**Введение в CSS**

Базовая структура каждой веб-страницы, **HTML**, очень проста сама по себе. Язык **CSS** расширяет возможности языка **HTML**. Красивые веб-сайты, которые вы видите в интернете, оформлены с помощью различных инструментов, включая **CSS**. Он позволяет менять цвета, шрифты, фон, в общем заниматься красотой сайта. А **HTML**, соответственно, отвечает за структуру сайта.

**1 уровень**

* 1. **Знакомство с CSS**

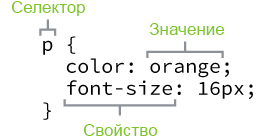
Каждому тегу в **HTML** соответствует так называемый **селектор** **CSS**. К примеру, тегу **<p>** соответствует **CSS** селектор **p**, с помощью которого мы можем обратиться ко всем абзацам HTML страницы и, например, покрасить их всех одновременно в оранжевый цвет и задать размер для шрифта как 16**px**. После селектора следует ставить фигурные скобки **{}**, внутри которых следует писать **CSS свойства**.

**CSS свойства** и задают цвет, размер шрифта и другие интересные вещи. Их следует писать в таком формате: **имя** свойства, потом **двоеточие**, потом **значение** этого свойства (например, свойство – это цвет, а "оранжевый" – это значение). Потом нужно поставить **точку с запятой** и можно писать следующее свойство.

**<p>**

Абзац

**</p>**



**Вывод**

Абзац

**Селектор-** Имя HTML элемента в начале набора правил. Он выбирает элемент для применения стиля.

**Свойство-** Способ, которым вы можете стилизовать определенный HTML элемент. Например изменение цвета или размера шрифта.

**Значение-** Выбирает одно из множества возможных значений для данного свойства

Давайте теперь разберемся с тем, как сделать так, чтобы написанный CSS применился к нашему **HTML коду**. Это можно сделать тремя способами.

**Первый (External)** способ заключается в том, что **CSS** будет написан в отдельном файле и специальным образом подключен к нашей HTML странице.

**Еще два способа(Internal и Inline)** заключаются в том, что **CSS** код будет написан прямо на HTML странице.

**Internal**

Internal способ заключается в том, что CSS код можно написать в теге **<style>** Его можно размещать как внутри **<head>** (чаще всего), так и внутри **<body>**.

**<!DOCTYPE html>**

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>Это заголовок тайтл </title>**

**<style>**

**p {**

**color: red;**

**}**

**</style>**

**</head>**

**<body>**

**<p>**

**Это абзац с текстом.**

**</p>**

**</body>**

**</html>**

**Inline**

Inbline способ заключается в том, что можно добавить атрибут **style** любому тегу на странице и прямо там написать для него HTML код. Обратите внимание на то, что при этом никаких селекторов писать не нужно, так как CSS код применится только к тому тегу, для которого он написа

**<!DOCTYPE html>**

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>Это заголовок </title>**

**</head>**

**<body>**

**<p style="color: red; font-weight: bold;">**

**Это абзац с текстом.**

**</p>**

**</body>**

**</html>**

**Первый способ** используется **гораздо чаще** (в подавляющем большинстве случаев). Одновременно можно применять все три способа.

**1.2 Знакомства с базовыми свойствами**

Свойство **color** позволяет задать **цвет текста**. Цвет можно задавать следующими способами: английским **словом**, через решетку **#**, через **rgb** и некоторыми другими способами, которые мы пока не будем разбирать.

Давайте разберем указанные способы подробнее

**Способ первый - английское слово**

Самый простой и понятный вариант задать цвет - указать его английским словом, например, **red** задает *красный* цвет, **blue** - *голубой*, **green** - *зеленый*, **black** - *черный*, **white** - *белый*

В следующем примере всем абзацам на странице задан красный цвет:

**<p>**

**Абзац с текстом.**

**</p>**

**p {**

**color: red;**

**}**

Абзац с текстом.

Первый способ задания цвета имеет некоторые недостатки. **Во-первых**, таким образом можно сделать не все оттенки цветов (все-таки количество названий ограничено, а оттенков можно придумать огромное множество). **Во-вторых**, невозможно помнить все названия цветов наизусть и, если нужен какой-то редкий цвет, приходится лазить по справочнику и искать его.

Поэтому в CSS существуют и более универсальные способы задать нужный вам цвет. Чтобы понять эти способы, для начала вам необходимо разобраться с тем, как получается нужный цвет на экране компьютера.

На самом деле отдельная точка экрана (*пиксель*) не может светиться всеми цветами, которые нужны, так как это технически было бы очень сложно. И на самом деле каждая точка экрана может светиться только тремя цветами: красным, зеленым и голубым. Причем одновременно и в разных пропорциях.

Комбинируя эти цвета мы можем получить любой нужный нам цвет подобно тому, как это делают художники с красками - если смешать два цвета - мы получим третий.

В CSS есть способ заставить пиксели светиться указанными тремя цветами в нужных нам пропорциях. Давайте его разберем:

### Способ второй - через rgb

Суть этого способа заключается в следующем - для свойства **color** я пишу следующее значение - *rgb(красный, зеленый, голубой)* - и указываю в каких пропорциях нужно брать эти три базовых цвета. Сами цвета могут изменяться от **0** до **255**. Причем ноль - это отсутствие цвета, а 255 - это чистый цвет (к примеру, чисто красный).

Сами буквы **rgb** расшифровываются как *red* (красный), *green* (зеленый), *blue* (голубой).

Давайте смешаем чисто красный и чисто зеленый цвет. Для этого первое значение ставим в **255**, второе тоже в **255**, а третье - в **ноль**, так как примесь голубого нам не нужна. У нас получится чисто желтый цвет (я поставил голубой фон блоку, так как желтый цвет на белом режет глаза):

<p>

Абзац с текстом.

</p>

p {

color: rgb(255, 255, 0);

}

Абзац с текстом.

### Способ третий - через #

Второй способ получения цвета (через rgb), не смотря на то, что позволяет получить любой оттенок любого цвета, все же является несколько громоздким. Поэтому существует и третий вариант задать цвет - через **шестнадцатеричное** значение.

Чтобы понять суть этого способа, вам нужно разобраться с **шестнадцатеричной** системой счисления. В ней, в отличии от десятичной, который мы с вами пользуемся не 10 цифр, а 16: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F** - как вы видите, за недостатком цифр после девяти **используются буквы**. Преимущество шестнадцатеричной системы в том, что число 255 (и менее) можно представить всего двумя символами - **FF** соответствует **255** и так далее.

