

# БЁРСТКА #1

**LoftSchool**  
от мыслителя к создателю

# Содержание

<b>1. Блочные и строчные элементы в CSS</b>	<b>3-11</b>
• Управление размерами контейнеров	7
• Управление размещением контейнеров	8
<b>2. Позиционирование, поток документа</b>	<b>12-18</b>
• Поток документа	13
• Виды позиционирования	15
• Применение позиционирования в работе	17
• Абсолютное позиционирование внутри другого элемента	17
<b>3. Поток документа, clearfix и зачем это нужно</b>	<b>19-20</b>
<b>4. Свойство z-index</b>	<b>21-23</b>
<b>5. Пользовательские шрифты</b>	<b>24-28</b>
<b>6. Отступы между инлайн-элементами</b>	<b>29-33</b>

# 1

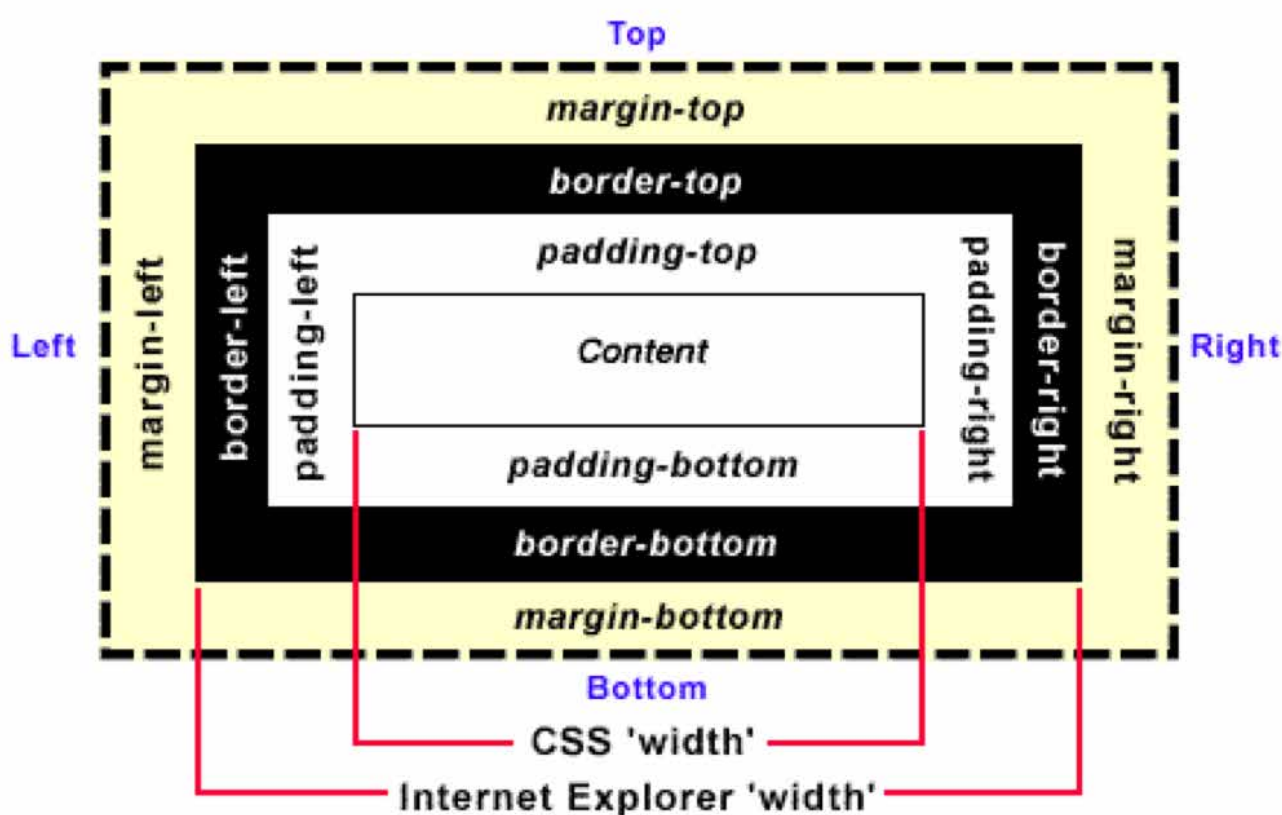
## **Блочные и строчные элементы в CSS**

**Блок** — это обычная прямоугольная область, обладающая рядом свойств, таких как: рамка, поля и отступы. Содержимым блока может быть что угодно: текст, картинка, видеоролик, список, форма для заполнения, меню навигации и т.п.

