



ЭКСПРЕСС-КУРСЫ ВЕРСТАЛЬЩИКОВ

ТЕМА 2: Верстка HTML/CSS

План

1. Введение;
2. Создание проекта для шаблона. Структура и основные принципы;
3. Правила построения страницы HTML;
4. Принципы и правила создания стилей CSS;
5. Принципы разработки шаблона с помощью Grid сетки;
6. Правила и принципы верстки типовых блоков;

1. Введение;

Этап верстки шаблона является основным этапом разработки шаблона. Сам процесс верстки является сугубо индивидуальным и зависит от шаблона.

Для успешного прохождения данной темы понадобятся следующие инструменты:

- Среда разработки (Sublime Text 3 или WebStorm/PHPStorm);
- Локальный веб-сервер (WAMP, XAMPP и т.п.);
- Менеджер изображений ACDSee 10;

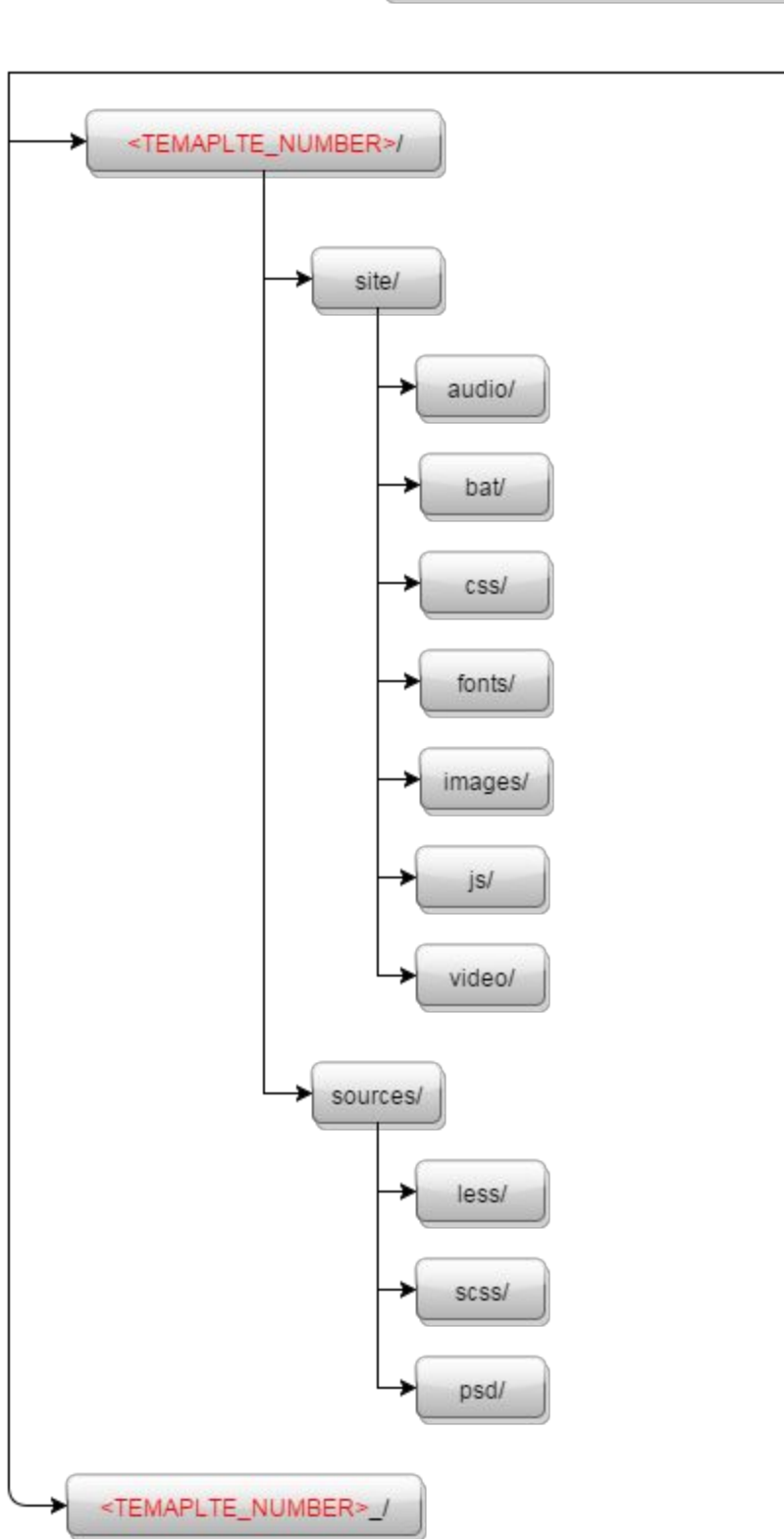
Сам процесс верстки необходимо сопровождать в браузере Mozilla Firefox последней версии.

2. Создание проекта для шаблона. Структура и основные принципы;

Первым этапом верстки шаблона является создание структуры проекта.

Базовая структура проекта должна иметь следующий вид:

<DESIGNER>.<TEMPLATE_NUMBER>.<CODER>.<TEMPLATE_TYPE>



Каждый разрабатываемый дизайн-макет, нарисованный дизайнером, имеет заранее определенный номер и тип.

Поэтому, основная директория вашего проекта должна именоваться по следующему принципу:

`<Имя дизайнера>.<Номер шаблона>.<Ваш никнейм>.<Тип шаблона>`

Данная директория должна содержать в себе еще 2 подкатегории, определенные по номеру шаблона:

`<Номер шаблона>` и `<Номер шаблона>_`, где

`<Номер шаблона>_` - это директория с оригинальными файлами дизайна, картинок и т.д.. В данную директорию необходимо помещать все материалы, которые вы получаете при выдаче шаблона (дизайн-макета) на разработку.

`<Номер шаблона>` - это директория, которая содержит все файлы разрабатываемого шаблона, включая документацию, сам шаблон и исходники к нему.

Папка `site` содержит, непосредственно, сам шаблон включая скрипты, стили, `html` страницы и т.д.

Папка `source` содержит исходный файлы шаблона.

Папка `audio` содержит звуковые файлы, который использовались в шаблоне.

Папка `bat` содержит `*.php` файлы для различных скриптов или модулей шаблона.

Папка `css` содержит файлы стилей для шаблона.

Папка `fonts` содержит все нестандартные локальные шрифты, которые использовались в шаблоне.

Папка `images` содержит все изображения, использованные в шаблоне включая иконку сайта.

Папка `js` содержит все скрипты использованные в шаблоне.

Папка `video` содержит все локальные видеофайлы использованные в шаблоне.

Папка `less` содержит файлы стилей, написанные для препроцессора LESS.

Папка `scss` содержит файлы стилей, написанные для препроцессора SCSS.

Папка psd содержит отсканованные *.psd страницы дизайна, а также иконку сайта.

Несколько слов о правилах оформления структуры проекта:

- В шаблоне должны быть только те файлы и папки, которые используются на страницах. Например, если у вас нет нестандартных локальных шрифтов, то папку fonts включать в проект не нужно.
- Иконка сайта должна иметь название favicon.ico, что в папке images/, что в папке psd/
- Основной файл стилей должен иметь название style.css.
- Названия html страниц должны строиться по следующему принципу. Первая страница шаблона - это index.html, вторая - index-1.html, третья - index-2.html и т.д.
- В файлах стилей не должно быть неиспользуемых или закомментированных стилей.
- HTML страницы не должны содержать закомментированных блоков, секций и т.д.
- Все теги, классы и стили должны писаться в строчном виде.

