Вёрстка

LoftSchool
от мыслителя к создателю

Семантический и качественный HTML 3

Viewport и scale 5

Сброс исходных стилей 6

Всё о центрировании 9

Outline 12

Псевдоэлементы 13

Поговорим о шрифтах 15

Свойство «float» и его очистка 21

Как писать правильный css 23

Семантический и качественный HTML

Что это?

Семантическая вёрстка - это вёрстка с правильным использованием HTML-тегов.
С использованием их по назначению, как их и задумывали разработчики языка HTML и веб- стандартов. Например, тег <p> — это абзац, и не стоит им размечать строки веб-форм. А <b> — это просто выделение текста жирным, а вовсе не заголовок.

Зачем это нужно?

Зачем это нужно, ведь информация на сайте отображается точно также, как в дизайн-макете? Зачем ломать голову, если никто это и не оценит, кроме другого разработчика? Давайте разбираться:

- Семантический код чище, в нём легче найти нужный кусок разметки или заметить ошибку.
- Благодаря семантической вёрстке увеличивается скорость загрузки сайта. Меньше кода – легче страница – она быстрее грузится и требует меньше трафика.
- Чаще всего над сайтом трудится команда разработчиков. Если вы даёте классам и id осмысленные имена, коллеги легко разберутся в структуре вашего кода.
- Многие слабовидящие люди полагаются на речевые браузеры для чтения страниц. Правильные теги и их атрибуты нужны, чтобы зачитать содержимое правильно, с нужной интонацией или наоборот не произнести лишнего.
- Семантический код облегчает обновления на сайте, так как вы можете применять стили к заголовкам по всему сайте, а не постранично.
- Поскольку семантический код не содержит элементы дизайна, то изменить внешний вид веб-сайта можно без переписывания всего HTML.
- Поисковые системы постоянно совершенствуют методы поиска, чтобы в результатах была та информация, которуюдействительно ищет пользователь. Семантический HTML способствует этому, т.к. поддается гораздо лучшему анализу — код чище, код логичен (четко видно где заголовки, где навигация, где содержимое).
  Правильный контент + качественная семантическая верстка — это серьезная заявка на хорошие позиции в выдачах поисковиков.

Сравните один блок кода

```
<header class="header">
  <div class="container">
    <div class="header__logo">
      <a href="/" class="logo__link">
        <img src="img/logo.png" alt="Loftschool" class="logo__img">
        <h1 class="logo__text">Школа онлайн-образования Loftschool</h1>
      </a>
    </div>
  </div>
</header>
```

с другим:

```
<div class="shapka">
  <div><p class="img1">
      <a href="/" class="li-221"><img src="img/logo.png"></a>
  </p></div>
</div>
```

Всё очевидно, не так ли?

Как писать семантический и качественный HTML?

- Помните, что HTML – это структура, логика документа. CSS – оформление, а JavaScript – поведение. Не используйте инлайновые стили!

- Каждый тег должен соответствовать своему прямому назначению:

