Введение в HTML5

Что такое HTML

HTML (HyperText Markup Language) представляет язык разметки гипертекста, используемый преимущественно для создания документов в сети интернет. HTML начал свой путь в начале 90-х годов как примитивный язык для создания веб-страниц, и в настоящий момент уже трудно представить себе интернет без HTML. Подавляющее большинство сайтов так или иначе используют HTML.

В 2014 году официально была завершена работа над новым стандартом - HTML5, который фактически произвел революцию, привнеся в HTML много нового.

Что именно привнес HTML5?

- HTML5 определяет новый алгоритм парсинга для создания структуры DOM
- добавление новых элементов и тегов, как например, элементы `video`, `audio` и ряд других
- переопределение правил и семантики уже существовавших элементов HTML

Фактически с добавлением новых функций HTML5 стал не просто новой версией языка разметки для создания веб-страниц, но и фактически платформой для создания приложений, а область его использования вышла далеко за пределы веб-среды интернет: HTML5 применяется также для создания мобильных приложений под Android, iOS, Windows Mobile и даже для создания десктопных приложений для обычных компьютеров (в частности, в ОС Windows 8/8.1/10).

В итоге, как правило, HTML 5 применяется преимущественно в двух значениях:

- HTML 5 как обновленный язык разметки гипертекста, некоторое развитие предыдущей версии HTML 4
- HTML 5 как мощная платформа для создания веб-приложений, которая включает не только непосредственно язык разметки гипертекста, обновленный HTML, но и язык программирования JavaScript и каскадные таблицы стилей CSS 3.

Кто отвечает за развитие HTML5? Этим занимается **World Wide Web Consortium** (сокращенно W3C - Консорциум Всемирной Паутины) - независимая международная организация, которая определяет стандарт HTML5 в виде спецификаций. Текущую полную спецификацию на английском языке можно посмотреть по адресу https://www.w3.org/TR/html5/. И надо отметить, что организация продолжает работать над HTML5, выпуская обновления к спецификации.

Поддержка браузерами

Надо отметить, что между спецификацией HTML5 и использованием этой технологии в веб-браузерах всегда был разрыв. Большинство браузеров стало внедрять стандарты HTML5 еще до их официальной публикации. И к текущему моменту большинство последних версий браузеров поддерживают большинство функциональностей HTML5 (Google Chrome, Firefox, Opera, Internet Explorer 11, Microsoft Edge). В то же время многие старые браузеры, как например, Internet Explorer 8 и более младшие версии, не поддерживают стандарты, а IE 9, 10 поддерживает лишь частично.

При этом даже те браузеры, которые в целом поддерживают стандарты, могут не поддерживать какие-то отдельные функции. И это тоже надо учитывать в работе. Но в целом с поддержкой данной технологии довольно хорошая ситуация.

Для проверки поддержки HTML5 конкретным браузером можно использовать специальный сервис http://html5test.com.

Необходимые инструменты

Что потребуется для работы с HTML5? В первую очередь, текстовый редактор, чтобы набирать текст веб-страниц на html. На данный момент одним из самых простых и наиболее популярных текстовых редакторов является **Notepad++**, который можно найти по адресу http://notepad-plus-plus.org/. К его преимуществам можно отнести бесплатность, подсветка тегов html. В дальнейшем я буду ориентироваться именно на этот текстовый редактор.

Также стоит упомянуть кроссплатформенный текстовый редактор Visual Studio Code. Данный редактор обладает несколько большими возможностями, чем Notepad++, и кроме того, может работать не только в ОС Windows, но и в MacOS и в операционных системах на основе Linux.

И также потребуется веб-браузер для запуска и проверки написанных веб-страничек. В качестве веб-браузера можно взять последнюю версию любого из распространенных браузеров - Google Chrome, Mozilla Firefox, Microsoft Edge, Opera.

Элементы и атрибуты HTML5

Прежде чем переходить непосредственно к созданию своих веб-страниц на HTML5, рассмотрим основные строительные блоки, кирпичики, из которых состоит веб-страница.

Документ HTML5, как и любой документ HTML, состоит из элементов, а элементы состоят из тегов. Как правило, элементы имеют открывающий и закрывающий тег, которые заключаются в угловые скобки. Например:

```
<div>Текст элемента div</div>
```

Здесь определен элемент `div`, который имеет открывающий тег `<div>` и закрывающий тег `</div>`. Между этими тегами находится содержимое элемента div. В данном случае в качестве содержимого выступает простой текст "Текст элемента div".

Элементы также могут состоять из одного тега, например, элемент `<br />`, функция которого - перенос строки.

```
<div>Текст <br /> элемента div</div>
```

Такие элементы еще называют пустыми элементами (void elements). Хотя я использовал закрывающий слеш, но его наличие согласно спецификации необязательно, и равнозначно использованию тега без слеша:

Каждый элемент внутри открывающего тега может иметь **атрибуты**. Например:

```
<div style="color:red;">Кнопка</div>
<input type="button" value="Нажать">
```

Здесь определено два элемента: div и input. Элемент `div` имеет атрибут **style**. После знака равно в кавычках пишется значение атрибута: `style="color:red;"`. В данном случае значение "color:red;" указывает, что цвет текста будет красным.

Второй элемент - элемент input, состоящий из одного тега, имеет два атрибута: `type` (указывает на тип элемента - кнопка) и `value` (определяет текст кнопки)

Существуют глобальные или общие для всех элементов атрибуты, как например, style, а есть специфические, применяемые к определенным элементам, как например, type.

Кроме обычных атрибутов существуют еще булевые или логические атрибуты (boolean attributes). Подобные атрибуты могут не иметь значения. Например, у кнопки можно задать атрибут `disabled`:

```
<input type="button" value="Нажать" disabled>
```

Атрибут disabled указывает, что данный элемент отключен.

Глобальные атрибуты

В HTML5 есть набор **глобальных атрибутов**, которые применимы к любому элементу HTML5:

- **accesskey**: определяет клавишу для быстрого доступа к элементу
- **class**: задает класс CSS, который будет применяться к элементу
- **contenteditable**: определяет, можно ли редактировать содержимое элемента
- **contextmenu**: определяет контекстное меню для элемента, которое отображается при нажатии на элемент правой кнопкой мыши
- **dir**: устанавливает направление текста в элементе
- **draggable**: определяет, можно ли перетаскивать элемент
- **dropzone**: определяет, можно ли копировать переносимые данные при переносе на элемент
- **hidden**: скрывает элемент
- **id**: уникальный идентификатор элемента. На веб-странице элементы не должны иметь повторяющихся идентификаторов
- **lang**: определяет язык элемента
- **spellcheck**: указывает, будет ли для данного элемента использоваться проверка правописания
- **style**: задает стиль элемента
- **tabindex**: определяет порядок, в котором по элементам можно переключаться с помощью клавиши TAB
- **title**: устанавливает дополнительное описание для элемента
- **translate**: определяет, должно ли переводиться содержимое элемента

Но, как правило, из всего этого списка наиболее часто используются три: **class**, **id** и **style**.

Пользовательские атрибуты

В отличие от предыдущей версии языка разметки в HTML5 были добавлены пользовательские атрибуты (custom attributes). Теперь разработчик или создатель веб-страницы сам может определить любой атрибут, предваряя его префиксом *data-*. Например:

```
<input type="button" value="Нажать" data-color="red" >
```

Здесь определен атрибут `data-color`, который имеет значение "red". Хотя для этого элемента, ни в целом в html не существует подобного атрибута. Мы его определяем сами и устанавливаем у него любое значение.

Одинарные или двойные кавычки

Нередко можно встретить случаи, когда в html при определении значений атрибутов применяются как одинарные, так и двойные кавычки. Например:

```
<input type='button' value='Нажать'>
```

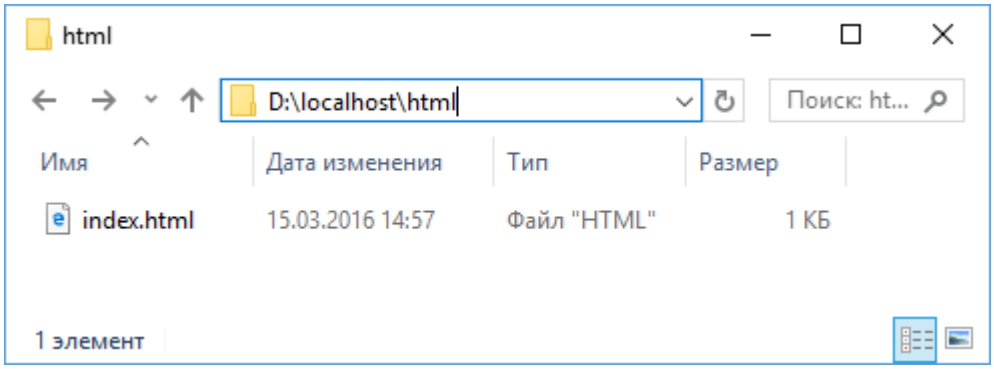
И одинарные, и двойные кавычки в данном случае допустимы, хотя чаще применяются именно двойные кавычки. Однако иногда само значение атрибута может содержать двойные кавычки, и в этом случае все значение лучше поместить в одинарные:

```
<input type="button" value='Кнопка "Привет мир"'>
```

Создание документа HTML5

Элементы являются кирпичиками, из которых складывается документ html5. Для создания документа нам надо создать простой текстовый файл, а в качестве расширения файла указать **.html*

Создадим текстовый файл, назовем его *index* и изменим его расширение на **.html**.



Затем откроем этот файл в любом текстовом редакторе, например, в Notepad++. Добавим в файл следующий текст:

```
<!DOCTYPE html>
<html>

</html>
```

Для создания документа HTML5 нам нужны в первую очередь два элемента: DOCTYPE и html. Элемент **doctype** или Document Type Declaration сообщает веб-браузеру тип документа. `<!DOCTYPE html>` указывает, что данный документ является документом html и что используется html5, а не html4 или какая-то другая версия языка разметки.

А элемент `html` между своим открывающим и закрывающим тегами содержит все содержимое документа.

Внутри элемента `html` мы можем разместить два других элемента: **head** и **body**.
Элемент `head` содержит метаданные веб-страницы - заголовок веб-страницы, тип кодировки и т.д., а также ссылки на внешние ресурсы - стили, скрипты, если они используются.
Элемент `body` собственно определяет содержимое html-страницы.

Теперь изменим содержимое файла index.html следующим образом:

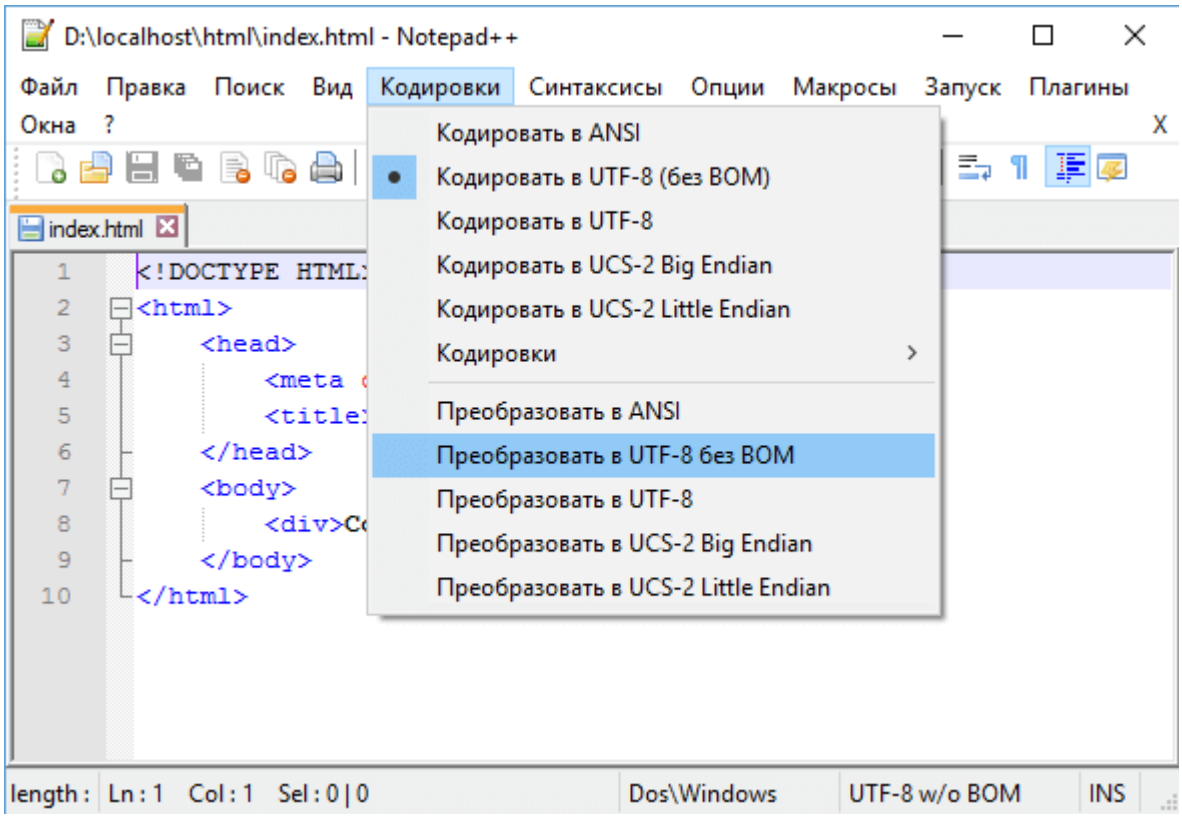
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Документ HTML5</title>
    </head>
    <body>
        <div>Содержание документа HTML5</div>
    </body>
</html>
```

В элементе head определено два элемента:

- элемент `title` представляет заголовок страницы
- элемент `meta` определяет метаинформацию страницы. Для корректного отображения символов предпочтительно указывать кодировку. В данном случае с помощью атрибута `charset="utf-8"` указываем кодировку utf-8.

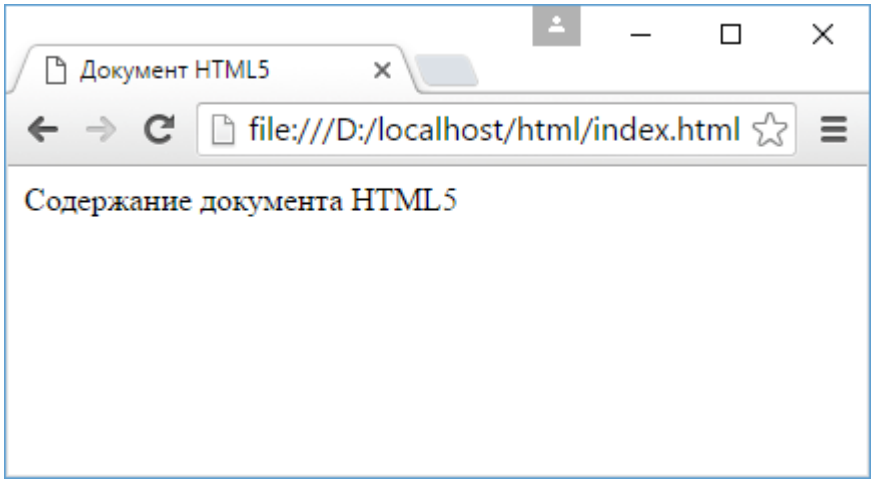
В пределах элемента элемента `body` используется только один элемент - `div`, который оформляет блок. Содержимым этого блока является простая строка.

Поскольку мы выбрали в качестве кодировки utf-8, то браузер будет отображать веб-страницу именно в этой кодировке. Однако необходимо чтобы сам текст документа также соответствовал выбранной кодировке utf-8. Как правило, в различных текстовых редакторах есть соотвествующие настройки для установки кодировки. Например, в Notepad++ надо зайти в меню **Кодировки** и в открывшемся списке выбрать пункт **Преобразовать в UTF-8 без BOM**:



После этого в статусной строке будет можно будет увидеть **UTF-8 w/o BOM, что будет указывать, что нужная кодировка установлена.**

Сохраним и откроем файл *index.html* в браузере:



Таким образом, мы создали первый документ HTML5. Так как мы указали в элементе `title` заголовок "Документ HTML5", то именно такое название будет иметь вкладка браузера.

Так как указана кодировка utf-8, то веб-браузер будет корректно отображать кириллические символы.

А весь текст, определенный внутри элемента `body` мы увидим в основном поле браузера.

Разновидности синтаксиса HTML5

При создании документа HTML5 мы можем использовать два различных стиля: HTML и XML.

Стиль HTML предполагает следующие моменты:

- Начальные открывающие теги могут отсутствовать у элементов
- Конечные закрывающие теги могут отсутствовать у элементов
- Только пустые элементы (void elements) (например, `br`, `img`, `link`) могут закрываться с помощью слеша `/>`
- Регистр названий тегов и атрибутов не имеет значения
- Можно не заключать значения атрибутов в кавычки
- Некоторые атрибуты могут не иметь значений (checked и disabled)
- Специальные символы не экранируются
- Документ должен иметь элемент DOCTYPE

Это так называемый "разрешительный" стиль, основанный на послаблениях при создании документа.

Документ HTML5 также может быть описан с помощью синтаксиса XML. Такой стиль еще называют "XHTML". Он используется, если заголовок content-type имеет значение `application/xml+xhtml`. Для данного стиля характерны следующие правила:

- Каждый элемент должен иметь начальный открывающий тег
- Непустые элементы (non-void elements) с начальным открывающим тегом также должны иметь конечный закрывающий тег
- Любой элемент может закрываться с помощью слеша `/>`
- Названия тегов и атрибутов регистрозависимы, как правило, используются в нижнем регистре
- Значения атрибутов должны быть заключены в кавычки
- Атрибуты без значений не допускаются (`checked="checked"` вместо просто `checked`)
- Специальные символы должны быть экранированы

Сравним два подхода. Подход HTML5:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset=utf-8>
        <title>Заголовок</title>
    </head>
    <body>
        <p>Содержание документа HTML5<br>
        <input type=button value=Нажать >
    </body>
</html>
```

И аналогичный пример с использованием подхода XHTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
```

```
5           <meta charset="utf-8">
6           <title>Заголовок</title>
7       </head>
8       <body>
9           <p>Содержание документа HTML5<br />
10          <input type="button" value="Нажать" /></p>
11      </body>
12  </html>
```

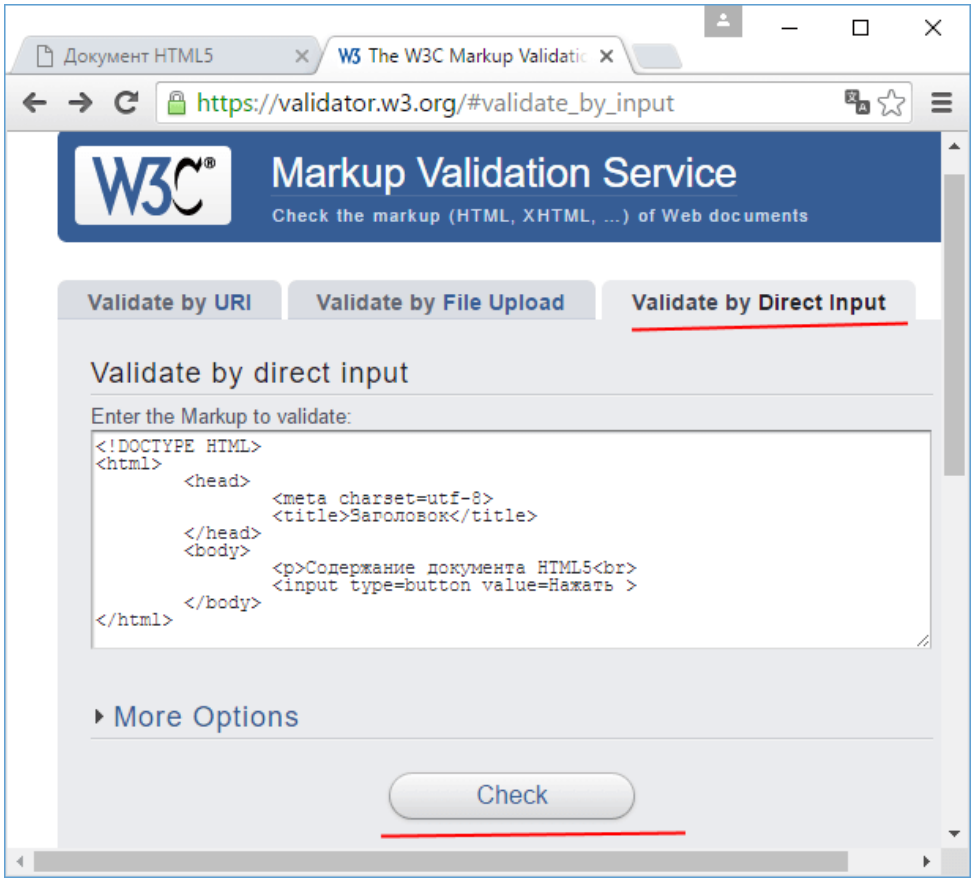
При использовании синтаксиса XHTML нам также надо указать пространство имен для данного документа: `<html xmlns="http://www.w3.org/1999/xhtml">`

Выбор конкретного стиля при написании html-документов зависит от предпочтений программиста или веб-дизайнера. Нередко используется смешенный стиль, который заимствует правила из первого, и из второго стилей.

В то же время надо учитывать, что наличие у элемента закрывающего и открывающего тегов снижает вероятность, что элемент будет неправильно интерпретирован браузером.

Также заключение значений атрибутов в кавычки поможет избежать потенциальных ошибок. Так, атрибут `class` может принимать несколько значений подряд. Например: `<div class="navmenu bigdesctop">`. Но если мы опустим кавычки, то в качестве значения будет использоваться "navmenu", а "bigdesctop" браузер будет пытаться интерпретировать как отдельный атрибут.

Если же возникают затруднения, насколько правильной является создаваемая разметка html, то ее можно проверить с помощью валидатора по адресу https://validator.w3.org:



Мы можем вставить в текстовое поле код веб-страницы, и после нажатия на кнопку "Check" внизу валидатор либо отобразит нам ошибки красным цветом, либо зеленым цветом уведомит, что ошибок нет, и код прошел валидацию.

Элементы группировки

Ряд элементов предназначен для группировки контента на веб-странице.

Элемент div

Элемент **div** служит для структуризации контента на веб-странице, для заключения содержимого в отдельные блоки. Div создает блок, который по умолчанию растягивается по всей ширине браузера, а следующий после div элемент переносится на новую строку. Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Документ HTML5</title>
    </head>
    <body>
        <div>Заголовок документа HTML5</div>
        <div>Текст документа HTML5</div>
    </body>
</html>
```

Параграфы

Параграфы создаются с помощью тегов ** ** и ** **, которые заключают некоторое содержимое. Каждый новый параграф располагается на новой строке. Применим параграфы:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Документ HTML5</title>
    </head>
    <body>
        <div>Заголовок документа HTML5</div>
        <div>
            <p>Первый параграф</p>
            <p>Второй параграф</p>
        </div>
    </body>
</html>
```

Если в рамках одного параграфа нам надо перенести текст на другую строку, то мы можем воспользоваться элементом **
**:

```
<p>Первая строка.<br/>Вторая строка.</p>
```

Элемент pre

Элемент **pre** выводит предварительно отформатированный текст так, как он определен:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Документ HTML5</title>
    </head>
    <body>
        <pre>
            Первая строка
            Вторая строка
            Третья строка
        </pre>
    </body>
</html>
```

Элемент span

Элемент **span** обтекает некоторый текст по всей его длине и служит преимущественно для стилизации заключенного в него текстового содержимого. В отличие от блоков div или параграфов `span` не переносит содержимое на следующую строку:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Документ HTML5</title>
    </head>
    <body>
        <div>Заголовок документа HTML5</div>
        <div>
            <p><span style="color:red;">Первый</span> параграф</p>
            <p><span>Второй</span> параграф</p>
        </div>
    </body>
</html>
```

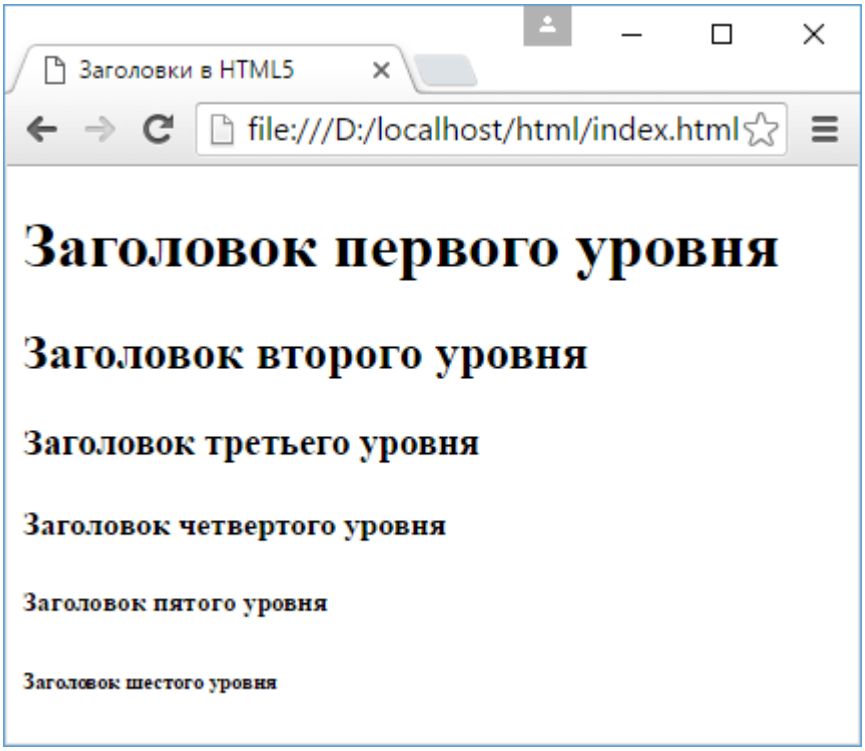
При этом стоит отметить, что сам по себе `span` ничего не делает. Так, во втором параграфе `span` никак не повлиял на внутренне текстовое содержимое. А в первом параграфе элемент `span` содержит атрибут стиля: `style="color:red;"`, который устанавливает для вложенного текста красный цвет фона.

При этом стоит отметить, что элементы `div` и `p` являются блочными, элемент div может содержать любые другие элементы, а элемент p - только строчные элементы. В отличие от них элемент `span` является строчным, то есть как бы встраивает свое содержимое во внешний контейнер - тот же div или параграф. Но при этом не следует помещать блочные элементы в строчный элемент span.

Заголовки

Элементы **<h1>, <h2>, <h3>, <h4>, <h5>** и **<h6>** служат для создания заголовков различного уровня:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Заголовки в HTML5</title>
    </head>
    <body>
        <h1>Заголовок первого уровня</h1>
        <h2>Заголовок второго уровня</h2>
        <h3>Заголовок третьего уровня</h3>
        <h4>Заголовок четвертого уровня</h4>
        <h5>Заголовок пятого уровня</h5>
        <h6>Заголовок шестого уровня</h6>
    </body>
</html>
```



Заголовки выделяют шрифт жирным и по умолчанию имеют некоторый размер: от самого крупного `<h1>` до самого мелкого `<h6>`.

При определении заголовков следует учитывать, что на странице должен быть только один заголовок первого уровня, то есть `<h1>`. Он выполняет роль основного заголовка веб-страницы.

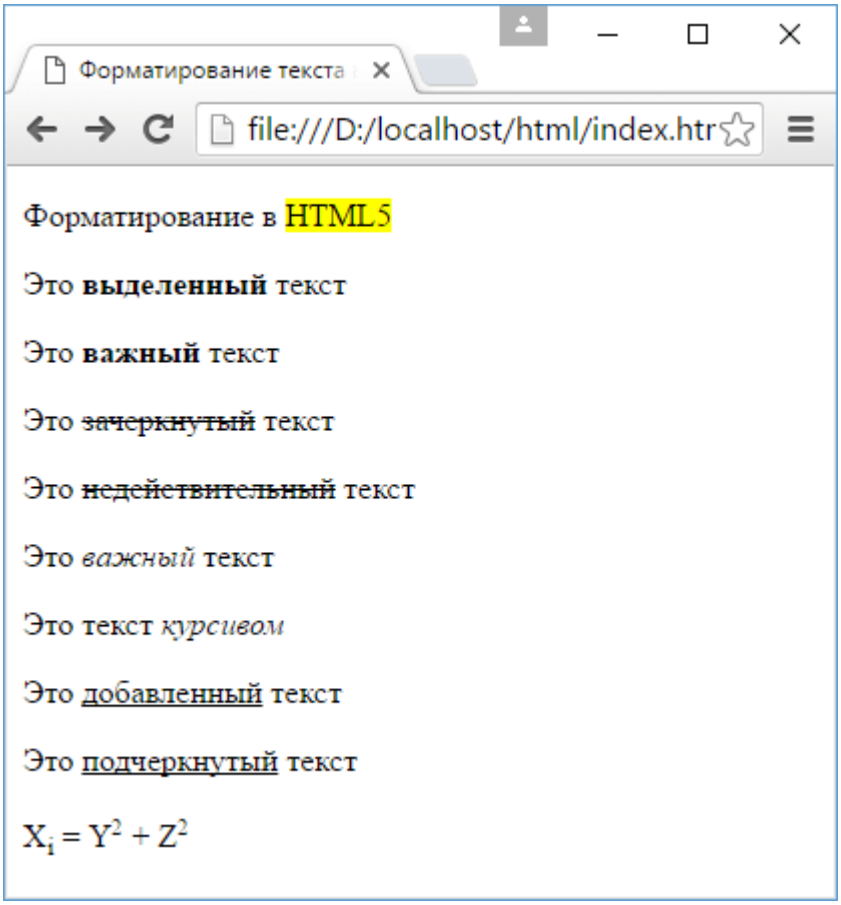
Форматирование текста

Ряд элементов html предназначены для форматирования текстового содержимого, например, для выделения жирным или курсивом и т.д. Рассмотрим эти элементы:

- **<b>**: выделяет текст жирным
- **<del>**: зачеркивает текст
- **<i>**: выделяет текст курсивом
- ****: выделяет текст курсивом, в отличие от тега `<i>` носит логическое значение, придает выделяемому тексту оттенок важности
- **<s>**: зачеркивает текст
- **<small>**: делает текст чуть меньше размером, чем окружающий
- ****: выделяет текст жирным. В отличие от тега `<b>` предназначен для логического выделения, чтобы показать важность текста. А `<b>` не носит характера логического выделения, выполняет функции только форматирования
- **<sub>**: помещает текст под строкой
- **<sup>**: помещает текст над строкой
- **<u>**: подчеркивает текст
- **<ins>**: определяет вставленный (или добавленный) текст
- **<mark>**: выделяет текст цветом, придавая ему оттенок важности

Применим все эти элементы:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Форматирование текста в HTML5</title>
    </head>
    <body>
        <p>Форматирование в <mark>HTML5</mark></p>
        <p>Это <b>выделенный</b> текст</p>
        <p>Это <strong>важный</strong> текст</p>
        <p>Это <del>зачеркнутый</del> текст</p>
        <p>Это <s>недействительный</s> текст</p>
        <p>Это <em>важный</em> текст</p>
        <p>Это текст <i>курсивом</i> </p>
        <p>Это <ins>добавленный</ins> текст</p>
        <p>Это <u>подчеркнутый</u> текст</p>
        <p>X<sub>i</sub> = Y<sup><small>2</small></sup> + Z<sup><small>2</small>
        </sup></p>
    </body>
</html>
```


Форматирование текста
file:///D:/localhost/html/index.htr
Форматирование в HTML5
Это выделенный текст
Это важный текст
Это зачеркнутый текст
Это недействительный текст
Это важный текст
Это текст курсивом
Это добавленный текст
Это подчеркнутый текст
$X_i = Y^2 + Z^2$

Работа с изображениями

Для вывода изображений в HTML используется элемент **img**. Этот элемент представляет нам два важных атрибута:

- **src**: путь к изображению. Это может быть относительный или абсолютный путь в файловой системе или адрес в интернете
- **alt**: текстовое описание изображения. Если браузер по каким-то причинам не может отобразить изображение (например, если у атрибута src некорректно задан путь), то браузер показывает вместо самой картинки данное текстовое описание.

  Атрибут `alt` еще важен тем, что поисковые системы по текстовому описанию могут индексировать изображение.

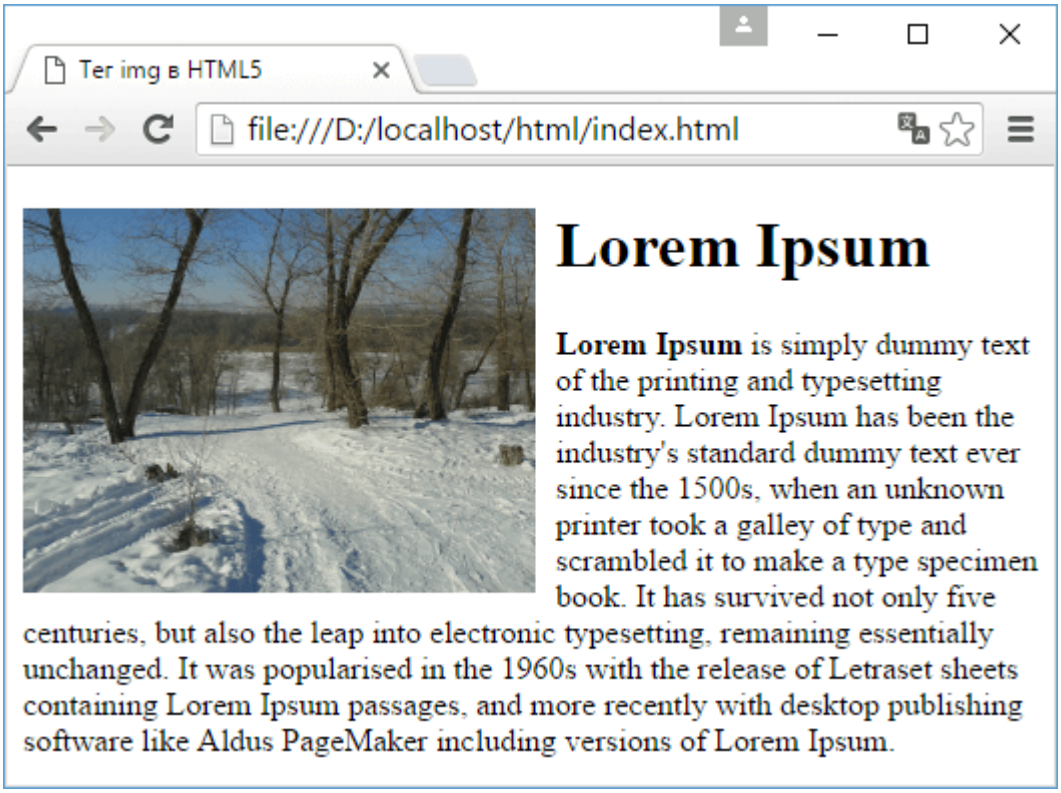
Например, положим в ту же папку, где у нас лежит файл *index.html*, какой-нибудь файл изображения. И затем отобразим его на веб-странице:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Тег img в HTML5</title>
    </head>
    <body>
        <img src="dubi.png" alt="Зимняя равнина" />
    </body>
</html>
```

В моем случае файл изображения называется *dubi.png*, и он находится в одной папке с веб-странице index.html. При этом надо учитывать, что `img` является пустым элементом, то есть не содержим закрывающегося тега.

Используя стилевые особенности, в частности, отступы и обтекание, можно комбинировать изображения с текстом. Например:

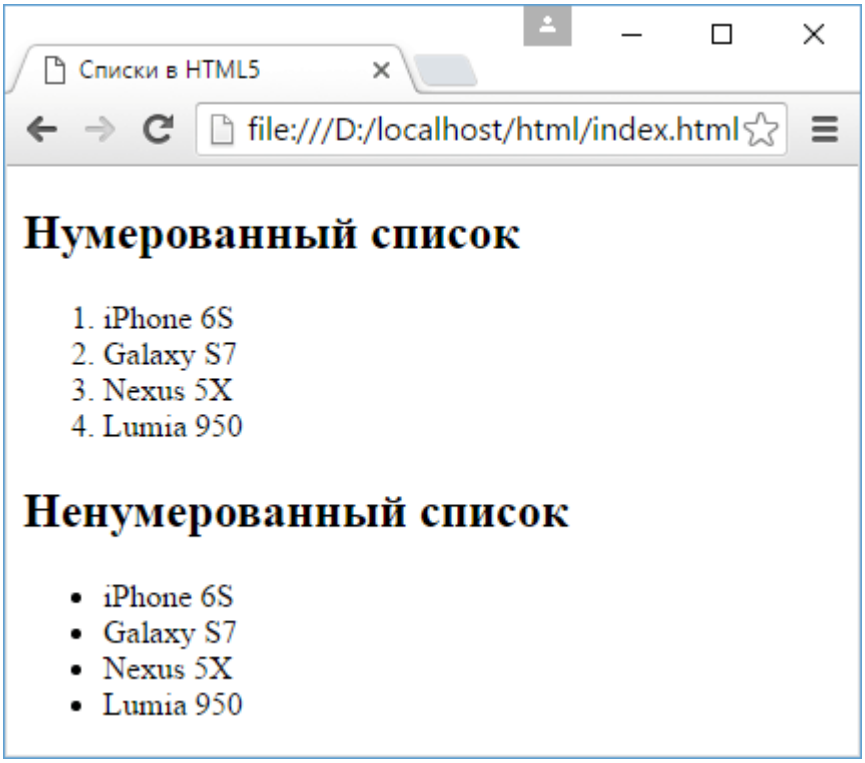
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Тег img в HTML5</title>
    </head>
    <body>
        <div>
            <img src="dubi.png" alt="Зимняя равнина" style="float:left;
             margin-right:10px;" />
            <h1>Lorem Ipsum</h1>
            <b>Lorem Ipsum</b> is simply dummy text of the printing and
               typesetting industry. Lorem Ipsum has been the industry....
        </div>
    </body>
</html>
```


Тег img в HTML5
file:///D:/localhost/html/index.html
Lorem Ipsum
Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Списки

Для создания списков в HTML5 применяются элементы `<ol>` (нумерованный список) и `<ul>` (ненумерованный список):

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Списки в HTML5</title>
    </head>
    <body>
        <h2>Нумерованный список</h2>
        <ol>
            <li>iPhone 6S</li>
            <li>Galaxy S7</li>
            <li>Nexus 5X</li>
            <li>Lumia 950</li>
        </ol>
        <h2>Ненумерованный список</h2>
        <ul>
            <li>iPhone 6S</li>
            <li>Galaxy S7</li>
            <li>Nexus 5X</li>
            <li>Lumia 950</li>
        </ul>
    </body>
</html>
```



В нумерованном списке для нумерации элементов по умолчанию используется стандартные цифры от 1. В ненумерованном списке каждый элемент предваряется черной точкой.

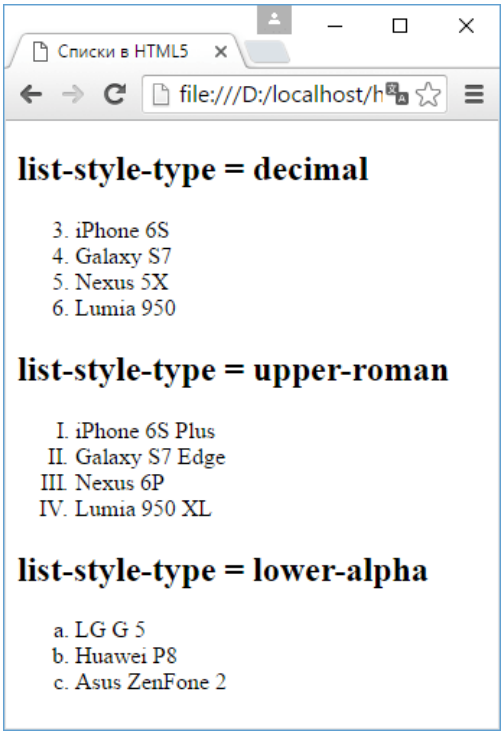
При необходимости мы можем настроить нумерацию или отражаемый рядом с элементом символ с помощью стиля **list-style-type**. Данный стиль может принимать множество различных

значений. Отметим только основные и часто используемые. Для нумерованных списков стиль `list-style-type` может принимать следующие значения:

- `decimal`: десятичные числа, отсчет идет от 1
- `decimal-leading-zero`: десятичные числа, которые предваряются нулем, например, 01, 02, 03, … 98, 99
- `lower-roman`: строчные римские цифры, например, i, ii, iii, iv, v
- `upper-roman`: заглавные римские цифры, например, I, II, III, IV, V…
- `lower-alpha`: строчные римские буквы, например, a, b, c..., z
- `upper-alpha`: заглавные римские буквы, например, A, B, C, … Z

Для нумерованных список с помощью атрибута **start** можно дополнительно задать символ, с которого будет начинаться нумерация. Например:

```
<h2>list-style-type = decimal</h2>
<ol style="list-style-type:decimal;" start="3">
    <li>iPhone 6S</li>
    <li>Galaxy S7</li>
    <li>Nexus 5X</li>
    <li>Lumia 950</li>
</ol>
<h2>list-style-type = upper-roman</h2>
<ul style="list-style-type:upper-roman;">
    <li>iPhone 6S Plus</li>
    <li>Galaxy S7 Edge</li>
    <li>Nexus 6P</li>
    <li>Lumia 950 XL</li>
</ul>
<h2>list-style-type = lower-alpha</h2>
<ul style="list-style-type:lower-alpha;">
    <li>LG G 5</li>
    <li>Huawei P8</li>
    <li>Asus ZenFone 2</li>
</ul>
```

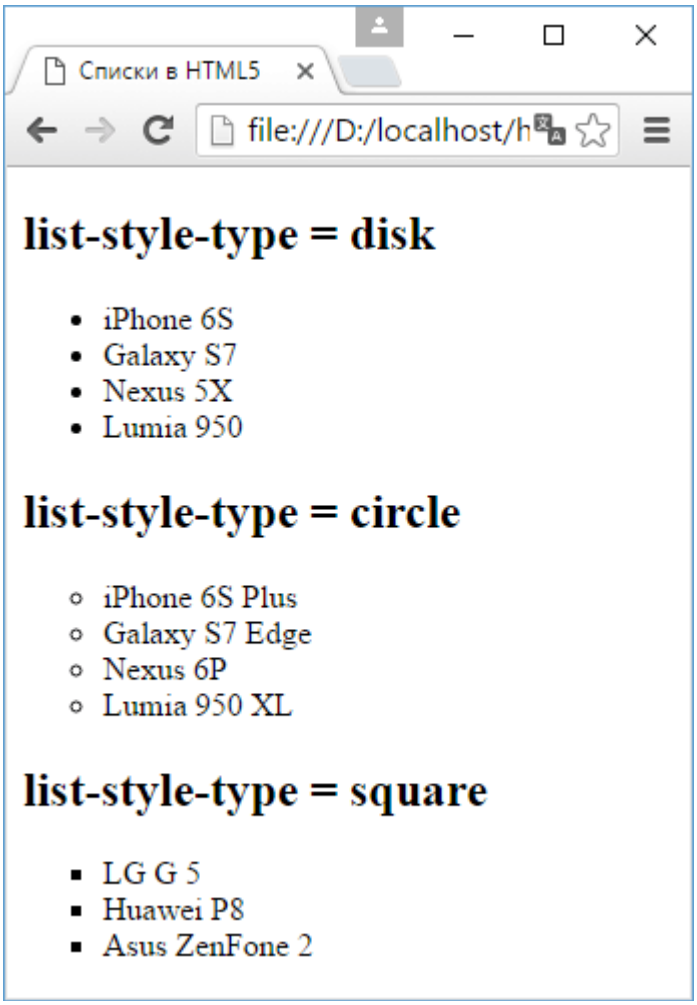


Для ненумерованного списка атрибут `list-style-type` может принимать следующие значения:

- `disc`: черный диск
- `circle`: пустой кружочек
- `square`: черный квадратик

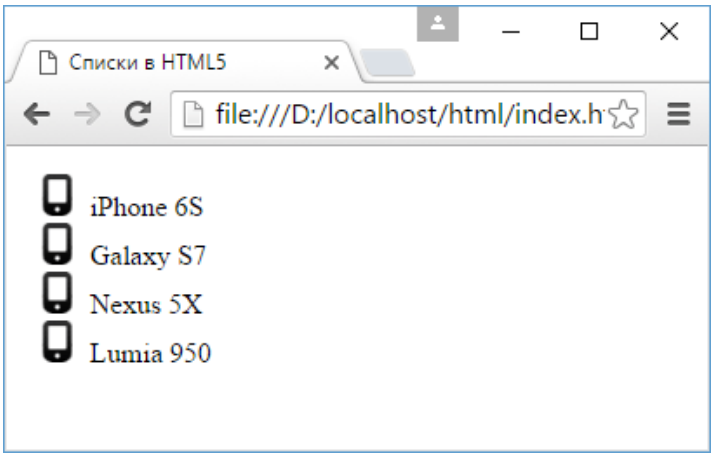
Например:

```
<h2>list-style-type = disc</h2>
<ul style="list-style-type:disc;">
    <li>iPhone 6S</li>
    <li>Galaxy S7</li>
    <li>Nexus 5X</li>
    <li>Lumia 950</li>
</ul>
<h2>list-style-type = circle</h2>
<ul style="list-style-type:circle;">
    <li>iPhone 6S Plus</li>
    <li>Galaxy S7 Edge</li>
    <li>Nexus 6P</li>
    <li>Lumia 950 XL</li>
</ul>
<h2>list-style-type = square</h2>
<ul style="list-style-type:square;">
    <li>LG G 5</li>
    <li>Huawei P8</li>
    <li>Asus ZenFone 2</li>
</ul>
```



Еще одну интересную возможность по настройке списков предоставляет стиль **list-style-image**. Он задает изображение, которое будет отображаться рядом с элементом списка:

```
<ul style="list-style-image:url(phone_touch.png);">
    <li>iPhone 6S</li>
    <li>Galaxy S7</li>
    <li>Nexus 5X</li>
    <li>Lumia 950</li>
</ul>
```



Стиль `list-style-image` в качестве значения принимает `url(phone_touch.png)`, где "phone_touch.png" - это название файла изображения. То есть в данном случае предполагается, что в одной папке с веб-страницей index.html у меня находится файл изображения *phone_touch.png*.

Горизонтальный список

Одним из распространенных способов стилизации списков представляет создание горизонтального списка. Для этого для всех элементов списка надо установить стиль `display:inline`:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Горизонтальный список в HTML5</title>
        <style>
            ul#menu li {
                display:inline;
            }
        </style>
    </head>
    <body>
        <ul id="menu">
            <li>Главная</li>
            <li>Блог</li>
            <li>Форум</li>
            <li>О сайте</li>
        </ul>
    </body>
</html>
```

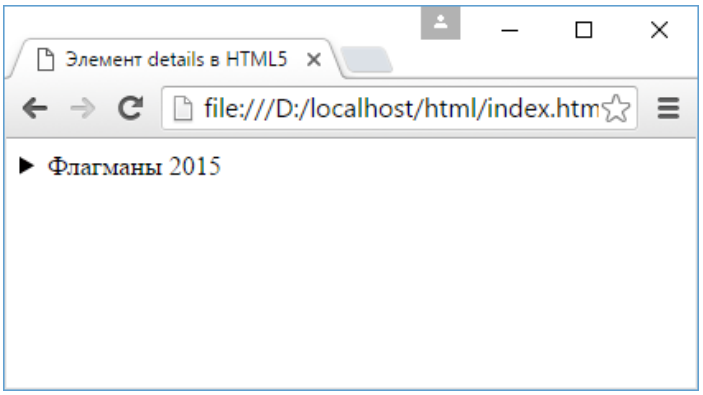
Элемент details

Элемент **details** позволяет создавать раскрываемый блок, который по умолчанию скрыт. Иногда такой элемент еще называют accordion.

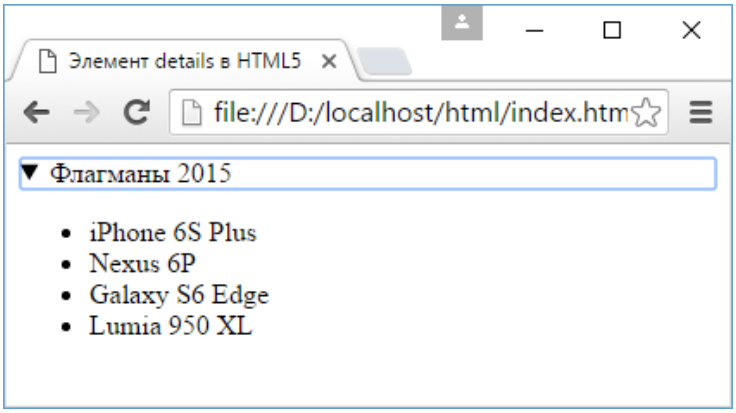
Данный элемент содержит элемент **summary**, который представляет заголовок для блока, и этот заголовок отображается в скрытом режиме. Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Элемент details в HTML5</title>
    </head>
    <body>
        <details>
            <summary>Флагманы 2015</summary>
            <ul>
                <li>iPhone 6S Plus</li>
                <li>Nexus 6P</li>
                <li>Galaxy S6 Edge</li>
                <li>Lumia 950 XL</li>
            </ul>
        </details>
    </body>
</html>
```

По умолчанию мы видим только заголовок summary:



Нажав на стрелку или заголовок, мы можем раскрыть блок:



Дополнительно про данный элемент можно прочитать в статье Стилизация элемента details

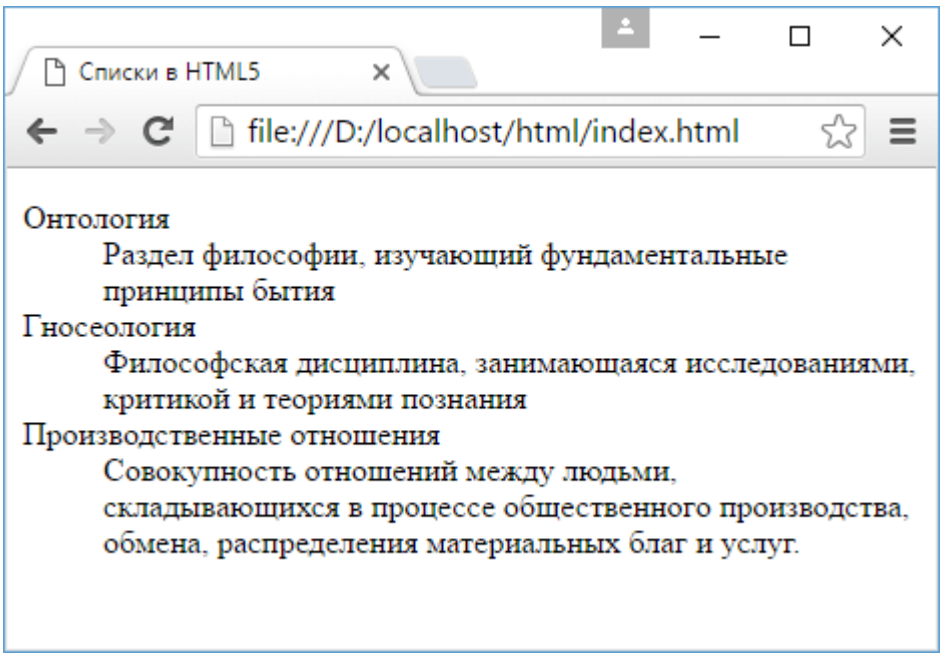
Список определений

Список определений (definition list) представляет такой список, который содержит термин и определение к этому термину. И таких пар термин-определение в списке может быть множество. Для создания списка определений применяются теги **<dl>** и **</dl>**. Внутрь этих тегов помещаются элементы списка.

Каждый элемент списка состоит из термина и определения. Термин помещается в теги **<dt>** и **</dt>** (dt - сокращение от "definition term"), а определение - в теги **<dd>** и **</dd>** (dd - сокращение от "definition description")

Рассмотрим простейший список определений:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Списки в HTML5</title>
    </head>
    <body>
        <dl>
            <dt>Онтология</dt>
            <dd>Раздел философии, изучающий фундаментальные принципы бытия</dd>
            <dt>Гносеология</dt>
            <dd>Философская дисциплина, занимающаяся исследованиями, критикой и
                теориями познания</dd>
            <dt>Производственные отношения</dt>
            <dd>Совокупность отношений между людьми, складывающихся в процессе
                общественного производства,
                обмена, распределения материальных благ и услуг.</dd>
        </dl>
    </body>
</html>
```

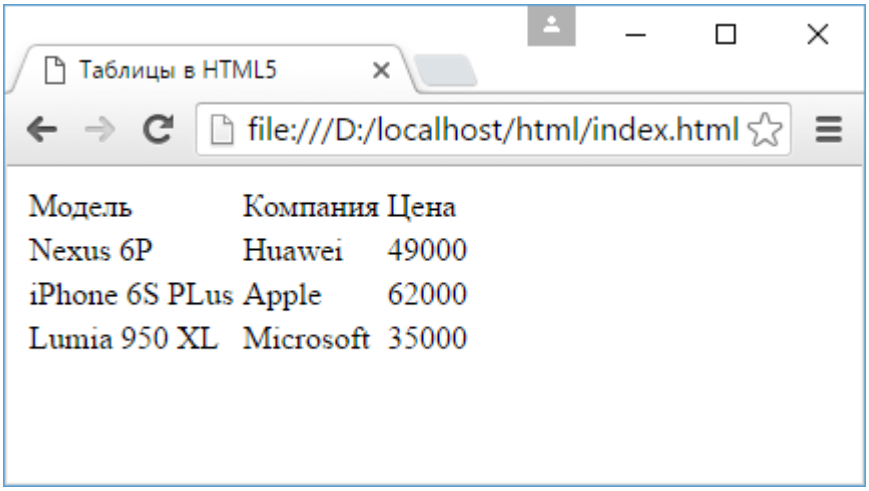


Таблицы

Для создания таблиц в html используется элемент **table**. Каждая таблица между тегами `<table>` и `</table>` содержит строки, который представлены элементом **tr**. А каждая строка между тегами `<tr>` и `</tr>` содержит ячейки в виде элементов **td**.

Создадим простейшую таблицу:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Таблицы в HTML5</title>
    </head>
    <body>
        <table>
            <tr>
                <td>Модель</td> <td>Компания</td> <td>Цена</td>
            </tr>
            <tr>
                <td>Nexus 6P</td> <td>Huawei</td> <td>49000</td>
            </tr>
            <tr>
                <td>iPhone 6S PLus</td> <td>Apple</td> <td>62000</td>
            </tr>
            <tr>
                <td>Lumia 950 XL</td> <td>Microsoft</td> <td>35000</td>
            </tr>
        </table>
    </body>
</html>
```



Здесь у нас в таблице 4 строки, и каждая строка имеет по три столбца.

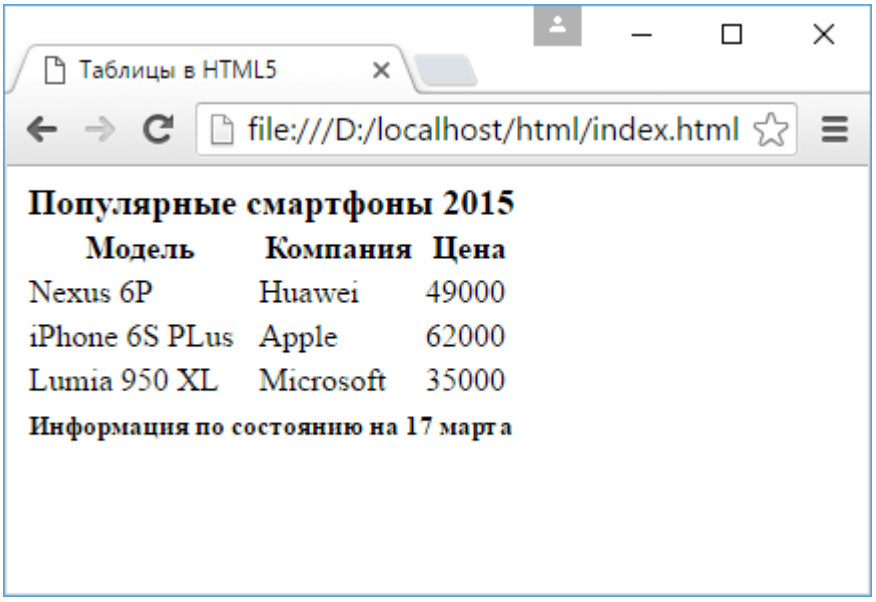
При этом в данном случае первая строка выполняет роль заголовка, а остальные три строки собственно являются содержимым таблицы. Разделения заголовков, футера и тела таблицы в html предусмотрены соответственно элементы **thead**, **tfoot** и **tbody**. Для их применения изменим таблицу следующим образом:

```
<table>
    <caption><b>Популярные смартфоны 2015</b></caption>
    <thead>
        <tr>
            <th>Модель</th> <th>Компания</th> <th>Цена</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Nexus 6P</td> <td>Huawei</td> <td>49000</td>
        </tr>
        <tr>
            <td>iPhone 6S PLus</td> <td>Apple</td> <td>62000</td>
        </tr>
        <tr>
            <td>Lumia 950 XL</td> <td>Microsoft</td> <td>35000</td>
        </tr>
    <tbody>
    <tfoot>
        <tr>
            <th colspan="3">Информация по состоянию на 17 марта</th>
        </tr>
    </tfoot>
</table>
```

В элемент `thead` заключается строка заголовков. Для ячеек заголовок используется не элемент `td`, а `th`. Элемент `th` выделяет заголовок жирным. А все остальные строки заключаются в `tbody`

Элемент `tfoot` определяет подвал таблицы или футер. Здесь обычно выводится некоторая вспомогательная информация по отношению к таблице.

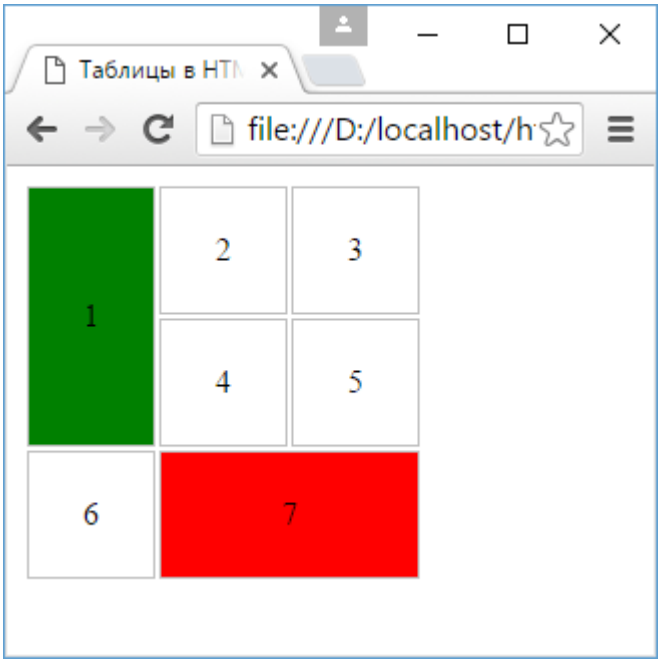
Кроме собственно заголовоков столбцов с помощью элемента **caption** мы можем задать общий заголовок для таблицы.



Также стоит отметить, что футер таблицы содержит только один столбец, который раздвигается по ширине трех столбцов с помощью атрибута `colspan="3"`.

Атрибут **colspan** указывает на какое количество столбцов раздвигается данная ячейка. Также с помощью атрибута **rowspan** мы можем раздвигать ячейку на определенное количество строк. Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Таблицы в HTML5</title>
        <style>
            td{
                width: 60px;
                height:60px;
                border: solid 1px silver;
                text-align:center;
            }
        </style>
    </head>
    <body>
        <table>
            <tr>
                <td rowspan="2" style="background-color:green;">1</td>
                <td>2</td>
                <td>3</td>
            </tr>
            <tr>
                <td>4</td>
                <td>5</td>
            </tr>
            <tr>
                <td>6</td>
                <td colspan="2" style="background-color:red;">7</td>
            </tr>
        </table>
    </body>
</html>
```



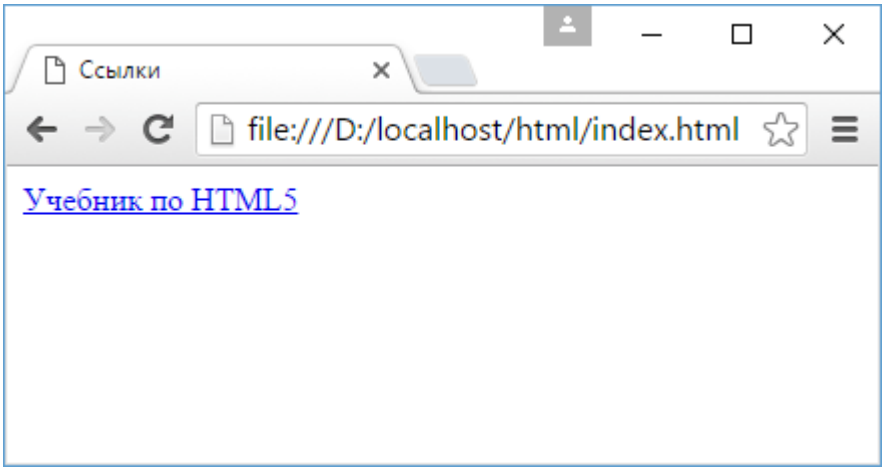
Ссылки

Ссылки, которые представлены элементом **<a>**, играют важную роль - они обеспечивают навигацию между отдельными документами. Этот элемент имеет следующие атрибуты:

- `href`: определяет адрес ссылки
- `hreflang`: указывает на язык документа, на который ведет данная ссылка
- `media`: определяет устройство, для которого предназначена ссылка
- `rel`: определяет отношение между данным документом и ресурсом, на который ведет ссылка
- `target`: определяет, как документ по ссылке должен открываться
- `type`: указывает на mime-тип ресурса по ссылке

Наиболее важным атрибутом является `href`:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Ссылки</title>
    </head>
    <body>
        <a href="content.html">Учебник по HTML5</a>
    </body>
</html>
```



Здесь для ссылки используется относительный путь *content.html*. То есть в одной папке с данным документом должен находиться файл content.html, на который будет идти переход по нажатию на ссылку.

Также мы можем использовать абсолютные пути с полным указанием адреса:

```
<a href="http://metanit.com/web/html5/">Учебник по HTML5</a>
```

Навигация внутри документа

И также мы можем задать внутренние ссылки, которые будут переходить к определенным блокам внутри элементов:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Внутренние ссылки</title>
    </head>
    <body>
        <a href="#paragraph1">Параграф 1</a> | <a href="#paragraph2">Параграф 2</a> | <a
         href="#paragraph3">Параграф 3</a>
        <h2 id="paragraph1">Параграф 1</h2>
        <p>Содержание параграфа 1</p>
        <h2 id="paragraph2">Параграф 2</h2>
        <p>Содержание параграфа 2</p>
        <h2 id="paragraph3">Параграф 3</h2>
        <p>Содержание параграфа 3</p>
    </body>
</html>
```

Чтобы определить внутреннюю ссылку, указывается знак решетки (#), после которого идет id того элемента, к которому надо осуществить переход. В данном случае переход будет идти к заголовкам h2.

Атрибут target

По умолчанию ресурсы, на которые ведут ссылке, открываются в том же окне. С помощью атрибута **target** можно переопределить это действие. Атрибут target может принимать следующие значения:

- `_blank`: открытие html-документа в новом окне или вкладке браузера
- `_self`: открытие html-документа в том же фрейме (или окне)
- `_parent`: открытие документа в родительском фрейме, если ссылка расположена во внутреннем фрейме
- `_top`: открытие html-документа на все окно браузера
- `framename`: открытие html-документа во фрейме, который называется framename (В данном случае framename - только пример, название фрейма может быть произвольным)

Например, открытие документа по ссылке в новом окне:

```
<a href="http://metanit.com/web/html5/" target="_blank">Учебник по HTML5</a>
```

Значение `_blank` как раз и указывает браузеру, что ресурс надо открыть в новой вкладке.

Стилизация ссылок

По умолчанию ссылка уже имеет некоторый цвет (один из оттенков синего), кроме того она имеет подчеркивание. При нажатии на ссылку она становится активной и приобретает красный цвет, а после перехода по ссылке эта ссылка может окраситься в другой цвет (как правило, в фиолетовый). Подобная стилизация задается многими браузерами по умолчанию, но мы можем ее переопределить. Например:

```
<!DOCTYPE html>
<html>
    <head>
```

```
        <meta charset="utf-8">
        <title>Ссылки</title>
        <style>
            a:link    {color:blue; text-decoration:none}
            a:visited {color:pink; text-decoration:none}
            a:hover   {color:red; text-decoration:underline}
            a:active  {color:yellow; text-decoration:underline}
        </style>
    </head>
    <body>
        <a href="index.html">Учебник по HTML5</a>
    </body>
</html>
```

Здесь определены стили для ссылок в различных состояниях. `a:link` применяется для ссылок в обычном состоянии, когда они не нажаты и на них не наведен указатель мыши.

`a:visited` указывает на состояние ссылки, по которой уже был осуществлен переход.

`a:hover` указывает на состояние ссылки, на которую навели указатель мыши.

`a:active` указывает на ссылку в нажатом состоянии.

Стиль `color` устанавливает цвет ссылки. А стиль `text-decoration` устанавливает подчеркивание: если значение `underline`, то ссылка поддчеркнута, если `none`, то подчеркивание отсутствует.

Ссылка-картинка

Поместив внутрь элемента `<a>` элемент `<img>`, можно сделать ссылку-изображение:

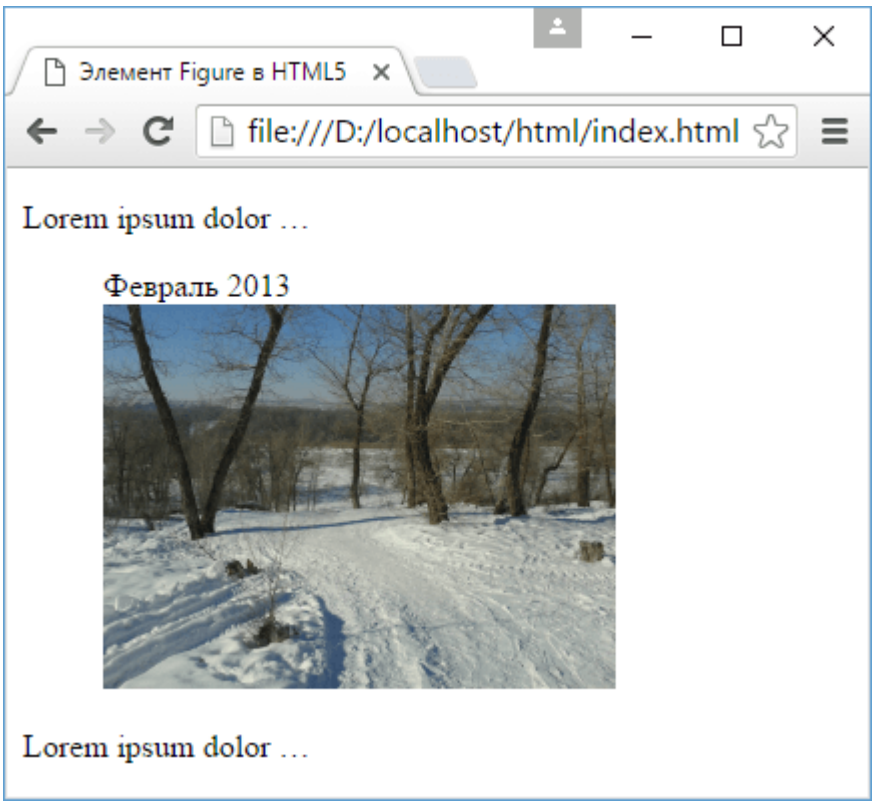
```
<a href="index.html">
    <img src="cover.png" alt="HTML tutorial">
</a>
```

Элементы figure и figcaption

Элемент **figure** применяется для аннотации различных иллюстраций, диаграмм, фотографий и т.д. А элемент **figcaption** просто обертывает заголовок для содержимого внутри элемента figure.

Для использования элемента figure нам надо поместить в него некоторое содержимое, например, изображение:

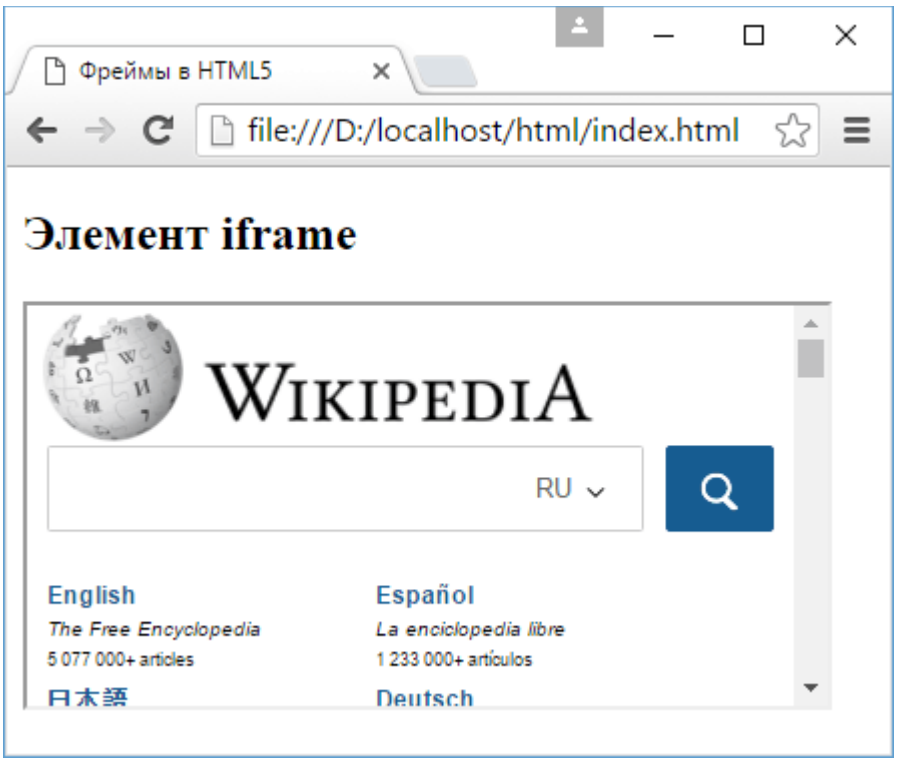
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Элемент Figure в HTML5</title>
    </head>
    <body>
        <div>
            <p>Lorem ipsum dolor … </p>
            <figure>
                <figcaption>Февраль 2013</figcaption>
                <img src="dubi.png" alt="Зимний пейзаж">
            </figure>
            <p>Lorem ipsum dolor … </p>
        </div>
    </body>
</html>
```



Фреймы

Фреймы позволяют встраивать на веб-страницу еще какую-нибудь другую веб-страницу. Фреймы представлены элементом **iframe**. Допустим, нам надо встроить на веб-страницу стартовую страницу википедии:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Фреймы в HTML5</title>
    </head>
    <body>
        <h2>Элемент iframe</h2>
        <iframe src="http://wikipedia.com" width="400" height="200">
        </iframe>
    </body>
</html>
```



Элемент `iframe` не содержит в себе никакого содержимого. Вся его настройка производится с помощью атрибутов:

- `src`: устанавливает полный путь к загружаемому ресурсу
- `width`: ширина фрейма
- `height`: высота фрейма

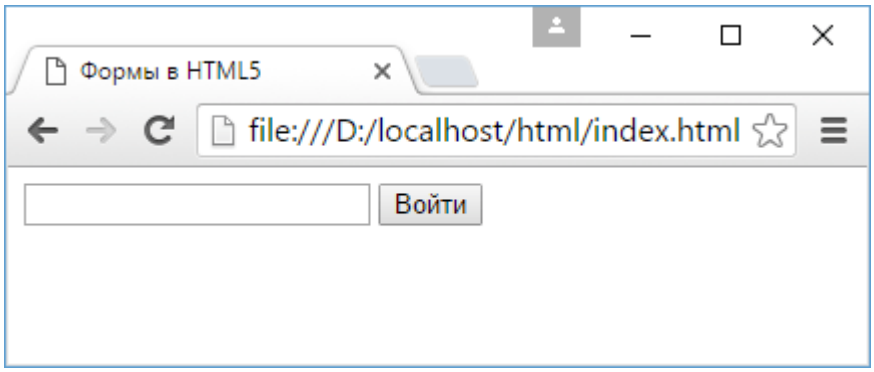
Надо отметить, что не все сайты могут открываться во фреймах, поскольку на стороне веб-сервера могут действовать ограничения на открытие во фреймах.

Работа с формами

Формы

Формы в html представляют один из способов для ввода и отправки данных. Все поля формы помещаются между тегами **<form>** и **</form>**. Например, создадим простейшую форму:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Формы в HTML5</title>
    </head>
    <body>
        <form method="post" action="http://localhost:8080/login.php">
            <input name="login"/>
            <input type="submit" value="Войти" />
        </form>
    </body>
</html>
```



Для настройки форм у элемента form определены следующие атрибуты:

- `method`: устанавливает метод отправки данных на сервер. Допустимы два значения: `post` и `get`.

  Значение `post` позволяет передать данные на веб-сервер через специальные заголовки. А значение `get` позволяет передать данные через строку запроса.

- `action`: устанавливает адрес, на который передаются данные формы
- `enctype`: устанавливает тип передаваемых данных. Он свою очередь может принимать следующие значения:
  - `application/x-www-form-urlencoded`: кодировка отправляемых данных по умолчанию
  - `multipart/form-data`: эта кодировка применяется при отправке файлов
  - `text/plain`: эта кодировка применяется при отправке простой текстовой информации

В выше использованном примере:

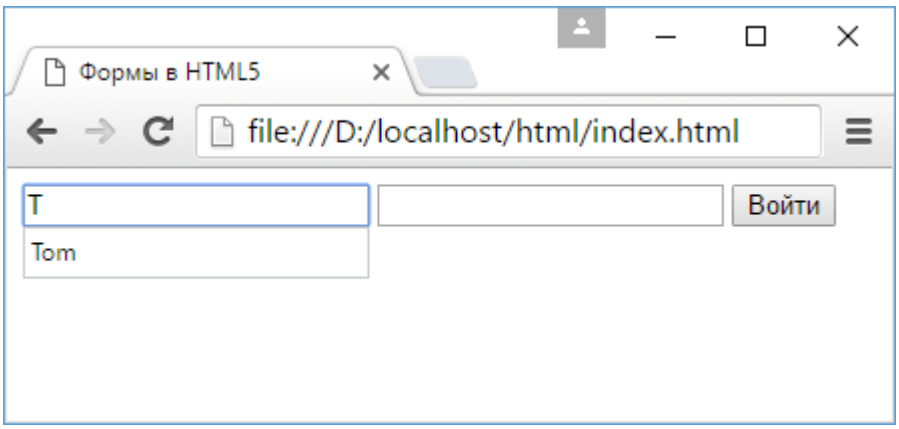
```
<form method="post" action="http://localhost:8080/login.php">
    <input name="login"/>
    <input type="submit" value="Войти" />
</form>
```

у формы установлен метод "post", то есть все значения формы отправляются в теле запроса, а адресом служит строка *http://localhost:8080/login.php*. Адрес здесь указан случайным образом.

Как правило, по указанному адресу работает веб-сервер, который, используя одну из технологий серверной стороны (PHP, NodeJS, ASP.NET и т.д.), может получать запросы и возвращать ответ. В данном же случае мы не будем акцентировать внимание на технологиях серверной стороны, сосредоточимся лишь на тех средствах HTML, которые позволяют отправлять данные на сервер.

Автодополнение

Часто веб-браузеры запоминают вводимые данные, и при вводе браузеры могут выдавать список подсказок из ранее введенных слов:



Это может быть не всегда удобно, и с помощью атрибута **autocomplete** можно отключить автодополнение:

```
<form method="post" autocomplete="off" action="http://localhost:8080/login.php">
    <input name="login" />
    <input name="password" />
    <input type="submit" value="Войти" />
</form>
```

Если нам надо включить автодополнение только для каких-то определенных полей, то мы можем применить к ним атрибут `autocomplete="on"`:

```
<form method="post" autocomplete="off" action="http://localhost:8080/login.php">
    <input name="login" />
    <input name="password" autocomplete="on" />
    <input type="submit" value="Войти" />
</form>
```

Теперь для всей формы, кроме второго поля, будет отключено автодополнение.