3. Правила и принципы построения страницы HTML

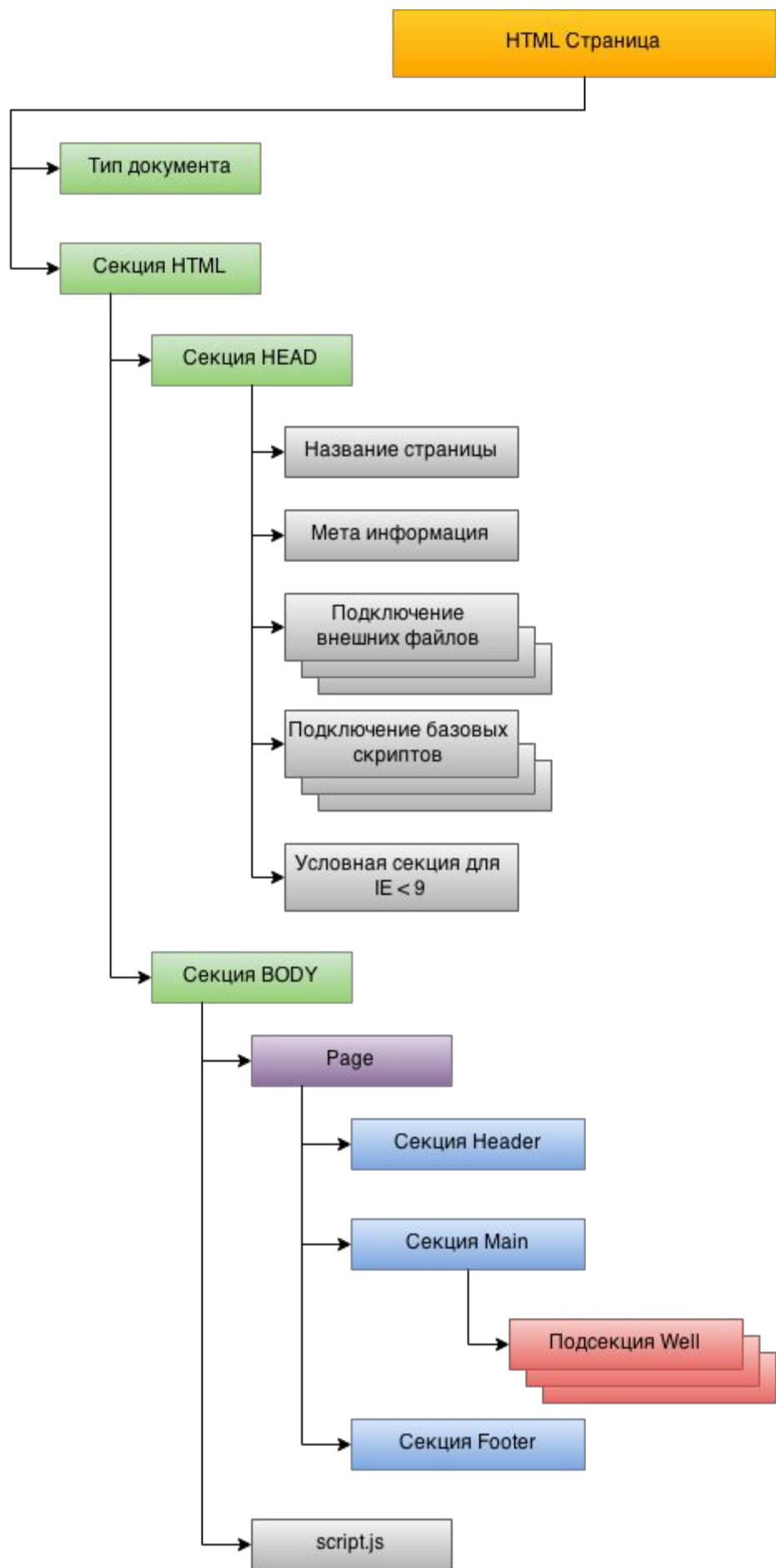
Процесс верстки HTML кода является неотъемлемой частью разработки шаблона. Качество HTML кода напрямую определяется степенью его семантики и минимализма.

Хороший семантический и минимизированный код работает гораздо быстрее, а поиск ошибок и его дальнейшая поддержка становится намного проще.

3.1. Понятие базовой страницы HTML

Под базовой страницей HTML понимается HTML код, который присутствует на каждой странице.

Базовую страницу для наших шаблонов можно представить в виде следующей иерархической структуры.



Типом документа определяется спецификация HTML, используемая на странице. В наших шаблонах используется спецификация 5 версии.

```
<!DOCTYPE html>
```

Секция HTML является контейнером для всего содержимого страницы и определяется с помощью тега

```
<html> ... </html>
```

Секция HEAD используется для определения служебных данных на странице

```
<head> ... </head>
```

Название страницы определяется с помощью тега

```
<title> ... </title>
```

Мета теги содержат информацию для браузеров и поисковых систем. В наших шаблонах определяются следующие мета - теги.

```
<meta charset="utf-8">  
<meta name="format-detection" content="telephone=no"/>
```

Остальные мета теги определяются, в зависимости от ситуации, с помощью скриптов.

Под подключением внешних файлов мы понимаем подключение иконки сайта, а также файлов стилей. В наших шаблонах базовыми внешними файлами являются

```
<link rel="icon" href="images/favicon.ico" type="image/x-icon">  
<link rel="stylesheet" href="css/grid.css">  
<link rel="stylesheet" href="css/style.css">
```

Другие файлы (например, файлы стилей для скриптов) подключаются в отдельном порядке, в зависимости от ситуации.

Под базовыми скриптами, мы понимаем скрипты, от которых напрямую зависит работа всех функциональных модулей страницы.

К таким скриптам в наших шаблонах относятся jQuery, jQuery Migrate и Device.js

```
<script src="js/jquery.js"></script>
<script src="js/jquery-migrate-1.2.1.js"></script>
<script src='js/device.min.js'></script>
```

Условная секция для браузера IE, версии ниже 9, содержит подключение скриптов для добавления поддержки HTML 5 тегов и т.д., определение компонента для отображения уведомления об устаревшем браузере, а также определение секции `<html/>` с классом `.lt-ie9`. Данный класс можно использовать, как определяющий, для создания стилей непосредственно для браузеров IE 8 и ниже. Браузер IE устроен так, что повторное определение данной секции не создает новый контейнер для содержимого страницы а переопределяет старый с добавлением класса `.lt-ie9`. В связи с этим подключение всех базовых скриптов, которые работают с секцией HTML (довешивают классы, атрибуты и т.д.), а именно скрипт Device, в случае с базовыми скриптами, необходимо определять после данного условного комментария, так как, в ином случае, в браузере IE 8 и ниже данные определения будут перезаписаны.

В коде данный комментарий выглядит следующим образом.

```
<!--[if lt IE 9]>
  <html class="lt-ie9">
  <div style=' clear: both; text-align:center; position: relative;'>
    <a href="http://windows.microsoft.com/en-US/internet-explorer/..">
      
    </a>
  </div>
  <script src="js/html5shiv.js"></script>
<![endif]-->
```

Секция BODY содержит тело страницы, то есть тот контент, который будет непосредственно отображаться в окне браузера.

```
<body> ... </body>
```

В наших шаблонах секция BODY содержит блок с классом `.page`, а также подключение скрипта `script.js`

Блок `.page` используется для того, что указать минимальную ширину страницы для браузеров, которые не поддерживают медиа запросы, а также для определения свойства `overflow:hidden` в нем, во избежание появления горизонтального скролла, в случае когда контент будет шире, чем окно браузера.

```
<div class='page'> ... </div>
```

Файл script.js подключается перед закрывающим тегом </body> и содержит в себе определение всех функциональных скриптов в шаблоне в целом.

```
<script src='js/script.js'></script>
```

Секция Header содержит в себе шапку сайта.

```
<header> ... </header>
```

Секция Main содержит в себе основной контент страницы

```
<main> ... </main>
```

Секция Footer содержит в себе подвал сайта

```
<footer> ... </footer>
```

Подсекция Well представляет собой логические подсекции, которые определялись на этапе анализа дизайн макета.

```
<section class='well'> ... </section>
```

В общем виде базовая структура страницы выглядит следующим образом.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Home</title>
  <meta charset="utf-8">
  <meta name="format-detection" content="telephone=no"/>
  <link rel="icon" href="images/favicon.ico" type="image/x-icon">
  <link rel="stylesheet" href="css/grid.css">
  <link rel="stylesheet" href="css/style.css">

  <script src="js/jquery.js"></script>
  <script src="js/jquery-migrate-1.2.1.js"></script>

  <!--[if lt IE 9]>
  <html class="lt-ie9">
  <div style=' clear: both; text-align:center; position: relative;'>
    <a href="http://windows.microsoft.com/en-US/internet-explorer/..">
      
    </a>
  </div>
  <script src="js/html5shiv.js"></script>
  <![endif]-->
```



```

    <script src='js/device.min.js'></script>
</head>

<body>
<div class="page">
    <!--=====
                                HEADER
    =====-->
    <header>

    </header>
    <!--=====
                                CONTENT
    =====-->
    <main>
        <section class="well">

        </section>
    </main>

    <!--=====
                                FOOTER
    =====-->
    <footer>

    </footer>
</div>

<script src="js/script.js"></script>
</body>
</html>

```

3.2. Правила и принципы построения структуры HTML

Процесс верстки шаблона можна привести к некоторому набору правил и принципов:

- Каждую подсекцию шаблона необходимо выделять с помощью тега <section/>;

`<section class='well'></section>`, где

в стилях для класса well задаются внутренний верхний и нижний отступы;

- На последней странице шаблона, перед закрывающим тегом </body>, необходимо указывать, с помощью комментария, кто автор шаблона;

`<!-- coded by Nickname -->`

- Сверстаный HTML код должен проходить валидацию на сайте w3.org;

- Разрабатываемый HTML код должен быть семантически правильным, то есть, для выделения специфических блоков, необходимо использовать соответствующие теги (blockquote для цитаты, address для адреса, dl для определений и т.д.);
- Каждая секция в HTML коде должна выделяться с помощью комментария;

Для секций

```
<!--=====
                        Название Секции
=====-->
```

Для подсекций

```
<!-- Название подсекции --> ..... <!-- END Название подсекции -->
```

В случае простой структуры кода, комментирование подсекций можно пропустить.

- Блоки должны стилизоваться с помощью классов. Идентификаторы можно использовать только для определения блока в скриптах или создания якорей на странице;
- Использование тегов <i>, , <u>, <s> запрещено в HTML коде. Вместо них необходимо использовать аналогичные классы или теги:
 - a. **.italic** для <i>;
 - b. **.bold** или **strong** для ;
 - c. **.underline** для <u>;
 - d. **.strike** для <s>;
- Ссылки на электронную почту должны выделяться с помощью директивы mailto;;

```
<a href='mailto: #'></a>
```

- Для неопределенных ссылок обязательно указывать атрибут href="#";

```
<a href=' #'></a>
```

- Ссылки, содержащие в себе номер телефона или факс, должны выделяться с помощью директивы callto;;