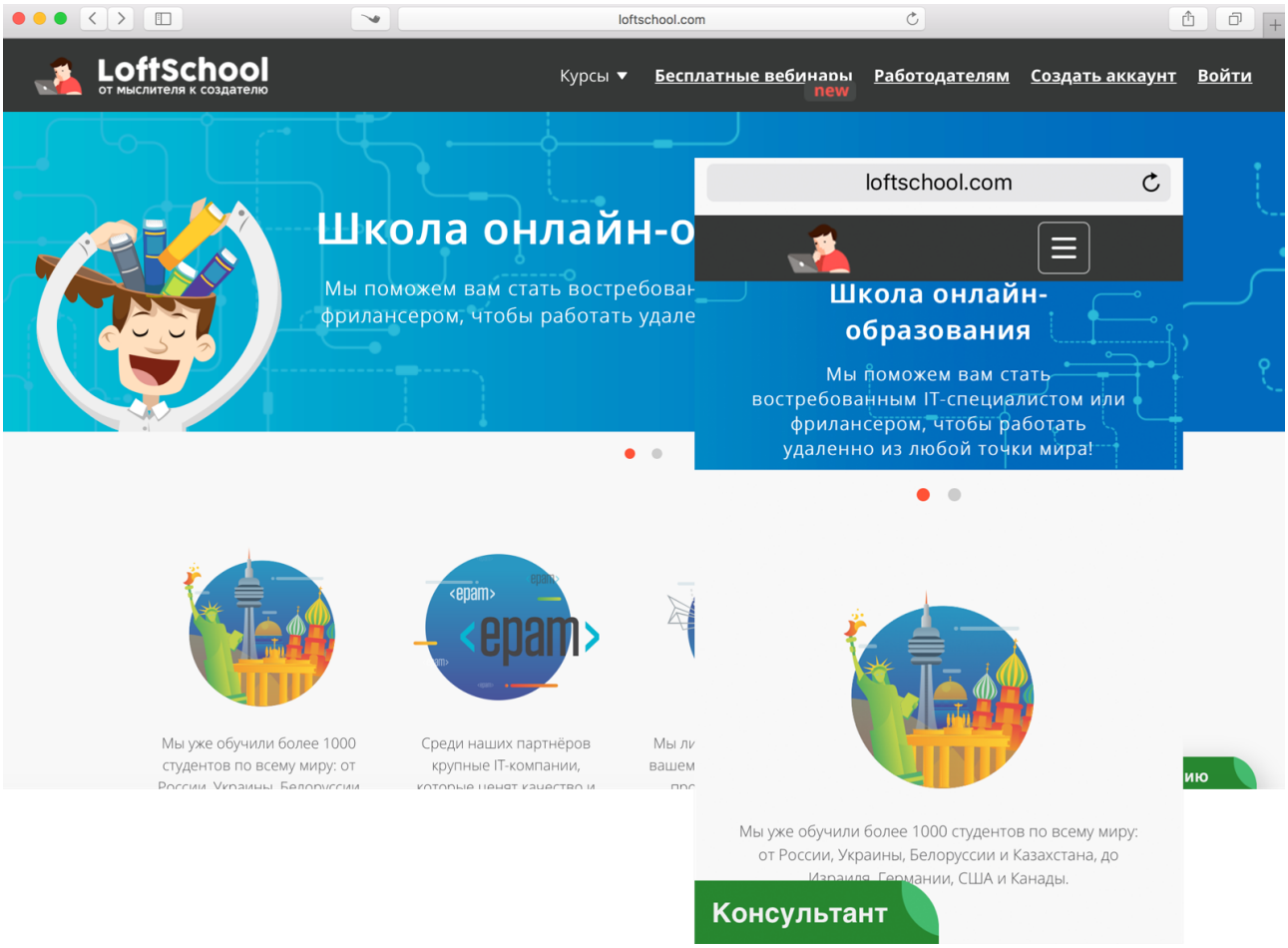
`<header>`	хедер
`<h1>`	заголовок первого уровня
`<p>`	параграф
`<footer>`	футер
`<nav>`	навигация
`<main>`	главный и уникальный контент на странице
`<ul>, <ol>`	списки
`<img>`	изображения
`<a>`	ссылки
`<table>`	таблица

  - Попробуйте отключить таблицу стилей сайта и посмотрите, возможно ли понять его суть и прочитать контент.
  - Все картинки должны иметь `alt` атрибут
  - Блочные элементы не должны находиться внутри строчных
  - А в ссылке обязатеьно должен быть текст, даже если вы помещаете внутрь ссылки картинку.
  - Про то, что тег table давно не используется для построения разметки страницы, еще нужно говорить?
  - Научитесь давать грамотные имена классам и id. Освойте БЭМ-нейминг.
  - Применяйте маркированные списки `<ul>`, если у вас есть перечень связанных элементов, например, ссылки навигации или набор кнопок соц. сетей.

- Если нет HTML-тега, соответствующего элементу или набору элементов, которые вы хотите выделить на странице для придания им определенного внешнего вида, пользуйтесь тегами `<div>` и `<span>`
- Но не злоупотребляйте тегом `<div>`, отдавайте предпочтение HTML5-тегам `<article>`, `<section>`, `<nav>`, `<header>`, `<footer>`

Viewport и scale

Viewport - это видимая пользователю область веб-страницы. Т.е. это то, что может увидеть пользователь, не прибегая к прокрутке. Размеры этой области определяются размером экрана устройства.



Метатег viewport был разработан компанией Apple для того, чтобы указывать браузерам на то, в каком масштабе необходимо отображать пользователю видимую область веб-страницы. Другими словами viewport предназначен для того, чтобы веб-страницы отображались (выглядели) правильно (корректно) на смартфонах, планшетах и других устройствах с высокой плотностью пикселей (>200ppi). Данный метатег предназначен в большой степени для адаптивных сайтов, но с помощью него можно улучшить представления веб-страниц, имеющих фиксированную или гибкую разметку.

Рассмотрим использование метатега viewport для адаптивных сайтов.

