<body class="review">**.

СОВЕТ  
Когда вы пользуетесь одновременно фоновым цветом и границами, помните: если стиль границы — точечная или пунктирная линия, то фоновый цвет проступает в промежутках между точками или штрихами линий границ. Другими словами, браузеры размещают линию границ поверх цвета фона.

**Создание скругленных углов**

**border-radius**

В качестве единиц измерения чаще всего используются пикселы и **em**, но можно также применять и проценты (хотя они ведут себя немного неожиданно).

Можно создавать не только скругленные ***вверху***, но и эллиптические углы ***внизу***,  
предоставляя либо одно значение — **border-radius: 20px**, либо комбинацию из двух значений, разделенных символом слеша, — **border-radius: 40px/20px;**

**border-radius: 0 30px 10px 5px;**

Сначала задается числовое значение для левого верхнего угла блока, а затем по часовой стрелке — для всех остальных углов. То есть первое значение применяется к **левому верхнему**, второе **(30px) — к правому верхнему**, третье **(10px) — к правому нижнему**, четвертое **(5px) — к левому нижнему** углу. Можно также задать только два значения, тогда первое число будет применено к левому верхнему и правому нижнему углам, а второе к правому верхнему и левому нижнему углам.

Для **эллиптической формы** угла требуется два значения радиуса: **первое для радиуса от центра до левого или правого края**, **а второе для определения расстояния от центра до верхнего или нижнего края.**

Для эллиптической формы угла требуется два значения радиуса: первое для радиуса от центра до левого или правого края, а второе для определения расстояния от центра до верхнего или нижнего края.

**border- radius: 1em 2em 1.5em .75em;**

можно также написать следующим образом:

**border-top-left-radius: 1em;  
border-top-right-radius: 2em;  
border-bottom-right-radius: 1.5em;  
border-bottom-left-radius: .75em;**

**Префиксы производителей**

**-webkit**- — применяется Chrome, Safari и другими браузерами на базе WebKit;

**-moz-** — используется Mozilla Firefox;

**-o-** — применяется Opera;

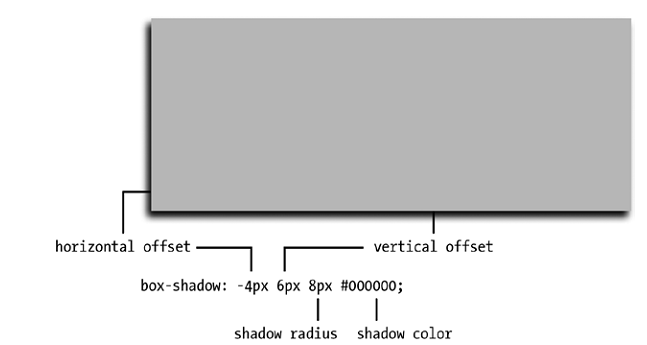
**-ms-** — используется Microsoft Internet Explorer.

Когда свойству требуется префикс производителя, то для получения одного и того же эффекта обычно использует­ся несколько строк кода, по одной для каждого произво­дителя, а в завершение —версия без префикса:

**-moz-box-sizing: border-box;  
-webkit-box-sizing: border-box;  
box- sizing: border- box;**

**Добавление теней**

**box-shadow: inset 4px 4px 8px rgba(0,0,0,.75); inset – добавляет тень внутрь блока.**



**Браузер Android и устаревшие версии Safari для iPhone для работы свойства box-shadow требуют использования префикса производителя:**

**-webkit-box-shadow: 2px 2px 10px #000000;**

**box-shadow: 2px 2px 10px #000000;**

**Определение параметров высоты и ширины**

**width: 30%;  
height: 20em;**

Пикселы как единицы измерения просты в использовании, понятны и удобны, обеспечивают точные ширину или высоту. Единица измерения **em** — это примерно то же самое, что и размер шрифта текста, но в условных единицах. Допустим, вы устанавливаете размер шрифта **24 пиксела**; единица **em** для этого стилизуемого элемента будет равна 24 пиксела, а если вы установите ширину равной **2 em**, она составит **2  ⋅ 24**, или **48 пикселов**. Если вы не определите в стиле размер шрифта

**ПРИМЕЧАНИЕ**  
В CSS3 представлены дополнительные единицы измерения под названием **rem**, которые основаны на значении свойства **font-size** элемента **html**. Эти единицы измерения работают во всех основных браузерах, за исключением **Internet Explorer 8** и более ранних версий.

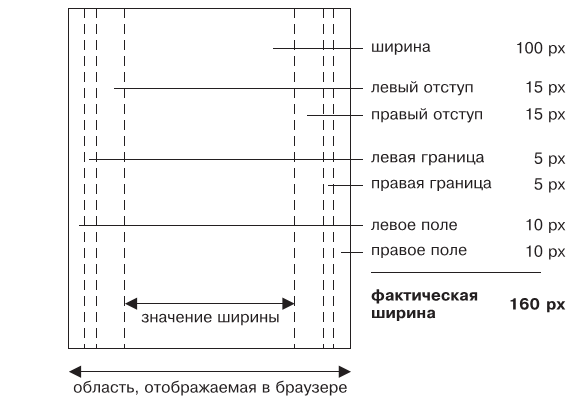
Процентные значения свойства ширины width рассчитываются на основании ширины элемента-контейнера. Если вы установите ширину заголовка равной **75 %,** и этот заголовок не вложен ни в какие другие элементы веб-страницы с явно определенной шириной, то ширина текста заголовка составит **75 %** от ширины окна браузера. Если посетитель изменит размер окна браузера, то ширина заголовка тоже изменится. Однако если заголовок заключен в блок **<div>** шириной **200 пикселов** (возможно, для создания столбца), то ширина данного заголовка составит **150 пикселов**. Процентные значения в свойстве высоты **height** работают точно так же, но расчет базируется на высоте элемента-контейнера, а не на его ширине.

**Вычисление фактических размеров блочных элементов**

Свойства **width** и **height** устанавливают ширину и высоту ***области содержимого***стилизуемого элемента - пространства, в котором заключены текст, изображения или другие вложенные теги

Фактическая ширина элемента веб-страницы представляет собой область экрана (окна браузера), выделяемую для отображения. Она **состоит из ширины полей, границ, отступов и явно указанного значения ширины** в свойстве стиля:

**width: 100px;  
padding: 15px;  
border-width: 5px;  
margin: 10px;**



Общее правило ***по регулированию высоты элементов на странице*** гласит***: не делайте этого!***В этом примере блок с цитатой, который используется для того, чтобы акцентировать внимание на интересном отрывке из статьи, имеет ширину и высоту по **100 пикселов**. Когда в блок добавляется больше текста, чем может уместиться в такую высоту, его содержимое выходит за пределы. Даже если вы уверены, что текст, который вы разместили в блоке с фиксированной высотой, соответствует его размерам, посетитель может увеличить размер шрифта в своем браузере и высота текста, соответственно, может стать больше, по сравнению с высотой блока.  
Другими словами, свойство **height** полезно для контроля высоты элемента **<div>**, содержащего, например, изображения, потому что в таком случае вы можете правильно определить его высоту. Однако, если вы используете это свойство для элементов, содержащих текст, не забудьте не только протестировать свои страницы в основных браузерах, но и проверить их при различных установленных размерах шрифта, увеличивая его в браузерах.



СОВЕТ  
Область баннеров на странице является еще одним подходящим кандидатом для установки высоты. Обычно баннер имеет ограниченное содержимое: логотип, поле поиска, может быть, какие-нибудь навигационные кнопки. Зачастую у баннеров остается довольно много пустого пространства (помогающего привлечь внимание посетителя к ключевым элементам навигационной панели), поэтому указание высоты для баннера обычно не вызывает проблем.

**Переопределение ширины блока с помощью box-sizing**

Как уже упоминалось, веб-браузеры традиционно вычисляют ширину элемента, складывая значения свойств **border**, **padding** и **width**.

Когда для нескольких столбцов используется процентное отношение, могут возникать довольно странные проблемы. Предположим, что есть два столбца (на самом деле два таких тега, как **<div>**) и нужно, чтобы каждый занимал **50 %** ширины окна. Соответственно, для двух столбцов устанавливается ширина **50 %,** но на тот момент, когда добавляется отступ или рамка к одному из столбцов, вы увеличиваете его ширину, которая становится больше **50 %** (если точнее, она составляет **50 %** плюс значение левого и правого отступов и значение ширины левой и правой частей). В большинстве случаев это заставит второй столбец опуститься ниже первого

К счастью, CSS3 предлагает свойство, позволяющее изменить порядок вычисления браузером экранной ширины (и высоты) элемента. Свойство **box-sizing** предоставляет три варианта.

- **content-box** устанавливает ранее рассмотренный способ, с помощью которого браузеры всегда определяют экранную ширину и высоту элемента. То есть браузер добавляет ширину рамки и толщину отступа к значениям, установленным для свойств ширины и высоты, чтобы определить экранную ширину и высоту заданного тегом элемента. Поскольку это поведение является исходным, указывать что-нибудь для **content-box** не нужно.

- **padding-box** сообщает браузеру, что при установке для стиля свойства ширины или высоты они должны включать отступы как часть своего значения. Например, предположим, что есть элемент с отступами слева и справа, равными 20 пикселам, ширина которого установлена равной 100 пикселам. Браузер будет рассматривать часть, приходящуюся на отступы, частью этого 100-пиксельного значения. То есть ширина области содержимого будет равна всего лишь 60 пикселам (100 – 20 [левый отступ] – 20 [правый отступ]).

**- border-box** сообщает браузеру о необходимости включения в качестве составляющей части значений свойств ширины и высоты толщину как отступа, так и рамки. Эта настройка решает проблему использования значений, выраженных в процентном отношении, о которой говорилось выше. То есть, когда для свойства **box-sizing** установлено значение

Для **Mozilla** и **Chrome** придется воспользоваться **префиксом производителя**

**-moz-box-sizing: border-box;  
-webkit-box-sizing: border-box;  
box- sizing: border- box;**

Многие веб-разработчики настолько оценили пользу настройки border-box, что создали универсальный селектор стиля, применяемый к каждому элементу на странице:

**\* {  
 -moz-box-sizing: border-box;  
 -webkit-box-sizing: border-box;  
 box- sizing: border- box;  
 }**

**ПРИМЕЧАНИЕ**  
Свойство **box-sizing** работает даже в **Internet Explorer 8** и более старших версиях, то есть оно поддерживается более чем 95 % используемых браузеров. К сожалению, **Internet Explorer 7** не понимает этого свойства, поэтому при использовании настройки

**border-box** **Internet Explorer 7** ее проигнорирует и нарисует по сравнению с другими браузерами более широкие (и более высокие) элементы. Если вам необходимо сохранить поддержку **Internet Explorer 7**, это свойство лучше не использовать.

**Управление поведением блочных элементов с помощью свойства overflow**

Когда содержимое стилизуемого тега имеет размеры больше определенных свойствами **width** и **height**, в браузерах содержимое отображается за пределами (выступает наружу) границ элемента, часто закладываясь на него.

Браузер использует в этой ситуации свойство **overflow**. В качестве значения можно указать одно из четырех ключевых слов, определяющих, как должно отображаться содержимое, которое выходит за пределы блочного элемента.