В этой системе вместо **rgb()** записывается решетка **#**, после которой идут **6** знаков. Первые 2 знака - красный цвет, вторые два знака - зеленый, и последние два знака - голубой. Цвета также изменяются от **0** до **255**, но в шестнадцатеричной системе это будет от **00** до **FF**.

К примеру, запись **rgb(255, 200, 255)** можно представить как **#FFC8FF** - это будет более компактно.

В следующем примере абзац красится в красный цвет:

**<p>**

**Абзац с текстом.**

**</p>**

**p {**

**color: #FF0000;**

**}**

Абзац с текстом.

Свойства **width и height** позволяют задать высоту и ширину элементу соответственно. Ширина и высота обычно измеряются в **пикселях** (обозначается **px**) или **процентах** (обозначается **%**) (можно измерять и других единицах, которые мы пока не будем разбирать).

**Пиксель** - это минимальная точка на экране. При некотором опыте вы сможете легко определять на глаз, сколько пикселей размер у того или иного элемента. Для этого также можно использовать и специальные измерительные инструменты.

Если задавать размеры в **процентах** - то эти проценты вычисляются **относительно родительского элемента**.

В данном примере абзацу задана ширина и высота, а также **граница**, чтобы можно было увидеть, где заканчивается наш абзац (границу мы еще не проходили, пока не обращайте на нее внимание, чуть позже вы узнаете, как ее сделать):

<p>

Абзац с текстом.

</p>

p {

width: 300px;

height: 100px;

border: 1px solid black;

}



## **Свойство text-align - выравнивание текста**

Свойство **text-align** позволяет задать выравнивание текста. Текст можно выравнять по **левому** краю (значение **left**), по **правому** (значение **right**), по **центру** (значение **center**) и **одновременно** и по правому, и по левому краю (значение **justify**).

Давайте посмотрим на примерах, что имеется ввиду.

### Значение left

Давайте сделаем так, чтобы текст был выровнен по **левому** краю. Для этого нужно свойство **text-align** поставить в значение **left**:

<p>

Абзац с текстом.

</p>

p {

text-align: left;

width: 300px;

}

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean a dapibus magna, ac interdum nisl. Suspendisse eget fringilla nibh, eu commodo arcu. Donec lacinia tempor velit sed tincidunt. Aliquam porttitor nulla purus, vel vulputate ipsum faucibus sed. Phasellus sodales, lorem vel cursus vehicula, ante purus lacinia dui, interdum fringilla massa eros ut dui.

В принципе, для абзацев значение **left** можно и не ставить - они по умолчанию и так выровнены по **левому** краю. Однако есть элементы, которые по умолчанию стоят **по центру** (это, например, теги **th**, которые делают ячейку-заголовок таблицы). И вот для них как раз-таки и может потребоваться выравнивание по левой стороне.

### Значение right

Давайте теперь поставим текст по **правому** краю:

<p>

Абзац с текстом.

</p>

p {

text-align: right;

width: 300px;

}

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean a dapibus magna, ac interdum nisl. Suspendisse eget fringilla nibh, eu commodo arcu. Donec lacinia tempor velit sed tincidunt. Aliquam porttitor nulla purus, vel vulputate ipsum faucibus sed. Phasellus sodales, lorem vel cursus vehicula, ante purus lacinia dui, interdum fringilla massa eros ut dui.

### Значение center

Поставим текст **по центру**:

<p>

Абзац с текстом.

</p>

p {

text-align: center;

width: 300px;

}

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean a dapibus magna, ac interdum nisl. Suspendisse eget fringilla nibh, eu commodo arcu. Donec lacinia tempor velit sed tincidunt. Aliquam porttitor nulla purus, vel vulputate ipsum faucibus sed. Phasellus sodales, lorem vel cursus vehicula, ante purus lacinia dui, interdum fringilla massa eros ut dui.

### Значение justify

Ну, и наконец выровняем текст одновременно и по правому, и по левому краю:

<p>

Абзац с текстом.

</p>

p {

text-align: justify;

width: 300px;

}

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean a dapibus magna, ac interdum

nisl. Suspendisse eget fringilla nibh, eu commodo arcu. Donec lacinia tempor velit sed tincidunt. Aliquam porttitor nulla purus, vel vulputate ipsum faucibus sed. Phasellus sodales, lorem vel cursus vehicula, ante purus lacinia dui, interdum fringilla massa eros ut dui.

## **Свойство font-weight - жирность**

Свойство **font-weight** позволяет сделать текст жирным или наоборот - отменить жирность (к примеру, для заголовков, которые жирные по умолчанию). Чтобы сделать текст жирным, следует добавить значение **bold**, а чтобы отменить жирность - значение **normal**.

Это свойство работает аналогично [тегу b](http://code.mu/html/b.html), который мы с вами уже разобрали. Разница в том, что через CSS управлять жирностью гораздо удобнее - я могу заставить все абзацы стать жирными, а все заголовки - нежирными, сделав всего пару записей в CSS файле.

Если бы я делал жирный через **тег <b>** - мне пришлось бы содержимое каждого абзаца обернуть в этот тег, а это было бы очень затратным по времени (а если я потом захочу убрать жирность абзацам - мне придется убирать все эти теги, представьте, сколько это лишней, бесполезной работы).

В следующем примере мы сделаем все абзацы жирными, а все заголовки h3 - нежирными:

<h3>Это заголовок</h3>

<p>

Абзац с текстом.

</p>

h3 {

font-weight: normal;

}

p {

font-weight: bold;

}

### Это заголовок

**Абзац с текстом.**

Для сравнения привожу их вид по умолчанию:

### Это заголовок

Абзац с текстом.

## **Свойство font-style - курсив**

Свойство **font-style** позволяет сделать текст курсивным или наоборот - отменить курсив. Чтобы сделать текст курсивом, следует добавить значение **italic**, а чтобы отменить курсив - значение **normal**. Смотрите пример:

<p>

Абзац с текстом.

</p>

p {

font-style: italic;

}

*Абзац с текстом.*

1. **уровень**
   1. **Блочные и Строчные элементы**

В предыдущих уроках мы с вами использовали только один тип селекторов - имя тега, по которому мы обращались ко всем таким тегам. Однако, чаще всего нам нужны более точные способы обращения: к примеру, мы хотим выбрать не все абзацы, а только некоторые из них. Способы сделать это мы и будем изучать в данном уроке. Однако, для начала разберем два тега, которые пригодятся нам в дальнейшем.