Включение поддержки тега <meta> viewport для адаптивных сайтов осуществляется посредством добавления всего одной строчки в раздел head веб-страницы:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Атрибут name предназначен для того чтобы указать браузеру, какую именно информацию о странице хотим ему сообщить. В данном случае эта информация касается viewport.
Контент (содержимое) этих сведений указывается в качестве значения атрибута content посредством пар ключ-значение, разделённых между собой запятыми. Для адаптивного дизайна значения атрибута content viewport должно определяться двумя параметрами:

width=device-width и initial-scale=1

Рассмотрим каждый из них более подробно.
Первый параметр width=device-width отвечает за то, чтобы ширина видимой области веб-страницы равнялась CSS ширине устройству (device-width). Эта ширина (CSS) - не физическое разрешение экрана. Это некоторая величина, независящая от разрешения экрана. Она предназначена для того, чтобы мобильный адаптивный дизайн сайта отображался на всех устройствах одинаково независимо от их плотности пикселей экрана.

Второй параметр initial-scale - устанавливает первоначальный масштаб веб-страницы. Значение 1 означает то, что масштаб равен 100%.

А что с неадаптивным дизайном?
Если сайт не имеет адаптивный дизайн, то его представление на экране смартфона тоже можно улучшить. Можно сделать так чтобы ширина страницы масштабировалась под ширину устройства.
Осуществляется это тоже с помощью установления параметру width значения device-width. Т.е. для не адаптивных сайтов в раздел head необходимо добавить следующую строчку:

```
<meta name="viewport" content="width=device-width">
```

Хорошая статья для тех, кто хочет узнать больше

Сброс исходных стилей

Если мы создадим страницу на «голом» HTML без оформления и стилей, браузер все равно отобразит содержание тега <h1> крупным и жирным, <h2> - чуть меньшим размером, выделит текст в теге <i> курсивом, <u> сделает подчеркнутым, а <b> - жирным.
Произойдет так потому, что каждый браузер имеет по умолчанию некий набор базовых стилей, которые он применяет к странице по умолчанию. И дело в том, что в разных

браузерах эти правила немного отличаются. Лет 10 назад эти отличия были кардинальными, и очень бросались в глаза. Сейчас они минимальны, но все же есть.
Чтобы убрать эти различия, и сделать по умолчанию отображение страницы во всех браузерах одинаковым - используются специальные .css файлы: Eric Meyer's CSS Reset или Normalize.css

Еще один способ, который многие любят применять, и который мы категорически не рекомендуем использовать:

```
* {
  margin: 0;
  padding: 0;
}
```

Мы рекомендуем использовать Normalize.css

Normalize.css обеспечивает для HTML-элементов лучшую кроссбраузерность в стилях по умолчанию. Используя его, можно сэкономить время, которое в случает с CSS Reset было бы потрачено на прописывание сброшенных стилей.

Он исправляет основные баги на мобильных и десктопных устройствах, которые не затрагивает CSS Reset. Это включает в себя параметры отображения элементов HTML5, исправление font-size для предварительно отформатированного текста, отображение SVG в IE9, и многие другие баги, связанные с отображаемым в разных браузерах и операционных системах.

Файл разбит на относительно независимые участки, каждый из которых прокомментирован, что даёт вам возможность удалить блоки свойств (например, нормализацию форм), если известно, что они никогда не понадобятся на сайте. Вот кусок кода из файла Normalize.css:

```
/* Links

========================================================================
==== */

/**
 * Remove the gray background color from active links in IE 10.
 */

a {
  background-color: transparent;
}

/**
 * Improve readability of focused elements when they are also in an
 * active/hover state.
 */

a:active,
a:hover {
  outline: 0;
}

/* Text-level semantics

========================================================================
==== */

/**
 * Address styling not present in IE 8/9/10/11, Safari, and Chrome.
 */

abbr[title] {
  border-bottom: 1px dotted;
```

Помните, файл сброса/ нормализации стилей – это первое, что должен увидеть браузер! И только после него вы подключаете свои стили.

border-box Reset

Свойство box-sizing применяется для изменения алгоритма расчета ширины и высоты элемента.
Согласно спецификации CSS ширина блока складывается из ширины контента (width), значений отступов (margin), полей (padding) и границ (border). Аналогично обстоит и с высотой блока. Свойство box-sizing позволяет изменить этот алгоритм так, чтобы свойства width и height задавали размеры не контента, а размеры блока.

Вероятно, вы использовали в работе вот такой универсальный reset:

```
* {
    box-sizing: border-box;
}
```

Или такой, который нормализует и поведение псевдоэлементов:

```
*, *:before, *:after {
    box-sizing: border-box;
}
```

Оба этих метода работают, однако существует лучшая практика, которую мы рекомендуем использовать. Дело в том, что свойство box-sizing не наследуется, и если мы где-то меняем значение box-sizing на content-box или padding-box, то мы ожидаем, что у всех его детей box-sizing будет тоже content-box, но с предыдущем кодом так не получится.
Поэтому в 2014 году Chris Coyier улучшил этот код, и теперь это считается лучшей практикой border-box Reset

```
html {
    box-sizing: border-box;
}

*, *:before, *:after {
    box-sizing: inherit;
}
```

Теперь все элементы ведут себя так, как мы и ожидаем. Все элементы, кроме html наследуют box-sizing от родителя.
Кстати, 1 февраля – международный день box-sizing!