Элементы форм

Формы состоят из определенного количества элементов ввода. Все элементы ввода помещаются между тегами `<form>` и `</form>`

Наиболее распространенным элементом ввода является элемент **input**. Однако реальное действие этого элемента зависит от того, какое значение установлено у его атрибута **type**. А он может принимать следующие значения:

- **text**: обычное текстовое поле
- **password**: тоже текстовое поле, только вместо вводимых символов отображаются звездочки, поэтому в основном используется для ввода пароля
- **radio**: радиокнопка или переключатель. Из группы радиокнопок можно выбрать только одну
- **checkbox**: элемент флажок, который может находиться в отмеченном или неотмеченном состоянии
- **hidden**: скрытое поле
- **submit**: кнопка отправки формы
- **color**: поле для ввода цвета
- **date**: поле для ввода даты
- **datetime**: поле для ввода даты и времени с учетом часового пояса
- **datetime-local**: поле для ввода даты и времени без учета часового пояса
- **email**: поле для ввода адреса электронной почты
- **month**: поле для ввода года и месяца
- **number**: поле для ввода чисел
- **range**: ползунок для выбора числа из некоторого диапазона
- **tel**: поле для ввода телефона
- **time**: поле для ввода времени
- **week**: поле для ввода года и недели
- **url**: поле для ввода адреса url
- **file**: поле для выбора отправляемого файла
- **image**: создает кнопку в виде картинки

Кроме элемента input в различных модификациях есть еще небольшой набор элементов, которые также можно использовать на форме:

- **button**: создает кнопку
- **select**: выпадающий список
- **label**: создает метку, которая отображается рядом с полем ввода
- **textarea**: многострочное текстовое поле

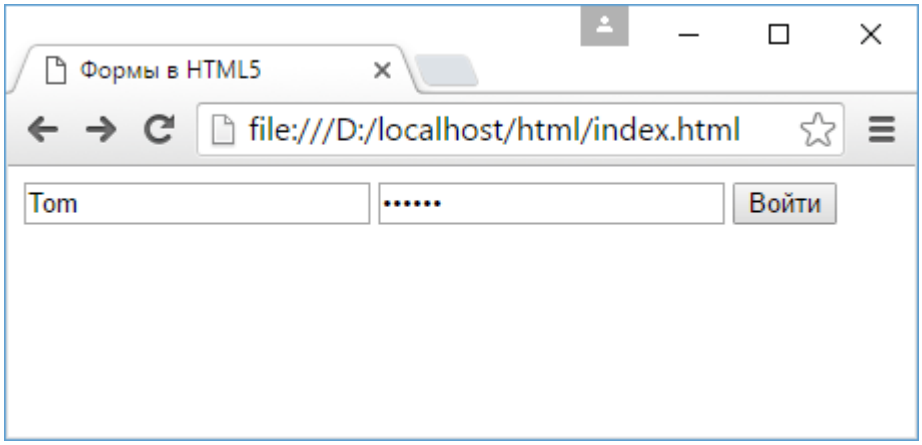
Атрибуты name и value

У всех элементов ввода можно установить атрибуты `name` и `value`. Эти атрибуты имеют важное значение. По атрибуту `name` мы можем идентифицировать поле ввода, а атрибут `value` позволяет установить значение поля ввода. Например:

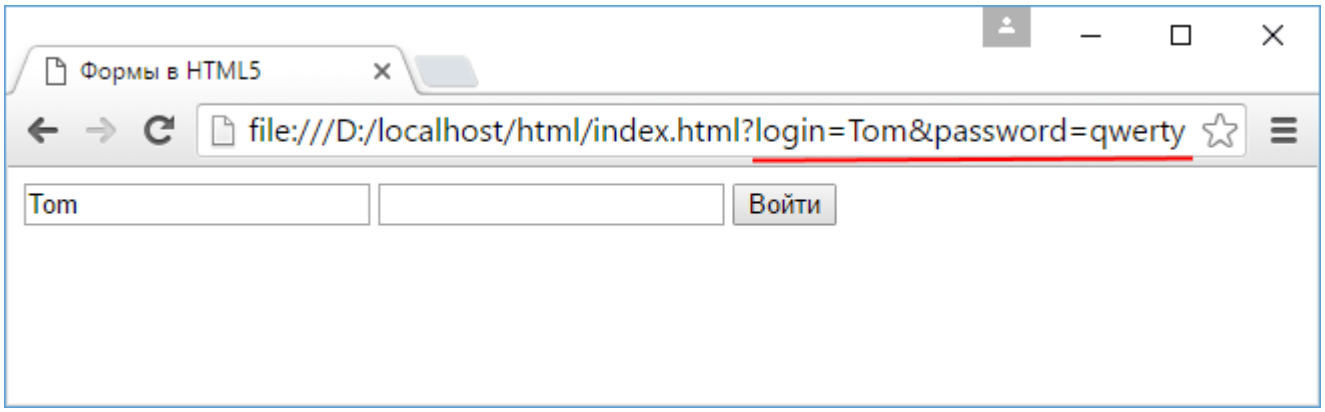
```
1   <!DOCTYPE html>
2   <html>
3       <head>
4           <meta charset="utf-8">
5           <title>Формы в HTML5</title>
```

```
    </head>
    <body>
        <form method="get" action="index.html">
            <input type="text" name="login" value="Tom"/>
            <input type="password" name="password"/>
            <input type="submit" value="Войти" />
        </form>
    </body>
</html>
```

Здесь текстовое поле имеет значение "Tom" (как указано в атрибуте value), поэтому при загрузке веб-страницы в этом поле мы увидим данный текст.



Поскольку методом отправки данных формы является метод "get", то данные будут отправляться через строку запроса. Так как нам в данном случае не важно, как данные будут приниматься, не важен сервер, который получает данные, поэтому в качестве адреса я установил ту же самую страницу - то есть файл index.html. И при отправке мы сможем увидеть введенные данные в строке запроса:



В строке запроса нас интересует следующий кусочек:

```
login=Tom&password=qwerty
```

При отправке формы браузер соединяет все данные в набор пар "ключ-значение". В нашем случае две таких пары: `login=Tom` и `password=qwerty`. Ключом в этих парах выступает название поля ввода, которое определяется атрибутом `name`, а значением - собственно то значение, которое введено в поле ввода (или значение атрибута `value`).

Получив эти данные, сервер легко может узнать, какие значения в какие поля ввода были введены пользователем.

Кнопки

Кнопки представлены элементом **button**. Они обладают широкими возможностями по конфигурации. Так, в зависимости от значения атрибута `type` мы можем создать различные типы кнопок:

- `submit`: кнопка, используемая для отправки формы
- `reset`: кнопка сброса значений формы
- `button`: кнопка без какого-либо специального назначения

Если кнопка используется для отправки формы, то есть у нее установлен атрибут `type="submit"`, то мы можем задать у нее ряд дополнительных атрибутов:

- `form`: определяет форму, за которой закреплена кнопка отправки
- `formaction`: устанавливает адрес, на который отправляется форма. Если у элемента form задан атрибут `action`, то он переопределяется
- `formenctype`: устанавливает формат отправки данных. Если у элемента form установлен атрибут `enctype`, то он переопределяется
- `formmethod`: устанавливает метод отправки формы (post или get). Если у элемента form установлен атрибут `method`, то он переопределяется

Например, определим на форме кнопку отправки и кнопку сброса:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Формы в HTML5</title>
    </head>
    <body>
        <form>
            <p><input type="text" name="login"/></p>
            <p><input type="password" name="password"/></p>
            <p>
                <button type="submit" formmethod="get" formaction="index.html">
                 Отправить</button>
                <button type="reset">Отмена</button>
            </p>
        </form>
    </body>
</html>
```

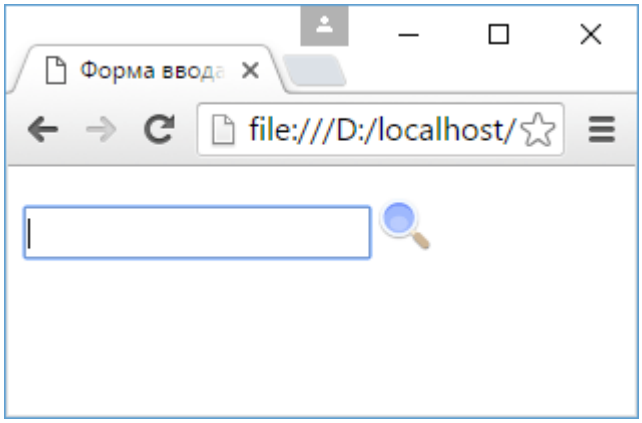
Кроме элемента `button` для создания кнопок можно использовать элемент `input`, у которого атрибут равен `submit` или `reset`. Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Формы в HTML5</title>
    </head>
    <body>
```

```
        <form>
            <p><input type="text" name="login"/></p>
            <p><input type="password" name="password"/></p>
            <p>
                <input type="submit" value="Отправить" />
                <input type="reset" value="Отмена" />
            </p>
        </form>
    </body>
</html>
```

И еще один элемент input с атрибутом `type="image"` позволяет использовать в качестве кнопки изображение:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Форма ввода в HTML5</title>
    </head>
    <body>
        <form>
            <p>
                <input type="text" name="search" />
                <input type="image" src="search.png" name="submit" />
            </p>
        </form>
    </body>
</html>
```



Кроме наличия изображения в остальном эта кнопка будет аналогична стандартной кнопке отправки `input type="submit"` или `button type="submit"`.

Текстовые поля

Однострочное текстовое поле создается с помощью элемента **input**, когда его атрибут `type` имеет значение `text`:

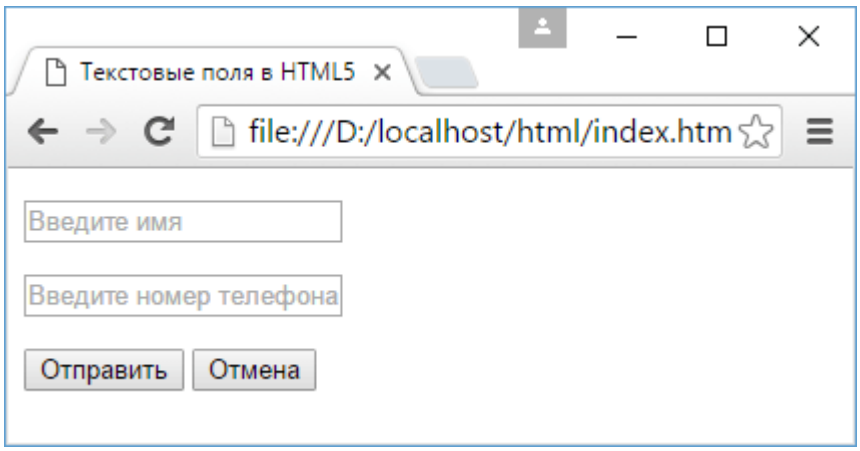
```
<input type="text" name="login" />
```

С помощью ряда дополнительных атрибутов можно настроить текстовое поле:

- **dirname**: устанавливает направление текста
- **maxlength**: максимально допустимое количество символов в текстовом поле
- **pattern**: определяет шаблон, которому должен соответствовать вводимый текст
- **placeholder**: устанавливает текст, который по умолчанию отображается в текстовом поле
- **readonly**: делает текстовом поле доступным только для чтения
- **required**: указывает, что текстовое поле обязательно должно иметь значение
- **size**: устанавливает ширину текстового поля в видимых символах
- **value**: устанавливает значение по умолчанию в текстовом поле

Применим некоторые атрибуты:

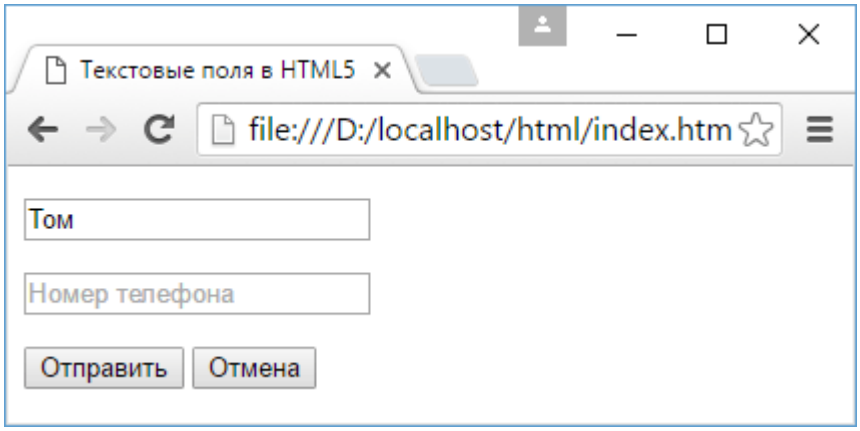
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Текстовые поля в HTML5</title>
    </head>
    <body>
        <form>
            <p><input type="text" name="userName" placeholder="Введите имя" size="18" />
            </p>
            <p><input type="text" name="userPhone" placeholder="Введите номер телефона"
            size="18" maxlength="11" /></p>
            <p>
                <button type="submit">Отправить</button>
                <button type="reset">Отмена</button>
            </p>
        </form>
    </body>
</html>
```



В этом примере во втором текстовом поле сразу устанавливаются атрибуты `maxlength` и `size`. При этом size - то есть количество символов, которые помещаются в видимое пространство поля больше, чем допустимое количество символов. Однако все равно ввести символов больше, чем maxlength, мы не сможем.

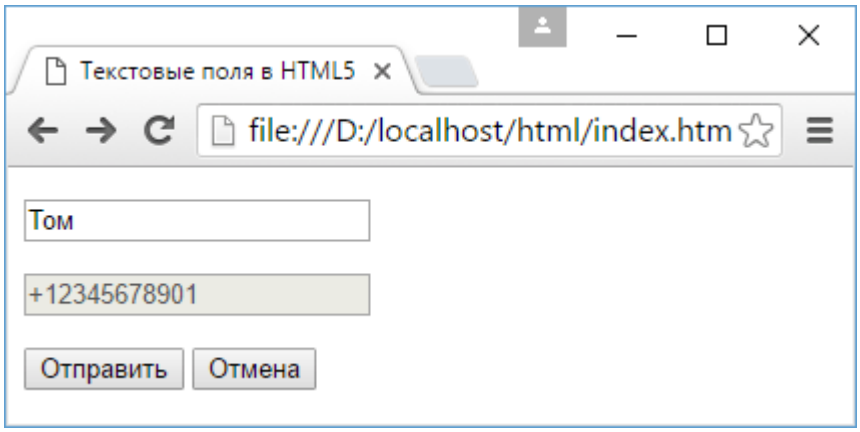
В данном случае также важно различать атрибуты `value` и `placeholder`, хотя оба устанавливают видимый текст в поле. Однако `placeholder` устанавливает своего рода подсказку или приглашение к вводу, поэтому он обычно отмечается серым цветом. В то время как значение `value` представляет введенный в поле текст по умолчанию:

```
<p><input type="text" name="userName" value="Том" /></p>
<p><input type="text" name="userPhone" placeholder="Номер телефона" /></p>
```



Атрибуты `readonly` и `disabled` делают текстовое поле недоступным, однако сопровождаются разным визуальным эффектом. В случае с disabled текстовое поле затеняется:

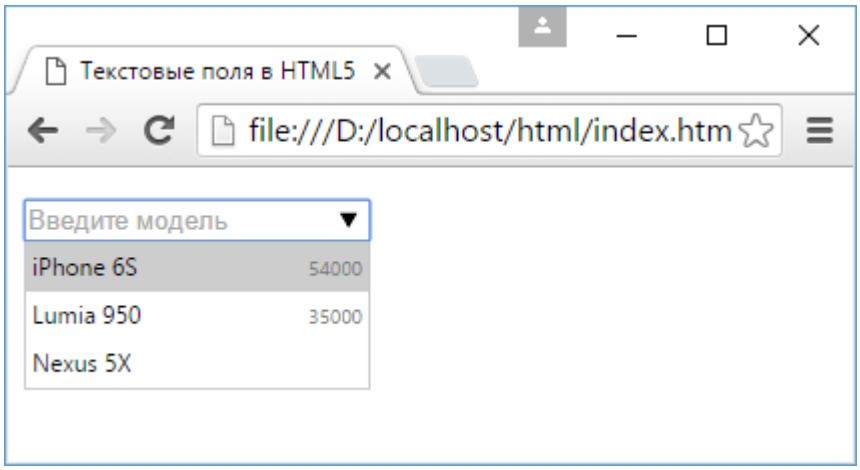
```
<p><input type="text" name="userName" value="Том" readonly /></p>
<p><input type="text" name="userPhone" value="+12345678901" disabled /></p>
```



Среди атрибутов текстового поля также следует отметить такой атрибут как **list**. Он содержит ссылку на элемент **datalist**, который определяет набор значений, появляющихся в виде подсказки при вводе в текстовое поле. Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Текстовые поля в HTML5</title>
    </head>
    <body>
```

```
        <form>
            <p><input list="phonesList" type="text" name="model" placeholder="Введите
               модель" /></p>
            <p>
                <button type="submit">Отправить</button>
            </p>
        </form>
        <datalist id="phonesList">
            <option value="iPhone 6S" label="54000"/>
            <option value="Lumia 950">35000</option>
            <option value="Nexus 5X"/>
        </datalist>
    </body>
</html>
```

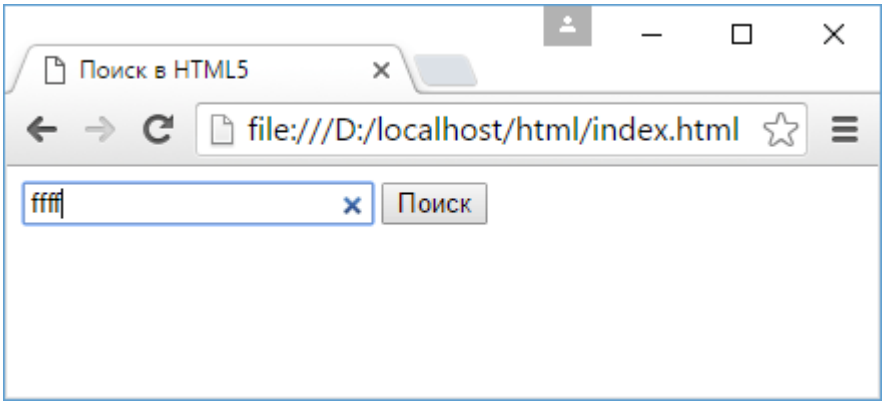


Атрибут `list` текстового поля указывает на id элемента datalist. Сам элемент datalist с помощью вложенных элементов `option` определяет элементы списка. И при вводе в текстовое поле этот список отображается в виде подсказки.

Поле поиска

Для создания полей поиска предназначен элемент input с атрибутом `type="search"`. Формально он представляет собой простое текстовое поле:

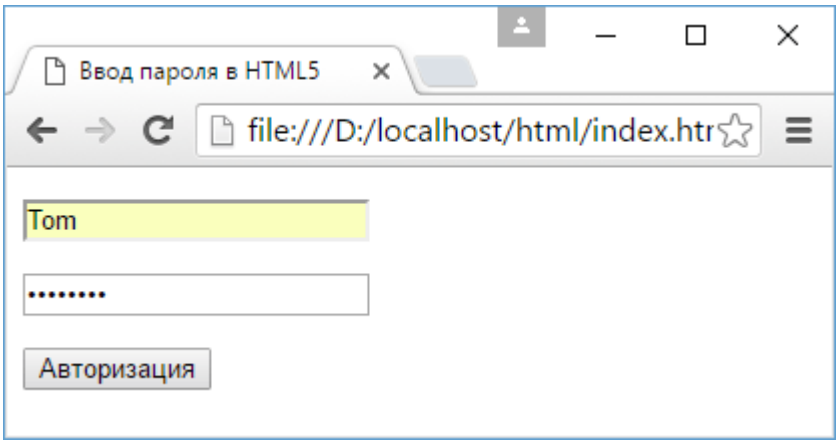
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Поиск в HTML5</title>
    </head>
    <body>
    <form>
        <input type="search" name="term" />
        <input type="submit" value="Поиск" />
    </form>
    </body>
</html>
```



Поле ввода пароля

Для ввода пароля используется элемент input с атрибутом `type="password"`. Его отличительной чертой является то, что вводимые символы маскируются точками:

```
<form>
    <p><input type="text" name="login" /></p>
    <p><input type="password" name="password" /></p>
    <input type="submit" value="Авторизация" />
</form>
```



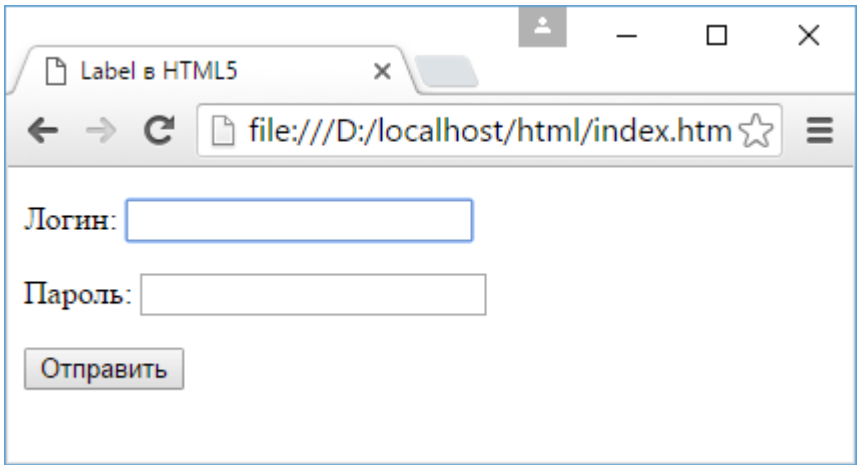
Метки и автофокус

Вместе с полями ввода нередко используются метки, которые представлены элементом **label**. Метки создают аннотацию или заголовок к полю ввода, указывают, для чего это поле предназначено.

Для связи с полем ввода метка имеет атрибут **for**, который указывает на id поля ввода:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Label в HTML5</title>
    </head>
    <body>
        <form>
            <p>
                <label for="login">Логин: </label>
                <input type="text" id="login" name="login" />
            </p>
            <p>
                <label for="password">Пароль: </label>
                <input type="password" id="password" name="password" />
            </p>
            <p>
                <button type="submit">Отправить</button>
            </p>
        </form>
    </body>
</html>
```

Так, текстовое поле здесь имеет атрибут `id="login"`. Поэтому у связанной с ним метки устанавливается атрибут `for="login"`. Нажатие на эту метку позволяет перевести фокус на текстовое поле для ввода логина:



Собственно на этом роль меток и заканчивается. Также мы можем установить автофокус по умолчанию на какое-либо поле ввода. Для этого применяется атрибут **autofocus**:

```
<form>
    <p>
        <label for="login">Логин: </label>
        <input type="text" autofocus id="login" name="login" />
    </p>
    <p>
        <label for="password">Пароль: </label>
        <input type="password" id="password" name="password" />
    </p>
    <p>
        <button type="submit">Отправить</button>
    </p>
</form>
```

Здесь при запуске страницы фокус сразу же переходит на текстовое поле.

Элементы для ввода чисел

Простое числовое поле

Для ввода чисел используется элемент `input` с атрибутом **type="number"**. Он создает числовое поле, которое мы можем настроить с помощью следующих атрибутов:

- `min`: минимально допустимое значение
- `max`: максимально допустимое значение
- `readonly`: доступно только для чтения
- `required`: указывает, что данное поле обязательно должно иметь значение
- `step`: значение, на которое будет увеличиваться число в поле
- `value`: значение по умолчанию

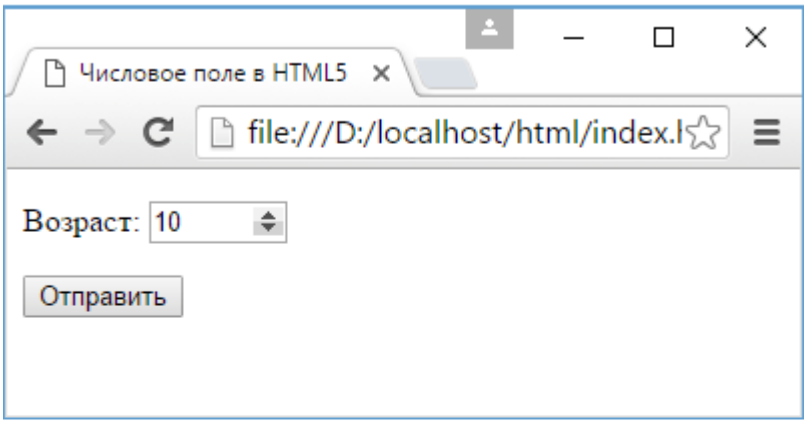
Используем числовое поле:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Числовое поле в HTML5</title>
    </head>
    <body>
        <form>
            <p>
                <label for="age">Возраст: </label>
                <input type="number" step="1" min="1" max="100" value="10" id="age"
                 name="age"/>
            </p>
            <p>
                <button type="submit">Отправить</button>
            </p>
        </form>
    </body>
</html>
```

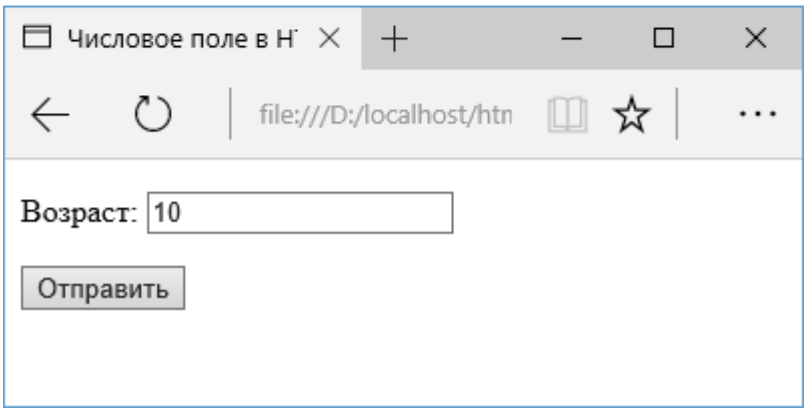
Здесь числовое поле по умолчанию имеет значение 10 (`value="10"`), минимально допустимое значение, которое мы можем ввести, - 1, а максимальное допустимое значение - 100. И атрибут `step="1"` устанавливает, что значение будет увеличиваться на единицу.

В зависимости от браузера визуализация этого поля может отличаться:

Google Chrome



Microsoft Edge

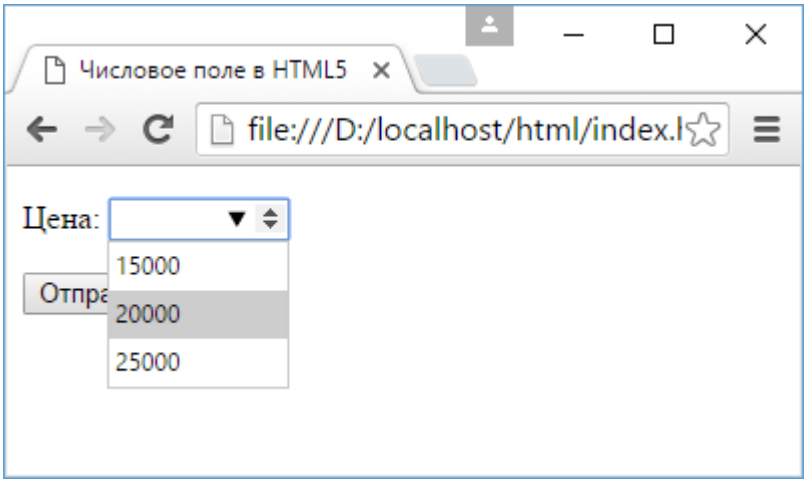


Но как правило, у большинства современных браузеров, кроме IE 11 и Microsoft Edge, справа в поле ввода имеются стрелки для увеличения/уменьшения значения на величину, указанную в атрибуте step.

Как и в случае с текстовым полем мы можем здесь прикрепить список datalist с диапазоном возможных значений:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Числовое поле в HTML5</title>
    </head>
    <body>
        <form>
            <p>
                <label for="price">Цена: </label>
                <input type="number" list="priceList"
                    step="1000" min="3000" max="100000" value="10000" id="price"
                    name="price"/>
            </p>
            <p>
                <button type="submit">Отправить</button>
            </p>
        </form>
        <datalist id="priceList">
            <option value="15000" />
            <option value="20000" />
```

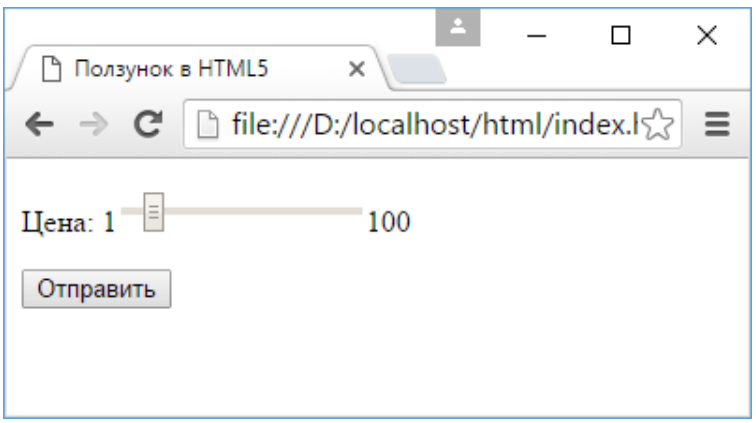
```
            <option value="25000" />
        </datalist>
    </body>
</html>
```



Ползунок

Ползунок представляет шкалу, на которой мы можем выбрать одно из значений. Для создания ползунка применяется элемент `input` с атрибутом **type="range"**. Во многом ползунок похож на простое поле для ввода чисел. Он также имеет атрибуты `min`, `max`, `step` и `value`:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Ползунок в HTML5</title>
    </head>
    <body>
        <form>
            <p>
                <label for="price">Цена:</label>
                1<input type="range" step="1" min="0" max="100" value="10" id="price"
                 name="price"/>100
            </p>
            <p>
                <button type="submit">Отправить</button>
            </p>
        </form>
    </body>
</html>
```

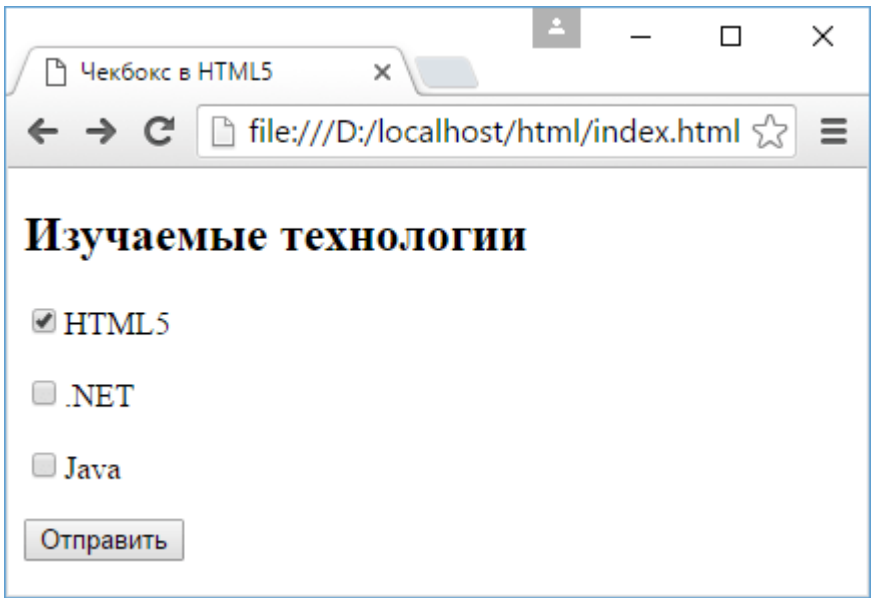


Флажки и переключатели

Флажок

Флажок представляет элемент, который может находиться в двух состояниях: отмеченном и неотмеченном. Флажок создается с помощью элемента `input` с атрибутом **type="checkbox"**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Чекбокс в HTML5</title>
    </head>
    <body>
        <h2>Изучаемые технологии</h2>
        <form>
            <p>
                <input type="checkbox" checked name="html5"/>HTML5
            </p>
            <p>
                <input type="checkbox" name="dotnet"/>.NET
            </p>
            <p>
                <input type="checkbox" name="java"/>Java
            </p>
            <p>
                <button type="submit">Отправить</button>
            </p>
        </form>
    </body>
</html>
```



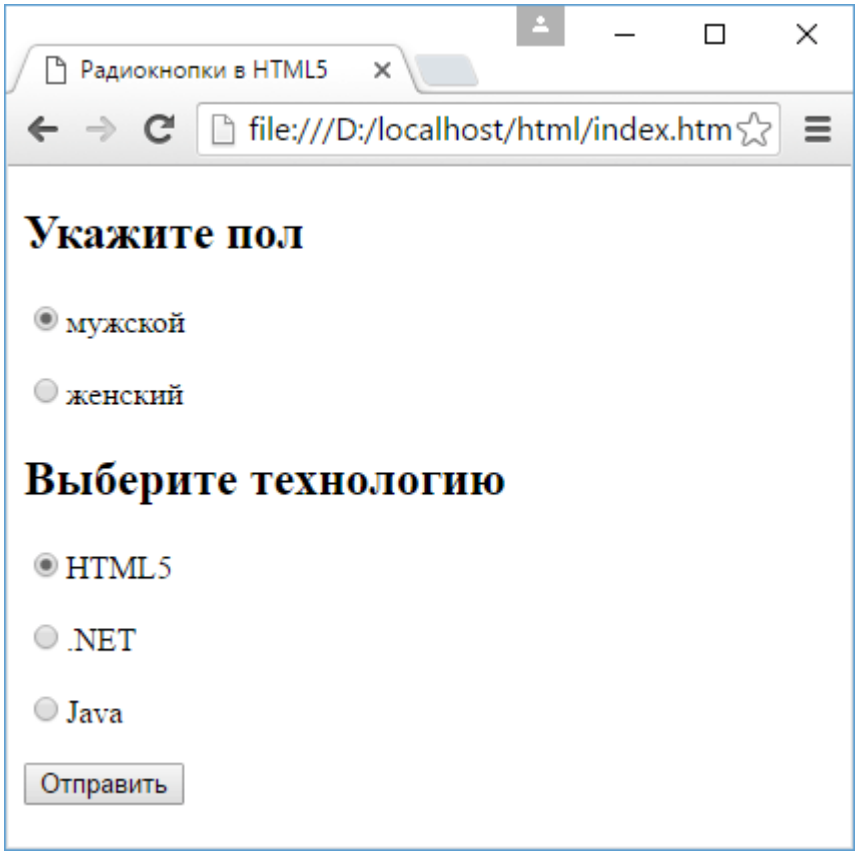
Атрибут **checked** позволяет установить флажок в отмеченное состояние.

Переключатели

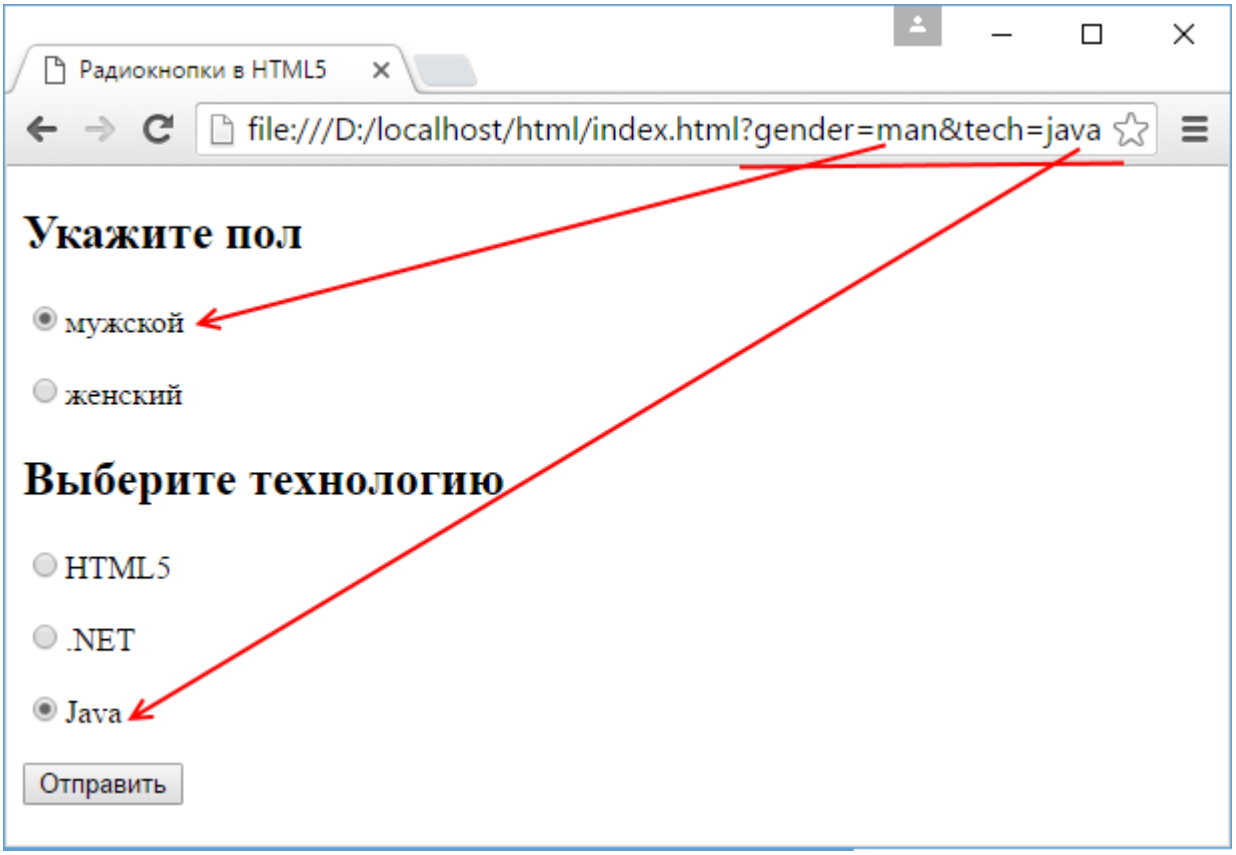
Переключатели или радиокнопки похожи на флажки, они также могут находиться в отмеченном или неотмеченном состоянии. Только для переключателей можно создать одну группу, в которой одновременно можно выбрать только один переключатель. Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Радиокнопки в HTML5</title>
    </head>
    <body>
        <form>
            <h2>Укажите пол</h2>
            <p>
                <input type="radio" value="man" checked name="gender"/>мужской
            </p>
            <p>
                <input type="radio" value="woman" name="gender"/>женский
            </p>
            <h2>Выберите технологию</h2>
            <p>
                <input type="radio" value="html5" checked name="tech"/>HTML5
            </p>
            <p>
                <input type="radio" value="net" name="tech"/>.NET
            </p>
            <p>
                <input type="radio" value="java" name="tech"/>Java
            </p>
            <p>
                <button type="submit">Отправить</button>
            </p>
        </form>
    </body>
</html>
```

Для создания радиокнопки надо указать атрибут `type="radio"`. И теперь другой атрибут `name` указывает не на имя элемента, а на имя группы, к которой принадлежит элемент-радиокнопка. В данном случае у нас две группы радиокнопок: `gender` и `tech`. Из каждой группы мы можем выбрать только один переключатель. Опять же чтобы отметить радиокнопку, у нее устанавливается атрибут `checked`:



Важное значение играет атрибут `value`, который при отправке формы позволяет серверу определить, какой именно переключатель был отмечен:



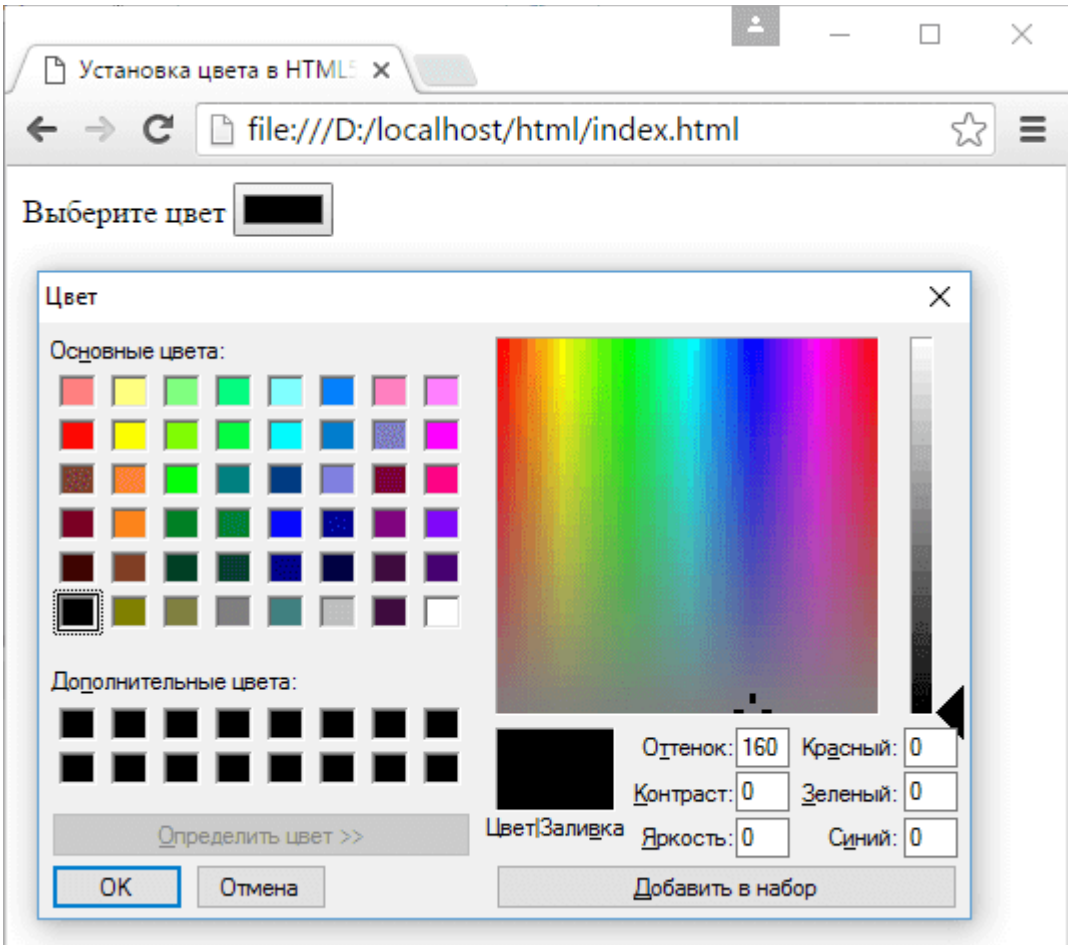
Элементы для ввода цвета, url, email, телефона

Установка цвета

За установку цвета в HTML5 отвечает специальный элемент input с типом **color**:

```
<label for="favcolor">Выберите цвет</label>
<input type="color" id="favcolor" name="favcolor" />
```

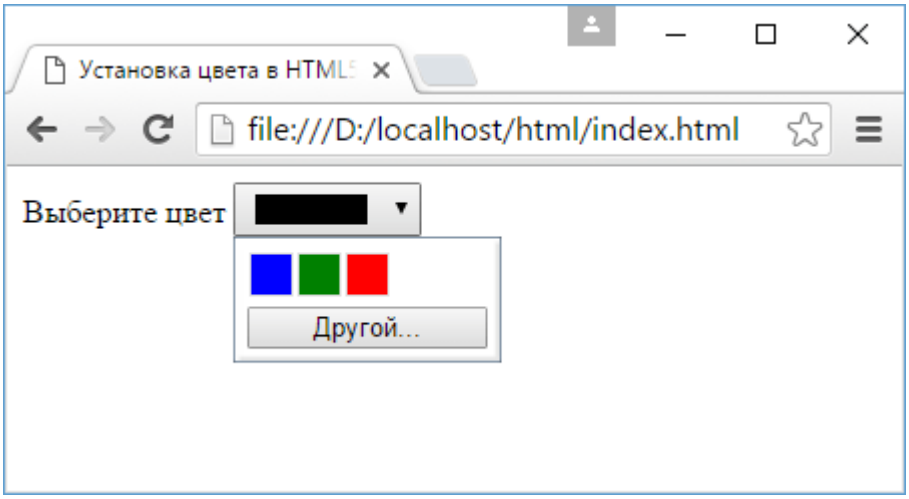
Элемент отображает выбранный цвет. А при нажатии на него появляется специальное диалоговое окно для установки цвета:



Значением этого элемента будет числовой шестнадцатеричный код выбранного цвета.

С помощью элемента `datalist` мы можем задать набор цветов, из который пользователь может выбрать нужный:

```
<label for="favcolor">Выберите цвет</label>
<input type="color" list="colors" id="favcolor" name="favcolor" />
<datalist id="colors">
    <option value="#0000FF" label="blue">
    <option value="#008000" label="green">
    <option value="#ff0000" label="red">
</datalist>
```



Каждый элемент `option` в `datalist` должен в качестве значения принимать шестнадцатеричный код цвета, например, `"#0000FF"`. После выбора цвета данный числовой код устанавливается в качестве значения в элементе input.

Поля для ввода url, email, телефона

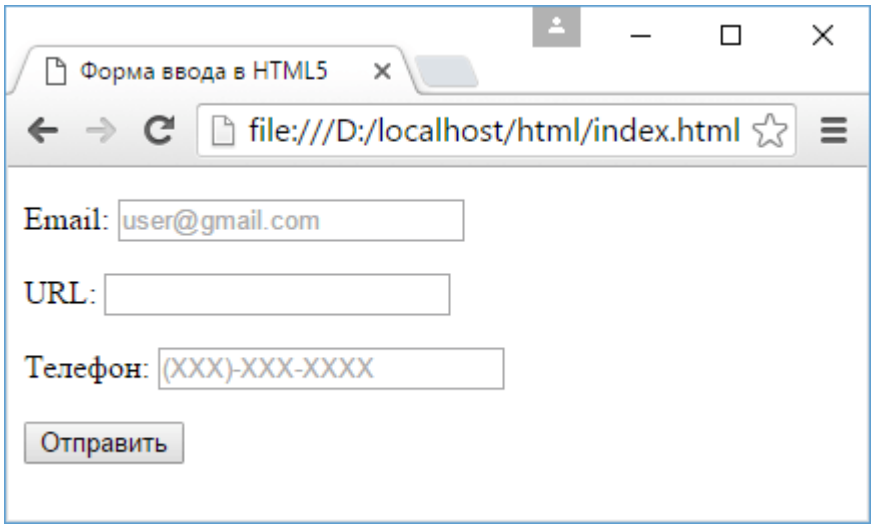
Ряд полей input предназначены для ввода таких данных, как url, адреса электронной почты и телефонного номера. Они однотипны и во многом отличаются только тем, что для атрибута `type` принимают соответственно значения `email`, `tel` и `url`.

Для их настройки мы можем использовать те же атрибуты, что и для обычного текстового поля:

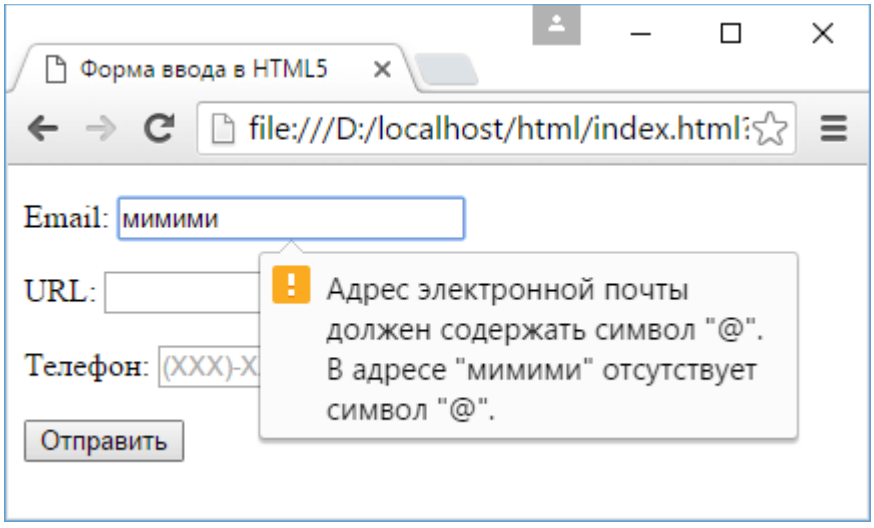
- **maxlength**: максимально допустимое количество символов в поле
- **pattern**: определяет шаблон, которому должен соответствовать вводимый текст
- **placeholder**: устанавливает текст, который по умолчанию отображается в поле
- **readonly**: делает текстовом поле доступным только для чтения
- **required**: указывает, что текстовое поле обязательно должно иметь значение
- **size**: устанавливает ширину поля в видимых символах
- **value**: устанавливает значение по умолчанию для поля
- **list**: устанавливает привязку к элементу datalist со списком возможных значений

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Форма ввода в HTML5</title>
    </head>
    <body>
        <form>
            <p>
                <label for="email">Email: </label>
                <input type="email" placeholder="user@gmail.com" id="email" name="email"/>
            </p>
            <p>
                <label for="url">URL: </label>
                <input type="url" id="url" name="url"/>
            </p>
            <p>
                <label for="phone">Телефон: </label>
                <input type="tel" placeholder="(XXX)-XXX-XXXX" id="phone" name="phone"/>
```

```
            </p>
            <p>
                <button type="submit">Отправить</button>
            </p>
        </form>
    </body>
</html>
```



Основное преимущество подобных полей ввода перед обычными текстовыми полями состоит в том, что поля ввода для email, url, телефона для проверки ввода используют соответствующий шаблон. Например, если мы введем в какое-либо поле некорректное значение и попробуем отправить форму, то браузер может отобразить нам сообщение о некорректном вводе, а форма не будет отправлена:



Элементы для ввода даты и времени

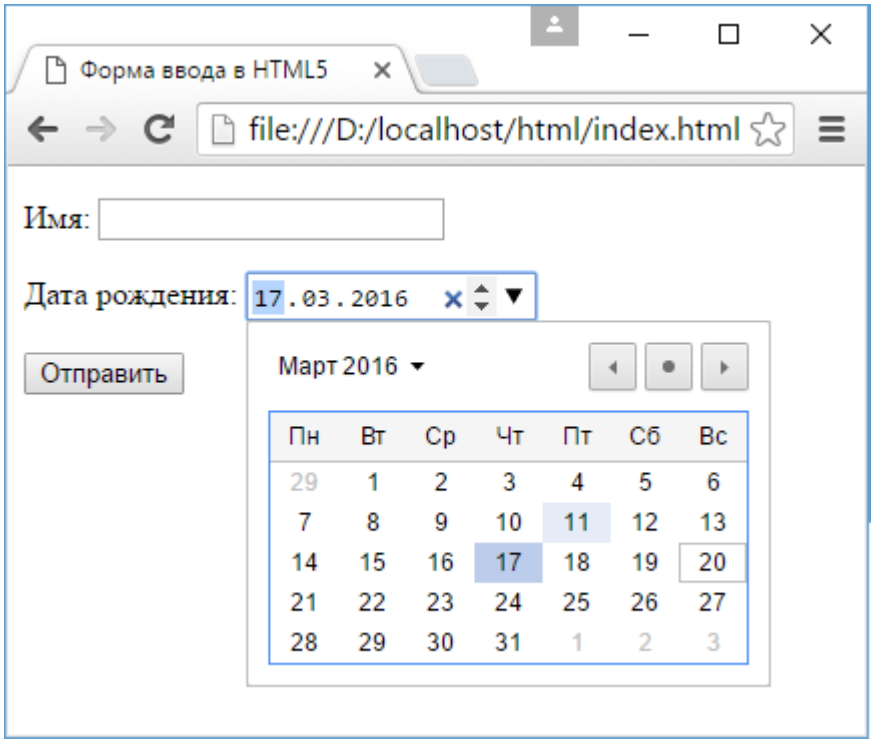
Для работы с датами и временем в HTML5 предназначено несколько типов элементов input:

- **datetime-local**: устанавливает дату и время
- **date**: устанавливает дату
- **month**: устанавливает текущий месяц и год
- **time**: устанавливает время
- **week**: устанавливает текущую неделю

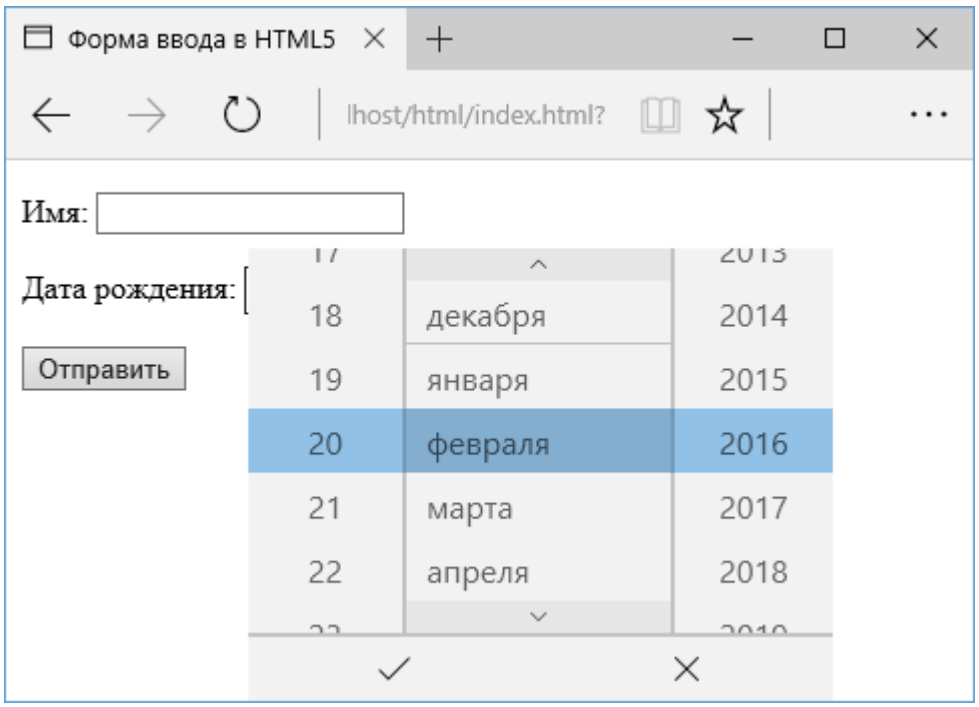
Например, используем поле для установки даты:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Форма ввода в HTML5</title>
    </head>
    <body>
        <form>
            <p>
                <label for="firstname">Имя: </label>
                <input type="text" id="firstname" name="firstname"/>
            </p>
            <p>
                <label for="date">Дата рождения: </label>
                <input type="date" id="date" name="date"/>
            </p>
            <p>
                <button type="submit">Отправить</button>
            </p>
        </form>
    </body>
</html>
```

И при вводе в поле для даты будет открываться календарик:



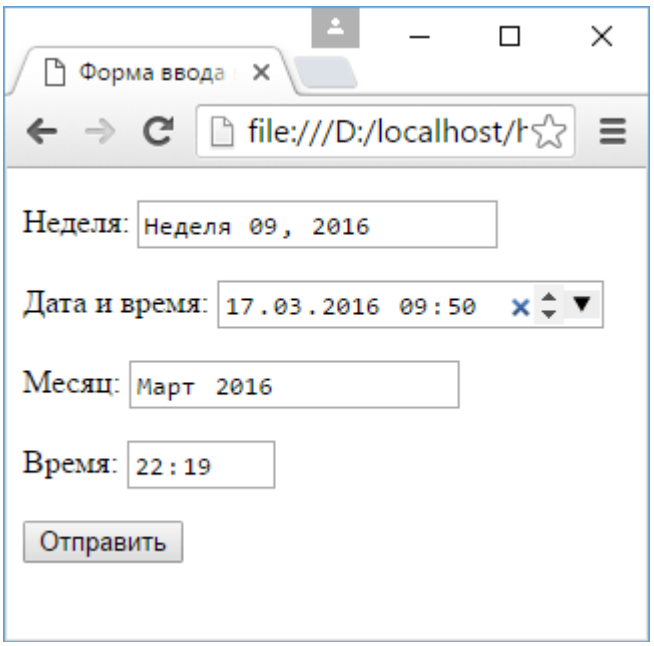
Правда, здесь надо отметить, что действие этого элемента зависит от браузера. В данном случае используется Google Chrome. В последних версиях Opera элемент не будет сильно отличаться. А вот в Microsoft Edge элемент будет выглядеть так:



Применение остальных элементов:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Форма ввода в HTML5</title>
    </head>
    <body>
        <form>
            <p>
                <label for="week">Неделя: </label>
```

```
                <input type="week" name="week" id="week" />
            </p>
            <p>
                <label for="localdate">Дата и время: </label>
                <input type="datetime-local" id="localdate" name="date"/>
            </p>
            <p>
                <label for="month">Месяц: </label>
                <input type="month" id="month" name="month"/>
            </p>
            <p>
                <label for="time">Время: </label>
                <input type="time" id="time" name="time"/>
            </p>
            <p>
                <button type="submit">Отправить</button>
            </p>
        </form>
    </body>
</html>
```

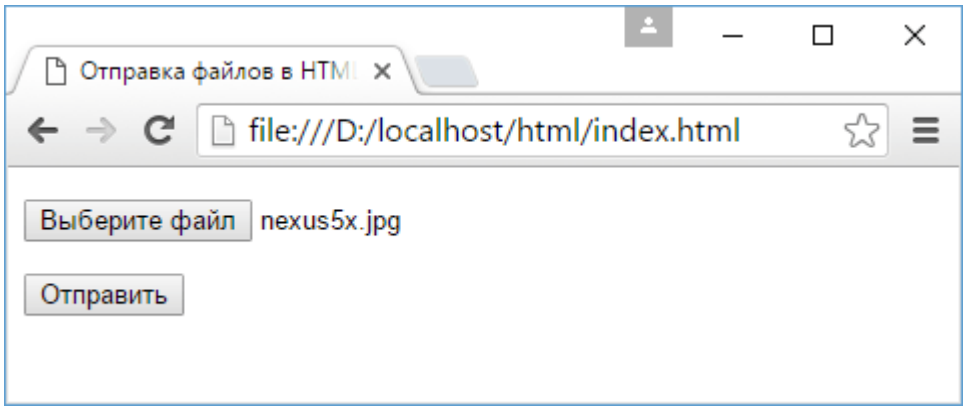


При использовании этих элементов также надо учитывать, что Firefox поддерживает только элементы `date` и `time`, для остальных создаются обычные текстовые поля. А IE11 и вовсе не поддерживают эти элементы.

Отправка файлов

За выбор файлов на форме отвечает элемент `input` с атрибутом **type="file"**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Отправка файлов в HTML5</title>
    </head>
    <body>
        <form enctype="multipart/form-data" method="post"
         action="http://localhost:8080/postfile.php">
            <p>
                <input type="file" name="file" />
            </p>
            <p>
                <input type="submit" value="Отправить" />
            </p>
        </form>
    </body>
</html>
```



При нажатии на кнопку "Выберите файл" открывается диалоговое окно для выбора файла. А после выбора рядом с кнопкой отображается имя выбранного файла.

Важно отметить, что для отправки файла на сервер форма должна иметь атрибут `enctype="multipart/form-data"`.

С помощью ряда атрибутов мы можем дополнительно настроить элементы выбора файла:

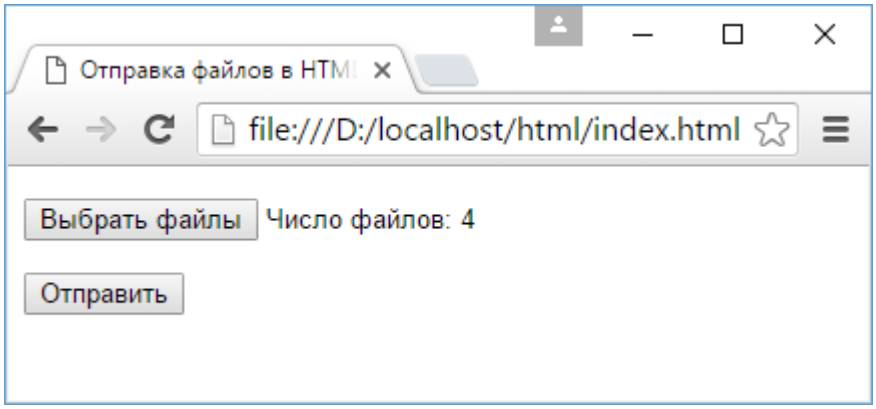
- **accept**: устанавливает тип файл, которые допустимы для выбора
- **multiple**: позволяет выбирать множество файлов
- **required**: требует обязательной установки файла

Например, множественный выбор файлов:

```
<form enctype="multipart/form-data" method="post" action="http://localhost:8080/postfile.php">
    <p>
        <input type="file" name="file" multiple />
    </p>
    <p>
```

```
6           <input type="submit" value="Отправить" />
7       </p>
8   </form>
```

При нажатии на кнопку также открывается диалоговое окно для выбора файлов, только теперь, зажав клавишу CTRL или Shift, мы можем выбрать несколько файлов, а после выбора рядом с кнопкой отобразится количество выбранных файлов:

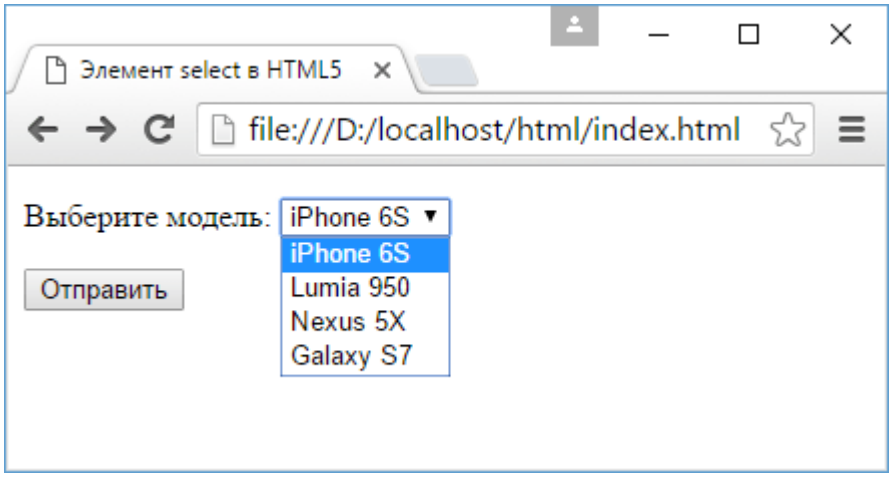


Список select

Элемент **select** создает список. В зависимости от настроек это может быть выпадающий список для выбора одного элемента, либо раскрытый список, в котором можно выбрать сразу несколько элементов.

Создадим выпадающий список:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Элемент select в HTML5</title>
    </head>
    <body>
        <form method="get">
            <p>
                <label for="phone">Выберите модель:</label>
                <select id="phone" name="phone">
                    <option value="iphone 6s">iPhone 6S</option>
                    <option value="lumia 950">Lumia 950</option>
                    <option value="nexus 5x">Nexus 5X</option>
                    <option value="galaxy s7">Galaxy S7</option>
                </select>
            </p>
            <p>
                <input type="submit" value="Отправить" />
            </p>
        </form>
    </body>
</html>
```



Внутрь элемента select помещаются элементы **option** - элементы списка. Каждый элемент `option` содержит атрибут `value`, который хранит значение элемента. При этом значение элемента option не обязательно должно совпадать с отображаемым им текстом. Например:

```
<option value="apple">iPhone 6S</option>
```

С помощью атрибута `selected` мы можем установить выбранный по умолчанию элемент - это необязательно должен быть первый элемент в списке:

```
<select id="phone" name="phone">
    <option value="iphone 6s">iPhone 6S</option>
    <option value="lumia 950">Lumia 950</option>
    <option value="nexus 5x" selected>Nexus 5X</option>
</select>
```

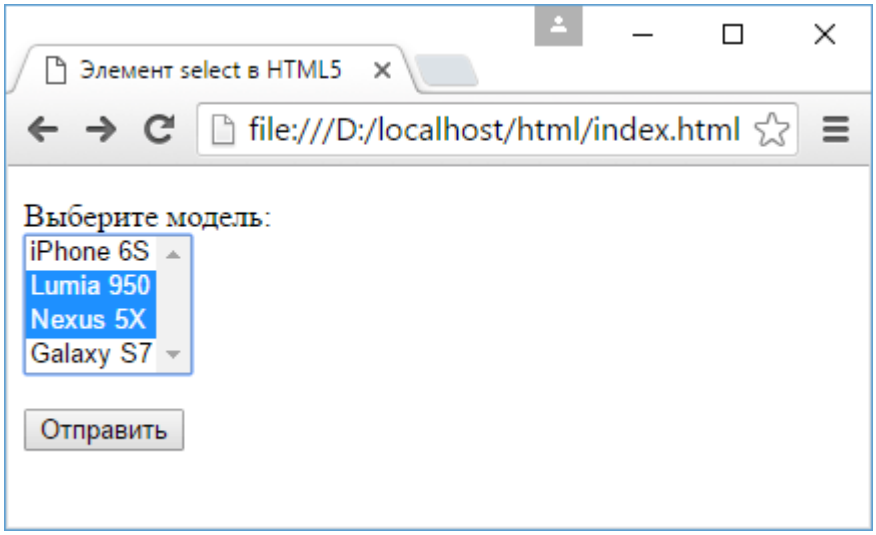
С помощью другого атрибута `disabled` можно запретить выбор определенного элемента. Как правило, элементы с этим атрибутом служат для создания заголовков:

```
<select id="phone" name="phone">
    <option disabled selected>Выберите модель</option>
    <option value="iphone 6s">iPhone 6S</option>
    <option value="lumia 950">Lumia 950</option>
    <option value="nexus 5x" selected>Nexus 5X</option>
</select>
```

Для создания списка с множественным выбором к элементу select надо добавить атрибут **multiple**:

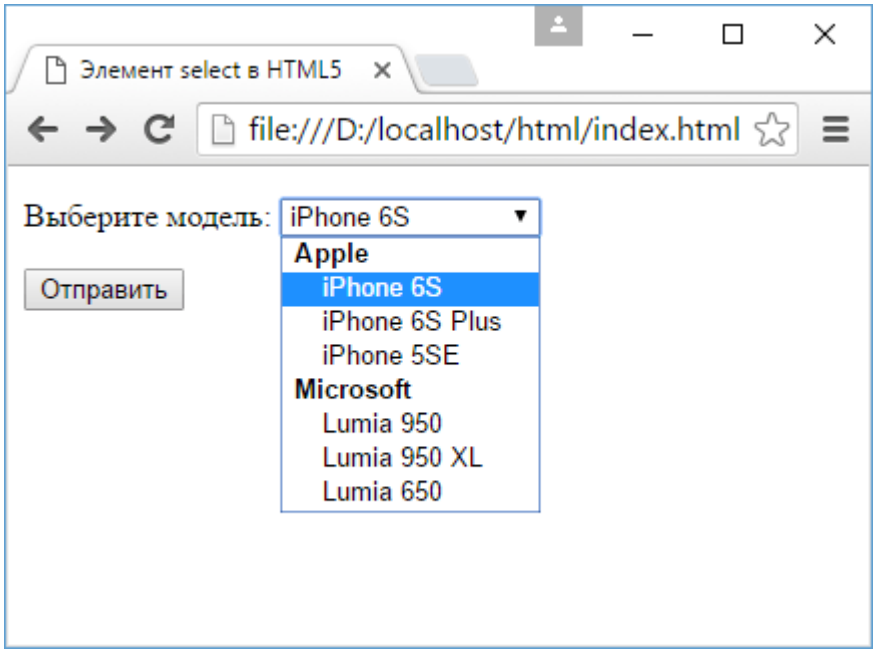
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Элемент select в HTML5</title>
    </head>
    <body>
        <form method="get">
            <p>
                <label for="phone">Выберите модель:</label> <br/>

                <select multiple id="phone" name="phone">
                    <option value="iphone 6s">iPhone 6S</option>
                    <option value="lumia 950">Lumia 950</option>
                    <option value="nexus 5x">Nexus 5X</option>
                    <option value="galaxy s7">Galaxy S7</option>
                </select>
            </p>
            <p>
                <input type="submit" value="Отправить" />
            </p>
        </form>
    </body>
</html>
```

Зажав клавишу Ctrl, мы можем выбрать в таком списке несколько элементов:



Select также позволяет группировать элементы с помощью тега **<optgroup>**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Элемент select в HTML5</title>
    </head>
    <body>
        <form method="get">
            <p>
                <label for="phone">Выберите модель:</label>

                <select id="phone" name="phone">
                    <optgroup label="Apple">
                        <option value="iphone 6s">iPhone 6S</option>
                        <option value="iphone 6s plus">iPhone 6S Plus</option>
                        <option value="iphone 5se">iPhone 5SE</option>
                    </optgroup>
                    <optgroup label="Microsoft">
                        <option value="lumia 950">Lumia 950</option>
                        <option value="lumia 950 xl">Lumia 950 XL</option>
                        <option value="lumia 650">Lumia 650</option>
                    </optgroup>
                </select>
            </p>
            <p>
                <input type="submit" value="Отправить" />
            </p>
        </form>
    </body>
</html>
```

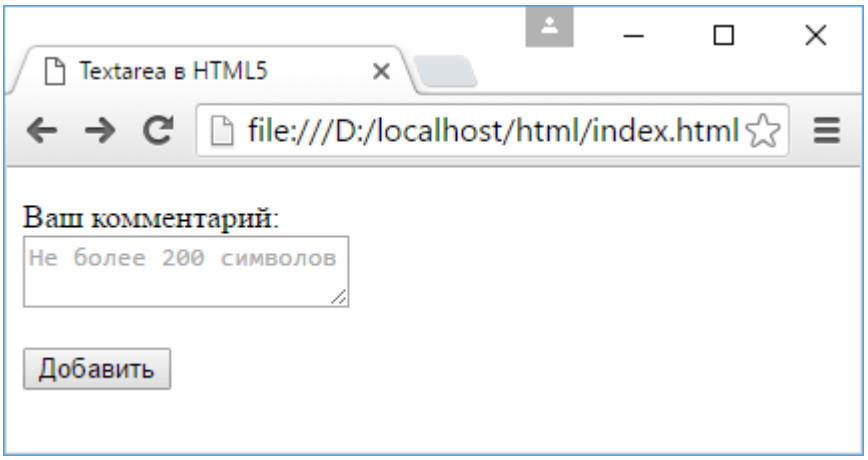


Использование групп элементов применимо как к выпадающему списку, так и к списку со множественным выбором.

Textarea

Элемент `<input type="text" />` позволяет создавать простое однострочное текстовое поле. Однако возможностей этого элемента по вводу текста бывает недостаточно, и в этой ситуации мы можем использовать многострочное текстовое поле, представленное элементом **textarea**:

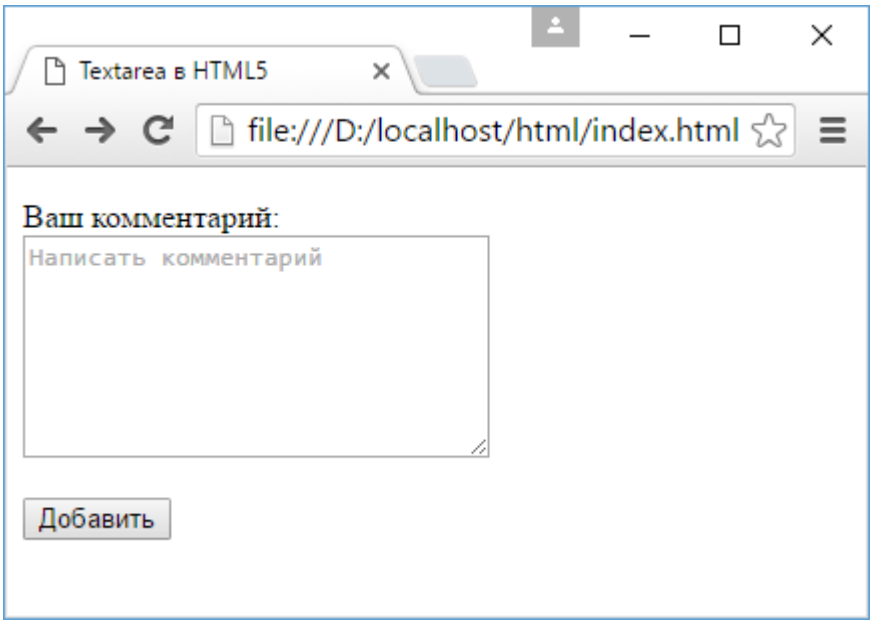
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Textarea в HTML5</title>
    </head>
    <body>
        <form method="get">
            <p>
                <label for="comment">Ваш комментарий:</label><br/>
                <textarea name="comment" id="comment" placeholder="Не более 200 символов"
                 maxlength="200"></textarea>
            </p>
            <p>
                <input type="submit" value="Добавить" />
            </p>
        </form>
    </body>
</html>
```



С помощью дополнительных атрибутов **cols** и `rows` можно задать соответственно количество столбцов и строк:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Textarea в HTML5</title>
    </head>
    <body>
        <form method="get">
            <p>
                <label for="comment">Ваш комментарий:</label><br/>
                <textarea id="comment" name="comment" placeholder="Написать комментарий"
                    cols="30" rows="7"></textarea>
```

```
13                </p>
14                <p>
15                    <input type="submit" value="Добавить" />
16                </p>
17            </form>
18        </body>
19    </html>
```



Валидация форм

Итак, в нашем распоряжении имеются различные элементы, которые мы можем использовать на форме. Мы можем вводить в них различные значения. Однако нередко пользователи вводят не совсем корректные значения: например, ожидается ввод чисел, а пользователь вводит буквы и т.д. И для предупреждения и проверки некорректного ввода в HTML5 существует механизм валидации.

Преимущество использования валидации в HTML5 заключается в том, что пользователь после некорректного ввода может сразу получить сообщение об ошибке и внести соответствующие изменения в введенные данные.

Для создания валидации у элементов форм HTML5 используется ряд атрибутов:

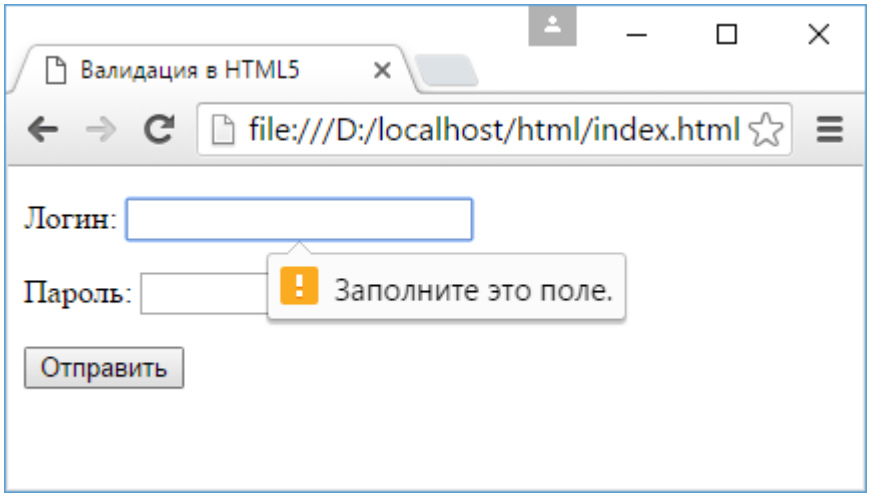
- **required**: требует обязательного ввода значения. Для элементов textarea, select, input (с типом text, password, checkbox, radio, file, datetime-local, date, month, time, week, number, email, url, search, tel)
- **min** и **max**: минимально и максимально допустимые значения. Для элемента input с типом datetime-local, date, month, time, week, number, range
- **pattern**: задает шаблон, которому должны соответствовать вводимые данные. Для элемента input с типом text, password, email, url, search, tel

Атрибут required

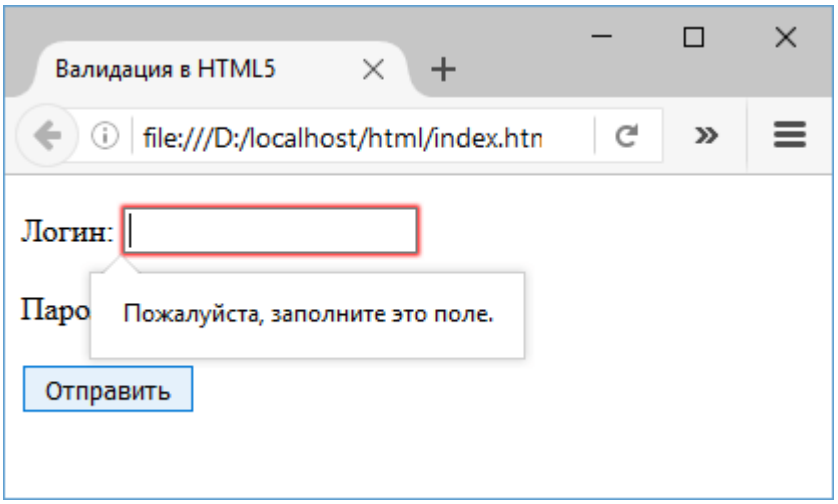
Атрибут required требует обязательного наличия значения:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Валидация в HTML5</title>
    </head>
    <body>
        <form method="get">
            <p>
                <label for="login">Логин:</label>
                <input type="text" required id="login" name="login" />
            </p>
            <p>
                <label for="password">Пароль:</label>
                <input type="password" required id="password" name="password" />
            </p>
            <p>
                <input type="submit" value="Отправить" />
            </p>
        </form>
    </body>
</html>
```

Если мы не введем в эти поля никаких данных, оставив их пустыми, и нажмем на кнопку отправки, то браузер высветит нам сообщения об ошибке, а данные не будут отправлены на сервер:



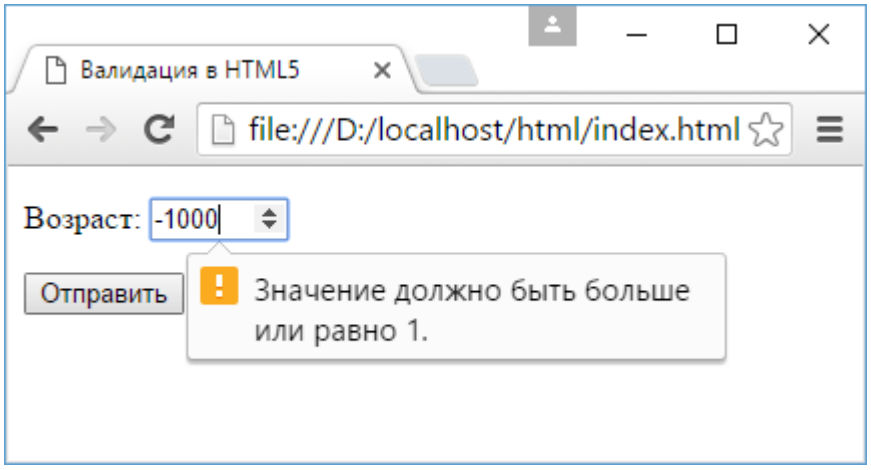
В зависимости от браузера визуализация сообщения может несколько отличаться. Также границы некорректного поля ввода могут окрашиваться в красный цвет. Например, поведение при отправке пустых сообщений в Firefox:



Атрибуты max и min

Для ограничения диапазона вводимых значений применяются атрибуты max и min:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Валидация в HTML5</title>
    </head>
    <body>
        <form method="get">
            <p>
                <label for="age">Возраст:</label>
                <input type="number" min="1" max="100" value="18" id="age" name="age"/>
            </p>
            <p>
                <input type="submit" value="Отправить" />
            </p>
        </form>
    </body>
</html>
```



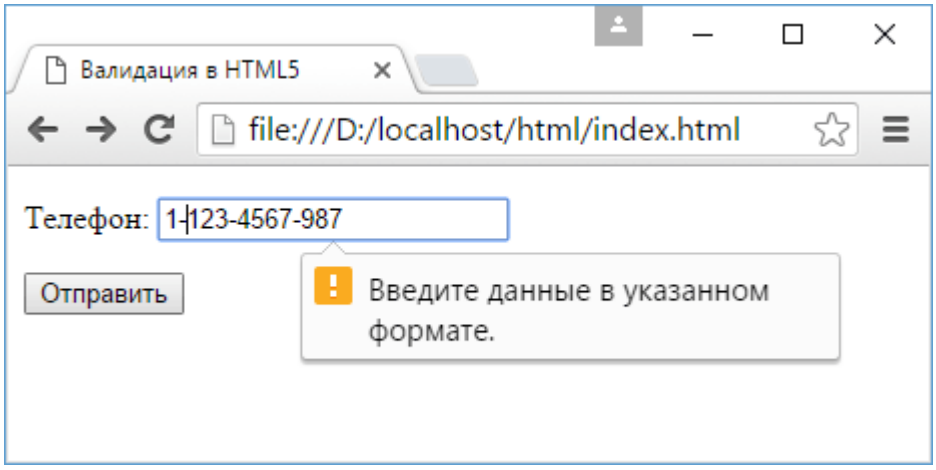
Атрибут pattern

Атрибут pattern задает шаблон, которому должны соответствовать данные. Для определения шаблона используется язык так называемых регулярных выражений. Рассмотрим самые простейшие примеры:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Валидация в HTML5</title>
    </head>
    <body>
        <form method="get">
            <p>
                <label for="phone">Телефон:</label>
                <input type="text" placeholder="+1-234-567-8901"
                        pattern="\+\d-\d{3}-\d{3}-\d{4}" id="phone" name="phone" />
            </p>
            <p>
                <input type="submit" value="Отправить" />
            </p>
        </form>
    </body>
</html>
```

Здесь для ввода номера телефона используется регулярное выражение `\+\d-\d{3}-\d{3}-\d{4}`. Оно означает, что первым элементом в номере должен идти знак плюс +.
Выражение **\d** представляет любую цифру от 0 до 9. Выражение `\d{3}` означает три подряд идущих цифры, а `\d{4}` - четыре цифры подряд. То есть это выражение будет соответствовать номеру телефона в формате "+1-234-567-8901".