## **Тег div**

Тег **div** служит контейнером для других тегов. Сам по себе он ничего не делает, однако в него можно положить много разных тегов, например, абзацев и заголовков, а затем для всех них **одновременно** применить различные CSS свойства. Для примера покрасим содержимое тега **div** в красный цвет (используя атрибут **style**):

<div style="color: red;">

<h2>

Lorem ipsum dolor sit amet.

</h2>

<p>

Lorem ipsum dolor sit amet.

</p>

<p>

Lorem ipsum dolor sit amet.

</p>

<p>

Lorem ipsum dolor sit amet.

</p>

</div>

## Lorem ipsum dolor sit amet.

Lorem ipsum dolor sit amet.

Lorem ipsum dolor sit amet.

Lorem ipsum dolor sit amet.

В настоящее время тег **div** является одним из основных строительных элементов на сайте. Вы еще столкнетесь с ним в дальнейшем много раз.

## **Тег span**

Тег **span** является контейнером для кусочка текста. Вы уже знаете, что для того, чтобы сделать, к примеру, жирный текст - следует использовать тег **b**. Однако, что делать, если я хочу покрасить *кусочек текста* в красный цвет? Для такого случая тега, подобного тегу **b**, не существует.

Для этого и предназначен тег **span** - сам по себе он абсолютно ничего не делает, но для него можно применить **CSS стили**, которые позволят нам добавить нужный эффект к кусочку текста.

Давайте покрасим некоторый текст в красный цвет:

<p>

Lorem ipsum <span style="color: red;">dolor</span> sit amet.

</p>

Lorem ipsum dolor sit amet.

Мы с вами изучили два необходимых нам тега и теперь приступаем к изучению более сложной работы с селекторами.

**2.1 Группируем Селекторы**

## **Группировка селекторов**

Иногда может возникнуть следующая ситуация: вам нужно сделать одно и то же с разными тегами, например, покрасить все заголовки **h2**, **h3** и **абзацы** в красный цвет. В этом случае вам придется написать что-то вроде такого:

h2 {

color: red;

}

h3 {

color: red;

}

p {

color: red;

}

Однако, существует способ сделать это немного короче: селекторы тегов можно объединять через **запятую** и CSS код применится к ним всем **одновременно**. Приведенный выше код можно переписать покороче таким образом:

h2, h3, p {

color: red;

}

<h2>Заголовок h2</h2>

<h3>Заголовок h3</h3>

<p>

Абзац с текстом.

</p>

**Заголовок h2**

**Заголовок h3**

Абзац с текстом

**2.2 Продолжаем Селекторы**

## **Вложенность тегов друг в друга**

Представим теперь ситуацию, когда мы хотим обратиться только к тем тегам **i**, которые находятся **внутри абзацев**, и не хотим трогать те теги **i**, которые находятся **внутри заголовков h2**.

Это делается с помощью так называемого *контекстного селектора*, который позволяет обращаться к тегам по их вложенности в другие теги.

Чтобы показать вложенность, между селекторами следует поставить **пробел** (этот пробел и есть знак контекстного селектора).

Например, так - **p i** - мы обратимся ко всем тегам **i**, находящимся внутри p, **p b** - так ко всем тегам **b** внутри **p**, а так - **div p b** - ко всем тегам **b**, которые находятся внутри абзацев p, которые в свою очередь находятся внутри тега div.

В следующем примере мы обратимся ко всем абзацам внутри div и покрасим их в красный цвет. Абзацы вне дива и другие элементы внутри дива (в нашем случае **h2**) красными не станут:

**div p {**

**color: red;**

**}**

**<div>**

**<h2>**

**Заголовок h2 внутри тега div.**

**</h2>**

**<p>**

**Абзац внутри тега div.**

**</p>**

**<p>**

**Абзац внутри тега div.**

**</p>**

**<p>**

**Абзац внутри тега div.**

**</p>**

**</div>**

**<p>**

**Абзац вне тега div.**

**</p>**

## Заголовок h2 внутри тега div.

Абзац внутри тега div.

Абзац внутри тега div.

Абзац внутри тега div.

Абзац вне тега div.

## **Выбор элемента по уникальному id**

Следующий способ может вам понадобится в такой ситуации: вам необходимо для всех абзацев поставить красный цвет, а **конкретно** для этого абзаца - зеленый. В этом случае, как вы уже знаете, можно воспользоваться атрибутом **style**, однако этот способ не самый оптимальный (он захламляет HTML код CSS кодом).

Второй способ заключается в том, что нужному элементу дается **атрибут id**, который содержит в себе **уникальное имя** нашего тега (*другого id с таким именем на странице быть не должно - будет конфликт*). Если мы дадим ему, к примеру, имя **test**, тогда в CSS мы сможем обратиться к нему таким образом: **#test** - то есть сначала пишется символ **"решетка"**, а затем - то **имя**, которое мы записали в атрибут **id**.

Давайте сейчас для всех абзацев зададим красный цвет, а для абзаца с **id="test"** - зеленый

**p {**

**color: red;**

**}**

**#test {**

**color: green;**

**}**

**<p id="test">**

**Абзац с атрибутом id в значении "test".**

**</p>**

**<p>**

**Обычный абзац с текстом.**

**</p>**

**<p>**

**Обычный абзац с текстом.**

**</p>**

Абзац с атрибутом id в значении "test".

Обычный абзац с текстом.

Обычный абзац с текстом.

А теперь атрибут **id** дадим **конкретному диву** и покрасим содержимое этого дива в красный цвет, обратившись к нему по его **id**:

#test {

color: red;

}

<div id="test">

<h2>

Заголовок h2 внутри #test.

</h2>

<h2>

Заголовок h2 внутри #test.

</h2>

<h2>

Заголовок h2 внутри #test.

</h2>

<p>

Абзац внутри #test.

</p>

<p>

Абзац внутри #test.

</p>

<p>

Абзац внутри #test.

</p>

</div>

<p>

Абзац вне #test.

</p>

## Заголовок h2 внутри #test.

## Заголовок h2 внутри #test.

## Заголовок h2 внутри #test.

Абзац внутри #test.

Абзац внутри #test.