Узнать больше:

О свойстве box-sizing на htmlbook
Международный день box-sizing и лучшие практики
О box-sizing на css-tricks
border-box Reset

Всё о центрировании

Строчные элементы по горизонтали

Это сработает для элементов с типом отображения inline, inline-block, inline-table,inline-flex и т.д.

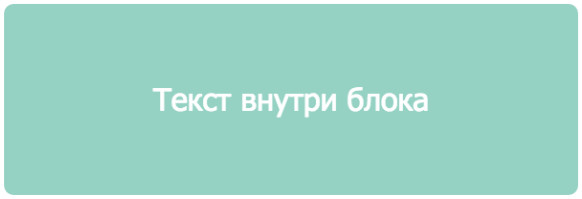
```
.block {
  text-align: center;
}
```

Текст внутри блока

Строчные элементы по горизонтали и по вертикали

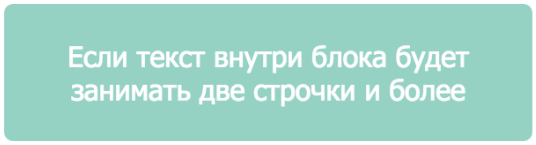
Способ с `line-height` годится лишь в том случае, если вы уверены, что текст всегда будет занимать только одну строку! Значение line-height:100% не будет работать, т.к. проценты берутся не от высоты блока, а от текущей line-height.

```
.block {
  height: 100px;
  line-height: 100px;
  text-align: center;
}
```



Если же текст занимает несколько строк, его можно центрировать так:

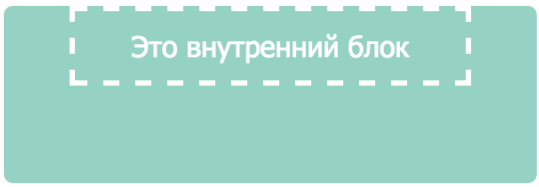
```
.block {
  padding-top: 20px;
  padding-bottom: 20px;
  text-align: center;
}
```



Блочные элементы по горизонтали

Блочный элемент можно центрировать, указав для margin-left и margin-right значение auto (а также прописав для него конкретный width, иначе он займёт всю ширину родительского контейнера и не будет нуждаться в центрировании).

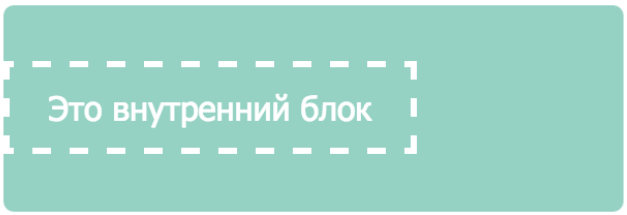
```
.inner {
  width: 200px;
  margin: 0 auto;
}
```



Блочные элементы по вертикали

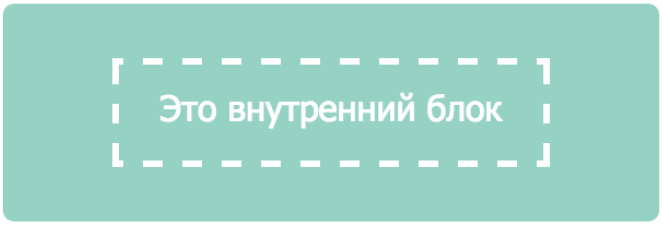
Обычно когда речь идёт о веб-странице, высота может быть неизвестна по многим причинам: если изменяется ширина элементы, перераспределение текста может изменить высоту. Вариации в стилизации текста могут изменить высоту. Изменение количества текста может изменить высоту элемента. Элементы с фиксированным соотношением сторон, например, изображения, могут изменить высоту при масштабировании, и так далее.
Но, если вам известна высота, вертикальное центрирование можно сделать так:

```css
.outer {
  position: relative;
}

.inner {
  position: absolute;
  height: 50px;
  top: 50%;
  margin-top: -25px;
}
```



Если же высота неизвестна, можно центрировать так:

```css
.outer {
  position: relative;
}

.inner {
  position: absolute;
  top: 50%;
  transform: translateY(-50%);
}
```

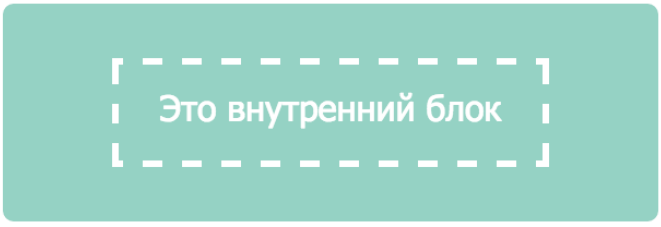


С помощью flexbox:

```css
.outer {
  display: flex;
  flex-direction: column;
  justify-content: center;
}
```

С помощью “элемента-прииизрака”. Перед элементом, который мы центрируем, ставим вспомогательный псевдоэлемент `:before`, занимающий всю возможную высоту. Центрируемый блок выровнен по его середине.