```
<a href='callto: #'></a>
```

- Атрибут alt для изображений можно оставлять пустым

```
<img src='images/page-1_img01.jpg' alt='' />
```

- HTML код не должен содержать инлайновых стилей, за исключением случаев, когда необходимо заранее определять размеры для множества динамически подгружаемых элементов
- HTML код не должен содержать стилей, определенных в тегах `<style></style>`
- HTML код не должен содержать JS скриптов в тегах `<script> </script>`. Все базовые скрипты необходимо подключать в секции HEAD в виде внешних файлов, а все функциональные скрипты должны быть определены в файле `script.js`
- Правильное использование тега `strong` и класса `.bold`. Тег `Strong` необходимо использовать в том случае, когда есть необходимость в указании важности выделенного участка текста. Класс `.bold` должен использоваться только в случае необходимости элементарной стилизации элемента.
- Не в коем случае нельзя определять как заголовок, контент, который таковым не является, например, адреса компаний, индексы в индексированном списке и т.д.. В такой ситуации допускается использовать соответствующие классы: `.heading-1`, `.heading-2`, `.heading-3`, `.heading-4`, `.heading-5`, `.heading-6`.
- Итоговый HTML код должен быть максимально минимизирован.

4. Принципы и правила создания стилей CSS

Процесс создания стилей для шаблона имеет очень важную роль. От качества полученных стилей зависит скорость работы сайта, а также степень простоты его поддержки и отладки.

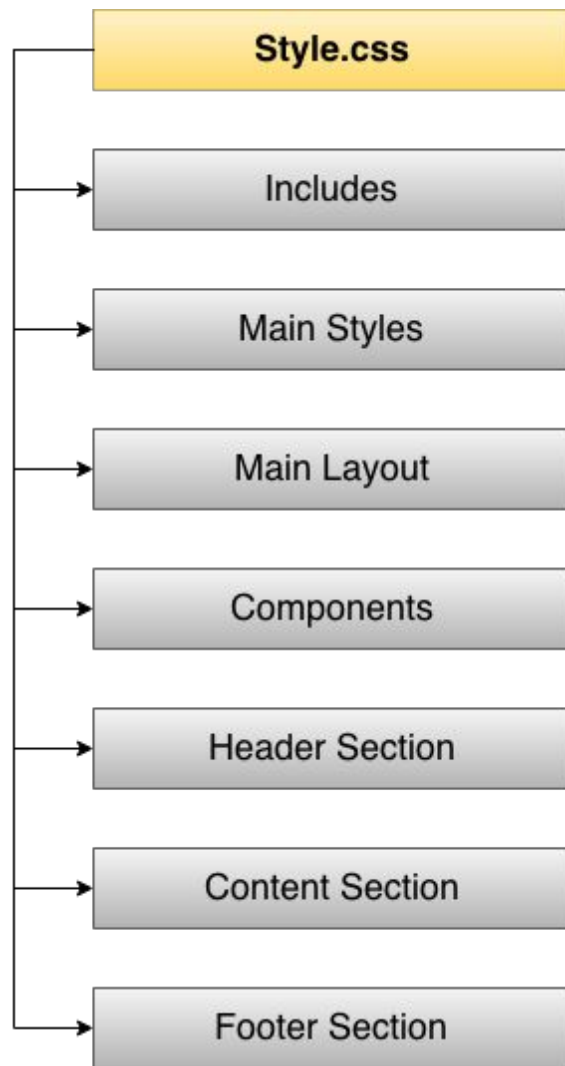
В наших шаблонах используется множество приемов для достижения чистого и прозрачного кода CSS стилей, которые описаны в подразделах ниже.

4.1. Правила и принципы структуризации стилей

Все стили, которые не касаются скриптовых составляющих шаблона (Слайдер, Карусель, Контакт форма и т.д), должны определяться в основном файле `style.css`.

Стили для скриптовых составляющих необходимо выносить в отдельные `*.css` файлы.

Структуру файла `style.css` необходимо разбивать на несколько логических секций, выделенных с помощью комментариев.



Секция Include содержит список директив для подключения необходимых для шаблона CSS файлов, таких как шрифты, animate.css и т.д.

Секция Main Styles содержит список стилей для тегов по умолчанию. В данной секции определяются стили для тегов body, a, h1, h2, h3, h4, h5, h6, p, address, time и т.п. Здесь же определяются базовые классы .heading-N, .italic, .bold, .underline, .color-N и т.д.

Секция Main Layout содержит стили, отображающие взаимосвязь между элементами в виде пар соседних селекторов. В большинстве случаев в данной секции определяются отступы между компонентами и элементами шаблона.

Секция Components содержит стили, определяющие типовые компоненты на странице, такие как индексированные списки, посты и т.д. Каждый компонент шаблона должен выделяться с помощью соответствующего комментария с названием компонента.

Также в данной секции выполняется подключение всех дополнительных компонентов вынесенных в отдельные файлы.

Секция Header Section содержит стили для оформления шапки сайта, а также переопределения стилей для компонентов, которые находятся в шапке. Например, добавление свойства float: left; для компонента “Логотип”, который находится в шапке.

Секция Content Section содержит стили для оформления контентной части сайта. Здесь определяются стили для подсекций контента, а также, при необходимости, переопределяются стили для компонентов, который находятся в контентной части.

Секция Footer Section содержит стили для оформления подвала сайта, а также переопределения стилей для компонентов, которые находятся в нем. Например, выравнивание абзаца по центру и т.д..

Ниже приведен пример для логической структуризации стилей:

```
/*=====
                                     Includes
=====*/
@import url(..../...)

/*=====
                                     Main Styles
=====*/
body{
    ...
}

...
/*=====
                                     Main Layout
=====*/
* + h2{
    ...
}

.....
/*=====
                                     Components
=====*/
.index-list{
    ...
```

```

    }

    ....

/*=====
                                Header Section
=====*/
    header{
        ....
    }

    header .brand{
        float: left;
    }

/*=====
                                Content Section
=====*/
    main{
        ....
    }

    section.well{
        padding-top: 50px;
        padding-bottom: 40px;
    }

/*=====
                                Footer Section
=====*/
    footer{
        ....
    }

```

4.2. Правила подключения типографических и иконочных шрифтов

В наших шаблонах используются типографические шрифты, представленные в сервисе Google Fonts, а также различные иконочные шрифты с сайтов FontAwesome и FlatIcon.

Если некоторые шрифты необходимо определить с помощью директивы `@font-face`, то все стили, связанные с определением каждого из них должны быть

вынесены в отдельный *.css файл. Сами файлы шрифтов при этом должны находиться в отдельной директории fonts/

Если шрифт необходимо подгружать с внешнего хранилища (например, Google Fonts), то в строке пути к внешнему шрифту необходимо убирать определение протокола для передачи файла.

Все шрифты необходимо подключать непосредственно в основном файле стилей вашего проекта (style.css) с помощью директивы @import.

Ниже приведены несколько примеров подключения шрифтов:

```
@import url(my-epic-font.css);  
@import url(//fonts.googleapis.com/css?family=Lato:400,700);
```

4.3. Правила построения базовых стилей

В секции базовых стилей необходимо определять следующие стили:

- Базовый фон (задается в body);
- Базовый цвет (задается в body);
- Базовый шрифт (задается в body);
- Шрифты для заголовков (h1,h2,h3,h4,h5,h6);
- Цвет для заголовков по-умолчанию (h1,h2,h3,h4,h5,h6);
- Базовое оформление ссылок;
- Базовое оформление для картинок;
- Базовые CSS классы: .color-N, .heading-N, .italic, .bold, .underline и т.д

Для всех классов ссылок в шаблоне (включая кнопки) необходимо определять 3 состояния: по-умолчанию, :hover, :active.

4.4. Принципы разделения стилей для элементов на “Собственные” и “Приобретенные”. Основы БЭМ.

Методология БЭМ - это методология построения приложений на основе независимых блоков.

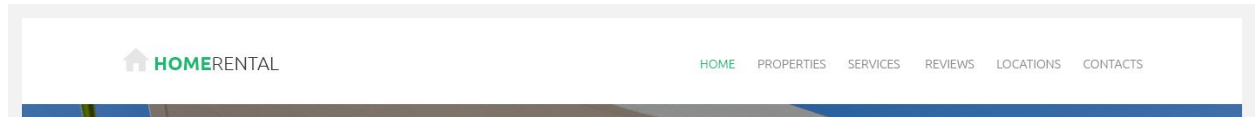
В наших шаблонах применяются некоторые концептуальные моменты данной методологии.

Разработка наших шаблонов заключается в создании набора независимых компонентов, которые можно размещать в любом месте страницы без повреждения отображения лейаута (отображения компонента как части страницы в целом).

Это значит, что разрабатываемые компоненты должны быть максимально независимы от блоков в которых они находятся, то есть содержать стили, которые

относиться непосредственно к самому компоненту, а не наследуются из родительских блоков или добавляются, в зависимости от блока в котором он лежит.

Рассмотрим пример. Пускай есть следующий участок дизайна.



Данный участок дизайна представляет собой шапку сайта, в которой можно логически выделить 2 компонента - бренд компании и навигацию. Рассмотрим компонент бренд.

В данной ситуации мы видим, что бренд содержит логотип и название компании. При этом сам компонент позиционирован в левой части контейнера в котором лежит.

Принцип разделения стилей в данной ситуации будет заключаться в том, что сама стилистика компонента (цвет текста, логотип, шрифт) должны определяться внутри самого компонента, а стили относящиеся к позиционированию компонента в левой части контейнера должны определяться как приобретенные внутри контейнера.

Если рассматривать данный пример с точки зрения коддинга, то мы получим примерно следующий набор стилей:

```
/*===== Brand ===== */
.brand{
  color: #888;
  font-size: 40px;
  line-height: 40px;
  text-transform: uppercase;
}

.brand span{
  font-weight: 700;
  color: #0f0;
}

.brand:before{
  font-size: 55px;
  line-height: inherit;
  color: #ccc;
}

/*=====
Header Section
=====*/
```

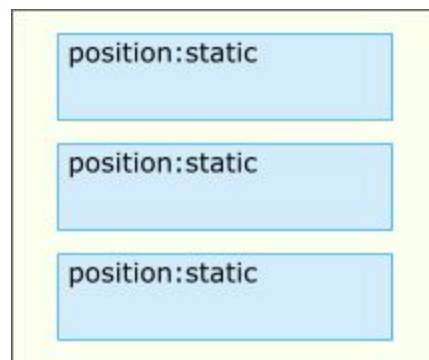


```
header .brand{  
    float: left;  
}
```

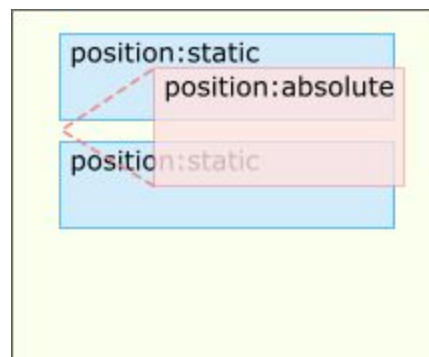
4.5. Принципы позиционирования блоков на странице

Существуют четыре способа позиционирования блоков:

- Static - это способ позиционирования блоков по умолчанию, то есть отсутствие какого бы то ни было специального позиционирования, а просто выкладывание блоков одного за другим сверху вниз.

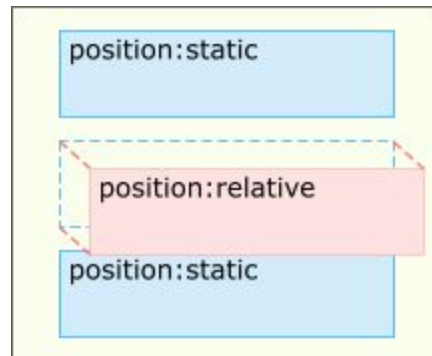


- Absolute - это блок с абсолютным позиционированием, который располагается по заданным координатам и исключается из потока.



- Relative - это блок, который можно спозиционировать с помощью координат относительно того места, где он был бы в потоке, но при этом из потока он не

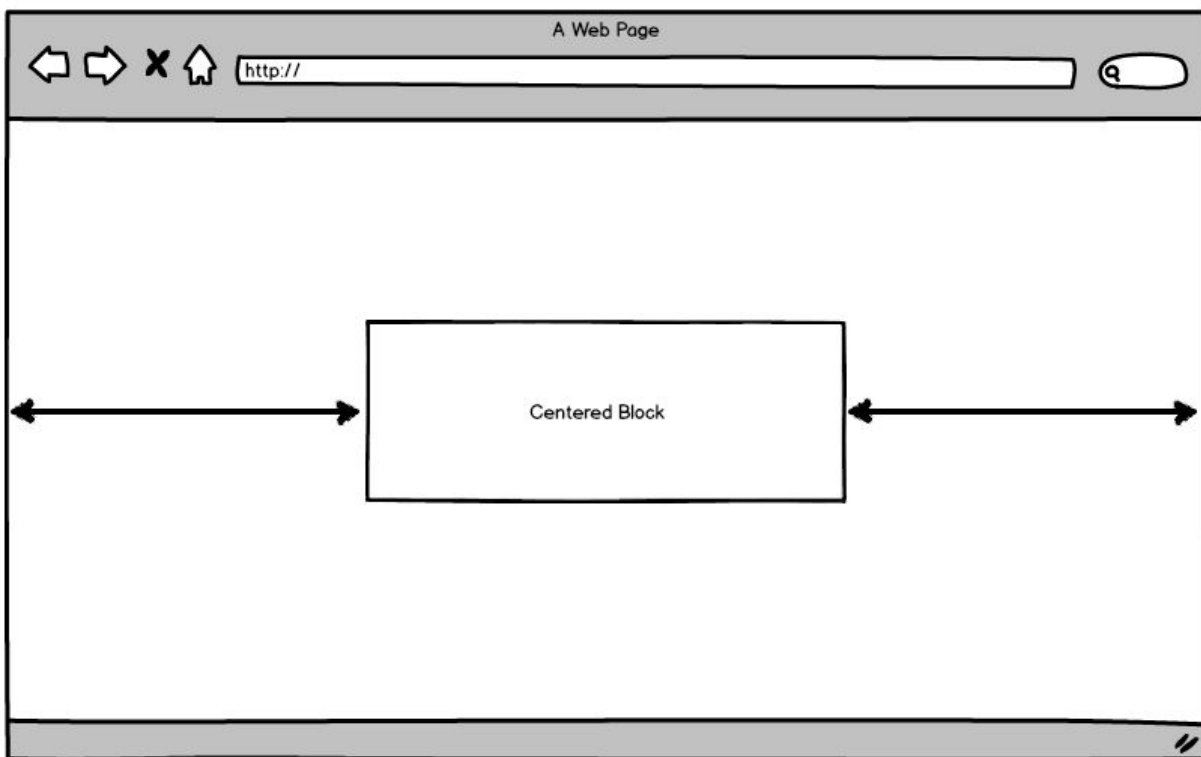
исключается, а продолжает занимать там свое место. То есть он позиционируется только визуально, а положение всех блоков вокруг него никак не меняется.



- Fixed - блок, который позиционируется также как и absolute. Разница заключается лишь в том, что блок позиционируется не относительно документа, а относительно видимой части окна браузера (viewport)

Рассмотрим также часто встречающиеся ситуации, связанные с позиционированием блоков.

- **Позиционирование блока по центру страницы по-горизонтали. Метод 1.**



Суть данного метода заключается в том, чтобы задать фиксированную или максимальную ширину для блока и указать ему внешние левый и правый отступы как автоматические.

На практике данный метод выглядит так:

```
.block{
    width: 500px;
    margin-left: auto;
    margin-right: auto;
}
```

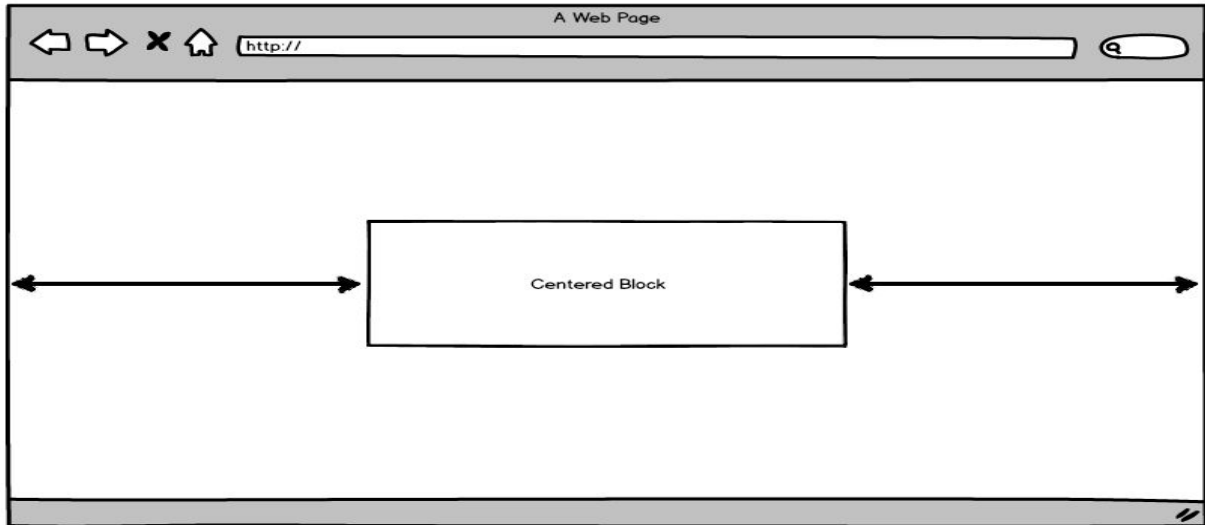
или