Абзац внутри #test.

Абзац вне #test.

А теперь совместим *контекстный селектор* (который пробел, показывающий вложенность) и **id** и сделаем так, чтобы **только абзацы** (но не заголовки **h2** в данном случае) из **#test** стали красного цвета:

#test p {

color: red;

}

<div id="test">

<h2>

Заголовок h2 внутри #test.

</h2>

<h2>

Заголовок h2 внутри #test.

</h2>

<h2>

Заголовок h2 внутри #test.

</h2>

<p>

Абзац внутри тега div.

</p>

<p>

Абзац внутри тега div.

</p>

<p>

Абзац внутри тега div.

</p>

</div>

<p>

Абзац вне тега div.

</p>

## Заголовок h2 внутри #test.

## Заголовок h2 внутри #test.

## Заголовок h2 внутри #test.

Абзац внутри #test.

Абзац внутри #test.

Абзац внутри #test.

Абзац вне #test.

## **Классы элементов**

Обращение к элементу по его **id** имеет некоторый недостаток - так мы можем обратиться только к одному элементу на странице. А что делать, если нам, к примеру, нужны абзацы двух типов - красные и зеленые и они могут чередоваться в произвольном порядке? В этом случае гораздо удобнее будет воспользоваться *классами*.

**Классы** задаются с помощью **атрибута class**, в котором мы пишем имя класса. Их преимущество в том, что один и тот же класс может быть у **многих** элементов на странице. То есть, если элементы должны вести себя одинаково - мы даем им один и тот же класс.

Как обратиться к элементам с определенным классом в **CSS**: пусть мы задали нужным нам элементам класс с именем **test**, тогда в **CSS** ко всем элементам с этим классом мы можем обратиться следующим образом - **.test** - вначале символ **"точка"** и потом **имя класса** из атрибута **class**.

В примере ниже и абзацы, и заголовки имеют один и тот же класс **test**, который красит эти элементы в красный цвет:

.test {

color: red;

}

<h2 class="test">

Заголовок h2 с классом test.

</h2>

<h2 class="test">

Заголовок h2 с классом test.

</h2>

<h2 class="test">

Заголовок h2 с классом test.

</h2>

<p class="test">

Абзац с классом test.

</p>

<p class="test">

Абзац с классом test.

</p>

<p class="test">

Абзац с классом test.

</p>

<p>

Абзац без класса.

</p>

## Заголовок h2 с классом test.

## Заголовок h2 с классом test.

## Заголовок h2 с классом test.

Абзац с классом test.

Абзац с классом test.

Абзац с классом test.

Абзац без класса.

## **Определенный тег с заданным классом**

Бывают такие ситуации, когда разные теги имеют один и тот же класс. К примеру, у абзацев и заголовков h2 одновременно задан класс **test**. Вам может потребоваться выбрать **только абзацы** с классом **test**, не затрагивая заголовков **h2** с этим же классом.

В этом случае вместо селектора **.test** я должен написать следующее: **p.test** - таким образом я выберу все абзацы с классом **test**, не затронув заголовки. Если же я напишу **h2.test** - то выберу все заголовки h2 с классом **test**, не затронув абзацев.

В следующем примере только абзацы с классом **test** станут красного цвета, а заголовки с таким же классом - не станут:

p.test {

color: red;

}

<h2 class="test">

Заголовок h2 с классом test.

</h2>

<h2 class="test">

Заголовок h2 с классом test.

</h2>

<h2 class="test">

Заголовок h2 с классом test.

</h2>

<p class="test">

Абзац с классом test.

</p>

<p class="test">

Абзац с классом test.

</p>

<p class="test">

Абзац с классом test.

</p>

<p>

Абзац без класса.

</p>

## Заголовок h2 с классом test.

## Заголовок h2 с классом test.

## Заголовок h2 с классом test.

Абзац с классом test.

Абзац с классом test.

Абзац с классом test.

Абзац без класса.

Итак, еще раз: **p.test** - такой селектор выбирает *абзацы с классом test*. А вот если я сделаю так: **p .test** (**поставлю пробел**), то выберу все элементы с классом **test**, находящиеся *внутри* абзацев. Прочувствуйте эту разницу.

Давайте теперь выберем **только h2** с классом **test**:

h2.test {

color: red;

}

<h2 class="test">

Заголовок h2 с классом test.

</h2>

<h2 class="test">

Заголовок h2 с классом test.

</h2>

<h2 class="test">

Заголовок h2 с классом test.

</h2>

<p class="test">

Абзац с классом test.

</p>

<p class="test">

Абзац с классом test.

</p>

<p class="test">

Абзац с классом test.

</p>

<p>

Абзац без класса.

</p>

## Заголовок h2 с классом test.

## Заголовок h2 с классом test.

## Заголовок h2 с классом test.

Абзац с классом test.

Абзац с классом test.

Абзац с классом test.

Абзац без класса.

**2.3 Трансформация Текста**

Свойство **text-decoration** позволяет задавать некоторые эффекты для текста: подчеркивание, ~~перечеркивание~~, линию сверху, а также отменять такие эффекты, если какой-либо тег имеет их по умолчанию. Чаще всего это свойство используются для отмены подчеркивания ссылок (они по умолчанию подчеркнуты). Давайте посмотрим, какие значения может принимать свойство **text-decoration**.

**Значение underline**

Значение **underline** добавляет подчеркивание тексту:

p {

text-decoration: underline;

}

<p>

Lorem ipsum dolor sit amet.

</p>

Lorem ipsum dolor sit amet.

**Значение overline**

Значение **overline** добавляет линию над текстом:

p {

text-decoration: overline;

}

<p>

Lorem ipsum dolor sit amet.

</p>

Lorem ipsum dolor sit amet.

**Значение line-through**

Значение **line-through** добавляет линию, перечеркивающую текст:

p {

text-decoration: line-through;

}

<p>

Lorem ipsum dolor sit amet.

</p>

~~Lorem ipsum dolor sit amet.~~

**Значение none**

Значение **none** отменяет все эффекты, обычно используется для отмены подчеркивания ссылок.