* **Visible** — это значение, принимаемое браузером по умолчанию. Указание этого ключевого слова имеет тот же эффект, что и отсутствие установки свойства **overflow**.
* **Scroll**  — позволяет добавить полосы прокрутки. Параметр создает своего рода окно мини-браузера внутри веб-страницы, которое выглядит подобно HTML-рамкам (фреймам). Вы можете использовать ключевое слово **scroll** , чтобы вместить объемное содержимое в ограниченной области. К сожалению, при таком варианте полосы прокрутки отображаются ***всегда***, даже если содержимое по размерам помещается внутри блока.
* **Auto** — чтобы сделать полосы прокрутки необязательными, пользуйтесь данным значением. Оно выполняет ту же функцию, что и scroll , с одним исключением - полосы прокрутки добавляются только при необходимости.
* **Hidden** — скрывает любое содержимое, выходящее за пределы блочного элемента. Это значение небезопасно, поскольку может привести к тому, что часть содержимого будет не видна. Но иногда оно применяется при создании плавающих разметок.

**Задание максимальных и минимальных значений высоты и ширины**

* **max-width** задает максимальную ширину элемента. Этот элемент может быть уже установленного предела, но он не может быть шире его. Данный вариант пригодится в том случае, когда ваша страница должна менять свои размеры, чтобы поместиться на дисплеях разной ширины.
* **max-height** работает во многом похоже на **max-width**, за исключением того, что **max-height** применяется к высоте элемента. Но, как уже упоминалось, с высотой элемента лучше все же не связываться.
* **min-width** устанавливает минимальную ширину элемента. Элемент может растянуться шире значения минимальной ширины, но никогда не может стать уже этого значения.
* **min-height** работает точно так же, как и свойство **min-width**, но применительно к высоте. Используя минимальную высоту, вы заставляете веб-браузер сделать элемент по крайней мере имеющим определенную высоту. Если содержимое элемента выше, то браузер сделает выше весь элемент.

Свойства минимума и максимума можно также применять вместе. Например, если нужно гарантировать минимальную ширину страницы **760 пикселов**, но не дать ей расширяться более чем до **1200 пикселов**, можно создать следующий стиль:

**body {  
min-width: 760px;  
max-width: 1200px;  
}**

**Управление обтеканием содержимого плавающих элементов**

Свойство **float** перемещает любой элемент в нужное место, выравнивая его по левому или правому краю веб-страницы. В процессе перемещения содержимое, находящееся ниже позиционируемого плавающего элемента, смещается вверх и плавно обтекает сам плавающий элемент. Плавающие (или перемещаемые) элементы — идеальное средство для того, чтобы выделить дополнительную информацию из основного текста. Изображения могут смещаться к любому краю веб-страницы, обеспечивая перенос рядом стоящего текстового содержимого и его изящное обтекание вокруг изображений. Точно так же вы можете передвинуть боковую панель с относящейся к тексту информацией и управляющие ссылки к одной из сторон веб-страницы.

* **left** — перемещает стилизуемый элемент влево, при этом содержимое, находящееся ниже его, обтекает правый край элемента;
* **right** — перемещает элемент вправо;
* **none** — отменяет обтекание и возвращает объект в его обычную позицию.

Например:

**float: left;**

Плавающие элементы могут быть расположены у левого или правого края окна содержащего их элемента-контейнера. В некоторых случаях это просто означает, что элемент перемещается к левому или правому краю окна браузера. Однако если вы перемещаете элемент, находящийся внутри другого тега, для которого установлены значения ширины или позиции на веб-странице, то перемещение будет осуществлено к левому или правому краю этого тега, который является контейнером.

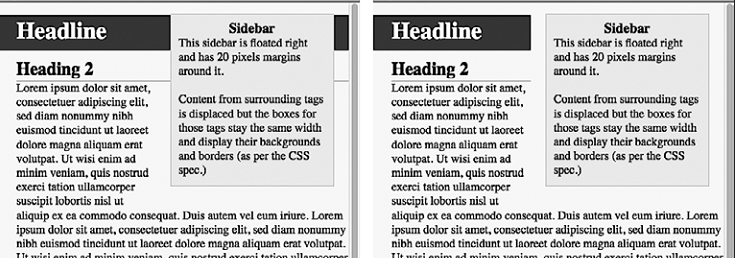
Вы можете применять свойство **float** к линейным элементам, например **<img>**. Надо сказать, использование выравнивания по левому или правому краю для фотографий — самый обычный, даже наиболее распространенный способ применения **float**. Браузер обрабатывает линейные элементы точно так же, как блочные. Поэтому, используя отступы и поля в линейных элементах, вы избежите проблем, которые обычно возникают в такой ситуации.

Вы можете также использовать **float** для блочных элементов, таких как заголовок или абзац текста (или один из новых элементов **HTML5**, например **<article>**, **<section>** или **<aside>**). Общая методика применения свойства к тегу **<div>**, содержащему другие **HTML**-теги, заключается в создании своего рода блока-контейнера. Таким способом вы можете создавать боковые панели, плавающие цитаты и другие обособленные элементы веб-страниц. При использовании свойства **float** для блочных элементов рекомендуется установить свойство **width** для них (на самом деле правила CSS требуют установки ширины плавающим элементам для всех тегов, кроме изображений). Таким образом вы можете управлять размером блока по горизонтали, оставив промежуток для текстового содержимого, расположенного после этого блочного элемента, и, соответственно, задать его поведение — перенос, обтекание.

**ПРИМЕЧАНИЕ**  
Порядок написания HTML-кода оказывает большое влияние на отображение плавающих элементов веб-страницы. HTML-код должен находиться до любого HTML-содержимого, которое обтекает плавающий элемент. Допустим, вы создали веб-страницу, в которой имеется тег заголовка **<h1>**, а за ним идет тег абзаца **<p>**. Внутри **<p>** вы вставили фотографию с помощью тега **<img>**. Теперь, если вы установите выравнивание фотографии по правому краю, заголовок **<h1>** и часть содержимого абзаца **<p>** будут по-прежнему отображены над фотографией. Только содержимое, следующее за тегом **<img>**, будет обтекать изображение с левой стороны.

**Фон, границы и плавающие элементы**

К недовольству многих веб-дизайнеров, фон и границы не переносятся таким же образом, как другие элементы веб-страницы. Допустим, у вас есть боковая панель, примыкающая к правому краю веб-страницы. Содержимое страницы, расположенное под ним, как ему и положено, приподнимается выше и обтекает панель. Однако, если у этого содержимого есть фоновые параметры или границы, они отображаются, фактически проступая под боковой панелью (рис, *слева*).   
По сути, браузер окружает плавающий элемент только текстом, но не границами или фоном. Как это ни удивительно, но именно так работает механизм обтекания. Возможно, вы не будете следовать этим правилам; а захотите, чтобы границы, фоновый цвет или рисунок не отображались при достижении плавающего элемента (см. рис справа)



**Первый метод** заключается в добавлении еще одного свойства в стиль, устанавливающий параметры фона и границ и оказывающий воздействие на плавающий элемент. Нужно добавить в данный стиль следующий код:

**overflow: hidden;**

**h1 { overflow: hidden;}**

**h2 { overflow: hidden;}**

Свойство **overflow** (описано выше) отменяет отображение продолжающихся фона или границ под плавающими элементами.

ПРИМЕЧАНИЕ  
Этот метод работает не во всех браузерах. Смотрите врезку далее.

**Другой подход** состоит в добавлении границ вокруг плавающего элемента. Если вы создаете достаточно толстую границу цвета фона страницы, то не будете даже догадываться о существовании лишних границ, находящихся под плавающим элементом, так как они располагаются поверх и скрывают аналогичные лишние атрибуты.

**Отмена правил обтекания**

Свойство **clear** допускает следующие значения:

* **left** — стилизуемый элемент смещается вниз относительно плавающего с установленным левым обтеканием, но правое обтекание остается в силе;
* **right** — стилизуемый элемент смещается вниз относительно плавающего с установленным правым обтеканием, параметр левого обтекания остается в силе;
* **both** — вызывает перемещение стилизуемого элемента вниз относительно плавающего с установленными левым и правым обтеканием;
* **none** — полностью отменяет очистку обтекания: говорит браузеру, что стилизуемый элемент должен вести себя стандартным образом, как предусмотрено, то есть он может обтекать плавающие элементы как с левой, так и с правой стороны.

В нашем примере текст примечания с авторским правом должен всегда отображаться внизу веб-страницы и не должен обтекать другое содержимое. Ниже приведен стилевой класс, который выполняет это:

**. copyright {  
clear: both;  
}**

**Добавление графики на веб-страницы**

Здесь можно работать с графическими элементами двумя способами: посредством тега **<img>** и свойства **background-image** (которое позволяет применить изображение к фону любого тега на странице).

**CSS и тег <img>**

Тег **<img>** является средством добавления изображений на веб-страницы фотогалерей еще со времен появления Интернета. Даже сайты без фотографий используют этот тег для добавления логотипов, управляющих элементов (кнопок) и иллюстраций.

Ниже представлен список CSS-свойств, наиболее часто применяемых в сочетании с изображениями.

* **border**
* **padding**
* **margin**
* **float**

В большинстве случаев вам не потребуется создавать стиль для самого тега **<img>**. Его форматирование затрагивает слишком большой диапазон элементов веб-страницы и изменит все изображения, включая элементы, обеспечивающие совершенно разные функции. Так, наряду с изображениями одинаковое форматирование будут иметь логотип, навигационные кнопки, фотографии и даже рекламные баннеры. Вместо этого следует использовать стилевые классы, например **.galleryImage** или **.logo**, для применения стилей к нужным элементам выборочно. Другой подход заключается в использовании селекторов потомков для целевого форматирования изображений, сгруппированных в одном фрагменте веб-страницы.

**Добавление фоновых изображений**

Свойство **background-image** добавляет рисунок в качестве фона элемента. Чтобы поместить его на заднем плане всей веб-страницы, можно создать стиль для тега **<body>**:

**body {  
background-image: url(images/bg.gif);  
}**

Свойство принимает единственное значение: ключевое слово **url** , за которым следует путь к файлу рисунка, заключенный в круглые скобки. Вы можете использовать абсолютный URL, например, так:

**url**(<http://www.cosmofarmer.com/image/bg.gif>)

Или относительный путь от документа или корневого каталога сайта:

**url (../images/bg. gif)** /\* относительный путь от документа \*/  
**url (/images/bg. gif)** /\* относительный путь от корневого каталога \*/

В CSS все три следующих строки кода идентичны:

**background- image: url(images/bg. gif);  
background-image: url("images/bg.gif");  
background-image: url('images/bg.gif');**

**Управление повтором фоновых изображений**

**background- repeat:**

Свойство может принимать четыре значения. Рассмотрим их по порядку.