Если мы введем данные, которые не соответствуют этому шаблону, и нажмем на отправку, то браузер отобразит ошибку:



Отключение валидации

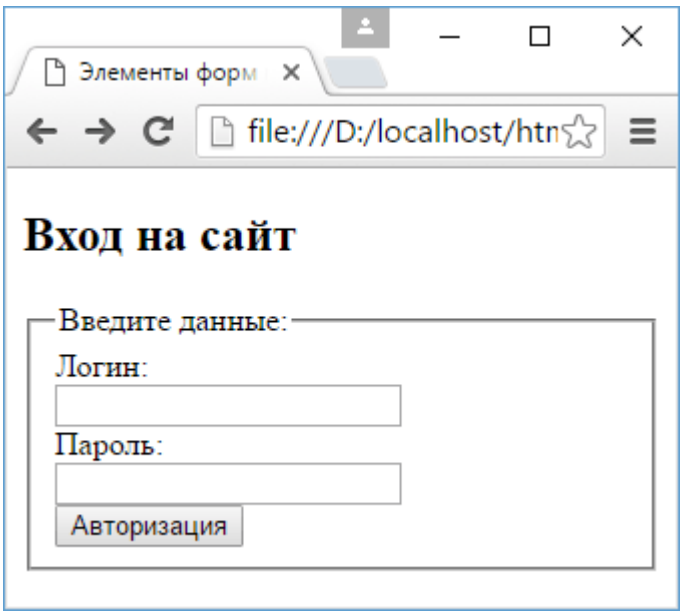
Не всегда валидация является желаемой, иногда требуется ее отключить. И в этом случае мы можем использовать либо у элемента формы атрибут **novalidate**, либо у кнопки отправки атрибут **formnovalidate**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Валидация в HTML5</title>
    </head>
    <body>
        <form novalidate method="get">
            <p>
                <label for="phone">Телефон:</label>
                <input type="text" placeholder="+1-234-567-8901"
                        pattern="\+\d-\d{3}-\d{3}-\d{4}" id="phone" name="phone" />
            </p>
            <p>
                <input type="submit" value="Отправить" formnovalidate />
            </p>
        </form>
    </body>
</html>
```

Элементы fieldset и legend

Для группировки элементов формы нередко применяется элемент **fieldset**. Он создает границу вокруг вложенных элементов, как бы создавая из них группу. Вместе с ним используется элемент **legend**, который устанавливает заголовок для группы элементов:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Элементы форм в HTML5</title>
    </head>
    <body>
        <h2>Вход на сайт</h2>
        <form>
            <fieldset>
                <legend>Введите данные:</legend>
                <label for="login">Логин:</label><br>
                <input type="text" name="login" id="login" /><br>
                <label for="password">Пароль:</label><br>
                <input type="password" name="password" id="password" /><br>
                <input type="submit" value="Авторизация">
            </fieldset>
        </form>
    </body>
</html>
```



При необходимости мы можем создать на одной форме несколько групп с помощью элементов fieldset.

Семантическая структура страницы

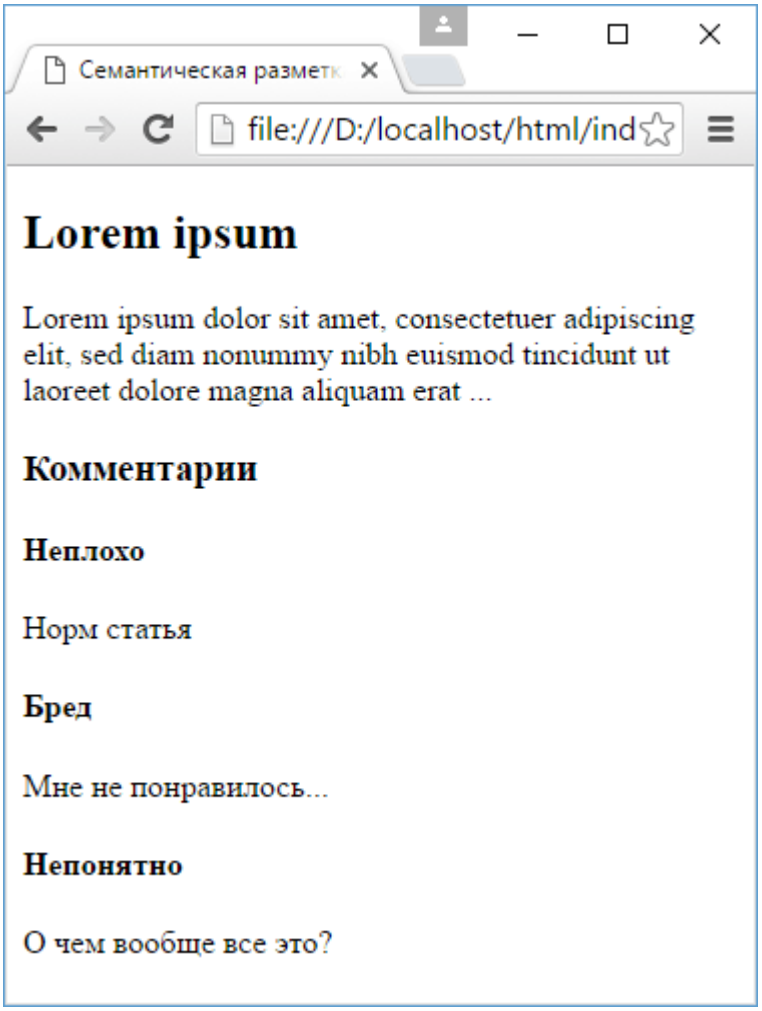
Элемент article

Элемент **article** представляет целостный блок информации на странице, который может рассматриваться отдельно и использоваться независимо от других блоков. Например, это может быть пост на форуме или статья в блоге, онлайн-журнале, комментарий пользователя.

Один элемент article может включать несколько элементов article. Например, мы можем заключить в элемент article всю статью в блоге, и этот элемент будет содержать другие элементы article, которые представляют комментарии к этой статье в блоге. То есть статья в блоге может рассматриваться нами как отдельная семантическая единица, и в то же время комментарии также могут рассматривать отдельно вне зависимости от другого содержимого.

Использование article на примере статьи из блога с комментариями:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Семантическая разметка в HTML5</title>
    </head>
    <body>
        <article>
            <h2>Lorem ipsum</h2>
            <div>
                Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh
                euismod tincidunt ut laoreet dolore magna aliquam erat ...
            </div>
            <div>
                <h3>Комментарии</h3>
                <article>
                    <h4>Неплохо</h4>
                    <p>Норм статья</p>
                </article>
                <article>
                    <h4>Бред</h4>
                    <p>Мне не понравилось...</p>
                </article>
                <article>
                    <h4>Непонятно</h4>
                    <p>О чем вообще все это?</p>
                </article>
            </div>
        </article>
    </body>
</html>
```



Здесь вся статья может быть помещена в элемент article, и в то же время каждый отдельный комментарий также представляет отдельный элемент article.

При использовании article надо учитывать, что каждый элемент article должен быть идентифицирован с помощью включения в него заголовка h1-h6.

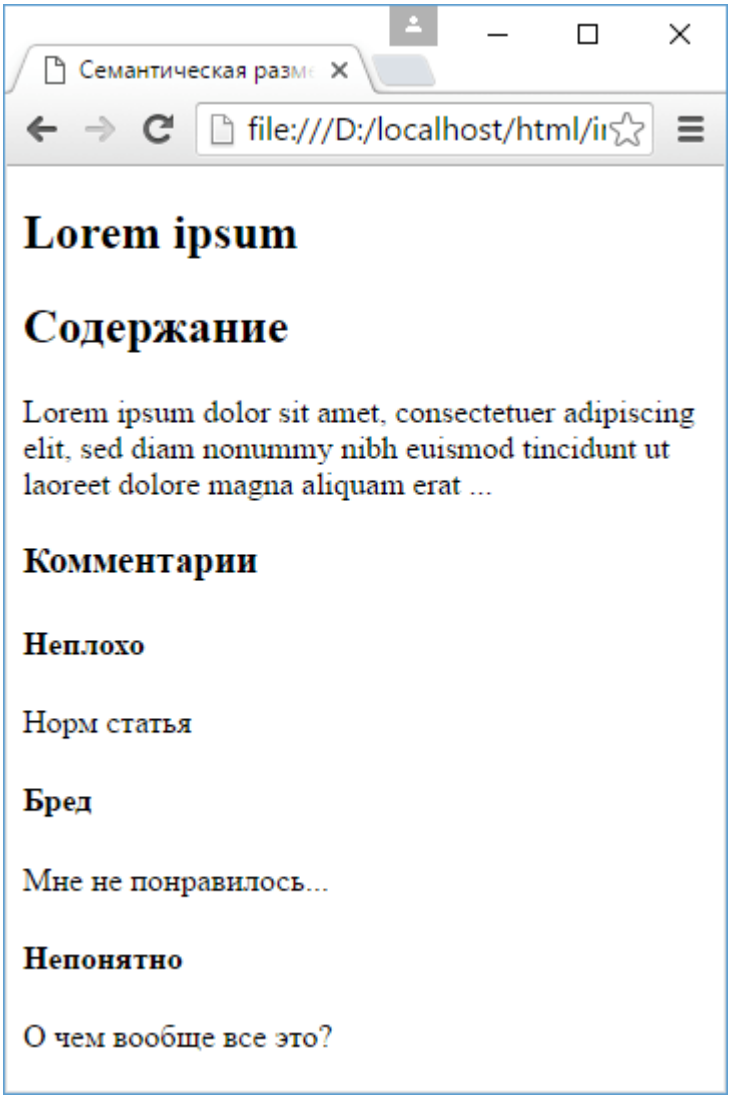
Элемент section

Элемент section объединяет связанные между собой куски информации html-документа, выполняя их группировку. Например, section может включать набор вкладок на странице, новости, объединенные по категории и т.д.

Каждый элемент section должен быть идентифицирован с помощью заголовка h1-h6.

При этом элемент section может содержать несколько элементов article, выполняя их группировку, так и один элемент article может содержать несколько элементов section.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Семантическая разметка в HTML5</title>
    </head>
    <body>
        <article>
            <h1>Lorem ipsum</h1>
            <section>
                <h2>Содержание</h2>
                <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh
                euismod tincidunt ut laoreet dolore magna aliquam erat ...</p>
            </section>
            <section>
                <h3>Комментарии</h3>
                <article>
                    <h4>Неплохо</h4>
                    <p>Норм статья</p>
                </article>
                <article>
                    <h4>Бред</h4>
                    <p>Мне не понравилось...</p>
                </article>
                <article>
                    <h4>Непонятно</h4>
                    <p>О чем вообще все это?</p>
                </article>
            </section>
        </article>
    </body>
</html>
```



Здесь для блока основного содержимого создается секция и для набора комментариев также создается элемент section.

Элемент nav

Элемент **nav** призван содержать элементы навигации по сайту. Как правило, это ненумерованный список с набором ссылок.

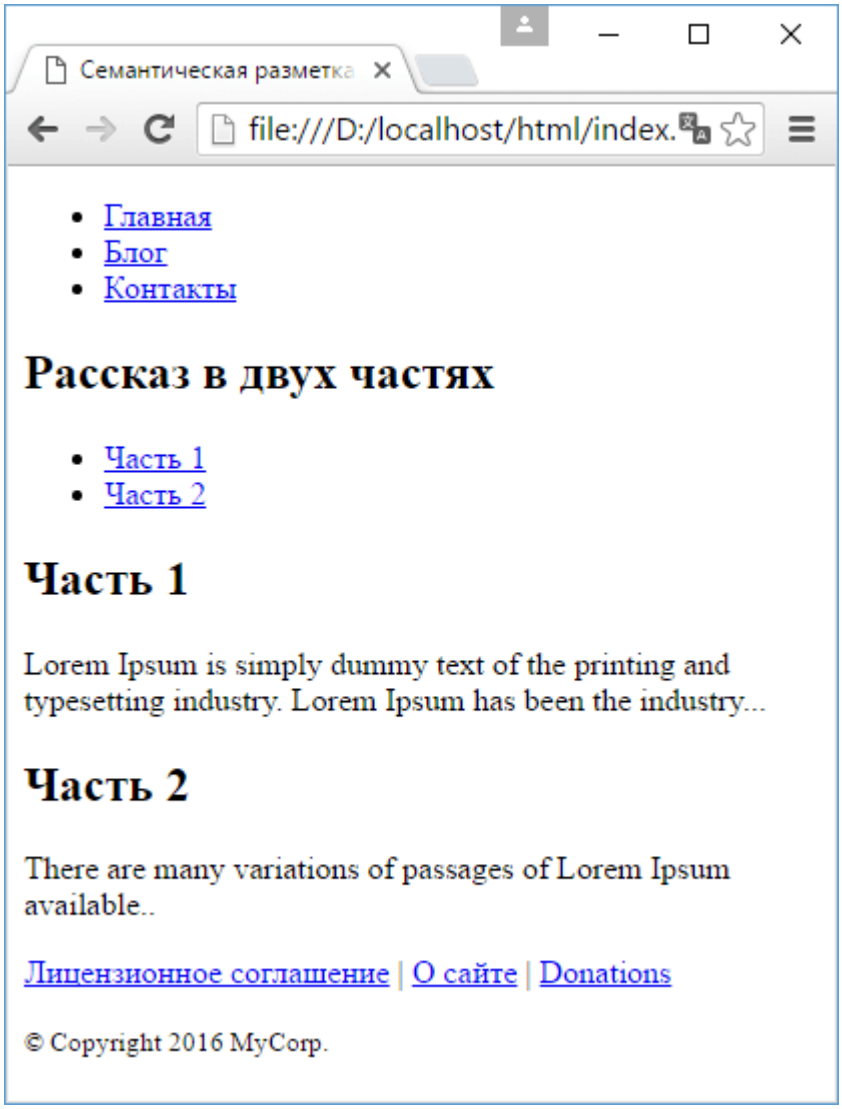
На одной веб-странице можно использовать несколько элементов nav. Например, один элемент навигации для перехода по страницам на сайте, а другой - для перехода внутри html-документа.

Не все ссылки обязательно помещать в элемент **nav**. Например, некоторые ссылки могут не представлять связанного блока навигации, например, ссылка на главную страницу, на лицензионное соглашение по поводу использования сервиса и подобные ссылки, которые часто помещаются внизу страницы. Как правило, их достаточно определить в элементе footer, а элемент nav для них использовать необязательно.

Используем элемент nav для создания навигационного меню:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Семантическая разметка в HTML5</title>
    </head>
    <body>
        <nav>
            <ul>
                <li><a href="/">Главная</a></li>
                <li><a href="/blog">Блог</a></li>
                <li><a href="/contacts">Контакты</a></li>
            </ul>
        </nav>
        <article>
            <header>
                <h2>Рассказ в двух частях</h2>
            </header>
            <nav>
                <ul>
                    <li><a href="#part1">Часть 1</a></li>
                    <li><a href="#part2">Часть 2</a></li>
                </ul>
            </nav>
            <div>
                <section id="part1">
                    <h2>Часть 1</h2>
                    <p>Lorem Ipsum is simply dummy text of the printing and
                    typesetting industry.
                    Lorem Ipsum has been the industry...</p>
                </section>
                <section id="part2">
                    <h2>Часть 2</h2>
                    <p>There are many variations of passages of Lorem Ipsum
                    available..</p>
                </section>
            </div>
            <footer>
```

```
            </footer>
        </article>
        <footer>
            <p><a href="/license">Лицензионное соглашение</a> |
            <a href="/about">О сайте</a> |
            <a href="/donation">Donations</a></p>
            <p><small>© Copyright 2016 MyCorp.</small></p>
        </footer>
    </body>
</html>
```



В данном случае определены два блока nav - один для межстраничной навигации, а другой - для навигации внутри страницы.

Необязательно все ссылки помещать в элементы nav. Так, в данном случае ряд ссылок располагаются в элементе footer.

Элементы header, footer и address

Header

Элемент **header** является как бы вводным элементом, предваряющим основное содержимое. Здесь могут быть заголовки, элементы навигации или какие-либо другие вспомогательные элементы, например, логотип, форма поиска и т.п. Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Семантическая разметка в HTML5</title>
    </head>
    <body>
        <header>
            <h1>Онлайн-магазин телефонов</h1>
                <nav>
                    <ul>
                        <li><a href="/apple">Apple</a>
                        <li><a href="/microsoft">Microsoft</a>
                        <li><a href="/samsung">Samsung</a>
                    </ul>
                </nav>
        </header>
        <div>
            Информация о новинках мобильного мира....
        </div>
    </body>
</html>
```

Элемент header нельзя помещать в такие элементы как address, footer или другой header.

Footer

Элемент **footer** обычно содержит информацию о том, кто автор контента на веб-странице, копирайт, дата публикации, блок ссылок на похожие ресурсы и т.д. Как правило, подобная информация располагается в конце веб-страницы или основного содержимого, однако, footer не имеет четкой привязки к позиции и может использоваться в различных местах веб-страницы.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Семантическая разметка в HTML5</title>
    </head>
    <body>
        <h1>Xiaomi Mi 5</h1>
        <div>
            Xiaomi Mi 5 оснащен восьмиядерным процессором Qualcomm Snapdragon 820.
            Размер внутреннего хранилища - 32 и 64 МБ.
        </div>
        <footer>
            <p><a href="/license">Лицензионное соглашение</a><br/>
```

```
            Copyright © 2016. SomeSite.com</p>
        </footer>
    </body>
</html>
```

Здесь определен футер для всей веб-страницы. В него помещена ссылка на лицензионное соглашение использования сервисом и информация о копирайте.

Футер необязательно должен быть определен для всей страницы. Это может быть и отдельная секция контента:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Семантическая разметка в HTML5</title>
    </head>
    <body>
        <section>
            <h1>Последние статьи</h1>
            <article>
                <h2>Анонс Samsung Galaxy S7</h2>
                <p>Состоялся выход нового флагмана от компании Samsung Galaxy S7</p>
                <footer>
                    Дата публикации: <time datetime="2016-03-16T15:16-00:00">
                    16.03.2016 15:16</TIME>
                </footer>
            </article>
            <article>
                <h2>Скидки на Microsoft Lumia 950</h2>
                <p>С 1 марта смартфон Microsoft Lumia 950 стоит на 10 000 рублей
                дешевле</p>
                <footer>
                    Дата публикации: <time datetime="2016-03-01T14:36-00:00">
                    01.03.2016 14:36</TIME>
                </footer>
            </article>
        </section>
    </body>
</html>
```

Элемент footer не следует помещать в такие элементы как address, header или другой footer.

Address

Элемент **address** предназначен для отображения контактной информации, которая связана с ближайшим элементом article или body. Нередко данный элемент размещается в футере:

```
<footer>
    <address>
        Контакты для связи <a href="mailto:js@example.com">Том Смит</a>.
    </address>
    <p>© copyright 2016 Example Corp.</p>
</footer>
```

Элемент aside

Элемент **aside** представляет содержимое, которое косвенно связано с остальным контентом веб-станицы и которое может рассматриваться независимо от него. Данный элемент можно использовать, например, для сайдбаров, для рекламных блоков, блоков навигационных элементов, различных плагинов типа твиттера или фейсбука и т.д.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Семантическая разметка в HTML5</title>
    </head>
    <body>
        <aside style="float:right; width:200px;">
            <h2>Скидки на Microsoft Lumia 950</h2>
            <p>Только до 31 марта смартфон Microsoft Lumia 950 стоит на 10 000
            рублей дешевле. В подарок вы получите бесплатный чупа-чупс.
            <a href="buy/id=3">Купить</a></p>
        </aside>
        <article>
            <h2>Релиз Samsung Galaxy S7</h2>
            <p>Состоялся выход нового флагмана от компании Samsung Galaxt S7.
            Вместе с новым флагманом компания
            Samsung представила новый шлем виртуальной реальности Gear VR...</p>
        </article>
    </body>
</html>
```

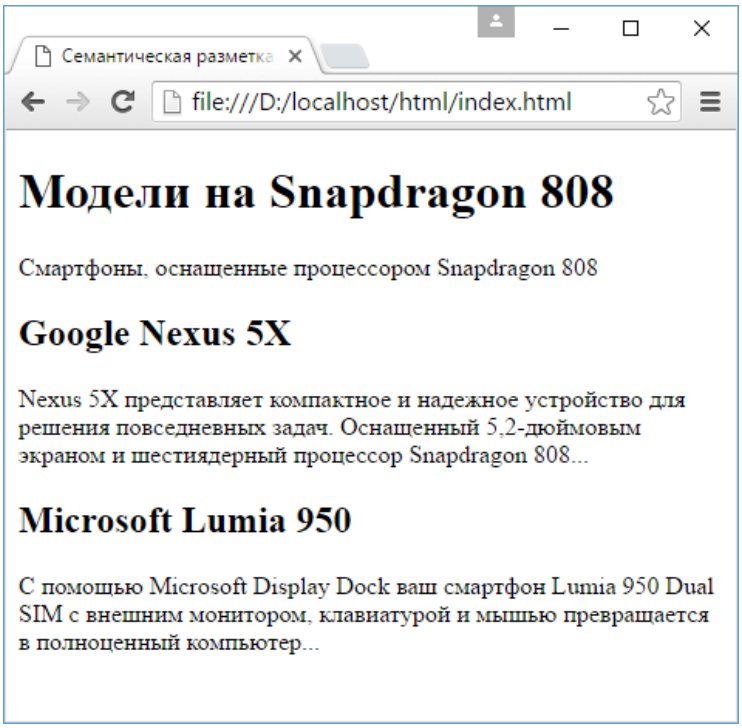
Здесь содержимое блока aside довольно косвенно связано с основным контентом из элемента article. Поэтому все это содержимое мы можем поместить в aside.

Элемент main

Элемент **main** представляет основное содержимое веб-страницы. Он представляет уникальный контент, в который не следует включать повторяющиеся на разных веб-страницах элементы сайдбаров, навигационные ссылки, информацию о копирайте, логотипы и тому подобное.

Используем элемент main:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Семантическая разметка в HTML5</title>
    </head>
    <body>
        <main>
          <h1>Модели на Snapdragon 808</h1>
          <p>Смартфоны, оснащенные процессором Snapdragon 808</p>

          <article>
            <h2>Google Nexus 5X</h2>
            <p>Nexus 5X представляет компактное и надежное устройство для решения повседневных задач. Оснащенный 5,2-дюймовым экраном и шестиядерный процессор Snapdragon 808...</p>
          </article>

          <article>
            <h2>Microsoft Lumia 950</h2>
            <p>С помощью Microsoft Display Dock ваш смартфон Lumia 950 Dual SIM с внешним монитором, клавиатурой и мышью превращается в полноценный компьютер...</p>
          </article>
        </main>
    </body>
</html>
```



Не стоит думать, что абсолютно все содержимое надо обязательно помещать в элемент main. Нет мы также можем использовать вне его другие элементы, например, header и footer:

```
<body>
    <header>
        ............
    </header>
    <main>
        ..............
    </main>
    <footer>
        ..............
    </footer>
</body>
```

Однако надо помнить, что элемент main не может быть вложенным в такие элементы, как article, aside, footer, header, nav. Кроме того, на веб-странице допустимо наличие только одного элемента main.

Также стоит отметить, что на данный момент есть небольшие проблемы с поддержкой этого элемента в браузерах. В частности, IE 11 не поддерживает данный элемент (в остальных браузерах полная поддержка), поэтому в этом случае стоит использовать атрибут роли:

```
<main role="main">
  ...
</main>
```

Указание роли позволит IE11 и более старшим версиям IE должным образом интерпретировать элемент.

Основы CSS3. Селекторы

Введение в стили

Любой html-документ, сколько бы он элементов не содержал, будет по сути "мертвым" без использования стилей. Стили или лучше сказать каскадные таблицы стилей (**Cascading Style Sheets**) или попросту CSS определяют представление документа, его внешний вид. Рассмотрим вкратце применение стилей в контексте HTML5.

Стиль в CSS представляет правило, которое указывает веб-браузеру, как надо форматировать элемент. Форматирование может включать установку цвета фона элемента, установку цвета и типа шрифта и так далее.

Определение стиля состоит из двух частей: **селектор**, который указывает на элемент, и **блок объявления стиля** - набор команд, которые устанавливают правила форматирования. Например:

```
1   div{
2       background-color:red;
3       width: 100px;
4       height: 60px;
5   }
```

В данном случае селектором является `div`. Этот селектор указывает, что этот стиль будет применяться ко всем элементам div.

После селектора в фигурных скобках идет **блок объявления стиля**. Между открывающей и закрывающей фигурными скобками определяются команды, указывающие, как форматировать элемент.

Каждая команда состоит из **свойства** и **значения**. Так, в следующем выражении:

```
1   background-color:red;
```

`background-color` представляет свойство, а `red` - значение. Свойство определяет конкретный стиль. Свойств css существует множество. Например, `background-color` определяет цвет фона. После двоеточия идет значение для этого свойства. Например, выше указанная команда определяет для свойства `background-color` значение `red`. Иными словами, для фона элемента устанавливается цвет "red", то есть красный.

После каждой команды ставится точка с запятой, которая отделяет данную команду от других.

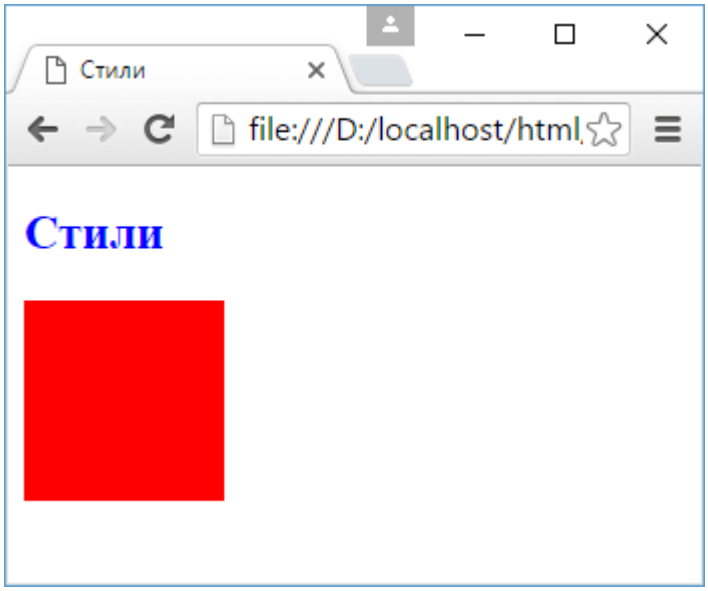
Наборы таких стилей часто называют таблицами стилей или **CSS** (**Cascading Style Sheets** или каскадные таблицы стилей). Существуют различные способы определения стилей.

Атрибут style

Первый способ заключается во встраивании стилей непосредственно в элемент с помощью атрибута **style**:

```
1    <!DOCTYPE html>
2    <html>
3        <head>
```

```
        <meta charset="utf-8">
        <title>Стили</title>
    </head>
    <body>
        <h2 style="color:blue;">Стили</h2>
        <div style="width: 100px; height: 100px; background-color: red;"></div>
    </body>
</html>
```



Здесь определены два элемента - заголовок h2 и блок div. У заголовка определен синий цвет текста с помощью свойства `color`. У блока div определены свойства ширины (`width`), высоты (`height`), а также цвета фона (`background-color`).

Второй способ состоит в использования элемента `style` в документе html. Этот элемент сообщает браузеру, что данные внутри являются кодом css, а не html:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Стили</title>
        <style>
        h2{
            color:blue;
        }
        div{
            width: 100px;
            height: 100px;
            background-color: red;
        }
        </style>
    </head>
    <body>
        <h2>Стили</h2>
        <div></div>
    </body>
</html>
```

Результат в данном случае будет абсолютно тем же, что и в предыдущем случае.

Часто элемент `style` определяется внутри элемента `head`, однако может также использоваться в других частях HTML-документа. Элемент `style` содержит наборы стилей. У каждого стиля указывается вначале **селектор**, после чего в фигурных скобках идет все те же определения свойств css и их значения, что были использованы в предыдущем примере.

Второй способ делает код html чище за счет вынесения стилей в элемент `style`. Но также есть и третий способ, который заключается в вынесении стилей во внешний файл.

Создадим в одной папке с html странице текстовый файл, который переименуем в *styles.css* и определим в нем следующее содержимое:

```
h2{
    color:blue;
}
div{
    width: 100px;
    height: 100px;
    background-color: red;
}
```

Это те же стили, что были внутри элемента `style`. И также изменим код html-страницы:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Стили</title>
        <link rel="stylesheet" type="text/css" href="styles.css"/>
    </head>
    <body>
        <h2>Стили</h2>
        <div></div>
    </body>
</html>
```

Здесь уже нет элемента `style`, зато есть элемент `link`, который подключает выше созданный файл styles.css: `<link rel="stylesheet" type="text/css" href="styles.css"/>`

Таким образом, определяя стили во внешнем файле, мы делаем код html чище, структура страницы отделяется от ее стилизации. При таком определении стили гораздо легче модифицировать, чем если бы они были определены внутри элементов или в элементе `style`, и такой способ является предпочтительным в HTML5.

Использование стилей во внешних файлах позволяет уменьшить нагрузку на веб-сервер с помощью механизма кэширования. Поскольку веб-браузер может кэшировать css-файл и при последующем обращении к веб-странице извлекать нужный css-файл из кэша.

Также возможна ситуация, когда все эти подходы сочетаются, а для одного элемента одни свойства css определены внутри самого элемента, другие свойства css определены внутри элемента `style`, а третьи находятся во внешнем подключенном файле. Например:

```
<!DOCTYPE html>
```

```
<html>
    <head>
        <link rel="stylesheet" type="text/css" href="styles.css"/>
        <style>
            div{
                width:200px;
            }
        </style>
    </head>
    <body>
        <div style="width:120px;"></div>
    </body>
</html>
```

А в файле *style.css* определен следующий стиль:

```
div{
    width:50px;
    height:50px;
    background-color:red;
}
```

В данном случае в трех местах для элемента `div` определено свойство `width`, причем с разным значением. Какое значение будет применяться к элементу в итоге? Здесь у нас действует следующая система приоритетов:

- Если у элемента определены встроенные стили (inline-стили), то они имеют высший приоритет, то есть в примере выше итоговой шириной будет 120 пикселей
- Далее в порядке приоритета идут стили, которые определены в элементе `style`
- Наименее приоритетными стилями являются те, которые определены во внешнем файле.

Атрибуты html и стили css

Многие элементы html позволяют устанавливать стили отображения с помощью атрибутов. Например, у ряда элементов мы можем применять атрибуты `width` и `height` для установки ширины и высоты элемента соответственно. Однако подобного подхода следует избегать и вместо встроенных атрибутов следует применять стили CSS. Важно четко понимать, что разметка HTML должна предоставлять только структуру html-документа, а весь его внешний вид, стилизацию должны определять стили CSS.

Валидация кода CSS

В процессе написания стилей CSS могут возникать вопросы, а правильно ли так определять стили, корректны ли они. И в этом случае мы можем воспользоваться валидатором css, который доступен по адресу http://jigsaw.w3.org/css-validator/.

Селекторы

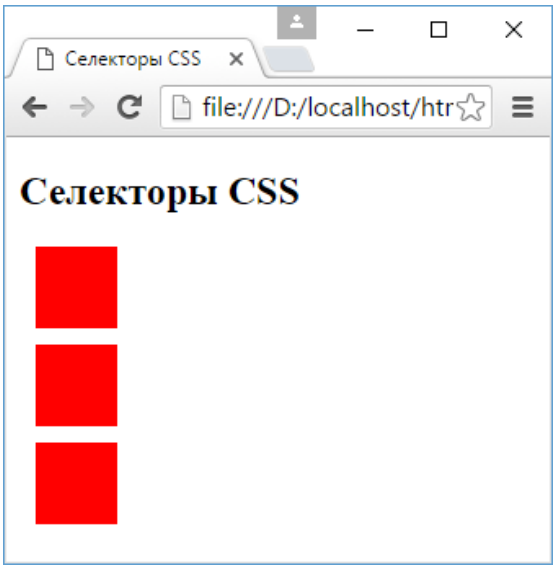
Определение стиля начинается с селектора. Например:

```
div{
    width:50px; /* ширина */
    height:50px; /* высота */
    background-color:red; /* цвет фона */
    margin: 10px; /* отступ от других элементов */
}
```

В данном случае селектором является `div`. Ряд селекторов наследуют название форматируемых элементов, например, div, p, h2 и т. д. При определении такого селектора его стиль будет применяться ко всем элементам соответствующих данному селектору. То есть выше определенный стиль будет применяться ко всем элементам `<div>` на веб-странице:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы CSS</title>
        <style>
            div{
                width:50px;
                height:50px;
                background-color:red;
                margin: 10px;
            }
        </style>
    </head>
    <body>
        <h2>Селекторы CSS</h2>
        <div></div>
        <div></div>
        <div></div>
    </body>
</html>
```

Здесь на странице 3 элемента div, и все они будут стилизованы:



Классы

Иногда для одних и тех же элементов требуется различная стилизация. И в этом случае мы можем использовать классы.

Для определения селектора класса в CSS перед названием класса ставится точка:

```
.redBlock{
    background-color:red;
}
```

Название класса может быть произвольным. Например, в данном случае название класса - "redBlock". Однако при этом в имени класса разрешается использовать буквы, числа, дефисы и знаки подчеркивания, причем начинать название класса должно обязательно с буквы.

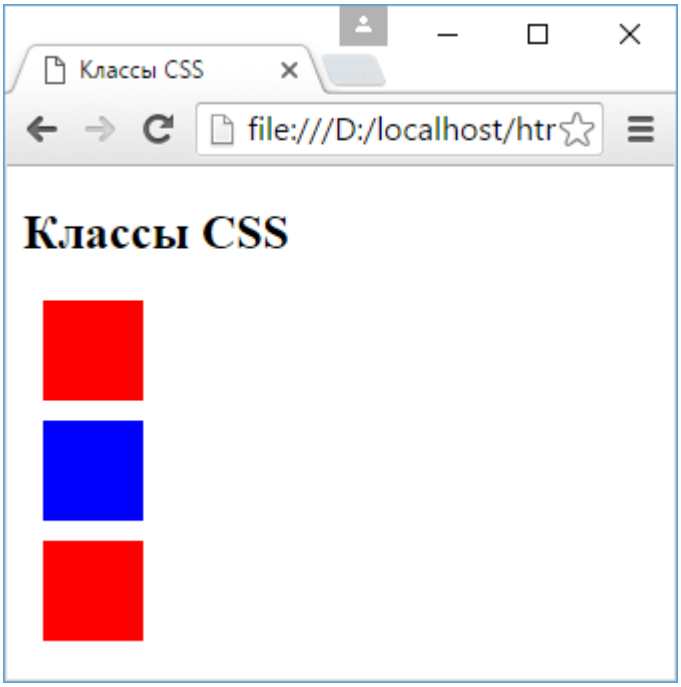
Также стоит учитывать регистр имен: названия "article" и "ARTICLE" будут представлять разные классы.

После определения класса мы можем его применить к элементу с помощью атрибута **class**. Например:

```
<div class="redBlock"></div>
```

Определим и используем несколько классов:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Классы CSS</title>
        <style>
            div{
                width: 50px;
                height: 50px;
                margin: 10px;
            }
            .redBlock{
                background-color: red;
            }
            .blueBlock{
                background-color: blue;
            }
        </style>
    </head>
    <body>
        <h2>Классы CSS</h2>
        <div class="redBlock"></div>
        <div class="blueBlock"></div>
        <div class="redBlock"></div>
    </body>
</html>
```



Идентификаторы

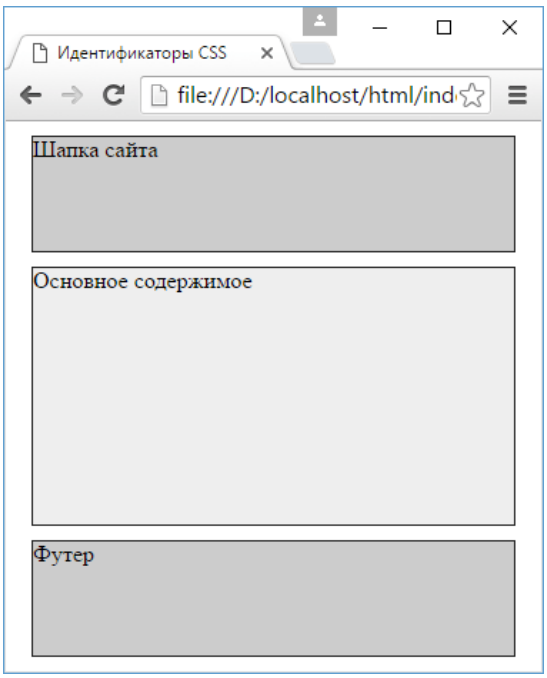
Для идентификации уникальных на веб-станице элементов используются идентификаторы, которые определяются с помощью атрибутов id. Например, на странице может быть головной блок или шапка:

```
<div id="header"></div>
```

Определение стилей для идентификаторов аналогично определению классов, только вместо точки ставится символ решетки #:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Идентификаторы CSS</title>
        <style>
            div{
                margin: 10px;
                border: 1px solid #222;
            }
            #header{
                height: 80px;
                background-color: #ccc;
            }
            #content{
                height: 180px;
                background-color: #eee;
            }
            #footer{
                height: 80px;
                background-color: #ccc;
            }
        </style>
    </head>
    <body>
```

```
        <div id="header">Шапка сайта</div>
        <div id="content">Основное содержимое</div>
        <div id="footer">Футер</div>
    </body>
</html>
```



Однако стоит заметить, что идентификаторы в большей степени относятся к структуре веб-странице и в меньшей степени к стилизации. Для стилизации преимущественно используются классы, нежели идентификаторы.

Универсальный селектор

Кроме селекторов тегов, классов и идентификаторов в css также есть так называемый **универсальный селектор**, который представляет знак звездочки (*). Он применяет стили ко всем элементам на html-странице:

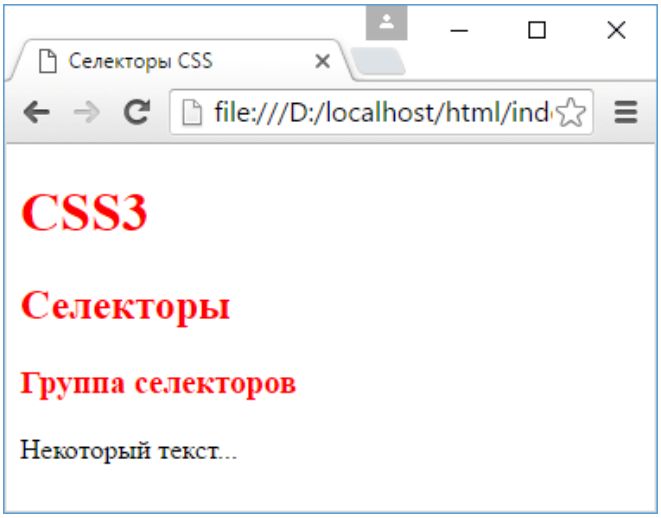
```
*{
    background-color: red;
}
```

Стилизация группы селекторов

Иногда определенные стили применяются к целому ряду селекторов. Например, мы хотим применить ко всем заголовкам подчеркивание. В этом случае мы можем перечислить селекторы всех элементов через запятую:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы CSS</title>
        <style>
            h1, h2, h3, h4{

                color: red;
            }
```

```
        </style>
    </head>
    <body>
        <h1>CSS3<h1>
        <h2>Селекторы</h2>
        <h3>Группа селекторов</h3>
        <p>Некоторый текст...</p>
    </body>
</html>
```



Группа селекторов может содержать как селекторы тегов, так и селекторы классов и идентификаторов, например:

```
h1, #header, .redBlock{

    color: red;
}
```

Селекторы потомков

Веб-страница может иметь сложную организацию, одни элементы внутри себя могут определять другие элементы. Вложенные элементы иначе можно назвать потомками. А контейнер этих элементов - родителем.

Например, пусть элемент body на веб-странице имеет следующее содержимое:

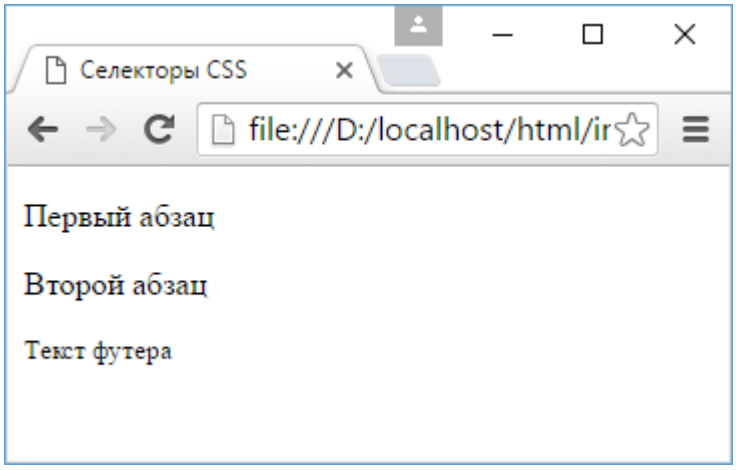
```
<body>
    <h2>Заголовок</h2>
    <div>
        <p>Текст</p>
    </div>
</body>
```

Внутри элемента body определено три вложенных элемента: h2, div, p. Все эти элемены являются потомками элемента body.

А внутри элемента div определен только один вложенный элемент - p, поэтому элемент div имеет только одного потомка.

Используя специальные селекторы, мы можем стилизовать вложенные элементы или потомков внутри строго определенных элементов. Например, у нас на странице могут быть параграфы внутри блока с основным содержимым и внутри блока футера. Но для параграфов внутри блока основного содержимого мы захотим установить один шрифт, а для параграфов футера другой.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы CSS</title>
        <style>
            #main p{
                font-size: 16px;
            }
            #footer p{
                font-size: 13px;
            }
        </style>
    </head>
    <body>
        <div id="main">
            <p>Первый абзац</p>
            <p>Второй абзац</p>
        </div>
        <div id="footer">
            <p>Текст футера</p>
        </div>
    </body>
</html>
```



Для применения стиля к вложенному элементу селектор должен содержать вначале родительский элемент и затем вложенный:

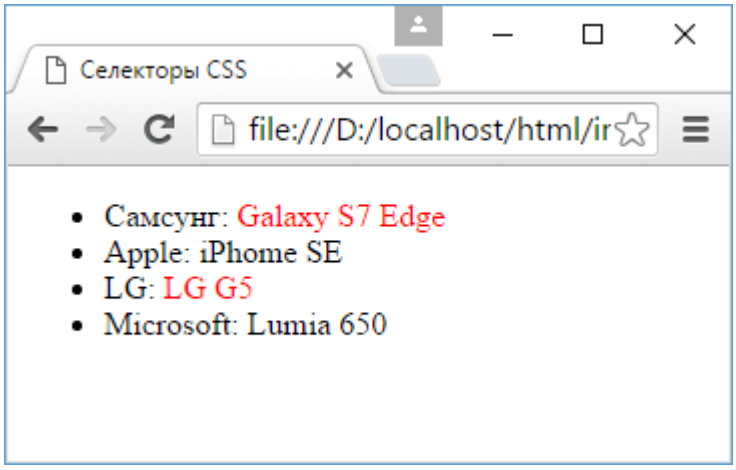
```
#main p{
    font-size: 16px;
}
```

То есть данный стиль будет применяться только к тем элементам p, которые находятся внутри элемента с идентификатором main.

Рассмотрим другой пример:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы CSS</title>
        <style>
            li .redLink{
                color: red;
            }
        </style>
    </head>
    <body>
        <ul>
            <li>Самсунг: <a class="redLink">Galaxy S7 Edge</a></li>
            <li>Apple: <a>iPhome SE</a></li>
            <li>LG: <a class="redLink">LG G5</a></li>
            <li>Microsoft: <a>Lumia 650</a></li>
        </ul>
    </body>
</html>
```

Здесь стиль применяется к элементам с классом "redLink", которые находятся внутри элемента `<li>`. И соответственно браузер окрасит эти элементы в красный цвет:



Но обратите внимание на пробел: `li .redLink`. Данный пробел играет большое значение и указывает как раз, что элементы с классом `redLink` должны быть вложенными по отношению к элементу `<li>`

Но если мы уберем пробел:

```
li.redLink{
    color: red;
}
```

то смысл селектора изменится. Теперь будет подразумеваться, что стиль применяется к элементам `<li>`, которые имеют класс `redLink`. Например, к следующему элементу:

```
<li class="redLink">Microsoft: <a>Lumia 650</a></li>
```

Но никак не к элементу:

```
<li>LG: <a class="redLink">LG G5</a></li>
```

Селекторы дочерних элементов

Селекторы дочерних элементов отличаются от селекторов потомков тем, что позволяют выбрать элементы только первого уровня вложенности. Например:

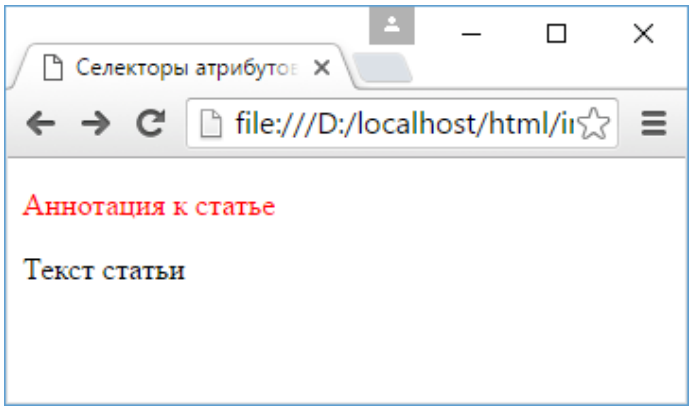
```
<body>
    <h2>Заголовок</h2>
    <div>
        <p>Текст</p>
    </div>
</body>
```

Хотя вложенными в элемент body элементами являются целых три - h2, div, p, но дочерними из них являются только два - div и h2, так как они находятся в первом уровне вложенности. А элемент p находится на втором уровне вложенности, так как вложен внутрь элемента div, а не просто элемента body.

Для обращения к дочерним элементам используется знак угловой скобки:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы атрибутов в CSS3</title>
        <style>
            .article > p{

                color: red;
            }
        </style>
    </head>
    <body>
        <div class="article">
            <p>Аннотация к статье</p>
            <div class="content">
                <p>Текст статьи</p>
            </div>
        </div>
    </body>
</html>
```

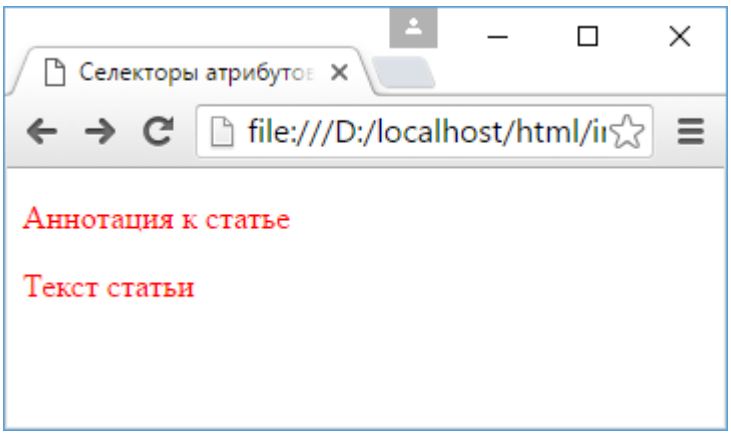
В блоке с классом article есть два параграфа. Селектор `.article > p` выбирает только те параграфы, который находятся непосредственно в блоке article:



Если бы мы использовали другой селектор без символа >

```
.article p{

    color: red;
}
```

Тогда стиль бы применялся ко всем параграфам на всех уровнях вложенности:



Селекторы элементов одного уровня

Селекторы элементов одного уровня или смежных элементов позволяют выбрать элементы, которые находятся на одном уровне вложенности. Иногда такие элементы еще называют сиблинги (siblings) или сестринскими элементами. Например:

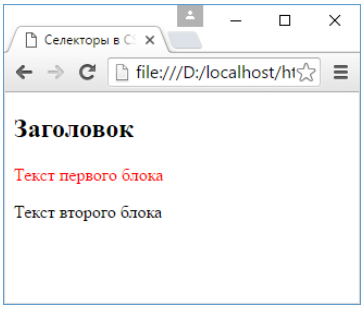
```
<body>
    <h2>Заголовок</h2>
    <div>
        <p>Текст первого блока</p>
    </div>
    <div>
        <p>Текст второго блока</p>
    </div>
</body>
```

Здесь элементы h2 и оба блока div являются смежными, так как находятся на одном уровне. А элементы параграфов и заголовок h2 не являются смежными, так как параграфы вложены в блоки div.

Чтобы стилизовать первый смежный элемент по отношению к определенному элементу, используется знак плюса +:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы в CSS3</title>
        <style>
            h2+div { color: red; }
        </style>
    </head>
    <body>
        <h2>Заголовок</h2>
        <div>
            <p>Текст первого блока</p>
        </div>
        <div>
            <p>Текст второго блока</p>
        </div>
    </body>
</html>
```

Селектор `h2+div` позволяет определить стиль (в данном случае красный цвет текста) для блока div, который идет непосредственно после заголовка h2:

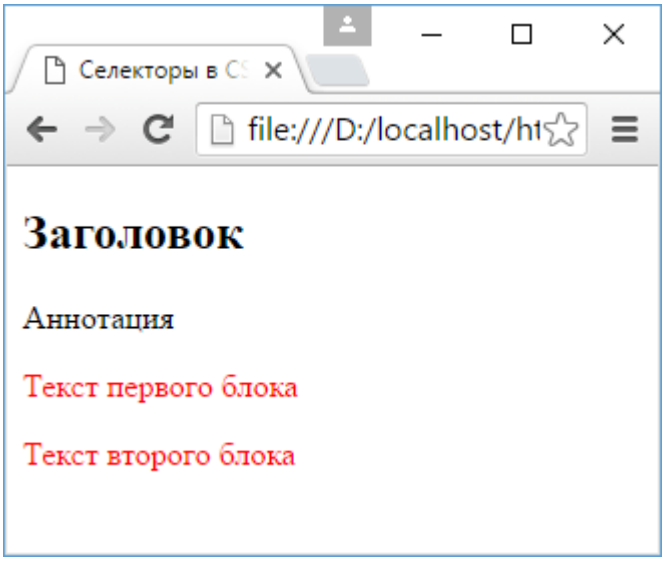


Причем этот селектор будет стилизовать блок div, если он будет идти непосредственно после заголовка. Если же между заголовком и блоков div будет находиться еще какой-либо элемент, то к нему не будет применяться стиль, например:

```
<h2>Заголовок</h2>

<p>Элемент между заголовком и блоком div</p>

<div>
    <p>Текст первого блока</p>
</div>
```

Если нам надо стилизовать вообще все смежные элементы одного уровня, неважно непосредственно идут они после определенного элемента или нет, то в этом случае вместо знака плюса необходимо использовать знак тильды "~":

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы в CSS3</title>
        <style>
            h2~div { color: red; }
        </style>
    </head>
    <body>
        <h2>Заголовок</h2>
        <p>Аннотация</p>
        <div>
            <p>Текст первого блока</p>
        </div>
        <div>
            <p>Текст второго блока</p>
        </div>
    </body>
</html>
```



Псевдоклассы

В дополнение к селекторам тегов, классов и идентификаторов нам доступны селекторы псевдоклассов, которые несут дополнительные возможности по выбору нужных элементов.

Список доступных псевдоклассов:

- **:root**: позволяет выбрать корневой элемент веб-страницы, наверное наименее полезный селектор, так как на правильной веб-странице корневым элементом практически всегда является элемент `<html>`
- **:link**: применяется к ссылкам и представляет ссылку в обычном состоянии, по которой еще не совершен переход
- **:visited**: применяется к ссылкам и представляет ссылку, по которой пользователь уже переходил
- **:active**: применяется к ссылкам и представляет ссылку в тот момент, когда пользователь осуществляет по ней переход
- **:hover**: представляет элемент, на который пользователь навел указатель мыши. Применяется преимущественно к ссылкам, однако может также применяться и к другим элементам, например, к параграфам
- **:focus**: представляет элемент, который получает фокус, то есть когда пользователь нажимает клавишу табуляции или нажимает кнопкой мыши на поле ввода (например, текстовое поле)
- **:not**: позволяет исключить элементы из списка элементов, к которым применяется стиль
- **:lang**: стилизует элементы на основании значения атрибута `lang`
- **:empty**: выбирает элементы, которые не имеют вложенных элементов, то есть являются пустыми

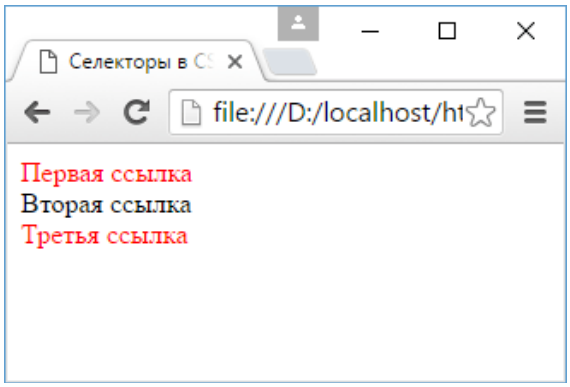
При применении псевдоклассов перед ними всегда ставится двоеточие. Например, стилизуем ссылки, используя псевдоклассы:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Псевдоклассы в CSS3</title>
        <style>
            a:link    {color:blue; text-decoration:none}
            a:visited {color:pink; text-decoration:none}
            a:hover   {color:red; text-decoration:underline}
            a:active  {color:yellow; text-decoration:underline}
            input:hover {border:2px solid red;}
        </style>
    </head>
    <body>
        <a href="index.html">Учебник по CSS3</a>
        <input type="text" />
    </body>
</html>
```

Селектор :not

Селектор **:not()** позволяет выбрать все элементы кроме определенных, то есть исключить некоторые элементы из выбора.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы в CSS3</title>
        <style>
            a:not(.blueLink) { color: red; }
        </style>
    </head>
    <body>
        <a>Первая ссылка</a><br/>
        <a class="blueLink">Вторая ссылка</a><br/>
        <a>Третья ссылка</a>
    </body>
</html>
```



Селектор `a:not(.blueLink)` применяет стиль ко всем ссылкам за исключением тех, которые имеют класс "blueLink". В скобки псевдоклассу not передается селектор элементов, которые надо исключить.

Псевдокласс :lang

Селектор **:lang** выбирает элементы на основании атрибута `lang`:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы в CSS3</title>
        <style>
            :lang(ru) {
                color: red;
            }
        </style>
    </head>
    <body>
        <form>
            <p lang="ru-RU">Я изучаю CSS3</p>
            <p lang="en-US">I study CSS3</p>
            <p lang="de-DE">Ich lerne CSS3</p>
        </form>
    </body>
</html>
```

Псевдоклассы дочерних элементов

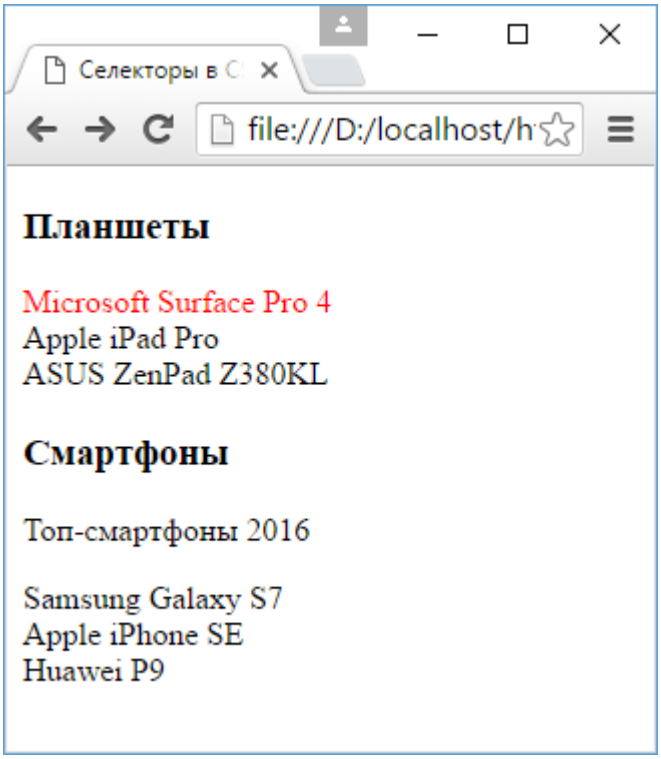
Особую группу псевдоклассов образуют псевдоклассы, которые позволяют выбрать определенные дочерние элементы:

- **:first-child**: представляет элемент, который является первым дочерним элементом
- **:last-child**: представляет элемент, который является последним дочерним элементом
- **:only-child**: представляет элемент, который является единственным дочерним элементом в каком-нибудь контейнере
- **:only-of-type**: выбирает элемент, который является единственным элементом определенного типа (тега) в каком-нибудь контейнере
- **:nth-child(n)**: представляет дочерний элемент, который имеет определенный номер n, например, второй дочерний элемент
- **:nth-last-child(n)**: представляет дочерний элемент, который имеет определенный номер n, начиная с конца
- **:nth-of-type(n)**: выбирает дочерний элемент определенного типа, который имеет определенный номер
- **:nth-last-of-type(n)**: выбирает дочерний элемент определенного типа, который имеет определенный номер, начиная с конца

Псевдокласс first-child

Используем псевдокласс first-child для выбора первых ссылок в блоках:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы в CSS3</title>
        <style>
            a:first-child{

                color: red;
            }
        </style>
    </head>
    <body>
        <h3>Планшеты</h3>
        <div>
            <a>Microsoft Surface Pro 4</a><br/>
            <a>Apple iPad Pro</a><br/>
            <a>ASUS ZenPad Z380KL</a>
        </div>
        <h3>Смартфоны</h3>
        <div>
            <p>Топ-смартфоны 2016</p>
            <a>Samsung Galaxy S7</a><br/>
            <a>Apple iPhone SE</a><br/>
            <a>Huawei P9</a>
        </div>
    </body>
</html>
```



Стиль по селектору `a:first-child` применяется к ссылке, если она является первым дочерним элементом любого элемента.

В первом блоке элемент ссылки является первым дочерним элементом, поэтому к нему применяется определенный стиль.

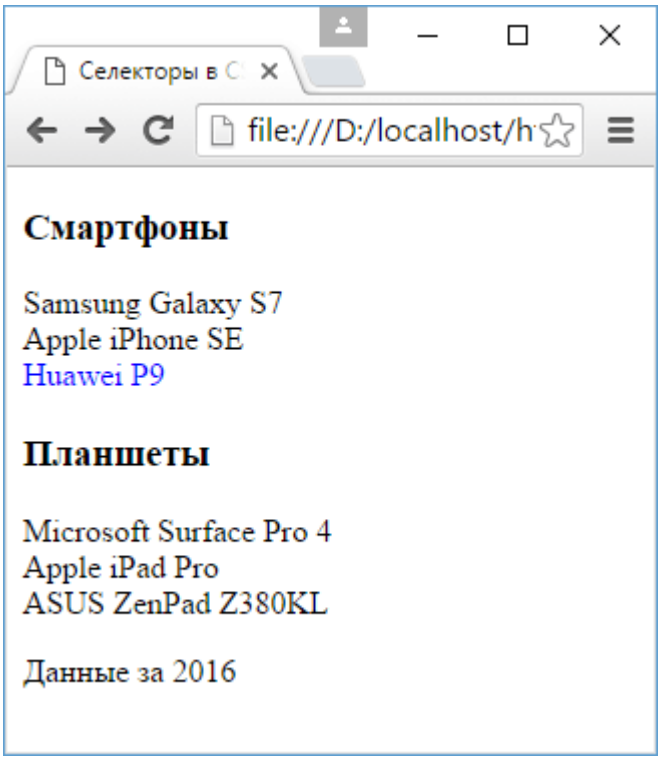
А во втором блоке первым элементом является параграф, поэтому ни к одной ссылке не применяется стиль.

Псевдокласс last-child

Используем псевдокласс `last-child`:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы в CSS3</title>
        <style>
            a:last-child{

                color: blue;
            }
        </style>
    </head>
    <body>
        <h3>Смартфоны</h3>
        <div>
            <a>Samsung Galaxy S7</a><br/>
            <a>Apple iPhone SE</a><br/>
            <a>Huawei P9</a>
        </div>
        <h3>Планшеты</h3>
        <div>
```

```
            <a>Microsoft Surface Pro 4</a><br/>
            <a>Apple iPad Pro</a><br/>
            <a>ASUS ZenPad Z380KL</a>
            <p>Данные за 2016</p>
        </div>
    </body>
</html>
```



Селектор `a:last-child` определяет стиль для ссылок, которые являются последними дочерними элементами.

В первом блоке как раз последним дочерним элементом является ссылка. А вот во втором последним дочерним элементом является параграф, поэтому во втором блоке стиль не применяется ни к одной из ссылок.

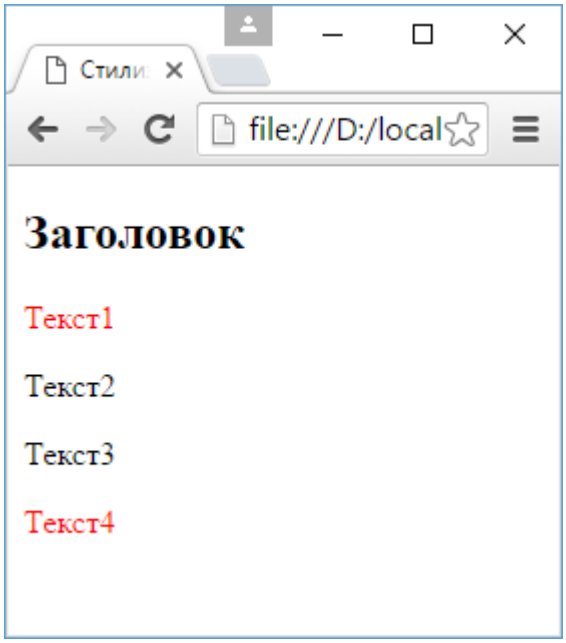
Селектор only-child

Селектор **:only-child** выбирает элементы, которые являются единственными дочерними элементами в контейнерах:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы в CSS3</title>
        <style>
        p:only-child{
            color:red;
        }
        </style>
    </head>
    <body>
        <h2>Заголовок</h2>
        <div>
```

```
            <p>Текст1</p>
        </div>
        <div>
            <p>Текст2</p>
            <p>Текст3</p>
        </div>
        <div>
            <p>Текст4</p>
        </div>
    </body>
</html>
```

Параграфы с текстами "Текст1" и "Текст4" являются единственными дочерними элементами в своих внешних контейнерах, поэтому к ним применяется стиль - красный цвет шрифта.



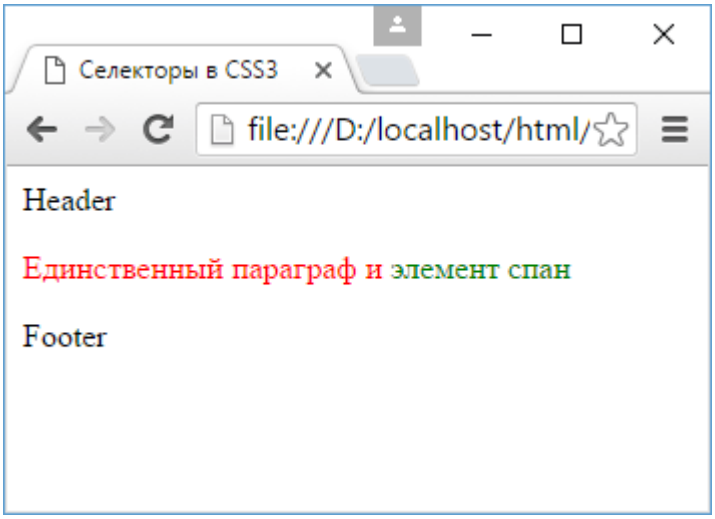
Псевдокласс only-of-type

Псевдокласс only-of-type выбирает элемент, который является единственным элементом определенного типа в контейнере. Например, единственный элемент div, при этом элементов других типов в этом же контейнере может быть сколько угодно.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы в CSS3</title>
        <style>
        span:only-of-type{

            color: green;   /* зеленый цвет */
        }
        p:only-of-type{

            color: red; /* красный цвет */
        }
        div:only-of-type{

```

```
            color: blue;    /* синий цвет */
        }
        </style>
    </head>
    <body>
        <div>
            Header
        </div>
        <p>Единственный параграф и <span>элемент спан</span></p>
        <div>
            Footer
        </div>
    </body>
</html>
```

Хотя для элементов div определен стиль, он не будет применяться, так как в контейнере body находится два элемента div, а не один. Зато в body есть только один элемент p, поэтому он получит стилизацию. И также в контейнере p есть только один элемент span, поэтому он также будет стилизован.



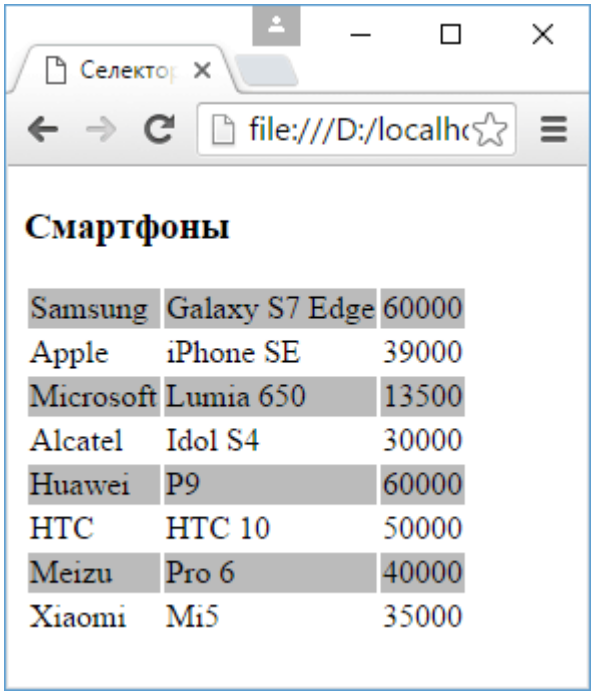
Псевдокласс nth-child

Псевдокласс **nth-child** позволяет стилизовать каждый второй, третий элемент, только четные или только нечетные элементы и т.д.