В следующем примере ссылка по умолчанию будет подчеркнута, а вторая ссылка с **id="link"** будет без подчеркивания, так как мы ей зададим **text-decoration** в значении **none**:

#link {

text-decoration: none;

}

<a href="#">Ссылка по умолчанию</a>

<a id="link" href="#">Ссылка без подчеркивания</a>

[Ссылка по умолчанию](http://code.mu/books/css/osnovy-raboty-so-ssylkami-i-granicami-cherez-css-dlya-novichkov.html) [Ссылка без подчеркивания](http://code.mu/books/css/osnovy-raboty-so-ssylkami-i-granicami-cherez-css-dlya-novichkov.html)

1. **уровень**
   1. **Псевдоклассы и псвдоэлементы**

Есть несколько элементов, которых нет в HTML, но они есть на странице (к примеру первая буква слова) их называют псевдоэлементами. Псевдоэлементам тоже можно задавать стиль.

**Псевдокласс** в CSS — это ключевое слово, добавленное к селектору, которое определяет его особое состояние. Например, [:hover](https://developer.mozilla.org/ru/docs/Web/CSS/:hover) может быть использован для изменения цвета кнопки при наведении курсора на неё.

Синтаксис:

селектор:псевдоклсс {

свойство: значение;

}

**Список Псевдоклассов:**

|  |  |  |
| --- | --- | --- |
| **Селектор** | **Пример** | **Объяснение** |
| [**:active**](https://www.w3schools.com/cssref/sel_active.asp) | a:active | Выбирает активную ссылку |
| [**:checked**](https://www.w3schools.com/cssref/sel_checked.asp) | input:checked | Выбор каждого проверяемого элемента <input> |
| [**:disabled**](https://www.w3schools.com/cssref/sel_disabled.asp) | input:disabled | Выбирает каждый не рабочий элемента <input> |
| [**:empty**](https://www.w3schools.com/cssref/sel_empty.asp) | p:empty | Выбирает каждый <p> элемент, который не имеет потомков(ребенком) |
| [**:enabled**](https://www.w3schools.com/cssref/sel_enabled.asp) | input:enabled | Выбирает каждый включенный <input> элемент |
| [**:first-child**](https://www.w3schools.com/cssref/sel_firstchild.asp) | p:first-child | Выбирает каждый <p> элементы, что является первым потомком(ребенком) своего родителя |
| [**:first-of-type**](https://www.w3schools.com/cssref/sel_first-of-type.asp) | p:first-of-type | Выбирает каждый <p> элемент, который является первым <p> элементом своего родителя |
| [**:focus**](https://www.w3schools.com/cssref/sel_focus.asp) | input:focus | Выбирает <input> элемент, который имеет фокус |
| [**:hover**](https://www.w3schools.com/cssref/sel_hover.asp) | a:hover | Выбирает ссылки на курсор |
| [**:in-range**](https://www.w3schools.com/cssref/sel_in-range.asp) | input:in-range | Выбирает <input> элементы со значением в заданном диапазоне |
| [**:invalid**](https://www.w3schools.com/cssref/sel_invalid.asp) | input:invalid | Выбирает все <input> элементов с недопустимым значением |
| [**:lang(*language*)**](https://www.w3schools.com/cssref/sel_lang.asp) | p:lang(it) | Выбирает каждый <p> элемент со значением атрибута lang, начиная с "it" |
| [**:last-child**](https://www.w3schools.com/cssref/sel_last-child.asp) | p:last-child | Выбирает каждый <p> элементы, которые является последним потомком (ребенком) своего родителя |
| [**:last-of-type**](https://www.w3schools.com/cssref/sel_last-of-type.asp) | p:last-of-type | Выбирает каждый <p> элемент, который является последним <p> элементом своего родителя |
| [**:link**](https://www.w3schools.com/cssref/sel_link.asp) | a:link | Выбирает все непосещенные ссылки |
| [**:not(selector)**](https://www.w3schools.com/cssref/sel_not.asp) | :not(p) | Выбирает каждый элемент, не являющийся <p> элемент |
| [**:nth-child(n)**](https://www.w3schools.com/cssref/sel_nth-child.asp) | p:nth-child(2) | Выбирает каждый <P> элемент, который является вторым потомком (ребенком) своего родителя |
| [**:nth-last-child(n)**](https://www.w3schools.com/cssref/sel_nth-last-child.asp) | p:nth-last-child(2) | Выбирает каждый <p> элемент, который является вторым потомком (ребенком) своего родителя, считая от последнего потомка (ребенка) |
| [**:nth-last-of-type(n)**](https://www.w3schools.com/cssref/sel_nth-last-of-type.asp) | p:nth-last-of-type(2) | Выбирает каждый <p> элемент, который является вторым <p> элемент своего родителя, считая от последнего потомка (ребенка) |
| [**:nth-of-type(n)**](https://www.w3schools.com/cssref/sel_nth-of-type.asp) | p:nth-of-type(2) | Выбирает каждый <p> элемент, который является вторым <p> элементом своего родителя |
| [**:only-of-type**](https://www.w3schools.com/cssref/sel_only-of-type.asp) | p:only-of-type | Выбирает каждый <p> элемент, который является единственным <p> элемент своего родителя |
| [**:only-child**](https://www.w3schools.com/cssref/sel_only-child.asp) | p:only-child | Выбирает каждый <p> элемент, который является единственным потомком (ребенком) своего родителя |
| [**:optional**](https://www.w3schools.com/cssref/sel_optional.asp) | input:optional | Выбирает <input> элементы с "обязательным атрибутом"(required) |
| [**:out-of-range**](https://www.w3schools.com/cssref/sel_out-of-range.asp) | input:out-of-range | Выбирает <input> элементы со значением, выходящим за пределы указанного диапазона |
| [**:read-only**](https://www.w3schools.com/cssref/sel_read-only.asp) | input:read-only | Выбирает <input> указанный элементы с атрибутом "только для чтения"(read-only) |
| [**:read-write**](https://www.w3schools.com/cssref/sel_read-write.asp) | input:read-write | Выбирает <input> элементы с атрибутом "только чтение" |
| [**:required**](https://www.w3schools.com/cssref/sel_required.asp) | input:required | Выбирает < input > указанные элементы с "обязательным атрибутом" |
| [**:root**](https://www.w3schools.com/cssref/sel_root.asp) | root | Выбор корневого элемента документа |
| [**:target**](https://www.w3schools.com/cssref/sel_target.asp) | #news:target | Выбирает текущий активный элемент #news (нажали на URL-адрес, содержащий имя якоря ) |
| [**:valid**](https://www.w3schools.com/cssref/sel_valid.asp) | input:valid | Выбирает все <input> элементы с допустимым значением |
| [**:visited**](https://www.w3schools.com/cssref/sel_visited.asp) | a:visited | Выбирает все посещенные ссылки |