* **repeat** — параметр по умолчанию, обеспечивает повторное отображение фонового рисунка слева направо и сверху вниз, до полного заполнения всего пространства веб-страницы.
* **no-repeat** — отображает фоновый рисунок один раз, без повторения и перекрытия.
* **repeat-x** — вызывает повторение фонового изображения горизонтально вдоль оси *Х*
* **repeat-y** — повторяет фоновое изображение вертикально вдоль оси *Y*

**Позиционирование фоновых изображений**

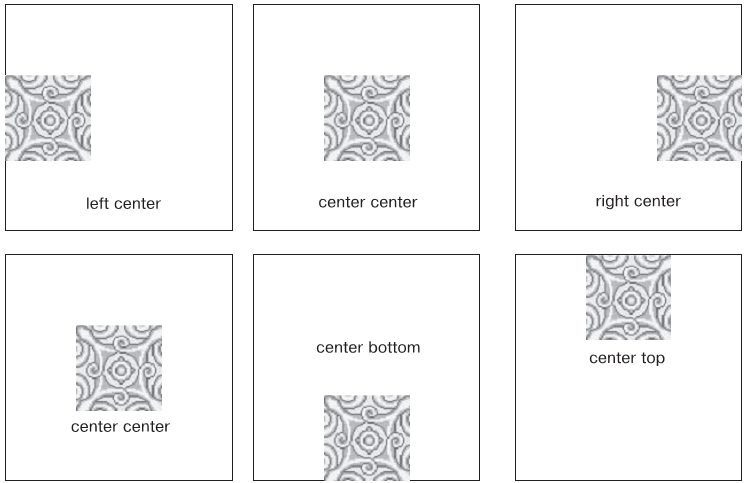
**Ключевые слова**

Можно работать с двумя наборами ключевых слов: один из них имеет три параметра управления горизонтальным позиционированием: **left, center, right**, а другой - три параметра вертикального позиционирования: **top, center, bottom**

Предположим, вы хотите поместить рисунок прямо в центре веб-страницы. Для этого можно создать следующий стиль:

**body {  
background- image: url ( bg\_page. jpg);  
background-repeat: no-repeat;  
background-position: center center;  
}**

Чтобы переместить его в верхний правый угол веб-страницы, просто измените параметр позиции фонового рисунка:  
**background-position: right top;**



ПРИМЕЧАНИЕ  
Если вы решили применить повторяющееся фоновое изображение (установив для свойства **background-repeat** одно из значений, описанных ранее), то начальную точку или координату первого отображаемого фонового рисунка определит свойство **background-position**. Например, если вы примените параметр repeat, то весь фон веб-страницы будет заполнен фоновым рисунком. Но именно background-position укажет, с какой позиции нужно начать повторное отображение.

Ключевые слова полезны, когда необходимо создать вертикальные или горизонтальные баннеры-заголовки. Если вы хотите, чтобы фоновый рисунок был горизонтально центрирован и повторялся от верхнего до нижнего края веб-страницы, создавая фон для всего содержимого, то можете выполнить следующий стиль:

**body {  
background-image: url(background.jpg);  
background-repeat: repeat-y;  
background-position: center top;  
}**

СОВЕТ  
На самом деле вы можете добавить фоновый рисунок к обоим тегам: **<html>** и **<body>**. Если вы повторяете оба изображения по горизонтали и располагаете рисунок тега **<body>** вверху, а изображение тега **<html>** внизу, то можете достичь эффекта двух полосок, идущих через верхнюю и нижнюю части страницы, вне зависимости от высоты страницы. Это работает во всех современных  
браузерах, даже в Internet Explorer 6!

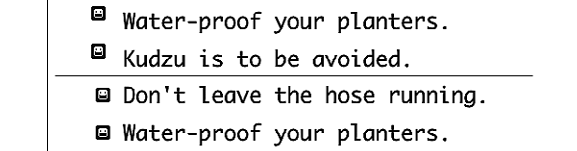
**ОШИБКИ БРАУЗЕРОВ**  
**Проблема с отображением фоновых рисунков в нижней части окна браузера Firefox**  
Отображая рисунок в качестве фона всей веб­ страницы, Firefox не всегда правильно устанавливает вертикальное позиционирование изображе­ния. Например, если вы используете вертикальную позицию bottom, изображение не всегда появляется в нижней части окна браузера. Это случается,когда основное содержимое веб­страницы мень­ше по высоте.  
Если веб­страница имеет всего несколько абзацев текста и отображается на большом мониторе, то Firefox принимает за нижний край окна браузера bottom нижний край последнего абзаца текста веб­ страницы. Если вы столкнетесь с такой ситуацией, то просто добавьте следующий стиль:

**html { height: 100%; }** .  
**ОШИБКИ БРАУЗЕРОВ**

**Точные значения**

Допустим, вы хотите установить собственные маркеры для списка. При добавлении в тег **<li>** фонового рисунка маркеры не всегда будут отображаться правильно центрированными на строке (рис. , *вверху*). Придется выровнять маркеры списка посредством свойства позиционирования фонового рисунка (см. рис *внизу*). В нашем случае элементы списка будут выглядеть гораздо симпатичнее, если расположить маркеры на 5 пикселов правее и на 8 пикселов ниже. Для этого добавим в стиль фонового изображения следующее свойство:

**background-position: 5px 8px;**



Первое значение - отступ о от левого края, а второе - от верхнего края контейнера. Можно также использовать отрицательные значения.

**Процентные значения**

Как и в случае с пикселами или значениями в **em**, вы должны указать два процентных значения: одно представляет собой горизонтальную координату, а второе — вертикальную. Относительно чего рассчитывается значение, указанное в процентах? Если объяснить в двух словах, то оно приравнивается к процентному значению стилизуемого элемента.  
Лучший способ понять — рассмотреть несколько практических примеров. Что-бы позиционировать изображение в центре веб-страницы (аналогично изображению в центре рис.), необходимо написать:

**background- position:50% 50%;**

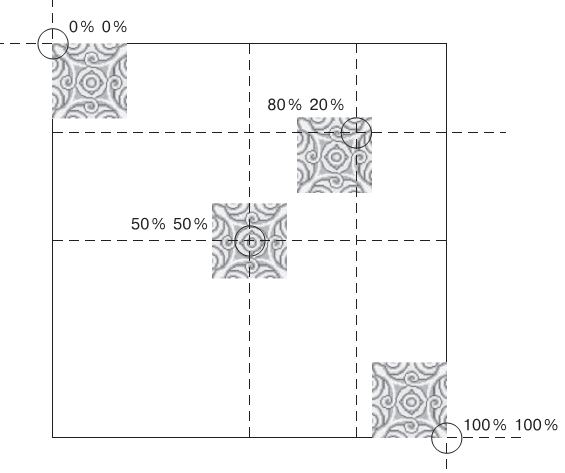
Свойство устанавливает координату *x* таким образом, что она указывает на точку изображения в **50%** от его левой стороны, которая находится в **50%** от левого края веб-страницы (или любого элемента, к которому применяется фоновое изображение). Координата *y* задается так, что она указывает на точку изображения в **50%** от верхней стороны, которая находится в **50%** от соответствующего края веб-страницы или стилизуемого элемента. Другими словами, центр изображения совпадает с центром элемента, для которого оно определено в качестве фонового. Это означает, что при использовании процентных значений точные координаты позиционирования динамически изменяются и похожи на «движущуюся мишень»

**background- position:80% 20%;**

в этом случае центр точки позиционирования берется от 80% по длине картинки и 20% по ширине.

ПРИМЕЧАНИЕ  
Размещение изображения по вертикали в фоне страницы с помощью процентов не обязательно поместит его в правильном месте, если содержимое страницы не заполняет всю высоту окна браузера

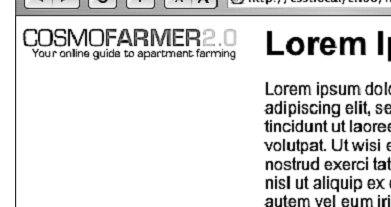
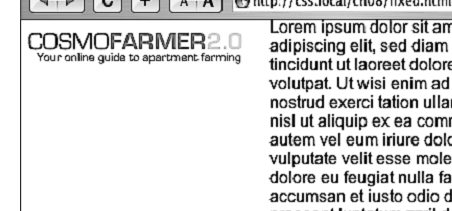
Как и в случае с пикселами и **em**, можно назначать отрицательные процентные значения, но результаты трудно предугадать. Вы можете также комбинировать и подбирать значения в пикселах, **em** и процентах одновременно. Например, чтобы поместить изображение на расстоянии **5 пикселов** от левого края элемента, но в центре элемента по высоте, применяют свойство со следующими параметрами:

**background-position: 5px 50%;**  


**Фиксация изображения на месте**

**background-attachment** может принимать два значения: **scroll** и **fixed**

**body {  
background-image: url(images/logo.gif);  
background-repeat: no-repeat;  
background-attachment: fixed;  
}**

Значение **fixed** также удобно использовать с повторяющимся мозаичным фоновым изображением. Когда вам требуется выполнить прокрутку, текст основного содержимого плавно перемещается (буквально плывет) и изящно исчезает вверху веб-страницы, а фоновое изображение остается на месте, создавая красивый эффект.

**Определение начальной позиции фонового изображения и порядка его отсечения**

начальную позицию изображения можно изменить, воспользовавшись свойством

**background-origin**. Этому свойству задается одно из трех значений:

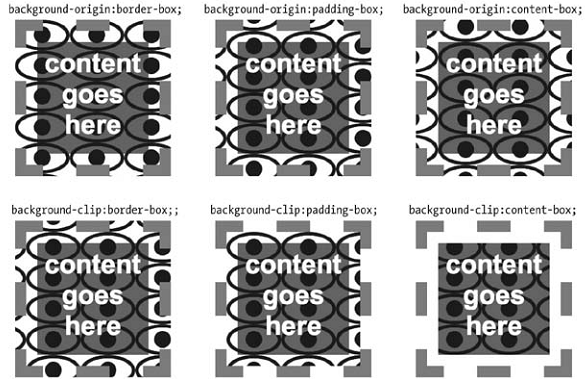
* **border-box** - изображение помещается в верхний левый угол области, которая выделена для рамки
* **padding-box** - изображение помещается в верхний левый угол области, которая отведена под отступы. Это обычное место, куда браузер помещает фоновое изображение;
* **content-box** - изображение помещается в верхний правый угол той области, которая отведена под содержимое

ПРИМЕЧАНИЕ  
Свойства **background-origin** и **background-clip** в **Internet Explorer 8** и более ранних версиях не работают. Поскольку **Internet Explorer** все еще пользуется популярностью, применяйте это свойство с оглядкой.

Свойство **background-clip** ограничивает область появления фонового изображения.

* **border-box** - позволяет изображению появляться позади содержимого полностью за любыми границами.
* **padding-box** - дает возможность ограничить любое фоновое изображение областью отступов и содержимого элемента
* **content-box** - позволяет ограничить фоновое изображение областью содержимого элемента

**background-origin: content-box;  
background-clip: content-box;**



**Масштабирование фоновых изображений**

Чтобы установить размер, можно выбирать либо значения, либо ключевые слова.