Например, стилизуем четные и нечетные строки таблицы:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы в CSS3</title>
        <style>
            tr:nth-child(odd) { background-color: #bbb; }
            tr:nth-child(even) { background-color: #fff; }
        </style>
    </head>
    <body>
        <h3>Смартфоны</h3>
        <table>
```

```
            <tr><td>Samsung</td><td>Galaxy S7 Edge</td><td>60000</td></tr>
            <tr><td>Apple</td><td>iPhone SE</td><td>39000</td></tr>
            <tr><td>Microsoft</td><td>Lumia 650</td><td>13500</td></tr>
            <tr><td>Alcatel</td><td>Idol S4</td><td>30000</td></tr>
            <tr><td>Huawei</td><td>P9</td><td>60000</td></tr>
            <tr><td>HTC</td><td>HTC 10</td><td>50000</td></tr>
            <tr><td>Meizu</td><td>Pro 6</td><td>40000</td></tr>
            <tr><td>Xiaomi</td><td>Mi5</td><td>35000</td></tr>
        </table>
    </body>
</html>
```



Чтобы определить стиль для нечетных элементов, в селектор передается значение "odd":

```
tr:nth-child(odd){}
```

Для стилизации четных элементов в селектор передается значение "even":

```
tr:nth-child(even){}
```

Также в этот селектор мы можем передать номер стилизуемого элемента:

```
tr:nth-child(3) { background-color: #bbb; }
```

В данном случае стилизуется третья строка.

Еще одну возможность представляет использование заменителя для номера, который выражается буквой **n**:

```
tr:nth-child(2n+1) { background-color: #bbb; }
```

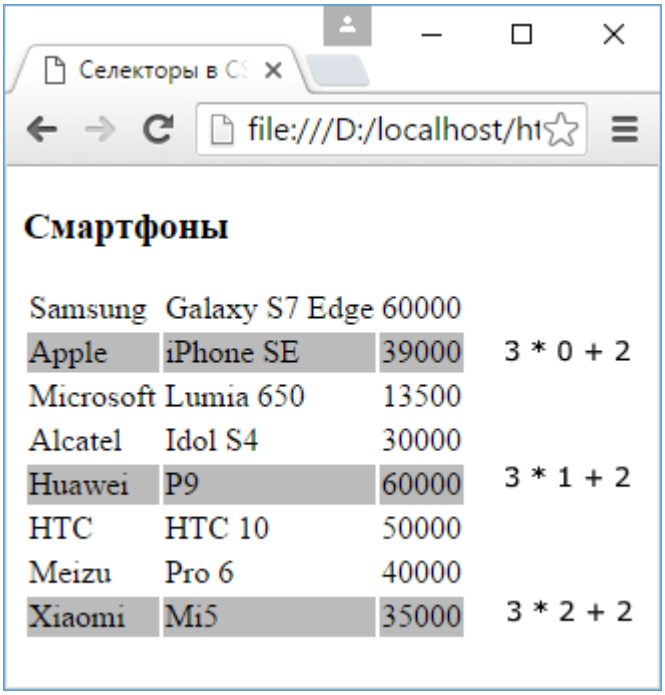
Здесь стиль применяется к каждой второй нечетной строке.

Число перед n (в данном случае 2) представляет тот дочерний элемент, который будет выделен следующим. Число, которое идет после знака плюс, показывают, с какого элемента нужно начинать выделение, то есть, +1 означает, что нужно начинать с первого дочернего элемента.

Таким образом, в данном случае выделение начинается с 1-го элемента, а следующим выделяется 2 * 1 + 1 = 3-й элемент, далее 2 * 2 + 1 = 5-й элемент и так далее.

К примеру, если мы хотим выделить каждый третий элемент, начиная со второго, то мы могли бы написать:

```
tr:nth-child(3n+2) { background-color: #bbb; }
```



Псевдокласс **:nth-last-child** по сути предоставляет ту же самую функциональность, только отсчет элементов идет не с начала, а с конца:

```
tr:nth-last-child(2) {
    background-color: #bbb; /* 2 строка с конца, то есть предпоследняя  */
}
tr:nth-last-child(2n+1) {
    background-color: #eee; /* нечетные строки, начиная с конца  */
}
```

Псевдокласс nth-of-type

Псевдокласс **:nth-of-type** позволяет выбрать дочерний элемент определенного типа по определенному номеру:

```
tr:nth-of-type(2) {
    background-color: #bbb;
}
```

Аналогично работает псевдокласс **nth-last-of-type**, только теперь отсчет элементов идет с конца:

```
tr:nth-last-of-type(2n) {
    background-color: #bbb;
}
```

Псевдоклассы форм

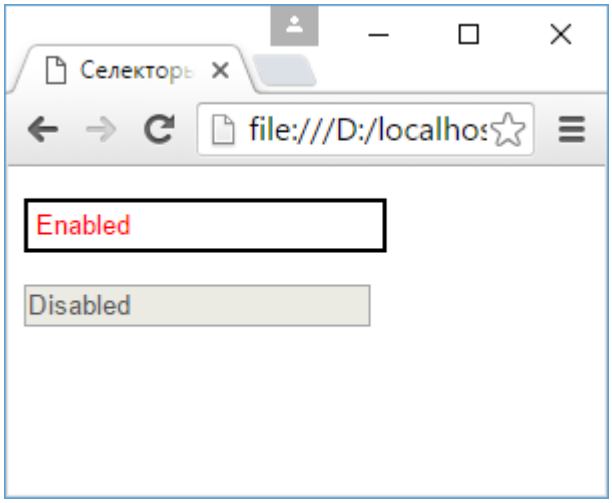
Ряд псевдоклассов используется для работы с элементами форм:

- **:enabled**: выбирает элемент, если он доступен для выбора (то есть у него не установлен атрибут disabled)
- **:disabled**: выбирает элемент, если он не доступен для выбора (то есть у него установлен атрибут disabled)
- **:checked**: выбирает элемент, если у него установлен атрибут checked (для флажков и радиокнопок)
- **:default**: выбирает элементы по умолчанию
- **:valid**: выбирает элемент, если его значение проходит валидацию HTML5
- **:invalid**: выбирает элемент, если его значение не проходит валидацию
- **:in-range**: выбирает элемент, если его значение находится в определенном диапазоне (для элементов типа ползунка)
- **:out-of-range**: выбирает элемент, если его значение не находится в определенном диапазоне
- **:required**: выбирает элемент, если у него установлен атрибут required
- **:optional**: выбирает элемент, если у него не установлен атрибут required

Псевдоклассы enabled и disabled

Псевдоклассы enabled и disabled выбирают элементы форм в зависимости от того, установлен ли у них атрибут **disabled**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы в CSS3</title>
        <style>
            :enabled {
                border: 2px black solid;
                color: red;
            }
        </style>
    </head>
    <body>
        <p><input type="text" value="Enabled" /></p>
        <p><input type="text" disabled value="Disabled" /></p>
    </body>
</html>
```

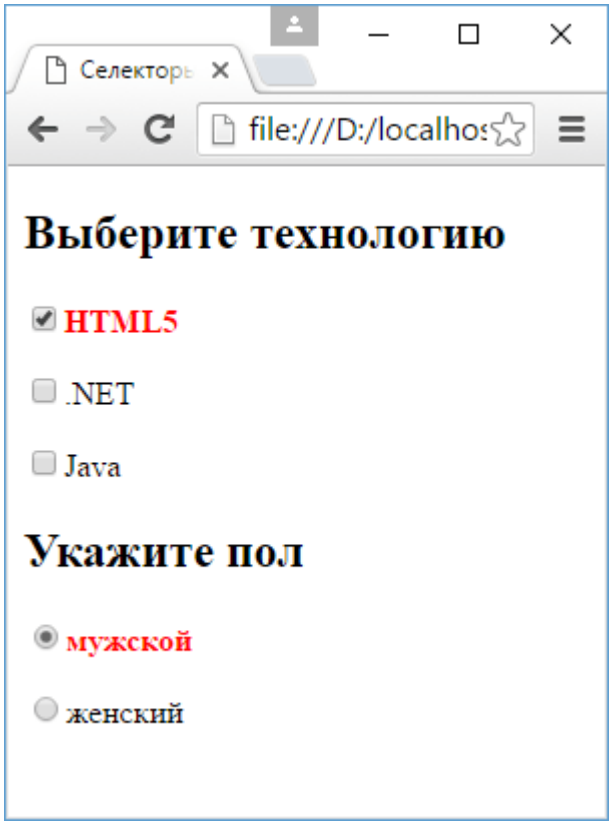


Псевдокласс checked

Псевдокласс **checked** стилизует элементы формы, у которых установлен атрибут `checked`:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы в CSS3</title>
        <style>
            :checked + span {
                color: red;
                font-weight: bold;  /* выделение жирным */
            }
        </style>
    </head>
    <body>
            <h2>Выберите технологию</h2>
            <p>
                <input type="checkbox" checked name="html5"/><span>HTML5</span>
            </p>
            <p>
                <input type="checkbox" name="dotnet"/><span>.NET</span>
            </p>
            <p>
                <input type="checkbox" name="java"/><span>Java</span>
            </p>

            <h2>Укажите пол</h2>
            <p>
                <input type="radio" value="man" checked name="gender"/><span>мужской
                </span>
            </p>
            <p>
                <input type="radio" value="woman" name="gender"/><span>женский</span>
            </p>
    </body>
</html>
```

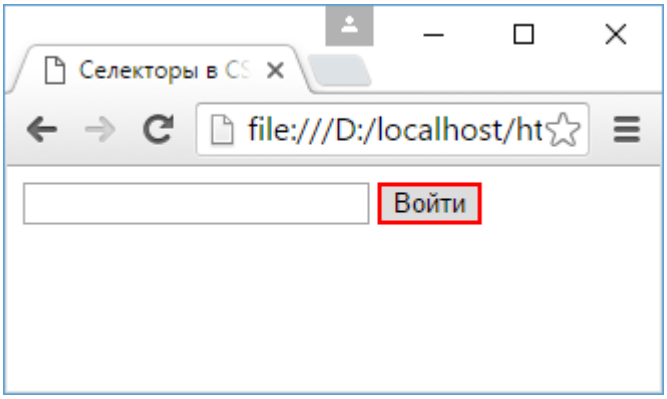
Селектор `:checked + span` позволяет выбрать элемент, соседний с отмеченным элементом формы.



Псевдокласс default

Псевдокласс **:default** выбирает стандартный элемент на форме. Как правило, в роли такого элемента выступает кнопка отправки:

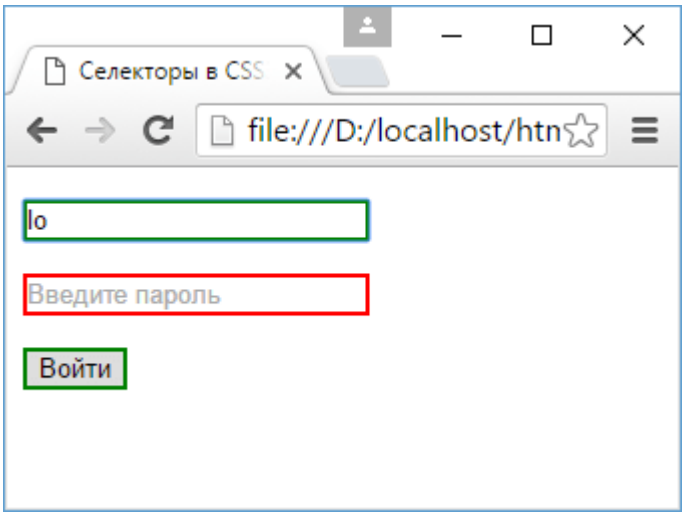
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы в CSS3</title>
        <style>
            :default {
                border: 2px solid red;
            }
        </style>
    </head>
    <body>
        <form>
            <input name="login"/>
            <input type="submit" value="Войти" />
        </form>
    </body>
</html>
```



Псевдоклассы valid и invalid

Псевдоклассы **:valid** и **:invalid** стилизуют элементы формы в зависимости от того, проходят они валидацию или нет.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы в CSS3</title>
        <style>
            input:invalid {
                border: 2px solid red;
            }
            input:valid {
                border: 2px solid green;
            }
        </style>
    </head>
    <body>
        <form>
            <p><input type="text" name="login" placeholder="Введите логин" required />
            </p>
            <p><input type="password" name="password" placeholder="Введите пароль"
            required /></p>
            <input type="submit" value="Войти" />
        </form>
    </body>
</html>
```

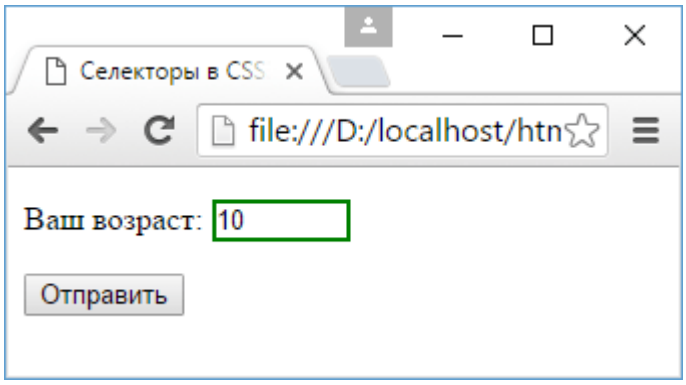


Псевдоклассы in-range и out-of-range

Псевдоклассы **:in-range** и **:out-of-range** стилизуют элементы формы в зависимости от того, попадает ли их значение в определенный диапазон. Это в первую очередь относится к элементу `<input type="number" >`.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы в CSS3</title>
        <style>
            :in-range {
                border: 2px solid green;
            }
            :out-of-range {
                border: 2px solid red;
            }
        </style>
    </head>
    <body>
        <form>
            <p>
                <label for="age">Ваш возраст:</label>
                <input type="number" min="1" max="100" value="10" id="age" name="age"/>
            </p>
            <input type="submit" value="Отправить" />
        </form>
    </body>
</html>
```

Здесь атрибуты `min` и `max` задают диапазон, в которое должно попадать вводимое в поле значение:

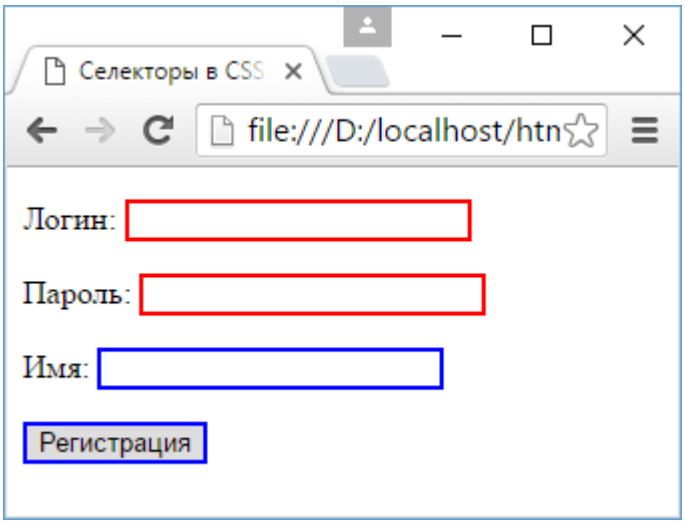


Псевдоклассы required и optional

Псевдоклассы **:required** и **:optional** стилизуют элемент в зависимости от того, установлен ли у него атрибут `required`:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы в CSS3</title>
```

```
6           <style>
7               :optional {
8                   border: 2px solid blue;
9               }
10              :required {
11                  border: 2px solid red;
12              }
13          </style>
14      </head>
15      <body>
16          <form>
17              <p>
18                  <label for="login">Логин:</label>
19                  <input type="text" id="login" name="login" required />
20              </p>
21              <p>
22                  <label for="password">Пароль:</label>
23                  <input type="password" id="password" name="password" required />
24              </p>
25              <p>
26                  <label for="name">Имя:</label>
27                  <input type="text" id="name" name="name"/>
28              </p>
29              <input type="submit" value="Регистрация" />
30          </form>
31      </body>
32  </html>
```



Псевдоэлементы

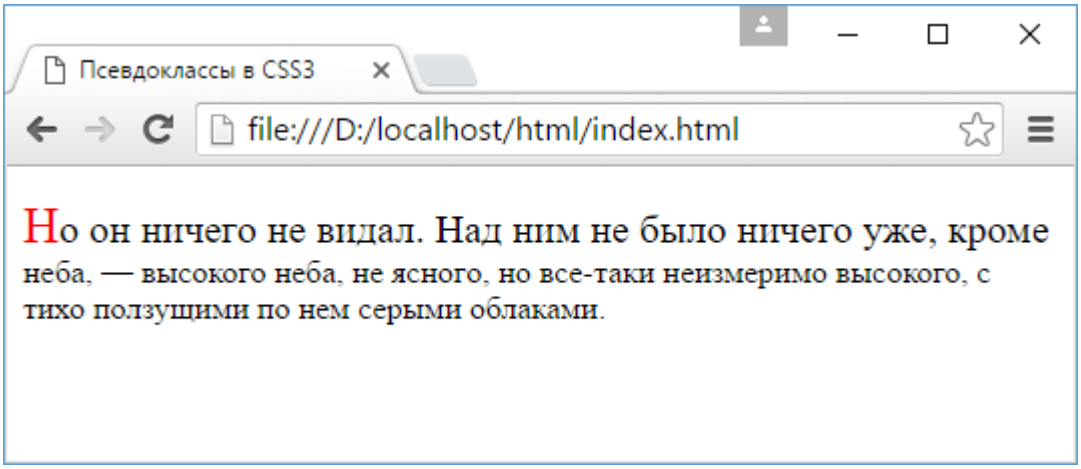
Псевдоэлементы обладают рядом дополнительных возможностей по выбору элементов веб-страницы и похожи на псевдоклассы. Список доступных псевдоэлементов:

- **::first-letter**: позволяет выбрать первую букву из текста
- **::first-line**: стилизует первую строку текста
- **::before**: добавляет сообщение до определенного элемента
- **::after**: добавляет сообщение после определенного элемента
- **::selection**: выбирает выбранные пользователем элементы

В CSS2 перед псевдоэлементами, как и перед псевдоклассами, ставилось одно двоеточие. В CSS3 для отличия их от псевдоклассов псевдоэлементы стали предваряться двумя двоеточиями. Однако для совместимости с более старыми браузерами, которые не поддерживают CSS3, допустимо использование одного двоеточия: `:before`.

Стилизуем текст, используя псевдоэлементы `first-letter` и `first-line`:

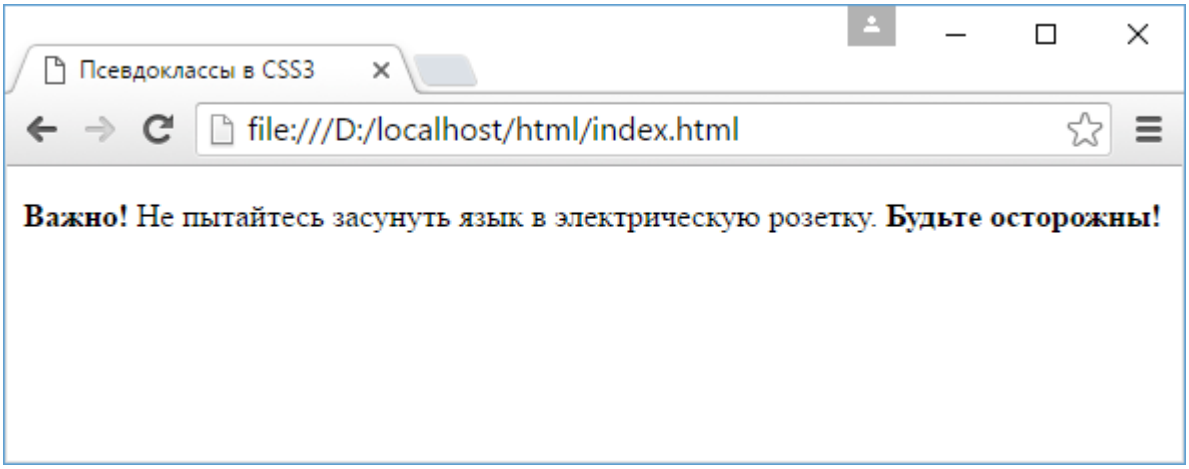
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Псевдоклассы в CSS3</title>
        <style>
            ::first-letter { color:red; font-size: 25px; }
            ::first-line { font-size: 20px; }
        </style>
    </head>
    <body>
        <p>Но он ничего не видал. Над ним не было ничего уже, кроме неба, — высокого неба, не ясного, но все-таки неизмеримо высокого, с тихо ползущими по нем серыми облаками.</p>
    </body>
</html>
```



Используем псевдоэлементы `before` и `after`:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
```

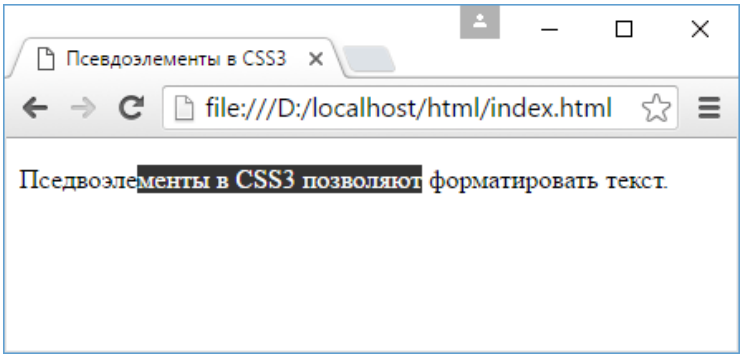
```
        <title>Псевдоклассы в CSS3</title>
        <style>
            .warning::before{ content: "Важно! "; font-weight: bold; }
            .warning::after { content: " Будьте осторожны!"; font-weight: bold;}
        </style>
    </head>
    <body>
        <p><span class="warning">Не пытайтесь засунуть язык в электрическую розетку.
        </span></p>
    </body>
</html>
```



Здесь псевдоэлеметы применяются к элементу с классом `warning`. Оба псевдоэлемента принимают свойство **content**, которое хранит вставляемый текст. И также для повышения внимания псевдоэлементы используют выделение текста жирным с помощью свойства `font-weight: bold;`.

Используем псевдоэлемент `selection` для стилизации выбранных элементов:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Псевдоэлементы в CSS3</title>
        <style>
            ::selection {
                color: white;
                background-color: black;
            }
        </style>
    </head>
    <body>
        <p>Пседвоэлементы в CSS3 позволяют форматировать текст.</p>
    </body>
</html>
```



Селекторы атрибутов

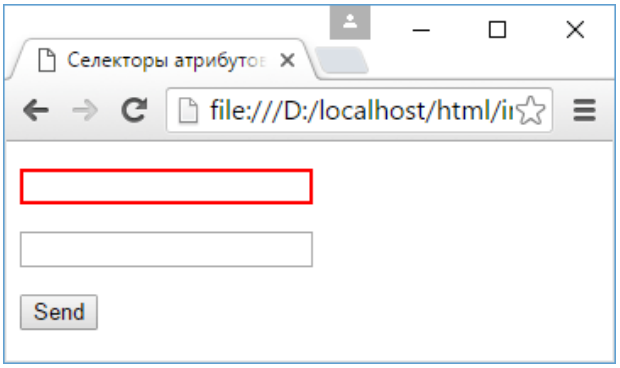
Кроме селекторов элементов мы также можем использовать селекторы их атрибутов. Например, у нас есть на веб-странице несколько полей input, а нам надо окрасить в красный цвет только текстовые поля. В этом случае мы как раз можем проверять значение атрибута `type`: если оно имеет значение `text`, то это текстовое поле, и соответственно его надо окрасить в красный цвет. Определение стиля в этом случае выглядело бы так:

```
input[type="text"]{

    border: 2px solid red;
}
```

После элемента в квадратных скобках идет атрибут и его значение. То есть в данном случае мы говорим, что для текстового поля надо установить границу красного цвета 2 пикселя толщиной сплошной линией.

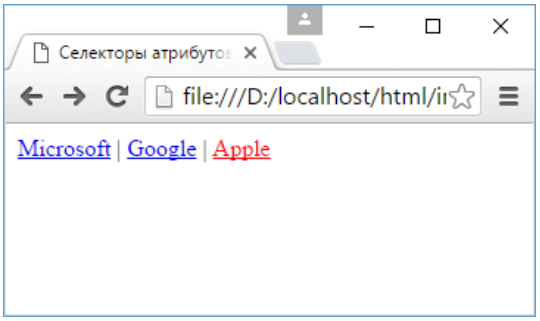
Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы атрибутов в CSS3</title>
        <style>
            input[type="text"]{
                border: 2px solid red;
            }
        </style>
    </head>
    <body>
        <p><input type="text" id="login" /></p>
        <p><input type="password" id="password" /></p>
        <input type="submit" value="Send" />
    </body>
</html>
```



Селекторы атрибутов можно применять не только к элементам, но и классам и идентификаторам. Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Селекторы атрибутов в CSS3</title>
        <style>
            .link[href="http://apple.com"]{

                color: red;
            }
        </style>
    </head>
    <body>
        <a class="link" href="http://microsoft.com">Microsoft</a> |
        <a class="link" href="https://google.com">Google</a> |
        <a class="link" href="http://apple.com">Apple</a>
    </body>
</html>
```



Специальные символы позволяют конкретизировать значение атрибутов. Например символ ^ позволяет выбрать все атрибуты, которые начинаются на определенный текст. Например, нам надо выбрать все ссылки, которые используют протокол https, то есть ссылка должна начинаться на "https://". В этом случае можно применить следующий селектор:

```
a[href^="https://"]{

    color: red;
}
```

Если значение атрибута должно иметь в конце определенный текст, то для проверки используется символ $. Например, нам надо выбрать все изображения в формате jpg. В этом случае мы можем проверить, оканчивается ли значение атрибута `src` на текст ".jpg":

```
img[src$=".jpg"]{

    width: 100px;
}
```

И еще один символ "*" (звездочка) позволяет выбрать все элементы с атрибутами, которые в своем значении имеют определенный текст (не важно где - в начале, середине или конце):

```
a[href*="microsoft"]{

    color: red;
}
```

Данный атрибут выберет все ссылки, которые в своем адресе имеют текст "microsoft".

Наследование стилей

Для упрощения определения стилей в CSS применяется механизм наследования стилей. Этот механизм предполагает, что вложенные элементы могут наследовать стили своих элементов-контейнеров. Например, пусть у нас на веб-странице есть заголовок и параграф, которые должны иметь текст красного цвета. Мы можем отдельно к параграфу и заголовку применить соответствующий стиль, который установит нужный цвет шрифта:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Наследование стилей в CSS3</title>
        <style>
            p {color: red;}
            h2 {color: red;}
        </style>
    </head>
    <body>
        <h2>Наследование стилей</h2>
        <p>Текст про наследование стилей в CSS 3</p>
    </body>
</html>
```

Однако поскольку и элемент p, и элемент h2 находятся в элементе body, то они наследуют от этого контейнера - элемента body многие стили. И чтобы не дублировать определение стиля, мы могли бы написать так:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Наследование стилей в CSS3</title>
        <style>
            body {color: red;}
```

```
            </style>
        </head>
        <body>
            <h2>Наследование стилей</h2>
            <p>Текст про наследование стилей в CSS 3</p>
        </body>
    </html>
```

В итоге определение стилей стало проще, а результат остался тем же.

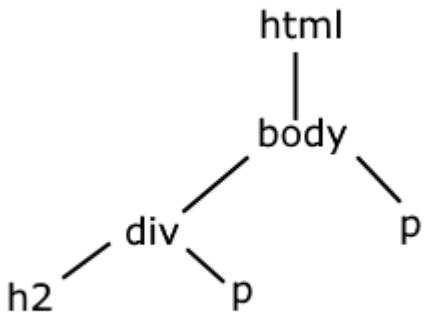
Если нам нежелателен унаследованный стиль, то мы его можем переопределить для определенных элементов:

```
body {color: red;}
p {color: green;}
```

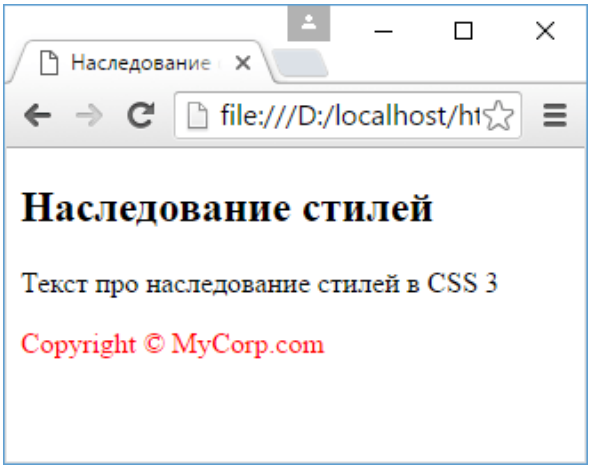
При нескольких уровнях вложенности элементы наследуют стили только ближайшего контейнера:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Наследование стилей в CSS3</title>
        <style>
            body {color: red;}
            div {color:black;}
        </style>
    </head>
    <body>
        <div>
            <h2>Наследование стилей</h2>
            <p>Текст про наследование стилей в CSS 3</p>
        </div>
        <p>Copyright © MyCorp.com</p>
    </body>
</html>
```

Здесь веб-страница имеет следующую структуру:



Для элемента div переопределяется цвет текста. И так как элемент h2 и один из параграфов находятся в элементе div, то они наследуют стиль именно элемента div. Второй параграф находится непосредственно в элементе body и поэтому наследует стиль элемента body.



Однако не ко всем свойствам CSS применяется наследование стилей. Например, свойства, которые представляют отступы (margin, padding) и границы (border) элементов, не наследуются.

Кроме того, браузеры по умолчанию также применяют ряд предустановленных стилей к элементам. Например, заголовки имеют определенную высоту и т.д.

Каскадность стилей

Когда к определенному элементу применяется один стиль, то все относительно просто. Однако если же к одному и тому же элементу применяется сразу несколько различных стилей, то возникает вопрос, какой же из этих стилей будет в реальности применяться?

В CSS действует механизм каскадности, которую можно определить как набор правил, определяющих последовательность применения множества стилей к одному и тому же элементу.

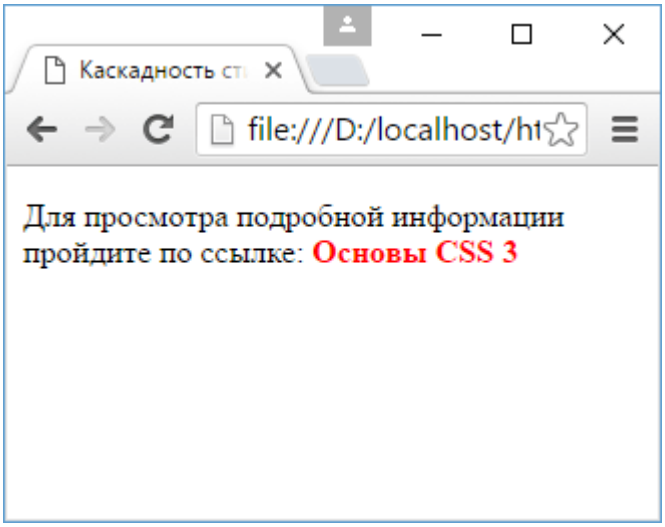
К примеру, у нас определена следующая веб-страница:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Каскадность стилей в CSS3</title>
        <style>
            .redLink {color: red;}  /* красный цет текста */
            .footer a {font-weight: bold;}  /* выделение жирным */
            a {text-decoration: none;}  /* отмена подчеркивания ссылки */
        </style>
    </head>
    <body>
        <p class="footer">Для просмотра подробной информации пройдите по ссылке:
            <a class="redLink" href="index.php">Основы CSS 3</a></p>
    </body>
</html>
```

В CSS определено три стиля и все они применяются к ссылке.

Если к элементу веб-страницы применяется несколько стилей, которые не конфликтуют между собой, то браузер объединяет их в один стиль.

Так, в данном случае, все три стиля не конфликтуют между собой, поэтому все эти стили будут применяться к ссылке:



Если же стили конфликтуют между собой, например, определяют разный цвет текста, то в этом случае применяется сложная система правил для вычисления значимости каждого стиля. Все эти правила описаны в спецификации по CSS: Calculating a selectors specificity. Вкратце разберем их.

Для определения стиля к элементу могут применяться различные селекторы, и важность каждого селектора оценивается в баллах. Чем больше у селектора пунктов, тем он важнее, и тем больший приоритет его стили имеют над стилями других селекторов.

- Селекторы тегов имеют важность, оцениваемую в 1 балл
- Селекторы классов, атрибутов и псевдоклассов оцениваются в 10 баллов
- Селекторы идентификаторов оцениваются в 100 баллов
- Встроенные inline-стили (задаваемые через атрибут `style`) оцениваются в 1000 баллов

Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Каскадность стилей в CSS3</title>
        <style>
            #index {color: navy;}   /* темно-синий цет текста */
            .redLink {color: red; font-size: 20px;} /* красный цет текста и высота шрифта 20 пикселей */
            a {color: black; font-weight: bold;}    /* черный цет текста и выделение жирным */
        </style>
    </head>
    <body>
        <a id="index" class="redLink" href="index.php">Основы CSS 3</a>
    </body>
</html>
```

Здесь к ссылке применяется сразу три стиля. Эти стили содержат два не конфликтующих правила:

```
font-size: 20px;
font-weight: bold;
```

которые устанавливают высоту шрифта 20 пикселей и выделение ссылки жирным. Так как каждое из эти правил определено только в одном стиле, то в итоге они будут суммироваться и применяться к ссылке без проблем.

Кроме того, все три стиля содержат определение цвета текста, но каждый стиль определяет свой цвет текста. Так как селекторы идентификаторов имеют больший удельный вес, то в конечном счете будет применяться темно-синий цвет, задаваемый селектором:

```
#index {color: navy;}
```

Если селектор является составным, то происходит сложение баллов всех входящих в селектор подселекторов. Так, рассмотрим следующий пример:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Каскадность стилей в CSS3</title>
        <style>
            a {font-size: 18px;}
            .nav li a {color: red;}  /* красный цет текста */
            #menu a {color: navy;}   /* темно-синий цет текста */
            .nav .menuItem {color: green;}  /* зеленый цет текста */
            a.menuItem:not(.newsLink) {color: orange;}  /* оранжевый цет текста */
            div ul li a {color: gray; } /* серый цет текста */
        </style>
    </head>
    <body>
        <div id="menu">
            <ul class="nav">
                <li><a class="menuItem">Главная</a></li>
                <li><a class="menuItem">Форум</a></li>
                <li><a class="menuItem">Блог</a></li>
                <li><a class="menuItem">О сайте</a></li>
            </ul>
        </div>
    </body>
</html>
```

В CSS определено аж пять различных селекторов, которые устанавливают цвет ссылок. В итоге браузер выберет селектор `#menu a` и окрасит ссылки в темно-синий цвет. Но почему, на каком основании браузер выберет этот селектор?

Рассмотрим, как у нас будут суммироваться баллы по каждому из пяти селекторов:

Селектор	Идентификаторы	Классы	Теги	Сумма
.nav li a	0	1	2	12
#menu a	1	0	1	101
.nav .menuItem	0	2	0	20
a.menuItem:not(.newsLink)	0	2	1	21
div ul li a	0	0	4	4

Итак, мы видим, что для селектора `#menu a` в колонке сумма оказалось больше всего баллов - 101. То есть в нем 1 идентификатор (100 баллов) и один тег(1 балл), которые в сумме дают 101 балл.

К примеру, в селекторе `.nav .menuItem` два селектора класса, каждый из которых дает 10 баллов, то есть в сумме 20 баллов.

При этом псевдокласс `:not` в отличие от других псевдокдассов не учитывается, однако учитывается тот селектор, который передается в псевдокласс not.

Правило !important

CSS предоставляет возможность полностью отменить значимость стилей. Для этого в конце стиля указывается значение **!important**:

```
a {font-size: 18px; color: red !important;}
#menu a {color: navy;}
```

В этом случае вне зависимости от наличия других селекторов с большим количеством баллов к ссылкам будет применяться красный цвет, определяемый первым стилем.

Псевдоклассы :is() и :where()

Псевдокласс :is()

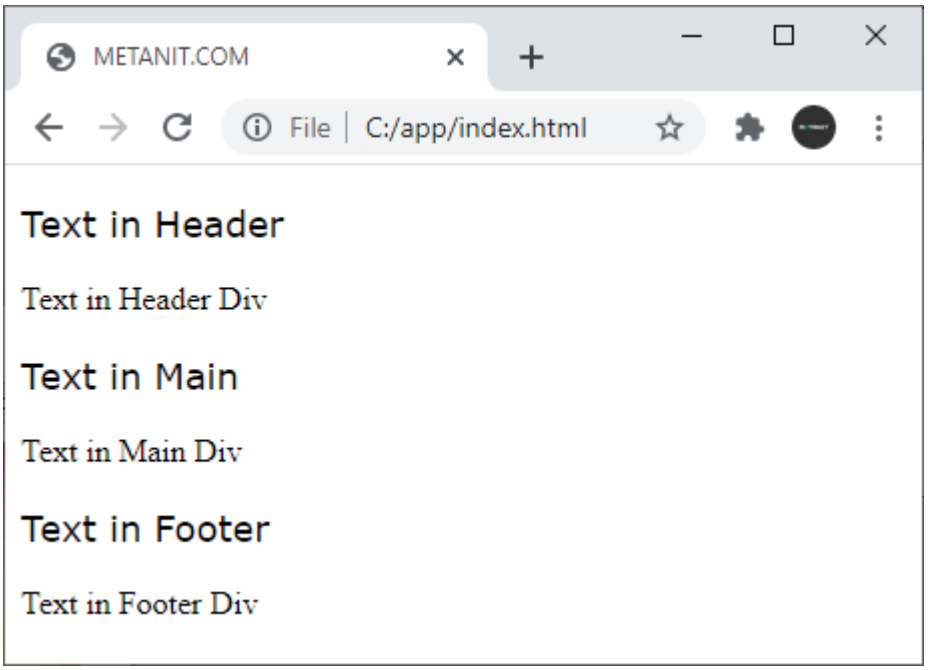
Псевдокласс **:is()** применяется для сокращения длинных селекторов. Функция псевдокласса **:is()** принимает список селекторов для выбора элементов html.

Например, мы хотим применить стиль к параграфам, которые находятся в первом уровне вложенности в элементах `header`, `main`, `footer`:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>METANIT.COM</title>
    <style>
        /* header > p, main > p, footer > p {font-size: 18px; font-family: Verdana;}
        альтернативный вариант без :is*/
        :is(header, main, footer) > p  {font-size: 18px; font-family: Verdana;}
    </style>
</head>
<body>
<header>
    <p>Text in Header</p>
    <div><p>Text in Header Div</p></div>
</header>
<main>
    <p>Text in Main</p>
    <div><p>Text in Main Div</p></div>
</main>
```

```
    <footer>
        <p>Text in Footer</p>
        <div><p>Text in Footer Div</p></div>
    </footer>
    </body>
    </html>
```

В данном случае для параграфов устанавливается шрифт Verdana высотой 18px:



Без использования псевдокласса **:is** нам пришлось бы написать:

```
header > p, main > p, footer > p {font-size: 18px; font-family: Verdana;}
```

Псевдокласс **:is** позволяет значительно сократить эту запись:

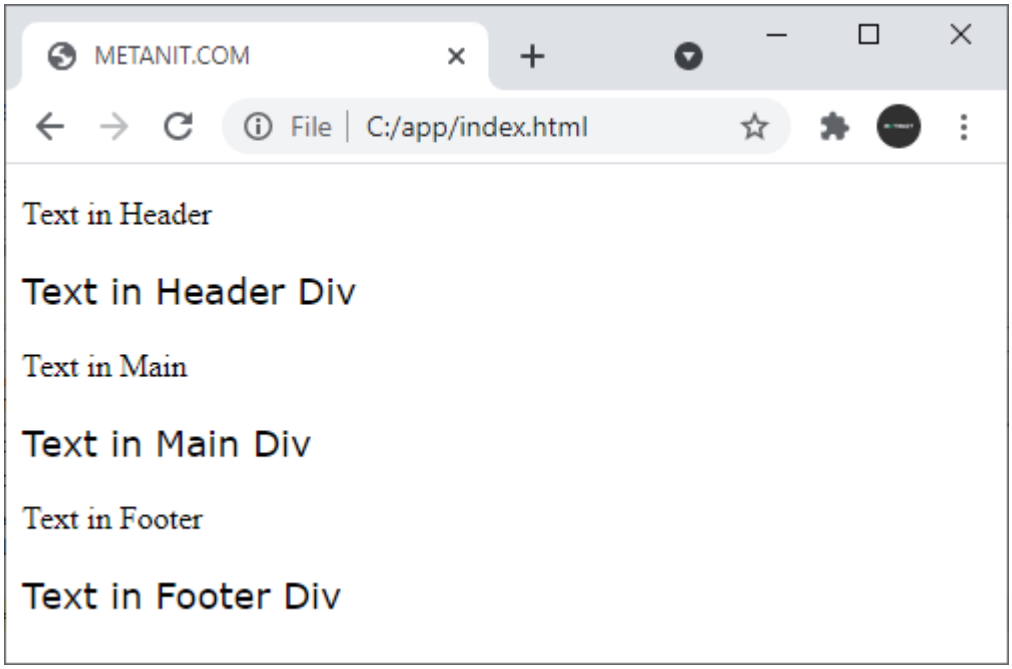
```
:is(header, main, footer) > p {font-size: 18px; font-family: Verdana;}
```

То есть в данном случае выражение `:is(header, main, footer)` выбирает все элементы header, main и footer и объединяет их с последуюзими селекторами.

Псевдокласс `:is` может быть вложенным, например, возьмем пример выше, только выберем теперь только те параграфы, которые находятся в элементах div:

```
    <!DOCTYPE html>
    <html>
    <head>
        <meta charset="utf-8" />
        <title>METANIT.COM</title>
        <style>
            :is(:is(header, main, footer) div) > p  {font-size: 18px; font-family:
            Verdana;}
        </style>
    </head>
    <body>
    <header>
        <p>Text in Header</p>
        <div><p>Text in Header Div</p></div>
```

```
</header>
<main>
    <p>Text in Main</p>
    <div><p>Text in Main Div</p></div>
</main>
<footer>
    <p>Text in Footer</p>
    <div><p>Text in Footer Div</p></div>
</footer>
</body>
</html>
```



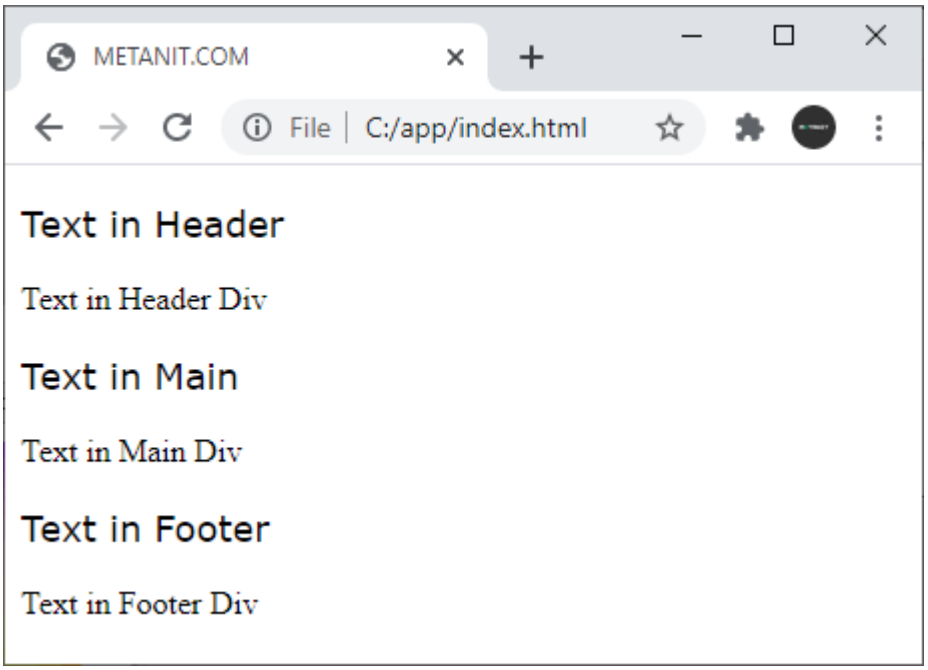
Псевдокласс :where

Псевдокласс **:where()** работает подобно `:is()`, он также принимает набор селекторов и выбирает все соответствующие селекторы. Например, первый пример, только вместо `:is()` теперь используем **:where**:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>METANIT.COM</title>
    <style>
        :where(header, main, footer) > p  {font-size: 18px; font-family: Verdana;}
    </style>
</head>
<body>
<header>
    <p>Text in Header</p>
    <div><p>Text in Header Div</p></div>
</header>
<main>
    <p>Text in Main</p>
    <div><p>Text in Main Div</p></div>
</main>
<footer>
```

```
    <p>Text in Footer</p>
    <div><p>Text in Footer Div</p></div>
</footer>
</body>
</html>
```

И мы получим тот же результат, что и для `:is()`:



Различие между :is() и :where()

В чем же разница между **:is()** и **:where()**? Псевдокласс **:is()** применяет каскадность стилей (selector specificity), которая определяется по селектору с самым большим рангом. А для стилей псевдокласса **:where()** ранг селекторов всегда равен 0.

Рассмотрим на схожем примере. Применение `:is()`:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>METANIT.COM</title>
    <style>
        div > :is(h2, #content) { background: red; }

        div > h2,
        div > #content {
            background: white;
        }
    </style>
</head>
<body>
<div>
    <h2>Заголовок</h2>
    <div id="content">Основное содержимое</div>
</div>
</body>
```

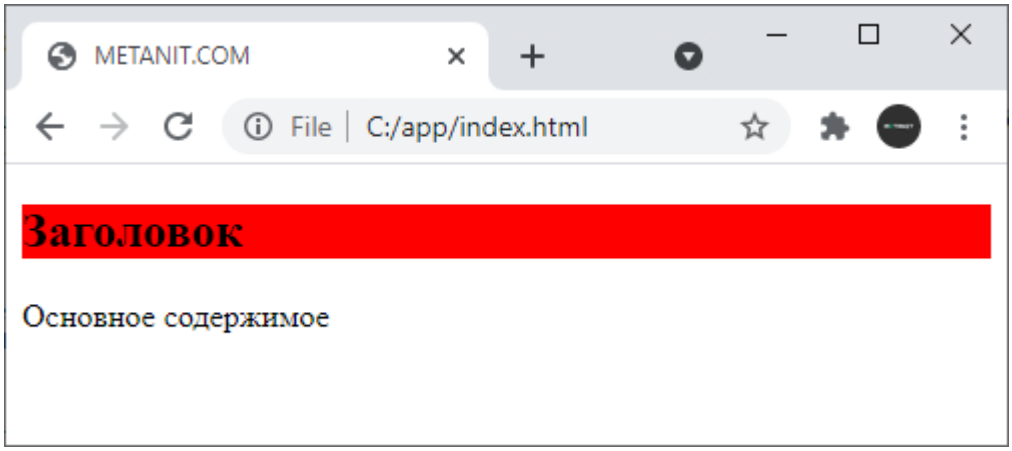
```
</html>
```

Здесь ранг селекторов в выражении `:is(h2, #content)` будет вычисляться по наибольшему селектору - в данном случае идентификатору `#content`. Таким образом, ранг селекторов в псевдоклассе `:is()` будет составлять 100 баллов (селекторы тегов оцениваются в 1 балл).

Дальше идет селектор `div > h2`, который переопределяет цвет фона. Но послкольку его ранг меньше, чем ранг `div > :is(h2, #content)`, то этот стиль применяться не будет.

Третий селектор `div > #content` имеет тот же ранг, что и `div > :is(h2, #content)`, поэтому его стиль будет работать.

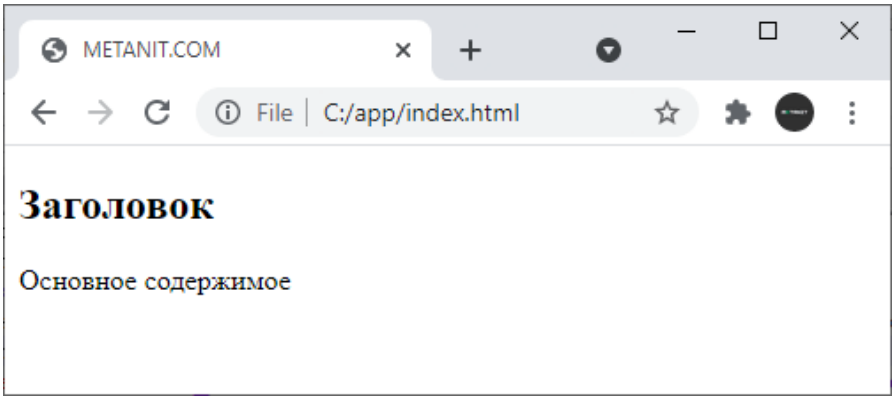
В итоге мы получим, что для элемента `<h2>` переопределение цвета фона не будет работать.



Теперь изменим на `:where()`:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>METANIT.COM</title>
    <style>
        div > :where(h2, #content) { background: red; }

        div > h2,
        div > #content {
            background: white;
        }
    </style>
</head>
<body>
<div>
    <h2>Заголовок</h2>
    <div id="content">Основное содержимое</div>
</div>
</body>
</html>
```

Ранг селектора `:where(h2, #content)` будет равен 0, поэтому стиль, задаваемый селектором `div > :where(h2, #content)` будет переопределен последующими стилями.


METANIT.COM
File | C:/app/index.html
Заголовок
Основное содержимое

Цвет в CSS

В CSS широкое распространение находит использование цветов. Чтобы установить цвет текста, фона или границы, нам надо указать цвет.

Например, определим красный цвет для фона элемента div:

```
div{
    background-color: red;
}
```

В CSS есть несколько различных свойств, которые в качестве значения требует определенный цвет. Например, за установку цвета текста отвечает свойство **color**, за установку фона элемента - свойство **background-color**, а за установку цвета границы - **border-color**.

Существует несколько различных способов определения цвета текста.

- **Шестнадцатеричного значение**. Оно состоит из отдельных частей, которые кодируют в шестнадцатеричной системе значения для красного, зеленого и синего цветов.

  Например, **#1C4463**. Здесь первые два символа `1C` представляю значение красной компоненты цвета, далее `44` - значение зеленой компоненты цвета и `63` - значение уровня синего цвета. Финальный цвет, который мы видим на веб-странице, образуется с помощью смешивания этих значений.

  Если каждое из трех двухзначных чисел содержит по два одинаковых символа, то их можно сократить до одного. Например, `#5522AA` можно сократить до `#52A`, или, к примеру, `#eeeeee` можно сократить до `#eee`. При этом не столь важно, в каком регистре будут символы.

- **Значение RGB**. Значение RGB также представляет последовательно набор значений для красного, зеленого и синего цветов (Red — красный, Green — зеленый, Blue — синий). Значение каждого цвета кодируется тремя числами, которые могут представлять либо процентные соотношения (0–100%), либо число от 0 до 255.

  Например:

```
background-color: rgb(100%,100%,100%);
```

  Здесь каждый цвет имеет значение `100%`. И в итоге при смешивании этих значений будет создаваться белый цвет. А при значениях в 0% будет генерироваться черный цвет:

```
background-color: rgb(0%, 0%, 0%);
```

  Между 0 и 100% будут находиться все остальные оттенки.

  Но, как правило, чаще применяются значения из диапазона от 0 до 255. Например,

```
background-color: rgb(28, 68, 99);
```

- **Значение RGBA**. Это тоже самое значение RGB плюс компонент прозрачности (Alpha). Компонент прозрачности имеет значение от 0 (полностью прозрачный) до 1 (не прозрачный). Например:

```
background-color: rgba(28, 68, 99, .6);
```

- **Значение HSL**. HSL представляет аббривиатуру: Hue — тон, Saturation — насыщенность и Lightness — осветленность. HSL задает три значения. Первое значение Hue угол в круге оттенков - значение в градусах от 0 до 360. Например, красный — 0 (или 360 при полном обороте круга). Каждый цвет занимает примерно 51°.

  Второе значение - Saturation - представляет насыщенность, то указывает, насколько чистым является цвет. Насыщенность определяется в процентах от 0 (полное отсутствие насыщенности) до 100% (яркий, насыщенный цвет).

  Третье значение - Lightness - определяет осветленность и указывается в процентах от 0 (полностью черный) до 100 (полностью белый). Для получения чистого цвет применяется значение 50 %.

  Например:

```
background-color: hsl(206, 56%, 25%);
```

  Данный цвет является эквивалентом значений `#1C4463` и `rgb(28, 68, 99)`

- **Значение HSLA**. Аналогично RGBA здесь к HSL добавляется компонента прозрачности в виде значения от 0 (полностью прозрачный) до 1 (не прозрачный). Например:

```
background-color: hsl(206, 56%, 25%, .6);
```

- **Строковые значения**. Существует ряд константных строковых значений, например, red (для красного цвета) или green (для зеленого цвета). К примеру,

```
color: red;
```

- является эквивалентом

```
color: #ff0000;
```

- Полный перечень цветов можно найти на странице https://developer.mozilla.org/en-US/docs/Web/CSS/color_value

В заключение следует отметить, что существует множество бесплатных онлайн-генераторов цвета, где можно настроить и посмотреть цвет в нужном формате. Например, генератор цвета на mdn.

Прозрачность

Ряд настроек цвета позволяют установить значение для альфа-компоненты, которая отвечает за прозрачность. Но также в CSS есть специальное свойство, которое позволяет установить прозрачность элементов - свойство **opacity**. В качестве значения оно принимает число от 0 (полностью прозрачный) до 1 (не прозрачный):

```
div{
    width: 100px;
    height: 100px;

```

```
5       background-color: red;
6       opacity: 0.4;
7   }
```

Стилизация шрифтов

Семейство шрифтов

Свойство **font-family** устанавливает семейство шрифтов, которое будет использоваться. Например:

```
1   body{
2       font-family: Arial;
3   }
```

В данном случае устанавливается шрифт Arial.

Шрифт свойства font-family будет работать, только если у пользователя на локальном компьютере имеется такой же шрифт. По этой причине нередко выбираются стандартные шрифты, которые широко распространены, как Arial, Verdana и т.д.

Также нередко применяется практика нескольких шрифтов:

```
1   body{
2       font-family: Arial, Verdana, Helvetica;
3   }
```

В данном случае основным шрифтом является первый - Arial. Если он на компьютере пользователя не поддерживается, то выбирается второй и т.д.

Если название шрифта состоит из нескольких слов, например, Times New Roman, то все название заключается в кавычки:

```
1   body{
2       font-family: "Times New Roman";
3   }
```

Кроме конкретных стилей также могут использоваться общие универсальные шрифты, задаваемые с помощью значений **sans-serif** и **serif**:

```
1   body{
2       font-family: Arial, Verdana, sans-serif;
3   }
```

Так, если ни Arial, ни Verdana не поддерживаются на компьютере пользователя, то используется sans-serif - универсальный шрифт без засечек.

Типы шрифтов

Шрифты с засечками

Шрифты с засечками названы так, потому что на на концах основных штрихов имеют небольшие засечки. Считается, что они подходят для больших кусков текста, так как визуально связывают одну букву с другой, делая текст более читабельным.

Распространенные шрифты с засечками: Times, Times New Roman, Georgia, Garamond. Универсальный обобщенный шрифт с засечками представляет значение **serif**.

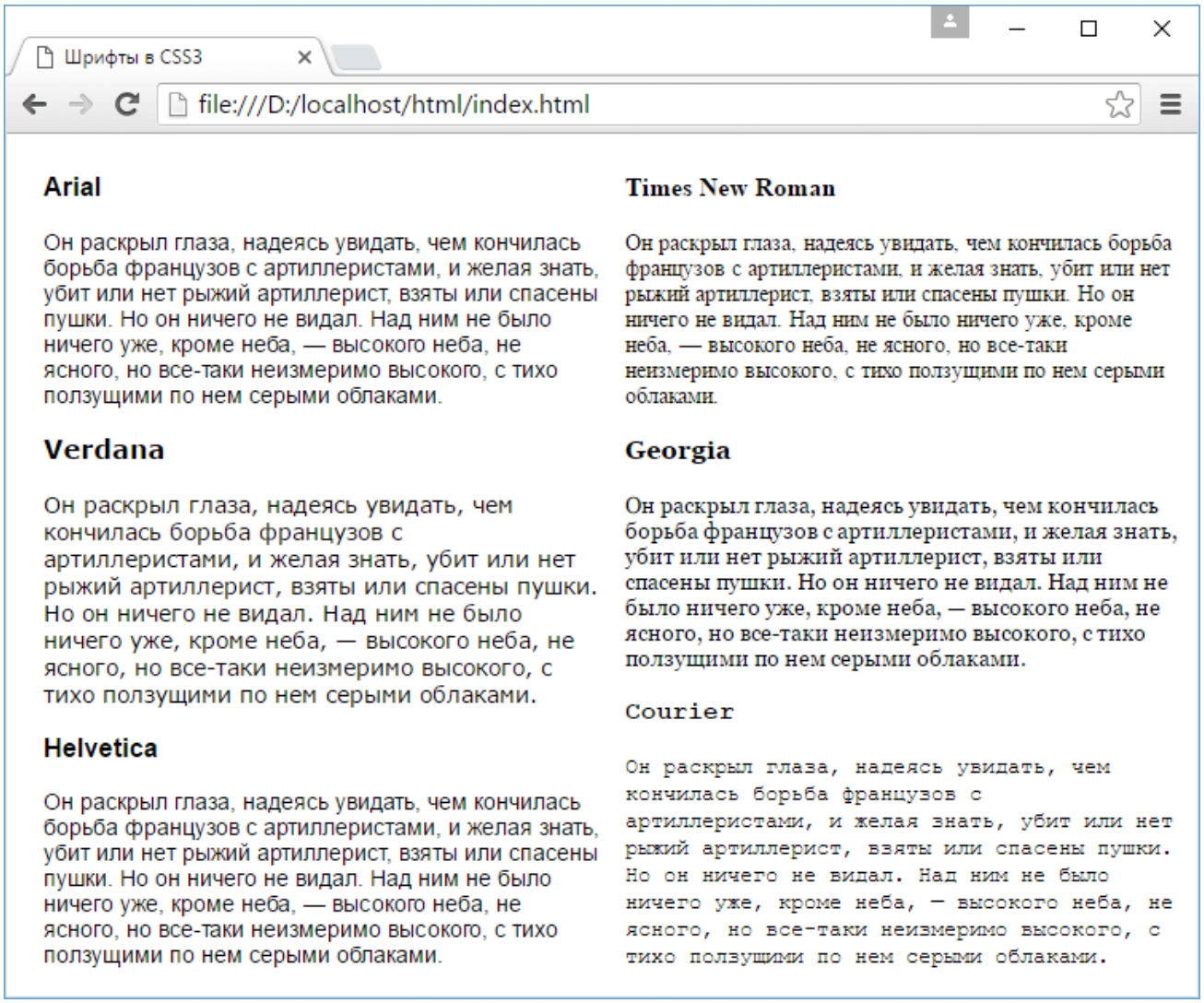
Шрифты без засечек

В отличие от шрифтов с засечками шрифты из этой группы не имеют засечек. Наиболее распространенные шрифты этой группы: Arial, Helvetica, Verdana.

Моноширинные шрифты

Моноширинный шрифт преимущественно применяется для отображения программного кода и не предназначен для вывода стандартного текста статей. Свое название эти шрифты получили от того, что каждая буква в таком шрифте имеет одинаковую ширину. Примеры подобных шрифтов: Courier, Courier New, Consolas, Lucida Console.

Примеры шрифтов:



Толщина шрифта

Свойство **font-weight** задает толщину шрифта. Оно может принимать 9 числовых значений: 100, 200, 300, 400,...900. 100 - очень тонкий шрифт, 900 - очень плотный шрифт.

В реальности чаще для этого свойства используют два значения: `normal` (нежирный обычный текст) и `bold` (полужирный шрифт):

```
font-weight: normal;
font-weight: bold;
```

Курсив

Свойство **font-style** позволяет выделить текст курсивом. Для этого используется значение `italic`:

```
p {font-style: italic;}
```

Если надо отменить курсив, то применяется значение `normal`:

```
p {font-style: normal;}
```

Цвет шрифта

Свойство **color** устанавливает цвет шрифта:

```
p {
    color: red;
}
```

Внешние шрифты

Не всегда стандартные встроенные шрифты, как Arial или Verdana, могут быть удобны. Нередко встречается ситуация, когда веб-дизайнер хочет воспользоваться возможностями какого-то другого шрифта, которого нет среди встроенных, но который доступен из внешнего файла. Такой шрифт можно подключить с помощью директивы **font-face**:

```
@font-face {

    font-family: 'Roboto';
    src: url(http://fonts.gstatic.com/s/roboto/v15/mErvLBYg_cXG3rLvUsKT_
    fesZW2xOQ-xsNqO47m55DA.woff2);
}
```

Свойство `font-family` задает название шрифта, а свойство `src` - путь к шрифту.

В данном случае веб-страница будет подгружать шрифт, который расположен в интернете по адресу *http://fonts.gstatic.com/s/roboto/v15/mErvLBYg_cXG3rLvUsKT_fesZW2xOQ-xsNqO47m55DA.woff2*

В качестве альтернативы мы можем загрузить файл шрифта на локальный компьютер и уже оттуда подгружать его на веб-страницу. Как правило, для хранения своих шрифтов рядом с веб-страницей создается папка *fonts*:

```
@font-face{

    font-family: 'Roboto';
    src:url('fonts/roboto.ttf');
}
```

После подключения шрифта, его можно использовать в стилях:

```
p{

    font-family: Roboto;
}
```

В данном случае используется шрифт Roboto, созданный компанией Google и определенный в файле в формате woff2. Однако не все браузеры поддерживают данный формат шрифтов.

Грубо говоря, существует несколько различных форматов шрифтов: TrueType(расширение ttf), Open Type (расширение otf), Embedded Open Type (расширение eot), Web Open Font Format (woff/woff2), Scalable Vector Graphic (svg). Разные браузеры могут поддерживать разные шрифты. И чтобы решить проблему поддержки шрифтов создатели шрифта часто создают сразу несколько форматов. И мы можем сразу эти форматы определить. Например:

```
 @font-face {

     font-family:'FontAwesome';
     src: url('https://maxcdn.bootstrapcdn.com/font-awesome/4.6.1/fonts/fontawesome-webfont.eot');
     src: url('https://maxcdn.bootstrapcdn.com/font-awesome/4.6.1/fonts/fontawesome-webfont.eot?
     #iefix') format('embedded-opentype'),
          url('https://maxcdn.bootstrapcdn.com/font-awesome/4.6.1/fonts/fontawesome-webfont.woff2')
```

```
            format('woff2'),
7           url('https://maxcdn.bootstrapcdn.com/font-awesome/4.6.1/fonts/fontawesome-webfont.woff')
            format('woff'),
8           url('https://maxcdn.bootstrapcdn.com/font-awesome/4.6.1/fonts/fontawesome-webfont.ttf')
            format('truetype'),
9           url('https://maxcdn.bootstrapcdn.com/font-awesome/4.6.1/fonts/fontawesome-webfont.svg')
            format('svg');
10  }
```

Как и в предыдущем случае свойства `font-family` и `src` задают название и путь к шрифту. Но теперь также для совместимости добавляется еще одно свойство `src`. Второе свойство src устанавливает сразу несколько шрифтов. Первым шрифтом также идет шрифт в формате EOT, но теперь к расширению файла .eot добавляется значение `?#iefix`. Это делается для совместимости с версиями Internet Explorer 6–8. Если после .eot не будет добавляться это значение, то шрифт может неправильно отображаться в Internet Explorer 6-8.

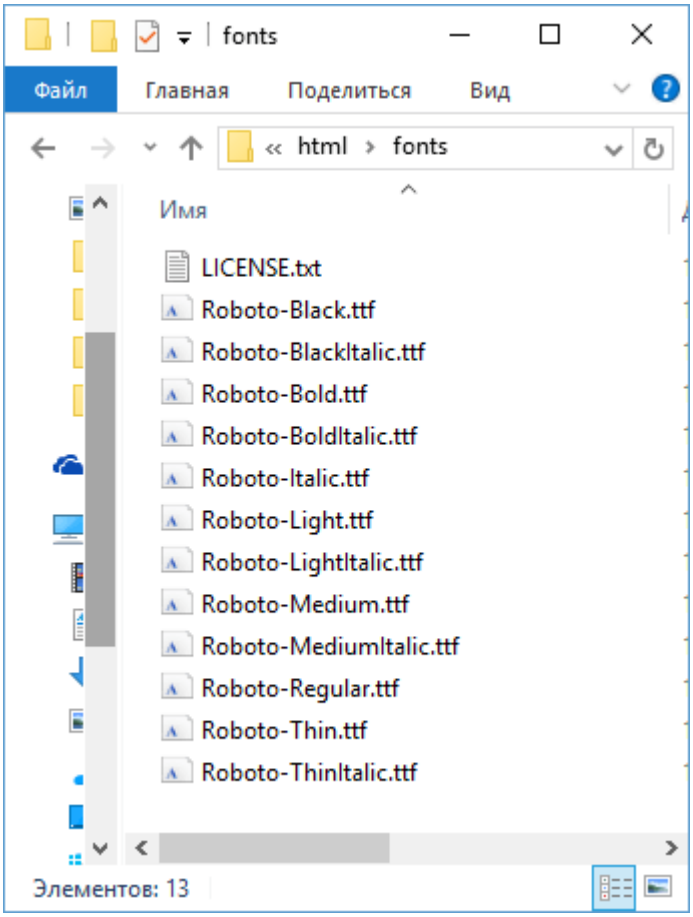
После параметра url идет определение формата шрифта:

```
1   format('embedded-opentype')
```

Когда браузер начнет загружать веб-страницу, на которой таким образом определен шрифт, то браузер не будет подгружать все шрифты во всех форматах, а загрузит только первый шрифт, который для него окажется понятным.

Разные версии шрифта

Загрузим шрифт Roboto по ссылке Roboto на локальный компьютер. Распакуем загруженный архив в папку, которую назовем fonts, и положим эту папку в один каталог рядом с веб-страницей. В загруженной папке шрифта Roboto мы сможем увидеть, что она содержит не один файл, а сразу несколько:



Зачем нам столько файлов? Дело в том, что каждый шрифт должен определять отдельный стиль для обычного текста, для текста, выделенного курсивом, для текста, выделенного жирным, для текста, сочетающего выделение жирным и курсивом и т.д.

Чтобы браузер мог автоматически распознавать разные варианты шрифта, к директиве @font-face добавляются свойства **font-weight** и **font-style**, которые соответственно устанавливают выделение жирным и выделение курсивом:

```
@font-face {
    font-family: 'Roboto';
    src: url(fonts/Roboto-Regular.ttf);
    font-weight: normal;
    font-style: normal;
}
p{
    font-family: Roboto;
}
```

Поскольку версия шрифта `Roboto-Regular.ttf` применяется для текста, не выделенного курсивом и жирным, то вместе с ним устанавливаются значения:

```
font-weight: normal;
font-style: normal;
```

То есть тем самым мы устанавливаем, что выделения курсивом не будет (`font-style: normal;`) и выделения жирным не будет (`font-weight: normal;`)

Кроме версии `Roboto-Regular.ttf`, как видно выше на картинке, в папке есть еще другие версии шрифта Roboto. Например, курсивная версия шрифта *Roboto-Italic.ttf* и ряд других.

Если мы хотим, чтобы при выделении курсивом браузер использовал именно курсивную версию, то нам надо добавить еще одну директиву font-face:

```
@font-face {
    font-family: 'Roboto';
    src: url(fonts/Roboto-Italic.ttf);
    font-weight: normal;
    font-style: italic;
}
```

Значение `font-style: italic` указывает, что данную версию шрифта следует применять при выделении курсивом.

Аналогично мы можем задать те версии шрифта, которые должны использоваться при выделении сразу и жирным, и курсивом, либо только при выделении жирным:

```
 @font-face {
     font-family: 'Roboto';
     src: url(fonts/Roboto-Bold.ttf);
     font-weight: bold;
     font-style: normal;
 }
 @font-face {
     font-family: 'Roboto';
```

```
        src: url(fonts/Roboto-BoldItalic.ttf);
        font-weight: bold;
        font-style: italic;
    }
```

Значение `font-weight: bold` указывает, что данная версия шрифта применяется при выделении жирным.

Теперь используем все эти шрифты:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Шрифты в CSS3</title>

        <style>
            @font-face {
                font-family: 'Roboto';
                src: url(fonts/Roboto-Regular.ttf);
                font-weight: normal;
                font-style: normal;
            }
            @font-face {
                font-family: 'Roboto';
                src: url(fonts/Roboto-Italic.ttf);
                font-weight: normal;
                font-style: italic;
            }
            @font-face {
                font-family: 'Roboto';
                src: url(fonts/Roboto-Bold.ttf);
                font-weight: bold;
                font-style: normal;
            }

            @font-face {
                font-family: 'Roboto';
                src: url(fonts/Roboto-BoldItalic.ttf);
                font-weight: bold;
                font-style: italic;
            }
            p{
                font-family: Roboto;
            }
        </style>
    </head>
    <body>
        <p>Стиль Roboto может выделять <i>курсивом</i> и <b>жирным</b>,
        либо <b><i>и тем, и другим</i></b></p>
    </body>
</html>
```

Теперь к тексту в тегах `<i></i>`, который использует курсив, будет применяться версия "Roboto-Italic.ttf", а к тексту в тегах `<b></b>` - шрифт "Roboto-Bold.ttf".

Источники шрифтов

В интернете можно найти множество нестандартных шрифтов. Наиболее популярным репозиторием шрифтов является https://www.google.com/fonts/ - набор шрифтов от компании Google.

Также другим известным репозиторием шрифтов является Font Squirrel.

Следует также упомянуть такой популярный шрифт как FontAwesome. Он предоставляет множество различных интересных иконок, которые можно использовать на веб-странице.

Высота шрифта

Для установки размера шрифта используется свойство **font-size**:

```
div{
    font-size: 18px;
}
```

В данном случае высота шрифта составит 18 пикселей. Пиксели представляют наиболее часто используемые единицы измерения. Чтобы задать значение в пикселях, после самого значения идет сокращение "px".

Если к тексту явным образом не применяется высота шрифта, то используются значения браузера по умолчанию. Например, для простого текста в параграфах это 16 пикселей. Это базовый стиль текста.

Базовый стиль для разных элементов текста отличается: если для параграфов это 16 пикселей, то для заголовков h1 это 32 пикселя, для заголовков h2 - 24 пикселя и т..д.

Для измерения шрифта также можно использовать самые разные единицы измерения.

Ключевые слова

В CSS имеется семь ключевых слов, которые позволяют назначить размер шрифта относительно базового:

- **medium**: базовый размер шрифта браузера (16 пикселей)
- **small**: 13 пикселей
- **x-small**: 10 пикселей
- **xx-small**: 9 пикселей
- **large**: 18 пикселей
- **x-large**: 24 пикселя
- **xx-large**: 32 пикселя

Например:

```
font-size: x-large;
```

Проценты

Проценты позволяют задать значение относительно базового или унаследованного шрифта. Например:

```
font-size: 150%;
```

В данном случае высота шрифта будет составлять 150% от базового, то есть 16px * 1,5 = 24px

Наследование шрифта может изменить финальное значение. Например, следующую ситуацию:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
```

```
5           <title>Шрифты в CSS3</title>
6
7           <style>
8               div {font-size: 10px;}
9               p {font-size: 150%;}
10          </style>
11      </head>
12      <body>
13      <div>
14          <p>Однажды в студеную зимнюю пору</p>
15      </div>
16      </body>
17  </html>
```

Здесь элемент p наследует от контейнера - блока div шрифт высотой в 10 пикселей. То есть 10 пикселей теперь будет базовым для параграфа.

Далее для элемента p определяется новая высота шрифта в 150%. Это значит, что финальная высота будет равна 10px * 1,5 = 15px.

Единица em

Единица измерения em во многом эквивалентна процентам. Так, `1em` равен `100%`, `.5em` равно `50%` и т.д.