**Псевдоклассы**

В CSS всего четыре псевдо-класса, псевдо-классы позволяют работать с ссылками. У ссылок бывает четыре состояния:

простая

активная

посещенная

на которую наведен курсор

Чтоб описать стиль для этих ссылок существуют псевдоклассы:

a:link - стиль простой ссылки

a:active - стиль активной ссылки

a:visited - стиль посещенной ссылки

a:hover - стиль ссылки, на которую наведен курсор

По умолчанию ссылки всегда синего цвета и подчеркнуты. Давайте потренируемся:

**<html>  
  <head>  
    <title>Заголовок документа</title>  
    <link rel="stylesheet" type="text/css" href="/style.css">  
  </head>  
  <body>  
    <a href="#">ссылка</a>  
  </body>  
</html>**

**Ссылка по умолчанию выглядит так:**

**стандартная ссылка**

Изменим цвет ссылке на зеленый и уберем подчеркивание:

**a{  
  color:green;  
  text-decoration:none;  
}**

**text-decoration - отвечает за подчеркивание, а значение none говорит, что подчеркивать не надо.**

**изменим цвет ссылке и уберем подчеркивание**

Пойдем дальше и изменим стиль ссылки, на которую навели курсор. Пусть она изменит цвет на красный:

**a{  
  color:green;  
  text-decoration:none;  
}  
a:hover{  
  color:red;  
}**

Проверяем нашу страничку и видим, что при наведении мышкой ссылка меняет цвет с зеленого на красный.

**Псевдоэлемент** в CSS — это ключевое слово, добавляемое к селектору, которое позволяет стилизовать определённую часть выбранного элемента. Например, псевдоэлемент   
[::first-line](https://developer.mozilla.org/ru/docs/Web/CSS/::first-line) может быть использован для изменения шрифта первой строки абзаца

селектор::псевдоэлемент {

свойство: значение;

}

**Список Псевдоэлементов:**

|  |  |  |
| --- | --- | --- |
| **Селектор** | **Пример** | **Объяснение** |
| [**::after**](https://www.w3schools.com/cssref/sel_after.asp) | p::after | Вставка содержимого после каждого <p> элемента |
| [**::before**](https://www.w3schools.com/cssref/sel_before.asp) | p::before | Вставка содержимого перед каждым <p> элементом |
| [**::first-letter**](https://www.w3schools.com/cssref/sel_firstletter.asp) | p::first-letter | Выбирает первую букву каждого <p> элемента |
| [**::first-line**](https://www.w3schools.com/cssref/sel_firstline.asp) | p::first-line | Выбирает первую строку каждого <p> элемента |
| [**::selection**](https://www.w3schools.com/cssref/sel_selection.asp) | p::selection | Выбирает часть элемента, выбранного пользователем |

**К псевдоэлементам относят:**

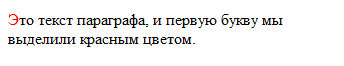
**first-letter - стиль для первой буквы слова. Пример:**

**<html>  
  <head>  
    <title>Заголовок документа</title>  
    <link rel="stylesheet" type="text/css" href="/style.css">  
  </head>  
  <body>  
    <p>  
    Это текст параграфа, и первую букву мы выделили красным цветом.  
    </p>  
  </body>  
</html>**

**Перекрасим первую букву параграфа в красный цвет, пишем:**

**p:first-letter{  
  color:red;  
}**

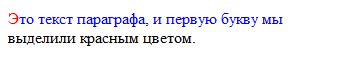
**Получаем:**

****

**first-line - стиль для первой строки абзаца. Перекрасим первую строку абзаца в синий цвет. Пишем:**

**p:first-letter{  
  color:red;  
}  
p:first-line{  
  color:blue;  
}**

**Получаем:**

****

* 1. **Блоки**

**Страница сверстанная при помощи блочной верстке представляет собой множество блоков. Каждый элемент в дереве элементов это самостоятельный блок.**

**Блок выглядит в виде прямоугольника:**

****

**У блока может быть содержимое, к примеру у элемента p - это текст. Вокруг содержимого можно прописать отступы (*padding*), они необходимы для внутреннего отступа текста от  границ блока.**

**Блок всегда обрамляет его граница *border (граница блока)*, она может быть видимой и скрытой.**

**Еще у блока можно прописать поля (*margin*), они задают дополнительное пространство между блоками.**

**Размер блока определяется его содержимым или параметрами *width*(ширина) и *height*(высота).**

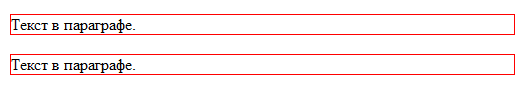
**Сейчас, чтоб стало более менее понятно давайте приведем простой пример:**

**<html>  
  <head>  
    <title>Заголовок документа</title>  
    <link rel="stylesheet" type="text/css" href="/style.css">  
  </head>  
  <body>  
    <p>Текст в параграфе.</p>  
    <p>Текст в параграфе.</p>  
  </body>  
</html>**

**Сейчас, чтобы увидеть отступы, поля и границы, зададим *border* (границу):**

**p{  
  border:1px solid red;  
}**

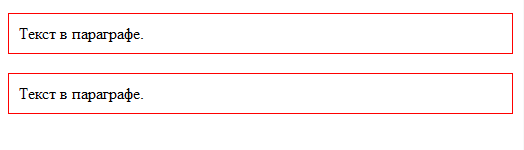
**Результат:**

****

**Давайте зададим тексту внутренний отступ от границы блока (*padding*):**

**p{  
  border:1px solid red;  
  padding:10px;  
}**

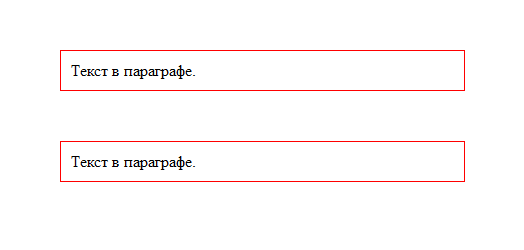
**Результат:**

****

**Мы задали отступы внутри блока, теперь давайте зададим дополнительно пространство между блоками (наружный отступ). Поможет нам в этом *margin*:**