**background-size: 100px 200px;** - устанавливает для фонового изображения ширину 100 пикселов и высоту 200 пикселов.

**background- size: 100px auto;** - В таком случае фоновое изображения будет иметь ширину 100 пикселов, а браузер  
автоматически установит его высоту, сохраняя пропорции изображения

**background-size: 100% 100%;** - Если  
нужно масштабировать изображение для точного попадания в фон, можно для обеих настроек использовать значение 100 % (изображение слева)



* Ключевое слово **contain** приводит к изменению размеров изображения. Оно заставляет его поместиться в область, сообразуясь с соотношением сторон (посередине изображение). В зависимости от формы изображения и элемента изображение растягивается, чтобы поместиться либо по ширине, либо по  
  высоте элемента.  
  **background-size: contain;**
* Ключевое слово **cover** заставляет ширину изображения соответствовать ширине элемента, а высоту изображения соответствовать высоте элемента. Обычно это приводит к искажению изображения путем либо сжатия, либо растяжения с тем, чтобы оно поместилось в размеры элемента (изображение справа).  
  **background-size: cover;**

**СОВЕТ**Свойство **background-size** может оказаться особенно удобным при работе с элементами, размер которых определяется с применением процентных показателей, допустим при использовании восприимчивых конструкций, рассматриваемых в гл. 14. Например, если поместить изображение в фон. Применение CSS баннера, занимающего 960 пикселов при просмотре на мониторе настольного компьютера, но сжать его до 480 пикселов при просмотре на телефоне, можно поместить в баннер большое изображение  
и воспользоваться следующей настройкой:

**background-size: 100% auto;**

Эта настройка заставит веб-браузер изменить размер изображения таким образом, чтобы оно поместилось в сжатый баннер.

**Сокращенный вариант свойства background**

Фактически вы можете объединить и перечислить все фоновые свойства (включая **background-color**) в единственной строке лаконичного кода CSS.

**body {  
background: url(bullseye.gif) center center no-repeat fixed #FFF; }**

Вовсе не обязательно указывать абсолютно все параметры свойства. Вы можете использовать один из них или любое сочетание. Например, **background: yellow** равнозначно **background-color: yellow**. Все те параметры свойства, которые вы не определите сами, будут иметь стандартные значения по умолчанию. Допустим, вы задали только само фоновое изображение:

**background: url ( image/bullseye. gif)**

Это эквивалентно следующему стилю:

**background: url(image/bullseye.gif) scroll left top repeat transparent;**

Поведение, когда при отсутствии определения значения происходит возвращение к значениям, по умолчанию может привести к весьма неожиданным результатам. Предположим, например, что к стилю добавляются два объявления:

**background-color: yellow;**

**background: url(image/bullseye.gif) no-repeat;**

Возможно, вы ожидали увидеть изображение бычьего глаза на желтом фоне. Но вы его не увидите, потому что при встрече свойства **background** без указания цвета, браузер переключает значение background-color на transparent (невидимый). Чтобы выйти из этой сложной ситуации нужно указать свойство **background-color** вторым:

**background: url(image/bullseye.gif) no-repeat;  
background-color: yellow;**

Кроме того, когда к одному и тому же элементу применяется сразу несколько свойств, можно в конечном итоге случайно уничтожить фоновые изображения.

**Приводим в порядок навигацию сайта**

**Понимание состояний ссылок**

* :link - непосещенная ссылка;
* :visited - посещенная ссылка (это означает, что по ссылке уже выполнялся переход, то есть URL-адрес сохранен в журнале истории браузера);
* :hover - ненажатая (над которой находится указатель мыши);
* :active - нажатая ссылка (удерживаемая мышью).

**Создание панелей навигации**

CSS-код немного отличается в зависимости от того, нужна нам горизонтальная или вертикальная панель управления. В любом случае вы должны выполнить два шага.  
1. **Удалить маркеры.** Чтобы панель навигации не была похожа на маркированный список, удалим маркеры, установив для свойства list-style-type значение  
none:  
 ul.nav {  
 list-style-type: none;  
}  
2. **Убрать отступы и поля.** Поскольку браузеры сами делают отступы для элементов списка, нам придется избавиться от них. Однако одни браузеры используют для этих целей свойство **padding**, а другие — **margin**, поэтому нам нужно установить для обоих свойств значение **0**:

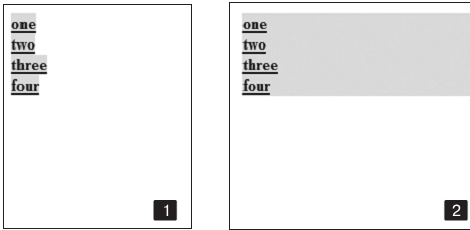
ul.nav {  
 list-style-type: none;  
 **padding-left: 0;**  
 **margin-left: 0;**  
}

По сути, эти два шага обеспечивают всем элементам списка простой вид, как у обычного блочного фрагмента текста — абзаца или заголовка (с одним исключением: браузер не добавляет полей между элементами списка).

**Вертикальные панели навигации**

**1. Отобразим ссылку в виде блочного элемента.** Поскольку тег **<a>** является линейным элементом, ссылка будет занимать по ширине пространство, равное ширине содержимого тега. Кнопки со ссылками в виде текста различной ширины (например, Home (Домой) и Our Products (Наши продукты)) будут иметь различную ширину. Ступенчатое вертикальное отображение кнопок друг над другом выглядит не очень хорошо (рис, *1*). Кроме того, верхние и нижние отступы и поля не оказывают никакого воздействия на линейные элементы. Чтобы обойти эти ограничения, преобразуем ссылки в блочные элементы:  
  
**ul.nav a {  
 display: block;  
}**

Параметр отображения **block** не только выравнивает все кнопки по ширине, но и делает всю область ссылок активной при щелчке на ней кнопкой мыши, подобно настоящим кнопкам. Это тот случай, когда посетители щелкают на области кнопки, где нет текста (например, отступы вокруг), но ссылка по-прежнему работает.



**2. Преобразование пунктов списка в блочные элементы обеспечивает ссылкам ширину тегов, в которые они вложены**. Поскольку ссылки просто помещены на веб-страницу, их ширина окажется равной ширине окна браузера (см. рис, *2*). Есть несколько способов ограничить ширину  
ссылок.  
Сначала можно просто определить тег **<a>**. Например, если вы хотите, чтобы все кнопки имели ширину **8 em**, добавьте в стиль свойство **width**:

ul.nav a {  
display: block;  
**width: 8em;**  
}

Установка ширины для любого тега, в который заключены ссылки, например **<li>** или **<ul>**, приведет к тому же результату. Если текст кнопки занимает всего одну строку, то можно его центрировать вертикально, чтобы над и под ним были одинаковые промежутки. Просто установите высоту ссылки и назначьте такое же значение свойства межстрочного интервала:

a {  
height: 1.25em;  
line-height: 1.25em;  
}

ПРИМЕЧАНИЕ  
Не обязательно явно назначать свойство **width**, если панель навигации вложена в элемент размет-  
ки веб-страницы, для которого уже установлена ширина. Боковая панель имеет свою ширину, и помещенный в нее маркированный список со ссылками кнопками автоматически приобретает эту ширину.

**ОБХОДНОЙ ПРИЕМ**  
**Если сливаются границы**

Если кнопки панели навигации расположены вплот­ную друг к другу и при этом вы применяете границы, окаймляющие каждую ссылку отдельно, то линии гра­ниц соседних ссылок сливаются между собой и превра­щаются в двойные. Чтобы избавиться от этого, можно добавить границу только к верхней стороне ссылок. Таким образом, мы получим всего одну границу, от­деляющую кнопки, расположенные рядом. Однако в результате этого обходного приема последняя, нижняя ссылка панели управления получается без границы. Чтобы решить проблему, можно либо  
создать стилевой класс с нижней границей и приме­нить его к последней ссылке, либо, что еще лучше, до­бавить нижнюю границу к тегу **<ul>**, завершающему

**Горизонтальные панели навигации**

Первый метод основывается на использовании свойства **display: inline-block** — он прост, но обычно приводит к маленьким расстояниям между кнопками. Этот вопрос можно решить, но для этого придется возиться с кодом HTML. Если нужна горизонтальная навигационная панель с соприкасающимися кнопками, обратитесь к методу с применением плавающих элементов **<ul>.**

**Использование display:inline и display:inline-block.** Самый простой метод создания горизонтальной панели навигации заключается в изменении для элементов списка значения **block** (блочный элемент) свойства **display** на **inline** (линейный элемент). Это делается средствами CSS.

1. Создаем стиль для маркированного списка, чтобы удалить отступы, поля и маркеры.

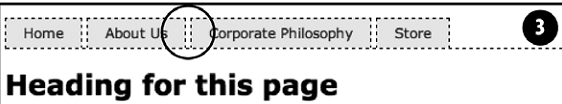
ul.nav {  
 margin-left: 0px;  
 padding-left: 0px;  
 list-style: none;  
 border-bottom: 1px dashed #000;  
}

Какой бы метод вы ни использовали, сначала удалите маркеры и левые отступы тегов **<ul>**, как показано на рис. *1*.  
**Использование display:inline и display:inline-block.** Самый простой метод создания горизонтальной панели навигации заключается в изменении для элементов списка значения **block** (блочный элемент) свойства **display** на **inline** (линейный элемент). Это делается средствами CSS.

**1.** Создаем стиль для маркированного списка, чтобы удалить отступы, поля и маркеры.  
ul.nav {  
 margin-left: 0px;  
 padding-left: 0px;  
 list-style: none;  
 border-bottom: 1px dashed #000;  
}  
В данном случае добавляется нижняя граница, которая появится ниже кнопок







2. Преобразуем пункты списка в линейные элементы. Линейные элементы не создают переносов строк ни перед элементом, ни после него, как это делают блочные. Установка значения **inline** свойства **display** для тегов **<li>** приведет к отображению элементов списка в одну строку (рис, *2*).

**.nav li { display: inline; }**

Вы должны быть уверены в том, что панель навигации кнопок-ссылок не слишком большая. Те кнопки, которые не поместятся в одну строку, перенесутся на другую.

3. Стилизуем ссылки. Можно убрать подчеркивание под ссылками и вместо этого добавить вокруг них окаймляющую границу. Чтобы обеспечить визуальную глубину и реалистичность настоящих кнопок, нужно изменить цвет фона или добавить рисунок. При необходимости добавьте отступы вокруг текста ссылок. Если требуется разнести кнопки друг от друга, назначьте для них **правое поле**. (см. рис, *3*).

СОВЕТ  
Определив ширину ссылок, все кнопки можно сделать одинакового размера.

**!!!** Чтобы эта навигационная панель появлялась в центре страницы, к стилю тега **<ul>** нужно добавить объявление **text-align: center;** . У данной технологии, по сравнению с рассматриваемой далее плавающей технологией, есть одно преимущество — при использовании **inline** и **inline-block** панель навигации можно центрировать, чего нельзя сделать с **float**.

Однако можно заметить, что кнопки не соприкасаются (см. область, обозначенную кружком, на рис. 3). Это связано с тем, как веб-браузеры рассматривают пустое пространство между **<li>**-тегами. Возьмем, например, следующий фрагмент кода HTML:

<ul class=" nav">  
 <li ><a href=" index. html ">Home</a></li >  
 <li><a href="news.html">News</a></li>  
 <li><a href="reviews.html">Reviews</a></li>  
</ul>

Символ возврата каретки после одного тега </li> и перед следующим тегом <li> веб-браузеры рассматривают как пустое пространство. Удалить его можно одним из двух способов.