**Рамка (border)** — это контур, для которого можно задать такие характеристики как: толщина, цвет и тип (пунктирная, сплошная, точечная). В отличие от таблиц вещь необязательная.

**Поля (padding)** — отделяют содержимое блока от его рамки, чтобы текст, например, не прижимался к стенкам блока.

**Отступы (margin)** — это пустое пространство между различными блоками, либо между блоком и стенками страницы сайта. Они позволяют расположить блоки на заданном расстоянии относительно друг друга.



## Теги **div** и **span**

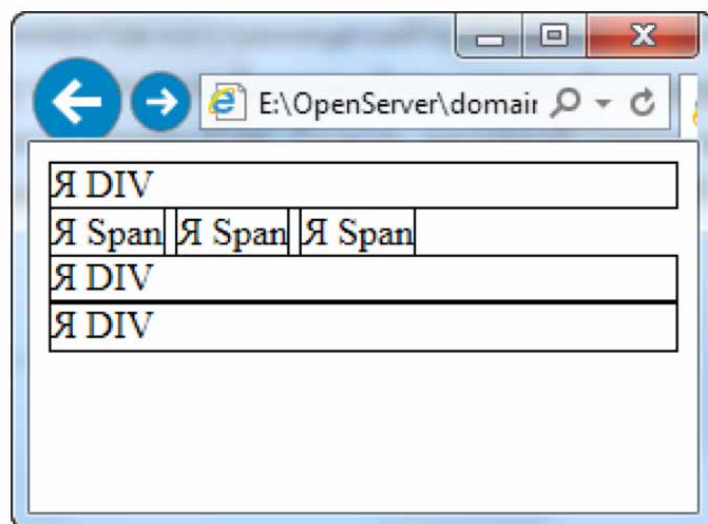
В отличие от стандартных HTML-тегов, к своему содержимому (**p** — к абзацам, **a** — к ссылкам, **img** — к изображениям), тег **div** является, по-сути, нейтральным. То есть ему всё равно, что содержать в себе, хоть всё разом. Обычно, тег **div** используют для задания больших функциональных областей на странице, таких как: шапка (в HTML5 это тег **header**), блок навигации (в HTML5 это тег **nav**), блок(и) основного содержимого, футер (в HTML5 это тег **footer**) или подвал и т.п. Но он может применяться и к более мелким деталям сайта. И он также является парным: **<div></div>**.

Чтобы различать однотипные теги им были придуманы специальные атрибуты. Наиболее часто используют два вида: идентификаторы (**id**) и классы (**class**).

**id** — атрибут, позволяющий придать тегу уникальный набор свойств, то есть такой, который на странице сайта используется только один раз.

**class** — атрибут, который позволяет один и тот же набор свойств задать нескольким элементам на странице сайта.

Тег **span**, так же как и **div**, является нейтральным. Он может применяться к любому элементу или группе элементов на странице сайта. Главное отличие его от дива — характер размещения на странице. Если **div** — это чисто блоковый тег, который, по-умолчанию, не терпит соседства с другим блоком, то **span** — это строковый тег. То есть, на одной строке может размещаться подряд несколько "спанов", тогда как "дивы" стремятся залезть один под другой.



**Блочные элементы** характеризуются тем, что занимают всю доступную ширину, высота элемента определяется его содержимым, и он всегда начинается с новой строки.

**Строчными** называются такие элементы веб-страницы, которые являются непосредственной частью другого элемента, например, текстового абзаца. В основном, они используются для изменения вида текста или его логического выделения.

Строчные элементы могут содержать только данные или другие строчные элементы, а в блочные допустимо вкладывать другие блочные элементы, строчные элементы, а также данные. Иными словами, строчные элементы никак не могут хранить блочные элементы (в HTML5 в строчный элемент ссылки `<a>` можно вкладывать блочные элементы).

Блочные элементы всегда начинаются с новой строки, а строчные таким способом не акцентируются.

Блочные элементы занимают всю доступную ширину, например, окна браузера, а ширина строчных элементов равна их содержимому плюс значения отступов, полей и границ.

# Управление размерами контейнеров

**<div>-контейнер** представляет собой прямоугольную область. Значения высоты и ширины данной области определяются такими стандартными атрибутами стилей, как:

## высота

<b>min-height</b>	задает минимальную высоту элемента
<b>height</b>	задает высоту элемента
<b>max-height</b>	задает максимально возможную высоту элемента

## ширина

<b>min-width</b>	задает минимальную ширину элемента
<b>width</b>	задает ширину элемента
<b>max-width</b>	задает максимально возможную ширину элемента

Также допустимо использование следующих значений атрибутов управления размерами:

- **auto** – размер устанавливается в зависимости от размеров контента;
- **inherit** – значение наследуется от родителя.