```
.block{
    max-width: 500px;
    margin-left: auto;
    margin-right: auto;
}
```

Разница между этими двумя примерами состоит в том, что в первом случае, когда размер блока будет превышать размер контейнера, в котором он лежит, он выйдет за

пределы данного контейнера, когда во втором примере блок полностью заполнит контейнер по ширине.

- **Позиционирование блока по центру страницы по-горизонтали. Метод 2.**



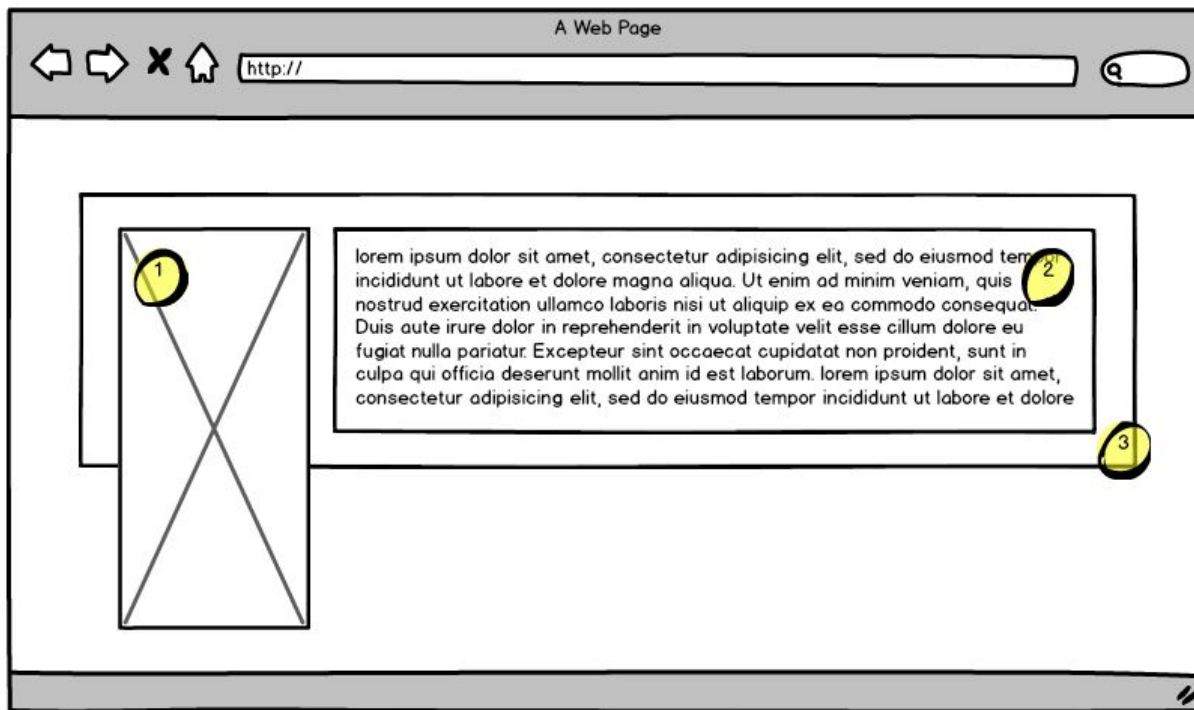
Принцип работы данного метода заключается в том, что мы определяем блок, который необходимо разместить по центру, как блочно-строчный, а контейнеру, в котором он лежит указываем выравнивание текста по центру.

Рассмотрим рабочий пример.

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

.block{
    display: inline-block;
}
```

- **Позиционирование блока справа или слева от основного контента**



Floated Image



Block with text



Container

Часто возникает ситуация, когда необходимо разместить контент так как на изображении выше, где есть некоторая картинка, которая идет слева от основного контента.

Для достижения данного результата необходимо использовать CSS свойство `float`.

В случаях, когда возникает необходимость использования свойства `float`, всегда необходимо помнить принцип его работы:

- Элемент отображается как блочный, так словно ему установлено свойство `display: block`;
- Элемент по ширине сжимается до размеров содержимого, если для элемента явно не установлена ширина `width`;
- Элемент прилипает к левому (`left`) или правому краю (`right`);
- Все остальное содержимое страницы, идущее в HTML коде **после** элемента с `float`, обтекает его;

Также стоит помнить о том, что высота зафлоченного блока игнорируется контейнером в котором он лежит. Это значит, что если возникнет ситуация, когда у вас будет определен некоторый контейнер (красный блок) с единственным зафлоченным блоком внутри (зеленый блок), то высота контейнера будет равна 0.

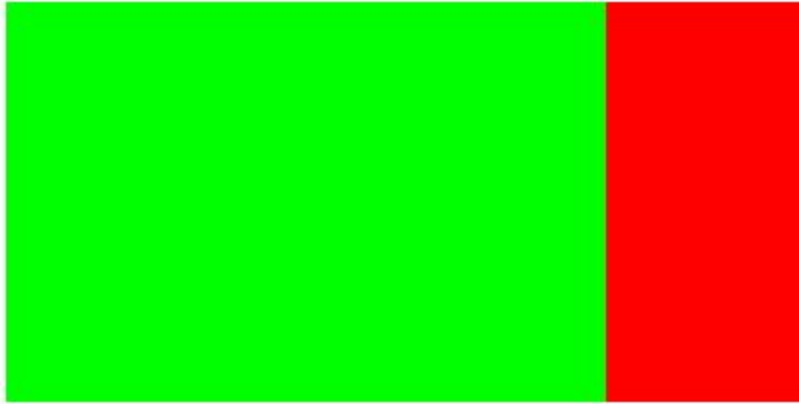


Для решения данной проблемы необходимо использовать специальный прием под названием “clearfix”. Его суть заключается в том, чтобы задать для контейнера в котором лежит зафлоченный блок, псевдо-классы :before и :after со следующими стилями:

```
.container:before,  
.container:after{  
    content: “”;  
    display: table;  
}
```

```
.container:after{  
    clear: both;  
}
```

В таком случае, если высота зафлоченного блока в контейнере будет больше чем высота контента в нем, то высота контейнера автоматически станет равной высоте зафлоченного блока.



Для избежания проблем стоит запомнить одно простое правило. В случаях использования зафлоченных блоков, обязательно необходимо использовать `clearfix` для контейнера в котором он лежит, даже если высота зафлоченного блока будет меньше чем высота контента в контейнере.

Ниже приведен итоговый пример, для решения данной задачи.

```
.box{

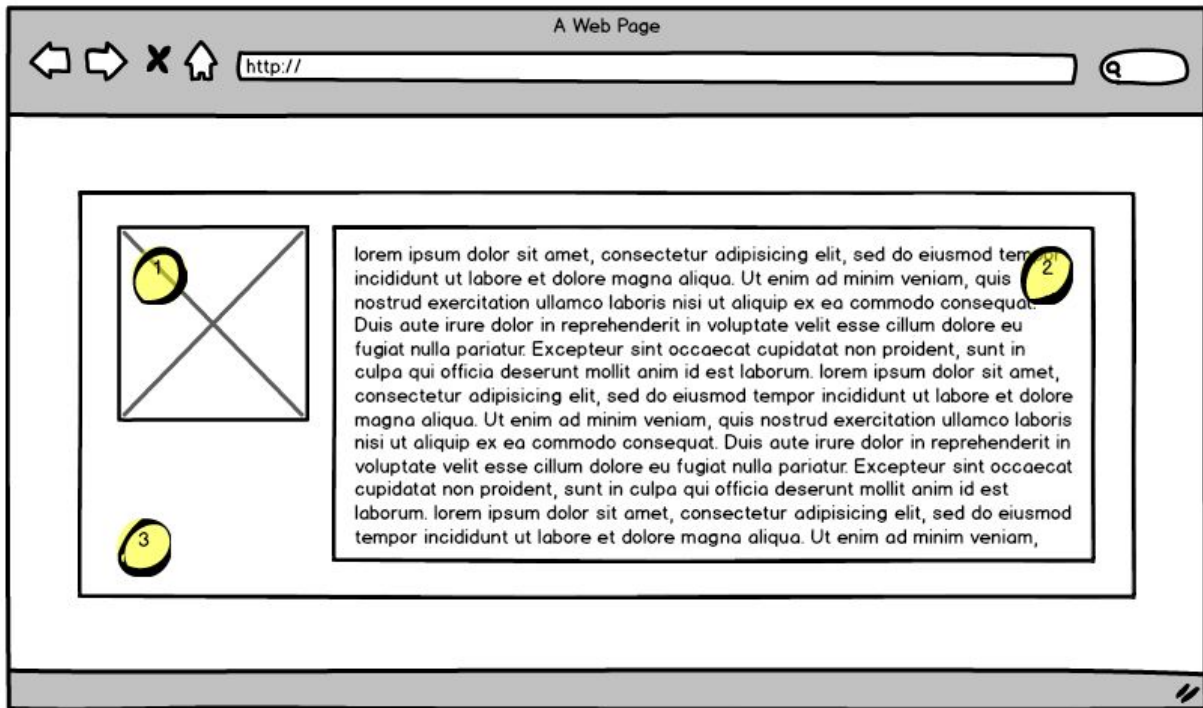
}

.box:before,
.box:after{
    content: '';
    display: table;
}

.box:after{
    clear: both;
}

.box .box_aside{
    float: left;
}
```

- **Позиционирование блока справа или слева от основного контента без обтекания**



Floated Image

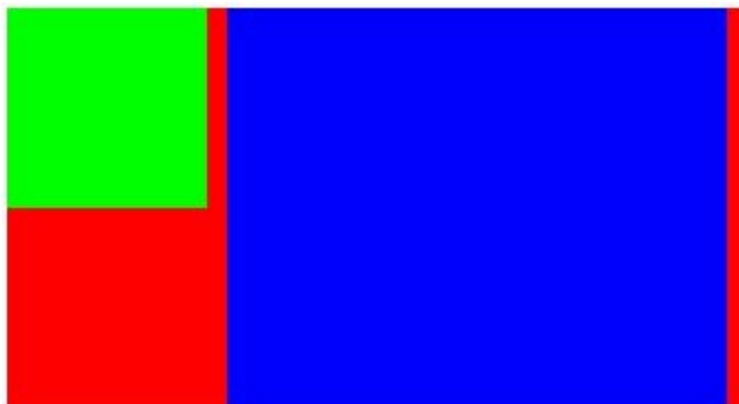


Block with text



Container

Не редко возникает ситуация как в предыдущем примере, когда необходимо разместить некоторый блок слева от основного контента, однако без обтекания при этом. В таком случае также необходимо воспользоваться свойством `float`, с соблюдением всех правил, описанных в предыдущем примере. Кроме этого, также необходимо весь остальной контент поместить в отдельный блок и указать ему свойство `overflow: hidden`. В таком случае контент не будет обтекать зафлоченый блок.