1. Поместить закрывающий тег **</li>** и открывающий тег **<li>** на одной и той же  
   строке:
2. <ul class="nav">  
    <li><a href="index.html">Home</a></li><li>  
    <a href="news.html">News</a></li><li>  
    <a href="reviews.html">Reviews</a></li>  
   </ul>
3. Обычно так код не записывают, и программы типа Dreamweaver так бы не сделали. Чтобы удалить пустое пространство, нужно зайти в код и внести изменения вручную.

2. Добавить к элементам списка правое поле с отрицательным значением. Например, в стиль **list** в показанном выше шаге 2 можно внести следующее изменение:  
.nav li {  
display: inline;  
margin-right: -5px;  
}  
Отрицательное значение поля приводит к накладке следующего элемента списка на 5 пикселов, закрывая просвет между кнопками. Проблема при использовании данной технологии заключается в том, что конкретное применяемое значение варьируется в зависимости от размера текста — 5 пикселов могут сработать, а могут и нет, поэтому для получения нужного значения следует провести эксперимент.

**Осуществление преобразований, переходов и анимации**  
**с помощью CSS**

**Преобразования**

**Вращение**

**transform: rotate(10deg);** поворот на 10 градусов, можно использовать отрицательные и положительные значения.

-webkit-transform: rotate(10deg);  
-moz-transform: rotate(10deg);  
-o-transform: rotate(10deg);  
-ms-transform: rotate(10deg);  
transform: rotate(10deg);

**Масштабирование**

transform: scale(2); увеличит элемент в двое по высоте и ширине

transform: scaleX(2); увеличит элемент в двое по ширине

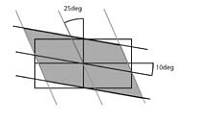
transform: scaleY(2); увеличит элемент в двое по высоте

**Перемещение**

transform: translate(50px,100px); переместит элемент на 50 пикселей вправо и на 100 вниз.

**Наклон**

transform: skew(25deg,10deg);

 Первое значение задает угол от **0deg** до **360deg**, действующий в направлении против часовой стрелки от верхней части элемента.

Вторым значением является угол от **0deg** до **360deg**. Но этот угол действует в направлении по часовой стрелке с правой стороны элемента.

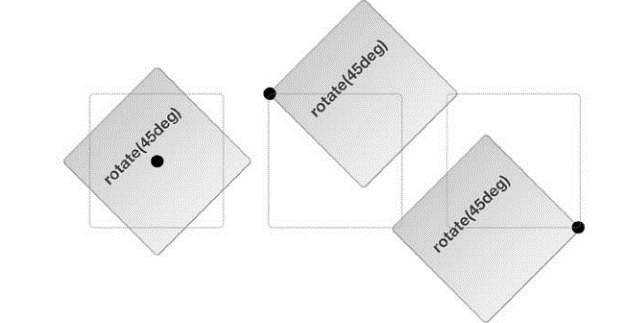
Как и в случае с **translate** и **scale**, в CSS3 предлагаются отдельные функции для осей ***X***и ***Y***: **skewX** и s**kewY**.

**Множественные преобразования**

transform: skew(45deg,0deg) scale(.5) translate(400px,500px) rotate(90deg);

**Исходная точка**  
Обычно, когда к элементу применяется преобразование, в качестве точки начала преобразования браузер использует центр элемента.

Но в CSS3 разрешается изменять точку преобразования, используя свойство **transform-origin**. Это свойство работает точно так же, как и ранее рассмотренное свойство **background-position**. Для него можно указывать **ключевые слова**, **абсолютные** значения и **относительные** значения в **em** и **процентах**.

Можно также использовать пиксельные значения:  
 transform-origin: 0 0;

или проценты:  
transform-origin: 0% 0%;

transform-origin: right bottom;

Что эквивалентно следующему объявлению:  
 transform-origin: 100% 100%;

**Переходы**

Чтобы переход заработал, нужно следующее:

* **Два стиля.** Один стиль должен представлять начальный вид элемента, например красную кнопку перехода, а второй — его конечный вид: синюю кнопку перехода.
* **Свойство transition.** В CSS3 добавляется свойство **transition**. Обычно свойство **transition** применяется к исходному стилю, который определяет внешний вид элемента до начала анимации.
* **Инициатор.** Инициатор представляет собой действие, вызывающее изменение от одного стиля к другому. В CSS для запуска анимации можно использовать несколько псевдоклассов. Наиболее часто применяется псевдокласс **:hover**. С его помощью можно анимировать изменение, происходящее от обычного  
  появления элемента и до его внешнего вида, который появляется, когда посетитель проводит над ним указатель мыши. Кроме того, у вас в активе такие псевдоклассы, как **:active** (связанный со щелчком кнопкой мыши на элементе), **:target** (связанный с элементом, ставшим целью перехода по ссылке) и :**focus** (связанный с переходом на ссылку с помощью клавиши табуляции, или со щелчком по полю формы, или с переходом на это поле с помощью клавиши табуляции). Кроме того, для динамической смены стиля любого тега можно воспользоваться JavaScript

|  |  |
| --- | --- |
| .navButton { background-color: orange; } .navButton:hover { background- color: blue; } | navButton { background-color: orange; **transition-property: background-color; transition-duration: 1s;** } .navButton:hover { background- color: blue; } |
| **Первое свойство** — **transition-property** — указывает на анимируемое свойство. Можно указать одно свойство (как в показанном выше примере), воспользоваться ключевым словом all для анимирования всех изменяемых CSS-свойств или применить списком с запятой в качестве разделителя для указания более чем одного свойства (но не всех свойств).  transition-property: color, background-color, border-color; или  transition-property: all; | |

указать продолжительность анимации, используется свойство **transition-duration**. Ему передается или значение в секундах, или значение в миллисекундах (тысячных долях секунды). Например, чтобы переход занимал полсекунды, можно использовать либо:

transition-duration: .5s;

либо

transition-duration: 500ms;

Можно даже указать отдельную продолжительность для каждого анимируемого свойства.

transition-property: color, background-color, border-color;

transition-duration: .25s, .75s, 2s;

**Распределение скорости выполнения перехода по времени**

Чтобы анимированный переход заработал, нужно только установить значения для свойств **transition-property** и **transition-duration**. Но с помощью свойства **transition-timing-function** можно также контролировать и скорость хода анимации. В предназначении этого свойства нетрудно и запутаться: оно управляет не продолжительностью анимации (для этого есть свойство **transition-duration**), а скоростью хода анимации. Например, можно начать анимацию медленно, а затем быстро ее завершить, создавая эффект незаметного в начале и быстрого в конце изменения фонового цвета.

Свойство **transition-timing-function** может получать одно из пяти ключевых слов: **linear, ease**, **ease-in**, **ease-out** и **ease-in-out**. Если функцию регулирования скорости не задавать, браузер будет использовать метод **ease**, при котором анимация начинается медленно, ускоряется к середине и замедляется к концу, предоставляя более естественное изменение. Вариант linear дает равномерное изменение на протяжении всего периода анимации. Ни один из вариантов не имеет заметных преимуществ перед другими, они просто предлагают различную общую картину, и для определения своих предпочтений нужно просмотреть все варианты.

transition-timing-function: ease-in-out;

transition-timing-function: cubic-bezier(.20, .96, .74, .07); Использует кривые Безье для анимации переходов.

**Задержка начала перехода**

Например, если вы хотите подождать полсекунды, прежде чем начать анимацию, можно добавить следующий код:

transition-delay: .5s;

. navButton {  
color: black;  
background-color: #FF6603;  
border: 5px solid #660034;  
transition-property: color, background-color, border-color;  
transition-duration: 1s, 1s, .5s;  
transition-delay: 0, 0, 1s;  
}  
.navButton:hover {  
color: white;  
background-color: #660034;  
border-color: #FF6603;  
}

!!! СОВЕТ  
Обычно свойства перехода помещаются в начальный **(.navButtonon** в предыдущем примере), а не в конечный стиль **(.navButton:hover**). Но при использовании раскрывающегося меню CSS возникают некоторые сложности. Проблема в том, что при задании раскрывающихся меню с помощью CSS они слишком быстро пропадают при случайном выходе указателя мыши за их пределы. Но с помощью свойства **transition-delay** можно заставить меню быстро появляться и медленно исчезать. Для этого добавьте в исходный стиль следующий код:  
**transition-delay: 5s;**  
Затем добавьте задержку к стилю **:hover**:  
**transition-delay: 0;**Каким бы странным этот код ни показался, он заставляет переход по **:hover** происходить немедленно, без задержки. А вот возвращение к обычному стилю (при котором меню исчезает) занимает **5** секунд. За это время посетитель успеет вернуть свою непослушную мышь обратно на меню, пока оно не исчезло.

**Краткая запись свойства transition**

Это свойство связывает все другие свойства в одно. Нужно просто перечислить через запятую свойство, продолжительность, функцию распределения скорости по времени и задержку. Например, для анимации всех CSS-свойств на 1 секунду, используя функцию **ease-in** и полусекундную задержку, нужно написать следующий код:

transition: all 1s ease-in .5s;

если нужно просто анимировать переход всех CSS-свойств на 1 секунду, нужно написать следующий код:

transition: all 1s;

Если нужно анимировать изменение фонового цвета, включите в список именно это свойство:

transition: background-color 1s;

в множественном варианте:

transition: color 1s,  
 background-color 1s,  
 border- color .5s 1s;

**АНИМАЦИЯ**

Создание анимации проходит в два приема.

1. **Определение анимации.** Включает определение ключевых кадров со списком анимируемых CSS-свойств.

2. **Применение анимации к элементу.** После определения анимации ее можно назначить любому количеству элементов страницы. Можно даже задать для каждого элемента разные распределения скорости выполнения, задержки и другие свойства анимации. То есть одну и ту же анимацию на странице можно использовать с немного разными настройками несколько раз.

**Определение ключевых кадров**

Первым делом нужно настроить ключевые кадры. Реализующий их синтаксис может показаться немного странным, но его основная структура имеет следующий вид:

@keyframes *имяАнимации* {  
 from {  
 /\* Здесь перечисляются CSS-свойства \*/  
 }  
 to {

/\* Здесь перечисляются CSS-свойства \*/  
 }  
 }

Сначала идет ключевое слово **@keyframes**. За ним следует имя, которым вы называете анимацию. Позже это имя будет использоваться при применении анимации к элементу страницы, поэтому оно должно быть описательным, например  
**fadeOut** или **fadeIn**.

Затем добавляются как минимум два ключевых кадра. В показанном выше примере ключевые слова **from** и **to** используются для создания начального ключевого кадра (**from**) и конечного ключевого кадра (**to**). Внутри каждого ключевого кадра находится одно или несколько CSS-свойств, как будто создается стиль. Фактически каждый ключевой кадр можно представить в виде простого CSS-стиля, содержащего одно или несколько свойств CSS. Предположим, к примеру, что нужно создать анимацию постепенного появления элемента в поле зрения. Можно начать со значения **opacity**, равного **0** (невидимый), и закончить значением этого свойства, равным  **1** (полностью видимый):

@keyframes fadeIn{  
 from {  
 opacity: 0;  
 }  
 to {  
 opacity: 1;  
 }  
**}**

С помощью процентных значений можно определить несколько ключевых кадров. Процентное значение указывает место на общей продолжительности анимации, где должно произойти изменение. Предположим, к примеру, что нужно создать эффект изменения фона элемента с желтого на синий, а потом на красный. Для этого можно написать следующий код:

@keyframes backgroundGlow {  
 from {  
 background-color: yellow;  
 }  
 50% {  
 background-color: blue;  
 }

to {  
 background-color: red;  
 }  
}

В данном случае синий фон появится в середине анимации. Если нужно, чтобы желтый цвет присутствовал в фоне дольше, а синим становился после трех четвертей времени анимации, используйте не 50%, а 75 %. Таким же образом можно продолжать добавлять дополнительные ключевые кадры (например, на 25, 66 % и т. д.).