# Управление размещением контейнеров

Для управления размещением элементов используется атрибут стиля **float**, принимающий следующие значения:

<b>left</b>	элемент выравнивается по левому краю родителя, остальные элементы "обтекают" указанный по правой стороне
<b>right</b>	элемент выравнивается по правому краю родителя, остальные элементы "обтекают" указанный по левой стороне
<b>none</b>	обтекание элемента не указано
<b>inherit</b>	значение наследуется от родителя

Для того, чтобы элементы расположились по горизонтали один за другим, необходимо задать одно и то же значение **float** для следующих друг за другом элементов.

Для размещения блоков ниже выровненных по горизонтали, необходимо использовать атрибут стиля **clear**, явно указывающий на то, что данный блок должен располагаться ниже предшествующих ему контейнеров.

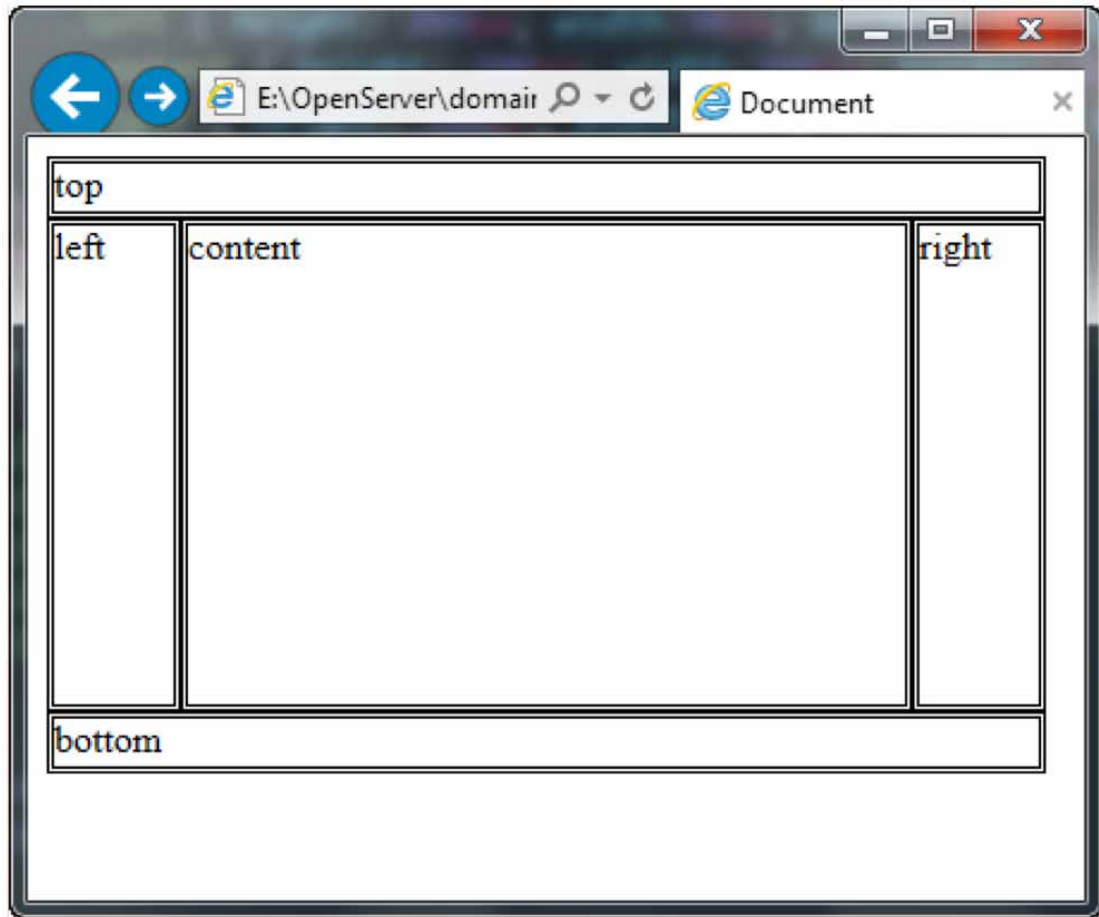
Атрибут **clear** может принимать следующие значения:

<b>left</b>	элемент будет расположен ниже всех элементов, значение атрибута float у которых равно left
<b>right</b>	элемент будет расположен ниже всех элементов, значение атрибута float у которых равно right
<b>both</b>	элемент будет расположен ниже всех элементов, значение атрибута float у которых равно left или right