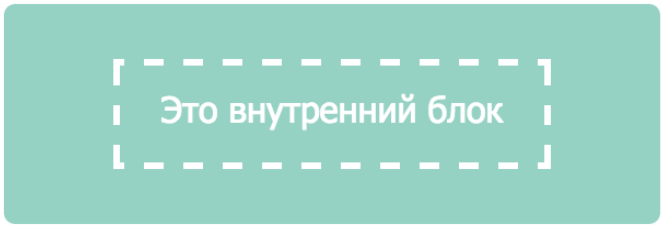
```css
.outer {
  position: relative;
}

.outer:before {
  content: " ";
  display: inline-block;
  height: 100%;
  vertical-align: middle;
}

.inner {
  display: inline-block;
  vertical-align: middle;
}
```



Блочные элементы по горизонтали и вертикали

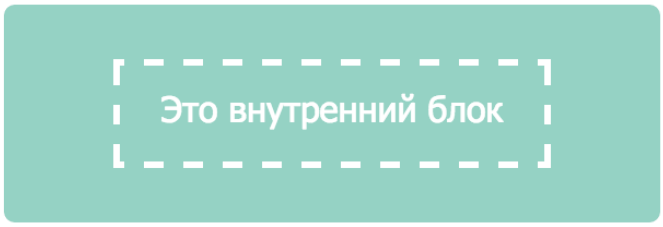
Если у элемента фиксированная высота и ширина, использование отступов с отрицательными значениями, равными половине высоты и ширины элемента после того, как для него было прописано абсолютное позиционирование в точке 50%/50%, отцентрирует элемент внутри родителя.

```
.outer {
  position: relative;
}

.inner {
  width: 200px;
  height: 50px;
  position: absolute;
  top: 50%;
  left: 50%;
  margin: -25px 0 0 -100px;
}
```



Если вам не известны ширина или высота, для центрирования можно использовать свойствоtransform и отрицательное значение translate по 50% в обеих направлениях (оно вычисляется от текущей ширины/высоты элемента):

```
.outer {
  position: relative;
}

.inner {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```



И с помощью flexbox:

```
.outer {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

Outline

Свойство outline задаёт контур элементу, так же как и border. Разница в том, что контур, заданный с помощью outline, не занимает места. То есть не увеличивает высоту и ширину самого элемента (как border), не учавствует в блочной модели CSS. Также, в отличие от border, рамку outline можно задать только со всех сторон: свойствоutline-top, outline-left не существует.

Чаще всего его часто используют для стилей :hover и других аналогичных, когда нужно выделить элемент, но чтобы ничего при этом не прыгало.

```
.block {
  outline: dashed 3px red;

}
```

```
.block {
  border: dashed 3px red;

}
```

Псевдоэлементы

Псевдоэлементы похожи на виртуальные элементы, которые мы можем обрабатывать как обычные HTML-элементы. Но они не существуют в дереве документа или в DOM, мы создаем их с помощью CSS.

Когда использовать и не использовать генерируемый контент в CSS

Генерируемый контент в CSS реализуется с помощью комбинации свойства content с псевдоэлементами :before или :after.
Контентом может быть простой текст или контейнер, которым мы манипулируем при помощи CSS, чтобы выводить графическую форму или декоративный.

Генерируемый контент не стоит использовать для важного текста по следующим причинам:

- Он будет недоступен скрин-ридерам;
- Он будет невыделяем;
- Если генерируемый контент использует излишнее содержание для украшения, скрин-ридеры будут читать его, что ухудшит впечатления пользователей.

Используйте генерируемый контент для декорации и незначительного текста, но убедитесь, что он правильно обрабатывается скрин-ридерами, чтобы использующие эту технологию не отвлекались на него. Основывайтесь на методике “прогрессивного улучшения”, когда собираетесь использовать генерируемый контент.

Полное руководство по псевдоклассам и псевдоэлементам

:before и :after

Псевдоэлемент :before применяется для отображения контента до содержимого элемента, к которому он добавляется. Работает совместно со свойством content.
По умолчанию :before создаёт строчный элемент.

Псевдоэлемент, который используется для вывода контента после содержимого элемента, к которому он добавляется. Псевдоэлемент :after работает совместно со свойством content.
По умолчанию :after создаёт строчный элемент.