ПРИМЕЧАНИЕ  
Ключевое слово **from** можно заменить значением **0%**, а ключевое слово **to** — значением **100%**

Но одним свойством CSS можно не ограничиваться. Внутри ключевого кадра может находиться любое количество пригодных к анимации свойств — backgroundcolor, opacity, width, height и т. д.:

@keyframes growAndGlow {  
 from {  
 background-color: yellow;  
 }  
 50% {  
 transform: scale(1.5);  
 background-color: blue;  
 }  
 to {  
 transform: scale(3);  
 background-color: red;  
 }  
}

В этом примере наряду с троекратным изменением цвета фона происходит увеличение масштаба элемента, размеры которого в итоге становятся втрое больше первоначальных.

Предположим, например, что вам нужно, чтобы сначала у содержимого тега **<div>** был **желтый** цвет фона. Затем нужно, чтобы цвет **сменился на синий**, некоторое время **оставался синим**, а затем **сменился на красный**. То есть **нужна пауза** в середине анимации, при которой цвет не менялся, а затем необходима новая смена цвета.

@keyframes glow {  
 from {  
 background-color: yellow;  
 }

25%, 75% {  
 background-color: blue;  
 }  
 to {  
 background- color: red;  
 }  
}

От отметки **25%** до отметки **75%** фон будет сохранять чисто-синий цвет, перед тем как к окончанию анимации превратиться в красный.

Предположим, например, что нужно провести еще одну анимацию фонового цвета, но на этот раз менять цвет от желтого к синему, затем к оранжевому, после чего к синему, к оранжевому и к красному:

@keyframes glow {  
 from {  
 background-color: yellow;  
 }  
 20%, 60% {  
 background-color: blue;  
 }  
 40%, 80% {  
 background-color: orange;  
 }  
 to {  
 background- color: red;  
 }  
}

В данном случае фоновый цвет превратится в синий на отметке 20%, в оранжевый — на отметке 40%, затем опять в синий на отметке 60%, и перед тем как стать в конце анимации красным, он еще один последний раз на отметке 80% превратится в оранжевый.

**ПРЕФИКСЫ ПРОИЗВОДИТЕЛЕЙ:**

@-webkit-keyframes

@-moz-keyframes

@-o-keyframes

**Применение анимации**

Анимацию можно добавить к любому стилю любого элемента страницы. Если просто добавить анимацию кстилю, который тут же применяется кэлементу, например к стилю тега **h1**, анимация будет применена при загрузке страницы.  
Этот прием можно использовать для добавления на страницу вводной анимации, которая увеличивает масштаб логотипа, помещая его в верхний левый угол страницы, или создает свечение конкретного блока информации, привлекая к нему внимание. Кроме того, анимацию можно применить к одному из псевдоклассов, включая  
**:hover, :active, :target** или **:focus**

Рассмотрим простой пример. Предположим, нужно, чтобы при загрузке страницы в поле зрения появился **div-контейнер** с важным объявлением. Этому **div-контейнеру** был назначен класс с именем **announcement**:

1. Создайте анимацию постепенного появления, задав правило **@keyframes**:  
   @keyframes fadeIn {  
    from { opacity: 0; }  
    to { opacity: 1; }  
    }
2. Примените эту анимацию к стилю, предназначенному для тега **<div>:**  
   .announcement {  
    animation-name: fadeIn;  
    animation-duration: 1s;  
    }

Свойство **animation-name** просто сообщает браузеру, какую анимацию нужно применить. Ему указывается то самое имя, которое предоставлялось анимации при выполнении шага 1. Свойство **animation-duration** устанавливает время, которое займет анимация от старта до финиша. В данном примере перечисляются только два свойства анимации, но вы можете (и, наверное, захотите) поместить в стиль и другие, не связанные с анимацией свойства. Например, в реальности к нашему стилю .announcement будут, наверное, добавлены такие свойства, как **width**, **background-color, border** и т. д.  
ПРИМЕЧАНИЕ  
С технической точки зрения помещать имя анимации в кавычки ('**fadeIn**') не требуется, и в этом примере так не делается, но помещение в кавычки поможет избежать конфликтов, возникающих при использовании в качестве имени анимации какого-нибудь ключевого слова CSS.

Как и в случае использования правила **@keyframes**, каждое из свойств анимации требует префиксов, определяющих производителя, поэтому показанный выше стиль **.announcement** для работы на как можно большем количестве браузеров нужно переписать следующим образом:  
.announcement {  
animation-name: fadeIn;  
animation-duration: 1s;  
-webkit-animation-name: fadeIn;  
-webkit-animation-duration: 1s;  
-moz-animation-name: fadeIn;  
-moz-animation-duration: 1s;  
-o-animation-name: fadeIn;  
-o-animation-duration: 1s;  
animation-name: fadeIn;  
animation-duration: 1s;  
}

Предположим, что вы создаете одну анимацию с именем f**adeIn**, чтобы анимировать появление элемента, и еще одну анимацию с именем **blink**, чтобы получить мигающий цвет фона. Чтобы применить к элементу обе анимации, нужно указать список имен с запятой в качестве разделителя:

animation- name: fadeIn, blink;

Чтобы задать этим анимациям разные параметры времени, нужно указать список этих параметров с запятой в качестве разделителя:

animation-name: fadeIn, blink;  
animation-duration: 1s, 3s;

Порядок применения параметров времени совпадает с порядком указания имен анимаций. Например, первая анимация получает время, указанное в списке первым. В показанном выше примере продолжительность **fadeIn** составляет 1 секунду, а продолжительность **blink** — 3 секунды.

**Распределение скорости выполнения анимации по времени**

**animation-duration** позволяет управлять продолжительностью анимации. Как и в случае с переходами (**transitions**), для задания продолжительности можно использовать миллисекунды (**например, 750ms**) или секунды (**например, .75s**).

анимацию можно начинать медленно и завершать быстро, используя для этого кубическую кривую Безье или одно из встроенных ключевых слов, определяющих метод: **linear, ease, ease-in,** **ease-out, ease-in-out**. Все это работает точно так же, как и в ранее рассмотренном случае с переходами.  
Свойство **animation-timing-function** можно использовать для управления всей анимацией или только для конкретных ключевых кадров. Например, чтобы применить функцию **ease-out** для представленной ранее анимации **fadeIn** (шаг 1 в предыдущей последовательности действий), добавьте функцию распределения скорости по времени к стилю **.announcement** (шаг 2 в предыдущей последовательности действий):

.announcement {  
animation-name: fadeIn;  
animation-duration: 1s;  
animation-timing-function: ease-out;

**Завершение анимации**

Переходы являются анимациями, выполняемыми однократно, например указатель мыши проходит над кнопкой и она увеличивается в размерах. А вот анимации благодаря свойству **animation-iteration-count** могут запускаться один, два и более раза или непрерывно. Если нужно запустить анимацию 10 раз (возможно, в виде десятикратного появления и исчезновения элемента), добавьте к анимируемому стилю следующий код:

animation-iteration-count: 10;

Если же необходимо, чтобы анимация повторялась непрерывно, для этого свойству **animationiteration-count** передается одно ключевое слово: **infinite**.

animation-iteration-count: infinite;

Обычно при многократном запуске анимации веб-браузер запускает анимацию буквально с самого начала. Поэтому, если анимируется превращение желтого фонового цвета в синий и это повторяется дважды, браузер покажет, как желтый блок превращается в синий, а затем внезапно вернется желтый блок, который превратится в синий еще раз. Этот эффект может слишком сильно резать глаз. Эффект будет выглядеть лучше, если через некоторое время после анимации браузер просто воспроизведет эффект в обратном порядке. Именно так работают переходы: например, когда указатель мыши проходит над элементом, браузер анимирует переход из обычного состояния к состоянию, заданному при проходе над элементом.  
Когда указатель мыши выходит за пределы элемента, браузер просто воспроизводит анимацию в обратном порядке, возвращая внешний вид элемента в обычное состояние. Чтобы анимация при нечетных проигрываниях шла в прямом, а при нечетных — в обратном порядке, используется свойство **animation-direction** и ключевое слово **alternate**. Например, чтобы элемент постепенно исчезал, а затем снова появлялся, можно создать анимацию **fadeOut**:

@keyframes fadeOut {  
 from { opacity: 1; }  
 to { opacity: 0; }  
}

А затем можно воспроизвести эту анимацию дважды, меняя при втором воспроизведении ее направление на противоположное:

.fade {  
 animation-name: fadeOut;  
 animation-duration: 2s;  
 animation-iteration-count: 2;  
 animation-direction: alternate;  
}

Этот код заставляет веб-браузер запустить анимацию **fadeOut** в отношении любого элемента, имеющего класс **fade**. Анимация должна продолжаться 2 секунды, а затем повторяться. Поскольку свойство **animation-direction** имеет значение **alternate**, анимация в первый раз приведет к исчезновению элемента (его свойство непрозрачности от полностью непрозрачного (значение opacity равно  1) постепенно поменяется на 0 — полностью прозрачное), а затем при втором проигрывании все пойдет вспять (значение **opacity** будет изменяться от 0 до 1) и элемент снова появится на экране.

СОВЕТ  
Чтобы анимация воспроизводилась несколько раз, но в конечном итоге возвращала элементу его первоначальный вид, используйте четное количество итераций и установите для свойства **animation-direction** значение **alternate**.

Неважно, сколько раз вы заставите браузер воспроизводить анимацию, но как только она завершится, браузер отобразит элемент в **его первоначальном виде**. Предположим, например, что изображение анимируется таким образом, чтобы его размер медленно увеличивался до достижения двойного размера. Как только анимация будет завершена, веб-браузер тут же вернет изображению его первоначальный размер, **создавая слишком резкий визуальный эффект**. К счастью, браузер можно заставить сохранить элементу тот вид, который у него был по завершении анимации.  
Для этого нужно установить для свойства **animation-fill-mode** значение **forwards**.

**animation-fill-mode: forwards;**

Это свойство применяется к анимируемому элементу вместе с такими свойствами, как **animation-name, animation-duration**, и другими свойствами, имеющими отношение к анимации.