**p{  
  border:1px solid red;  
  padding:10px;  
  margin:50px;  
}**

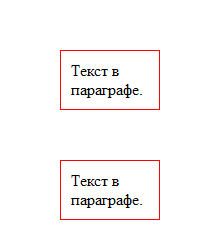
**Результат:**

****

**Ну и давайте теперь зададим размеры блоков (*width*(ширина) и *height*(высота)):**

**p{  
  border:1px solid red;  
  padding:10px;  
  margin:50px;  
  width:100px;  
  height:50px;  
}**

**Результат:**

****

**Думаю Вам уже стало понятно логика блочной верстки сайта. Переходим в следующий урок и более близко знакомимся с margin, padding, border!**

* 1. **Margin, padding, border**

**Давайте поучимся задавать свойства блоков. Тренироваться мы будем на элементах div и span.**

**Div - это элемент контейнером для остальных. Элемент span создает строчный блок.**

**Давайте рассмотрим пример:**

**<html>**

**<head>**

**<title>Заголовок документа</title>**

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

**</head>**

**<body>**

**<div id="first">Это текст в тегах div с id="first".**

**</div>**

**<div id="second">**

**<div id="third">Это текст в тегах div с id="third".</div>**

**<div id="fourth">Это текст в тегах div с id="fourth".</div>**

**</div>**

**</body>**

**</html>**

**Давайте посмотрим на страницу в браузере:**

**css div и span**

**Обратите внимания что элементы div располагаются один под одним.**

**Border (граница)**

**Границы в css можно задавать отдельно для каждой стороны:**

**border-top - верхняя граница**

**border-right - правая граница**

**border-bottom - нижняя граница**

**border-left - левая граница**

Каждая сторона границы может иметь свои характеристики. Для необходимо через дефис дописать ключевые слова: border-top-color или border-left-style.

Если Вы не собираетесь выделять конкретную сторону границы, то тогда можно воспользоваться сокращенной записью:

**border-color - цвет всех границ (именные цвета, шестнадцатеричные коды цветов и десятичные коды в модели RGB)**

**border-width - толщина всех границ (ключевые слова: thin (тонкая граница), medium (средняя граница) и thick (толстая граница) или любая единица измерения.)**

**border-style - стиль всех границ (ключевые слова)**

**Ключевые слова style:**

**none - граница отсутствует**

**dotted - граница состоит из точек**

**dashed - граница в виде пунктирной линии**

**solid - граница отображается сплошной линией**

**double - граница отображается двойной сплошной линией**

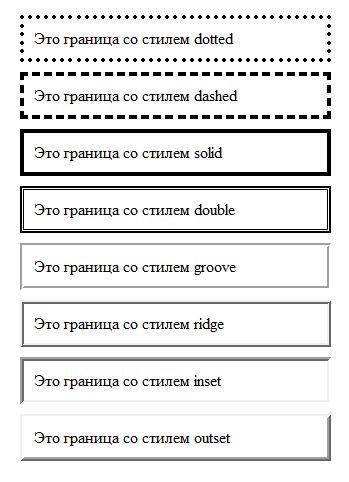
**groove - граница отображается вдавленной объемной линией**

**ridge - граница отображается выпуклой объемной линией**

**inset - граница отображается так, что весь блок выглядит вдавленным**

**outset - граница отображается так, что весь блок выглядит выпуклым**

Вот так выглядят разные стили границ в браузере:



**css Border свойства**

Сейчас давайте сами посмотрим как это работает, зададим нашему первому div-у разные границы:

**#first{**

**border-bottom-style:double;**

**border-bottom-color:red;**

**border-left-style:solid;**

**border-left-width:2px;**

**border-left-color:blue;**

**border-right-style:solid;**

**border-right-width:2px;**

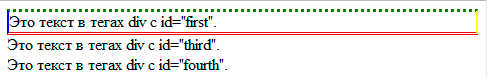
**border-right-color:yellow;**

**border-top-style:dotted;**

**border-top-color:green;**

**}**

**Результат:**

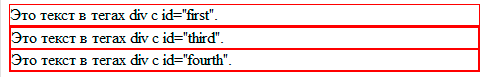


Парой возможность изменять стили каждой стороны границы очень помогают, но по факту практически все и всегда задают единый стиль. Бонусом появляется возможность пользоваться сокращенной записью border (через пробел указываем толщину, тип и цвет (только в таком порядке)):

**#first, #second, #third, #fourth{**

**border: 1px solid red;**

**}**



Сейчас границы накладываются друг на друга, так как мы не задали внешний отступ (margin):

**Margin (поля)**

поля создают свободное пространство вокруг блока (элемента). Поля в css, аналогично границам можно определять отдельно:

**margin-top - ширина верхнего поля**

**margin-right - ширина правого поля**

**margin-bottom - ширина нижнего поля**

**margin-left - ширина левого поля**

Можно воспользоваться сокращенной записью:

**p{**

**margin:5px 10px 15px 10px; /\*ширина верхнего, правого, нижнего и левого полей\*/**

**}**

**p{**

**margin:5px 10px; /\*ширина верхнего и нижнего полей совпадают, и ширина правого и левого полей совпадают\*/**

**}**

**p{**

**margin:5px; /\*ширина всех полей одинакова\*/**

**}**

Ширину полей можно задавать не только пикселях (px), но и в % или других единицах длины. Margin может иметь отрицательное значение, парой это очень удобно. Давайте зададим всем элементам одинаковый отступ - в 10px:

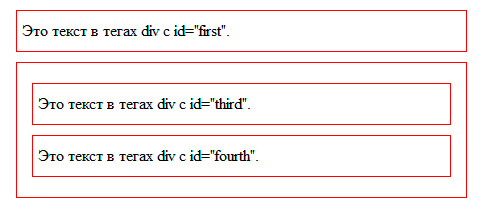
**#first, #second, #third, #fourth{**

**border: 1px solid red;**

**margin:10px;**

**}**

**Результат:**



Мы видим, что между блоками появился внешний отступ, но текст внутри блоков прижат к границам. Давайте зададим внутренний отступ (Padding).

**Padding (отступы)**

Padding позволяет задать внутренний отступ (отделить содержимое блока от border). Padding можно задать для каждой стороны отдельно:

**padding-top - ширина верхнего отступа**

**padding-right - ширина правого отступа**

**padding-bottom - ширина нижнего отступа**

**padding-left - ширина левого отступа**

Задавать Padding можно в разных единицах длины или в процентах. Проценты вычисляются относительно ширины блока. Задавать можно только положительные значения.