Форматирование текста

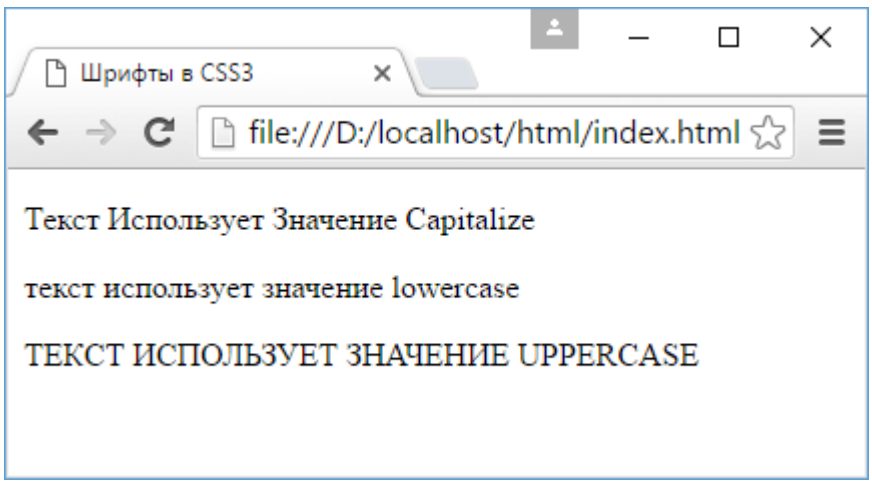
text-transform

Свойство **text-transform** изменяет регистр текста. Оно может принимать следующие значения:

- `capitalize`: делает первую букву слова заглавной
- `uppercase`: все слово переводится в верхний регистр
- `lowercase`: все слово переводится в нижний регистр
- `none`: регистр символов слова никак не изменяется

Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Шрифты в CSS3</title>

        <style>
            p.lowercase {text-transform: lowercase;}
            p.uppercase {text-transform: uppercase;}
            p.capitalize { text-transform: capitalize;}
        </style>
    </head>
    <body>
    <div>
        <p class="capitalize">Текст использует значение capitalize</p>
        <p class="lowercase">Текст использует значение lowercase</p>
        <p class="uppercase">Текст использует значение uppercase</p>
    </div>
    </body>
</html>
```



Свойство text-decoration

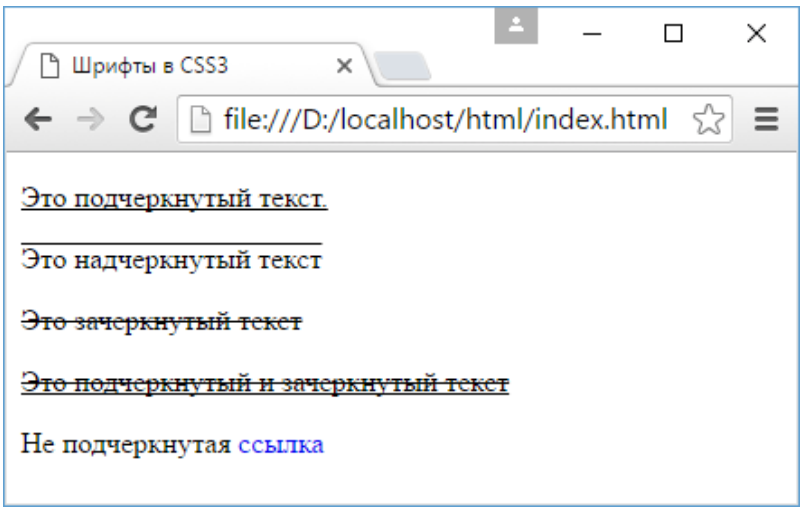
Свойство **text-decoration** позволяет добавить к тексту некоторые дополнительные эффекты. Это свойство может принимать следующие значения:

- `underline`: подчеркивает текст

- `overline`: надчеркивает текст, проводит верхнюю линию
- `line-through`: зачеркивает текст
- `none`: к тексту не применяется декорирование

Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Шрифты в CSS3</title>

        <style>
            p.under {
                text-decoration: underline;
            }
            p.over {
                text-decoration: overline;
            }
            p.line {
                text-decoration: line-through;
            }
            p.mixed {
                text-decoration: underline line-through;
            }
            a.none {
                text-decoration: none;
            }
        </style>
    </head>
    <body>
    <div>
        <p class="under">Это подчеркнутый текст.</p>
        <p class="over">Это надчеркнутый текст</p>
        <p class="line">Это зачеркнутый текст</p>
        <p class="mixed">Это подчеркнутый и зачеркнутый текст</p>
        <p>Не подчеркнутая <a href="index.php" class="none">ссылка<a></p>
    </div>
    </body>
</html>
```



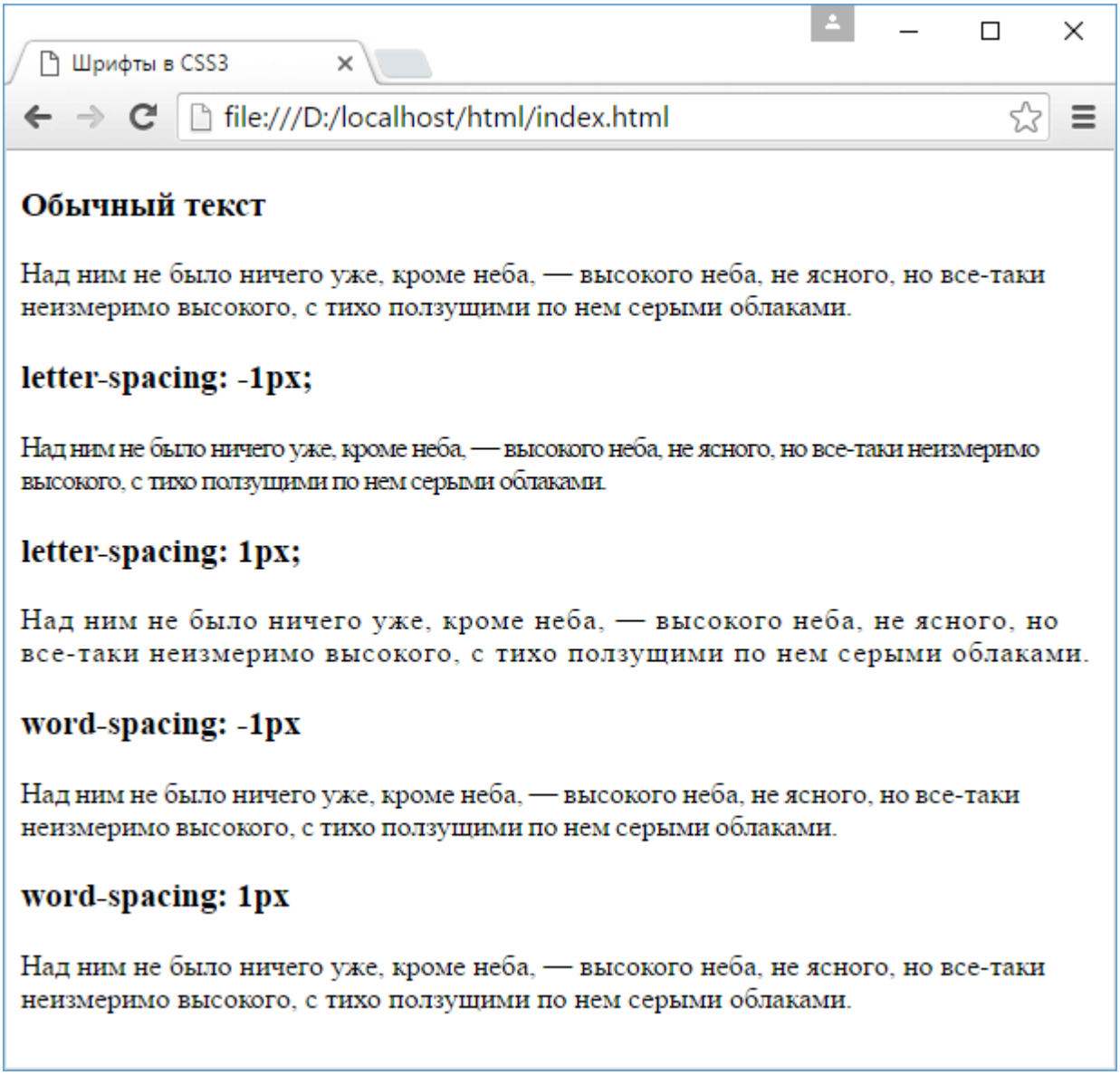
При необходимости мы можем комбинировать значения. Так, в предпоследнем случае применялся стиль:

```
p.mixed { text-decoration: underline line-through; }
```

Межсимвольный интервал

Два свойства CSS позволяют управлять интервалом между символами и словами текста. Для межсимвольного интервала применяется атрибут **letter-spacing**, а для интервала между словами - **word-spacing**:

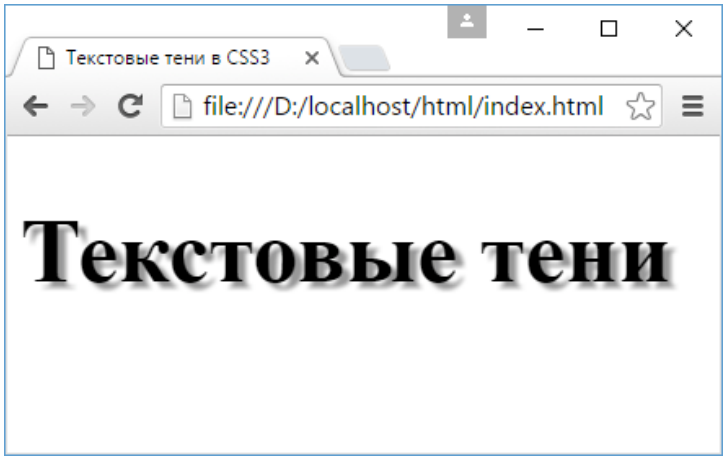
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Шрифты в CSS3</title>

        <style>
            p.smallLetterSpace {
                letter-spacing: -1px;
            }
            p.bigLetterSpace {
                letter-spacing: 1px;
            }
            p.smallWordSpace{
                word-spacing: -1px;
            }
            p.bigWordSpace{
                word-spacing: 1px;
            }
        </style>
    </head>
    <body>
    <div>
        <h3>Обычный текст</h3>
        <p>Над ним не было ничего уже, кроме неба...</p>
        <h3>letter-spacing: -1px;</h3>
        <p class="smallLetterSpace">Над ним не было ничего уже, кроме неба...</p>
        <h3>letter-spacing: 1px;</h3>
        <p class="bigLetterSpace">Над ним не было ничего уже, кроме неба...</p>
        <h3>word-spacing: -1px</h3>
        <p class="smallWordSpace">Над ним не было ничего уже, кроме неба...</p>
        <h3>word-spacing: 1px</h3>
        <p class="bigWordSpace">Над ним не было ничего уже, кроме неба...</p>
    </div>
    </body>
</html>
```



text-shadow

С помощью свойства **text-shadow** можно создать тени для текста. Для этого свойства необходимо задать четыре значения: горизонтальное смещение тени относительно текста, вертикальное смещение тени относительно текста, степень размытости тени и цвет отбрасываемой тени. Например:

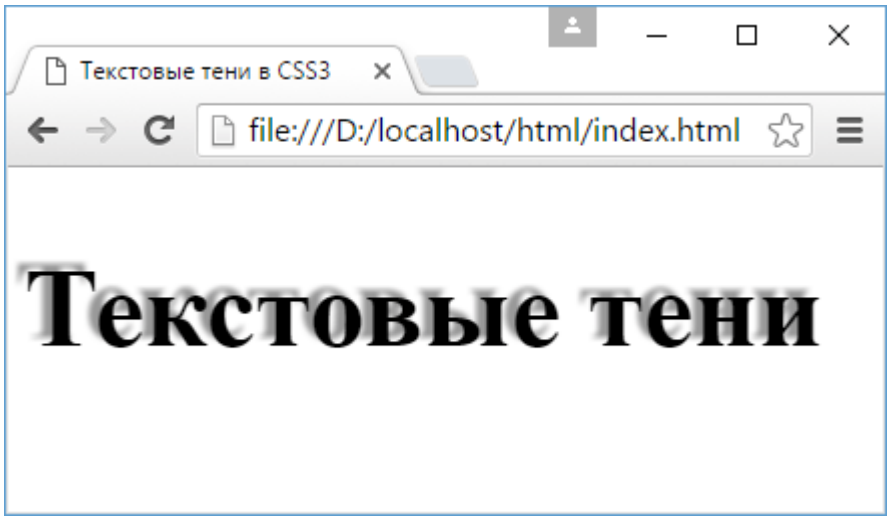
```
1   h1{
2       text-shadow: 5px 4px 3px #999;
3   }
```



В данном случае горизонтальное смещение тени относительно букв составляет 5 пикселей, а вертикальное смещение вниз - 4 пикселя. Степень размытости - 3 пикселя, и для тени используется цвет #999.

Если нам надо было бы создать горизонтальное смещение влево, а не вправо, как по умолчанию, то в этом случае надо было бы использовать отрицательное значение. Аналогично для создания вертикального смещения вверх также надо использовать отрицательное значение. Например:

```
1  h1{
2      text-shadow: -5px -4px 3px #999;
3  }
```



Стилизация абзацев

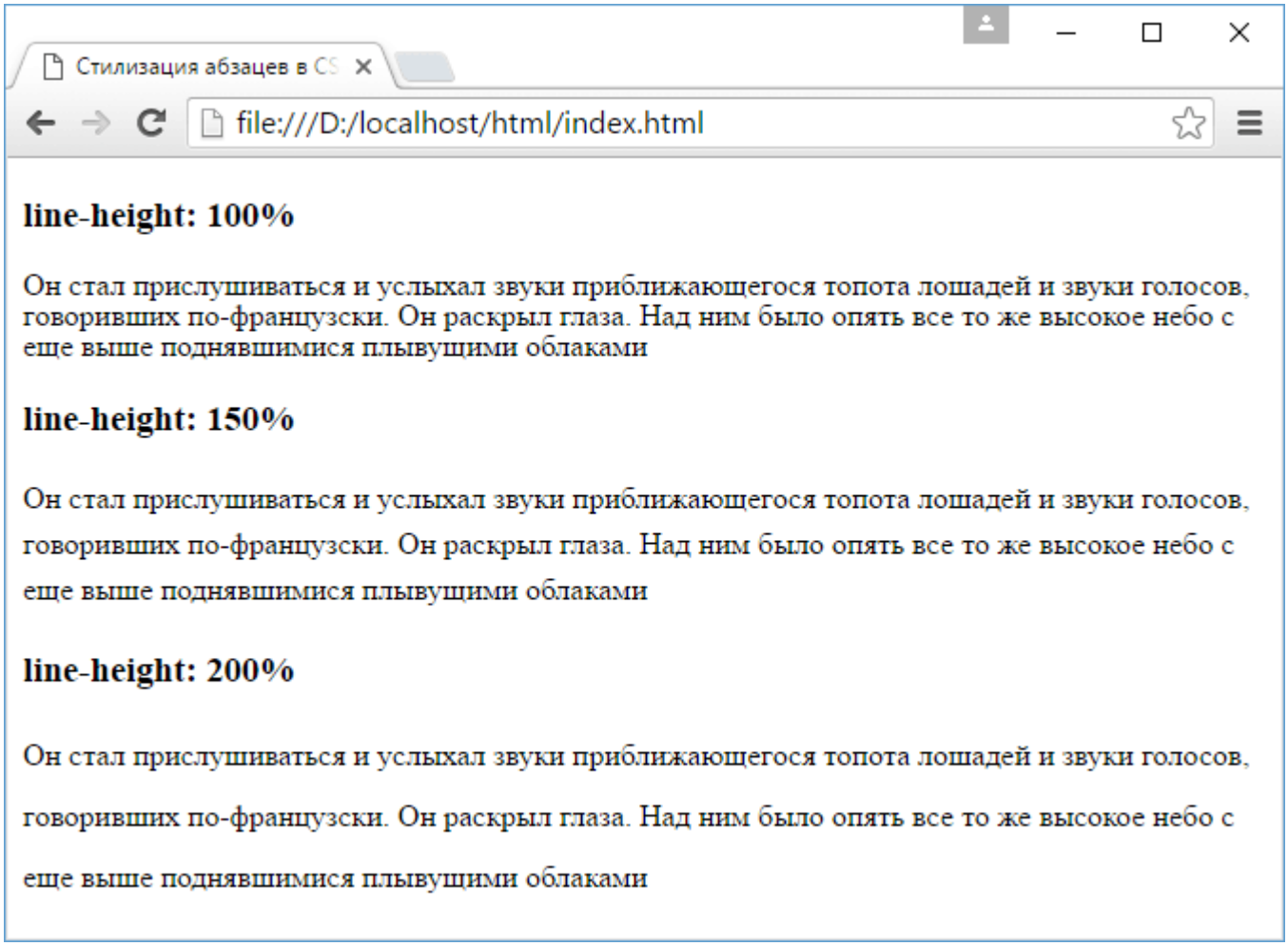
Отдельная группа свойств CSS позволяет стилизовать большие группы текста, например, установить высоту строки или выравнивание текста.

line-height

Свойство **line-height** определяет межстрочный интервал. Для его установки можно использовать пиксели, проценты или единицы em. Как правило, применяются либо проценты, либо em. Например:

```
1   p{
2       line-height: 150%;
3   }
```

Если это свойство не установлено, то по умолчанию используется значение `line-height: 120%;`.



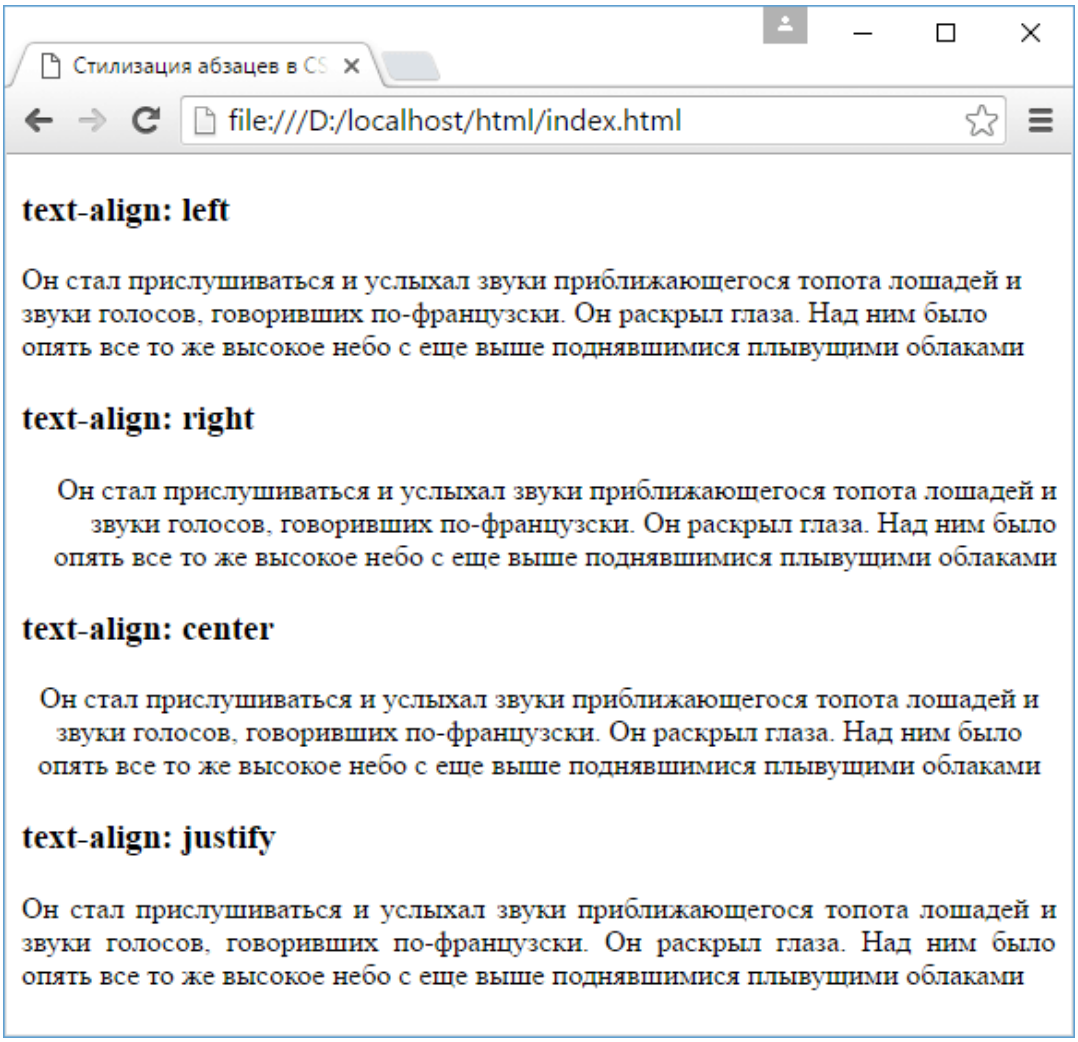
text-align

Свойство **text-align** выравнивает текст относительно одной из сторон веб-страницы. Оно принимает следующие значения:

- `left`: текст выравнивается по левой стороне
- `right`: текст выравнивается по правой стороне
- `justify`: выравнивание по ширине, слова равномерно распределяются по строке
- `center`: выравнивание по центру

Например:

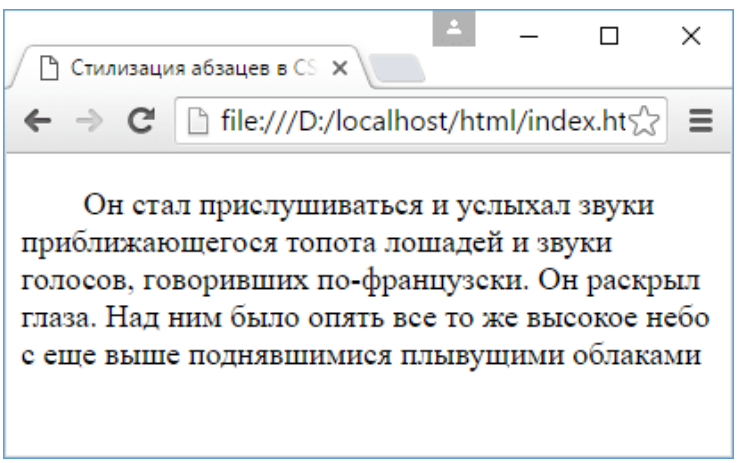
```
p{
    text-align: left;
}
```



text-indent

Свойство **text-indent** задает отступ первой строки абзаца. Для установки отступа могут применяться стандартные единицы измерения, например, em или пиксели:

```
p{
    text-indent: 35px;
}
```



Стилизация списков

CSS предоставляет специальные свойства по стилизации списков. Одним из таких свойств является **list-style-type**. Оно может принимать следующие значения для нумерованных списков:

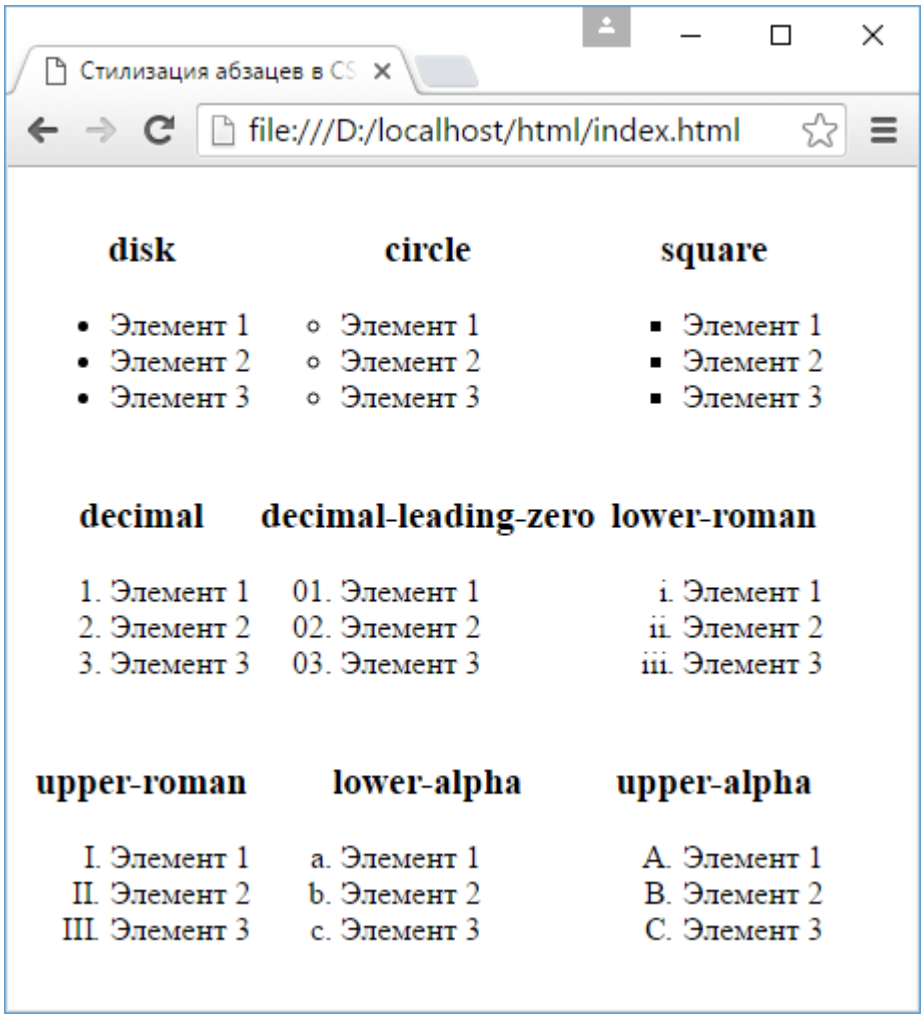
- `decimal`: десятичные числа, отсчет идет от 1
- `decimal-leading-zero`: десятичные числа, которые предваряются нулем, например, 01, 02, 03, … 98, 99
- `lower-roman`: строчные латинские цифры, например, i, ii, iii, iv, v
- `upper-roman`: заглавные латинские цифры, например, I, II, III, IV, V…
- `lower-alpha`: строчные латинские буквы, например, a, b, c..., z
- `upper-alpha`: заглавные латинские буквы, например, A, B, C, … Z

Для ненумерованных списков:

- `disc`: черный диск
- `circle`: пустой кружочек
- `square`: черный квадратик

Например:

```
1   ul{
2       list-style-type: square;
3   }
```



Чтобы вообще отключить маркеры у элементов списка, используется значение `none`:

```
ul{
    list-style-type: none;
}
```

Данное свойство может применяться как ко всему списку, так и к отдельным элементам. Например:

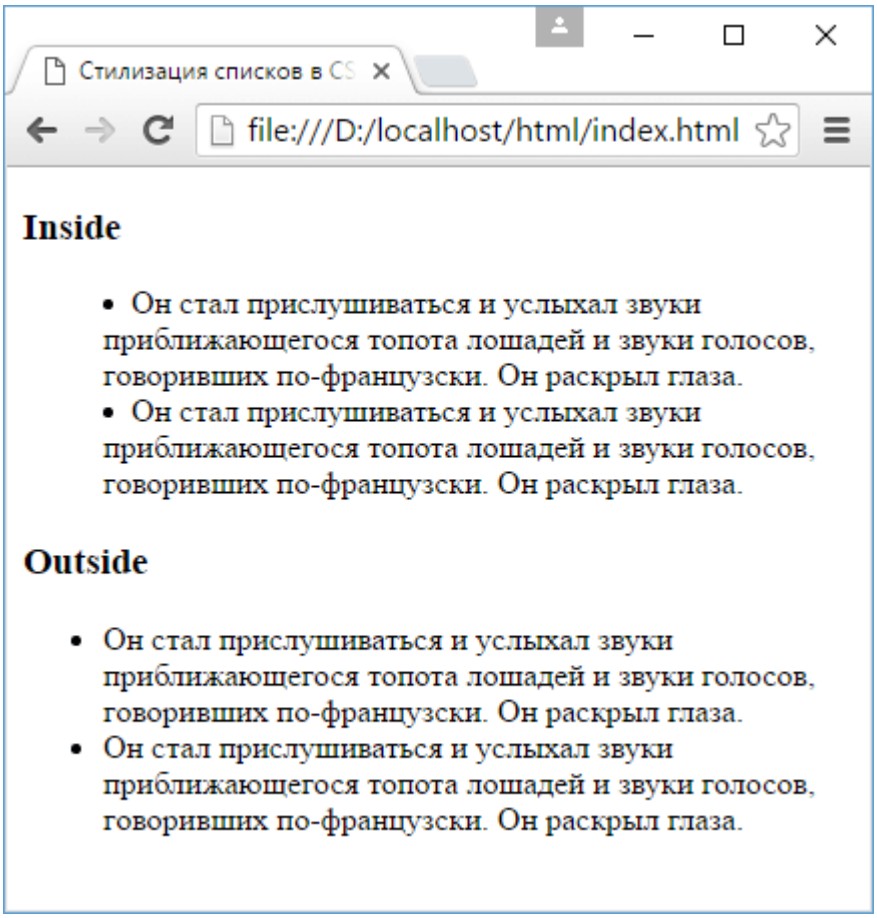
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Стилизация списков в CSS3</title>
        <style>

            .decimal{
                list-style-type: decimal;
            }
            ol{
                list-style-type: lower-roman;
            }
        </style>
    </head>
    <body>
        <ol>
            <li>Элемент 1</li>
            <li class="decimal">Элемент 2</li>
            <li>Элемент 3</li>
            <li>Элемент 4</li>
        </ol>
    </body>
</html>
```

list-style-position

Веб-браузеры обычно отображают маркеры списка слева от элементов списка. С помощью свойства **list-style-position** мы можем настроить их позиционирование. Это свойство принимает два значения: `outside` (по умолчанию) и `inside` (обеспечивает равномерное распределение по ширине).

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Стилизация списков в CSS3</title>
        <style>

            ul.outside{
                list-style-position: outside;
            }
            ul.inside{
                list-style-position: inside;
            }
```

```
        </style>
    </head>
    <body>
        <h3>Inside</h3>
        <ul class="inside">
            <li>Он стал прислушиваться и услыхал звуки приближающегося топота лошадей и звуки голосов...</li>
            <li>Он стал прислушиваться и услыхал звуки приближающегося топота лошадей и звуки голосов...</li>
        </ul>
        <h3>Outside</h3>
        <ul class="outside">
            <li>Он стал прислушиваться и услыхал звуки приближающегося топота лошадей и звуки голосов...</li>
            <li>Он стал прислушиваться и услыхал звуки приближающегося топота лошадей и звуки голосов...</li>
        </ul>
    </body>
</html>
```

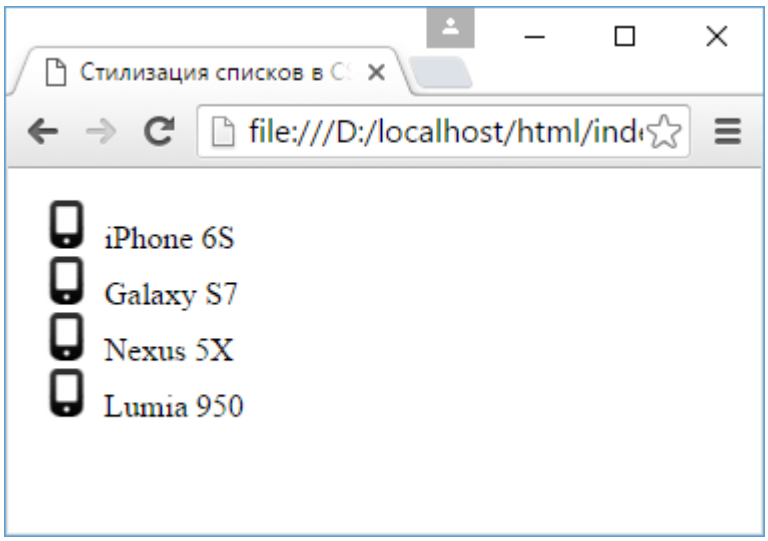


list-style-image

Свойство **list-style-image** позволяет задать в качестве маркера изображение:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Стилизация списков в CSS3</title>
```

```
6           <style>
7               ul{
8                   list-style-image:url(phone_touch.png);
9               }
10          </style>
11      </head>
12      <body>
13          <ul>
14              <li>iPhone 6S</li>
15              <li>Galaxy S7</li>
16              <li>Nexus 5X</li>
17              <li>Lumia 950</li>
18          </ul>
19      </body>
20  </html>
```



Свойство `list-style-image` в качестве значения принимает путь к изображению `url(phone_touch.png)`, где "phone_touch.png" - это название файла изображения. То есть в данном случае предполагается, что в одной папке с веб-страницей находится файл изображения phone_touch.png.

Стилизация таблиц

CSS предоставляет ряд свойств, которые помогают стилизовать таблицу:

- **border-collapse**: устанавливает, как будет стилизоваться граница смежных ячеек
- **border-spacing**: устанавливает промежутки между границами смежных ячеек
- **caption-side**: устанавливает положение элемента caption
- **empty-cells**: задает режим отрисовки для пустых ячеек
- **table-layout**: определяет размеры таблицы

Установка таблицы

Ранее для установки границы в таблице широко использовался атрибут `border`, например:

```
<table border="2px" >
```

Сейчас же тенденция для стилизации использовать только стили CSS, поэтому граница также задается через CSS с помощью стандартного свойства **border**:

```
table {
    border: 1px solid #ccc;  /* граница всей таблицы */
}
tr {
    border: 1px solid #ccc;  /* границы между строками */
}
td, th {
    border: 1px solid #ccc;  /* границы между столбцами */
}
```

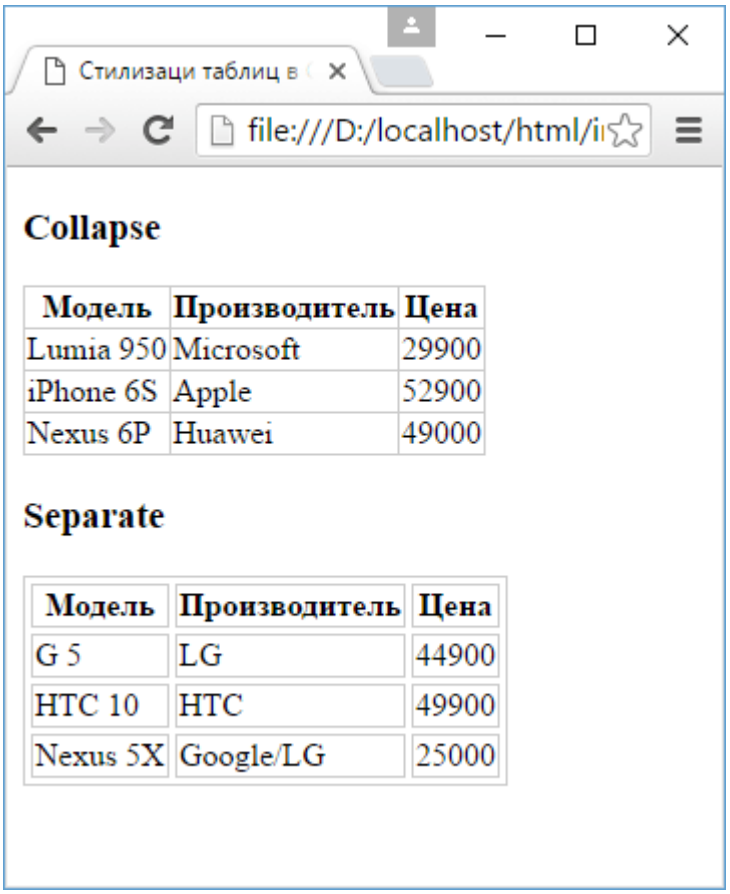
При установке границ между столбцами с помощью свойства **border-collapse** можно установить общую или раздельную границу между смежными ячейками:

- `collapse`: смежные ячейки имеют общую границу
- `separate`: смежные ячейки имеют отдельные границы, которые разделяются пространством

Если смежные ячейки имеют раздельные границы, то с помощью свойства **border-spacing** можно установить пространство между границами:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Стилизаци таблиц в CSS3</title>
        <style>
        table {
            border: 1px solid #ccc;
            border-spacing: 3px;
        }

        td, th{
```

```
            border: solid 1px #ccc;
        }
        .collapsed{
            border-collapse: collapse;
        }
        .separated{
            border-collapse: separate;
        }
        </style>
    </head>
    <body>
        <h3>Collapse</h3>
        <table class="collapsed">
            <tr><th>Модель</th><th>Производитель</th><th>Цена</th></tr>
            <tr><td>Lumia 950</td><td>Microsoft</td><td>29900</td></tr>
            <tr><td>iPhone 6S</td><td>Apple</td><td>52900</td></tr>
            <tr><td>Nexus 6P</td><td>Huawei</td><td>49000</td></tr>
        </table>
        <h3>Separate</h3>
        <table class="separated">
            <tr><th>Модель</th><th>Производитель</th><th>Цена</th></tr>
            <tr><td>G 5</td><td>LG</td><td>44900</td></tr>
            <tr><td>HTC 10</td><td>HTC</td><td>49900</td></tr>
            <tr><td>Nexus 5X</td><td>Google/LG</td><td>25000</td></tr>
        </table>
    </body>
</html>
```

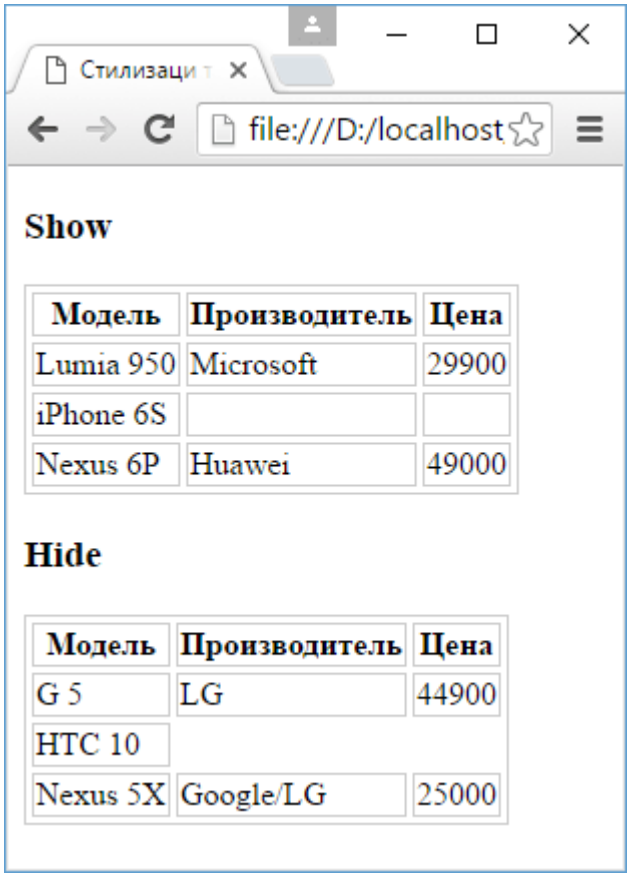


Пустые ячейки

Свойство **empty-cells** позволяет стилизовать пустые ячейки с помощью одного из следующих значений:

- `show`: пустые ячейки отображаются, значение по умолчанию
- `hide`: пустые ячейки не отображаются

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Стилизаци таблиц в CSS3</title>
        <style>
        table {
            border: 1px solid #ccc;
            border-spacing: 3px;
        }

        td, th{
            border: solid 1px #ccc;
        }
        .hidden-empty-cells{
            empty-cells: hide;
        }
        </style>
    </head>
    <body>
        <h3>Show</h3>
        <table>
            <tr><th>Модель</th><th>Производитель</th><th>Цена</th></tr>
            <tr><td>Lumia 950</td><td>Microsoft</td><td>29900</td></tr>
            <tr><td>iPhone 6S</td><td></td><td></td></tr>
            <tr><td>Nexus 6P</td><td>Huawei</td><td>49000</td></tr>
        </table>
        <h3>Hide</h3>
        <table class="hidden-empty-cells">
            <tr><th>Модель</th><th>Производитель</th><th>Цена</th></tr>
            <tr><td>G 5</td><td>LG</td><td>44900</td></tr>
            <tr><td>HTC 10</td><td></td><td></td></tr>
            <tr><td>Nexus 5X</td><td>Google/LG</td><td>25000</td></tr>
        </table>
    </body>
</html>
```



Позиционирование заголовка

Свойство **caption-side** управляет позицией заголовка и может принимать следующие значения:

- `top`: позиционирование заголовка вверху (значение по умолчанию)
- `bottom`: позиционирование заголовка внизу

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Стилизаци таблиц в CSS3</title>
        <style>
        table {
            border: 1px solid #ccc;
            border-spacing: 3px;
        }

        caption {

            font-weight: bold;
        }

        td, th{
            border: solid 1px #ccc;
        }
        .captionBottom{
            caption-side: bottom;
        }
        </style>
    </head>
```

```
25      <body>
26          <h3>Top</h3>
27          <table>
28              <caption>Флагманы 2015 года</caption>
29              <tr><th>Модель</th><th>Производитель</th><th>Цена</th></tr>
30              <tr><td>Lumia 950</td><td>Microsoft</td><td>29900</td></tr>
31              <tr><td>iPhone 6S</td><td>Apple</td><td>52900</td></tr>
32              <tr><td>Nexus 6P</td><td>Huawei</td><td>49000</td></tr>
33          </table>
34          <h3>Bottom</h3>
35          <table class="captionBottom">
36              <caption>Новинки 2016 года</caption>
37              <tr><th>Модель</th><th>Производитель</th><th>Цена</th></tr>
38              <tr><td>G 5</td><td>LG</td><td>44900</td></tr>
39              <tr><td>HTC 10</td><td>HTC</td><td>49900</td></tr>
40              <tr><td>iPhone SE</td><td>Apple</td><td>37000</td></tr>
41          </table>
42      </body>
43  </html>
```

Управление размером таблицы

С помощью свойства **table-layout** можно управлять размером таблицы. По умолчанию это свойство имеет значение `auto`, при котором браузер устанавливает ширину столбцов таблицы автоматически, исходя из ширины самой широкой ячейки в столбце. А из ширины отдельных столбцов складывается ширина всей таблицы.

Однако с помощью другого значения - `fixed` можно установить фиксированную ширину:

```
table {
    border: 1px solid #ccc;
    border-spacing: 3px;
    table-layout: fixed;
    width:350px;
}
```

Вертикальное выравнивание ячеек

Как правило, содержимое ячеек таблицы выравнивается по центру ячейки. Но с помощью свойства **vertical-align** это поведение можно переопределить. Это свойство принимает следующие значения:

- `top`: выравнивание содержимого по верху ячейки
- `baseline`: выравнивание первой строки текста по верху ячейки
- `middle`: выравнивание по центру (значение по умолчанию)
- `bottom`: выравнивание по низу

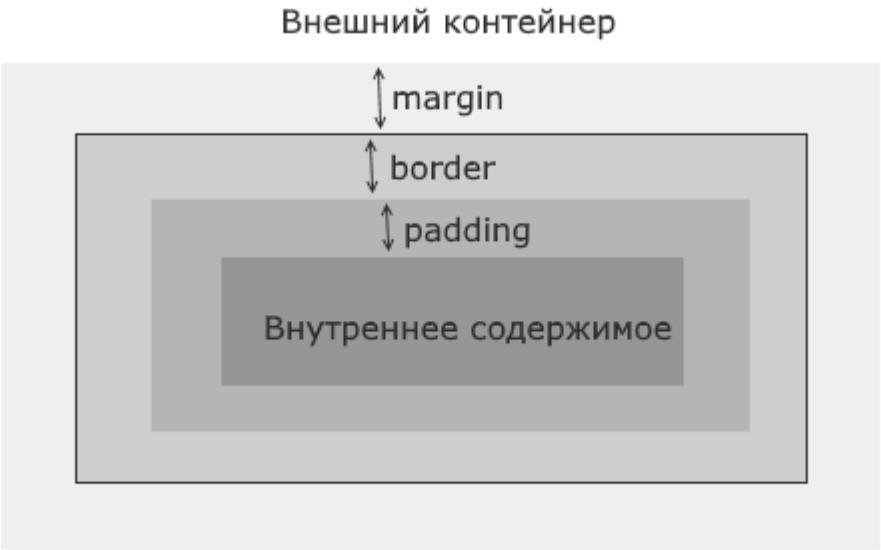
Свойство vertical-align применяется только к элементам `<th>` и `<td>`:

```
td, th{
    border: solid 1px #ccc;
    vertical-align: bottom;
    height: 30px;
}
```

Блочная модель

Для веб-браузера элементы страницы представляют небольшие контейнеры или блоки. Такие блоки могут иметь различное содержимое - текст, изображения, списки, таблицы и другие элементы. Внутренние элементы блоков сами выступают в качестве блоков.

Схематично блочную модель можно представить следующим образом:



Пусть элемент расположен в каком-нибудь внешнем контейнере. Это может быть элемент body, div и так далее. От других элементов он отделяется некоторым расстоянием - внешним отступом, которое описывается свойством CSS **margin**. То есть свойство margin определяет расстояние от границы текущего элемента до других соседних элементов или до границ внешнего контейнера.

Далее начинается сам элемент. И в начале идет его граница, которая в CSS описывается свойством **border**.

После границы идет внутренний отступ, который в CSS описывается свойством **padding**. Внутренний отступ определяет расстояние от границы элемента до внутреннего содержимого.

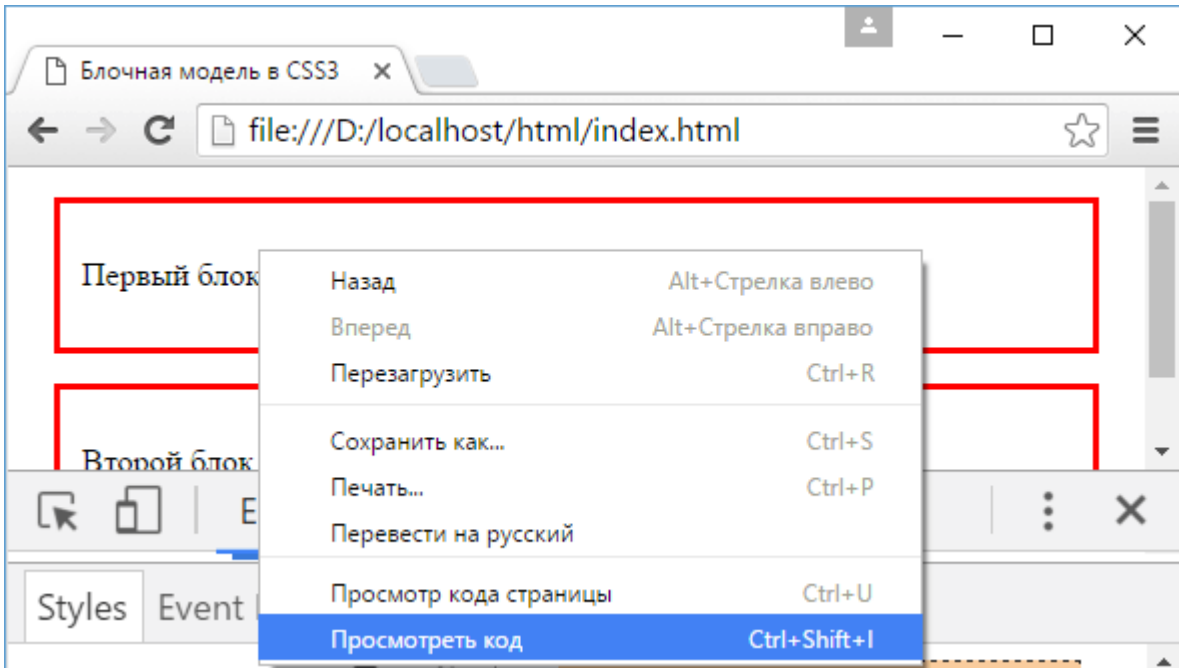
Далее идет внутреннее содержимое, которое также реализует ту же блочную модель и также может состоять из других элементов, которые имеют внешние и внутренние отступы и границу.

Например, определим следующую веб-страницу:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Блочная модель в CSS3</title>
        <style>
            div{
                margin: 15px; /* внешний отступ */
                padding: 11px; /* внутренний отступ */
                border: 3px solid red; /* границы шириной в 3 пикселя сплошной
                красной линией */
            }
```

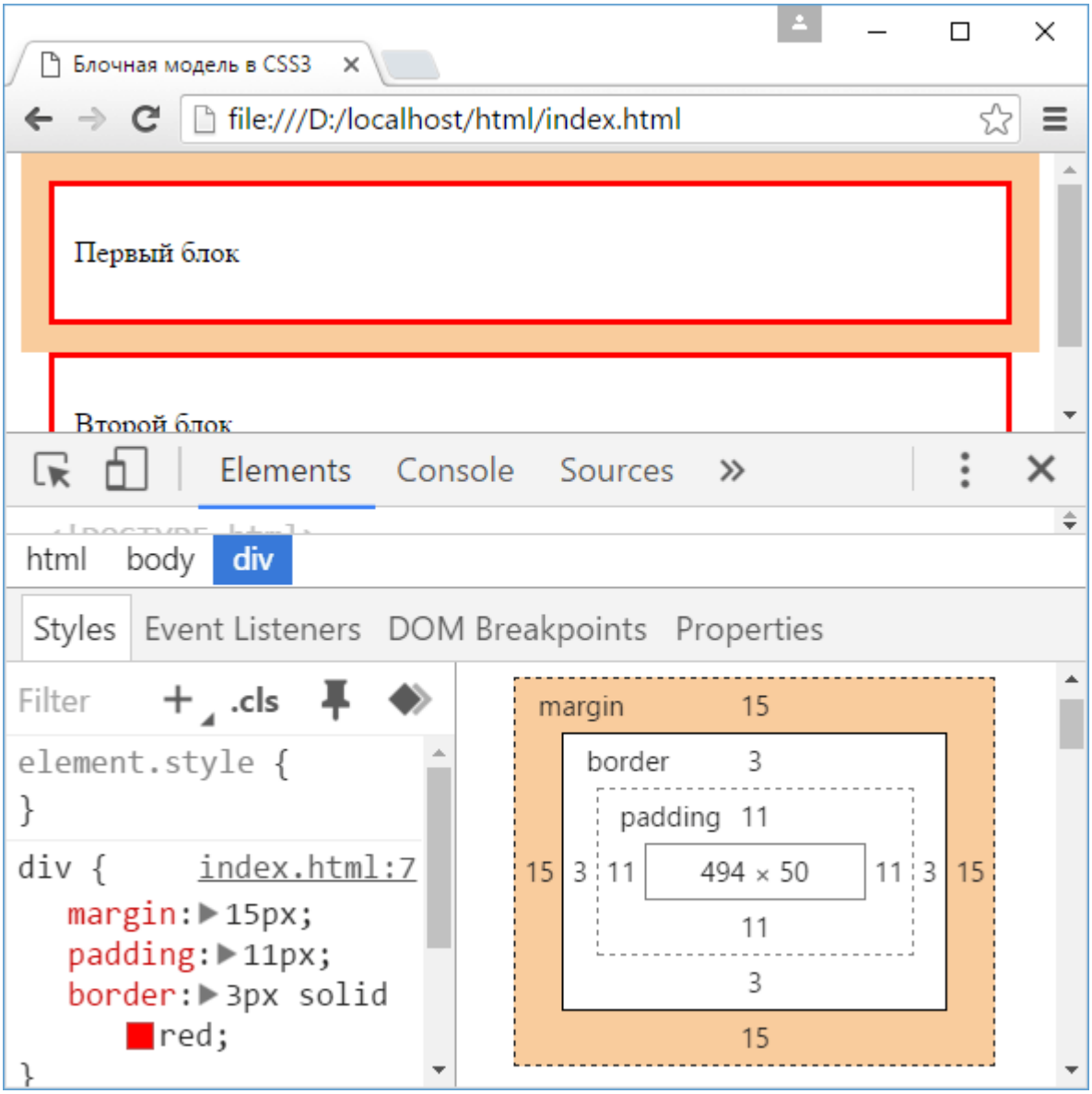
```
        </style>
    </head>
    <body>
        <div>
            <p>Первый блок</p>
        </div>
        <div>
            <p>Второй блок</p>
        </div>
    </body>
</html>
```

После запуска веб-страницы в браузере мы можем посмотреть блочную модель конкретных элементов. Для этого надо нажать на нужный элемент правой кнопкой мыши и открывающемся контекстном меню выбрать пункт, который позволяет просмотреть исходный код элемента. Для разных браузеров этот пункт может называться по разному. К примеру в Google Chrome это **Посмотреть код**:



В Mozilla Firefox этот пункт называется **Исследовать элемент**.

И по выбору данного пункта браузер откроет панель, где будет показан код элемента его стили и блочная модель:



В этой модели мы можем увидеть, как задаются отступы элемента, его граница, посмотреть отступы от других элементов и при необходимости динамически поменять значения их стилей.

Если мы явным образом не указываем значения свойств margin, padding и border, то браузер применяет предустановленные значения.

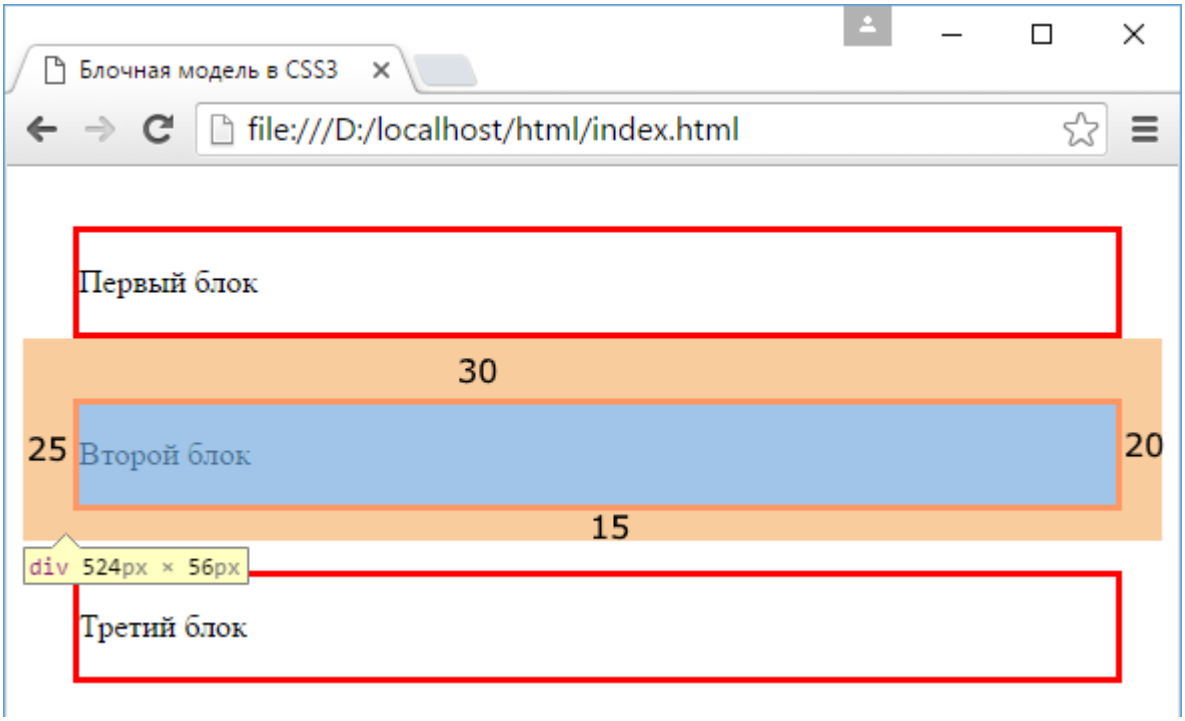
Внешние отступы

Свойство **margin** определяет отступ элемента от других элементов или границы внешнего контейнера. Существуют специальные свойства CSS для задания отступов для каждой стороны:

- **margin-top**: отступ сверху
- **margin-bottom**: отступ снизу
- **margin-left**: отступ слева
- **margin-right**: отступ справа

Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Блочная модель в CSS3</title>
        <style>
            div{
                margin-top: 30px; /* отступ сверху */
                margin-left: 25px; /* отступ слева */
                margin-right: 20px; /* отступ справа */
                margin-bottom: 15px; /* отступ снизу */

                border: 3px solid red; /* граница */
            }
        </style>
    </head>
    <body>
        <div>
            <p>Первый блок</p>
        </div>
        <div>
            <p>Второй блок</p>
        </div>
        <div>
            <p>Третий блок</p>
        </div>
    </body>
</html>
```



Можно вместо четырех свойств задать одно:

```
1   div{
2       margin: 30px 20px 15px 25px;
3
4       border: 3px solid red;
5   }
```

Свойство задается в формате:

```
1   margin: отступ_сверху отступ_справа отступ_снизу отступ_слева;
```

Если значения для всех четырех отступов совпадает, то мы можем указать только одно значение:

```
1   div{
2       margin: 25px;
3   }
```

В этом случае для всех четырех отступов будет использоваться 25 пикселей.

Для установки отступов можно использовать точные значения в пикселях (px) или em, либо процентные отношения, либо значение `auto` (автоматическая установка отступов).

Например:

```
1   margin-left: 2em;
```

Значение `2 em` определяет расстояние, которое в два раза больше размера шрифта элемента.

При использовании процентов веб-браузеры вычисляют размер отступов на основе ширины элемента-контейнера, в который заключен стилизуемый элемент.

В то же время, если несколько элементов у нас соприкасаются, то браузер выбирает наибольший отступ элемента, который затем и используется. Так, выше в примере использовался следующий стиль:

```
div{
    margin-top: 30px; /* отступ сверху */
    margin-left: 25px; /* отступ слева */
    margin-right: 20px; /* отступ справа */
    margin-bottom: 15px; /* отступ снизу */
}
```

Между первым и вторым блоками расстояние будет равно 30 пикселям - значение свойства `margin-top` второго блока div, несмотря на то, что у первого блока div имеется свойство `margin-bottom`, равное 15 пикселям.

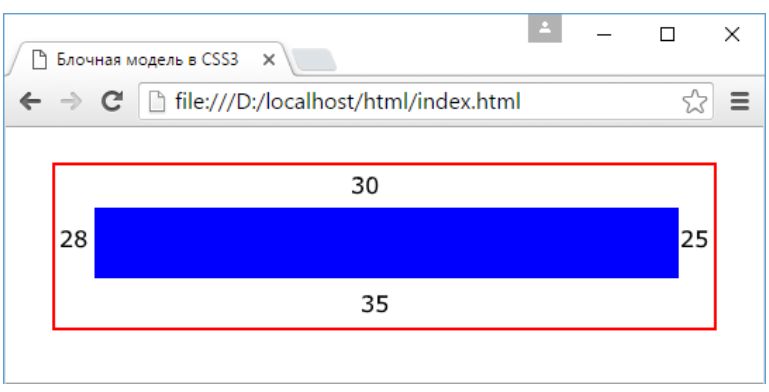
Внутренние отступы

Свойство **padding** задает внутренние отступы от границы элемента до его внутреннего содержимого. Как и для свойство margin, в CSS имеются четыре свойства, которые устанавливают отступы для каждой из сторон:

- **padding-top**: отступ сверху
- **padding-bottom**: отступ снизу
- **padding-left**: отступ слева
- **padding-right**: отступ справа

Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Блочная модель в CSS3</title>
        <style>
            div.outer{

                margin: 25px;

                padding-top:30px;
                padding-right: 25px;
                padding-bottom: 35px;
                padding-left: 28px;

                border: 2px solid red;
            }
            div.inner{

                height: 50px;
                background-color:blue;
            }
        </style>
    </head>
    <body>
        <div class="outer">
            <div class="inner"></div>
        </div>
    </body>
</html>
```



Для установки значения отступов, как и в margin, могут применяться либо конкретные значения в пикселях, так и процентные значения (относительно размеров элементов).

Для записи отступов также можно использовать сокращенную запись:

```
padding: отступ_сверху отступ_справа отступ_снизу отступ_слева;
```

Например:

```
div.outer{

    margin: 25px;

    padding: 30px 25px 35px 28px;

    border: 2px solid red;
}
```

Если все четыре значения совпадают, то можно писать указать только одно значение для всех отступов:

```
div.outer{

    margin: 25px;

    padding: 30px;

    border: 2px solid red;
}
```

Границы

Граница отделяется элемент от внешнего по отношению к нему содержимого. При этом граница является частью элемента.

Для настройки границы могут использоваться сразу несколько свойств:

- **border-width**: устанавливает ширину границы
- **border-style**: задает стиль линии границы
- **border-color**: устанавливает цвет границы

Свойство `border-width` может принимать следующие типы значений:

- Значения в единицах измерения, таких как em, px или cm

```
border-width: 2px;
```

- Одно из константных значений: `thin` (тонкая граница - 1px), `medium` (средняя по ширине - 3px), `thick` (толстая - 5px)

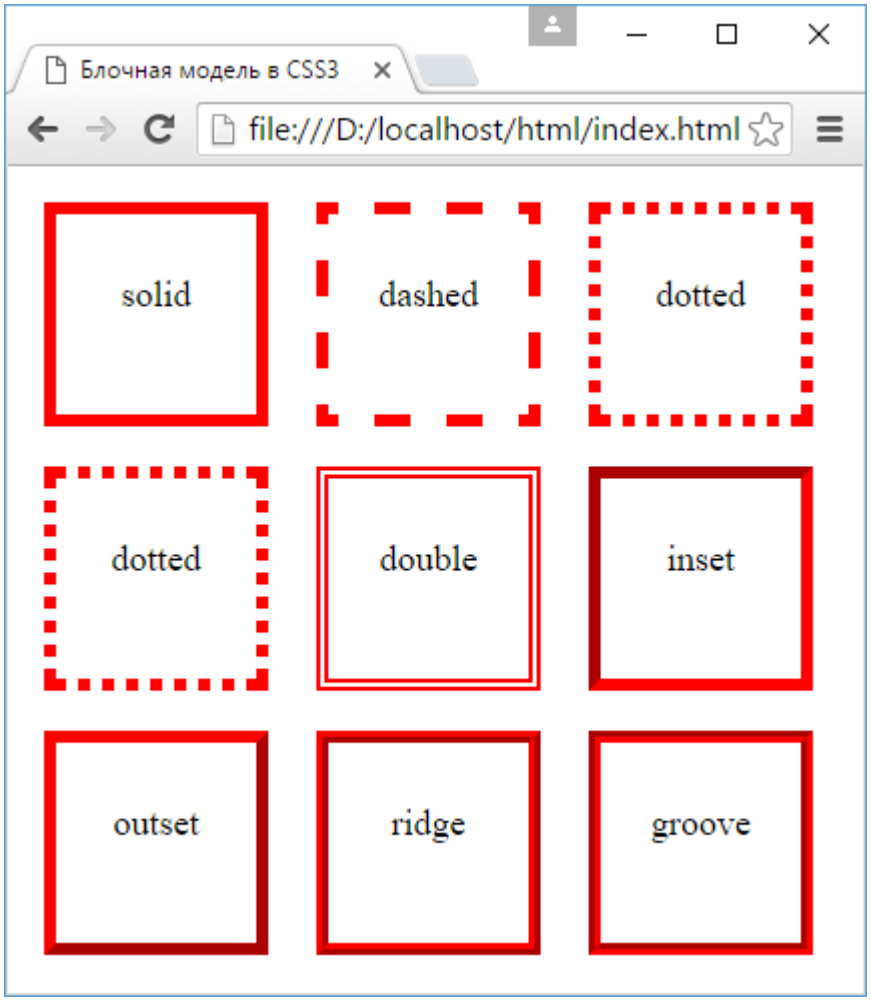
```
border-width: medium;
```

Свойство `border-color` в качестве значения принимает цвет CSS:

```
border-color: red;
```

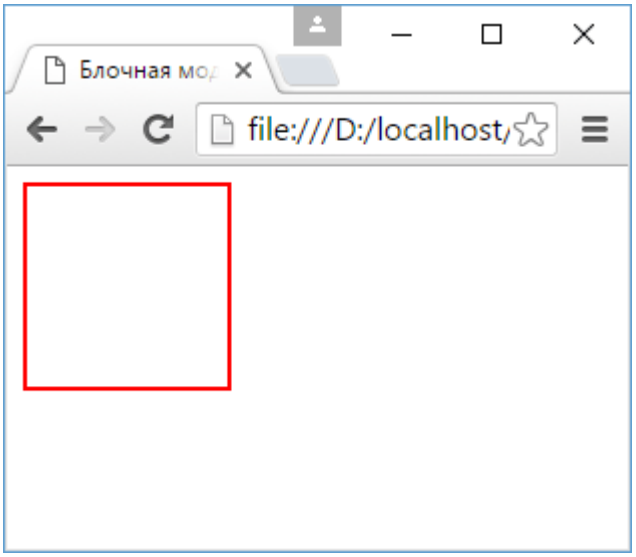
Свойство `border-style` оформляет тип линии границы и может принимать одно из следующих значений:

- `none`: граница отсутствует
- `solid`: граница в виде обычной линии
- `dashed`: штриховая линия
- `dotted`: линия в виде последовательности точек
- `double`: граница в виде двух параллельных линий
- `groove`: граница имеет трехмерный эффект
- `inset`: граница как бы вдавливается во внутрь
- `outset`: аналогично inset, только граница как бы выступает наружу
- `ridge`: граница также реализует трехмерный эффект



Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Блочная модель в CSS3</title>
        <style>
            div{
                width: 100px;
                height:100px;
                border-style: solid;
                border-color: red;
                border-width: 2px;
            }
        </style>
    </head>
    <body>
        <div></div>
    </body>
</html>
```



При необходимости мы можем определить цвет, стиль и ширину границы для каждой из сторон используя следующие свойства:

```
/* для верхней границы */
border-top-width
border-top-style
border-top-color

/* для нижней границы */
border-bottom-width
border-bottom-style
border-bottom-color

/* для левой границы */
border-left-width
border-left-style
border-left-color

/* для правой границы */
border-right-width
border-right-style
border-right-color
```

Свойство border

Вместо установки по отдельности цвета, стиля и ширины границы мы можем использовать одно свойство - **border**:

```
border: ширина стиль цвет
```

Например:

```
border: 2px solid red;
```

Для установки границы для отдельных сторон можно использовать одно из свойств:

```
border-top
border-bottom
border-left
```

```
4   border-right
```

Их использование аналогично:

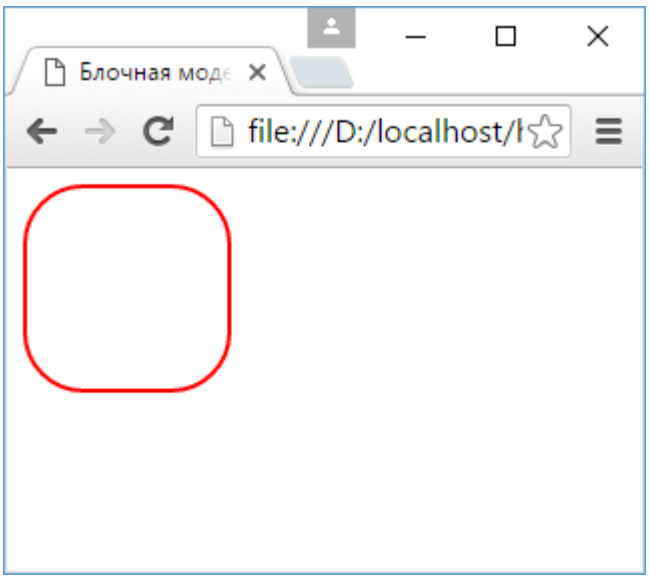
```
1   border-top: 2px solid red;
```

Радиус границы

Свойство **border-radius** позволяет округлить границу. Это свойство принимает значение радиуса в пикселях или единицах em.

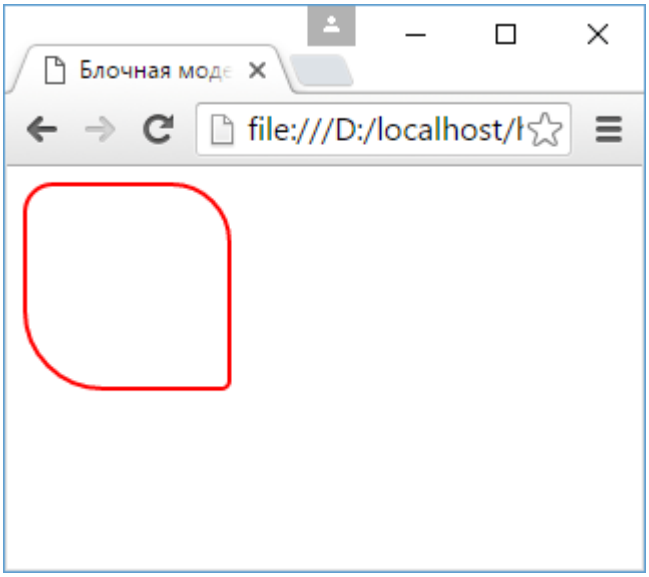
```
1   <!DOCTYPE html>
2   <html>
3       <head>
4           <meta charset="utf-8">
5           <title>Блочная модель в CSS3</title>
6           <style>
7               div{
8                   width: 100px;
9                   height:100px;
10                  border: 2px solid red;
11                  border-radius: 30px;
12              }
13          </style>
14      </head>
15      <body>
16          <div></div>
17      </body>
18  </html>
```

Теперь каждый угол будет скругляться по радиусу в 30 пикселей:



Так как у элемента может быть максимально четыре угла, то мы можем указать четыре значения для установки радиуса у каждого углов:

```
border-radius: 15px 30px 5px 40px;
```



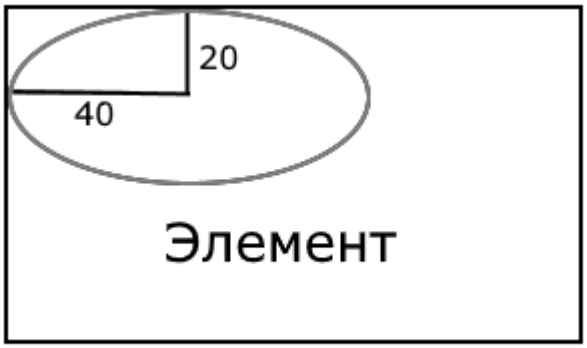
Вместо общей установки радиусов для всех углов, можно их устанавливать по отдельности. Так, предыдущее значение border-radius можно переписать следующим образом:

```
border-top-left-radius: 15px;   /* радиус для верхнего левого угла */
border-top-right-radius: 30px;  /* радиус для верхнего правого угла */
border-bottom-right-radius: 5px;    /* радиус для нижнего левого угла */
border-bottom-left-radius: 40px;    /* радиус для нижнего правого угла */
```

Также `border-radius` поддерживает возможность создания эллиптических углов. То есть угол не просто скругляется, а использует два радиуса, образуя в итоге душу эллипса:

```
border-radius: 40px/20px;
```

В данном случае полагается, что радиус по оси X будет иметь значение 40 пикселей, а по оси Y - 20 пикселей.



Размеры элементов

Размеры элементов задаются с помощью свойств **width** (ширина) и **height** (высота).

Значение по умолчанию для этих свойств - `auto`, то есть браузер сам определяет ширину и высоту элемента. Можно также явно задать размеры с помощью единиц измерения (пикселей, em) или с помощью процентов:

```
width: 150px;
width: 75%;
height: 15em;
```

Пиксели определяют точные ширину и высоту. Единица измерения **em** зависит от высоты шрифта в элементе. Если размер шрифта элемента, к примеру, равен 16 пикселей, то 1 em для этого элемента будет равен 16 пикселям. То есть если у элемента установить ширину в 15em, то фактически она составит 15 * 16 = 230 пикселей. Если же у элемента не определен размер шрифта, то он будет взят из унаследованных параметров или значений по умолчанию.

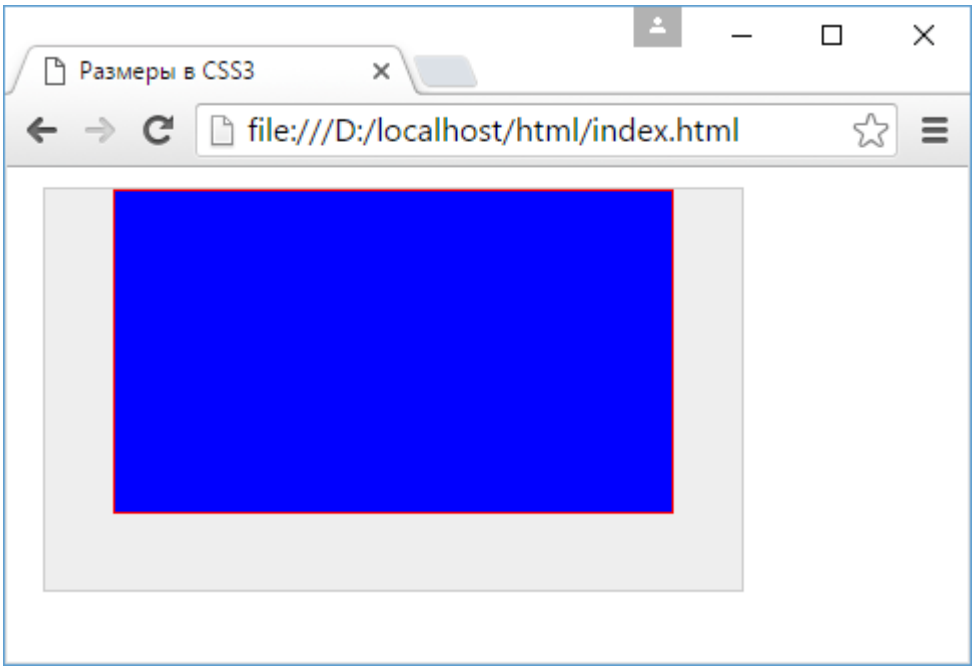
Процентные значения для свойства width вычисляются на основании ширины элемента-контейнера. Если, к примеру, ширина элемента body на веб-странице составляет 1000 пикселей, а вложенный в него элемент `<div>` имеет ширину 75%, то фактическая ширина этого блока `<div>` составляет 1000 * 0.75 = 750 пикселей. Если пользователь изменит размер окна браузера, то ширина элемента body и соответственно ширина вложенного в него блока div тоже изменится.

Процентные значения для свойства height работают аналогично свойству width, только теперь высота вычисляется по высоте элемента-контейнера.

Например:

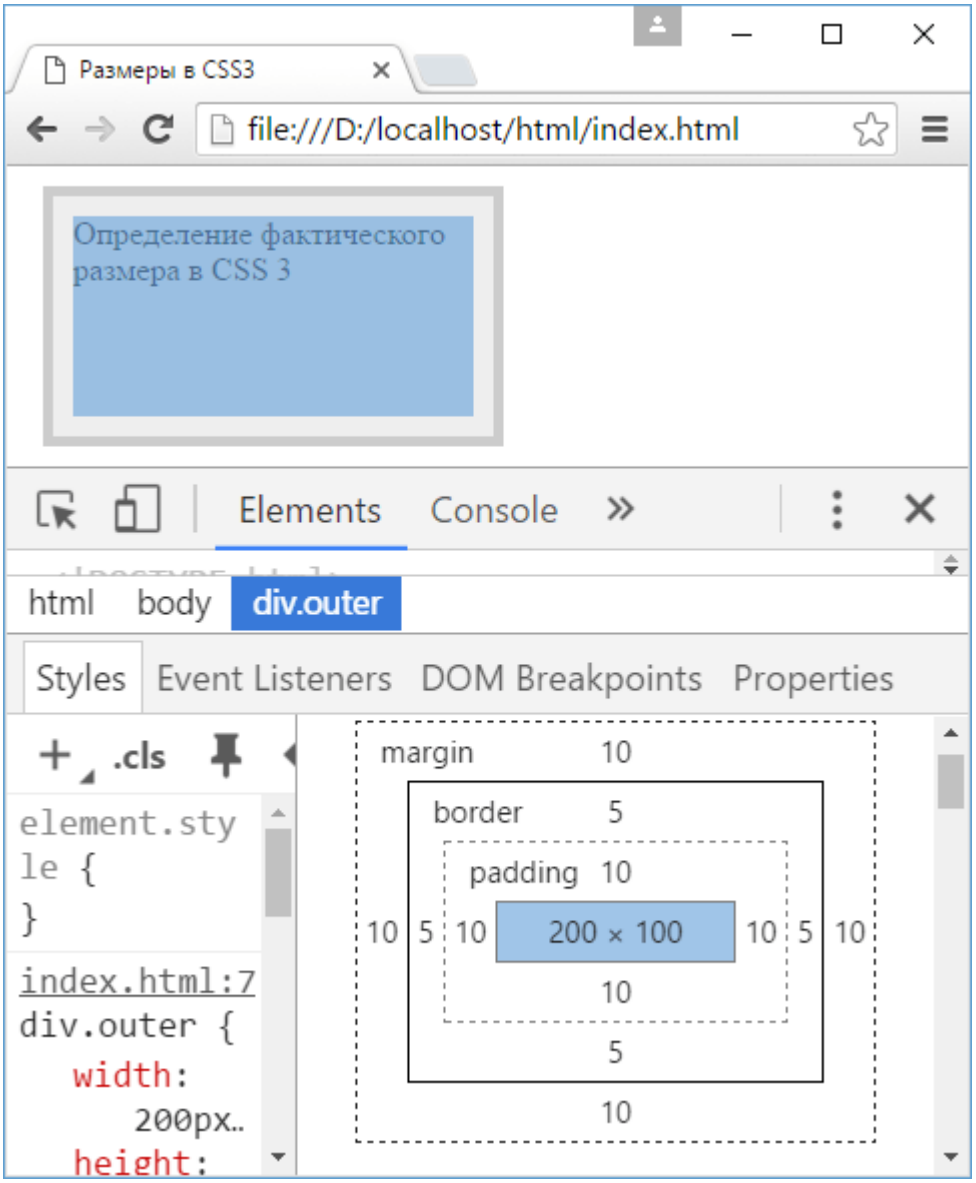
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Размеры в CSS3</title>
        <style>
        div.outer{
            width: 75%;
            height: 200px;
            margin: 10px;
            border: 1px solid #ccc;
            background-color: #eee;
        }
        div.inner{

            width: 80%;
            height: 80%;
            margin: auto;
            border: 1px solid red;
            background-color: blue;
        }
        </style>
    </head>
```

```
    <body>
        <div class="outer">
            <div class="inner"></div>
        </div>
    </body>
</html>
```



В то же время фактические размеры элемента могут в итоге отличаться от тех, которые установлены в свойствах `width` и `height`. Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Размеры в CSS3</title>
        <style>
        div.outer{
            width: 200px;
            height: 100px;
            margin: 10px;
            padding: 10px;
            border: 5px solid #ccc;
            background-color: #eee;
        }
        </style>
    </head>
    <body>
        <div class="outer">
            Определение фактического размера в CSS 3
        </div>
    </body>
</html>
```



Как видно на скриншоте, в реальности значение свойства `width` - 200px - определяет только ширину внутреннего содержимого элемента, а под блок самого элемента будет выделяться пространство, ширина которого равна ширине внутреннего содержимого (свойство width) + внутренние отступы (свойство padding) + ширина границы (свойство border-width) + внешние отступы (свойство margin). То есть элемент будет иметь ширину в 230 пикселей, а ширина блока элемента с учетом внешних отступов составит 250 пикселей.

Подобные расчеты следует учитывать при определении размеров элементов.

С помощью дополнительного набора свойств можно установить минимальные и максимальные размеры:

- **min-width**: минимальная ширина
- **max-width**: максимальная ширина
- **min-height**: минимальная высота
- **max-height**: максимальная высота

```
min-width: 200px;
width:50%;
max-width: 300px;
```

В данном случае ширина элемента равна 50% ширины элемента-контейнера, однако при этом не может быть меньше 200 пикселей и больше 300 пикселей.

Переопределение ширины блока

Свойство **box-sizing** позволяет переопределить установленные размеры элементов. Оно может принимать одно из следующих значений:

- `content-box`: значение свойства по умолчанию, при котором браузер для определения реальных ширины и высоты элементов добавляет берет соответственно значения свойств `width` и `height` элемента

  Например:

```
width: 200px;
height: 100px;
margin: 10px;
padding: 10px;
border: 5px solid #ccc;
background-color: #eee;
box-sizing: content-box;
```

  В данном случае элемент будет иметь ширину в 200 пикселей и высоту в 100 пиксей.

- `padding-box`: указывает веб-браузеру, что ширина и высота элемента должны включать внутренние отступы как часть своего значения. Например, пусть у нас есть следующий стиль:

```
width: 200px;
height: 100px;
margin: 10px;
padding: 10px;
border: 5px solid #ccc;
background-color: #eee;
box-sizing: padding-box;
```

- Здесь реальная ширина внутреннего содержимого блока будет равна 200px (width) - 10px (padding-left) - 10px (padding-right) = 180px.
- Стоит отметить, что большинство современных браузеров не поддерживают данное свойство.
- `border-box`: указывает веб-браузеру, что ширина и высота элемента должны включать внутренние отступы и границы как часть своего значения. Например, пусть у нас есть следующий стиль:

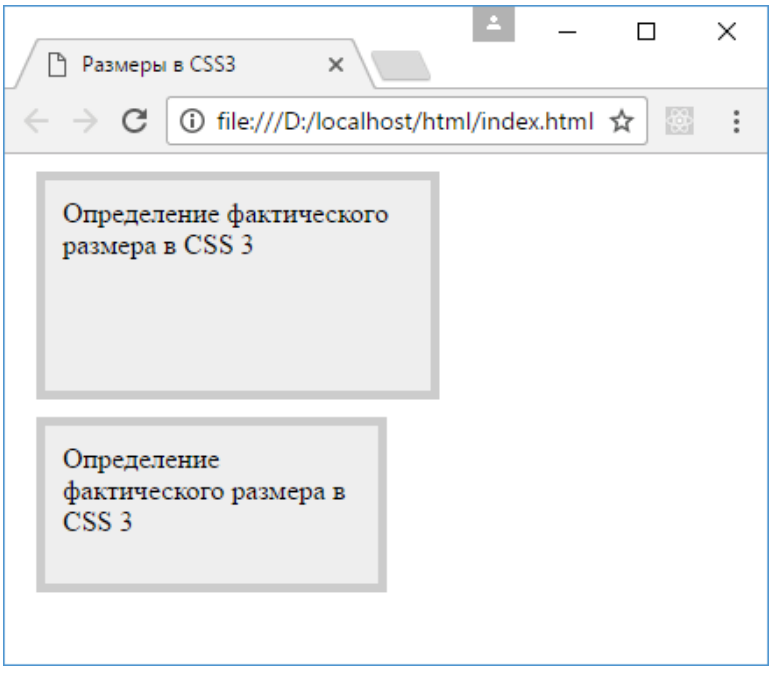
```
width: 200px;
height: 100px;
margin: 10px;
padding: 10px;
border: 5px solid #ccc;
background-color: #eee;
box-sizing: border-box;
```

- Здесь реальная ширина внутреннего содержимого блока будет равна 200px (width) - 10px (padding-left) - 10px (padding-right) - 5px (border-left-width) - 5px (border-right-width) = 170px.

Например, определим два блока, которые отличаются только значением свойства box-sizing:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Размеры в CSS3</title>
        <style>
        div{
            width: 200px;
            height: 100px;
            margin: 10px;
            padding: 10px;
            border: 5px solid #ccc;
            background-color: #eee;
        }
        div.outer1{
            box-sizing: content-box;
        }
        div.outer2{
            box-sizing: border-box;
        }
        </style>
    </head>
    <body>
        <div class="outer1">
            Определение фактического размера в CSS 3
        </div>
        <div class="outer2">
            Определение фактического размера в CSS 3
        </div>
    </body>
</html>
```