**none**      отмена свойств атрибута clear

**inherit**    значение наследуется от родителя



```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Document</title>
    <style type="text/css">
      .top {
        height: 20px;
        width:412px;
        border: 3px double black
      }
```

```

        .left {
            height: 200px;
            width:50px;
            border: 3px double black;
            float: left
        }
        .content {
            height: 200px;
            width:300px;
            border: 3px double black;
            float:left
        }
        .right {
            height: 200px;
            width:50px;
            border: 3px double black;
            float:left
        }
        .bottom {
            height: 20px;
            width:412px;
            border: 3px double black;
            clear: left
        }
    </style>
</head>
<body>
    <div class = "top">top</div>
    <div class = "left">left </div>
    <div class = "content">content</div>
    <div class = "right">right</div>
    <div class = "bottom">bottom</div>
</body>
</html>

```

## Переполнение контейнеров

Для управления отображением содержимого контейнера в таких случаях используется атрибут стиля **overflow**, принимающий следующие значения:

<b>visible</b>	отображается весь контент даже за пределами контейнера;
<b>hidden</b>	отображается только область внутри контейнера, оставшаяся часть контента скрывается
<b>scroll</b>	добавление полос прокрутки контейнеру, полосы будут отображаться даже если в них нет необходимости
<b>auto</b>	полосы прокрутки появятся только в случае необходимости

Существует возможность управления отображением содержания контейнера отдельно по горизонтали и вертикали, соответственно при помощи атрибутов **overflow-x** и **overflow-y**, значения которых аналогичны значениям **overflow**.

# 2

## **Позиционирование, поток документа**

Позиционирование элемента на странице определяется свойством **position**. С помощью позиционирования можно менять положение элемента относительно других элементов на странице.

Position может принимать несколько значений: **static**, **relative**, **absolute**, **fixed** и **inherit** (т.е. наследуется от родителя).

Свойство **static** присвоено всем элементам по умолчанию. Поэтому каждый раз определять его явно **position: static;** необязательно. Оно встречается редко, в основном, при переопределении другого значения position.

## Поток документа

Теперь, для рассмотрения других свойств необходимо определить понятие "потока".

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Position</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <div class="wrapper">
      <span>Lorem ipsum dolor sit amet, consectetur
adipisicing elit. Expedita perferendis esse nobis
blanditiis dolores magnam, ipsum possimus et corrupti
laudantium, alias corporis dolore totam vel labore
repellendus sapiente numquam aperiam!</span>
      
      <span>Lorem ipsum dolor sit amet, consectetur
```

```
adipisicing elit. Doloribus ut dolores id fuga illum  
beatae, alias, nostrum vitae quasi accusantium, numquam  
consequuntur consectetur error, quos voluptates animi  
optio quo quisquam!</span>  
    </div>  
</body>  
</html>
```



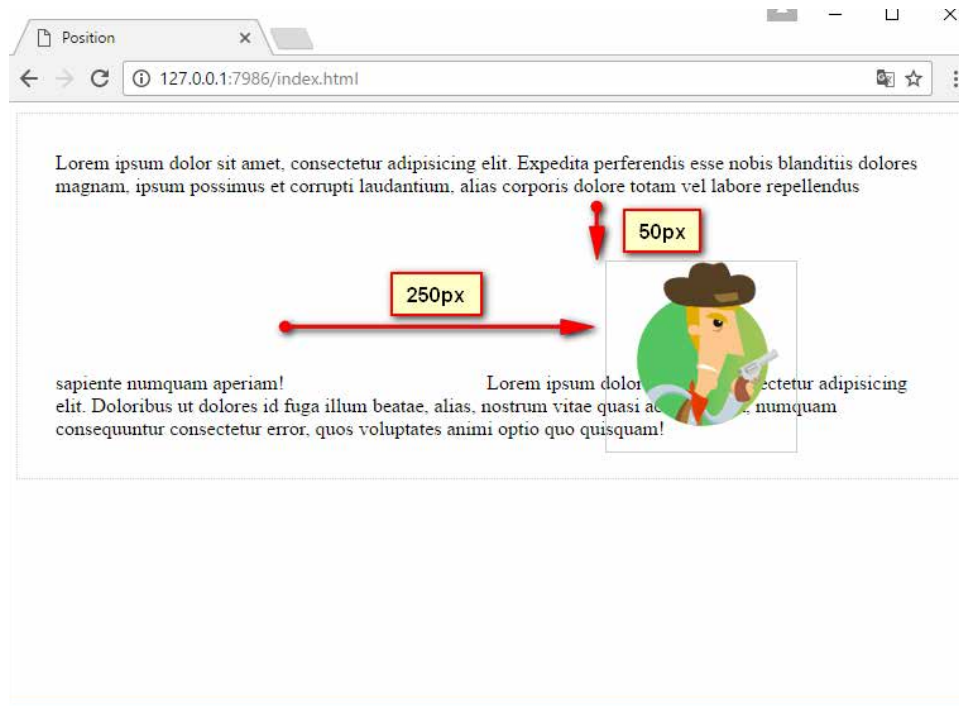
Мы видим, что заполнение документа идет слева направо, сверху вниз, в соответствии с html-разметкой.

При формировании документа, по умолчанию, элементы занимают позицию в соответствии с их местом в коде (в html-разметке). Т.е. слой, размещенный в самом верху кода, отобразится раньше слоя, который расположен в коде ниже.

# Виды позиционирования

Присвоим изображению свойства, задающие относительное позиционирование.