Свойство content обязательное, иначе эти два псевдоэлемента работать не будут. В него мы пишем тот текст, который хотим что б добавлялся до или после выбранного селектора. Если же мы не хотим никакого текста, просто оставляем в кавычках пустоту.

Одно из самых распространенных применений псевдоэлементов – стрелки на тултипах. Давайте создадим их:


```
.tooltip {
  position: relative;
}

.tooltip:after {
  content: '';
  display: block;
  position: absolute;
  left: 100%;
  top: 50%;
  transform:
translateY(-50%);
  border: 10px solid
transparent;
  border-left-color:
#88b7d5;

}
```

Добавим декоративную кавычку к цитате с помощью элемента :before

```
.inspire {
  position: relative;
}


.inspire:before {
  content: '';
  position: absolute;
  left: -60px;
  top: -10px;
  width: 50px;
  height: 50px;
  background: url("prod/img/quote.png") no-repeat;
  background-size: contain;
}
```

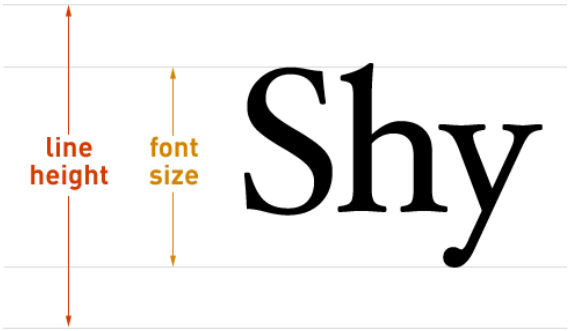
> Непреодолимое стремление к знаниям может стать тем единственным весомым фактором, который поможет вам проделать поразительный путь к успеху. Учиться — значит быть любознательным и искать ситуации для нахождения ответов на свои вопросы. Задавать вопросы до тех пор, пока не узнаете и не поймете все до конца. Только так можно стать самой лучшей версией самих себя.

Поговорим о шрифтах

Font-size и line-height

Что такое «размер шрифта»? Это вовсе не «размер самой большой буквы в нём», как можно было бы подумать.

font-size – это некоторая «условная единица», которая встроена в шрифт.
Она обычно чуть больше, чем расстояние от верха самой большой буквы до низа самой маленькой. То есть, предполагается, что в эту высоту помещается любая буква или их сочетание. Но при этом «хвосты» букв, таких как p, g могут заходить за это значение, то есть вылезать снизу. Поэтому обычно высоту строки делают чуть больше, чем размер шрифта.
Если сделать блок такой же высоты, как шрифт, то хвосты букв будут вылезать из-под него.



line-height - размер строки, обычно он больше размера шрифта. При установке множителем рассчитывается каждый раз относительно текущего шрифта, при установке в единицах измерения – фиксируется.
По умолчанию в браузерах используется специальное значение line-height:normal.
Оно означает, что браузер может принимать решение о размере строки самостоятельно. Как правило, оно будет в диапазоне 1.1 - 1.25, но стандарт не гарантирует этого, он говорит лишь, что оно должно быть «разумным» (дословно – англ. reasonable).
Значение line-height можно указать при помощи px или em, но гораздо лучше – задать его числом.

Значение, заданное множителем, наследуется и применяется в каждом элементе относительно его размера шрифта.
То есть, при `line-height: 2` означает, что высота строки будет равна удвоенному размеру шрифта, не важно какой шрифт.

Значение, заданное в единицах измерения, запоминается и наследуется «как есть».
Это означает, что `line-height: 32px` будет всегда жёстко задавать высоту строки, даже если шрифт во вложенных элементах станет больше или меньше текущего.

Установить font-size и line-height можно одновременно. При этом нужно обязательно указать сам шрифт, например Arial,

sans-serif. Укороченный вариант font: 20px/1.5 работать не будет.

```
font: 20px/1.5 Arial,sans-serif;
```

Дополнительно можно задать и свойства font-style, font-weight:

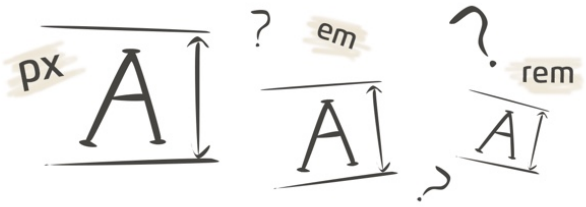
```
font: italic bold 20px/1.5 Arial,sans-serif;
```

Абсолютные единицы измерения. Пиксели

Пиксели (px) — это самая базовая, абсолютная и окончательная единица измерения, которая не зависит от родителя элемента. Количество пикселей задаётся в настройках разрешения экрана, один px — это как раз один такой пиксель на экране. Все значения браузер в итоге пересчитает в пиксели.