Можно использовать сокращенную запись (аналогична записи для margin ):

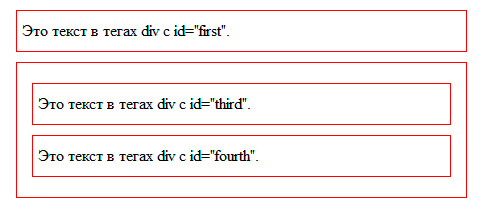
**#first, #second, #third, #fourth{**

**border: 1px solid red;**

**margin:10px;**

**padding:10px 5px;**

**}**



Давайте зададим фон нашим элементам margin и padding:

**#first, #second, #third, #fourth{**

**border: 1px solid red;**

**margin:10px;**

**padding:10px 5px;**

**}**

**#first, #second{**

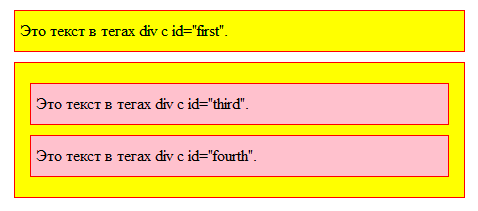
**background:yellow;**

**}**

**#third, #fourth{**

**background:pink;**

**}**



Мы видим, что фон padding совпадает с фоном блока, а фон margin - прозрачный. Теперь давайте зададим ширину и высоту нашим блокам:

**#first, #second, #third, #fourth{**

**border: 1px solid red;**

**margin:10px;**

**padding:10px 5px;**

**}**

**#first{**

**background:yellow;**

**width:300px;**

**height:100px;**

**}**

**#second{**

**background:yellow;**

**width:300px;**

**}**

**#third, #fourth{**

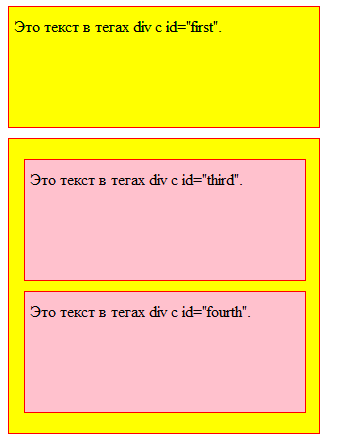
**background:pink;**

**width:270px;**

**height:100px;**

**}**

**Смотрим:**



Вы узнали все способы задания полей, отступов и границ. Для закрепления рекомендуем Вам потренироваться, а если уверены в своих силах то переходите на следующий урок.

* 1. **Позиционирование**

Свойство **position** задает способ позиционирования элементов. Свойство **position** чаще всего используется совместно со свойствами **top**, **right**, **bottom**, **left**, которые устанавливают отступы сверху, справа, снизу, слева соответственно.

## Синтаксис

селектор {

position: absolute | relative | fixed | static | sticky | inherit;

}

## Значения для position

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| **absolute** | Абсолютное позиционирование. |
| **relative** | Относительное позиционирование. |
| **fixed** | Фиксированное позиционирование. |
| **static** | Статичное позиционирование. |
| **sticky** | Стикер (CSS3). |
| **inherit** | Наследует значение родителя. |

## **позиция: статическая;**

Элементы HTML по умолчанию позиционируют статический.

На статические позиционные элементы не влияют верхние, нижние, левые и правые свойства.

Элемент с position: static; не позиционируется каким-либо особым образом; он всегда позиционируется в соответствии с нормальным потоком страницы:

## **позиция: относительная;**

Элемент с position: relative; располагается относительно его нормального положения.

Установка верхнего, правого, нижнего и левого свойств относительно расположенного элемента приведет к Он должен быть отрегулирован от своего нормального положения. Другое содержимое не будет скорректировано, чтобы уместиться в зазор, оставленный элементом.

## **позиция: фиксированная;**

Элемент с position: fixed; располагается относительно видового экрана, что означает, что он всегда остается в том же месте, даже если страница прокручивается. Свойства сверху, справа, снизу и слева используются для размещения элемента.

Фиксированный элемент не оставляет зазора на странице, где он обычно находился.

## **позиция: абсолютная;**

Элемент с position: absolute; располагается относительно ближайшего расположенного предка (вместо расположения относительно видового экрана, например "фиксированный").

Однако Если абсолютный позиционный элемент не имеет позиционных предков, он использует тело документа и перемещается вместе с прокруткой страницы.

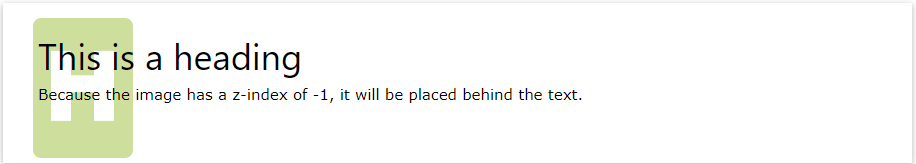
**Примечание:** "позиционируется" элемент, позиция которого является чем-либо, кроме static .

## **Перекрывающиеся элементы**

При позиционировании элементов они могут перекрывать другие элементы.

Свойство z-index указывает порядок стека элемента (какой элемент должен быть помещен перед, или позади, другие).

Элемент может иметь положительный или отрицательный порядок стека:



**Все свойства позиционирования CSS**

|  |  |
| --- | --- |
| **Свойство** | **Описание** |
| [**bottom**](https://html5css.ru/cssref/pr_pos_bottom.php) | **Задание нижнего края поля для позиционного прямоугольника** |
| [**clip**](https://html5css.ru/cssref/pr_pos_clip.php) | **Зажимы абсолютно позиционированного элемента** |
| [**left**](https://html5css.ru/cssref/pr_pos_left.php) | **Задание левого края поля для позиционного прямоугольника** |
| [**position**](https://html5css.ru/cssref/pr_class_position.php) | **Определяет тип позиционирования для элемента** |
| [**right**](https://html5css.ru/cssref/pr_pos_right.php) | **Устанавливает правый край поля для позиционного прямоугольника** |
| [**top**](https://html5css.ru/cssref/pr_pos_top.php) | **Задание верхнего края поля для позиционного прямоугольника** |
| [**z-index**](https://html5css.ru/cssref/pr_pos_z-index.php) | **Задает порядок стека элемента** |

**3.5**

**3.6**