```
.pic {  
  position: relative;  
  top: 50px;  
  left: 250px;  
  border: 1px solid #ccc;  
}
```



Мы видим, что положение картинки изменилось, но в документе осталось свободное место, как если бы она там и была.

Таким образом, с помощью свойства **position: relative;** можно изменять положение элемента относительно его первоначального положения в документе, при этом:

1. Сохраняется стандартный поток документа.
2. Отсчёт координат идёт от места, где должен был бы располагаться элемент.

Посмотрим теперь на действие абсолютного позиционирования **position: absolute;** в этом же примере:

```
.pic {  
  position: absolute;  
  top: 50px;  
  left: 250px;  
  border: 1px solid #ccc;  
}
```



Мы видим, что на место нашего изображения встал следующий элемент, заполнив пробел, а картинка переместилось на другое место, чем при относительном позиционировании.

Таким образом, при абсолютном позиционировании:

1. Элемент выпадает из потока.



2. Отсчёт координат идёт от левого верхнего угла всего документа (либо его позиционированного родителя, см. ниже).

При фиксированном позиционировании **position: fixed;** свойства элементу присваиваются такие же, т.е:

1. Элемент выпадает из потока
2. Отсчёт координат идёт от левого верхнего угла экрана

При прокрутке документа элемент остаётся на месте и не прокручивается вместе со страницей.

## Применение позиционирования в работе

Координаты элемента во всех случаях задаются через свойства **left / top / right / bottom**. В качестве значений можно указывать единицы измерений css, а так же проценты.

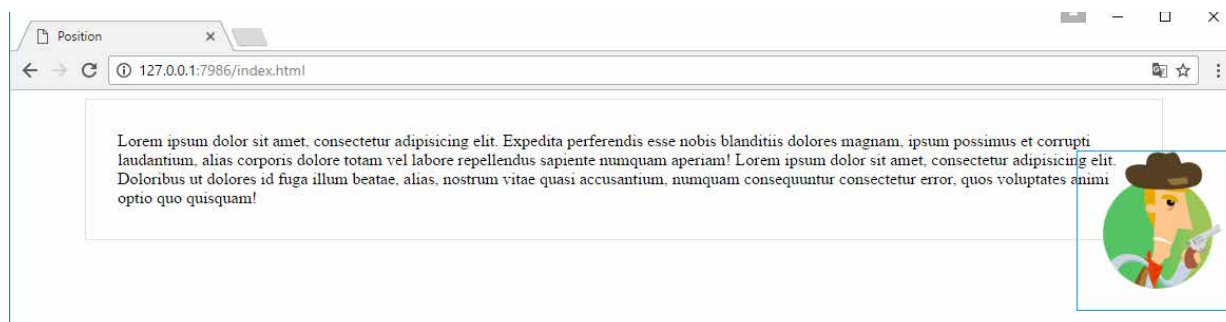
В случае абсолютного или фиксированного позиционирования для того, чтобы прижать элемент к какому-то краю, необходимо соответствующему свойству присвоить **0**. Например, чтобы прижать элемент к низу достаточно выставить значение **bottom: 0;**

## Абсолютное позиционирование внутри другого элемента

Рассмотрим пример, зададим нашему элементу свойства **position: absolute; right: 0;**

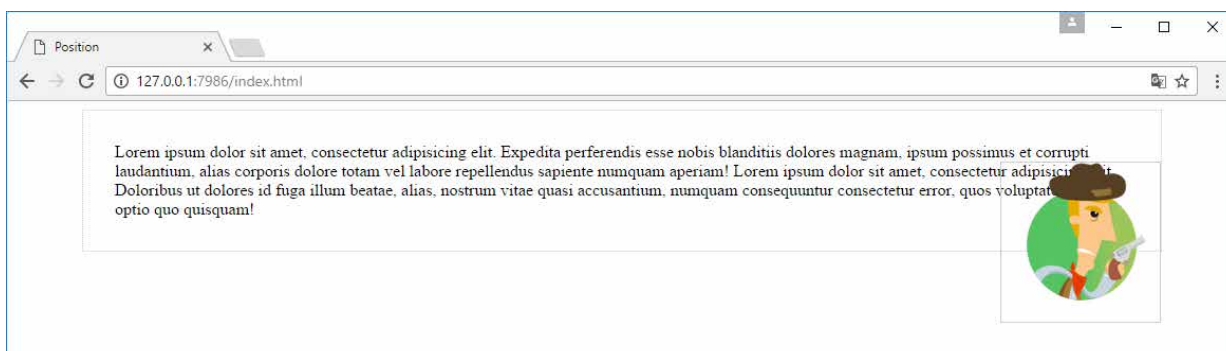
Т.к. не было найдено ни одного родительского элемента для нашей картинки, с каким-либо позиционированием, отличным от static, то наш

элемент позиционируется относительно всего документа.



Присвоим родительскому элементу `position: relative;` и посмотрим на результат.