Пиксели могут быть дробными, например размер можно задать в 16.5px. Это совершенно нормально, браузер сам использует дробные пиксели для внутренних вычислений. К примеру, есть элемент шириной в 100px, его нужно разделить на три части – волей-неволей появляются 33.333...px. При окончательном отображении дробные пиксели, конечно же, округляются и становятся целыми.

Пиксели удобны, когда вы делаете фиксированный дизайн сайта, а вот если вы делаете адаптивный дизайн, то через медиазапросы размер шрифта придется переопределять для каждого элемента.



Относительные единицы измерения. em

Единица em определяется как текущий font-size. Так, если вы, например, установите font-size на элементе body, то значение em любого младшего элемента в пределах body будет равна этому font-size. Она задаёт размер относительно шрифта родителя.

Кстати, а что если задавать размер шрифта в %? Разницы между % и em здесь нет, так как при задании font-size в процентах, эти проценты берутся от font-size родителя, то есть ведут себя так же, как и em.

Самый главный недостаток этой единицы измерения – «каскадность». Например, если задать элементам обычного списка

```
li {
  font-size: 1.1em;
}
```

то дочерние (вложенные) в `li` элементы будут суммировать это значение. Каждый вложенный `li` получит размер шрифта 1.1 от родителя.
Чтобы избежать этого, нам придется добавлять дополнительные стили:

```
li li {
  font-size: 1em;
}
```

Или же воспользоваться единицей измерения rem, которая для таких случаев как раз и предназначена.

Относительные единицы измерения. rem

rem – задаёт размер относительно шрифта <html>, используется для удобства глобального масштабирования: элементы которые планируется масштабировать, задаются в rem, а JS меняет шрифт у <html>.

Элементы, размер которых задан в rem, не зависят друг от друга и от контекста – и этим похожи на px, а с другой стороны они все заданы относительно размера шрифта <html>.
Единица rem не поддерживается в IE8-.

Единицы измерения относительно экрана. vw, vh, vmin, vmax

Во всех современных браузерах, исключая IE8-, поддерживаются новые единицы из черновика стандарта CSS Values and Units 3:

- vw – 1% ширины окна
- vh – 1% высоты окна
- vmin – наименьшее из (vw, vh), в IE9 обозначается vm
- vmax – наибольшее из (vw, vh)

Эти значения были созданы, в первую очередь, для поддержки мобильных устройств. Их основное преимущество – в том, что любые размеры, которые в них заданы, автоматически масштабируются при изменении размеров окна.

7 единиц CSS, о которых вы должны знать
REM vs EM
Как я научился любить скучные мелочи CSS
REM vs EM – классный ролик на англ.

Конвертация и подключение шрифтов

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.
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:

```
/* Generated by Font Squirrel (http://www.fontsquirrel.com) on January 22, 2016
*/


@font-face {
    font-family: 'fira_sansregular';
    src: url('../fonts/firasans-regular-webfont.eot');
    src: url('../fonts/firasans-regular-webfont.eot?#iefix') format('embedded-
opentype'),
         url('../fonts/firasans-regular-webfont.woff2') format('woff2'),
         url('../fonts/firasans-regular-webfont.woff') format('woff'),
         url('../fonts/firasans-regular-webfont.ttf') format('truetype');
    font-weight: normal;
    font-style: normal;

}
```

...

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.

Подключение шрифта в общий файл стилей:

```
.content {
  font-family: "fira_sansregular", "Arial Narrow", sans-serif;
}
```

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

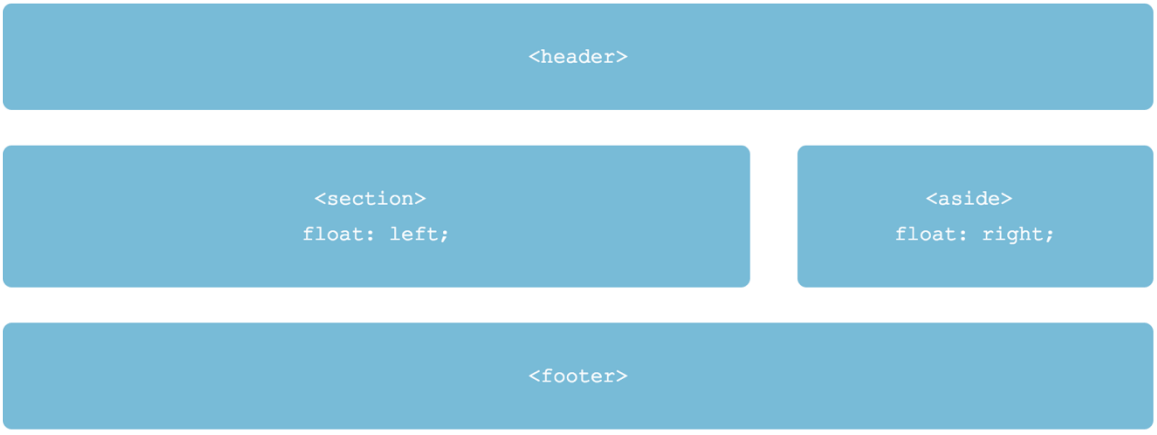
Свойство «float» и его очистка

11 фактов о свойстве float

- Свойство float чаще всего используется для позиционирования элементов на странице. С его помощью мы можем менять их стандартное отображение.