Ниже приведен полный код данного примера

```
.box{
```



```
}

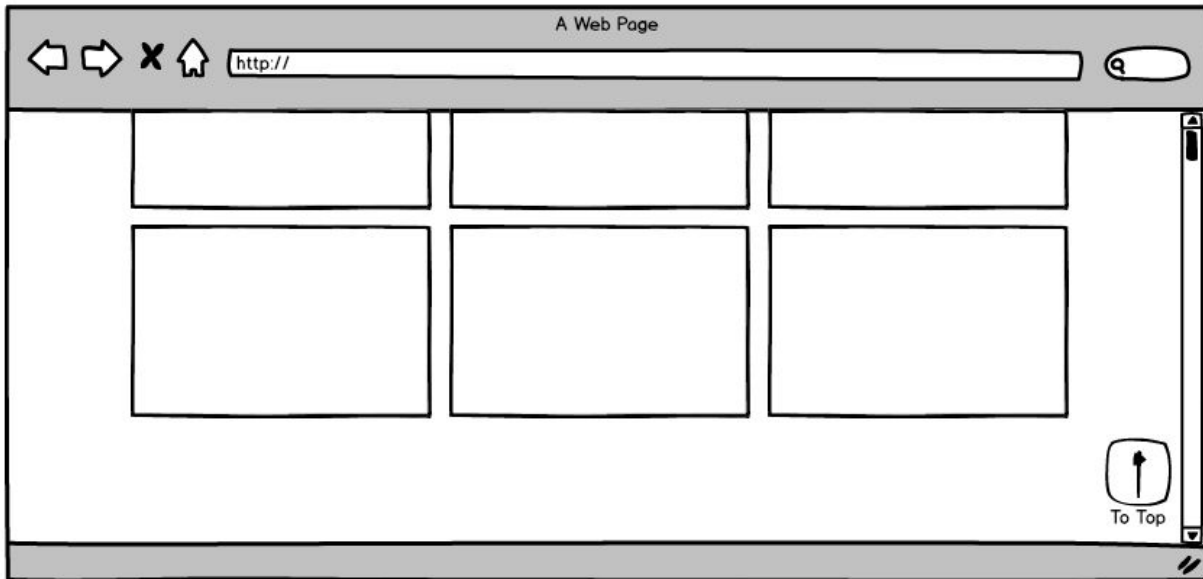
.box:before,
.box:after{
    content: '';
    display: table;
}

.box:after{
    clear: both;
}

.box .box_aside{
    float: left;
}

.box_cnt{
    overflow: hidden;
}
```

- **Фиксированное позиционирование компонента на странице**



Часто возникает ситуация, когда необходимо зафиксировать некоторый компонент в определенном месте видимой части окна браузера (viewport).

На изображении выше необходимо зафиксировать кнопку “To Top”. Для этого необходимо воспользоваться CSS свойством `position: fixed` и разместить компонент в нужном месте с помощью свойств `left`, `top`, `right`, `bottom`;

В данном случае итоговый код будет следующим:

```
.toTop{
    position: fixed;
    right: 20px;
    bottom: 20px;
}
```

4.6. Соседние селекторы с общим предком

Понятие соседних CSS селекторов с общим предком определено в спецификации CSS 2.0.

Соседними называются элементы веб-страницы, которые следуют непосредственно друг за другом в коде документа.

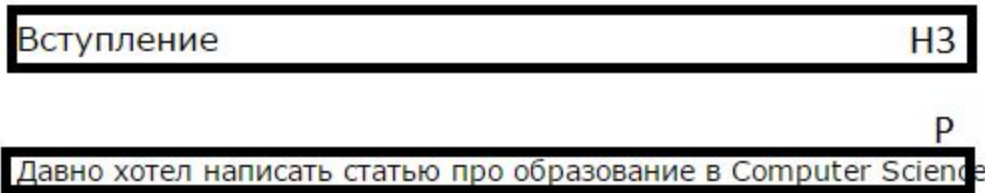
Для управления стилем соседних элементов используется символ плюса (+), который устанавливается между двумя селекторами. Общий синтаксис следующий.

$$E1 + E2 \{ \}$$

Стили указанные внутри фигурных скобок будут применены к элементу E2.

В наших шаблонах данный механизм используется для указания связывающих стилей (например, отступ) между двумя соседними элементами.

Рассмотрим пример.



Предположим есть следующий участок дизайна, где между заголовком и абзацом необходимо задать некоторый отступ.

В наших шаблонах данный отступ необходимо задавать с помощью соседнего селектора **H3 + P**, в случаях, когда отступ заданный по-умолчанию на элемент не подходит по дизайн-макету.

4.7. Приоритеты CSS стилей. Правила специфичности

Специфичность селекторов определяет их приоритетность в таблице стилей. Чем специфичнее селектор, тем выше его приоритет.

Специфичность определяется в виде системы из 4 чисел:

$A - B - C - D$, где

A - равен 1, если в атрибуте `style` у тега указано значение проверяемого CSS свойства;

B - количество идентификаторов в CSS селекторе, в котором указано проверяемое CSS свойство;

C - количество классов, атрибутов и псевдо-классов в CSS селекторе, в котором указано проверяемое CSS свойство;

D - количество тегов и псевдоэлементов в CSS селекторе, в котором указано проверяемое CSS свойство;

Другими словами, можно выделить несколько правил по которым вычисляется специфичность селекторов:

- стили в теге, указанные с помощью атрибута style, имеют специфичность 1,0,0,0;
- каждый присутствующий в селекторе идентификатор добавляет к специфичности 0,1,0,0;
- каждый класс, псевдокласс или атрибут добавляет к специфичности 0,0,1,0;
- каждый элемент и псевдо-элемент добавляет к специфичности 0,0,0,1;
- универсальный селектор и комбинаторы не учитываются.

В случаях, когда для двух и более селекторов определена одинаковая специфичность, к элементу применяется селектор, который в CSS коде идет ниже.

Наличие атрибута !important для CSS свойства в селекторе делает его важным, то есть таким, который перекрывает все остальные. Приоритет !important атрибутов считается по такому же принципу.

Рассмотрим пример:

Пример 1

```
<head>
  <title></title>
  <style>
    a.class1
    {
      color: blue;
    }
    a[href="#"]
    {
      color: green;
    }
    a[href^="#"]
    {
      color: red;
    }
  </style>
</head>
<body>
<a class="class1" href="#">?</a>
</div>
</body>
```

Вопрос: Какого цвета будет знак вопроса в ссылке?

Ответ: Красного, неважно что селектор на точное совпадение атрибута выглядит более специфичным, чем селектор который выбирает все что «начинается с». Вес они имеют одинаковый - [0, 0, 1, 1].

Рассмотрим также несколько примеров из самой спецификации CSS 2.1

```
*          {} /* a=0 b=0 c=0 d=0 -> specificity = 0,0,0,0 */
li         {} /* a=0 b=0 c=0 d=1 -> specificity = 0,0,0,1 */
li:first-line {} /* a=0 b=0 c=0 d=2 -> specificity = 0,0,0,2 */
ul li      {} /* a=0 b=0 c=0 d=2 -> specificity = 0,0,0,2 */
ul ol+li   {} /* a=0 b=0 c=0 d=3 -> specificity = 0,0,0,3 */
h1 + *[rel=up] {} /* a=0 b=0 c=1 d=1 -> specificity = 0,0,1,1 */
ul ol li.red {} /* a=0 b=0 c=1 d=3 -> specificity = 0,0,1,3 */
li.red.level {} /* a=0 b=0 c=2 d=1 -> specificity = 0,0,2,1 */
#x34y      {} /* a=0 b=1 c=0 d=0 -> specificity = 0,1,0,0 */
style=""    {} /* a=1 b=0 c=0 d=0 -> specificity = 1,0,0,0 */
```

4.8 Наследование стилей в CSS

Наследование — это механизм, с помощью которого стили применяются не только к указанным элементам, но и к их потомкам.

Данный механизм позволяет минимизировать код, что позволяет увеличить скорость его работы, а также как следствие влияет на читабельность кода.

Унаследованные CSS свойства не имеют специфичности как таковой, поэтому они успешно перекрываются даже универсальным селектором (*) со специфичностью 0.

Спецификация CSS определяет, что в CSS наследуются все текстовые или косвенно текстовые свойства, часть свойств для таблиц и свойство cursor:

- border-collapse
- border-spacing
- caption-side
- color
- cursor
- direction
- empty-cells
- font-family
- font-size
- font-style
- font-variant
- font-weight

- font
- letter-spacing
- line-height
- list-style-image
- list-style-position
- list-style-type
- list-style
- orphans
- quotes
- text-align
- text-indent
- text-transform
- visibility
- white-space
- widows
- word-spacing

Рассмотрим пример:

```
.body{
    font: 400 14px/24px 'Ubuntu', sans-serif;
}

a{
    text-decoration: underline
}
```

В данном примере, все ссылки, которые будут находится внутри тега body будут иметь шрифт “400 14px/24px ‘Ubuntu’, sans-serif”.