```
.wrapper {  
  position: relative;  
  margin: 0 auto;  
  min-width: 640px;  
  max-width: 960px;  
  border: 1px dotted #ccc;  
  padding: 30px;  
}
```



Теперь нужный нам элемент позиционируется внутри родительского элемента – выпав из потока, прижимается к его правому краю.

Примечание. Нужно осторожно пользоваться таким позиционированием в случае с инлайновыми элементами. Мы видим, что хоть картинка и прижалась к правому краю, но она перекрывает наш текст. Для того, чтобы текст не перекрывался, необходимо использовать свойство **`float: right;`**

# 3

**Поток  
документа,  
clearfix и зачем  
это нужно**

При использовании свойства **float**, многие новички сталкиваются с целым рядом проблем. И все они вызваны непониманием принципа работы свойства float, ведь когда мы задаем данное свойство для элемента, то он становится "плавающим" элементом и все другие элементы, которые идут за ним, начинают его обтекать.

Основные проблемы следующие:

1. Родительский блок, который внутри себя содержит только лишь элементы со свойством float, "схлопывается" и фактически принимает высоту равную нулю.
2. Если же в родительском блоке, помимо "плавающих" элементов есть и статические элементы к которым мы не применяли float, то высота родителя станет равной высоте этого самого статического элемента.
3. Если после родительского блока, внутри которого находятся "плавающие" элементы, расположить еще какой-то блок, к которому мы не применяем свойство float, то все равно данный блок продолжит свою "поездку" вслед за остальными элементами.

Поэтому верстальщики стали придумывать различные методы, которые бы избавляли от перечисленных выше проблем и при этом не требовали бы добавления дополнительных элементов в структуру HTML. Эти методы называются методами **Clearfix**.

Поэтому блоку-родителю, который содержит элементы с float, достаточно добавить такой класс и записать этот кусок кода в ваш файл стилей:

```
.clearfix:before,  
.clearfix:after {  
    content: "";  
    display: table;  
}  
.clearfix:after {  
    clear: both;  
}
```

# 4

## СВОЙСТВО z-index

Любые позиционированные элементы на веб-странице могут накладываться друг на друга в определённом порядке, имитируя тем самым третье измерение, перпендикулярное экрану. Каждый элемент может находиться как ниже, так и выше других объектов веб-страницы, их размещением по z-оси и управляет **z-index**. Это свойство работает только для элементов, у которых значение **position** задано как **absolute**, **fixed** или **relative**.

Синтаксис:

```
z-index: <число> | auto
```

В качестве значения используются целые числа (положительные, отрицательные и ноль). Чем больше значение, тем выше находится элемент по сравнению с теми элементами, у которых оно меньше. При равном значении **z-index**, на переднем плане находится тот элемент, который в коде HTML описан ниже. Хотя спецификация и разрешает использовать отрицательные значения **z-index**, но такие элементы не отображаются в браузере Firefox до версии 2.0 включительно.

Кроме числовых значений применяется **auto** — порядок элементов в этом случае строится автоматически, исходя из их положения в коде HTML и принадлежности к родителю, поскольку дочерние элементы имеют тот же номер, что и их родительский элемент.

Отрицательный **z-index** перемещает элемент перед **background**, **border**, **box-shadow** контекста наложения. Вложенные элементы отображаются над родителем. Единственный способ поместить дочерний тег ниже родителя — присвоить ему отрицательный **z-index**.

Поскольку **z-index** создаёт свой контекст отображения (что-то вроде своей пирамидки для родителя и вложенных элементов), то возможна ситуация

когда вроде бы большее значение оказывается под меньшим. Вот вам пример, разберите его, чтобы понимать, что происходит.

Советы по использованию свойства **z-index**:

- Если можно не использовать **z-index**, то лучше его не использовать и, например, сразу в HTML коде разместить теги в требуемом порядке.
- Если **z-index** всё же нужен, скажем, для выпадающего меню, то значений 0, 1, 2, 3, 4, 5 вполне достаточно в большинстве случаев, не нужно переходить на десятки, а то и сотни.
- Если **z-index** не работает, то нужно подняться вверх по дереву DOM, чтобы понять где контекст наложения.

# 5

## Пользовательские шрифты



Часто заказчики просят уникальный дизайн, уникальные шрифты, поэтому не всегда получается воспользоваться сервисами, вроде Google Fonts. В этом случае шрифты мы получаем от дизайнера.

Разные браузеры поддерживают разные форматы шрифтов. Поэтому нам нужно будет подключить несколько форматов. Чаще всего от дизайнера мы получаем шрифт лишь в одном формате, поэтому наша задача - сконвертировать их в специальном сервисе.

Существует много способов сконвертировать шрифт, чтобы подключить его к веб-странице, в том числе, онлайн-конвертеры, лучший из которых — [Font Squirrel](#)

Так же можно еще воспользоваться сервисами:

- <https://www.web-font-generator.com/>
- <http://onlinefontconverter.com/>

Достаточно загрузить свой шрифт, выставить необходимые настройки и сервис сгенерирует набор файлов шрифтов, css-файл и файл с демо-примером для подключения шрифта на страницу.

Для того, чтобы подключить шрифты, необходимо:

1. Закинуть шрифты в нужную папку.
2. В CSS-стилях указать следующий код:

```
@font-face {  
  font-family: Graublauweb;  
  
  src: url('Graublauweb.eot'); /* IE9 */
```

```
src: url('Graublauweb.eot?') format('eot'), /* IE6–IE8 */  
      url('Graublauweb.woff') format('woff'), /* Современные  
браузеры */  
      url('Graublauweb.ttf') format('truetype'), /* Safari,  
Android, iOS */  
      url('Graublauweb.svg#svgGraublauweb') format('svg');  
/* iOS */  
}
```

3. Для того, чтобы шрифты отображались во всех браузерах одинаково, на помощь нам приходит CSS-правило **@fontface**. Если у пользователя не установлен нужный нам шрифт, он подгрузится с сервера без установки в операционную систему.

```
p {  
    font-family: Graublauweb, "Arial Narrow", sans-serif;;  
}
```

Не забывайте указывать альтернативный (веб-безопасный шрифт) и общее семейство шрифтов через запятую.

Обратите внимание, что обычные шрифты, установленные на компьютере, включают варианты стиля и плотности (**font-weight, font-style**).

А вот в случае с веб-шрифтами для каждого варианта шрифта вам понадобится отдельный файл.

Например, нам нужно подключить OpenSans шрифты: Open Sans Light, Open Sans regular, Open Sans Semibold, Open Sans Italic (проще говоря, тонкий, обычный, жирный и курсив). Для этого нам нужно создать директорию @font-face:

```
@font-face {
  font-family: 'OpenSans';
  src: url('opensans-regular-webfont.woff') format('woff');
  font-weight: normal;
  font-style: normal;
}

@font-face {
  font-family: 'OpenSans';
  src: url('opensans-light-webfont.woff') format('woff');
  font-weight: 300;
  font-style: normal;
}

@font-face {
  font-family: 'OpenSans';
  src: url('opensans-bold-webfont.woff') format('woff');
  font-weight: bold;
  font-style: normal;
}