В первом случае при определении размеров блока к свойствам width и height будут добавляться толщина границы, а также внутренние и внешние отступы, поэтому первый блок будет иметь большие размеры:



Фон элемента

Фон элемента описывается в CSS свойством **background**. Фактически это свойство представляет сокращение набора следующих свойств CSS:

- **background-color**: устанавливает цвет фона

```
background-color: #ff0507;
```

- **background-image**: в качестве фона устанавливается изображение

```
background-image: url(dubi.png);
```

- Это свойство принимает одно значение: ключевое слово url, после которого в скобках идет путь к файлу изображения. В данном случае имеется в виду, что в одной папке рядом с веб-страницей находится файл `dubi.png`.
- Также можно использовать абсолютные адреса URL, например:

```
background-image: url(http://localhost.com/someimage.png);
```

- Либо можно использовать относительные адреса - относительно html-документа или корневого каталога сайта:

```
background-image: url(../images/someimage.png); /* путь относительно html-документа */
```

- **background-repeat**: устанавливает режим повторения фонового изображения по всей поверхности элемента
- **background-size**: устанавливает размер фонового изображения
- **background-position**: указывает позицию фонового изображения
- **background-attachment**: устанавливает стиль прикрепления фонового изображения к элементу
- **background-clip**: определяет область, которая вырезается из изображения и используется в качестве фона
- **background-origin**: устанавливает начальную позицию фонового изображения

Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Блочная модель в CSS3</title>
        <style>
        div{
            width: 250px;
            height: 200px;
            margin: 10px;
        }

        .colored{
            background-color: #ff0507;
        }

```

```
17          .imaged{
18              background-image: url(dubi.png);
19          }
20          </style>
21      </head>
22      <body>
23          <div class="colored">Первый блок</div>
24          <div class="imaged">Второй блок</div>
25      </body>
26  </html>
```

Первый блок окрашен в оттенок красного цвета, а второй блок устанавливает в качестве фона изображение. Все содержимое блока накладывается поверх фона:

Повторение изображения

На выше приведенном скриншоте видно, что CSS должным образом масштабирует изображение, чтобы наиболее оптимально вписать его в пространство элемента. Однако в связи с масшатбированием изображение может не полностью покрывать поверхность элемента, и поэтому для полного покрытия автоматически CSS начинает повторять изображение.

С помощью свойства **background-repeat** можно изменить механизм повторения. Оно может принимать следующие значения:

- `repeat-x`: повторение по горизонтали
- `repeat-y`: повторение по вертикали

- `repeat`: повторение по обеим сторонам (действие по умолчанию)
- `space`: изображение повторяется для заполнения всей поверхности элемента, но без создания фрагментов
- `round`: изображение должным образом масштабируется для полного заполнения всего пространства
- `no-repeat`: изображение не повторяется

Например:

```
div{
    width: 200px;
    height: 150px;

    background-image: url(dubi.png);
    background-repeat: round;
}
```

Размер изображения

Свойство **background-size** позволяет установить размер фонового изображения. Для установки размера можно использовать либо единицы измерения, например, пиксели, либо проценты, либо одно из предустановленных значений:

- `contain`: масштабирует изображение по наибольшей стороне, сохраняя аспектное отношение
- `cover`: масштабирует изображение по наименьшей стороне, сохраняя аспектное отношение
- `auto`: значение по умолчанию, изображение отображается в полный размер

Если нужно масштабировать изображение таким образом, чтобы оно оптимальнее было вписано в фон, то для обеих настроек можно установить значение 100%:

```
background-size: 100% 100%;
```

Если задаются точные размеры, то вначале указывается ширина, а потом высота изображения:

```
background-size: 200px 150px;   /* ширина 200 пикселей, высота 150 пикселей */
```

Можно задать точное значение для одного измерения - ширины или высоты, а для другого задать автоматические размеры, чтобы браузер сам выводил точные значения:

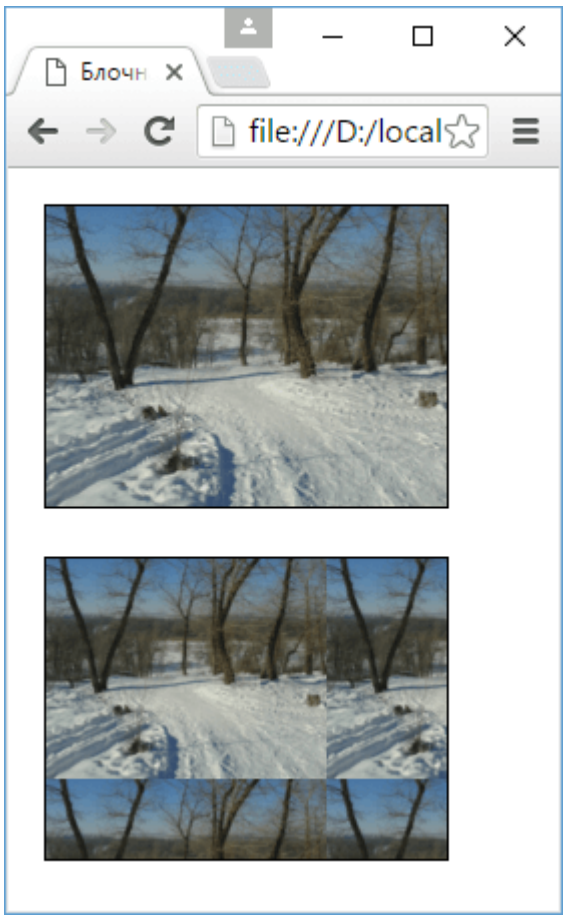
```
background-size: 200px auto;    /* ширина 200 пикселей, автоматическая высота */
```

Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Блочная модель в CSS3</title>
        <style>
        div{
            width: 200px;
            height: 150px;
            margin: 10px;
```

```
11
12              border: black solid 1px;
13              background-image: url(dubi.png);
14
15          }
16          .imaged1{
17
18              background-size: cover;
19          }
20          .imaged2{
21
22              background-size: 140px 110px;
23          }
24          </style>
25      </head>
26      <body>
27          <div class="imaged1"></div>
28          <div class="imaged2"></div>
29      </body>
30  </html>
```

Во втором случае изображение будет масштабироваться до размеров 140x110. Поскольку у нас еще остается место на элементе, то по умолчанию изображение будет повторяться для заполнения всей поверхности:



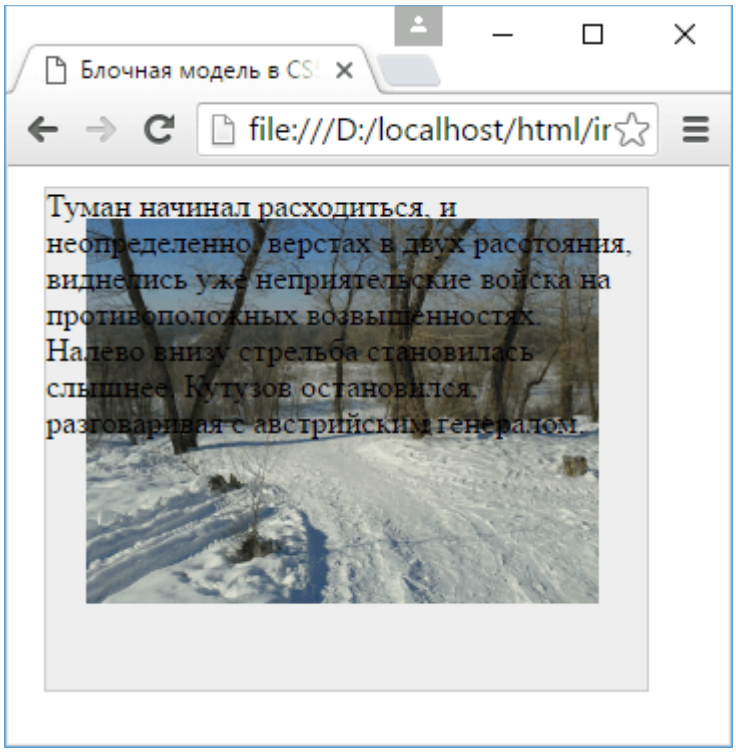
Позиция изображения

Свойство **background-position** управляет позицией фонового изображения внутри элемента. Оно может принимать отступы от верхнего левого угла элемента в единицах измерения, например, в пикселях в следующем формате:

```
background-position: отступ_по_оси_X отступ_по_оси_Y;
```

Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Блочная модель в CSS3</title>
        <style>
        div{
            width: 300px;
            height: 250px;
            margin: 10px;

            border: 1px solid #ccc;
            background-color: #eee;
            background-image: url(dubi.png);
            background-repeat: no-repeat;
            background-position: 20px 15px;
        }
        </style>
    </head>
    <body>
        <div>Туман начинал расходиться, и неопределенно, верстах в двух расстояния,
        виднелись уже неприятельские
            войска на противоположных возвышенностях...</div>
    </body>
</html>
```



Кроме того, данное свойство может принимать одно из следующих значений:

- `top`: выравнивание по верхнему краю элемента
- `left`: выравнивание по левому краю элемента

- `right`: выравнивание по правому краю элемента
- `bottom`: выравнивание по нижнему краю элемента
- `center`: изображение располагается по центру элемента

Например:

```
background-position: top right;
```

Здесь изображение выравнивается по верху и правому краю, то есть будет располагаться в правом верхнем углу элемента.

background-attachment

Свойство **background-attachment** управляет, как фоновое изображение будет прикреплено к элементу. Это свойство может принимать следующие значения:

- `fixed`: фон элемента фиксирован вне зависимости от прокрутки внутри элемента
- `local`: по мере прокрутки внутри элемента фон изменяется
- `scroll`: фон фиксирован и не меняется при прокрутке, но в отличие от fixed несколько элементов могут использовать свой фон, тогда как при fixed создается один фон для всех элементов

Например:

```
div{
    width: 300px;
    height: 250px;


    overflow:scroll;    /* добавляем прокрутку  */
    border: 1px solid #ccc;

    background-image: url(dubi.png);
    background-size: 512px 384px;
    background-attachment: scroll;
    background-repeat: no-repeat;
}
```

background-origin

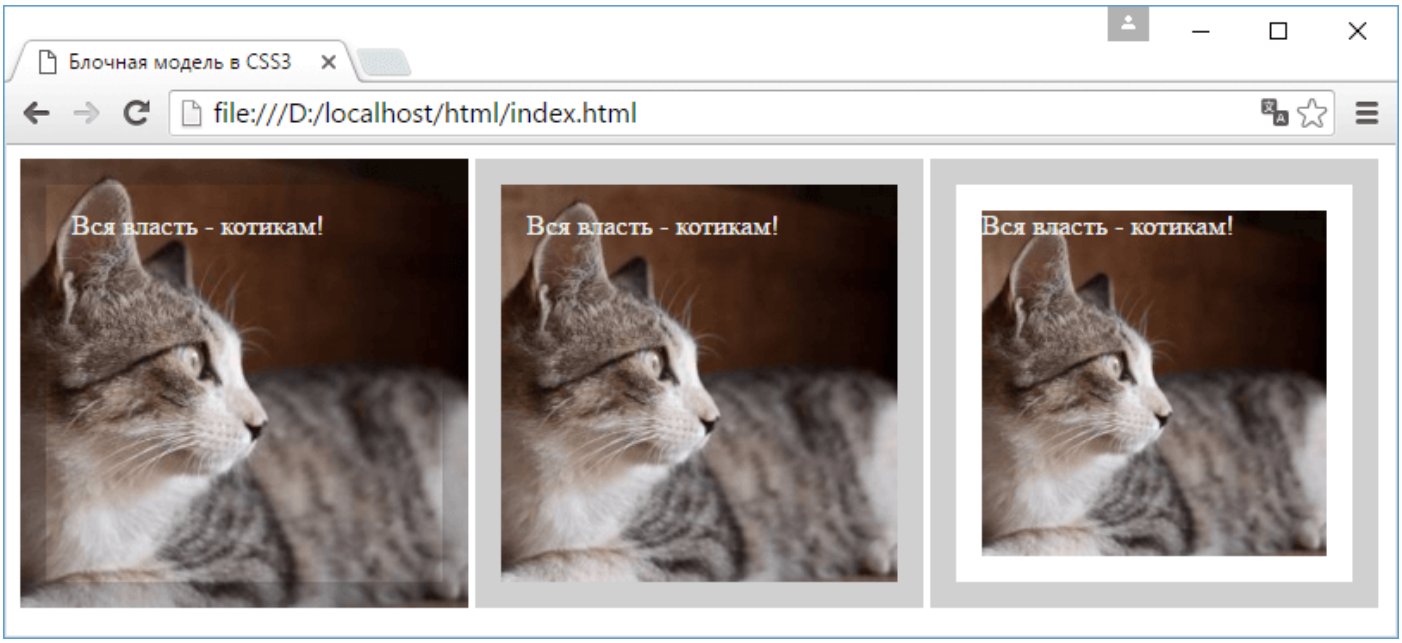
Свойство **background-origin** указывает позицию на изображении, с которой будет начинаться собственно фоновое изображение для элемента. Оно может принимать следующие значения:

- `border-box`: фон у элемента устанавливается начиная с его внешней границы, определяемой свойством border
- `padding-box`: фон устанавливается с учетом внутренних отступов
- `content-box`: фон устанавливается по содержимому элемента

Используем все три значения:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
```

```
5           <title>Блочная модель в CSS3</title>
6           <style>
7           div{
8               width: 200px;
9               height: 200px;
10              margin: 10px;
11              display: inline-block;  /* располагаем блоки в ряд */
12              color: #eee;
13
14              padding:15px;
15              border: 15px solid rgba(23,23,23,0.2);
16
17              background-image: url(cats.jpg);
18              background-size: cover;
19              background-repeat: no-repeat;
20          }
21          .borderBox {background-origin: border-box;}
22          .paddingBox {background-origin: padding-box;}
23          .contentBox {background-origin: content-box;}
24          </style>
25      </head>
26      <body>
27          <div class="borderBox">
28              Вся власть - котикам!
29          </div>
30          <div class="paddingBox">
31              Вся власть - котикам!
32          </div>
33          <div class="contentBox">
34              Вся власть - котикам!
35          </div>
36      </body>
37  </html>
```



background-clip

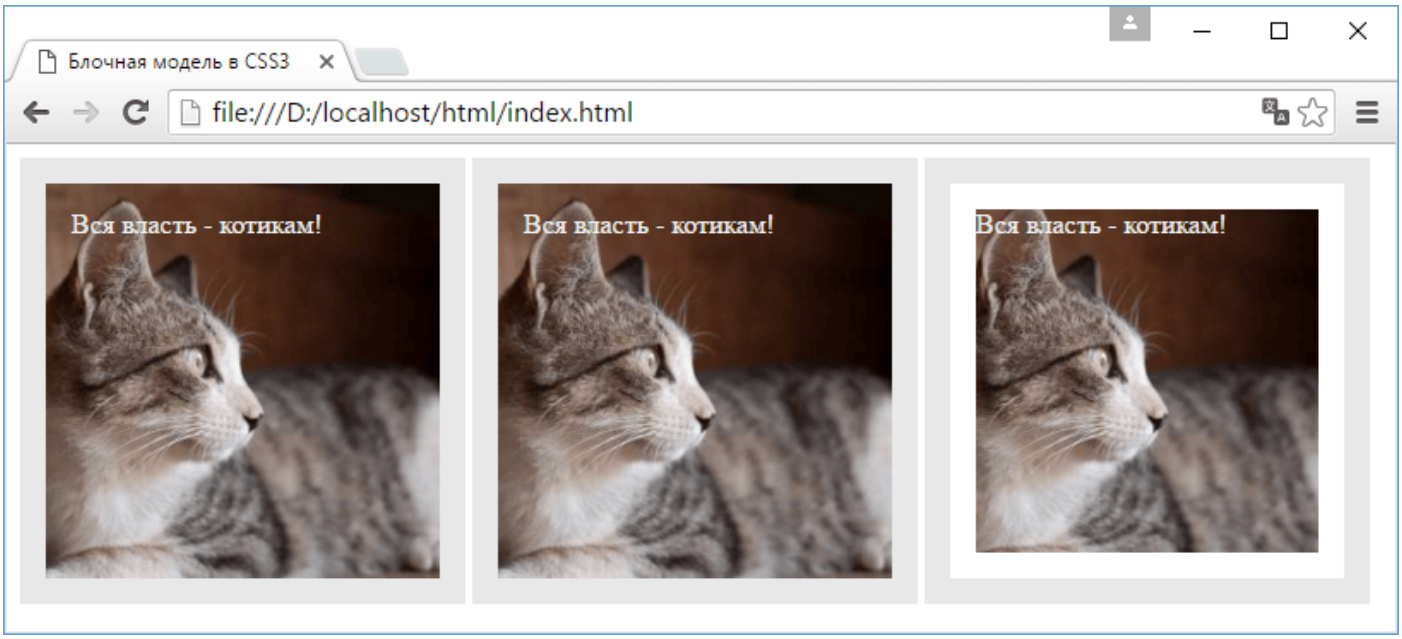
Свойство **background-clip** определяет, какая часть изображения используется для фона. Он принимает те же значения:

- `border-box`: изображение обрезается по границам элемента
- `padding-box`: из изображения исключается та часть, которая находится под границами элемента
- `content-box`: изображение обрезается по содержимому с учетом внутренних отступов

Например, если к предыдущей разметке мы применим следующие стили:

```
div{
    width: 200px;
    height: 200px;
    margin: 10px;
    display: inline-block;

    color: #eee;
    padding:15px;
    border: 15px solid rgba(23,23,23,0.1);

    background-image: url(cats.jpg);
    background-size: cover;
    background-repeat: no-repeat;
}
.borderBox{background-clip: border-box;}
.paddingBox{background-clip: padding-box;}
.contentBox{background-clip: content-box;}
```

Тогда мы получим следующий результат:



Свойство background

Свойство background по сути является сокращением всех ранее рассмотренных свойств CSS в формате:

```
background: <background-color> <background-position> <background-size>
<background-repeat> <background-origin> <background-clip> <background-attachment>
<background-image>
```

Например, если у нас есть следующий набор свойств:

```
background-image: url(cats.jpg);
background-color: #eee;
background-repeat: no-repeat;
background-clip: border-box;
background-origin: border-box;
background-attachment: local;
```

То мы их можем сократить следующим образом:

```
background: #eee no-repeat border-box local url(cats.jpg);
```

Создание тени у элемента

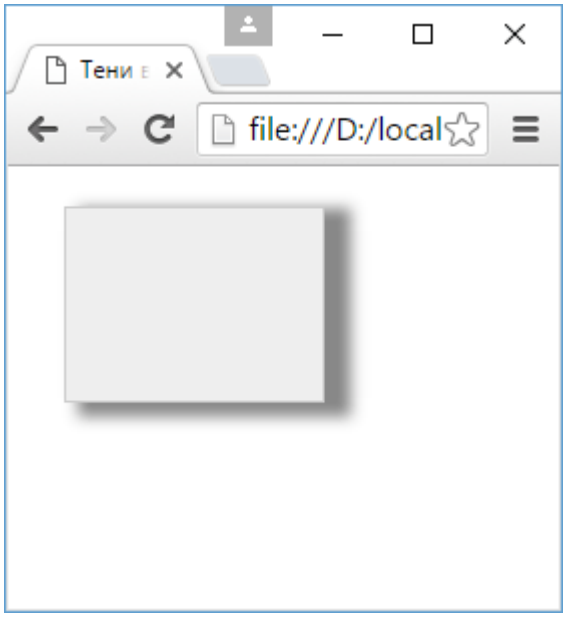
Свойство **box-shadow** позволяет создать у элемента тень. Это свойство может принимать сразу несколько значений:

```
box-shadow: hoffset voffset blur spread color inset
```

- `hoffset`: горизонтальное смещение тени относительно элемента. При положительном значении тень смещается вправо, а при отрицательном - влево
- `voffset`: вертикальное смещение тени относительно элемента. При положительном значении тень смещается вниз, а при отрицательном - вверх
- `blur`: необязательное значение, которое определяет радиус размытия тени. Чем больше это значение, тем более размытыми будут края тени. По умолчанию имеет значение 0.
- `spread`: необязательное значение, которое определяет направление тени. Положительное значение распространяет тень во вне во всех направлениях от элемента, а отрицательное значение направляет тень к элементу
- `color`: необязательное значение, которое устанавливает цвет тени
- `inset`: необязательное значение, которое заставляет рисовать тент внутри блока элемента

Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Тени в CSS3</title>
        <style>
        div{
            width: 128px;
            height: 96px;
            margin: 20px;
            border: 1px solid #ccc;
            background-color: #eee;
            box-shadow: 10px 4px 10px 3px #888;
        }
        </style>
    </head>
    <body>
        <div></div>
    </body>
</html>
```



Через запятую можно определить несколько различных теней:

```
box-shadow: 5px 3px 8px 3px #faa, 10px 4px 10px 3px #888 inset;
```

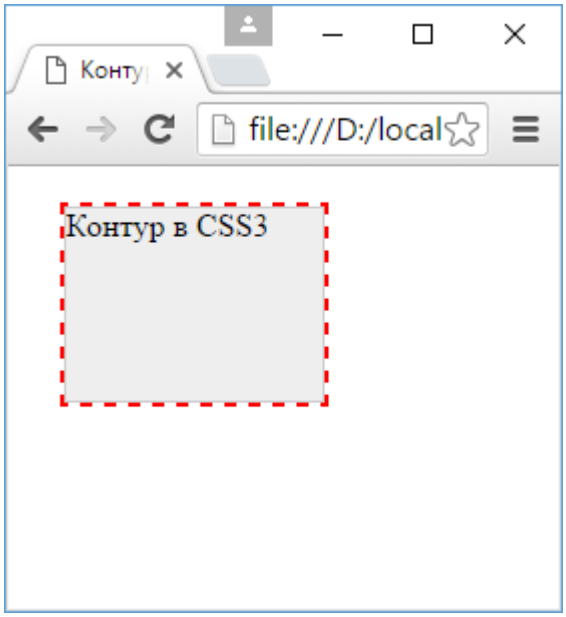
Контуры элементов

Концепция контуров похожа на использование границ у элементов. Не стоит путать или заменять границы контурами, они имеют разное значение. Контуры полезны тем, что позволяют выделить элемент, чтобы привлечь к нему внимание в какой-то ситуации. Контуры располагаются вне элемента сразу за его границами.

Контур в CSS 3 представлен свойством **outline**, хотя данное свойство является сокращением следующих свойств:

- **outline-color**: цвет контура
- **outline-offset**: смещение контура
- **outline-style**: стиль контура. Оно принимает те же значения, что и `border-style`:
  - `none`: контур отсутствует
  - `solid`: контур в виде обычной линии
  - `dashed`: штриховая линия
  - `dotted`: линия в виде последовательности точек
  - `double`: контур в виде двух параллельных линий
- **outline-width**: толщина контура

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Контур в CSS3</title>
        <style>
        div{
            width: 128px;
            height: 96px;
            margin: 20px;
            border: 1px solid #ccc;
            background-color: #eee;
            outline-color: red;
            outline-style: dashed;
            outline-width: 2px;
        }
        </style>
    </head>
    <body>
        <div>Контур в CSS3</div>
    </body>
</html>
```



С помощью свойства `outline` данное определение контура можно сократить следующим образом:

```
1   outline: red dashed 2px;
```

Обтекание элементов

Как правило, все блоки и элементы на веб-странице в браузере появляются в том порядке, в каком они определены в коде html. Однако CSS предоставляет специальное свойство **float**, которое позволяет установить обтекание элементов, благодаря чему мы можем создать более интересные и разнообразные по своему дизайну веб-страницы.

Это свойство может принимать одно из следующих значений:

- `left`: элемент перемещается влево, а все содержимое, которое идет ниже его, обтекает правый край элемента
- `right`: элемент перемещается вправо
- `none`: отменяет обтекание и возвращает объект в его обычную позицию

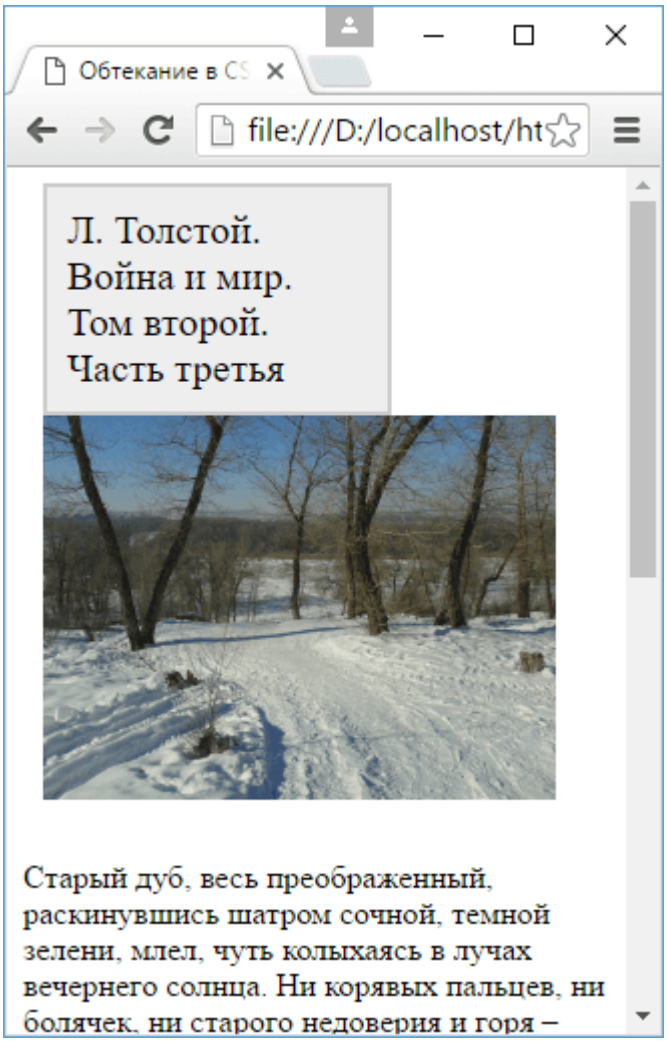
При применении свойства **float** для стилизуемых элементов, кроме элемента img, рекомендуется установить свойство width.

Итак, представим, что нам надо на странице вывести слева от основного текста изображение, справа должен быть сайдбар, а все остальное место должно быть занято основным текстом статьи. Определим интерфейс страницы сначала без свойства float:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Обтекание в CSS3</title>
        <style>

        .image {
            margin:10px;
            margin-top:0px;
        }
        .sidebar{
            border: 2px solid #ccc;
            background-color: #eee;
            width: 150px;
            padding: 10px;
            margin-left:10px;
            font-size: 20px;
        }
        </style>
    </head>
    <body>
        <div>
            <div class="sidebar">Л. Толстой. Война и мир. Том второй.
            Часть третья</div>
            <img src="dubi.png" class="image" alt="Война и мир" />
            <p>Старый дуб, весь преображенный, раскинувшись шатром сочной,
            темной зелени, млел, чуть колыхаясь в лучах вечернего солнца...</p>
            <p>«Нет, жизнь не кончена в 31 год, – вдруг окончательно,
            бесповоротно решил князь Андрей...</p>
        </div>
    </body>
```

```
31    </html>
```

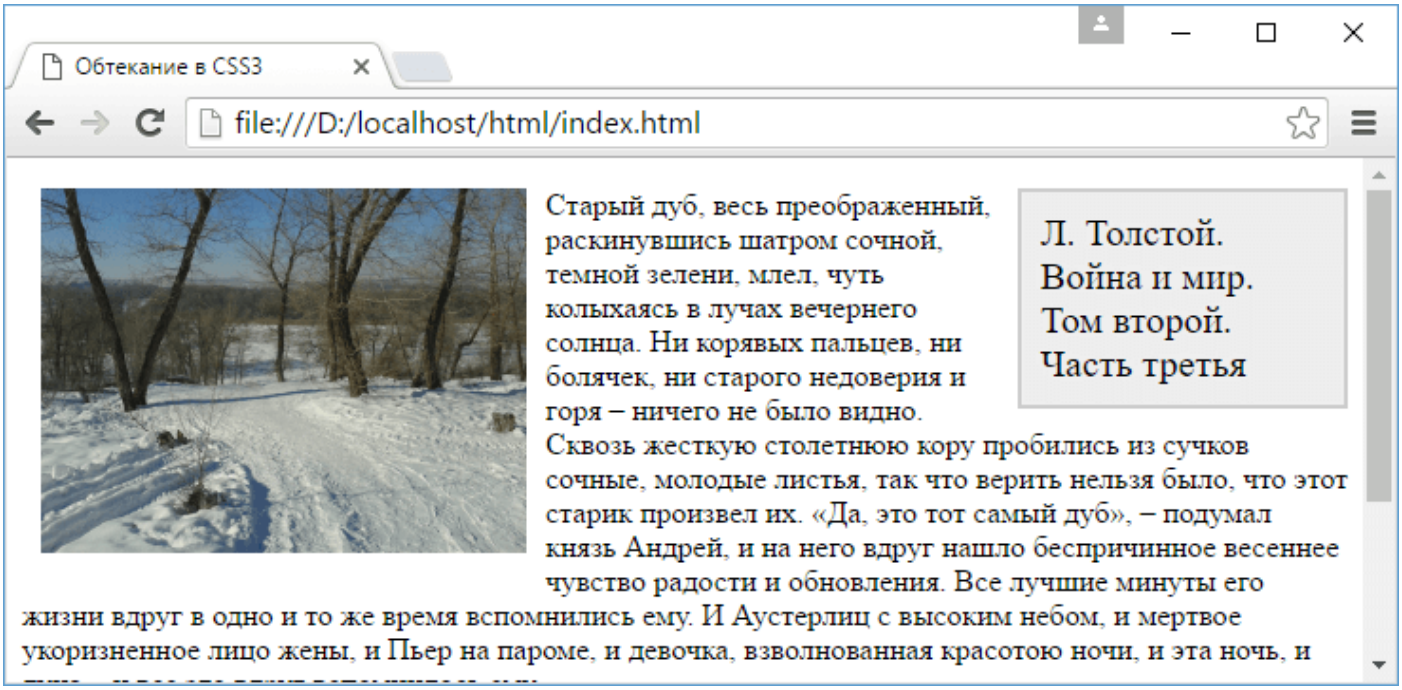
В данном случае мы получим последовательное размещение элементов на странице:



Теперь на той же странице применим свойство `float`, изменив стили следующим образом:

```
1     .image {
2         float:left; /* обтекание слева */
3         margin:10px;
4         margin-top:0px;
5     }
6     .sidebar{
7         border: 2px solid #ccc;
8         background-color: #eee;
9         width: 150px;
10        padding: 10px;
11        margin-left:10px;
12        font-size: 20px;
13        float: right; /* обтекание справа */
14    }
```

Соответственно изменится и размещение элементов на странице:



Элементы, к которым применяется свойство `float`, еще называют floating elements или плавающими элементами.

Запрет обтекания. Свойство clear

Иногда возникает необходимость запретить обтекания. Подобная задача может быть актуальна, если какой-то блок должен переноситься вниз на новую строку, а не обтекать плавающий элемент. Например, футер, как правило, должен находиться строго внизу и растягиваться по всей ширине страницы. Если же перед футером находится плавающий элемент, то футер может обтекать этот элемент, что не желательно.

Для запрета обтекания элементов в CSS применяется свойство **clear**, которое указывает браузеру, что к стилизуемому элементу не должно применяться обтекание.

Свойство `clear` может принимать следующие значения:

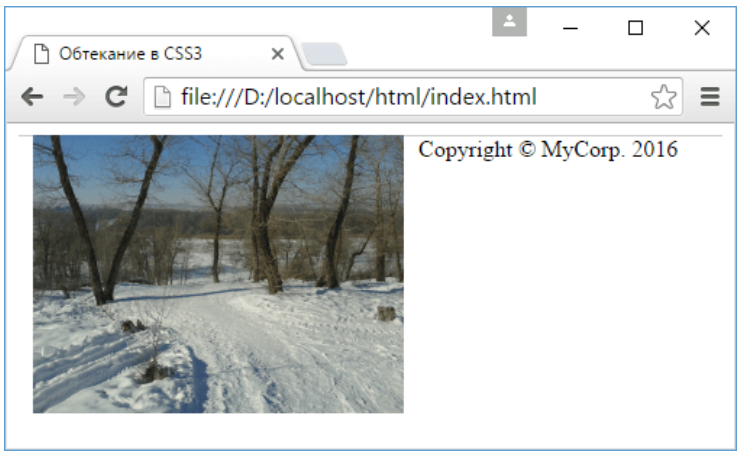
- `left`: стилизуемый элемент может обтекать плавающий элемент справа. Слева же обтекание не работает
- `right`: стилизуемый элемент может обтекать плавающий элемент только слева. А справа обтекание не работает
- `both`: стилизуемый элемент может обтекать плавающие элементы и относительно них смещается вниз
- `none`: стилизуемый элемент ведет себя стандартным образом, то есть принимает участие в обтекании справа и слева

Например, пусть на веб-странице будет определен футер:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Обтекание в CSS3</title>
        <style>
```

```
            .image {
                float:left;
                margin:10px;
                margin-top:0px;
            }
            .footer{
                border-top: 1px solid #ccc;
            }
            </style>
        </head>
        <body>

            <img src="dubi.png" class="image" alt="Дубы" />
            <div class="footer">Copyright © MyCorp. 2016</div>
        </body>
    </html>
```

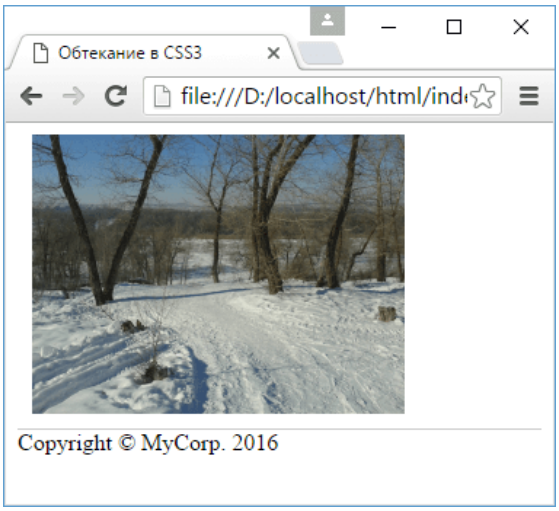
Наличие обтекания будет создавать некорректное отображение, при котором футер смещается вверх:



Изменим стиль футера:

```
    .footer{
        border-top: 1px solid #ccc;
        clear: both;
    }
```

Теперь футер не будет обтекать изображение, а будет уходить вниз.



Прокрутка элементов

Нередко при создании веб-страниц можно столкнуться с ситуацией, когда содержимое блока занимает гораздо больше места, чем сам определено шириной и высотой блока. В этой ситуации по умолчанию браузер все равно отображает содержимое, даже если оно выходит за границы блока.

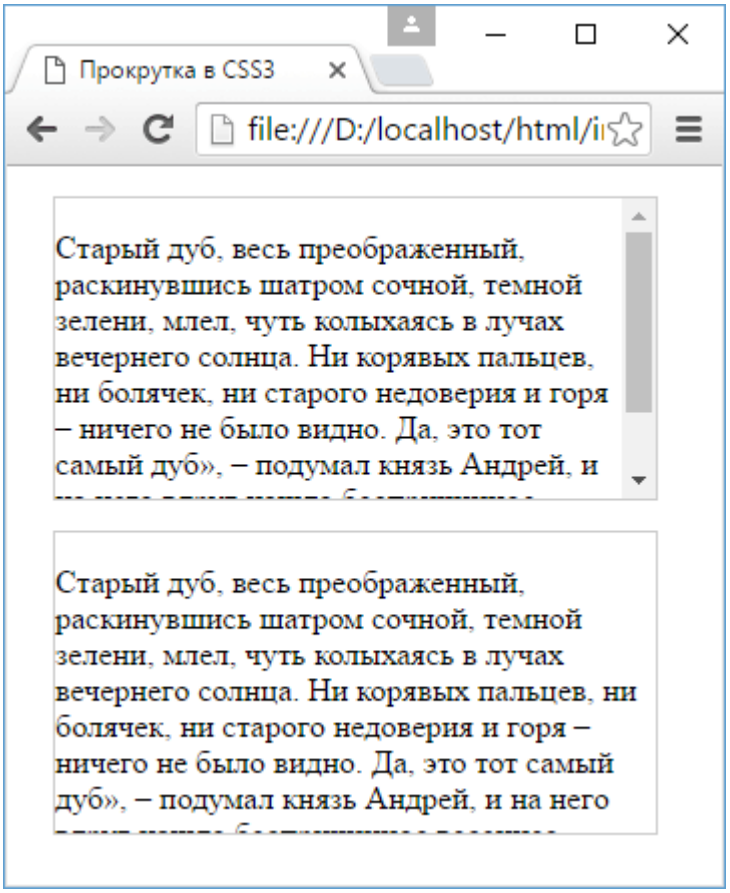
Однако свойство **overflow** позволяет настроить поведение блока в подобной ситуации и добавить возможность прокрутки. Это свойство может принимать следующие значения:

- `auto`: если контент выходит за границы блока, то создается прокрутка. В остальных случаях полосы прокрутки не отображаются
- `hidden`: отображается только видимая часть контента. Контент, который выходит за границы блока, не отображается, а полосы прокрутки не создаются
- `scroll`: в блоке отображаются полосы прокрутки, даже если контент весь помещается в границах блока, и таких полос прокрутки не требуется
- `visible`: значение по умолчанию, контент отображается, даже если он выходит за границы блока

Рассмотрим применение двух значений:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Прокрутка в CSS3</title>
        <style>
        .article1{
            width: 300px;
            height: 150px;
            margin:15px;
            border: 1px solid #ccc;
            overflow: auto;
        }
        .article2{
            width: 300px;
            height: 150px;
            margin:15px;
            border: 1px solid #ccc;
            overflow: hidden;
        }
        </style>
    </head>
    <body>
        <div class="article1">
        <p>Старый дуб, весь преображенный, раскинувшись шатром сочной, темной
        зелени, млел, чуть колыхаясь в лучах вечернего солнца. Ни корявых пальцев,
        ни болячек, ни старого недоверия и горя – ничего не было видно. Да, это тот
        самый дуб», – подумал князь Андрей, и на него вдруг нашло беспричинное
        весеннее чувство радости и обновления.</p>
        </div>
        <div class="article2">
        <p>Старый дуб, весь преображенный, раскинувшись шатром сочной, темной
```

```
        зелени, млел, чуть колыхаясь в лучах вечернего солнца. Ни корявых пальцев,
        ни болячек, ни старого недоверия и горя – ничего не было видно. Да, это тот
        самый дуб», – подумал князь Андрей, и на него вдруг нашло беспричинное
        весеннее чувство радости и обновления.</p>
        </div>
    </body>
</html>
```



Свойство `overflow` управляет полосами прокрутки как по вертикали, так и по горизонтали. С помощью дополнительных свойств **overflow-x** и **overflow-y** можно определить прокрутку соответственно по горизонтали и по вертикали. Данные свойства принимают те же значения, что и `overflow`:

```
overflow-x: auto;
overflow-y: hidden;
```

Линейный градиент

Градиенты представляют плавный переход от одного цвета к другому. В CSS3 имеется ряд встроенных градиентов, которые можно использовать для создания фона элемента.

Градиенты в CSS не представляют какого-то специального свойства. Они лишь создают значение, которое присваивается свойству **background-image**.

Линейный градиент распространяется по прямой от одного конца элемента к другому, осуществляя плавный переход от одного цвета к другому.

Для создания градиента нужно указать его начало и несколько цветов, например:

```
background-image: linear-gradient(left,black,white);
```

В данном случае началом градиента будет левый край элемента, который обозначается значением `left`. Цвета градиента: черный (black) и белый (white). То есть начиная с левого края элемента до правого будет плавно идти переход из черного цвета в белый.

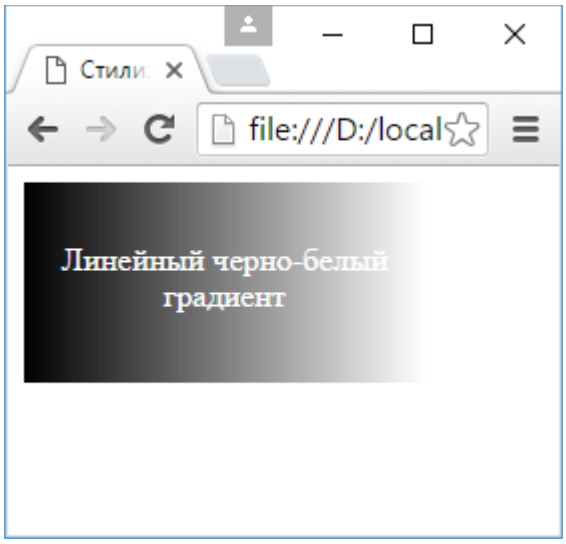
В использовании градиентов есть один недостаток - многообразие браузеров вынуждает использовать префикс вендора:

```
-webkit- /* Для Google Chrome, Safari, Microsoft Edge, Opera выше 15 версии  */
-moz- /* Для Mozilla Firefox  */
-o- /* Для Opera старше 15 версии (Opera 12)  */
```

Так, полноценное использование градиента будет выглядеть следующим образом:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Стилизация таблиц в CSS3</title>
        <style>
        div {
            width: 200px;
            height: 100px;
            background-color: #eee;
            background-image: linear-gradient(left,black,white);
            background-image: -o-linear-gradient(left,black,white);
            background-image: -moz-linear-gradient(left,black,white);
            background-image: -webkit-linear-gradient(left,black,white);
        }
        p{
            margin: 0;
            padding-top: 30px;
            text-align: center;
            color:white;
        }
        </style>
    </head>
    <body>
        <div><p>Линейный черно-белый градиент</p></div>
    </body>
```

```
27  </html>
```

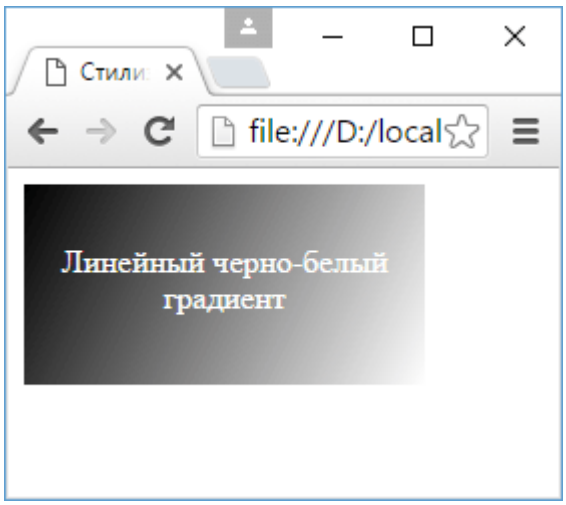


Для установки начала градиента можно использовать следующие значения: `left` (слева направо), `right` (справа налево), `top` (сверху вниз) или `bottom` (снизу вверх). Например, вертикальный градиент будет выглядеть следующим образом:

```
1  background-image: linear-gradient(bottom,black,white);
```

Также можно задать диагональное направление с помощью двух значений:

```
1  background-image: linear-gradient(top left,black,white);
```



Кроме конкретных значений типа top или left также можно указать угол от 0 до 360, который определит направление градиента:

```
1  background-image: linear-gradient(30deg,black,white);
```

После величины углы указывается слово **deg**.

К примеру, 0deg означает, что градиент начинается в левой части и перемещается в правую часть, а при указании 45deg он начинается в нижнем левом углу и перемещается под углом 45° в верхний правый угол.

После определения начала градиента, можно указать применяемых цветов или опрные точки. Цветов не обязательно должно быть два, их может быть и больше:

```
background-image: linear-gradient(top, red, #ccc, blue);
```

Все применяемые цвета распределяются равномерно. Однако можно также указать конкретные места фона для цветовых точек. Для этого после цвета добавляется второе значение, которое и определяет положение точки.

```
background-image: linear-gradient(left, #ccc, red 20%, red 80%, #ccc);
```

Повторяющийся градиент

С помощью `repeating-linear-gradient` можно создавать повторяющиеся линейные градиенты. Например:

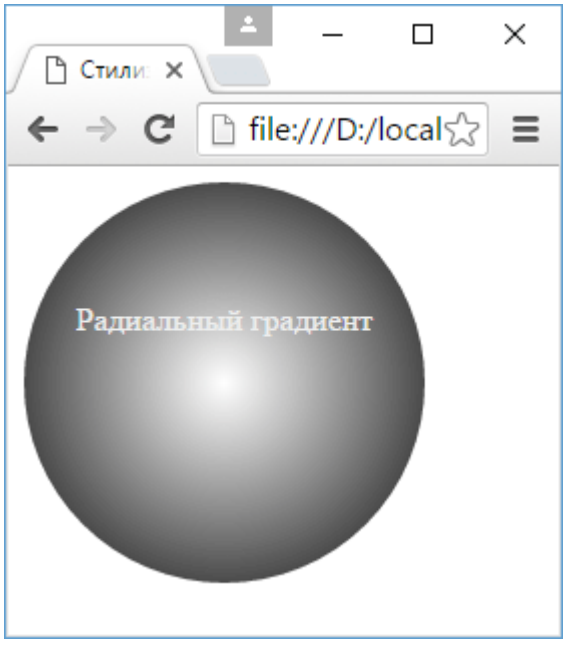
```
background-image: repeating-linear-gradient(left, #ccc 20px, red 30px, rgba(0, 0, 126, .5) 40px);
background-image: -moz-repeating-linear-gradient(left, #ccc 20px, red 30px, rgba(0, 0, 126, .5) 40px);
background-image: -webkit-repeating-linear-gradient(left, #ccc 20px, red 30px, rgba(0, 0, 126, .5) 40px);
```

В данном случае градиент начинается с левого края элемента с полоски серого цвета (#ccc) шириной 20 пикселей, далее до 30 пикселей идет переход к красному цвету, а затем до 40 пикселей выполняется обратный переход к светло-синему цвету (rgba(0, 0, 126, .5)). После этого браузер повторяет градиент, пока не заполнит всю поверхность элемента.

Радиальный градиент

Радиальные градиенты в отличие от линейных распространяются от центра наружу по круговой схеме. Для создания радиального градиента достаточно указать цвет, который будет в центре градиента, и цвет, который должен быть снаружи. Эти цвета передаются в функцию **radial-gradient()**. Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Стилизация таблиц в CSS3</title>
        <style>
        div {
            width: 200px;
            height: 200px;
            border-radius: 100px;

            background-color: #eee;
            background-image: radial-gradient(white, black);
            background-image: -moz-radial-gradient(white, black);
            background-image: -webkit-radial-gradient(white, black);
        }
        p{
            margin: 0;
            padding-top: 60px;
            text-align: center;
            color: #eee;
        }
        </style>
    </head>
    <body>
        <div><p>Радиальный градиент</p></div>
    </body>
</html>
```



Как и в случае с линейным градиентом здесь также надо использовать префиксы вендоров для поддержки браузерами.

Радиальный градиент может иметь две формы: круговую и эллиптическую. Эллиптическая форма представляет распространение градиента в виде эллипса и задается с помощью ключевого слова `ellipse`:

```
background-image: radial-gradient(ellipse, white, black);
```

Поскольку это значение для градиента по умолчанию, то оно может опускаться при использовании.

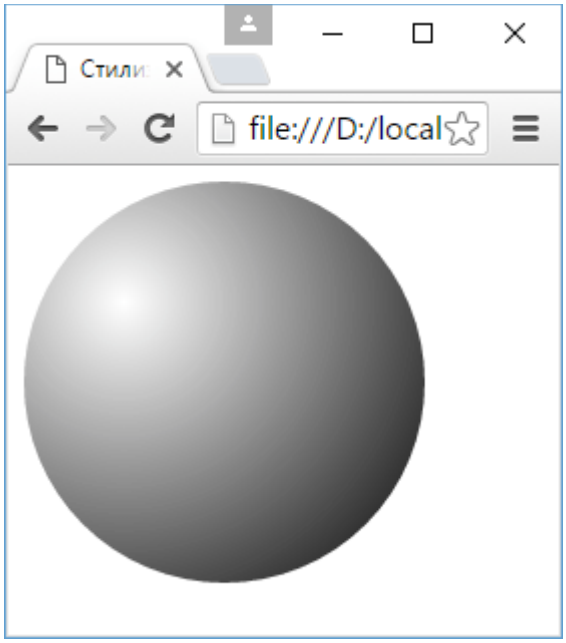
Круговая форма представляет распространение градиента в виде кругов от центра во вне. Для этого используется ключевое слово `circle`:

```
background-image: radial-gradient(circle, white, black);
```

Как правило, центр радиального градиента расположен в центре элемента, но это поведение можно переопределить, указав значение для параметра `background-position`:

```
background-image: radial-gradient(25% 30%, circle, white, black);
```

Числа `25% 30%` означают, что центр градиента будет находиться на расстоянии в 25% от левой границы и в 30% от верхней границы элемента.



С помощью специальных дополнительных значений можно задать размер градиента:

- `closest-side`: градиент распространяется из центра только до ближайшей к центру стороне элемента. То есть градиент остается внутри элемента
- `closest-corner`: ширина градиента вычисляется по расстоянию из его центра до ближайшего угла элемента, поэтому градиент может выйти за пределы элемента.
- `farthest-side`: градиент распространяется из центра до самой дальней стороны элемента
- `farthest-corner`: ширина градиента вычисляется по расстоянию из его центра до самого дальнего угла элемента

```
background-image: radial-gradient(25% 30%, circle farthest-corner, white, black);
```

Стилизация элемента details

Рассмотрим некоторые возможности по стилизации элемента **details**, который представляет раскрываемый блок.

Атрибут open

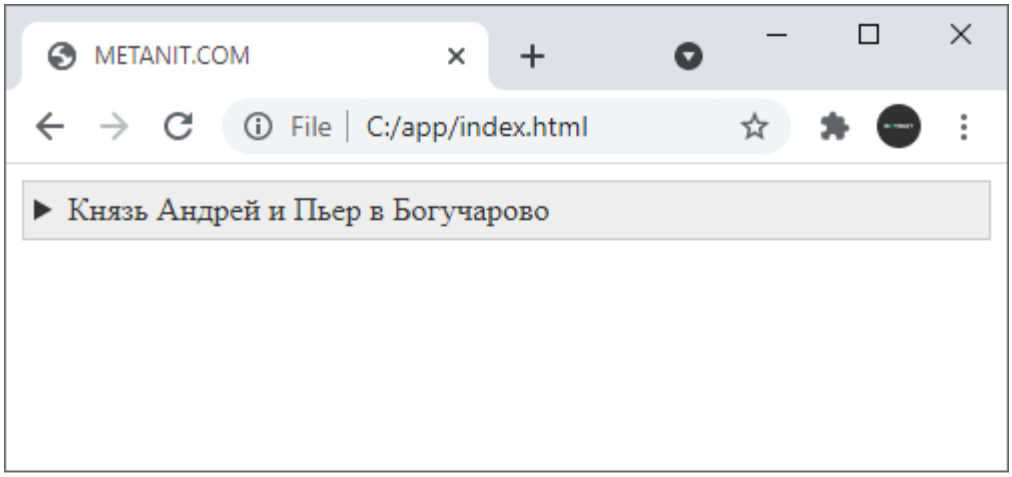
Прежде всего в раскрытом состоянии к элементу **details** добавляется атрибут **open**. Соответственно, используя атрибут, можно задать разные стили для элемента в скрытом и раскрытом состоянии. Например:

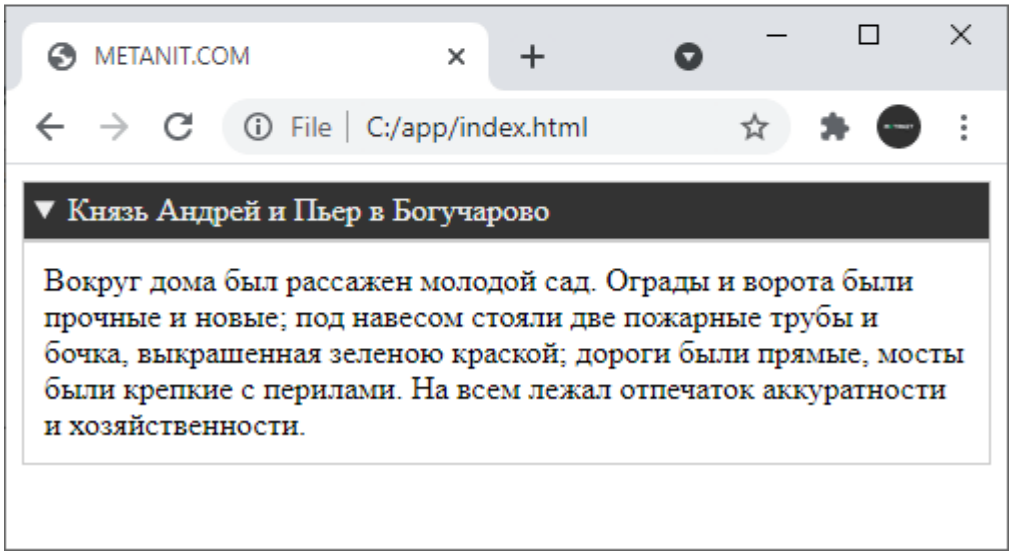
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>METANIT.COM</title>
    <style>
    details > summary {
        padding: 5px;
        background-color: #eee;
        color: #333;
        border: 1px #ccc solid;
        cursor: pointer;
    }

    details > div {
        border: 1px #ccc solid;
        padding: 10px;
    }
    details[open] > summary {
        color:#eee;
        background-color:#333;
    }
</style>
</head>
<body>
<details>
    <summary>Князь Андрей и Пьер в Богучарово</summary>
    <div>Вокруг дома был рассажен молодой сад. Ограды и ворота были прочные и новые; под навесом
    стояли две пожарные трубы и бочка, выкрашенная зеленою краской; дороги были прямые, мосты были крепкие с перилами.
    На всем лежал отпечаток аккуратности и хозяйственности.</div>
</details>
</body>
</html>
```

С помощью селектора `details[open]` мы можем обратиться к элементу `details` в раскрытом состоянии. Соответственно селектор

```
details[open] > summary {
    color:#eee;
    background-color:#333;
}
```

Позволяет задать стили для элемента **summary** в раскрытом состоянии. То есть в данном случае при раскрытии элемента details мы переключаем цвет стиля и фона заголовка.





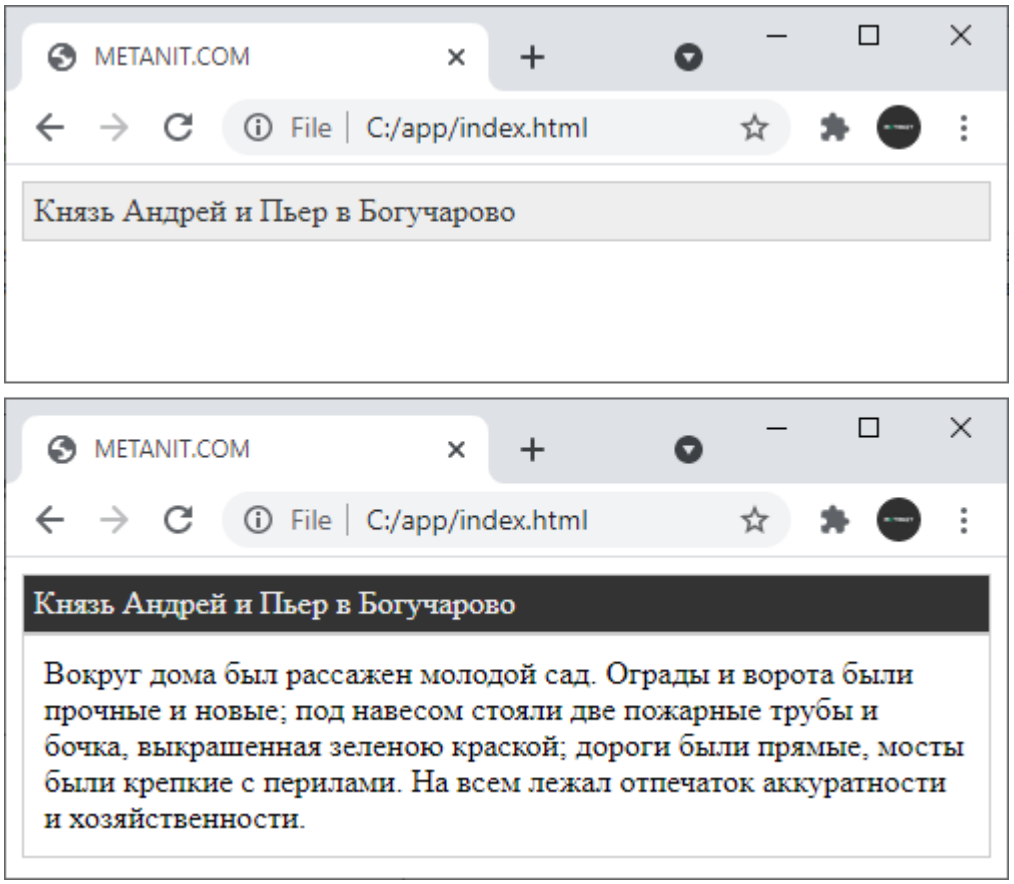
Настройка маркера

По умолчанию элемент **summary** в качестве маркера скрытости/раскрытости использует символ треугольника. Но его также можно настроить.

Для настройки изображения маркера можно использовать свойство **list-style**, а также дополнительные свойства типа **list-style-type** или **list-style-image**, которые применяются для стилизации списков. Например, если необходимо убрать этот маркер, то можно применить стиль `list-style: none`:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>METANIT.COM</title>
    <style>

    details > summary {
        padding: 5px;
        background-color: #eee;
        color: #333;
        border: 1px #ccc solid;
        cursor: pointer;
```

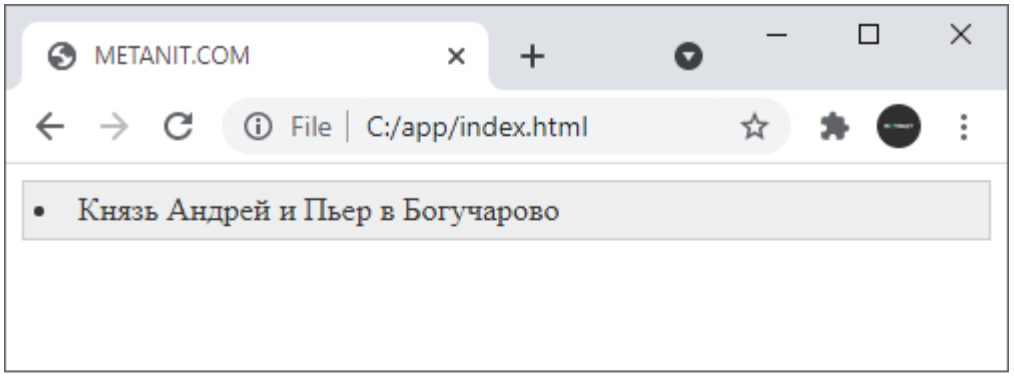
```
            list-style: none;
        }

        details > div {
            border: 1px #ccc solid;
            padding: 10px;
        }

    details[open] > summary {
        color:#eee;
       background-color:#333;
    }
    </style>
    </head>
    <body>
    <details>
        <summary>Князь Андрей и Пьер в Богучарово</summary>
        <div>Вокруг дома был рассажен молодой сад. Ограды и ворота были прочные
        и новые; под навесом стояли две пожарные трубы и бочка, выкрашенная
        зеленою краской; дороги были прямые, мосты были крепкие с перилами.
        На всем лежал отпечаток аккуратности и хозяйственности.</div>
    </details>
    </body>
    </html>
```

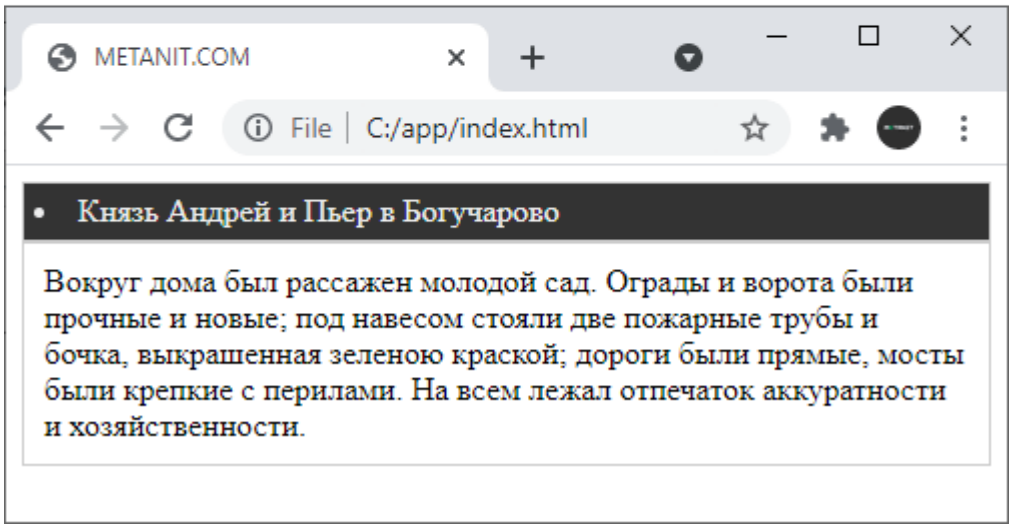


Другой пример - примение стиля `list-style-type: disc;`:

```
<!DOCTYPE html>
<html>
<head>
```

```
    <meta charset="utf-8" />
    <title>METANIT.COM</title>
    <style>

    details > summary {
        padding: 5px;
        background-color: #eee;
        color: #333;
        border: 1px #ccc solid;
        cursor: pointer;

        list-style-type: disc;  /* задаем маркер*/
    }

    details > div {
        border: 1px #ccc solid;
        padding: 10px;
    }

details[open] > summary {
    color:#eee;
   background-color:#333;
}
</style>
</head>
<body>
<details>
    <summary>Князь Андрей и Пьер в Богучарово</summary>
    <div>Вокруг дома был рассажен молодой сад. Ограды и ворота были прочные
    и новые; под навесом стояли две пожарные трубы и бочка,
    выкрашенная зеленою краской; дороги были прямые, мосты были крепкие с
    перилами. На всем лежал отпечаток аккуратности и хозяйственности.</div>
</details>
</body>
</html>
```





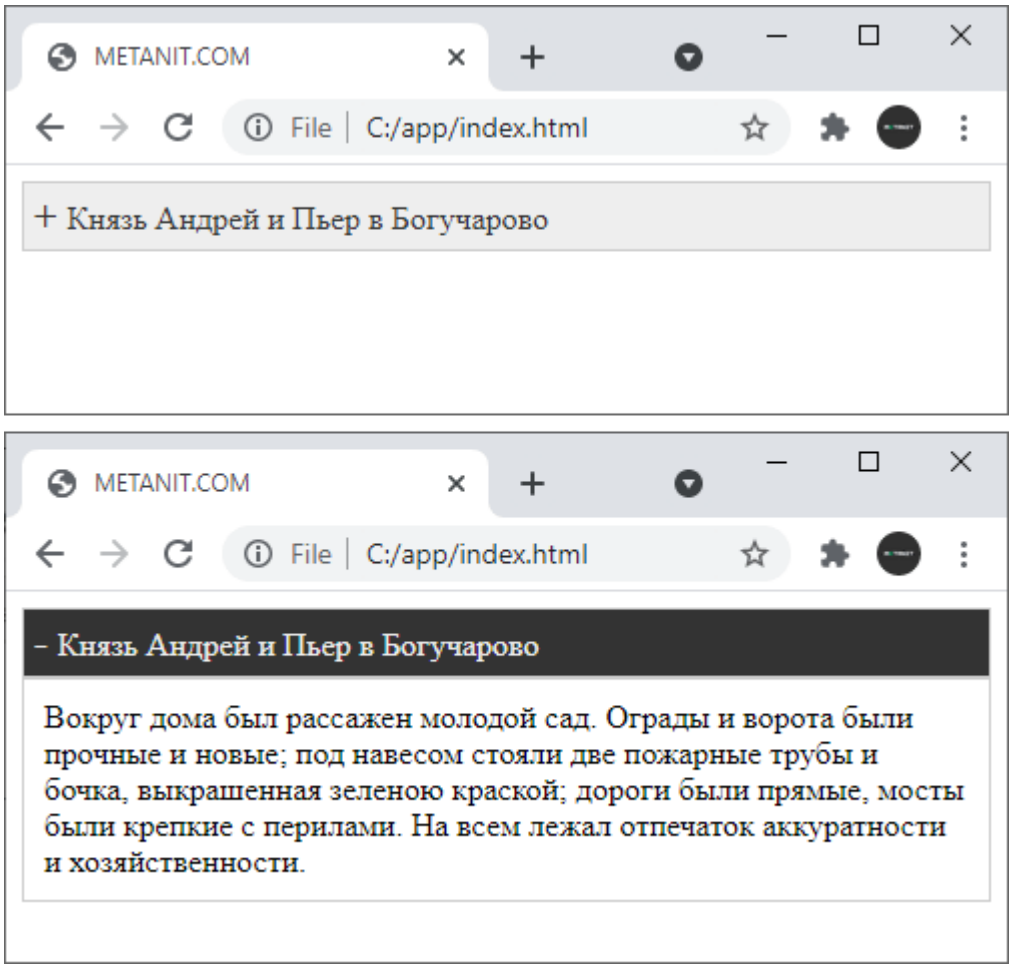
Настройка с помощью свойства content

Но естественно свойством `list-style` мы не ограничены и по своему усмотрению можем более тонко управлять заголовком, например, с помощью свойства **content**:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>METANIT.COM</title>
    <style>

    details > summary {
        padding: 5px;
        background-color: #eee;
        color: #333;
        border: 1px #ccc solid;
        cursor: pointer;
        list-style: none;
    }

    details > div {
        border: 1px #ccc solid;
        padding: 10px;
    }

details[open] > summary {
    color:#eee;
   background-color:#333;
}
```

```
summary:before {
   content: "+";
   font-size: 20px;
   font-weight: bold;
   margin: -5px 5px 0 0;
}

details[open] summary:before {
   content: "-";
}
</style>
</head>
<body>
<details>
    <summary>Князь Андрей и Пьер в Богучарово</summary>
    <div>Вокруг дома был рассажен молодой сад. Ограды и ворота были прочные
    и новые; под навесом стояли две пожарные трубы и бочка,
    выкрашенная зеленою краской; дороги были прямые, мосты были крепкие с
    перилами. На всем лежал отпечаток аккуратности и хозяйственности.</div>
</details>
</body>
</html>
```

В данном случае с помощью селектора `summary:before` устанавливаем содержимое, которое будет располагаться перед основным содержимым элемента `summary`. Селектор `details[open] summary:before` позволяет сделать то же самое, только в раскрытом виде. В итоге в скрытом виде маркер будет отображать символ +, а в раскрытом - символ -.



Создание макета страницы и верстка

Блочная верстка. Часть 1

Как правило, веб-страница состоит из множества различных элементов, которые могут иметь сложную структуру. Поэтому при создании веб-страницы возникает необходимость нужным образом позиционировать эти элементы, стилизовать их так, чтобы они располагались на странице нужным образом. То есть возникает вопрос создания макета страницы, ее верстки.

Существуют различные способы, стратегии и виды верстки. Изначально распространенной была верстка на основе таблиц. Так как таблицы позволяет при необходимости очень легко и просто разделить все пространство веб-страницы на строки и столбцы. Строками и столбцами довольно легко управлять, в них легко позиционировать любое содержимое. Именно это и определило популярность табличной верстки.

Однако табличная верстка создает не самые гибкие по дизайну страницы, что является особенно актуальным аспектом в мире, где нет одного единственного разрешения экрана, за то есть большие экраны на телевизорах, малые экраны на планшетах и фаблетах, очень маленькие экраны на смартфонах и т.д. Все это многообразие экранов табличная верстка оказалась не в состоянии удовлетворить. Поэтому постепенно ей на смену пришла блочная верстка. Блочная верстка - это отосительно условное название способов и приемов верстки, когда в большинстве веб-страниц для разметки используется CSS-свойство **float**, а основным строительным элементов веб-страниц является элемент ** **, то есть по сути блок. Используя свойство float и элементы div или другие элементы, можно создать структуру страницы из нескольких столбцов, как при табличной верстке, которая будет значительно гибче.

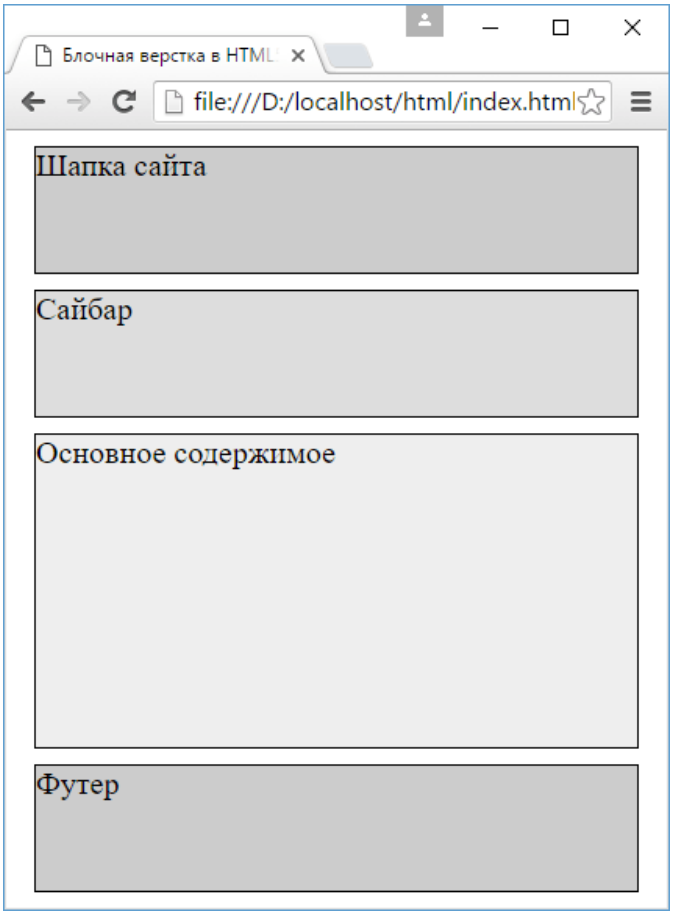
Ранее в одной из прошлых тем рассматривалось действие свойства `float`. Теперь используем его для создания двухколоночной веб-страницы. Допустим, вверху и внизу у нас будут стандартно шапка и футер, а в центре - две колонки: колонка с меню или сайдбар и колонка с основным содержимым.