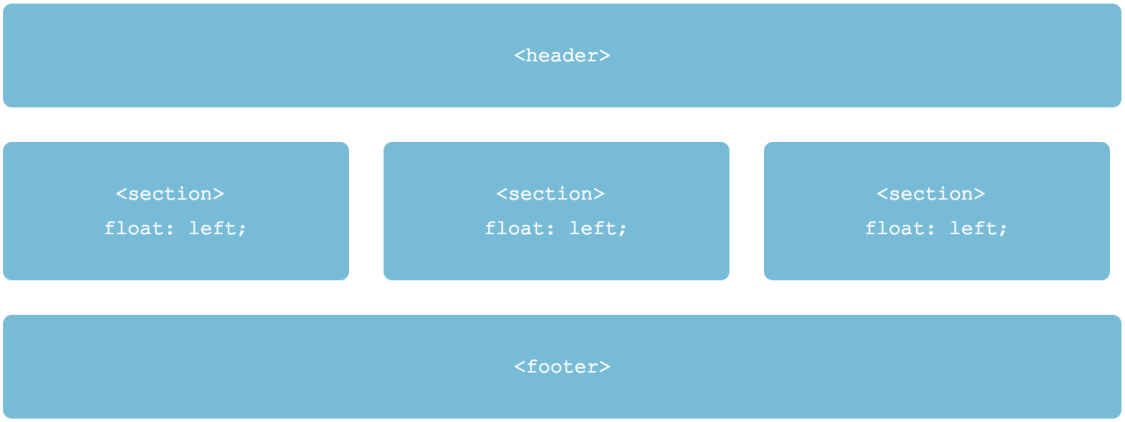
  Макет без float:

  Макет с float:



- Свойство float принимает несколько значений, два самых популярных — это left и right, они позволяют элементу располагаться слева или справа от своего родителя.
- float берет элемент и убирает его из обычного потока, элемент становится плавающим.
- Строчные элементы страницы будут обтекать float-элемент.
- Другие непозиционированные блочные элементы без float ведут себя так, как будто элемента с float нет, так как он убран из потока.

- Элемент, к которому применено свойство float, ведет себя таким образом, как будто соседних блочных элементов (с правилом display:block) не существует.
- Свойство float, примененное к элементу, делает его блочным (какой бы он ни был до этого).
- Если явно не указывать ширину плавающего, то она будет определена его содержимым.
- Если пространства по горизонтали не хватает для того, чтобы вместить плавающий элемент, то он сдвигается вниз до тех пор, пока не начнёт помещаться.
- Когда свойство float применяется к нескольким элементам одновременно, это даёт возможность создать макет с обтекаемыми элементами, расположенными рядом или напротив друг друга. Вот так:



- Вертикальные отступы margin элементов с float не сливаются с отступами соседей, в отличие от обычных блочных элементов.

Очистка float

Когда внутри элемента-родителя содержатся только элементы с float, он схлопывается так, как будто в нем вообще нет дочерних элементов.

Чтобы это исправить, есть специальное свойство clear. Оно устанавливает, с какой стороны элемента запрещено его обтекание другими элементами. Если задано обтекание элемента с помощью свойства float, то clear отменяет его действие для указанных сторон.

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

Как писать правильный css

- Стилизуйте классы, а не теги или id. Стилизация по id и по тегам несет в себе явные (“войны значимости”) и скрытые проблемы (проблемы с масштабированием верстки и пр.). Рекомендуем использовать классы.
- Добавляйте классы ко всем тегам, которые хотите стилизовать.
- Используйте БЭМ-методологию именования классов
- Разбивайте ваш CSS-код по нескольким отдельным файлам (сборка и минификация будет реализована при помощи gulp или любого другого таск-менеджера)
- Однотипные элементы должны иметь одинаковые (общие) классы.
- Размеры картинок нужно ограничивать, а не задавать.
- Для тега <html> нужно прописывать атрибут lang=”ru-RU”
- У всех кнопок и ссылок должны быть отстилизованы 3 состояния: hover, active, focus

Руководство по написанию кода от @mdo