@font-face {
  font-family: 'OpenSans';
  src: url('opensans-italic-webfont.woff') format('woff');
  font-weight: normal;
  font-style: italic;
}
```

“Насыщенность шрифта задаётся с помощью ключевых слов: **bold** — жирное начертание и **normal** — нормальное начертание. Также допустимо использовать условные единицы от 100 до 900.

Значения **bolder** и **lighter** изменяют жирность относительно насыщенности родителя, соответственно, в большую и меньшую сторону.

Вот как числовые значения влияют на насыщенность шрифта.

- 100** — тонкое начертание (thin);
- 200** — сверхлегкое (ultra light);
- 300** — легкое (light);
- 400** — нормальное (аналогично normal);
- 500** — среднее (medium);
- 600** — полужирное (semi bold);
- 700** — жирное (аналогично bold);
- 800** — сверхжирное (ultra bold);
- 900** — тяжёлое (black).

Учтите, что не все шрифты поддерживают этот набор. Если указанное значение не поддерживается, то браузер приведёт шрифт к ближайшей насыщенности. К примеру, если вы указали **900** и оно не может быть показано, то браузер в действительности применит значение **700** как ближайшее, которое работает корректно.

Начертание шрифта задается значением **font-style** — обычное, курсивное или наклонное. Когда для текста установлено курсивное или наклонное начертание, браузер обращается к системе для поиска подходящего шрифта. Если заданный шрифт не найден, браузер использует специальный алгоритм для имитации нужного вида текста. Результат и качество при этом могут получиться неудовлетворительными, особенно при печати документа.

Поэтому в примере выше подключены разные файлы (с подходящим начертанием и жирностью). Все они объединены под одним **font-family**, и различаются значением **font-weight** если это различия в жирности шрифта, и свойством **font-style**, если различия в начертании.

# 6

## **Отступы между инлайн- элементами**

Строчные блоки (**inline-block**), во многих случаях, очень удобное средство разметки. Примеры их использования: выравнивание навигации из блоков по центру, центрирование резинового блока по горизонтали. Вместе с тем, с ними связан подводный камень. Каждому верстальщику известны неприятности с интервалом между элементами, пробелами, которые вставляются между словами. Эти пробелы часто мешают при вёрстке того или иного блока. Избавиться от них бывает не так-то просто, а зачастую эти межсловные расстояния и вовсе ставят нас перед выбором, выбором способа решения данной проблемы.

Итак, inline-block представляет из себя блок с поведением строки, т.е. по сути, является **строчно-блочным** элементом. Строчное поведение inline-block позволяет ему оставаться в одной строке с другими строчными элементами, а благодаря своим блочным способностям, inline-block-у можно смело задавать любые свойства, которые присущи блочным элементам: ширину, высоту, верхний и нижний margin, например, уже будет действовать, как у блоков.

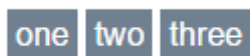
Одна из главных вещей, которые вы должны знать, это то, что наш inline-block является, по сути, обычной буквой — символом, т.е. весь наш строчный блок составляет всего лишь одну букву в строке, одну единицу. Даже несмотря на то, что содержит в себе кучу других символов или элементов. Именно по этой причине inline-block-и не "разрываются", как строчные элементы, а переносятся на следующую строку целиком. Например, у нас есть список:

```
<ul>
  <li>one</li>
  <li>two</li>
  <li>three</li>
</ul>
```

И следующие стили:

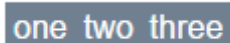
```
body {  
  font-family: sans-serif;  
  font-size: 16px;  
  padding: 5px 20px;  
}  
  
ul {  
  list-style: none  
}  
  
li {  
  background: slategrey;  
  display: inline-block;  
  padding: 4px;  
  color: white  
}
```

В браузере мы получим такую картинку:



one two three

Нам нужно убрать отступы между элементами, чтобы блоки плотно прилегали друг к другу:



one two three

Рассмотрим несколько способов.

Первый способ. Избавимся от пробелов в разметке и напишем код:

```
<ul>
  <li>
    one</li><li>
    two</li><li>
    three</li>
</ul>
```

Выглядит не очень красиво и вместо этого мы можем использовать комментарии:

```
<ul>
  <li>one</li><!--
--><li>two</li><!--
--><li>three</li>
</ul>
```

Но этот вариант тоже оставляет желать лучшего. Еще можно просто не закрывать теги:

```
<ul>
  <li>one
  <li>two
  <li>three
</ul>
```

Но это нарушает валидацию кода и испытание валидатором наш код не выдержит, хотя методом вполне может пользоваться.

Четвёртый способ использует стили. Зададим нашему списку такой класс:



```
<ul class="white-space-fix">
  <li>one</li>
  <li>two</li>
  <li>three</li>
</ul>
```

```
ul.white-space-fix li {
  margin-right: -4px;
}
```

Просто уберём этот отступ отрицательным margin-ом у элемента списка li, но здесь мы привязываемся к текущему размеру шрифта в блоке, что не очень удобно. Но мы видим, что пробел завязан на текущий размер шрифта, а что если сделать размер шрифта у списка нулевым, а потом восстановить у элементов списка? Это и будет пятый способ.

Код:

```
<ul class="zero-size">
  <li>one</li>
  <li>two</li>
  <li>three</li>
</ul>
```

Стили:

```
ul.zero-size {
  font-size: 0px;
}
ul.zero-size li {
  font-size: 16px;
}
```

Наверное, это самый распространённый способ на данный момент.