4.10. Нормализация стилей

Все стандартные теги имеют свою стилистическую реализацию в различных браузерах.

Для разработки кроссбраузерной верстки необходимо выполнить сброс всех базовых стилей для браузеров. Мы выполняем сброс стилей с помощью небольшого файла Normalize.css, разработанного Николасом Галахером.

В наших шаблонах этот файл по умолчанию включен в файл с грид сеткой, по этому подключать его отдельно к шаблону не имеет смысла.

5. Принципы разработки шаблона с помощью Grid сетки

Как уже упоминалось в лекции “Работа с дизайн-макетом”, контент в макете выравнивается по специальной 12 колоночной сетке.

Данная сетка имеет CSS реализацию в специальном файле grid.css и основана на реализации известной Bootstrap сетки.

В данном файле определены специальные классы, которые позволяют очень удобным и быстрым способом размещать и выравнивать контент на странице.

По принципу своей работы и структуре построения кода, Grid сетка весьма схожа с обычными таблицами HTML, где применимо к Grid сетке, сама таблица определяется как блок с классом `.container`, строка - как блок с классом `.row`, а ячейка - как блок с классом `.col-RS-X`, где `X` - количество колонок Grid сетки для определения ячейки, а `RS` - соответствующий breakpoint для применения правил (`xs`, `sm`, `md`, `lg`).

При этом, Grid сетка является гораздо более гибким механизмом чем таблицы, так как имеет заранее определенные механизмы для указания размеров блоков и реализации адаптивной верстки.

Рассмотрим следующий пример.



Как видно, на изображении, контент представляется в 3 блоках, равным по ширине 4 колонок.

В таком случае данный участок дизайна можно отобразить в коде с помощью Grid сетки следующим образом.

```
<div class='container'>
  <div class='row'>
    <div class='col-xs-4'> ... </div>
    <div class='col-xs-4'> ... </div>
    <div class='col-xs-4'> ... </div>
  </div>
</div>
```

Использование Grid сетки влечет за собой соблюдение некоторых правил:

- Для размещения контента под Grid сетке, обязательно использование контейнера, то есть блока с классом `.container`.

Правильно:

```
<section class='well'>
  <div class='container'>
    <div class='row'>
```

```

        <div class='col-xs-4'> ... </div>
        <div class='col-xs-4'> ... </div>
        <div class='col-xs-4'> ... </div>
    </div>
</div>
</section>

```

Не правильно:

```

<section class='well'>
    <div class='row'>
        <div class='col-xs-4'> ... </div>
        <div class='col-xs-4'> ... </div>
        <div class='col-xs-4'> ... </div>
    </div>
</section>

```

- Если необходимо разместить несколько “гридов” в один ряд, их необходимо определять внутри блока с классом `.row`. Именно по этому очень важным моментом является определение логической структуры страницы на этапе анализа дизайн-макета.

Правильно:

```

<section class='well'>
    <div class='container'>
        <div class='row'>
            <div class='col-xs-4'> ... </div>
            <div class='col-xs-4'> ... </div>
            <div class='col-xs-4'> ... </div>
        </div>
    </div>
</section>

```

Не правильно:

```

<section class='well'>
    <div class='container'>
        <div class='col-xs-4'> ... </div>
        <div class='col-xs-4'> ... </div>
        <div class='col-xs-4'> ... </div>
    </div>
</section>

```

- Если необходимо задать блок шириной в 12 колонок, то сам блок можно пропустить, а его контент размещать непосредственно в блоке с классом `.container`

Правильно:

```
<section class='well'>
  <div class='container'>
    <h2> ... </h2>
  </div>
</section>
```

Не правильно:

```
<section class='well'>
  <div class='container'>
    <div class='row'>
      <div class='col-xs-12'>
        <h2> ... </h2>
      </div>
    </div>
  </div>
</section>
```

Колонки в Grid сетке также можно использовать для указания отступа слева у элемента.

```
<section class='well'>
  <div class='container'>
    <div class='row'>
      <div class='col-offset-2 col-xs-8'>
        <h2> ... </h2>
      </div>
    </div>
  </div>
</section>
```

Так, в данном примере, в некоторой секции определяется блок шириной в 8 колонов с отступом слева в 2 колонки.

Адаптивность грид сетки реализуется с помощью @media запросов.

При достижении соответствующих разрешений, Grid сетка перестраивает свою структуру:

- Ширина окна X > 1200px - первый breakpoint для Grid сетки;
- Ширина окна X > 992px - второй breakpoint для Grid сетки;
- Ширина окна X > 768px - третий breakpoint для Grid сетки;
- Ширина окна X > 480px - четвертый breakpoint для Grid сетки;

- Ширина окна $X < 480px$ - пятый breakpoint для Grid сетки;

В большинстве случаев адаптивность в наших шаблонах необходимо предусматривать только для данных разрешений.

Файл Grid сетки необходимо подключать в секции HEAD с помощью тега link.

Дополнительная информация по использованию Grid сетки доступна на официальном сайте Bootstrap: <http://getbootstrap.com/css/#grid>.

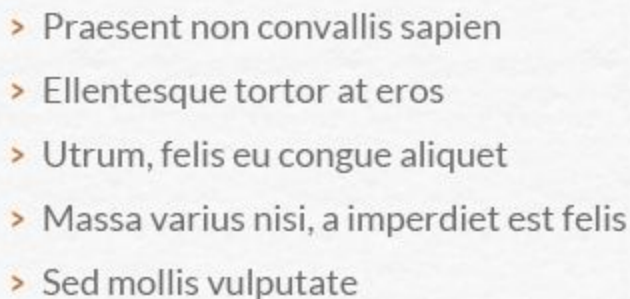
6. Правила и принципы верстки типовых блоков

К типовым блокам относятся следующие компоненты на страницах:

- Маркированный список;
- Индексированный список;
- Строчный список;
- Однострочный список;
- Боксы;
- Иконки;
- Кнопки;
- Секции;
- Миниатюры (thumbs);
- Бренд;

Рассмотрим принципы верстки каждого из них.

Маркированный список. Примером маркированного списка может послужить следующий участок дизайна:



```
> Praesent non convallis sapien  
> Ellentesque tortor at eros  
> Utrum, felis eu congue aliquet  
> Massa varius nisi, a imperdiet est felis  
> Sed mollis vulputate
```

В большинстве случаев маркер в таких списках реализован с помощью шрифтовых иконок FontAwesome.

Так как данный элемент является типовым в наших шаблонах, он должен определяться с помощью класса `.marked-list`.

Ниже приведен пример с базовыми стилями для маркированного списка.

```

.marked-list{

}

.marked-list li{
    position: relative;
    padding-left: 20px;
}

.marked-list li:before{
    content: '\f105';
    position: absolute;
    left: 0;
    top: 0;
    font: 400 14px 'FontAwesome';
    line-height: inherit;
}

```

Значение свойства content в псевдоклассе :before элемента данного списка является кодовым отображением для шрифтовой иконки FontAwesome. Список всех иконок и их кодов можно посмотреть по данной ссылке: <http://fontawesome.github.io/Font-Awesome/cheatsheet/>

Индексированный список. Примером индексированного списка может послужить следующий участок дизайна.

<p>01 Sed do eiusmod tempo</p> <p>Lorem ipsum dolor sit amet conse ctetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor.</p>	<p>02 Ut enim ad minim veniam</p> <p>Lorem ipsum dolor sit amet conse ctetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor.</p>	<p>03 Quis nostrud exercitation</p> <p>Lorem ipsum dolor sit amet conse ctetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor.</p>
<p>04 Lorem ipsum dolor sit amet</p> <p>Lorem ipsum dolor sit amet conse ctetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor.</p>	<p>05 Ullamco laboris nisi ut</p> <p>Lorem ipsum dolor sit amet conse ctetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor.</p>	<p>06 Ipsum dolor sit amet</p> <p>Lorem ipsum dolor sit amet conse ctetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor.</p>

Данный тип списка необходимо реализовывать с помощью счетчиков.

Так как данный элемент является типовым в наших шаблонах, он должен определяться с помощью класса .index-list.

Размер каждого пункта списка не является частью списка, а указывается как часть
 грид сетки.

Ниже приведен пример с базовыми стилями для реализации данного компонента.

HTML.

```
<ol class='row index-list'>
  <li class='col-xs-12 col-sm-6 col-md-4'>
    ...
  </li>
  <li class='col-xs-12 col-sm-6 col-md-4'>
    ...
  </li>
  <li class='col-xs-12 col-sm-6 col-sm-clear col-md-4
col-md-release'>
    ...
  </li>
  <li class='col-xs-12 col-sm-6 col-md-4 col-md-clear'>
    ...
  </li>
  <li class='col-xs-12 col-sm-6 col-sm-clear col-md-4
col-md-release'>
    ...
  </li>
  <li class='col-xs-12 col-sm-6 col-md-4'>
    ...
  </li>
</ol>
```

Класс `.col-*-clear` используется для очищения потока на соответствующем разрешении. Данный класс необходимо использовать на 1 элементе нового ряда соответствующего потока

Класс `.col-*-release` отменяет очищение потока на соответствующем разрешении.