В начале определим все блоки. При работе с элементами, которые используют обтекание и свойство `float`, важен их порядок. Так, код плавающего элемента, у которого устанавливается свойство `float`, должен идти перед элементом, который обтекает плавающий элемент. То есть блок сайдбара будет идти до блока основного содержимого:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Блочная верстка в HTML5</title>
        <style>
            div{
                margin: 10px;
                border: 1px solid black;
                font-size: 20px;
                height: 80px;
            }
            #header{
                background-color: #ccc;
            }
            #sidebar{
```

```
                background-color: #ddd;
            }
            #main{
                background-color: #eee;
                height: 200px;
            }
            #footer{
                background-color: #ccc;
            }
        </style>
    </head>
    <body>
        <div id="header">Шапка сайта</div>
        <div id="sidebar">Сайбар</div>
        <div id="main">Основное содержимое</div>
        <div id="footer">Футер</div>
    </body>
</html>
```

То есть пока получается примерно следующая страница:



Высота, граница и отступы блоков в данном случае добавлены только для красоты, чтобы идентифицировать пространство блока и отделять его от других.

Далее, чтобы переместить блок сайдбара влево по отношению к блоку основного содержимого и получить эффект обтекания, нам надо указать у блока сайдбара свойство `float: left` и предпочтительную ширину. Ширина может быть фиксированной, например, 150 px или 8 em. Либо также можно использовать проценты, например, 30% - 30% от ширины контейнера body. С одной стороны, блоками с фиксированной шириной легче управлять, но с другой процентные

значения ширины позволяют создавать более гибкие, резиновые блоки, которые изменяют размеры при изменении размеров окна браузера.

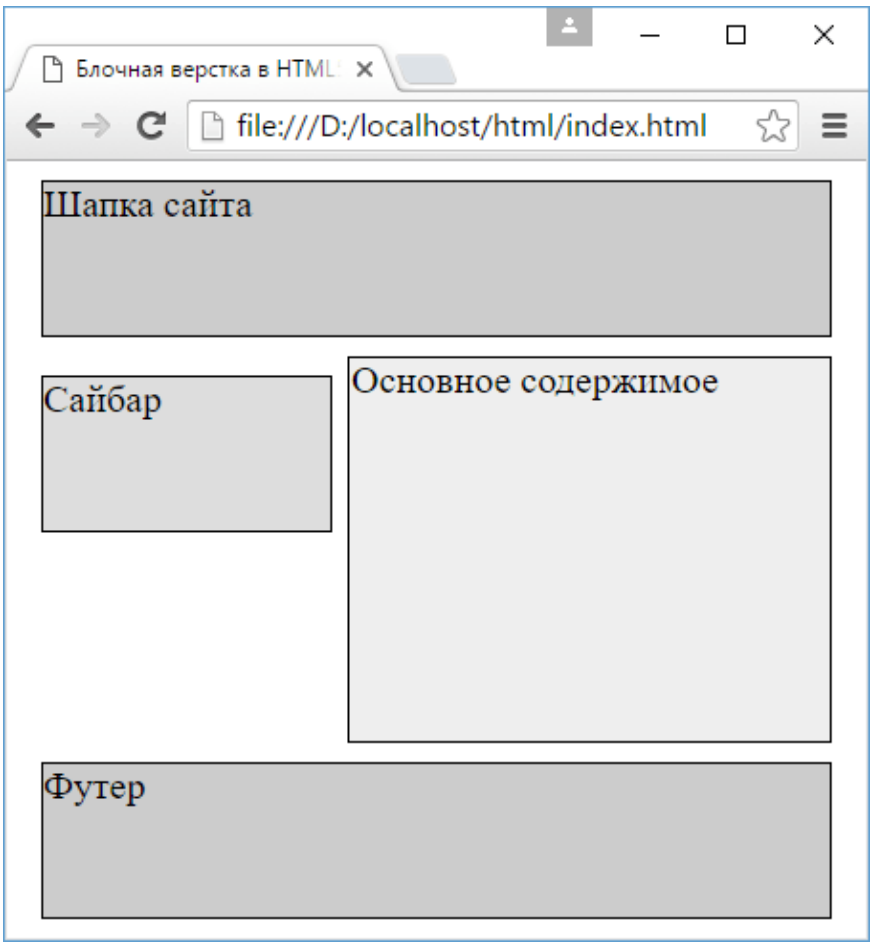
Последним шагом является установка отступа блока с основным содержимым от блока сайдбара. Поскольку при обтекании обтекающий блок может обтекать плавающий элемент и справа и снизу, если плавающий элемент имеет меньшую высоту, то нам надо установить отступ, как минимум равный ширине плавающего элемента. Например, если ширина сайдбара равна 150px, то для блока основного содержимого можно задать отступ в 170px, что позволит создать пустое пространство между двумя блоками.

При этом не стоит у блока основного содержимого указывать явным образом ширину, так как браузеры расширяют его автоматически, чтобы он занимал все доступное место.

Итак, принимая во внимание все выше сказанное, изменим стили блоков сайдбара и основного содержимого следующим образом:

```
#sidebar{
    background-color: #ddd;
    float: left;
    width: 150px;
}
#main{
    background-color: #eee;
    height: 200px;
    margin-left: 170px; /* 150px (ширина сайдбара) + 10px + 10px (2 отступа) */
}
```

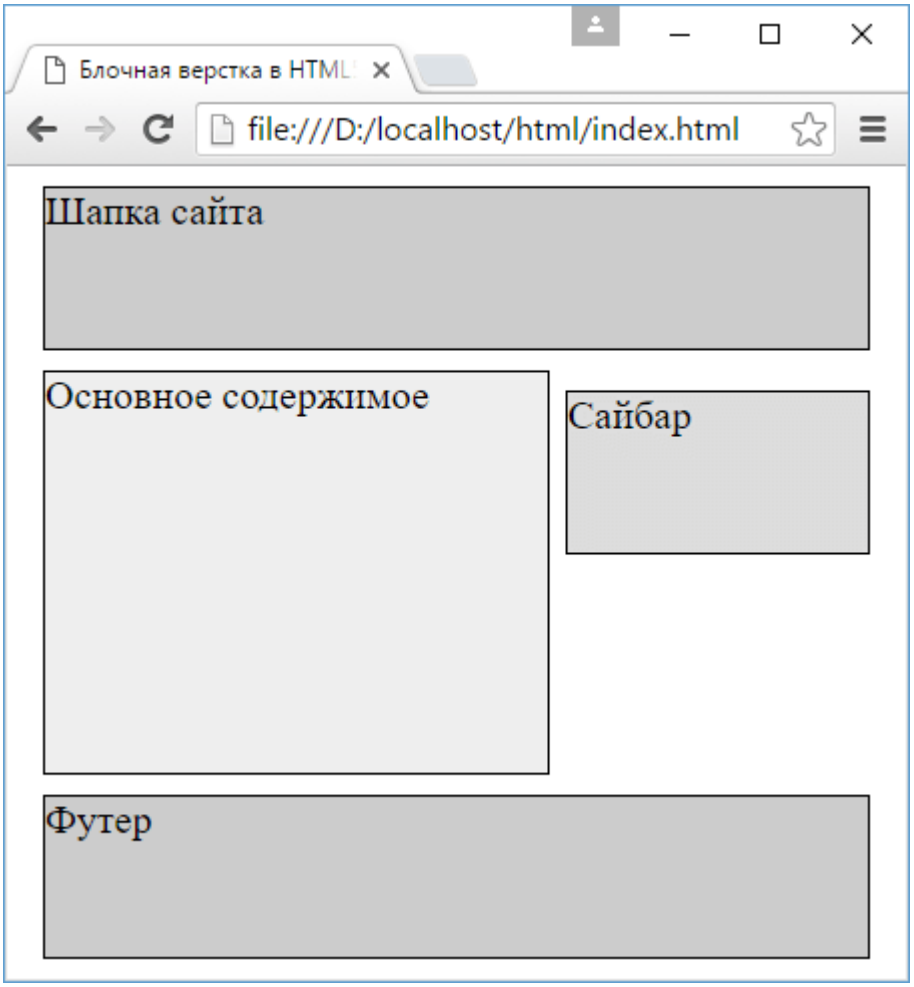
В итоге у нас получится сайдбар по левую сторону от основного блока:



Высота блоков в данном случае указана условно для большей наглядности, в реальности, как правило, высоту будет автоматически устанавливать браузер.

Создание правого сайдбара будет аналогично, только теперь нам надо установить у сайдбара значение `float: right`, а у блока основного содержимого - отступ справа:

```
#sidebar{
    background-color: #ddd;
    float: right;
    width: 150px;
}
#main{
    background-color: #eee;
    height: 200px;
    margin-right: 170px;
}
```

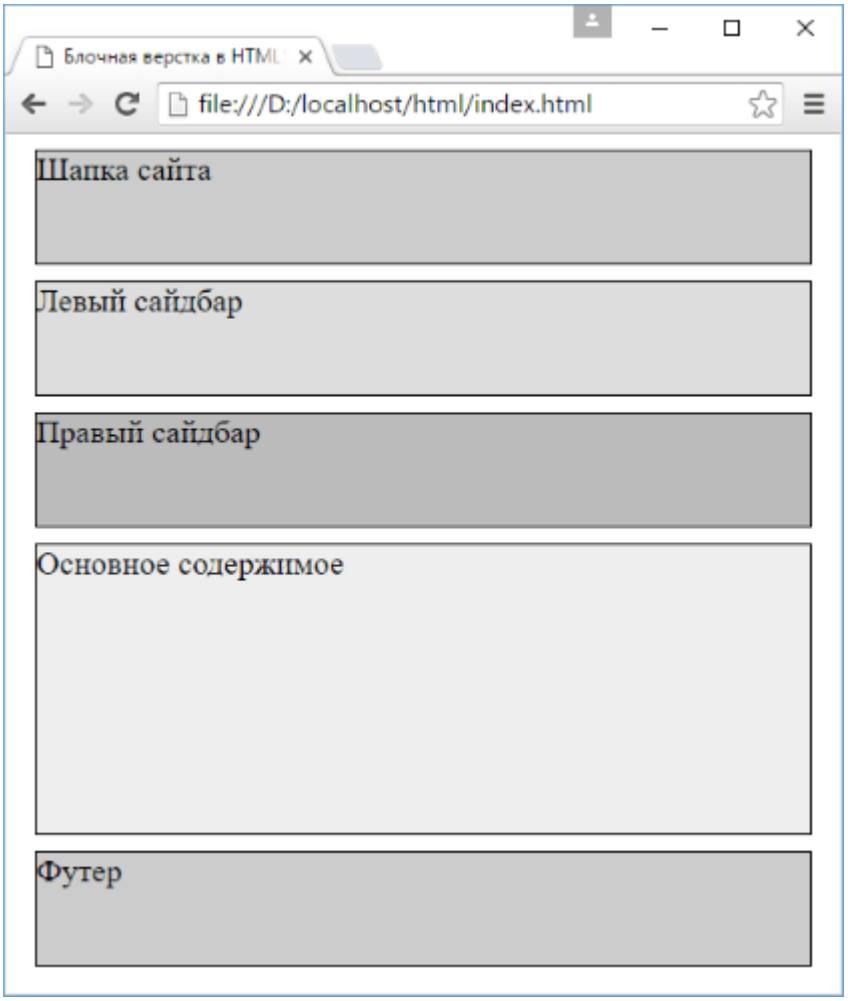


При этом разметка html остается такой же, блок сайдбара по прежнему должен предшествовать блоку основного содержимого.

Блочная верстка. Часть 2

В прошлой теме было рассмотрено создание страницы с двумя колонками. Подобным образом мы можем добавить на страницу и большее количество колонок и сделать более сложную структуру. Например, добавим на веб-страницу второй сайдбар:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Блочная верстка в HTML5</title>
        <style>
            div{
                margin: 10px;
                border: 1px solid black;
                font-size: 20px;
                height: 80px;
            }
            #header{
                background-color: #ccc;
            }
            #leftSidebar{
                background-color: #ddd;
            }
            #rightSidebar{
                background-color: #bbb;
            }
            #main{
                background-color: #eee;
                height: 200px;
            }
            #footer{
                background-color: #ccc;
            }
        </style>
    </head>
    <body>
        <div id="header">Шапка сайта</div>
        <div id="leftSidebar">Левый сайдбар</div>
        <div id="rightSidebar">Правый сайдбар</div>
        <div id="main">Основное содержимое</div>
        <div id="footer">Футер</div>
    </body>
</html>
```

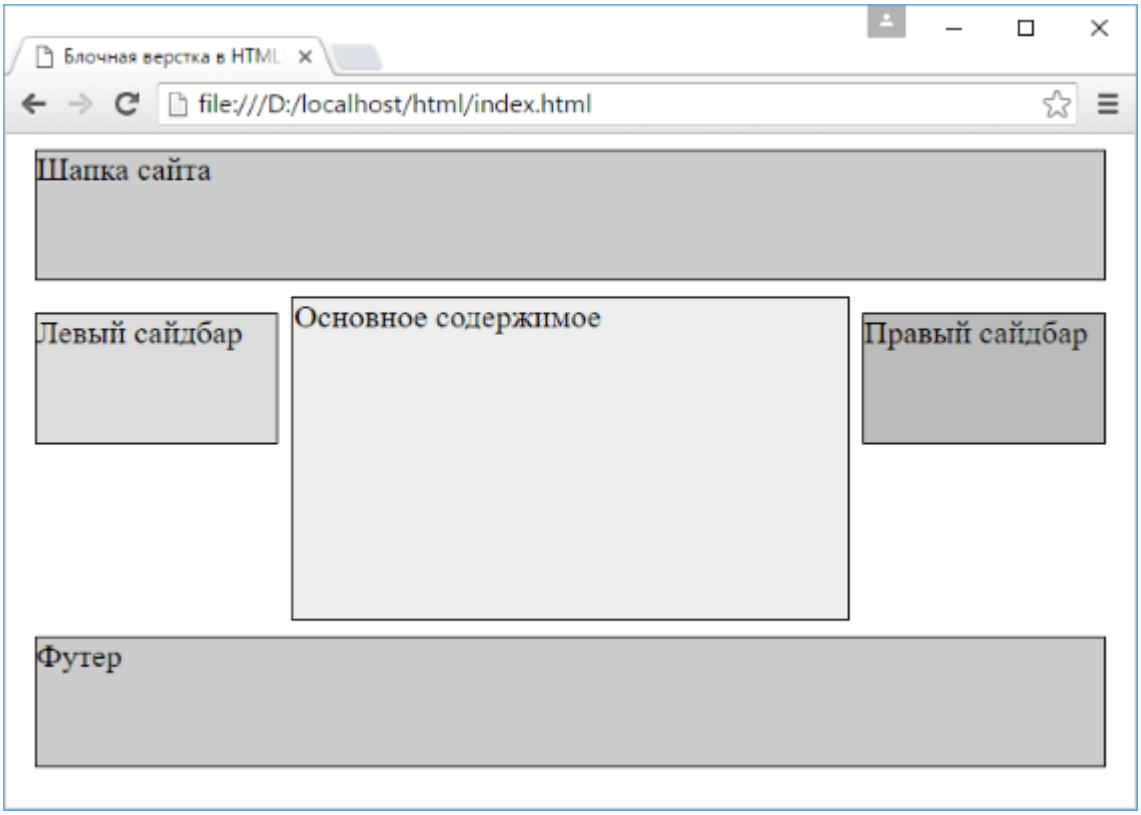


Здесь опять же код обоих сайдбаров должен идти до блока с основным содержимым, который обтекает их.

Теперь изменим стили обоих сайдбаров и основного блока:

```
#leftSidebar{
    background-color: #ddd;
    float: left;
    width: 150px;
}
#rightSidebar{
    background-color: #bbb;
    float: right;
    width: 150px;
}
#main{
    background-color: #eee;
    height: 200px;
    margin-left: 170px;
    margin-right: 170px;
}
```

Опять же у обоих плавающих блоков - сайдбаров нам надо установить ширину и свойство `float` - у одного значение `left`, а у другого `right`.


Блочная верстка в HTML
file:///D:/localhost/html/index.html
Шапка сайта
Левый сайдбар
Основное содержимое
Правый сайдбар
Футер

Вложенные плавающие блоки

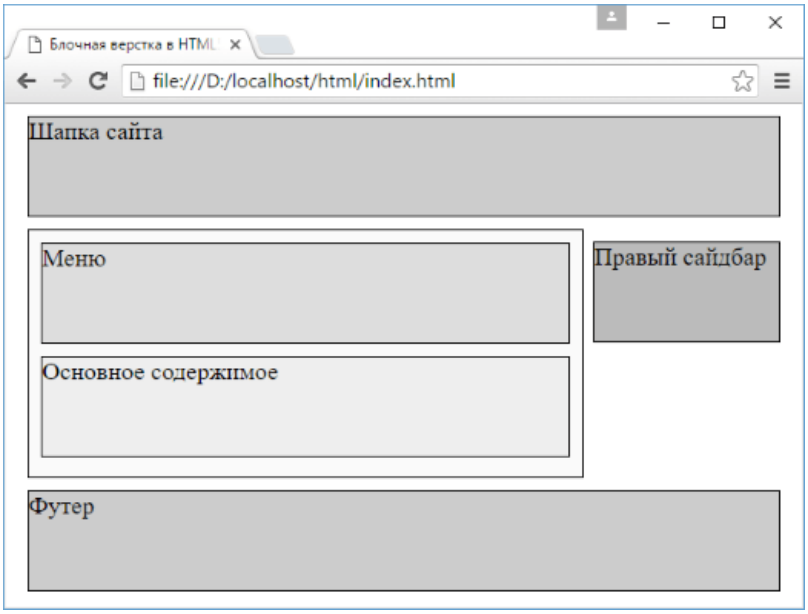
Нередко встречается ситуация, когда к вложенным в обтекающий блок элементам также применяется обтекание. Например, блок основного содержимого может включать блок собственно содержимого и блок меню. В принципе к таким блокам будут применяться все те же правила, что были рассмотрены ранее.

Определим сначала последовательно все блоки веб-страницы:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Блочная верстка в HTML5</title>
        <style>
            div{
                margin: 10px;
                border: 1px solid black;
                font-size: 20px;
                height: 80px;
            }
            #header{
                background-color: #ccc;
            }
            #sidebar{
                background-color: #bbb;
                float: right;
                width: 150px;
            }
            #main{
                background-color: #fafafa;
                height: 200px;
                margin-right: 170px;
            }
            #menu{
                background-color: #ddd;
            }
            #content{
                background-color: #eee;
            }
            #footer{
                background-color: #ccc;
            }
        </style>
    </head>
    <body>
        <div id="header">Шапка сайта</div>
        <div id="sidebar">Правый сайдбар</div>
        <div id="main">
            <div id="menu">Меню</div>
            <div id="content">Основное содержимое</div>
        </div>
        <div id="footer">Футер</div>
    </body>
```

```
</html>
```

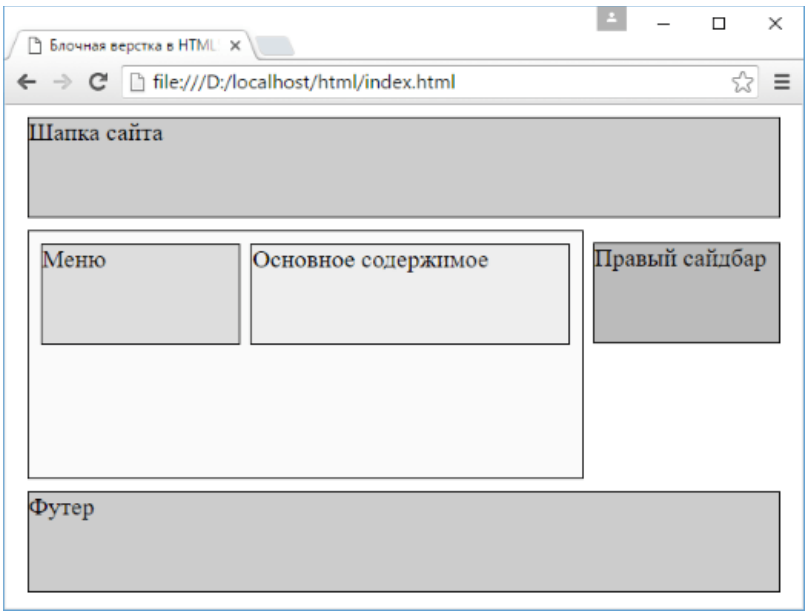
Опять же в главном блоке вложенные блоки идут последовательно: сначала блок меню, а потом блок основного текста.



Теперь применим обтекание к блоку меню:

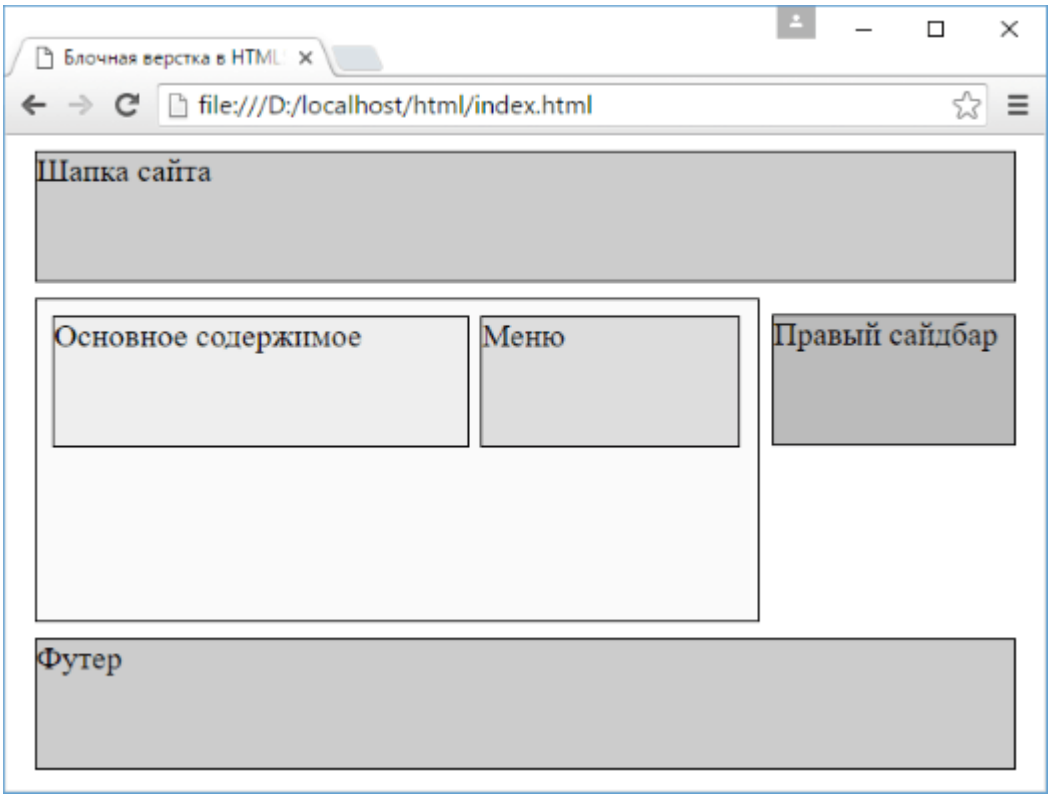
```
#menu{
    background-color: #ddd;
    float: left;
    width: 160px;
}
#content{
    background-color: #eee;
    margin-left: 180px;
}
```

Опять же у плавающего элемента, коим является блок меню, устанавливаются свойства `float` и `width`. А у обтекающего его блока content устанавливается отступ слева.



Аналогично можно сделать блок меню справа:

```
#menu{
    background-color: #ddd;
    float: right;
    width: 160px;
}
#content{
    background-color: #eee;
    margin-right: 180px;
}
```



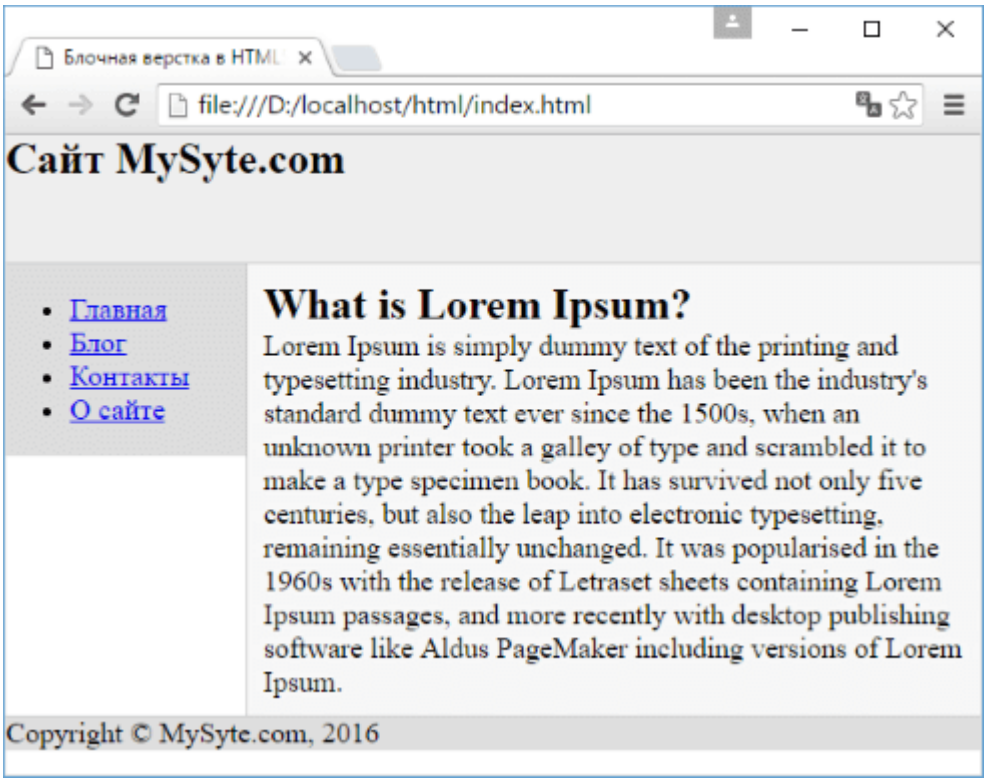
Выравнивание столбцов по высоте

При блочной верстке мы можем столкнуться с проблемой высоты с столбцов, которая может особенно сильно проявиться, если плавающие блоки имеют определенный фон. Рассмотрим проблему на примере:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Блочная верстка в HTML5</title>
        <style>
            body, h2, p{
                margin:0;
                padding:0;
            }
            body{
                font-size: 18px;
            }
            #header{
                background-color: #eee;
                border-bottom: 1px solid #ccc;
                height: 80px;
            }
            #menu{
                background-color: #ddd;
                float: left;
                width: 150px;
            }
            #main{
                background-color: #f7f7f7;
                border-left: 1px solid #ccc;
                margin-left: 150px;
                padding: 10px;
            }
            #footer{
                border-top: 1px solid #ccc;
                background-color: #dedede;
            }
        </style>
    </head>
    <body>
        <div id="header"><h2>Сайт MySyte.com</h2></div>
        <div id="menu">
            <ul>
                <li><a href="#">Главная</a></li>
                <li><a href="#">Блог</a></li>
                <li><a href="#">Контакты</a></li>
                <li><a href="#">О сайте</a></li>
            </ul>
        </div>
        <div id="main">
            <h2>What is Lorem Ipsum?</h2>
            <p>Lorem Ipsum is simply dummy text of the printing and typesetting
```

```
            industry. Lorem Ipsum has been the industry...</p>
        </div>
        <div id="footer">
            <p>Copyright © MySyte.com, 2016</p>
        </div>
    </body>
</html>
```

Здесь уже установлено обтекание, и оно прекрасно работает, вот только в зависимости от количества содержимого одна колонка может быть больше другой:



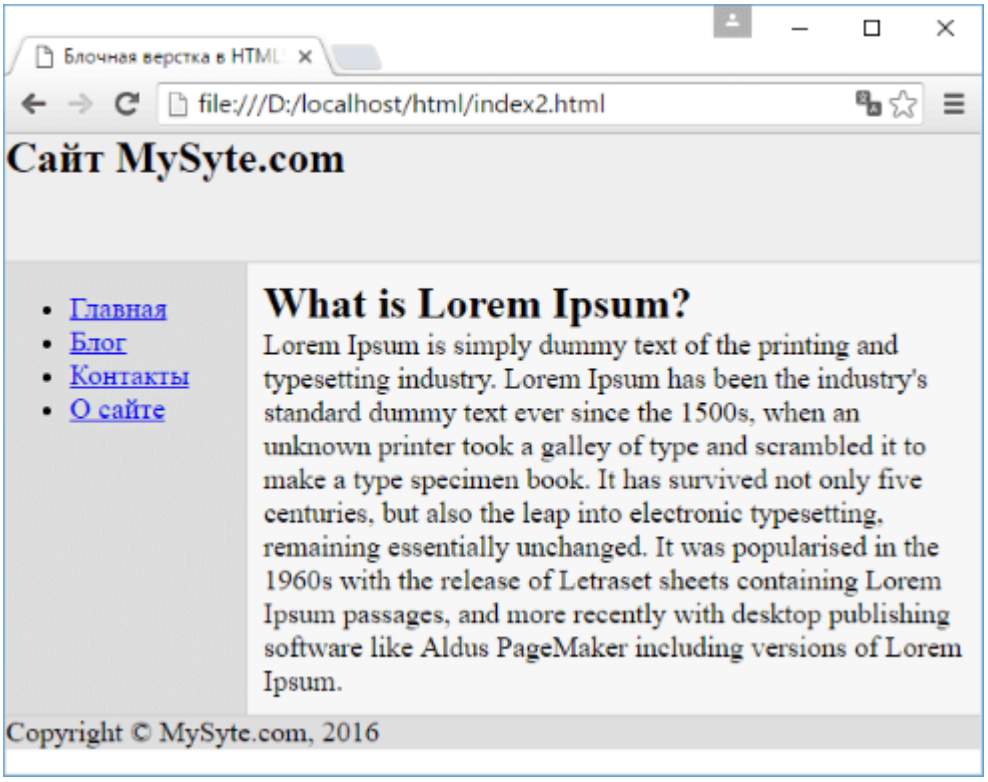
В данном случае, если в главном блоке много текста, то блок с меню имеет недостаточную длину.

Наиболее распространенным решением данной проблемы является оборачивание плавающего элемента и блока, его обтекающего, в отдельный элемент, у которого устанавливается фон. Затем этот фон используется наименьшим по длине блоком. В итоге получается иллюзия, что блоки имеют равную длину, а фон у блоков установлен корректно.

Так, изменим выше созданную страницу следующим образом:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Блочная верстка в HTML5</title>
        <style>
            body, h2, p{
                margin:0;
                padding:0;
            }
            body{
                font-size: 18px;
            }
```

```
            #header{
                background-color: #eee;
                border-bottom: 1px solid #ccc;
                height: 80px;
            }
            #wrapper{
                background-color: #ddd;
            }
            #menu{
                float: left;
                width: 150px;
            }
            #main{
                background-color: #f7f7f7;
                border-left: 1px solid #ccc;
                margin-left: 150px;
                padding: 10px;
            }
            #footer{
                border-top: 1px solid #ccc;
                background-color: #dedede;
            }
        </style>
    </head>
    <body>
        <div id="header"><h2>Сайт MySyte.com</h2></div>
        <div id="wrapper">
            <div id="menu">
                <ul>
                    <li><a href="#">Главная</a></li>
                    <li><a href="#">Блог</a></li>
                    <li><a href="#">Контакты</a></li>
                    <li><a href="#">О сайте</a></li>
                </ul>
            </div>
            <div id="main">
                <h2>What is Lorem Ipsum?</h2>
                <p>Lorem Ipsum is simply dummy text of the printing and
                typesetting industry. Lorem Ipsum has been the industry...</p>
            </div>
        </div>
        <div id="footer">
            <p>Copyright © MySyte.com, 2016</p>
        </div>
    </body>
</html>
```



В данном случае плавающий блок menu и обтекающий его блок main обертываются в один элемент wrapper, в котором устанавливается фон для элемента menu. А элемент main, как наибольший элемент, может использовать собственную установку фона.

Свойство display

Кроме свойства `float`, которое позволяет изменять позицию элемента, в CSS есть еще одно важное свойство - **display**. Оно позволяет управлять блоком элемента и также влиять на его позиционирование относительно соседних элементов.

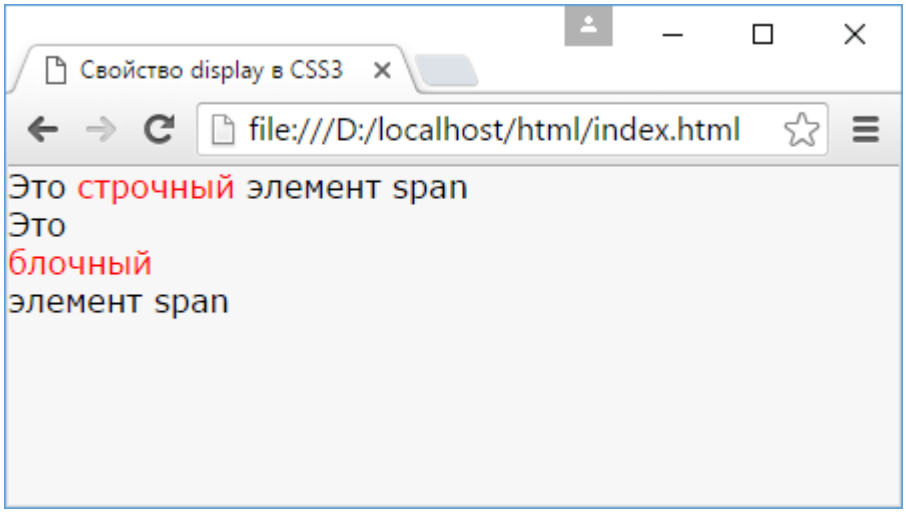
Это свойство может принимать следующие значения:

- `inline`: элемент становится строчным, подобно словам в строке текста
- `block`: элемент становится блочным, как параграф
- `inline-block`: элемент располагается как строка текста
- `list-item`: элемент позиционируется как элемент списка обычно с добавление маркера виде точки или порядкового номера
- `run-in`: тип блока элемента зависит от окружающих элементов
- `flex`: позволяет осуществлять гибкое позиционирование элментов
- `table`, `inline-table`: позволяет расположить элементы в виде таблицы
- `none`: элемент не виден и удален из разметки html

Итак, значение **block** позволяет определить блочный элемент. Такой элемент визуально отделяется от соседних элементов переносом строки, как, например, элемент параграфа p или элемент div, которые по умолчанию являются блочными и при визуализации веб-страницы визуально переносятся на новую строку.

Однако элемент `span` в отличие от элемента div блочным не является. Поэтому посмотрим, какие с ним произойдут изменения при применении значения `block`:

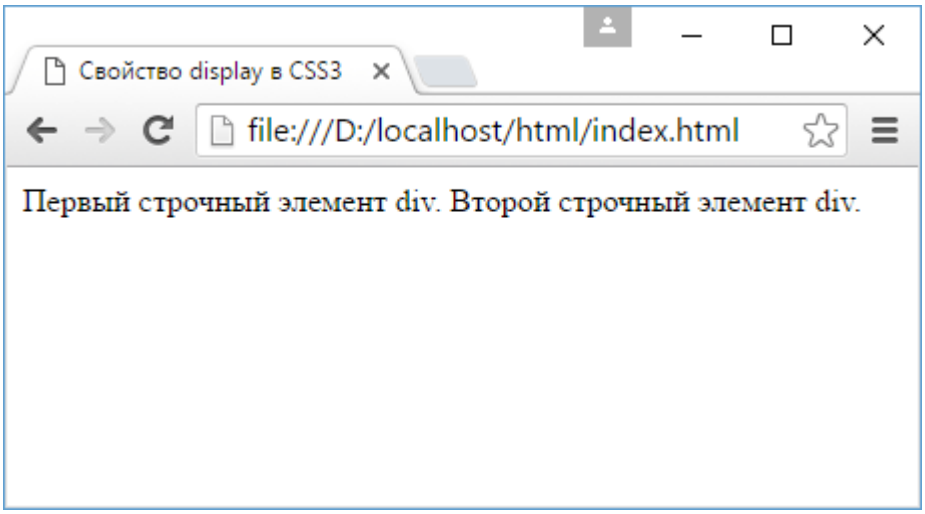
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <link href="styles.css" rel="stylesheet">
        <title>Свойство display в CSS3</title>
        <style>
            span{
                color: red;
            }
            .blockSpan{
                display: block;
            }
        </style>
    </head>
    <body>
        <div>Это <span>строчный</span> элемент span</div>
        <div>Это <span class="blockSpan">блочный</span> элемент span</div>
    </body>
</html>
```



Здесь определено два элемента span, но один из них является блочным, так как к нему применяется стиль `display: block;`. Поэтому этот элемент span переносится на новую строку.

В отличие от блочных элементов строчные встраиваются в строку, так как имеют для свойства `display` значение **inline**. Элемент span как раз по умолчанию имеет стиль `display: inline`, поэтому и встраивается в строку, а не переносится на следующую, как параграфы или div. И теперь произведем обратную процедуру - сделаем блочный элемент `div` строчным:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Свойство display в CSS3</title>
        <style>
            div{
                display: inline;
            }
        </style>
    </head>
    <body>
        <div>Первый строчный элемент div.</div>
        <div>Второй строчный элемент div.</div>
    </body>
</html>
```

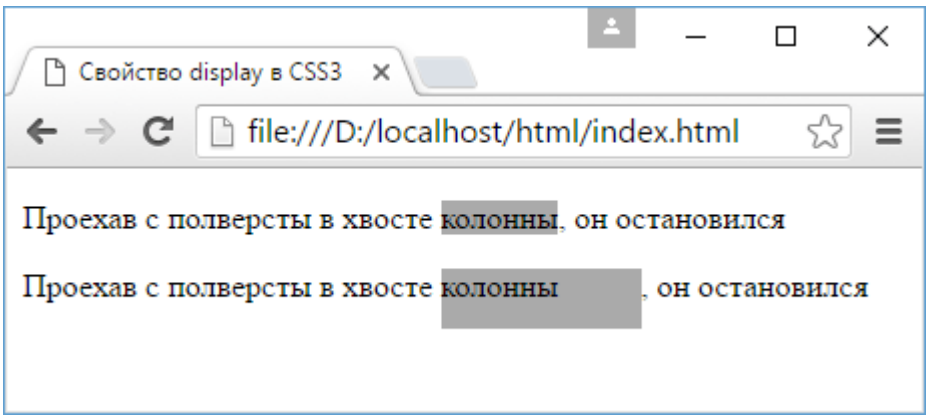


Следует учитывать, что при применении значения `inline` браузер игнорирует некоторые свойства, такие как `width`, `height`, `margin`.

inline-block

Еще одно значение - **inline-block** - представляет элемент, который обладает смесью признаков блочного и строчного элементов. По отношению к соседним внешним элементам такой элемент расценивается как строчный. То есть он не отделяется от соседних элементов переводом строки. Однако по отношению к вложенным элементам он рассматривается как блочный. И к такому элементу применяются свойства `width`, `height`, `margin`.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Свойство display в CSS3</title>
        <style>
            span{
                width: 100px;
                height: 30px;
                background-color: #aaa;
            }
            .inineBlockSpan{
                display: inline-block;
            }
        </style>
    </head>
    <body>
        <p>Проехав с полверсты в хвосте <span>колонны</span>, он остановился</p>
        <p>Проехав с полверсты в хвосте <span class="inineBlockSpan">колонны</span>,
        он остановился</p>
    </body>
</html>
```



Первый элемент span является строчным, у него значение `inline`, поэтому для него бессмысленно применять свойства `width` и `height`. А вот второй элемент `span` имеет значение `inline-block`, поэтому к нему жуе применяются и ширина, и высота, и при необходимости еще можно установить отступы.

run-in

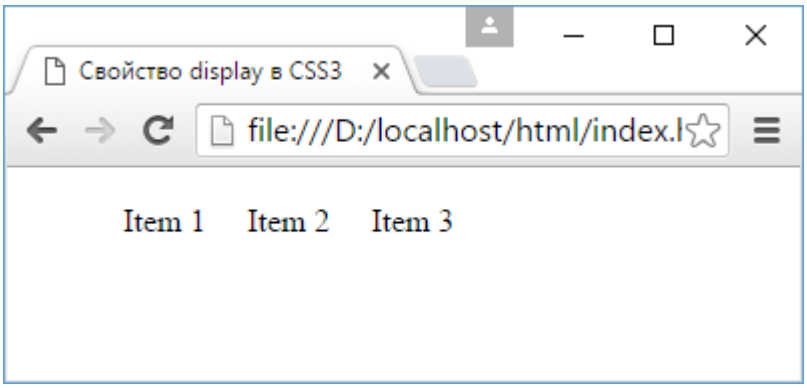
Значение **run-in** определяет элемент, который зависит от соседних элементов. И здесь есть три возможных варианта:

- Элемент окружен блочными элементами, тогда фактически он имеет стиль `display: block`, то есть сам становится блочным
- Элемент окружен строчными элементами, тогда фактически он имеет стиль `display: inline`, то есть сам становится строчным
- Во всех остальных случаях элемент считается блочным

Табличное представление

Значение **table** по сути превращает элемент в таблицу. Посмотрим на примере списка:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Свойство display в CSS3</title>
        <style>
            ul{
                display: table;
                margin: 0;
            }
            li{
                list-style-type: none;
                display: table-cell;
                padding: 10px;
            }
        </style>
    </head>
    <body>
        <ul>
            <li>Item 1</li>
            <li>Item 2</li>
            <li>Item 3</li>
        </ul>
    </body>
</html>
```

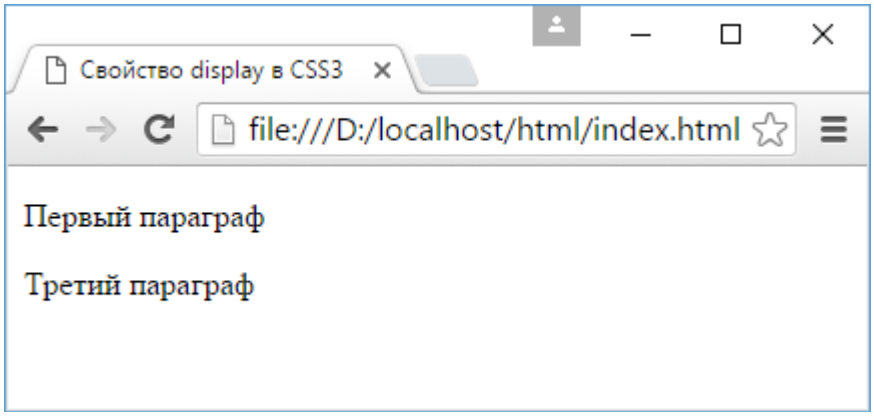


Здесь список превращается в таблицу, а каждый элемент списка - в отдельную ячейку. Для этого у элемента списка устанавливается стиль `display: table-cell`. Фактически вместо этого списка мы могли бы использовать стандартную таблицу.

Сокрытие элемента

Значение **none** позволяет скрыть элемент, которого как-будто нет на веб-странице:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Свойство display в CSS3</title>
        <style>
            .invisible{
                display: none;
            }
        </style>
    </head>
    <body>
        <p>Первый параграф</p>
        <p class="invisible">Второй параграф</p>
        <p>Третий параграф</p>
    </body>
</html>
```



Создание панели навигации

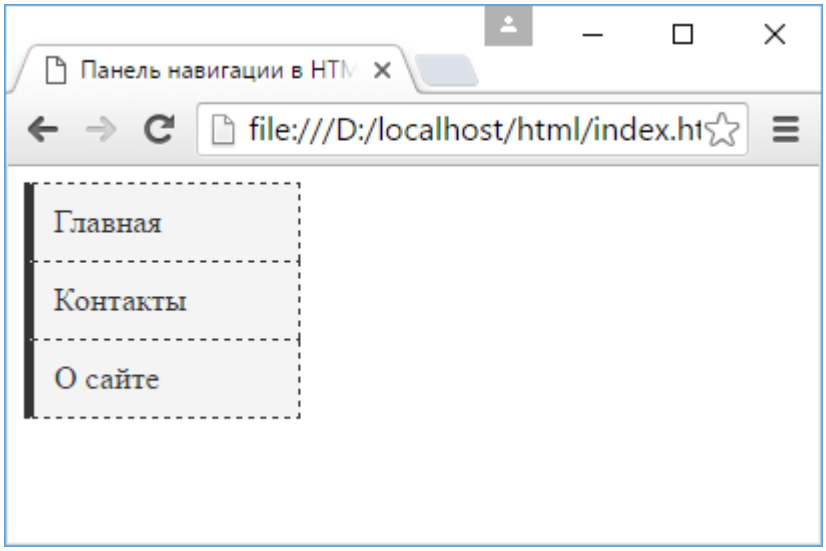
Панель навигации играет важную роль на сайте, так как обеспечиват переходы между страницами сайта или на внешние ресурсы. Рассмотрим, как создать простенькую панель навигации.

Фактически панель навигации - это набор ссылок, часто в виде ненумерованного списка. Панели навигации бывают самыми различными: вертикальными и горизонтальными, одноуровневыми и многоуровневыми, но в любом случае в центре каждой навигации находится элемент **<a>**. Поэтому при создании панели навигации мы можем столкнуться с рядом трудностей, которые вытекают из ограничений элемента ссылки. А именно, элемент `<a>` является строчным, а это значит, что мы не можем указать для него ширину, высоту, отступы. По ширине ссылка автоматически занимает то место, которое ей необходимо.

Вертикальное меню

Распространенное решение данной проблемы для создания вертикального меню состоит в том, чтобы сделать ссылку блочным элементом.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Панель навигации в HTML5</title>
        <style>
            ul.nav{
                margin-left: 0px;
                padding-left: 0px;
                list-style: none;
            }
            ul.nav a {
                display: block;
                width: 7em;
                padding:10px;
                background-color: #f4f4f4;
                border-top: 1px dashed #333;
                border-right: 1px dashed #333;
                border-left: 5px solid #333;
                text-decoration: none;
                color: #333;
            }
            ul.nav li:last-child a {
                border-bottom: 1px dashed #333;
            }
        </style>
    </head>
    <body>
        <ul class="nav">
            <li><a href="#">Главная</a></li>
            <li><a href="#">Контакты</a></li>
            <li><a href="#">О сайте</a></li>
        </ul>
    </body>
</html>
```



После установки свойства `display: block` мы можем определить у блока ссылки ширину, отступы и т.д.

Горизонтальное меню

Для создания горизонтального меню есть два метода. Первый заключается в применении свойства `float` и создании из ссылок плавающих элементов, которые обтекают друг друга с слева. И второй способ состоит в создании строки ссылок с помощью установки свойства `display: inline-block`.

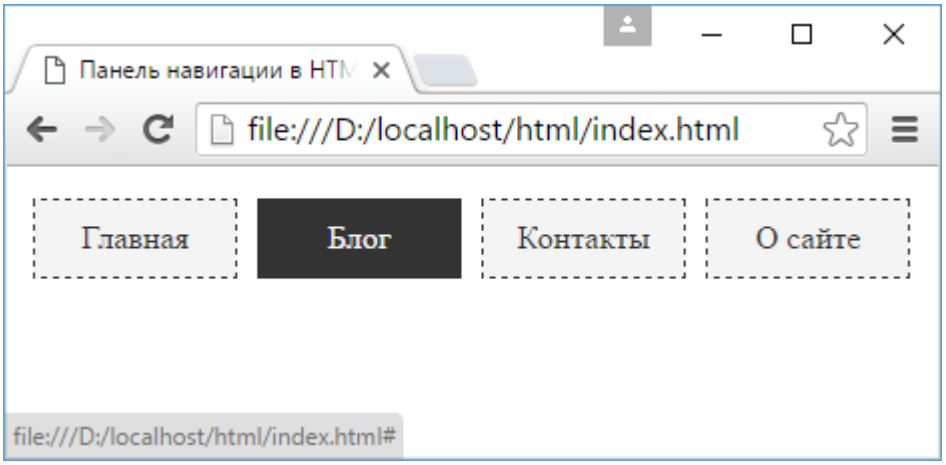
Использование float

Алгоритм создания панели навигации с помощью float разделяется на два этапа. На первом этапе у элемента **li**, в который заключена ссылка, устанавливается `float: left;`. Это позволяет расположить все элементы списка в ряд при достаточной ширине, когда правый элемент списка обтекает левый элемент списка.

Второй этап заключается в установке у элемента ссылки `display: block`, что дает нам возможность устанавливать ширину, отступы, вообщем все те признаки, которые характерны для блочных элементов.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Панель навигации в HTML5</title>
        <style>
            ul.nav{
                margin-left: 0px;
                padding-left: 0px;
                list-style: none;
            }
            .nav li {
                float: left;
            }
            ul.nav a {
                display: block;
                width: 5em;
                padding:10px;
```

```
                margin: 0 5px;
                background-color: #f4f4f4;
                border: 1px dashed #333;
                text-decoration: none;
                color: #333;
                text-align: center;
            }
            ul.nav a:hover{
                background-color: #333;
                color: #f4f4f4;
            }
        </style>
    </head>
    <body>
        <ul class="nav">
            <li><a href="#">Главная</a></li>
            <li><a href="#">Блог</a></li>
            <li><a href="#">Контакты</a></li>
            <li><a href="#">О сайте</a></li>
        </ul>
    </body>
</html>
```

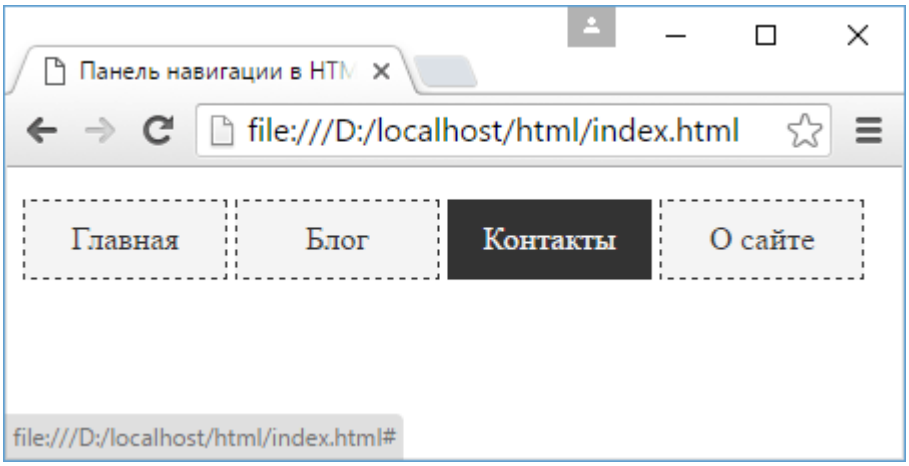


inline и inline-block

Для создания горизонтальной панели навигации нам надо сделать каждый элемент **li** строчным, то есть установить для него `display: inline`. После этого для элемента ссылки, которая располагается в элементе `li`, мы можем установить `display: inline-block`:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Панель навигации в HTML5</title>
        <style>
            ul.nav{
                margin-left: 0px;
                padding-left: 0px;
                list-style: none;
            }
            .nav li {
```

```
                display: inline;
            }
            ul.nav a {
                display: inline-block;
                width: 5em;
                padding:10px;
                background-color: #f4f4f4;
                border: 1px dashed #333;
                text-decoration: none;
                color: #333;
                text-align: center;
            }
            ul.nav a:hover{
                background-color: #333;
                color: #f4f4f4;
            }
        </style>
    </head>
    <body>
        <ul class="nav">
            <li><a href="#">Главная</a></li>
            <li><a href="#">Блог</a></li>
            <li><a href="#">Контакты</a></li>
            <li><a href="#">О сайте</a></li>
        </ul>
    </body>
</html>
```



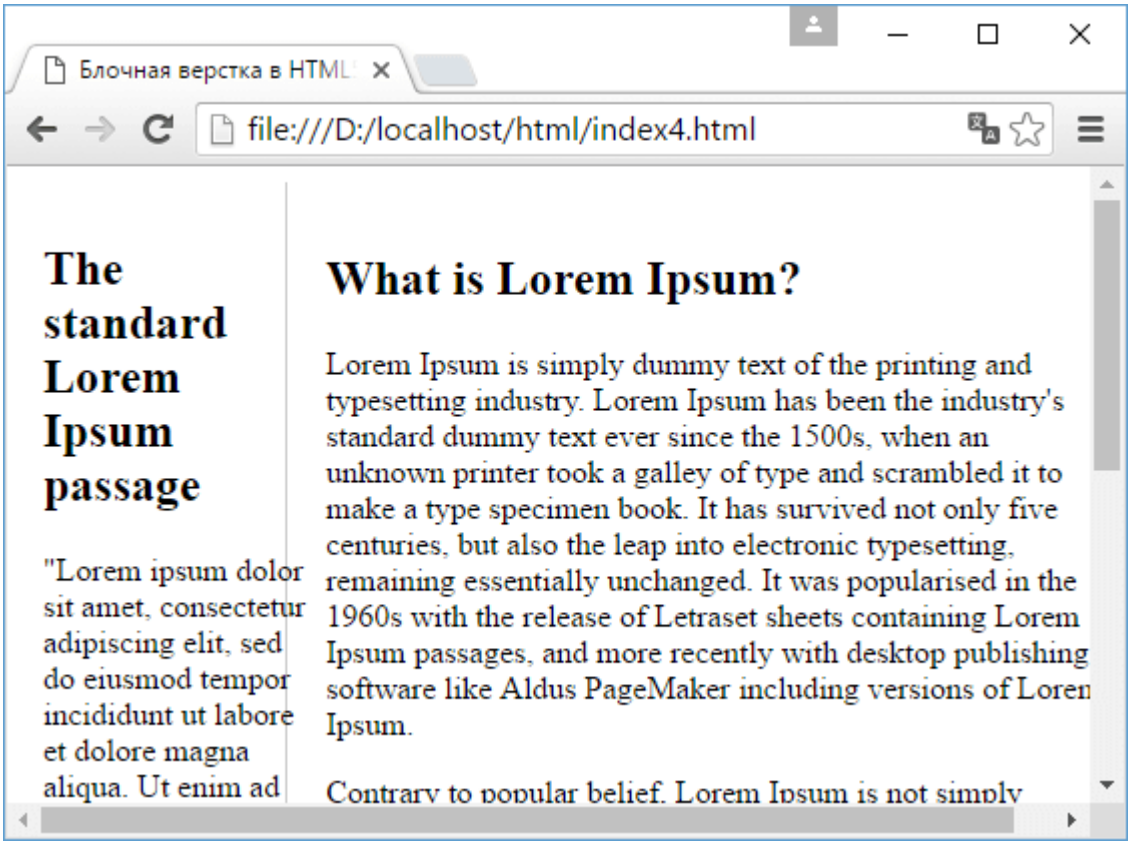
Выравнивание плавающих элементов

При работе с плавающими элементами и свойством **float** довольно часто можно столкнуться с проблемой выпадения из страницы плавающих элементов. У этой проблемы есть различные аспекты и их решения. Рассмотрим эти аспекты.

Например, пусть у нас задан следующий блок:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Блочная верстка в HTML5</title>
        <style>
            #sidebar{
                float: left;
                width: 25%;
                padding: 10px;
            }
            #main{
                border-left: 1px solid #ccc;
                width:75%;
                padding: 15px;
                margin-left: 25%;
            }
        </style>
    </head>
    <body>
        <div id="sidebar">
            <h2>The standard Lorem Ipsum passage</h2>
            <p>"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
                eiusmod tempor incididunt ut labore et dolore...</p>
        </div>
        <div id="main">
            <h2>What is Lorem Ipsum?</h2>
            <p>Lorem Ipsum is simply dummy text of the printing and typesetting
            industry...</p>
            <p>Contrary to popular belief, Lorem Ipsum is not simply random </p>
        </div>
    </body>
</html>
```

В сайдбаре довольно много текста, который, как ожидается, будет эффективно вписан в границы плавающего блока. Однако в реальности мы можем получить проблему:

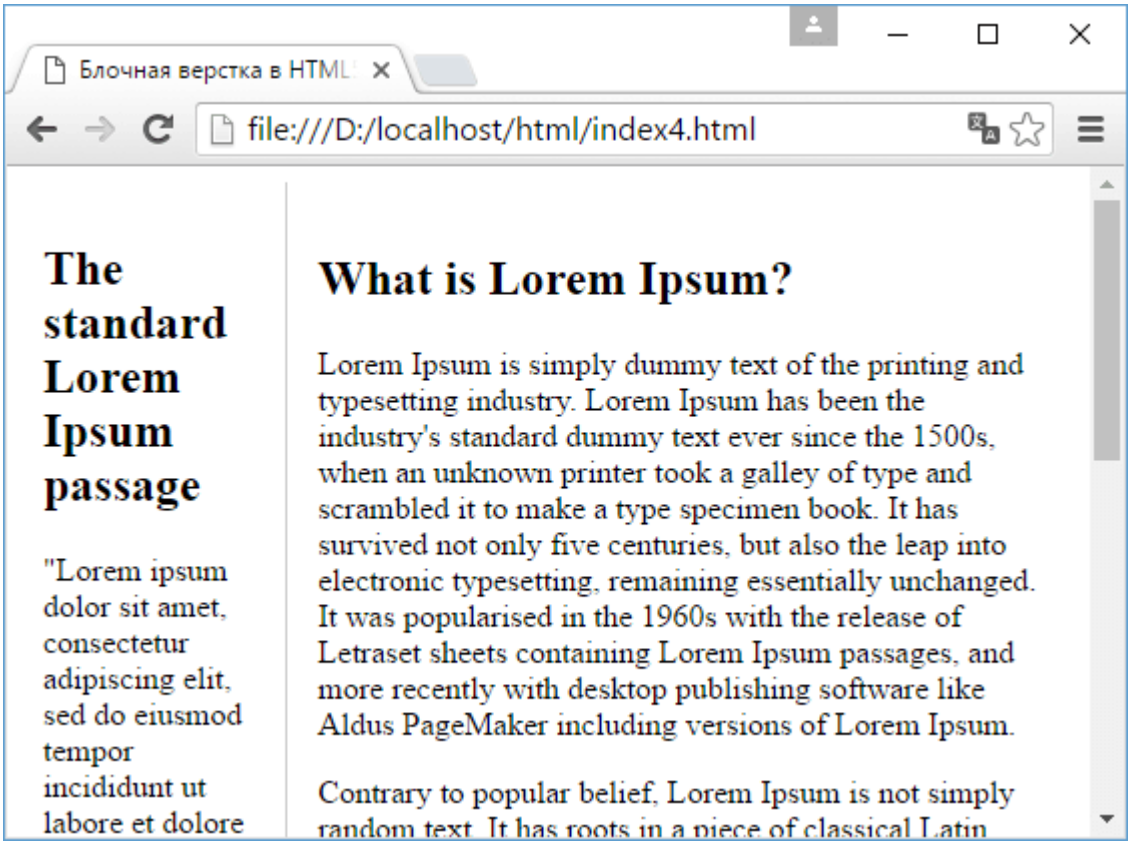


Как видно на скриншоте буквы вылезают из плавающего блока за границу, несмотря даже на то, что по идее в плавающем блоке должен быть установлен еще и внутренний отступ в 10 пикселей от правой границы.

Почему так происходит? Зачастую браузеры своеобразно интерпретируют размеры элемента. В частности, у всех элементов по умолчанию для свойства **box-sizing** используется значение **content-box**, то есть при определении ширины и высоты элемента браузер будет прибавлять к значению свойств `width` и `height` также и внутренние отступы padding и ширину границы. В итоге это может привести к выпадению плавающих элементов из тех блоков, которые для них предназначены. Поэтому часто для всех элементов рекомендуется устанавливать для свойства **box-sizing** значение **border-box**, чтобы все элементы измерялись одинаково, а их ширина представляла только значение свойства `width`. Поэтому нередко в стилях добавляется следующий стиль:

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

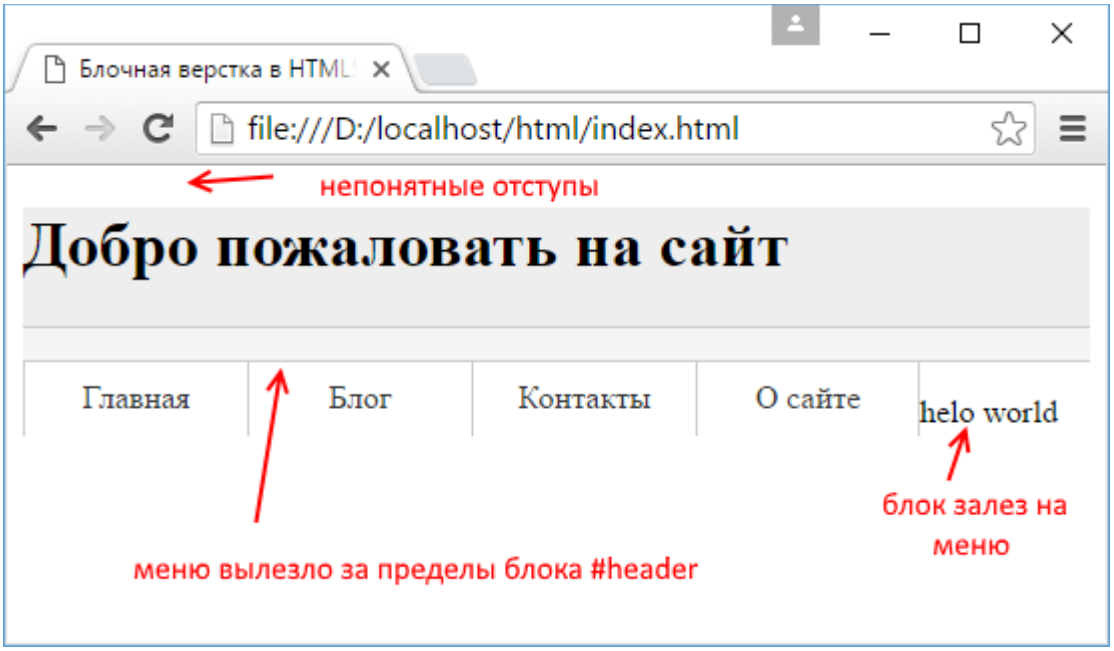
То есть значение `box-sizing: border-box;` устанавливается для всех элементов, и все они интерпретируются браузером одинаково. К примеру, добавим этот стиль в выше определенную страницу и мы получим уже несколько другой результат:



Рассмотрим другую проблему, которая связана с позиционированием плавающих элементов в контейнере:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Блочная верстка в HTML5</title>
        <style>
            *{
                box-sizing: border-box;
            }
            #header{
                background-color: #eee;
            }
            #nav{
                background-color: #f4f4f4;
                border-top: 1px solid #ccc;
                border-bottom: 1px solid #ccc;
            }
            #nav ul{
                margin-left: 0px;
                padding-left: 0px;
                list-style: none;
            }
            #nav li {
                float: left;
            }
            #nav ul a {
                display: block;
                width: 7em;
                padding:10px;
```

```
                border-left: 1px solid #ccc;
                text-decoration: none;
                color: #333;
                text-align: center;
            }
            #nav ul li:last-child a {

                border-right: 1px solid #ccc;
            }
            #nav ul a:hover{
                background-color: #aaa;
                color: #f4f4f4;
            }
        </style>
    </head>
    <body>
        <div id="header">
            <h1>Добро пожаловать на сайт</h1>
            <div id="nav">
                <ul>
                    <li><a href="#">Главная</a></li>
                    <li><a href="#">Блог</a></li>
                    <li><a href="#">Контакты</a></li>
                    <li><a href="#">О сайте</a></li>
                </ul>
            </div>
        </div>
        <div id="content"><p>helo world</p></div>
    </body>
</html>
```



Несмотря на то, что панель навигации определена в блоке с id header, визуально она явно не находится в элементе head. Ну и кроме того, здесь также можно увидеть, что появляются непонятные отступы, а следующий после хедера блок залезает на меню.

Проблема отступов заключается в том, что браузер по умолчанию определяет для различных элементов встроенные стили. Поэтому может немного сбивать с толку, как и где эти стили определены, почему они применются. Нередко для решения этой проблемы разработчики просто сбрасывают некоторые наиболее значимые стили для большинства элементов:

```
html, body, div, span, h1, h2, h3, h4, h5, h6, p, a, img, dl, dt, dd,
ol, ul, li, form, table, caption, tr, th, td, article, aside, footer, header{

    margin: 0;
    padding: 0;
    border: 0;
    font-size: 100%;
    vertical-align: baseline;
}
```

Другая проблема - наложение элемента div с основным контентом на плавающий блок навигационной панели решается довольно просто - установкой для этого элемента div следующего стиля:

```
clear: both;
```

Более сложной является проблема с выпадением плавающих элементов меню из границ блока-контейнера. Здесь есть два возможных варианта Решения. Первое решение состоит в добавлении к элементу, который представляет панель навигации, следующего стиля:

```
ul:after {
    content: " ";
    display: table;
    clear: both;
}
```

Второе решение состоит в том, чтобы сделать сам блок панели навигации плавающим:

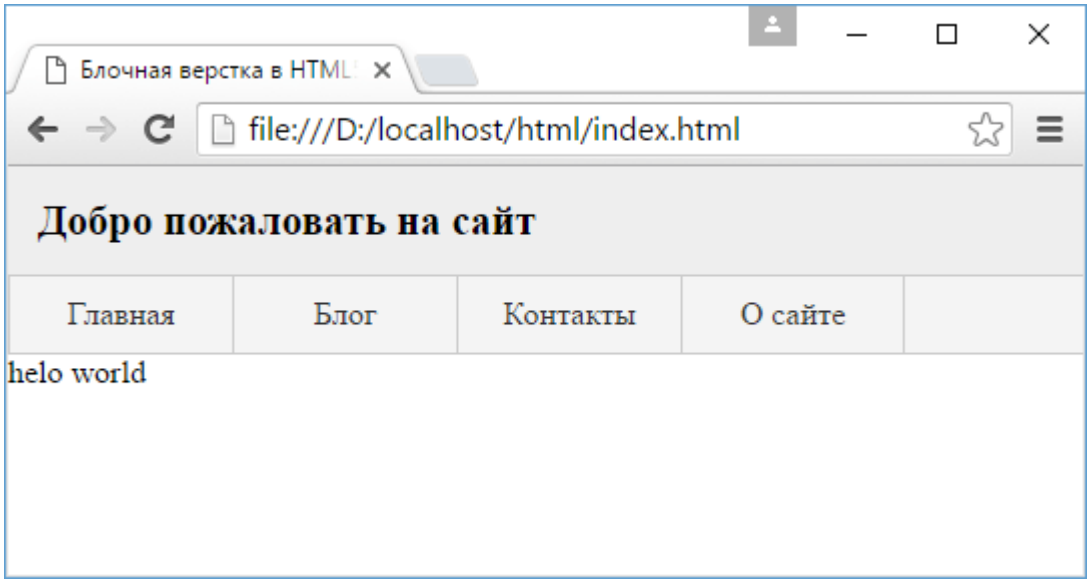
```
#nav{
    background-color: #f4f4f4;
    border-top: 1px solid #ccc;
    border-bottom: 1px solid #ccc;

    float:left;
    width: 100%;
    clear: both;
}
```

Итак, с учетом выше сказанного изменим стили для веб-страницы (код html остается прежним):

```
 *{
     box-sizing: border-box;
 }
 html, body, div, span, h1, h2, h3, h4, h5, h6, p, a, img, dl, dt, dd,
 ol, ul, li, form, table, caption, tr, th, td, article, aside, footer, header{

     margin: 0;
     padding: 0;
     border: 0;
```

```
    font-size: 100%;
    vertical-align: baseline;
}
#header{
    background-color: #eee;
}
#header h1{
    font-size: 1.3em;
    padding: 15px;
}
#nav{
    background-color: #f4f4f4;
    border-top: 1px solid #ccc;
    border-bottom: 1px solid #ccc;
}
#nav ul{
    margin-left: 0px;
    padding-left: 0px;
    list-style: none;
}
#nav li {
    float: left;
}
#nav ul a {
    display: block;
    width: 7em;
    padding: 10px;
    border-left: 1px solid #ccc;
    text-decoration: none;
    color: #333;
    text-align: center;
}
#nav ul li:last-child a {

    border-right: 1px solid #ccc;
}
#nav ul a:hover{
    background-color: #aaa;
    color: #f4f4f4;
}
#nav ul:after {
    content: " ";
    display: table;
    clear: both;
}
#content{
    clear: both;
}
```

И теперь веб-страница будет выглядеть иначе, собственно как и должна:


Блочная верстка в HTML
file:///D:/localhost/html/index.html
Добро пожаловать на сайт
Главная
Блог
Контакты
О сайте
helo world

Создание простейшего макета

Используем полученные из прошлых тем сведения и создадим более менее осмысленный макет самой простейшей веб-страницы. Для начала определим базовую структуру веб-страницы:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <link href="styles.css" rel="stylesheet">
        <title>Блочная верстка в HTML5</title>
    </head>
    <body>
        <div id="header">
            <h1>MySyte.com - Сайт о Lorem Ipsum</h1>
            <div id="nav">
                <ul>
                    <li><a href="#">Главная</a></li>
                    <li><a href="#">Блог</a></li>
                    <li><a href="#">Форум</a></li>
                    <li><a href="#">Контакты</a></li>
                    <li><a href="#">О сайте</a></li>
                </ul>
            </div>
        </div>
        <div class="wrapper">
            <div id="sidebar1" class="aside">
                <h2>The standard Lorem Ipsum passage</h2>
                <p>"Lorem ipsum dolor sit amet, consectetur adipiscing elit,
                sed do eiusmod tempor incididunt
                ut labore et dolore magna aliqua..."</p>
                </div>
            <div id="sidebar2" class="aside">
                <h2>1914 translation by H. Rackham</h2>
                <p>It is a long established fact that a reader will be distracted
                by the readable content of a page when looking at its layout.</p>
                <h3>Options</h3>
                <ul>
                    <li>Item1</li>
                    <li>Item2</li>
                    <li>Item3</li>
                </ul>
            </div>
            <div id="article">
                <h2>What is Lorem Ipsum?</h2>
                <p>Lorem Ipsum is simply dummy text of the printing and
                typesetting industry...</p><p>Contrary to popular belief,
                Lorem Ipsum is not simply random text. It has roots in a piece of
                classical Latin literature from 45 BC, making it over 2000 years
                old. Richard McClintock, a Latin professor at Hampden-Sydney
                College in Virginia...</p>
            </div>
        </div>
        <div id="footer">
```

```
            <p>Contacts: admin@mysyte.com</p>
            <p>Copyright © MySyte.com, 2016</p>
        </div>
    </body>
</html>
```

В начале идет шапка сайта - блок с header, который содержит заголовок страницы и панель навигации. Далее идет блок wrapper, в котором два сайдбара и блок основного содержимого страницы. Сайдбары условно тоже содержат некоторое содержимое, но главное, что они определены до основного блока. И в самом низу небольшой футер.

В начале веб-страницы определено поключение файла `styles.css`, который будет стилизовать веб-страницу. Поэтому создадим в одном каталоге с веб-страницей файл *styles.css* и определим в нем следующее содержимое:

```
* {
    box-sizing: border-box;
}
html, body, div, span, h1, h2, h3, h4, h5, h6, p, a, ul, li{

    margin: 0;
    padding: 0;
    border: 0;
    font-size: 100%;
    vertical-align: baseline;
}
body {
    font-family: Verdana, Arial, sans-serif;
    background-color: #f7f7f7;
}
#header{
    background-color: #f4f4f4;
}
#header h1 {
    font-size: 24px;
    text-transform: uppercase;
    font-weight: bold;
    padding: 30px 30px 30px 10px;
    clear: both;
}
#nav {
    background-color: #eee;
    border-top: 1px solid #ccc;
    border-bottom: 1px solid #ccc;
}
#nav li {
    float: left;
    list-style: none;
}
#nav a {
    display: block;
    color: black;
    padding: 10px 25px;
    text-decoration: none;
```

```
    border-right: 1px solid #ccc;
}
#nav li:last-child a {
    border-right: none;
}
#nav a:hover {
    font-weight: bold;
}
#nav:after {
    content: " ";
    display: table;
    clear: both;
}
.wrapper{
    background-color: #f7f7f7;
}
.aside h2 {
    font-size: 0.95em;
    margin-top: 15px;
}
.aside h3 {
    font-size: 0.85em;
    margin-top: 10px;
}
.aside p, .aside li {
    font-size: .75em;
    margin-top: 10px;
}
.aside li{
    list-style-type: none;
}
#sidebar1 {
    float: left;
    width: 20%;
    padding: 0 10px 0 20px;
}
#sidebar2 {
    float: right;
    width: 20%;
    padding: 0 20px 0 10px;
}
#article{
    background-color: #fafafa;
    border-left: 1px solid #ccc;
    border-right: 1px solid #ccc;
    margin-left: 20%;
    margin-right: 20%;
    padding: 15px;
    width: 60%;
}
#article:after{
    clear:both;
    display:table;
    content:'';
}
```

```
#article h2{
    font-size: 1.3em;
    margin-bottom:15px;
}
#article p{
    line-height: 150%;
    margin-bottom: 15px;
}
#footer{
    border-top: 1px solid #ccc;
    font-size: .8em;
    text-align: center;
    padding: 10px 10px 30px 10px;
}
#nav ul, #header h1, .wrapper, #footer p {
    max-width: 1200px;
    margin: 0 auto;
}
.wrapper, #nav, #header, #footer{
    min-width: 768px;
}
```

Первые три стиля сбрасывают стилевые настройки по умолчанию для используемых нами элементов, а также устанавливат стиль элемента body.

Следующая пара стилей управляет отображением шапки (хедера) и заголовка страницы:

```
#header{
    background-color: #f4f4f4;
}
#header h1 {
    font-size: 24px;
    text-transform: uppercase;
    font-weight: bold;
    padding: 30px 30px 30px 10px;
    clear: both;
}
```

Следующий набор стилей управляет созданием горизонтальной панели навигации:

```
#nav {
    background-color: #eee;
    border-top: 1px solid #ccc;
    border-bottom: 1px solid #ccc;
}
#nav li {
    float: left;
    list-style: none;
}
#nav a {
    display: block;
    color: black;
    padding: 10px 25px;
    text-decoration: none;
    border-right: 1px solid #ccc;
```

```
}
#nav li:last-child a {
    border-right: none;
}
#nav a:hover {
    font-weight: bold;
}
#nav:after {
    content: " ";
    display: table;
    clear: both;
}
```

В одной из прошлых тем подробно обсуждалось создание горизонтальной панели навигации. В принципе здесь ничего нового не добавляется: для элементов `<li>` устанавливается обтекание (`float: left;`), благодаря чему они размещаются в ряд, а каждая ссылка делается блочным элементом (`display: block;`)

Далее идет настройка средней части страницы, в частности, сайдбаров:

```
.wrapper{
    background-color: #f7f7f7;
}
.aside h2 {
    font-size: 0.95em;
    margin-top: 15px;
}
.aside h3 {
    font-size: 0.85em;
    margin-top: 10px;
}
.aside p, .aside li {
    font-size: .75em;
    margin-top: 10px;
}
.aside li{
    list-style-type: none;
}
#sidebar1 {
    float: left;
    width: 20%;
    padding: 0 10px 0 20px;
}
#sidebar2 {
    float: right;
    width: 20%;
    padding: 0 20px 0 10px;
}
```

Стиль класса wrapper позволяет установить фоновый цвет для боковых панелей. Для каждого сайдбара определяется ширина в 20% от ширины страницы. Процентные значения позволят автоматически подстраивать ширину блоков под ширину окна браузера при его расширении или сужении.

Далее следуют стили блока основного содержимого и футера:

```
#article{
    background-color: #fafafa;
    border-left: 1px solid #ccc;
    border-right: 1px solid #ccc;
    margin-left: 20%;
    margin-right: 20%;
    padding: 15px;
    width: 60%;
}
#article:after{
    clear:both;
    display:table;
    content:'';
}
#article h2{
    font-size: 1.3em;
    margin-bottom:15px;
}
#article p{
    line-height: 150%;
    margin-bottom: 15px;
}
#footer{
    border-top: 1px solid #ccc;
    font-size: .8em;
    text-align: center;
    padding: 10px 10px 30px 10px;
}
```

Поскольку боковые панели имеют ширину в 20% каждая, то для главного блока устанавливается ширина в 60% и отступы справа и слева в 20%.

И в конце идет пара довольно важных стилей:

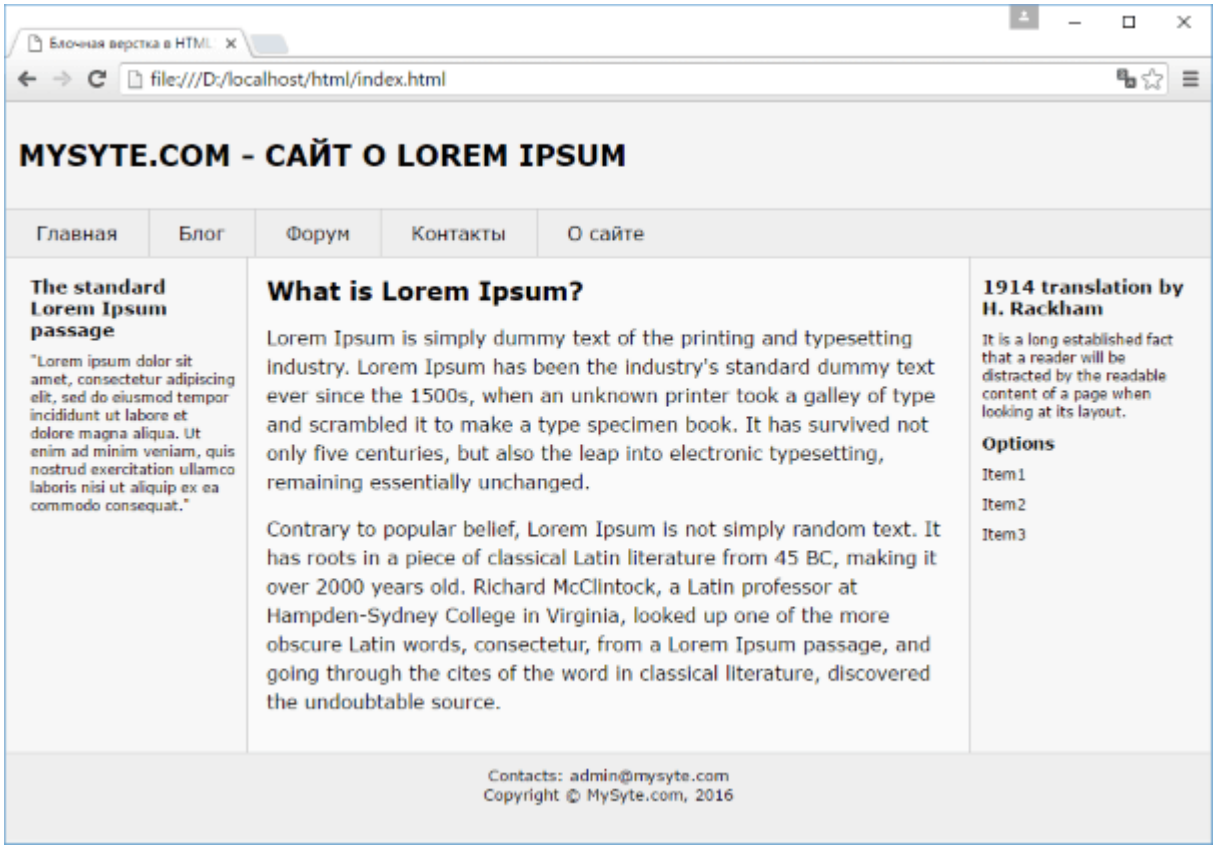
```
#nav ul, #header h1, .wrapper, #footer p {
    max-width: 1200px;
    margin: 0 auto;
}
.wrapper, #nav, #header, #footer{
    min-width: 768px;
}
```

В начале для ряда селекторов определяется максимальная ширина в 1200 пикселей. Это значит, что основные элементы страницы не выйдут за пределы 1200 пикселей. А автоматический внешний отступ слева и справа позволит центрировать содержимое элементов. То есть при ширине браузера в 1400 пикселей эти элементы с шириной в 1200 пикселей будут размещаться как бы по середине, а справа и слева будут отступы шириной в (1400-1200)/2 = 100 пикселей.

Второй стиль позволят сделать фиксированную минимальную ширину для ряда элементов. То есть в итоге при сжатии окна браузера сайдбары и основной блок будут выглядеть более менее, а при сжатии окна менее 768 пикселей образуется полоса прокрутки.

Данная модель размеров не идеальна. И далее мы рассмотрим более гибкие и распространенные концепции на основе адаптивной верстки.