**Краткая запись свойства animation**

**animation**: В одном этом свойстве сочетаются такие свойства, как **animation-name**, **animation-duration, animation-timing-function, animation-iterationcount, animation-direction, animation-delay** и **animation-fill-mode**. Можно, например, взять следующий код:

. fade {  
 animation- name: fadeOut;  
 animation-duration: 2s;  
 animation-timing-function: ease-in-out;  
 animation-iteration-count: 2;  
 animation-direction: alternate;  
 animation-delay: 5s;  
 animation-fill-mode: forwards;  
}

и придать ему следующий вид:

.fade {  
 animation: fadeOut 2s ease-in-out 2 alternate 5s forwards;  
}

Значения свойств нужно перечислять в ранее упомянутом порядке: **имя, продолжительность, функция распределения скорости по времени, количество раз, направление, задержка и режим заполнения**. Кроме того, нужно поставить между значениями запятые. Обязательными являются только имя и продолжительность. Все остальные значения являются необязательными.  
Если к элементу нужно применить несколько анимаций, следует воспользоваться списком их свойств с применением запятой в качестве разделителя.

animation: fadeOut 2s ease-in-out 2 alternate 5s forwards,

fadeIn 2s ease-in-out 2 alternate 3s forwards;

**Приостановка анимации**

В CSS3 имеется еще одно свойство анимации — animation-play-state. Оно предназначено для управления проигрыванием анимации. Этому свойству передается одно из двух ключевых слов — running или paused. Чтобы приостановить анимацию, нужно просто применить к стилю следующее объявление:

animation-play-state: paused;

Но фактически применить это свойство к CSS можно только с помощью псевдокласса. Как и в случае с переходами, для приостановки анимации нужен инициатор. Один из вариантов предусматривает приостановку анимации при проходе  
над подвергающимся анимации элементом указателя мыши. В следующем примере используется стиль для класса **fade**:

.fade {  
 animation: fadeOut 2s ease-in-out 2 alternate 5s forwards,  
 glow 5s;  
}

Этот код запускает две анимации — fadeOut и glow — для любого элемента, к которому применен класс fade. Предположим, что нужно позволить посетителям приостанавливать данную анимацию при проводе над ней указателя мыши. Для этого нужно только добавить еще один стиль:

. fade: hover {

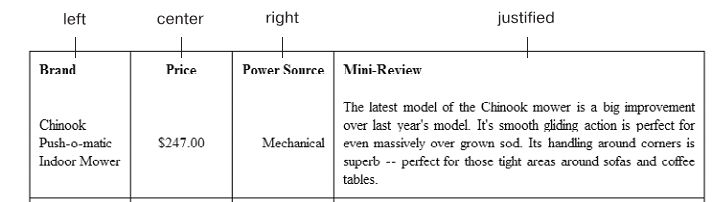
animation- play- state: paused;  
}

Разумеется, вам понадобятся и все версии с префиксами производителей. Более эффективным способом приостановки анимации будет динамическое применение свойства **animation-play-state** к элементу с помощью JavaScript.

**Форматированиетаблиц и форм**

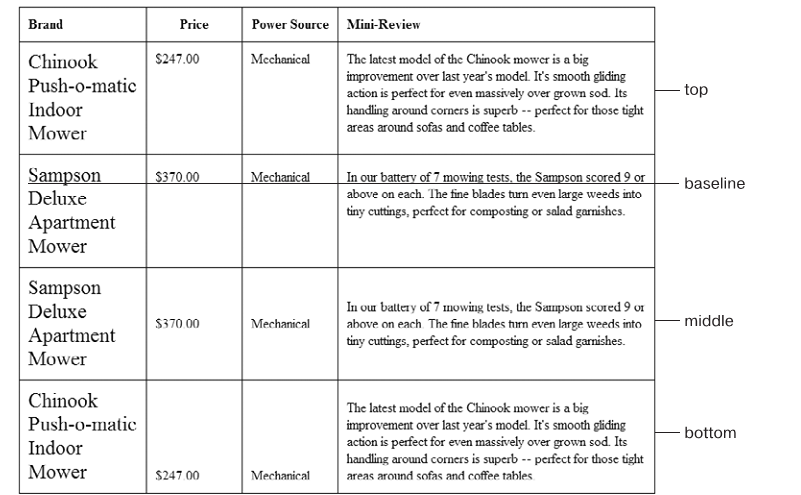
**Настройка горизонтального и вертикального выравнивания**

table { text-align: right; }



Это свойство удобно использовать в тегах <th>, так как браузеры обычно выравнивают его по центру. Простой стиль вида th { text-align: left; } выровняет и заголовки таблицы.

У ячеек в таблицах есть также такой параметр, как высота. Обычно браузеры выравнивают содержимое вертикально по центру. Вы можете изменить это с помощью свойства vertical-align. Примените одно из этих четырех значений: top, baseline, middle или bottom.



**Создание границ**

**Управление промежутком между ячейками таблицы.** Если не указано другое, браузеры расставляют в таблице интервалы между ячейками около 2 пикселов. Этот промежуток действительно заметен, когда вы задаете границы для ячеек.

В CSS доступно свойство **border-spacing**, которое можно использовать, чтобы контролировать размер этого промежутка.

table {  
 border- spacing: 0;  
}

**Удаление двойных границ.** Даже если вы уберете промежутки между ячейками, границы, заданные для ячеек, будут удваиваться. Это происходит потому, что нижняя граница одной ячейки добавляется к верхней границе лежащей внизу ячейки, и в итоге образуется линия, которая в два раза толще заданной ширины границы

Лучший способ избавиться от этого — использовать свойство **border-collapse**. Имеет два значения **separate** и **collapse**

**table { border-collapse: collapse; }** – убирает расстояние между границами и избавляет от двойных границ, если используется, то значение **border**-**spacing**: игнорируется.

**Применение стилей к строкам и столбцам**

Используя ранее рассмотренный селектор **nth-of-type**, можно добавить четным и нечетным строкам разный фон:

tr:nth-of-type(odd) { background-color: red; }  
tr:nth-of-type(even) { background-color: blue; }

процессе форматирования столбцов нужно немного схитрить. В HTML есть теги **<colgroup>** и **<col>**, указывающие на группу столбцов и на один отдельный столбец соответственно. Вы можете добавить по одному тегу **<col>** для каждого столбца в таблице и далее идентифицировать их с помощью класса или ID.  
Для этих тегов есть только два вида свойств: **width** и свойства фона **(backgroundcolor, background-image** и т. д.). Однако и они могут быть очень полезными. Когда вы хотите установить ширину всех строк в столбце, можете пропустить все атрибуты HTML и просто применить к столбцам стиль, используя стиль, назначенный тегу **<col>**. Скажем, у вас есть следующий кусок кода HTML: **<col class="price">**. Вы можете добавить этот стиль к таблице стилей и установить ширину каждой ячейки в данном столбце равной 200 пикселам:

.price { width: 200px; }

Чтобы выделить столбец, вы можете воспользоваться свойствами фона. И снова считайте, что у вас есть тег **<col>**, к которому применен класс **price**:

**.price { background- color: #F33; }**

Имейте в виду, что фон в столбцах помещается под ячейками, поэтому, если вы установите фоновый цвет или рисунок в тегах **<td>** или **<th>**, фон столбцов не будет виден.

**Разметка страницына основе плавающих**  
**элементов**

**Простой дизайн с двумя столбцами, как на рис.**



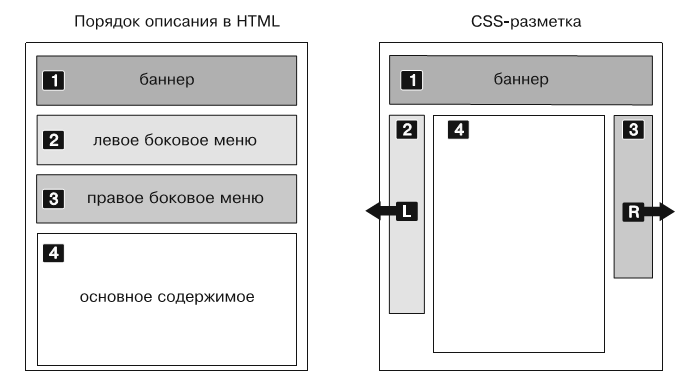
1. **Укажите вокруг каждого столбца тег <div> с атрибутом класса или ID.** На рис. заголовки новостей, перечисленные в левой части, заключены в один раздел (**<div class = "news">**), а главное содержимое страницы — в другой **(<div class = "main">**).
2. **Переместите тег <div> с боковой панелью вправо или влево.** Когда вы работаете с плавающими элементами, важен порядок исходного кода(порядок, в котором вы добавляете HTMLк файлу). HTMLдля плавающего элемента должен появиться ***перед***HTML для элемента, который указывается вокруг него.
3. **Установите ширину для плавающей боковой панели.** Всегда задавайте плавающим элементам ширину. Таким образом вы позволите браузеру создать место для другого содержимого, чтобы получить эффект обтекания. У ширины может быть фиксированный размер, такой как **170px** или **10em**. Вы также можете использовать проценты для гибкого дизайна, основанного на ширине окна браузера
4. **Добавьте левый край к главному содержимому.** Если боковая панель короче другого содержимого на странице, то текст из главного столбца обтекает панель снизу. Добавление левого края, большего или равного ширине боковой панели, выравнивает главное содержимое страницы, создавая иллюзию второго столбца:  
   **.main { margin-left: 180px ; }**

Удобно делать левый край чуть больше по ширине, чем боковая панель: это добавляет промежуток между двумя элементами. Если для установки ширины боковой панели вы используете проценты, то задавайте немного большее значение для левого края.

Избегайте установки ширины для раздела **div** с главным содержимым — браузеры расширяют его, чтобы оно занимало доступное место. Даже если вы хотите иметь дизайн с фиксированной шириной, вам не нужно устанавливать ширину для

**Разметить страницу на три столбца также нетрудно.**

Сначала добавьте другой тег **<div>** между этими двумя столбцами и переместите его вправо. Затем добавьте правый край к среднему столбцу так, чтобы, если текст в нем окажется длиннее новой правой боковой панели, он не обтекал боковую панель.

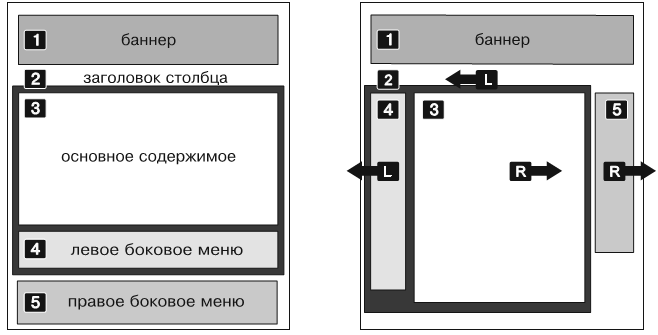


**Перемещение всех столбцов**

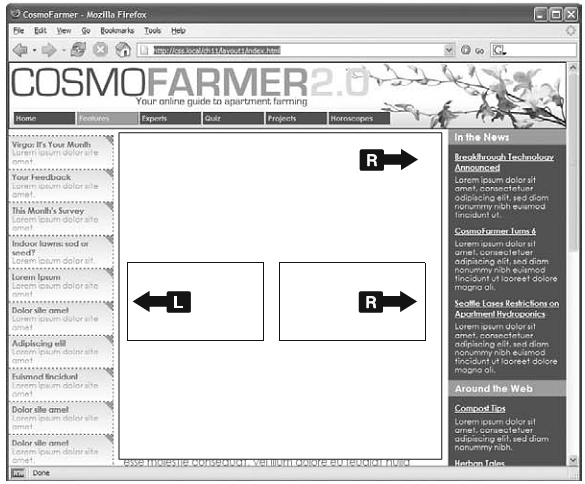
Когда вы перемещаете все столбцы в дизайне, вы должны обратить пристальное внимание на ширину каждого столбца. Если совокупная ширина всех столбцов меньше доступного места, например, когда окно браузера меньше или столбцы помещены внутри другого тега **<div>** с установленной шириной, то ***последний столбец опускается вниз!***

Вдобавок, перемещая не только боковые панели, вы сможете изменить порядок своих разделов **div** в HTML.