CSS

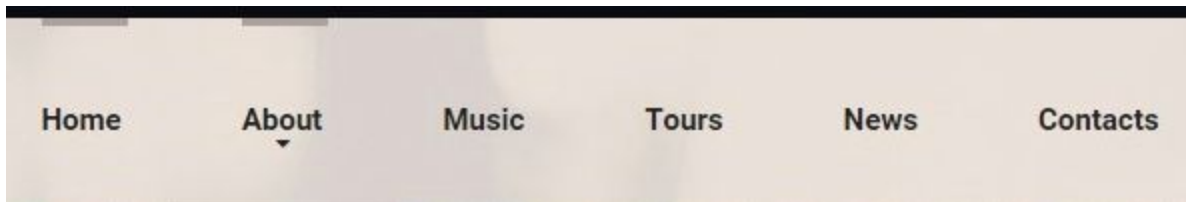
[illegible]

```

        цифры с лидирующим
        нулем */
    counter-increment: li; /* повышаем значение счетчика */
}

```

Строчный список. Примером строчного списка может послужить следующий участок дизайна.



Так как данный тип списка является типовым для наших шаблонов, он должен иметь название `.inline-list` и следующую реализацию.

```

.inline-list {
    word-spacing: 30px;
}

.inline-list > li {
    display: inline-block;
    word-spacing: normal;
}

```

Однострочный список с элементами одинаковой ширины. Примером строчного списка может послужить следующий участок дизайна.



В данном случае в списке определены ссылки с логотипами партнеров некоторой компании. Все элементы списка, при этом, имеют одинаковую ширину. Если убрать или добавить 1 из элементов ширина каждого элемента автоматически пересчитается заново, и все элементы снова будут иметь одинаковую ширину.

Данный тип списка является типовым для наших шаблонов и должен иметь название `.flex-list` и следующую реализацию.

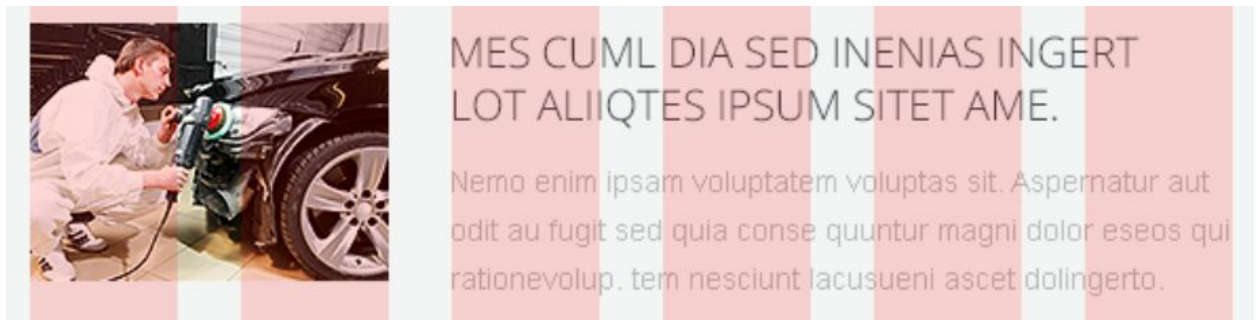
```
.flex-list{
  display: table;
  table-layout: fixed;
  width: 100%;
}

.flex-list li{
  display: table-cell;
}
```

Учитывая табличную структуру данного списка, на мобильных разрешениях данный тип списка необходимо превращать в обычный многострочный список.

```
@media (max-width: 767px){
  .flex-list,
  .flex-list li{
    display: block;
  }
}
```

Боксы. Примером бокса может послужить следующий участок дизайна.



Если в дизайне используется горизонтальное позиционирование некоторого элемента относительно контента, а сам элемент прорисован не по грид сетке, или в `.col-md-1`, `.col-md-2`, то в данном участке дизайна необходимо использовать бокс.

Бокс является типовым элементом в наших шаблонах и должен иметь название `.box`.

Исходный код бокса выглядит следующим образом.

```
.box, .box-xs, .box-sm, .box-md, .box-lg{
  .box__middle {
```

```

        vertical-align: middle;
    }

    .box__bottom {
        vertical-align: bottom;
    }
}

.box{
    .box_left img, .box_right img{
        max-width: none;
    }

    .box_left,
    .box_right,
    .box_cnt {
        padding: 0;
        display: table-cell;
        vertical-align: top;
    }

    .box_left{
        padding-right: 20px;
    }

    .box_right{
        padding-left: 20px;
    }
}

@media (min-width: $screen-xs-min) {
    .box-xs{
        .box_left img, .box_right img{
            max-width: none;
        }

        .box_left,
        .box_right,
        .box_cnt {
            display: table-cell;
            vertical-align: top;
        }

        .box_left{
            padding-right: 20px;
        }
    }
}

```

```

    }

    .box_right{
        padding-left: 20px;
    }
}

.box-xs-clear{
    .box_left img, .box_right img{
        max-width: 100%;
    }

    .box_left,
    .box_right,
    .box_cnt {
        padding: 0;
        display: block;
        vert-align: top;
    }
}

}

@media (min-width: $screen-sm-min) {
    .box-sm{
        .box_left img, .box_right img{
            max-width: none;
        }

        .box_left,
        .box_right,
        .box_cnt {
            display: table-cell;
            vertical-align: top;
        }

        .box_left{
            padding-right: 20px;
        }

        .box_right{
            padding-left: 20px;
        }
    }

    .box-sm-clear{

```



```

    .box_left img, .box_right img{
        max-width: 100%;
    }

    .box_left,
    .box_right,
    .box_cnt {
        padding: 0;
        display: block;
        vert-align: top;
    }
}
}

```

```

@media (min-width: $screen-md-min) {
    .box-md{
        .box_left img, .box_right img{
            max-width: none;
        }

        .box_left,
        .box_right,
        .box_cnt {
            display: table-cell;
            vertical-align: top;
        }

        .box_left{
            padding-right: 20px;
        }

        .box_right{
            padding-left: 20px;
        }
    }
}

```

```

.box-md-clear{
    .box_left img, .box_right img{
        max-width: 100%;
    }

    .box_left,
    .box_right,
    .box_cnt {
        padding: 0;
    }
}

```

```

        display: block;
        vert-align: top;
    }
}

@media (min-width: $screen-lg-min) {
    .box-lg{
        .box_left img, .box_right img{
            max-width: none;
        }

        .box_left,
        .box_right,
        .box_cnt {
            display: table-cell;
            vertical-align: top;
        }

        .box_left{
            padding-right: 20px;
        }

        .box_right{
            padding-left: 20px;
        }
    }

    .box-lg-clear{
        .box_left img, .box_right img{
            max-width: 100%;
        }

        .box_left,
        .box_right,
        .box_cnt {
            padding: 0;
            display: block;
            vert-align: top;
        }
    }
}

```

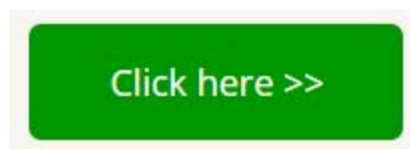
Иконки. Примером иконок может послужить следующий участок дизайна.



Данная иконка является шрифтовой из набора FontAwesome. Ниже приведен пример стилей для ее оформления.

```
.icon{
  width: 90px;
  height: 90px;
  text-align: center;
  line-height: 90px;
  border-radius: 50%;
  background: #FF0;
  color: #fff;
}
```

Кнопки. Примером кнопки может послужить следующий участок дизайна.



Данный компонент является типовым для наших шаблонов, должен иметь название `.btn` и следующую структуру.

```
.btn{
  display: inline-block;
  padding: 8px 20px;
  background: #00f;
  color: #fff;
```

```
        text-align: center;
        line-height: 30px;
    }

    .btn:hover{
        background: #009;
    }

    .btn:active{
        box-shadow: 0 0 10px 0 rgba(0,0,0,.15);
    }
```

Принципиально важно, чтобы кнопка имела 3 состояния: по-умолчанию, :hover и :active.

Если различных кнопок на странице несколько, тогда необходимо вводить градацию кнопок по различным признакам.

Стилизация цветов, теней определяется в классах:

1. btn-default - для кнопки по-умолчанию, (в большинстве случаев она имеет белый фон или белое обрамление);
2. btn-primary - для кнопки с первичным цветом (например, красный цвет, который повсеместно используется в дизайне);
3. btn-secondary - для кнопки с вторичным цветом (в случае, когда вторичных цветов несколько: btn-secondary-1, btn-secondary-2 и т.д.).

Стилизация размеров кнопок определяется с помощью классов:

1. btn-xs - экстремаленькая кнопка;
2. btn-sm - маленькая кнопка;
3. btn-md - средняя по размеру кнопка;
4. btn-lg - большая кнопка;
5. btn-xl - экстрабольшая кнопка;

Если в дизайне, определено больше 5 размеров для кнопок - это расценивается как ошибка дизайна. В такой ситуации необходимо вернуть данный дизайн на исправления дизайнеру или подстраничнице, или исправить в дизайне ошибку самому.

Запрещается использовать классы .btn-1, .btn-2, .btn-3 и аналогичные им.