Вы можете даже поместить описание главного содержимого перед HTML-кодом для боковых панелей, определяя для него и левой боковой панели один тег **<div>** и перемещая его влево. Затем внутри этого тега **<div>** нужно переместить основное содержимое вправо, а левую боковую панель — влево (см. рис *внизу*). Вуаля! HTML-код главного столбца идет перед остальными тегами **<div>**.



Разметка на рис.  иллюстрирует другую полезную методику — перемещение элементов *внутри* плавающих элементов. Предположим, что разделов главного содержимого *(3)* и левой боковой панели *(4)* не существует, а были оставлены только «обертка» столбца *(2)* и правая боковая панель *(5)*. У вас будет просто базовый дизайн с двумя столбцами, где один столбец перемещен влево, а другой — вправо. На самом деле это все еще дизайн с двумя столбцами, хотя два раздела (*3* и *4*) помещены внутри «обертки» столбца *(2)*. Различие в том, что левый столбец сам разделен на два столбца.



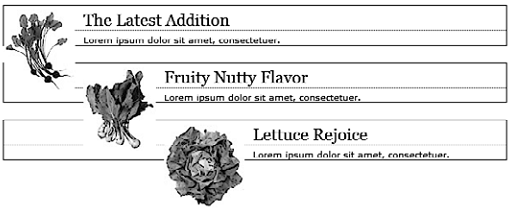
Чтобы добиться такого размещения объектов, создайте столбцы в столбцах, передвигая элементы внутри других плавающих элементов. В среднем столбце предусмотрена область подсказок, которая позволяет добавить простую заметку с двумя колонками, что придает привлекательности странице.  
На рис., видно, что неважно, в каком направлении перемещается контейнер (в этом случае — вправо), — вы просто передвигаете два дополнительных столбца влево и вправо.

**Решение проблем плавающих элементов**



Для нижнего колонтитула и других элементов, которые должны оказаться у основания страницы, вы должны устранить как левое, так и правое обтекания:

**#footer { clear: both; }**



Перемещаемый элемент может выйти за пределы содержащего его блока. Если в блоке определены фоновый цвет или граница, то выходящие элементы могут выглядеть так, как будто даже не являются частью блока *(вверху)*. Кроме того, перемещаемый элемент может врезаться в другие элементы, создавая «ступенчатый» эффект

Есть много способов избавиться от проблем, возникающих с перемещаемыми элементами. Хорошо, когда под рукой есть не одно, а несколько решений, поэтому перечислю наиболее популярные из них.

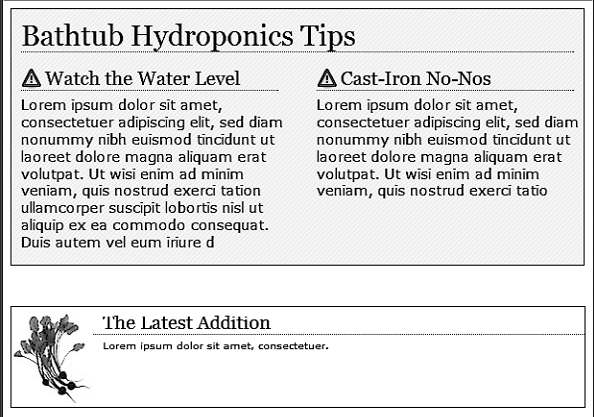
1. **Добавьте «освобождающий» элемент к основанию блока.** Это решение — наиболее простое. Нужно лишь добавить тег, определяющий, например, разрыв строки или горизонтальную линию, в качестве последнего элемента в теге **<div>**, содержащем перемещаемый элемент (то есть прямо перед закрывающим тегом **</div>**). Затем используйте свойство **clear**, чтобы укрепить этот дополнительный тег под перемещаемыми элементами.  
Этот метод расширяет блок, выявляя его фон и границу. Вы можете указать разрыв строки — **<br>** (HTML) или **<br/>** (XHTML) — и добавить к нему класс: **<br class = "clear"/>**.

Затем создайте для него такой стиль:

**br.clear { clear: both; }**

**с br не сработало, только с добавлением <hr>: hr.clear { clear: both; }**

**2.Сделайте перемещаемым элемент-контейнер.** Более легкий путь состоит в том, чтобы просто сделать перемещаемым тег **<div>**, который содержит плавающие элементы. Перемещаемый контейнер **<div>** расширяется так, чтобы полностью вмещать любые плавающие элементы. На рис., *вверху*, тег **<div>,** содержащий заголовок и два плавающих столбца, перемещен в левую сторону страницы.  
При необходимости вся его область — фон и границы — расширятся, чтобы соответствовать всему, что находится внутри, включая перемещаемые элементы. Это странно, но это так.  
Если вы выбрали этот путь, убедитесь в том, что добавили свойство **clear** к любому элементу, который следует за перемещаемым контейнером. Так вы гарантируете, что следующий элемент будет находиться под контейнером.



**3.Используйте свойство overflow:hidden.** Другой распространенный метод состоит в добавлении следующих двух свойств к стилю блока-контейнера:

**overflow: hidden;**

Свойство **overflow:hidden** — одно из странностей CSS. Оно принуждает контейнер расширяться и содержать плавающие элементы. В целом этот метод работает очень хорошо. Тем не менее, если у вас есть абсолютно расположенные элементы внутри контейнера, они могут не отображаться. Вы можете попасть и, когда появляются раскрывающиеся элементы, кажется, что они находятся за пределами элемента-контейнера. Если это так, применяйте какой-либо другой способ.

**3.Воспользуйтесь методом Micro Clear Fix.** Чтобы воспользоваться ею, нужно добавить к таблице стилей два различных стиля, один из которых будет применяться всеми современными браузерами, а другой — Internet Explorer 7  
и более ранними версиями. Все это имеет следующий вид:  
**.clear:after {  
 content: " ";  
 display: table;  
 clear: both;  
}  
.clear {  
 zoom:1;  
}**

Последний стиль заставляет Internet Explorer 6 и 7 «получить разметку», как описано во врезке «Информация для опытных пользователей. Получили разметку?» в подразделе «Предотвращение выпадения плавающего элемента с помощью свойства **box-sizing**» данного раздела.

**Простой способ создания нескольких колонок**

Основной синтаксис очень прост: для задания количества колонок можно воспользоваться свойством column-count, для задания разрывов меду ними — свойством column-gap, а для рисования ли­ний между колонками — свойством column-rule.  
Чтобы установить стиль, размер и цвет линий между колонками, используется точно такой же синтаксис, как и для границ.

.multicol {  
 column-count: 3;  
 column-gap: 1em;  
 column-rule: 1px dotted black;  
}

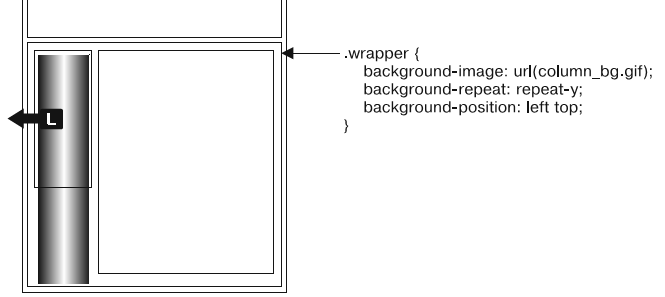
Следует напомнить, что Internet Explorer 9 и более ранние версии не разбираются в свойствах задания нескольких колонок. Кроме того, нужно использовать префиксы производителей для Chrome и Safari (-webkitcolumn-count, -webkit-column-gap и –webkit-columnrule) и для Firefox (-moz-column-count, -moz-columngap и -moz-column-rule). А вот Opera понимает версию этих свойств без префиксов.

**Создание столбцов на всю высоту**

Высота плавающего элемента обычно определяется его содержимым. Когда содержимого немного, элемент не очень высокий. Поскольку фоновое изображение или фоновый цвет заполняют только плавающий элемент, у вас могут  
получиться окрашенные столбцы, которые не доходят до основания страницы, как в обведенных овалом областях на рис.

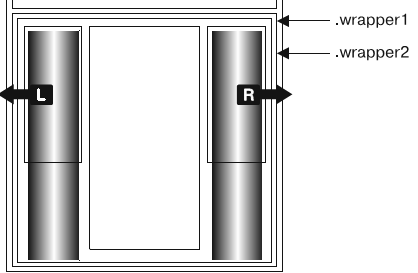


Решитьпроблему можно обернув боковую панель и содержимое в тег div и добавив с одной стороны фоновое изображение равное ширине боковой панели:



Если у нас две колонки, тогда их и содержимое оборачиваем в два div тега и добавляем в один изображение слева, а во второй справа:

.wrapper1 { background: url(images/col1\_bg.gif) repeat-y left top; }  
.wrapper2 { background: url(images/col2\_bg.gif) repeat-y **right** top; }



Основная проблема технологии искусственных столбцов связанас тем, что ее очень трудно заставить работать, когда у всех столбцов ширина задана в процентах. Если ширина боковых панелей задана в процентах от ширины окна браузера, они могут быть уже, или шире в зависимости от характеристик монитора посетителя. Технология искусственных столбцов требует помещения графики в элемент, являющийся контейнером. Эта графика имеет определенную ширину и не будет масштабироваться при изменениях ширины окна браузера, а соответственно, и ширины столбцов.

***Эту проблему можно решить с помощью градиентов:***

Предположим, что есть конструкция из трех столбцов. Первый столбец имеет ширину 25 %, второй — 50 %, и третий — 25 %. Нужно чтобы у первого столбца фоновым цветом был красный, у второго — белый, а у третьего — синий. Для этого сделайте следующее.

***Заключите все три столбца в элемент-контейнер***:  
**<div class="wrapper">  
<div class="sidebar1"> … сюда помещается содержимое …</div>  
<div class="main"> … сюда помещается содержимое …</div>  
<div class="sidebar2"> … сюда помещается содержимое …</div>  
</div>**Контейнер является тем самым элементом, к которому добавляется градиент. Кроме того, если все три столбца внутри контейнера являются плавающими, то для содержания всех этих плавающих элементов нужно будет воспользоваться  
одним из методов, рассмотренных ранее в разделе «Решение проблем плавающих элементов».

***Добавьте линейный градиент с цветовыми опорными точками, соответствующими ширине столбцов:***

**.wrapper {**

**background-image: linear-gradient (left, red 0%, red 25%, white 25%, white 75%, blue 75%, blue 75%, blue 100%);  
}**  
  
Недостаток использования линейного градиента заключается в том, что он работает только с чистыми цветами (и, разумеется, с градиентами, если вы того пожелаете), поэтому вы не можете применять в качестве фона графику или границы вокруг каждого столбца. Кроме того, Internet Explorer 9 и более ранние версии вообще не понимают, что такое градиенты.