В итоге у нас получится следующий простенький макет:



Позиционирование

CSS предоставляет возможности по позиционированию элемента, то есть мы можем поместить элемент в определенное место на странице

Основным свойством, которые управляют позиционированием в CSS, является свойство **position**. Это свойство может принимать одно из следующих значений:

- **static**: стандартное позиционирование элемента, значение по умолчанию
- **absolute**: элемент позиционируется относительно границ элемента-контейнера, если у того свойство `position` не равно `static`
- **relative**: элемент позиционируется относительно его позиции по умолчанию. Как правило, основная цель относительного позиционирования заключается не в том, чтобы переместить элемент, а в том, чтобы установить новую точку привязки для абсолютного позиционированния вложенных в него элементов
- **fixed**: элемент позиционируется относительно окна бразуера, это позволяет создать фиксированные элементы, которые не меняют положения при прокрутке

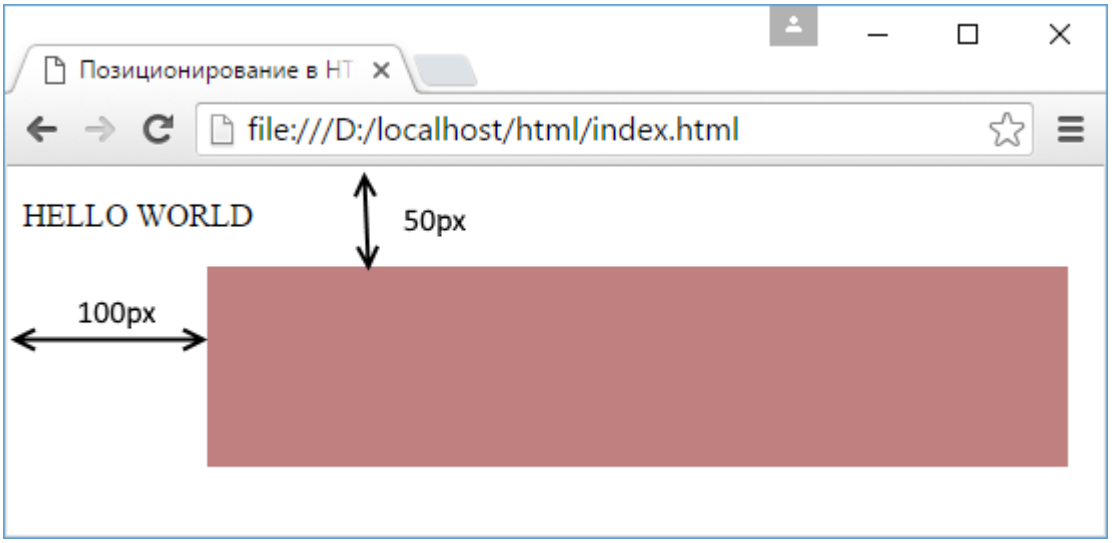
Не следует одновременно применять к элементу свойство float и любой тип позиционирования, кроме static (то есть тип по умолчанию).

Абсолютное позиционирование

Область просмотра браузера имеет верхний, нижний, правый и левый края. Для каждого из этих четырех краев есть соответствующее свойство CSS: `left` (отступ от края слева), `right` (отступ от края справа), `top` (отступ от края контейнера сверху) и `bottom` (отступ снизу). Значения этих свойств указываются в пикселях, em или процентах. Необязательно задавать значения для всех четырех сторон. Как правило, устанавливают только два значения - отступ от верхнего края top и отступ от левого края left.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Блочная верстка в HTML5</title>
        <style>
            .header {
                position: absolute;
                left: 100px;
                top: 50px;
                width: 430px;
                height: 100px;
                background-color: rgba(128, 0, 0, 0.5);
            }
        </style>
    </head>
    <body>
        <div class="header"></div>
        <p>HELLO WORLD</p>
    </body>
</html>
```

Здесь элемент div с абсолютным позиционированием будет находиться на 100 пикселей слева от границы области просмотра и на 50 снизу.

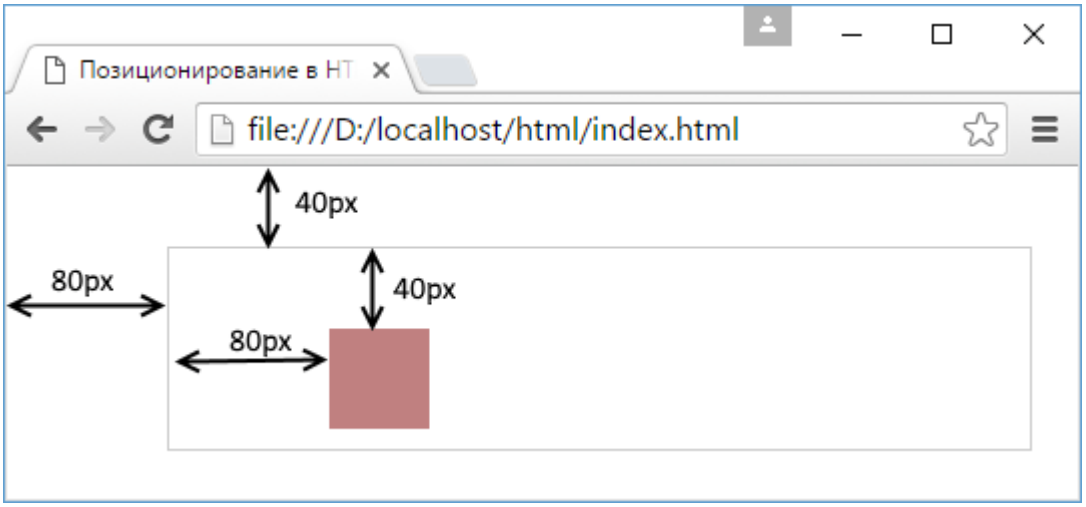


При этом не столь важно, что после этого элемента div идут какие-то другие элементы. Данный блок div в любом случае будет позиционироваться относительно границ области просмотра браузера.

Если элемент с абсоютным позиционированием располагается в другом контейнере, у которого в свою очередь значение свойства `position` не равно `static`, то элемент позиционируется относительно границ контейнера:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Позиционирование в HTML5</title>
        <style>
            .outer {
                position: absolute;
                left: 80px;
                top: 40px;
                width: 430px;
                height: 100px;
                border: 1px solid #ccc;
            }
            .inner{
                position: absolute;
                left: 80px;
                top: 40px;
                width: 50px;
                height: 50px;
                background-color: rgba(128, 0, 0, 0.5);
            }
        </style>
    </head>
    <body>
        <div class="outer">
            <div class="inner"></div>
        </div>
```

```
    </body>
</html>
```



Относительное позиционирование

Относительное позиционирование также задается с помощью значения **relative**. Для указания конкретной позиции, на которую сдвигается элемент, применяются те же свойства top, left, right, bottom:

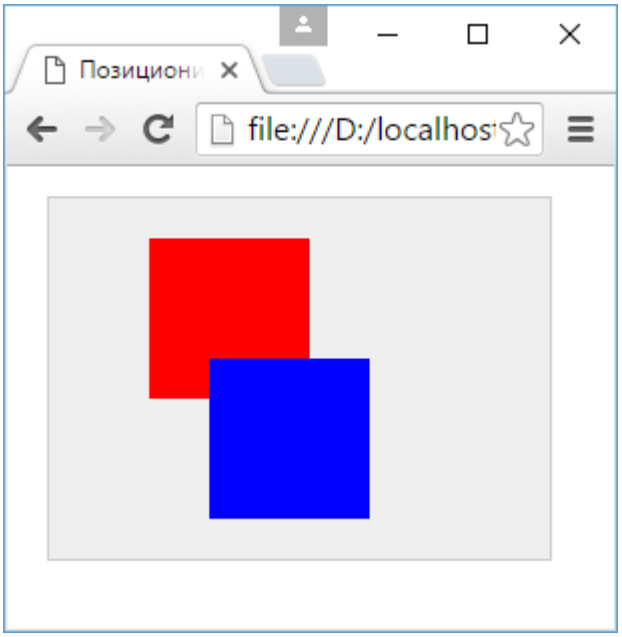
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Позиционирование в HTML5</title>
        <style>
            .outer {
                position: relative;
                left: 80px;
                top: 40px;
                width: 300px;
                height: 100px;
                border: 1px solid #ccc;
            }
            .inner{
                position: absolute;
                left: 80px;
                top: 40px;
                width: 50px;
                height: 50px;
                background-color: rgba(128, 0, 0, 0.5);
            }
        </style>
    </head>
    <body>
        <div class="outer"><div class="inner"></div></div>
    </body>
</html>
```

Свойство z-index

По умолчанию при совпадении у двух элементов границ, поверх другого отображается тот элемент, который определен в разметке html последним. Однако свойство **z-index** позволяет изменить порядок следования элементов при их наложении. В качестве значения свойство принимает число. Элементы с большим значением этого свойства будут отображаться поверх элементов с меньшим значением z-index.

Например:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Позиционирование в HTML5</title>
        <style>
            body{
                margin:0;
                padding:0;
            }
            .content{
                position: relative;
                top: 15px;
                left: 20px;
                width: 250px;
                height: 180px;
                background-color: #eee;
                border: 1px solid #ccc;
            }
            .redBlock{
                position: absolute;
                top: 20px;
                left:50px;
                width: 80px;
                height: 80px;
                background-color: red;
            }
            .blueBlock{
                position: absolute;
                top: 80px;
                left: 80px;
                width: 80px;
                height: 80px;
                background-color: blue;
            }
        </style>
    </head>
    <body>
        <div class="content">
            <div class="redBlock"></div>
            <div class="blueBlock"></div>
        </div>
    </body>
</html>
```

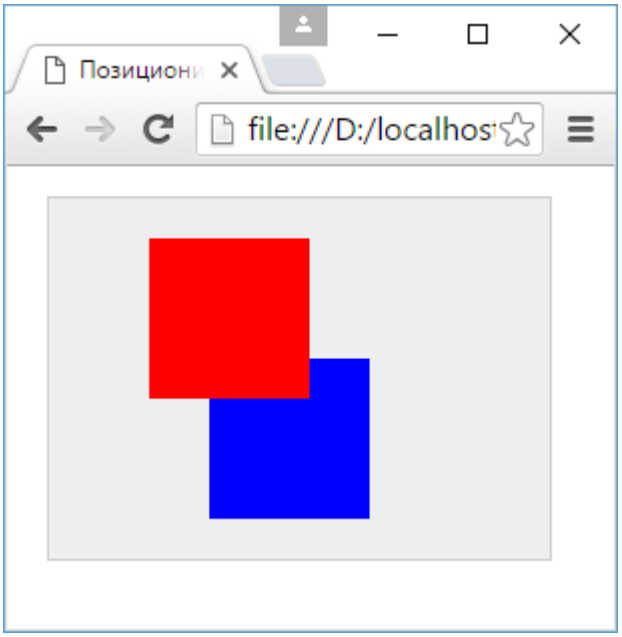


Теперь добавим к стилю блока redBlock новое правило:

```
1    .redBlock{
2        z-index: 100;
3
4        position: absolute;
5        top: 20px;
6        left:50px;
7        width: 80px;
8        height: 80px;
9        background-color: red;
10   }
```

Здесь z-index равен 100. Но это необязательно должно быть число 100. Так как у второго блока z-index не определен и фактически равен нулю, то для redBlock мы можем установить у свойства z-index любое значение больше нуля.

И теперь первый блок будет накладываться на второй, а не наоборот, как было в начале:



Фиксированное позиционирование

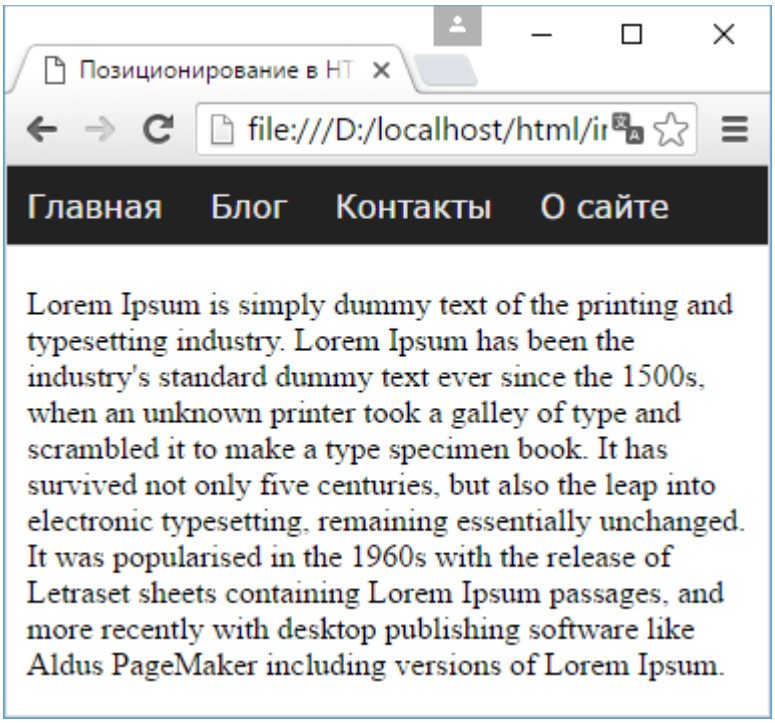
Фиксированное позиционирование является распространенным способом удержать в области просмотра браузера некоторые элементы. Достаточно часто на различных сайтах можно увидеть фиксированную панель навигации, которая не изменяет своего положения вне зависимости от прокрутки.

Для фиксированного позиционирования у элементов нужно установить значение **fixed** для свойства **position**. После этого с помощью стандартных свойств `left`, `right`, `top` и `bottom` можно определить конкретную позицию фиксированного элемента.

Создадим к примеру фиксированную панель навигации:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Позиционирование в HTML5</title>
        <style>
            body{
                margin:0;
                padding:0;
            }
            .toolbar{
                position: fixed;
                top: 0;
                left: 0;
                right: 0;
                background-color: #222;
                border-bottom: 1px solid #ccc;
            }
            .toolbar a{
                color: #eee;
                display: inline-block;
                padding: 10px;
                text-decoration: none;
                font-family: Verdana;
            }
            .content{
                margin-top: 50px;
                padding: 10px;
            }
        </style>
    </head>
    <body>
        <div class="toolbar">
            <a href="#">Главная</a>
            <a href="#">Блог</a>
            <a href="#">Контакты</a>
            <a href="#">О сайте</a>
        </div>
        <div class="content">Lorem Ipsum is simply dummy text of the printing and
        typesetting industry. Lorem Ipsum has been the industry....</div>
    </body>
```

```
</html>
```



Чтобы растянуть фиксированный блок от левой до правой границы страницы, устанавливаются три свойства:

```
top: 0;
left: 0;
right: 0;
```

Для нижележащего блока с основным содержанием фиксированный элемент фактически не существует разметке, так как блок с фиксированным, как и с абсолютным позиционированием не учавствуют в стандартном потоке html. Поэтому по умолчанию оба блока будут накладываться друг на друга и размещатьс в одной точке. И нам надо должным образом разместить блок содержимого относительно фиксированного блока, например, установив нужный отступ:

```
margin-top: 50px;
```

Фактически отступ идет от границ области просмотра браузера, поэтому высота отступа должна быть больше высоты фиксированного элемента.

Трансформации, переходы и анимации

Трансформации

Одним из нововведений CSS3 по сравнению с предыдущей версией является встроенная возможность трансформаций элемента. К трансформациям относятся такие действия, как вращение элемента, его масштабирование, наклон или перемещение по вертикали или горизонтали. Для создания трансформаций в CSS3 применяется свойство **transform**.

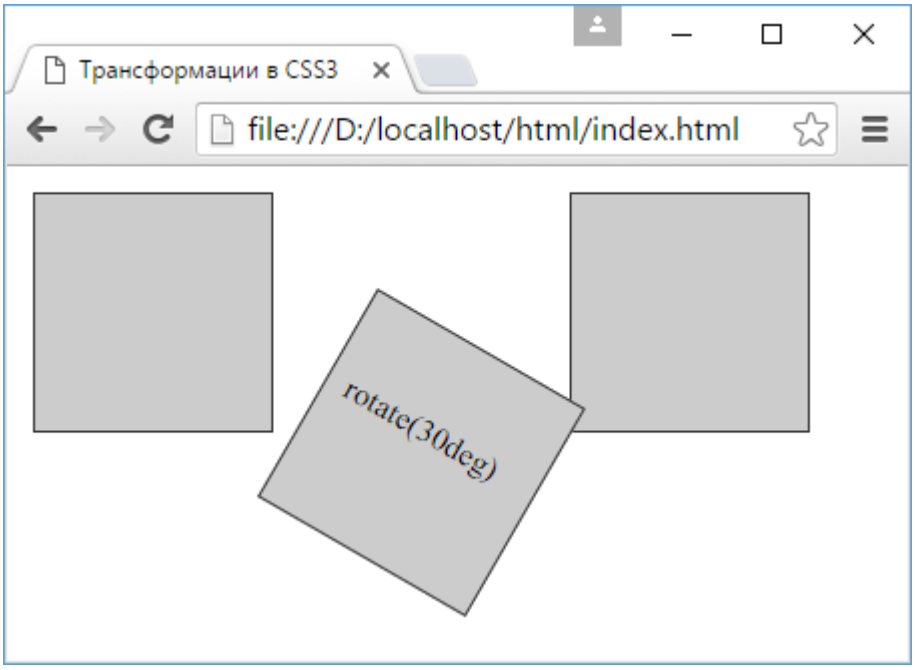
Вращение

Для поворота элемента свойство `transform` использует функцию **rotate**:

```
transform: rotate(угол_поворота deg);
```

После слова **rotate** в скобках идет величина угла поворота в градусах. Например, повернем блок на 30 градусов:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Трансформации в CSS3</title>
        <style>
            div{
                background-color: #ccc;
                width: 120px;
                height: 120px;
                margin:5px;
                padding: 40px 10px;
                box-sizing: border-box;
                border: 1px solid #333;
                display: inline-block;
            }
            .rotated{
                transform: rotate(30deg);
            }
        </style>
    </head>
    <body>
        <div></div>
        <div class="rotated">rotate(30deg)</div>
        <div></div>
    </body>
</html>
```



При этом можно отметить, что при повороте вращаемый элемент может накладываться на соседние элементы, так как сначала происходит установка положения элементов и только затем поворот.

Угол поворота может представлять как положительное, так и отрицательное значение. В случие отрицательного значения поворот производится в противоположную сторону.

Масштабирование

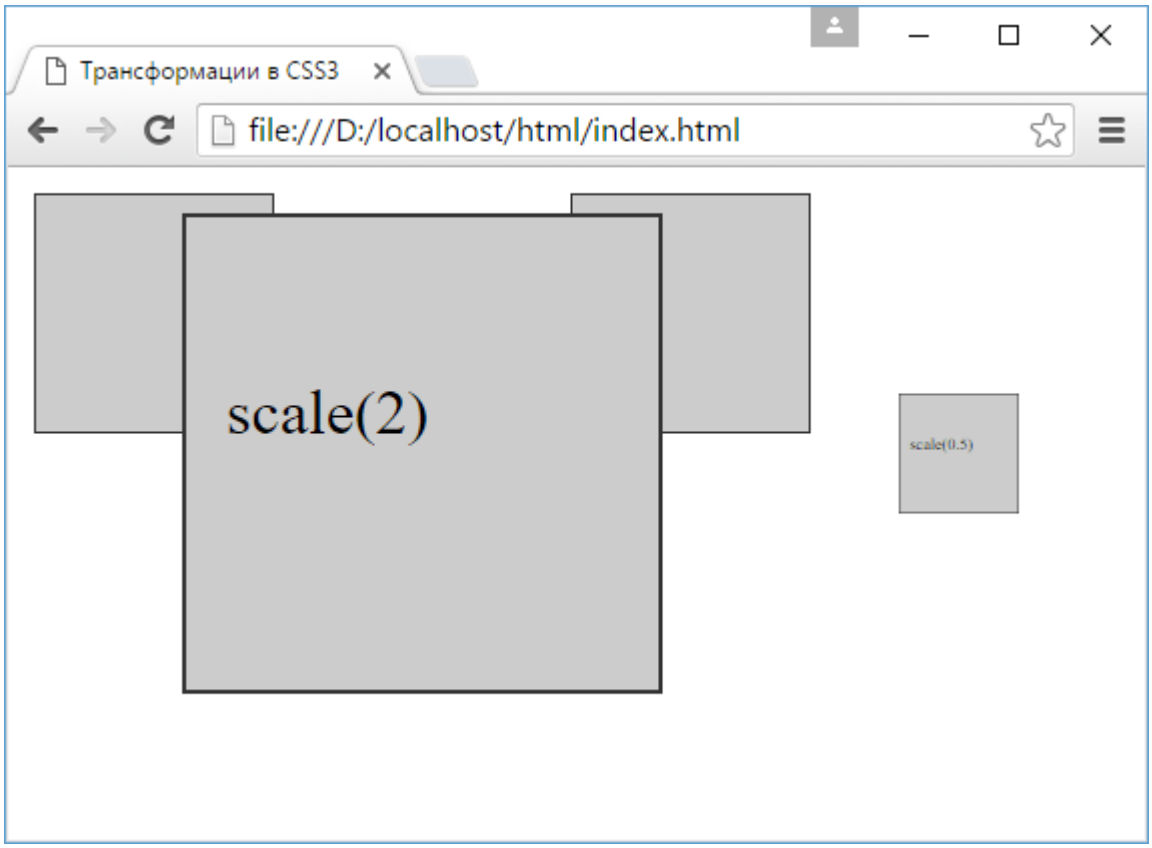
Применение масштабирования имеет следующую форму:

```
transform: scale(величина_масштабирования);
```

Используем маштабирование:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Трансформации в CSS3</title>
        <style>
            div{
                background-color: #ccc;
                width: 120px;
                height: 120px;
                margin:5px;
                padding: 40px 10px;
                box-sizing: border-box;
                border: 1px solid #333;
                display: inline-block;
            }
            .halfScale{
                transform: scale(0.5);
            }
            .doubleScale{
                transform: scale(2);
```

```
            }
        </style>
    </head>
    <body>
        <div></div>
        <div class="doubleScale">scale(2)</div>
        <div></div>
        <div class="halfScale">scale(0.5)</div>
    </body>
</html>
```

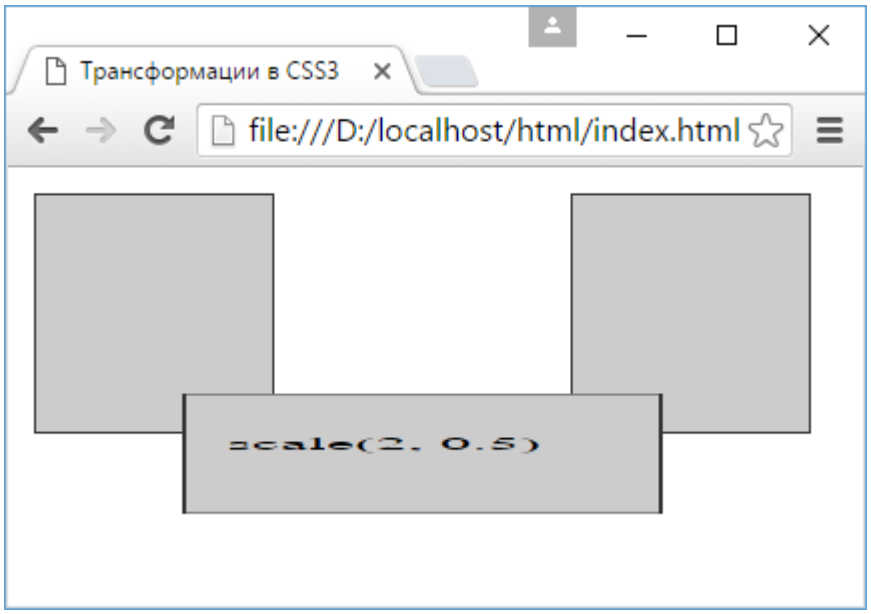


Значение больше 1 приводит к растяжению по вертикали и горизонтали, а меньше 1 - к сжатию. То есть значение 0.5 приводит к уменьшению в два раза, а значение 1.5 - к увеличению в полтора раза.

Можно также задать величины масштабирования отдельно для вертикали и горизонтали:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Трансформации в CSS3</title>
        <style>
            div{
                background-color: #ccc;
                width: 120px;
                height: 120px;
                margin:5px;
                padding: 40px 10px;
                box-sizing: border-box;
                border: 1px solid #333;
                display: inline-block;
```

```
            }
            .scale1{
                transform: scale(2, 0.5);
            }
        </style>
    </head>
    <body>
        <div></div>
        <div class="scale1">scale(2, 0.5)</div>
        <div></div>
    </body>
</html>
```



В данном случае по горизонтали будет идти масштабирование в 2 раза, а по вертикали - в 0.5 раз.

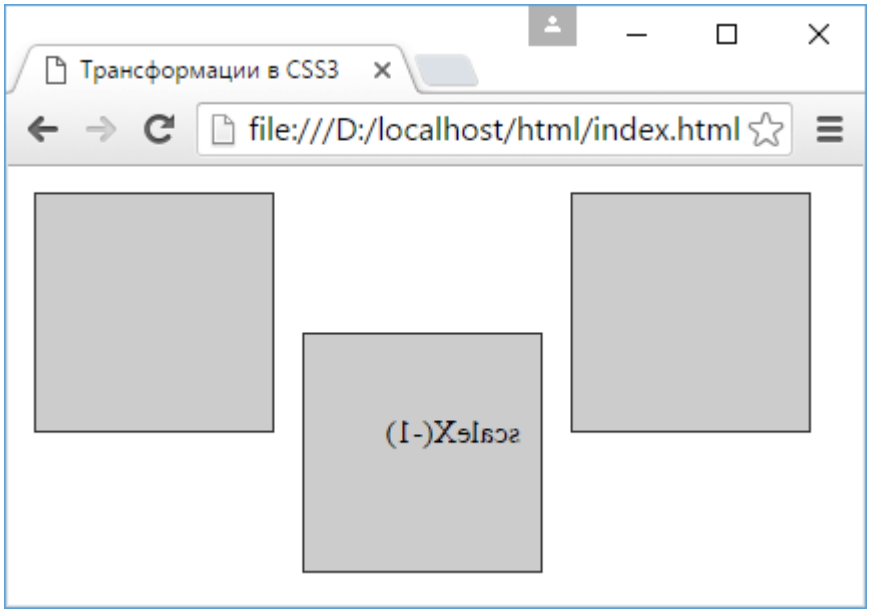
Также можно по отдельности задать параметры масштабирования: функция **scaleX()** задает изменение по горизонтали, а **scaleY()** - по вертикали. Например:

```
.scale1{
    transform: scaleX(2);
}
```

Используя отрицательные значения, можно создать эффект зеркального отражения:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Трансформации в CSS3</title>
        <style>
            div{
                background-color: #ccc;
                width: 120px;
                height: 120px;
                margin:5px;
                padding: 40px 10px;
                box-sizing: border-box;
```

```
                border: 1px solid #333;
                display: inline-block;
            }
            .scale1{
                transform: scaleX(-1);
            }
        </style>
    </head>
    <body>
        <div></div>
        <div class="scale1">scaleX(-1)</div>
        <div></div>
    </body>
</html>
```



Перемещение

Для перемещения элемента используется функция **translate**:

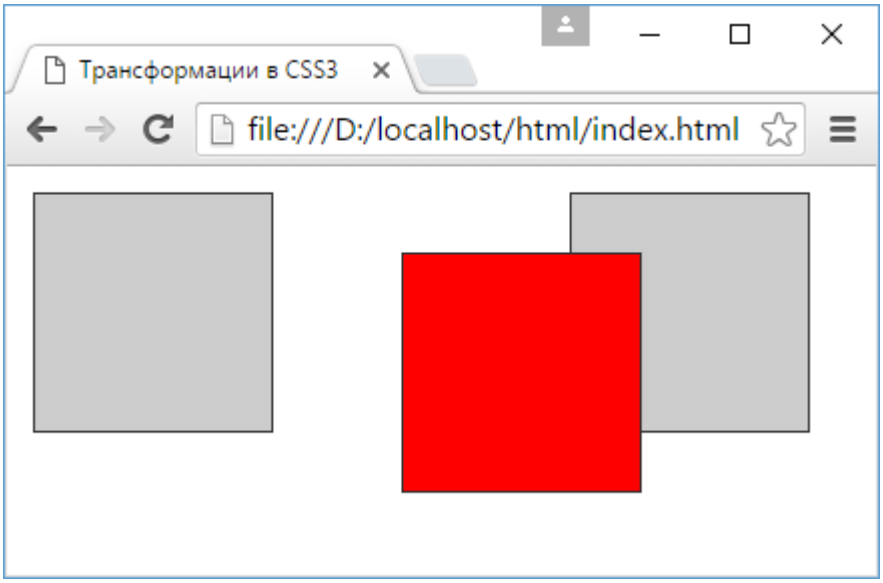
```
transform: translate(offset_X, offset_Y);
```

Значение `offset_X` указывает, на сколько элемент смещается по горизонтали, а `offset_Y` - по вертикали.

К примеру, сместим блок на 30 пикселей вниз и на 50 пикселей вправо:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Трансформации в CSS3</title>
        <style>
            div{
                background-color: #ccc;
                width: 120px;
                height: 120px;
                margin:5px;
                padding: 40px 10px;
```

```
                box-sizing: border-box;
                border: 1px solid #333;
                display: inline-block;
            }
            .translated{
                transform: translate(50px, 30px);
                background-color:red;
            }
        </style>
    </head>
    <body>
        <div></div>
        <div class="translated"></div>
        <div></div>
    </body>
</html>
```



В качестве единиц измерения смещения можно применять не только пиксели, но и любые другие единицы измерения длины в CSS - em, % и тд.

С помощью дополнительных функций можно отдельно применять смещения к горизонтали или вертикали: **translateX()** (перемещение по горизонтали) и **translateY()** (перемещение по вертикали). Например:

```
transform: translateX(30px);
```

Кроме положительных значений также можно использовать и отрицательные - они перемещают элемент в противоположную сторону:

```
transform: translateY(-2.5em);
```

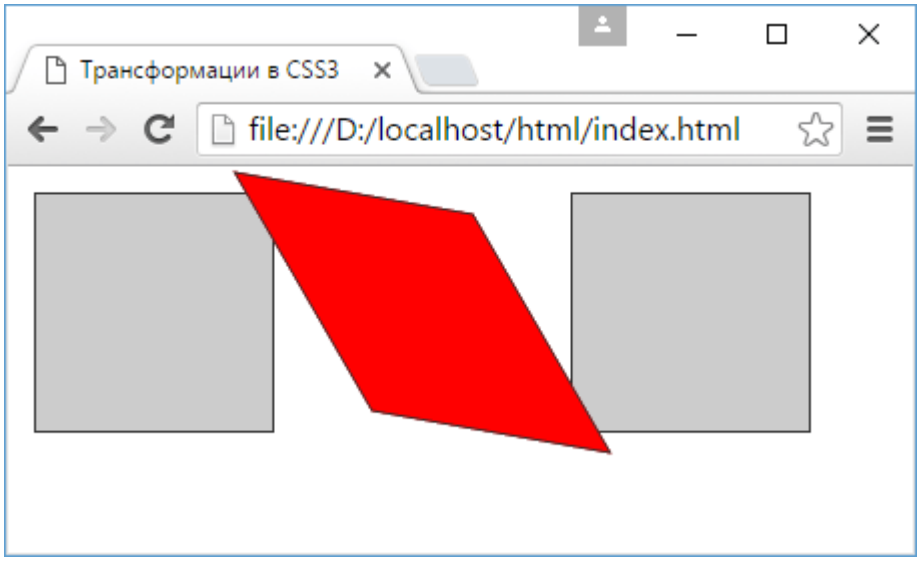
Наклон

Для наклона элемента применяется функция **skew()**:

```
transform: skew(X, Y);
```

Первый параметр указывает, на сколько градусов наклонять элемент по оси X, а второй - значение наклона по оси Y.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Трансформации в CSS3</title>
        <style>
            div{
                background-color: #ccc;
                width: 120px;
                height: 120px;
                margin:5px;
                padding: 40px 10px;

                box-sizing: border-box;
                border: 1px solid #333;
                display: inline-block;
            }
            .skewed{
                transform: skew(30deg, 10deg);
                background-color:red;
            }
        </style>
    </head>
    <body>
        <div></div>
        <div class="skewed"></div>
        <div></div>
    </body>
</html>
```



Для создания наклона только по одной оси для другой оси надо использовать значение 0. Например, наклон на 45 градусов по оси X:

```
transform: skew(45deg, 0);
```

Или наклон на 45 градусов только по оси Y:

```
transform: skew(0,45deg);
```

Для создания наклона отдельно по оси X и по оси Y в CSS есть специальные функции: **skewX()** и **skewY()** соответственно.

```
transform: skewX(45deg);
```

Также можно передавать отрицательные значения. Тогда наклон будет осуществляться в противоположную сторону:

```
transform: skewX(-30deg);
```

Комбинирование преобразований

Если нам надо применить к элементу сразу несколько преобразований, скажем, вращаение и перемещение, то мы можем их комбинировать. Например, применение всех четырех преобразований:

```
transform: translate(50px, 100px) skew(30deg, 10deg) scale(1.5) rotate(90deg);
```

Браузер применяет все эти функции в порядке их следования. То есть в данном случае сначала к элементу применяется перемещение, потом наклон, потом масштабирование и в конце вращение.

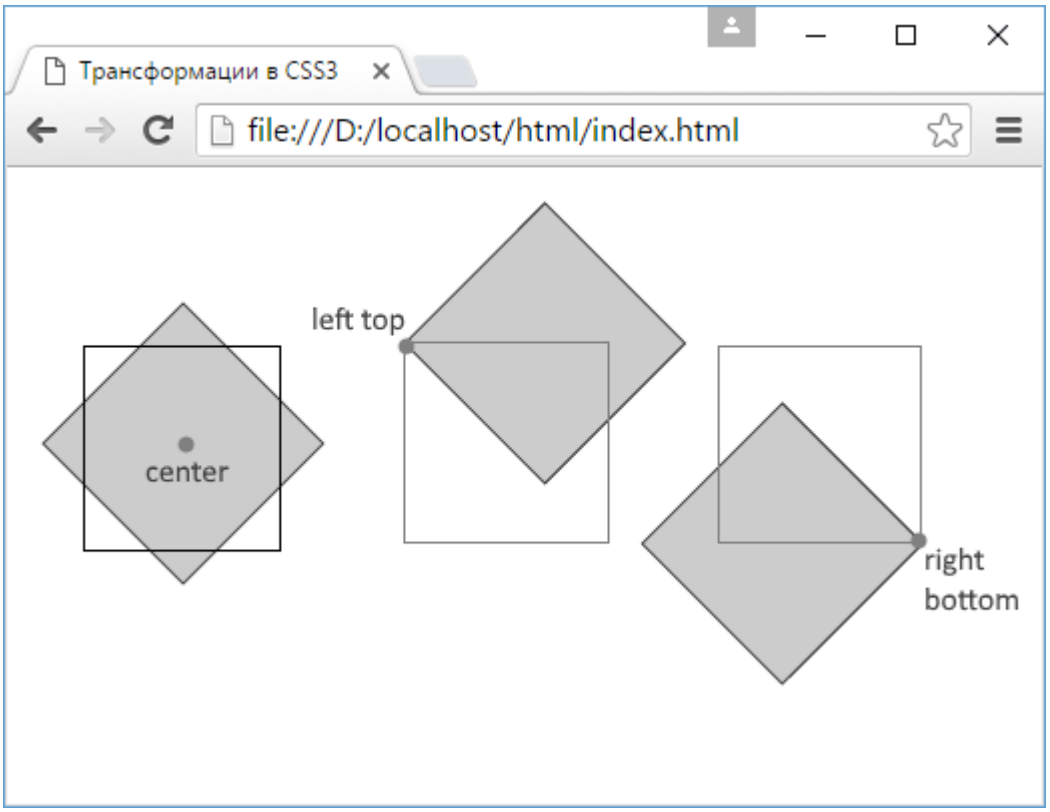
Исходная точка трансформации

По умолчанию при применении трансформаций браузер в качестве точки начала преобразования использует центр элемента. Но с помощью свойства **transform-origin** можно изменить исходную точку. Это свойство в качестве значения принимает значения в пикселях, em и процентах. Также для установки точки можно использовать ключевые слова:

- `left top`: левый верхний угол элемента
- `left bottom`: левый нижний угол элемента
- `right top`: правый верхний угол элемента
- `right bottom`: правый нижний угол элемента

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Трансформации в CSS3</title>
        <style>
            div{
                background-color: #ccc;
                width: 100px;
                height: 100px;
                margin: 80px 30px;
                float: left;
```

```
                box-sizing: border-box;
                border: 1px solid #333;
            }
            .transform1{
                transform: rotate(-45deg);
            }
            .transform2{
                transform-origin: left top;
                transform: rotate(-45deg);
            }
            .transform3{
                transform-origin: right bottom;
                transform: rotate(-45deg);
            }
        </style>
    </head>
    <body>
        <div class="transform1"></div>
        <div class="transform2"></div>
        <div class="transform3"></div>
    </body>
</html>
```



Переходы

Переход (transition) представляет анимацию от одного стиля к другому в течение определенного периода времени.

Для создания перехода необходимы прежде всего два набора свойств CSS: начальный стиль, который будет иметь элемент в начале перехода, и конечный стиль - результат перехода. Так, рассмотрим простейший переход:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Переход в CSS3</title>
        <style>
            div{
                width: 100px;
                height: 100px;
                margin: 40px 30px;
                border: 1px solid #333;

                background-color: #ccc;
                transition-property: background-color;
                transition-duration: 2s;
            }
            div:hover{
                background-color: red;
            }
        </style>
    </head>
    <body>
        <div></div>
    </body>
</html>
```

Итак, здесь анимируется свойство `background-color` элемента div. При наведении указателя мыши на элемент он будет менять цвет с серого на красный. А при увеении указателя мыши с пространства элемента будет возвращаться исходный цвет.

Чтобы указать свойство как анимируемое, его название передается свойству **transition-property**

```
transition-property: background-color;
```

Вообще анимировать можно множество разных свойств, такие как color, background-color, border-color. Полный список свойств CSS, которые поддаются анимации, можно найти по адресу www.w3.org/TR/css3-transitions/#animatable-properties

Далее идет установка времени перехода в секундах с помощью свойства **transition-duration**:

```
transition-duration: 2s;
```

Кроме секунд можно устанавливать значения в миллисекундах, например, 500 миллисекунд:

```
transition-duration: 500ms;
```

И в конце нам надо определить инициатор действия и финальное значение анимируемого свойства background-color. Инициатор представляет действие, которое приводит к изменению одного стиля на другой. В CSS для запуска перехода можно применять псевдоклассы. Например, здесь для создания перехода используется стиль для псевдокласса `:hover`. То есть при наведении указателя мыши на элемент div, будет срабатывать переход.

Кроме псевдокласса `:hover` можно использовать и другие псевдоклассы, например, `:active` (ссылка в нажатом состоянии) или `:focus` (получение элементом фокуса).

Также для запуска перехода можно использовать код JavaScript.

Анимация набора свойств

При необходимости мы можем анимировать сразу несколько свойств CSS. Так, в выше приведенном примере изменим стили следующем образом:

```
div{
    width: 100px;
    height: 100px;
    margin: 40px 30px;
    border: 1px solid #333;
    background-color: #ccc;

    transition-property: background-color, width, height, border-color;
    transition-duration: 2s;
}
div:hover{
    background-color: red;
    width: 120px;
    height: 120px;
    border-color: blue;
}
```

Здесь анимируются сразу четыре свойства. Причем анимация для них всех длится 2 секунды, но мы можем для каждого свойства задать свое время:

```
transition-property: background-color, width, height, border-color;
transition-duration: 2s, 3s, 1s, 2s;
```

Подобно тому как в свойстве `transition-property` через запятую идет перечисление анимируемых свойств, в свойстве `transition-duration` идет перечисление через запятую временных периодов для анимации этих свойств. Причем сопоставление времени определенному свойству идет по позиции, то есть свойство `width` будет анимироваться 3 секунды.

Кроме перечисления через запятую всех анимируемых свойств мы можем просто указать ключевое слово **all**:

```
transition-property: all;
transition-duration: 2s;
```

Теперь будут анимироваться все необходимые свойства, которые меняют значения в стиле для псевдокласса `:hover`.

Функции анимации

Свойства **transition-timing-function** позволяет контролировать скорость хода и выполнение анимации. То есть данное свойство отвечет за то, как и в какие моменты времени анимация будет ускоряться или замедляться.

В качестве значения это свойство может принимать одну из функций:

- **linear**: линейная функция плавности, изменение свойства происходит равномерно по времени
- **ease**: функция плавности, при которой анимация ускоряется к середине и замедляется к концу, предоставляя более естественное изменение
- **ease-in**: функция плавности, при которой происходит только ускорение в начале
- **ease-out**: функция плавности, при которой происходит только ускорение в конце анимации
- **ease-in-out**: функция плавности, при которой анимация ускоряется к середине и замедляется к концу, предоставляя более естественное изменение
- **cubic-bezier**: для анимации применяется кубическая функция Безье

Применим функцию `ease-in-out`:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Переход в CSS3</title>
        <style>
            div{
                width: 100px;
                height: 100px;
                margin: 40px 30px;
                border: 1px solid #333;

                background-color: #ccc;
                transition-property: background-color, width;
                transition-duration: 2s, 10s;
                transition-timing-function: ease-in-out;
            }
            div:hover{
                background-color: red;
                width: 200px;
            }
        </style>
    </head>
    <body>
        <div></div>
    </body>
</html>
```

Для использования кубической Безье в функцию `cubic-bezier` необходимо передать ряд значений:

```
transition-timing-function: cubic-bezier(.5, .6, .24, .18);
```

Задержка перехода

Свойство **transition-delay** позволяет определить задержку перед выполнением перехода:

```
transition-delay: 500ms;
```

Временной период также указывается в секундах (s) или миллисекундах (ms).

Свойство transition

Свойство **transition** представляет сокращенную запись выше рассмотренных свойств. Например, следующее описание свойств:

```
transition-property: background-color;
transition-duration: 3s;
transition-timing-function: ease-in-out;
transition-delay: 500ms;
```

Будет аналогично следующей записи:

```
transition: background-color 3s ease-in-out 500ms;
```

Анимация

Анимация предоставляет большие возможности за изменением стиля элемента. При переходе у нас есть набор свойств с начальными значениями, которые имеет элемент до начала перехода, и конечными значениями, которые устанавливают после завершения перехода. Тогда как при анимации мы можем иметь не только два набора значений - начальные и конечные, но и множество промежуточных наборов значений.

Анимация опирается на последовательню смену ключевых кадров (keyframes). Каждый ключевой кадр определяет один набор значений для анимируемых свойств. И последовательная смена таких ключевых кадров фактически будет представлять анимацию.

По сути переходы применяют ту же модель - так же неявно определяются два ключевых кадра - начальный и конечный, а сам переход представляет переход от начального к конечному ключевому кадру. Единственное отличие анимации в данном случае состоит в том, что для анимации можно определить множество промежуточных ключевых кадров.

В целом объявление ключевого кадра в CSS3 имеет следующую форму:

```
@keyframes название_анимации {
    from {
        /* начальные значения свойств CSS */
    }
    to {
        /* конечные значения свойств CSS */
    }
}
```

После ключевого слова **@keyframes** идет имя анимации. Затем в фигурных скобках определяются как минимум два ключевых кадра. Блок после ключевого слова **from** объявляется начальный ключевой кадр, а после клюевого слова **to** в блоке определяется конечный ключевой кадр. Внутри каждого ключевого кадра определяется одно или несколько свойств CSS, подобно тому, как создается обычный стиль.

Например, определим анимацю для фонового цвета элемента:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Анимация в CSS3</title>
        <style>
            @keyframes backgroundColorAnimation {
                from {
                    background-color: red;
                }
                to {
                    background-color: blue;
                }
            }
            div{
                width: 100px;
                height: 100px;
```

```
                margin: 40px 30px;
                border: 1px solid #333;

                background-color: #ccc;
                animation-name: backgroundColorAnimation;
                animation-duration: 2s;
            }
        </style>
    </head>
    <body>
        <div></div>
    </body>
</html>
```

В данном случае анимация называется `backgroundColorAnimation`. Анимация может иметь произвольное название.

Эта анимация предоставляет переход от красного цвета фона к синему. Причем после завершения анимации будет устанавливться тот цвет, который определен у элемента div.

Чтобы прикрепить анимацию к элементу, у него в стиле применяется свойство **animation-name**. Значение этого свойства - название применяемой анимации.

Также с помощью свойства **animation-duration** необходимо задать время анимации в секундах или миллисекундах. В данном случае время анимации - это 2 секунды.

При подобном определении анимация будет запускаться сразу после загрузки страницы. Однако можно также запускать анимацию по действию пользователя. Например, с помощью определения стиля для псевдокласса `:hover` можно определить запуск анимации при наведении указателя мыши на элемент:

```
@keyframes backgroundColorAnimation {
    from {
        background-color: red;
    }
    to {
        background-color: blue;
    }
}
div{
    width: 100px;
    height: 100px;
    margin: 40px 30px;
    border: 1px solid #333;
    background-color: #ccc;
}
div:hover{

    animation-name: backgroundColorAnimation;
    animation-duration: 2s;
}
```

Множество ключевых кадров

Как уже выше говорилось выше, анимация кроме двух стандартных ключевых кадров позволяет задействовать множество промежуточных. Для определения промежуточного кадра применяется процентное значение анимации, в котором этот кадр должен использоваться:

```
@keyframes backgroundColorAnimation {
    from {
        background-color: red;
    }
    25%{
        background-color: yellow;
    }
    50%{
        background-color: green;
    }
    75%{
        background-color: blue;
    }
    to {
        background-color: violet;
    }
}
```

В данном случае анимация начинается с красного цвета. Через 25% времени анимации цвет меняется на желтый, еще через 25% - на зеленый и так далее.

Также можно в одном ключевом кадре анимировать сразу несколько свойств:

```
@keyframes backgroundColorAnimation {
    from {
        background-color: red;
        opacity: 0.2;
    }
    to {
        background-color: blue;
        opacity: 0.9;
    }
}
```

Также можно определить несколько отдельных анимаций, но применять их вместе:

```
@keyframes backgroundColorAnimation {
    from {
        background-color: red;
    }
    to {
        background-color: blue;
    }
}
@keyframes opacityAnimation {
    from {
        opacity: 0.2;
    }
```

```
    to {
        opacity: 0.9;
    }
}
div{
    width: 100px;
    height: 100px;
    margin: 40px 30px;
    border: 1px solid #333;
    background-color: #ccc;

    animation-name: backgroundColorAnimation, opacityAnimation;
    animation-duration: 2s, 3s;
}
```

В качестве значения свойства `animation-name` через запятую перечисляются анимации, и также через запятую у свойства `animation-duration` задается время этих анимаций. Название анимации и ее время сопоставляются по позиции, то есть анимация opacityAnimation будет длиться 3 секунды.

Завершение анимации

В общем случае после завершения временного интервала, указанного у свойства `animation-duration`, завершается и выполнение анимации. Однако с помощью дополнительных свойств мы можем переопределить это поведение.

Так, свойство **animation-iteration-count** определяет, сколько раз будет повторяться анимация. Например, 3 повтора анимации подряд:

```
animation-iteration-count: 3;
```

Если необходимо, чтобы анимация запускалась бесконечное количество раз, то этому свойству присваивается значение **infinite**:

```
animation-iteration-count: infinite;
```

При повторе анимация будет начинаться снова с начального ключевого кадра. Но с помощью свойства `animation-direction: alternate;` противоположное направление анимации при повторе. То есть она будет начинаться с конечного ключевого кадра, а затем будет переход к начальному кадру:

```
animation-name: backgroundColorAnimation, opacityAnimation;
animation-duration: 2s, 2s;
animation-iteration-count: 3;
animation-direction: alternate;
```

При окончании анимации браузер устанавливает для анимированного элемента стиль, который был бы до применения анимации. Но свойство `animation-fill-mode: forwards` позволяет в качестве окончательного значения анимируемого свойства установить именно то, которое было в последнем кадре:

```
animation-name: backgroundColorAnimation;
animation-duration: 2s;
animation-iteration-count: 3;
```

```
animation-direction: alternate;
animation-fill-mode: forwards;
```

Задержка анимации

С помощью свойства **animation-delay** можно определить время задержки анимации:

```
animation-name: backgroundColorAnimation;
animation-duration: 5s;
animation-delay: 1s;    /* задержка в 1 секунду */
```

Функция плавности анимации

Как и к переходам, к анимации можно применять все те же функции плавности:

- **linear**: линейная функция плавности, изменение свойства происходит равномерно по времени
- **ease**: функция плавности, при которой анимация ускоряется к середине и замедляется к концу, предоставляя более естественное изменение
- **ease-in**: функция плавности, при которой происходит только ускорение в начале
- **ease-out**: функция плавности, при которой происходит только ускорение в начале
- **ease-in-out**: функция плавности, при которой анимация ускоряется к середине и замедляется к концу, предоставляя более естественное изменение
- **cubic-bezier**: для анимации применяется кубическая функция Безье

Для установки функции плавности применяется свойство **animation-timing-function**:

```
@keyframes backgroundColorAnimation {
    from {
        background-color: red;
    }
    to {
        background-color: blue;
    }
}
div{
    width: 100px;
    height: 100px;
    margin: 40px 30px;
    border: 1px solid #333;

    animation-name: backgroundColorAnimation;
    animation-duration: 3s;
    animation-timing-function: ease-in-out;
}
```

Свойство animation

Свойство **animation** является сокращенным способом определения выше рассмотренных свойств:

```
animation: animation-name animation-duration animation-timing-function animation-itera
animation-fill-mode
```

Первые два параметра, которые предоставляют название и время анимации, являются обязательными. Остальные значения не обязательны.

Возьмем следующий набор свойств:

```
animation-name: backgroundColorAnimation;
animation-duration: 5s;
animation-timing-function: ease-in-out;
animation-iteration-count: 3;
animation-direction: alternate;
animation-delay: 1s;
animation-fill-mode: forwards;
```

Этот набор будет эквивалентен следующему определению анимации:

```
animation: backgroundColorAnimation 5s ease-in-out 3 alternate 1s forwards;
```

Создание баннера с анимацией

В качестве примера с анимацией создадим простейший анимированный баннер:

```
<!DOCTYPE html>
<html>
<head>
    <title>HTML-баннер</title>
    <meta charset="utf-8" />
    <style type="text/css">
    @keyframes text1
    {
        10%{opacity: 1;}
        40%{opacity: 0;}
    }
    @keyframes text2
    {
        30%{opacity: 0;}
        60%{opacity:1;}
    }
    @keyframes banner
    {
        10%{background-color: #008978;}
        40%{background-color: #34495e;}
        80%{background-color: green;}
    }
    .banner
    {
        width: 600px;
        height: 120px;
        background-color: #777;
        margin: 0 auto;
        position: relative;
    }
    .text1,.text2
    {
        position: absolute;
```

```
            width: 100%;
            height: 100%;
            line-height: 120px;
            text-align: center;
            font-size: 40px;
            color: #fff;
            opacity: 0;
        }

        .text1
        {
            animation : text1 6s infinite;
        }

        .text2
        {
            animation : text2 6s infinite;
        }

        .animated
        {
            opacity: 0.8;
            position: absolute;
            width: 100%;
            height: 100%;
            background-color: #34495e;
            animation: banner 6s infinite;
        }
        </style>
    </head>
    <body>
        <div class="banner">
            <div class="animated">
                <div class="text1">Только в этом месяце</div>
                <div class="text2">Скидки по 20%</div>
            </div>
        </div>
    </body>
    </html>
```

Здесь одновременно срабатывают три анимации. Анимация "banner" изменяет цвет фона баннера, а анимации text1 и text2 отображают и скрывают текст с помощью настроек прозрачности. Когда первый текст виден, второй не виден и наоборот. Тем самым мы получаем анимацию текста в баннере.

Адаптивный дизайн

Введение в адаптивный дизайн

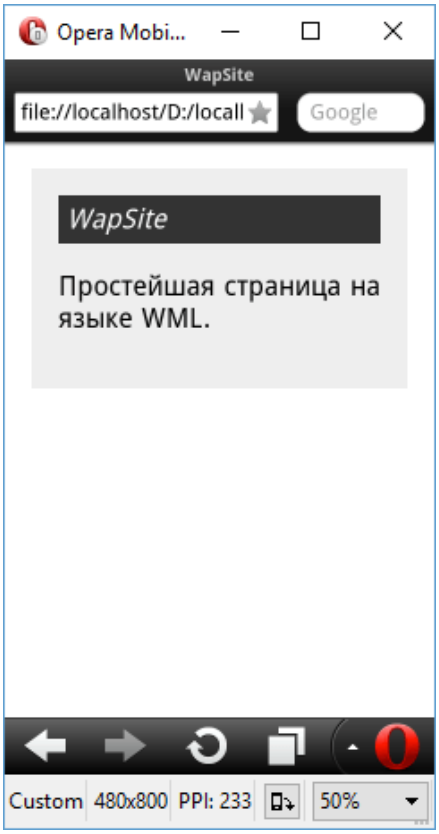
Сейчас все большее распространение находят различные гаджеты - смартфоны, планшеты, "умные часы" и другие устройства, которые позволяют выходить в интернет, просматривать содержимое сайтов. По некоторым оценкам уже чуть ли не половина интернет-траффика генерируется подобными гаджетами, разрешение экрана которых отличается от разрешения экранов стандартных компьютеров. Подобное распространение гаджетов несет новые возможности по развитию веб-сайтов, привлечения новых посетителей, продвижению информационных услуг и т.д. Но вместе с тем появляются и новые проблемы.

Главная проблема заключается в том, что стандартная веб-страница будет по разному выглядеть для разных устройств с разным разрешением экрана. Первоначальным решением данной проблемы было создание специальных версий для мобильных устройств.

На заре распространения мобильных телефонов пользователи могли через телефон по протоколу WAP получать доступ к специальным wap-сайтам, которые были написаны на языке wml - языке на основе xml, похожем на html. К примеру, простейшая веб-страница на этом языке разметке могла иметь следующий код:

```
1  <?xml version="1.0"?>
2  <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "
3  http://www.wapforum.org/DTD/wml_1.1.xml" >
4  <wml>
5    <card id="main" title="WapSite">
6      <p mode="wrap">Простейшая страница на языке WML.</p>
7    </card>
   </wml>
```

Визуально она выглядела так:



Однако развитие самих гаджетов, их возможностей привело к тому, что сейчас мобильные телефоны представляют куда большие возможности по получению и отображению содержимого сайтов, а в написании подобных сайтов используется те же HTML5 и CSS3, что и для обычных сайтов.

Кроме того, появление все большего количества разнообразных устройств привело к тому, что веб-страницы необходимо подстраивать не только под небольшие экраны смартфонов или планшетов, но и под огромные экраны полноформатных широкоэкранных телевизоров или гигантских планшетов типа Surface Hub, которые также могут иметь доступ к интернету.

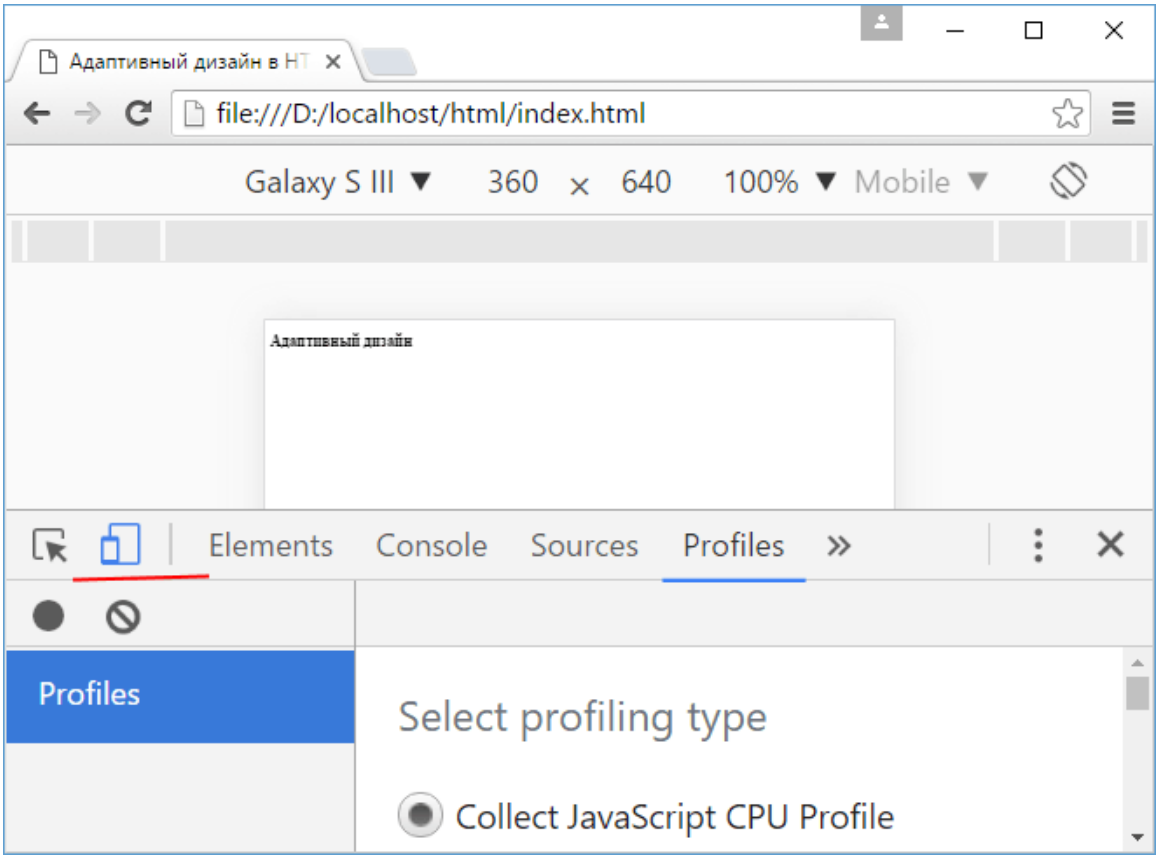
И для решения проблемы совместимости веб-страниц с самыми различными разрешениями самых различных устройств возникла концепция адаптивного дизайна. Ее суть заключается в том, чтобы должным образом масштабировать элементы веб-страницы в зависимости от ширины экрана.

Хотя нередко до сих пор можно встретить ситуацию, когда для сайта создается отдельная мобильная версия, часто с префиксом **m**, например, *m.vk.com*. Однако концепция адаптивного становится все более распространенной и доминирующей.

Тестирование адаптивного дизайна

При разработке адаптивных веб-страниц мы можем столкнуться с трудностями тестирования, так как, как правило, разработка идет на обычных компьютерах. Но к счастью многие совеременные браузеры позволяют нам эмулировать запуск веб-страницы на том или ином устройстве с различной шириной экрана.

Например, в Google Chrome надо перейти в меню **Дополнительные инструменты -> Инструменты разработчика**. После октрытия панели разработчика в начале меню самой панели можно нажать на иконку мобильного телефона, и после этого можно будет эмулировать отображение страницы на различных устройствах - от небольших телефонов до широкоформатных телевизоров:



В данном случае отображается веб-страница как она бы выглядела на устройстве Samsung Galaxy S III. Но при желании можно выбрать другое устройство, либо даже создать эмуляцию какого-то нового устройства, которого нет во встроенном списке.

Подобные инструменты есть и в других современных веб-браузерах. Например, в Mozilla Firefox для их открытия надо перейти в меню **Разработка -> Адаптивный дизайн**

Еще одно решение заключается в использовании эмуляторы мобильных устройств. Небольшой список подобных эмуляторов можно найти по следующему адресу: http://www.mobilexweb.com/emulators.

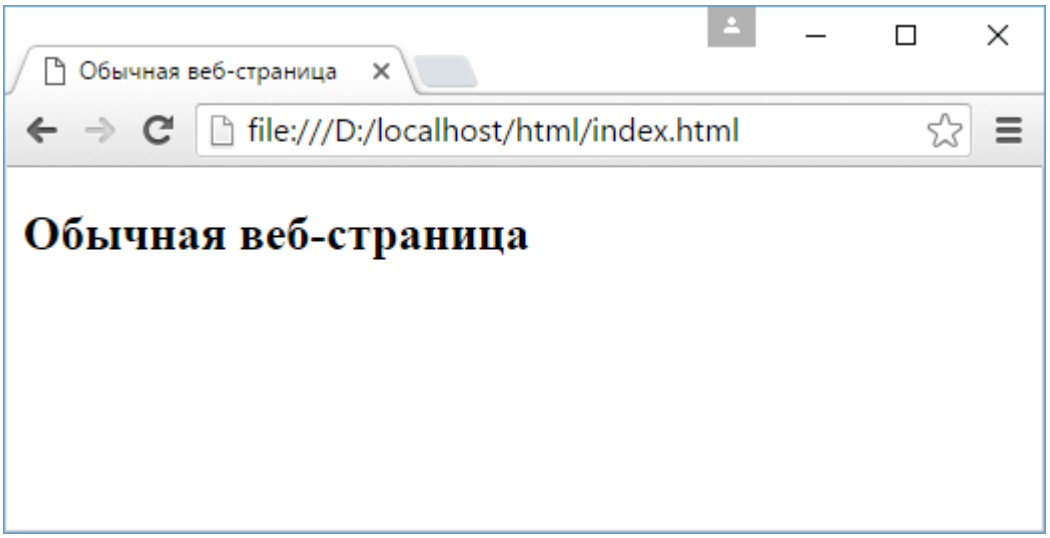
Наиболее популярным является эмулятор браузера Opera Mobile, который можно найти по адресу http://www.opera.com/ru/developer/mobile-emulator.

Метатег Viewport

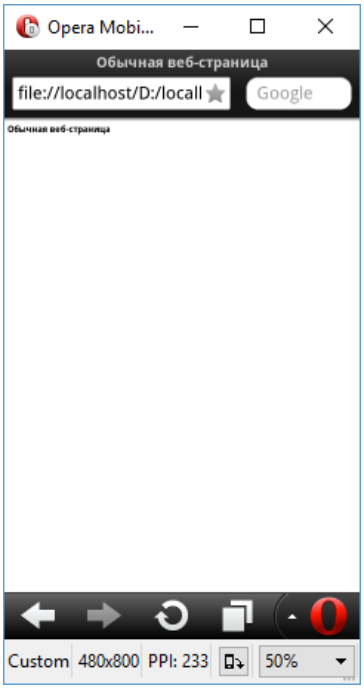
Прежде всего рассмотрим один из ключевых моментов применения адаптивного дизайна - метатег **viewport** (по сути с этого тега и начивается адаптиный дизайн). Пусть для начала у нас есть следующая веб-страница:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Обычная веб-страница</title>
    </head>
    <body>
        <h2>Обычная веб-страница</h2>
    </body>
</html>
```

Это стандартная веб-страница, которая в обычном браузере будет выглядеть следующим образом:



Однако если мы запустим ту же самую веб-страницу в эмуляторе мобильного устройства или на реальном мобильном устройстве, то мы с трудом сможем прочитать, что же на ней написано:



Применяя масштабирование, пользователь может наконец-то увидеть, что же там все таки написано. Однако это не очень удобно. При этом веб-страница имеет много пустого места, что не очень красиво.

Почему так происходит? Дело в том, что каждый мобильный браузер задает странице некоторые начальные размеры и потом пытается приспособить под размеры экрана текущего мобильного устройства.

Вся видимая область на экране браузера описывается понятием **Viewport**. По сути viewport представляет область, в которую веб-браузер пытается "впихнуть" веб-страницу. Например, браузер Safari на iPone и iPod определяет ширину viewport по умолчанию равной 980 пикселям. То есть, получив страницу и вписав в viewport с шириной 980 пикселей, браузер сжимает ее до размеров мобильного устройства. Например, если ширина экрана смартфона составляет 320 пикселей, то до этого размера потом будет сжата страница. И ко всем элементам страницы будет применен коэффициент масштабирования, равный 320/980.

Почему в данном случае используется именно 980 пикселей, а, скажем, не реальный размер экрана? Все дело в том, что по умолчанию браузер считает, что каждая веб-страница предназначена для десктопов. А обычной шириной десктопного сайта можно считать 980 пикселей.

При этом для каждого браузера устанавливается своя ширина области viewport, необязательно 980 пикселей. Другие браузеры могут поддерживать в качестве начальной ширины другие значения. Но они также будут выполнять масштабирование.

Чтобы избежать подобной не очень приятной картины, следует использовать метатег viewport. Он имеет следующее определение:

```
1  <meta name="viewport" content="параметры_метатега">
```

В атрибуте content мета-тега мы можем определить следующие параметры:

Параметр	Значения	Описание
width	Принимает целочисленное значение в пикселях или значение `device-width`	Устанавливает ширину области viewport
height	Принимает целочисленное значение в пикселях или значение `device-height`	Устанавливает высоту области viewport
initial-scale	Число с плавающей точкой от 0.1 и выше	Задает коэффициент масштабирования начального размера viewport. Значение 1.0 задает отсутствие масштабирования
user-scalable	no/yes	Указывает, может ли пользователь с помощью жестов масштабировать страницу
minimum-scale	Число с плавающей точкой от 0.1 и выше	Задает минимальный масштаб размера viewport. Значение 1.0 задает отсутствие масштабирования
maximum-scale	Число с плавающей точкой от 0.1 и выше	Задает максимальный масштаб размера viewport. Значение 1.0 задает отсутствие масштабирования

Теперь изменим предыдущий пример веб-страницу, использовав метатег:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width">
        <title>Обычная веб-страница</title>
    </head>
    <body>
        <h2>Обычная веб-страница</h2>
    </body>
</html>
```

Веб-страничка определенно выглядит лучше благодаря использованию метатега viewport. Используя параметр `width=device-width` мы говорим веб-браузеру, что в качестве начальной ширины области viewport надо считать не 980 пикселей или какое-то другое число, а непосредственную ширину экрана устройства. Поэтому затем веб-браузер не будет проводить никакого масштабирования, так как у нас ширина viewport и ширина одинаковы.

Мы также можем использовать другие параметры, например, запретить пользователю масштабировать размеры страницы:

```
<meta name="viewport" content="width=device-width, maximum-scale=1.0,
minimum-scale=1.0">
```

Media Query в CSS

Другим важным элементом в построении адаптивого дизайна являются правила **Media Query**, которые поволяют определить стиль в зависимости от размеров браузера пользователя.

Например, некоторые элементы могут

В CSS2 уже было решение в виде правила `media`, которое позволяет указать устройство, для которого используется данный стиль:

```
<html>
 <head>
  <title>Адаптивная веб-страница</title>
  <link media="handheld" rel="stylesheet" href="mobile.css">
  <link media="screen" rel="stylesheet" href="desktop.css">
 </head>
 <body>
 ......................
```

Правило `media="handheld"` указывает, что стили в mobile.css будут применяться к мобильным устройствам, в то время как правило `media="screen"` применяется к десктопным браузерам.

Однако многие современные мобильные браузеры по умолчанию считают, что страница предназначена для десктопов, поэтому в любом случае применяет правило `media="screen"`. Поэтому на подобное решение не стоит полагаться.

Для решения этой проблемы в CSS3 были введен механизм **CSS3 Media Query**. Например, чтобы применить стиль только к мобильным устройствам мы можем написать так:

```
<html>
 <head>
  <title>Адаптивная веб-страница</title>
  <meta name="viewport" content="width=device-width">
  <link rel="stylesheet" type="text/css" href="desctop.css" />
  <link rel="stylesheet" type="text/css" media="(max-device-width:480px)"
   href="mobile.css" />
 </head>
 <body>
 ..............................
```

Значение атрибута media `(max-device-width:480px)` говорит нам о том, что стили из файла mobile.css будут применяться к тем устройствам, максимальная ширина экрана которых составляет 480 пикселей - то есть фактически это и есть мобильные устройства.

С помощью ключевого слова **and** можно комбинировать условия, например:

```
<link rel="stylesheet" type="text/css" media="(min-width:481px) and (max-width:768px
" href="mobile.css" />
```

Данный стиль будет применяться, если ширина браузера находится в диапазоне от 481 до 768 пикселей.

С помощью директивый **@import** можно определить один css-файл и импортировать в него стили для определенных устройств:

```
@import url(desctop.css);
@import url(tablet.css) (min-device-width:481px) and (max-device-width:768);
@import url(mobile.css) (max-device-width:480px);
```

Также можно не разлелять стили по файлам, а использовать правила CSS3 Media Query в одном файле css:

```
 body {
     background-color: red;
 }
 /*Далее остальные стили*/
 @media (max-device-width:480px){
     body {
         background-color: blue;
     }
     /*Далее остальные стили*/
 }
```

Применяемые функции в CSS3 Media Query:

- **aspect-ratio**: отношение ширины к высоте области отображения (браузера)
- **device-aspect-ratio**: отношение ширины к высоте экрана устройства
- **max-width/min-width и max-height/min-height**: максимальная и минимальная ширина и высота области отображения (браузера)
- **max-device-width/min-device-width и max-device-height/min-device-height**: максимальная и минимальная ширина и высота экрана мобильного устройства
- **orientation**: ориентация (портретная или альбомная)

Например, мы можем задать разные стили для разных ориентаций мобильных устройств:

```
/*Стили для портретной ориентации*/
@media only screen and (orientation: portrait){

}
/*Стили альбомной ориентации*/
@media only screen and (orientation: landscape){

}
```

Таким образом, мы меняем лишь определение стилей в зависимости от устройства, а сами стили css по сути остаются теми же, что мы используем для создания обычных сайтов.

Как правило, при определении стилей предпочтение отдается стилям для самых малых экранов - так называемый подход **Mobile First**, хотя это необязательно. Например, определим следующую веб-страницу:

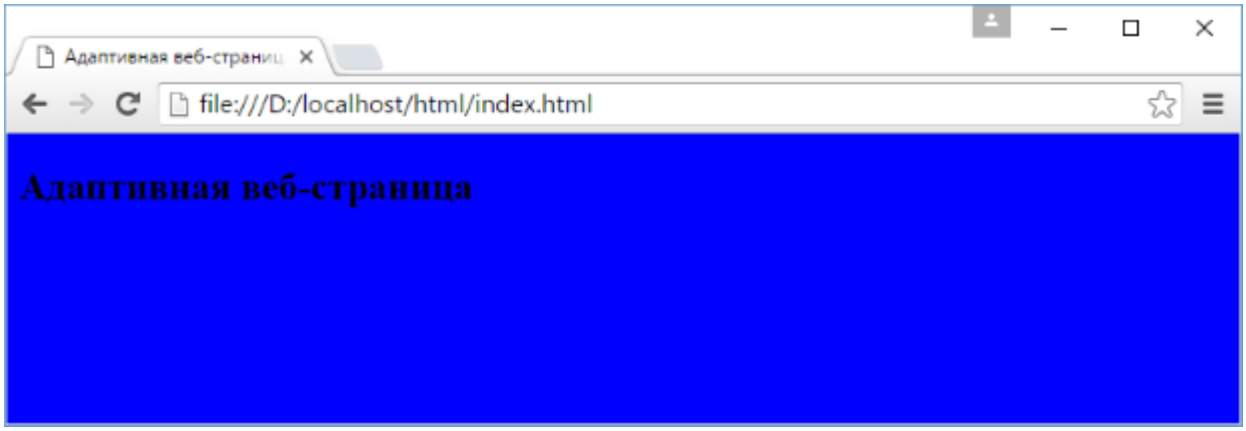
```
 <!DOCTYPE html>
 <html>
     <head>
         <meta charset="utf-8">
         <meta name="viewport" content="width=device-width">
```

```
6           <title>Адаптивная веб-страница</title>
7
8           <style>
9           body {
10              background-color: red;
11          }
12          /* для планшетов и фаблетов */
13          @media (min-width: 481px) and (max-width:768px) {
14              body {
15                  background-color: green;
16              }
17          }
18          /* для декстопов */
19          @media (min-width:769px) {
20              body {
21                  background-color: blue;
22              }
23          }
24  </style>
25      </head>
26      <body>
27          <h2>Адаптивная веб-страница</h2>
28      </body>
29  </html>
```

Сначала идут общие стили, которые актуальны прежде всего для мобильных устройств с небольшими экранами. Затем идут стили для устройств с экранами средней ширины: фаблеты, планшеты. И далее идут стили для десктопов.

И например, на эмуляторе Opera Mobile при эмуляции устройства с шириной в 480 пикселей страница приобретет красный цвет:

А в браузере обычного компьютера страница будет иметь синий цвет, как и определено в стилях:



Мультимедиа

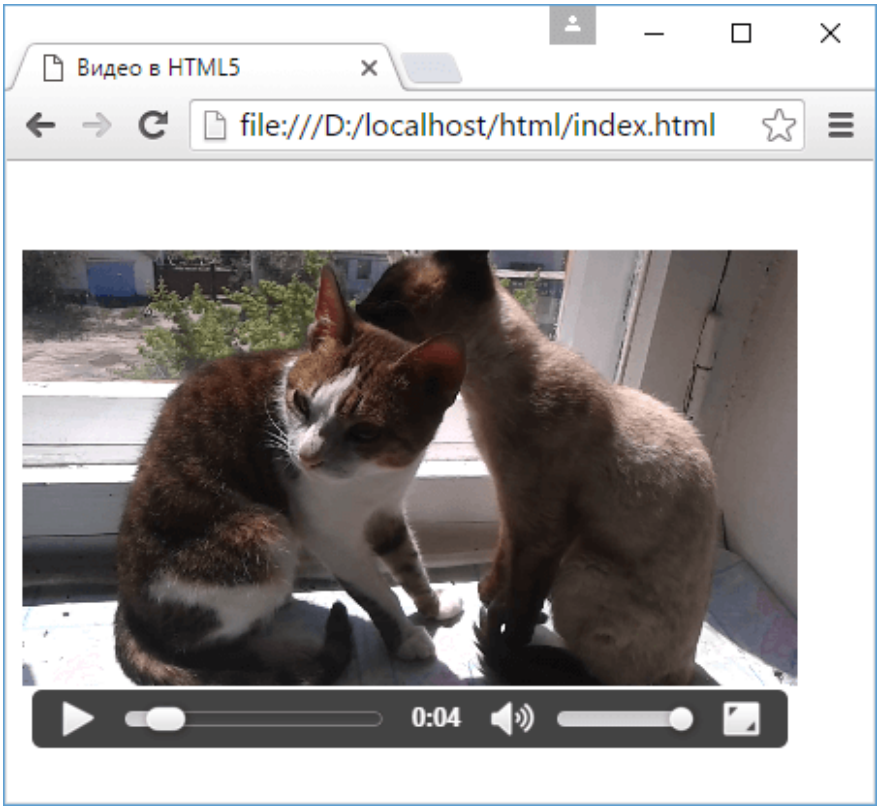
Видео

Для воспроизведения видео в HTML5 используется элемент **video**. Чтобы настроить данный элемент, мы можем использовать следующие его атрибуты:

- `src`: источник видео, это может быть какой-либо видеофайл
- `width`: ширина элемента
- `height`: высота элемента
- `controls`: добавляет элементы управления воспроизведением
- `autoplay`: устанавливает автовоспроизведение
- `loop`: задает повторение видео
- `muted`: отключает звук по умолчанию

Хотя мы можем установить ширину и высоту, но они не окажут никакого влияния на аспектное отношения ширины и высоты самого видео. Например, если видео имеет формат 375×240, то, к примеру, при настройках `width="375" height="280"` видео будет центрироваться на 280-пиксельном пространстве в HTML. Что позволяет избавить видео от искажений, которые были бы при растягивании.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Видео в HTML5</title>
    </head>
    <body>
        <video src="cats.mp4" width="400" height="300" controls ></video>
    </body>
</html>
```



Применим атрибуты `autoplay` и `loop`:

```
<video src="cats.mp4" width="400" height="300" controls autoplay loop ></video>
```

Теперь видео будет автоматически проигрываться бесконечное число раз.

Если при воспроизведении надо отключить звук, то мы можем воспользоваться атрибутом `muted`:

```
<video src="cats.mp4" width="400" height="300" controls muted ></video>
```

Атрибут preload

Еще один атрибут - **preload** призван управлять загрузкой видео. Он принимает следующие значения:

- `auto`: видео и связанные с ним метаданные будут загружаться до того, как видео начнет воспроизводиться.
- `none`: видео не будет загружаться в фоне, пока пользователь не нажмет на кнопку начала проигрывания
- `metadata`: в фоне до воспроизведения будут загружаться только метаданные (данные о формате, длительности и т.д), само видео не загружается

```
<video src="cats.mp4" width="400" height="300" controls preload="auto"></video>
```

Атрибут poster

Атрибут poster позволяет установить изображение, которое будет отображаться до запуска видео. Этому атрибуту в качестве значения передается путь к изображению:

```
<video src="cats.mp4" width="400" height="300" controls poster="mycat.jpg"></video>
```

Поддержка форматов видео

Главной проблемой при использовании элемента `video` является поддержка различными веб-браузерами определенных форматов. С помощью вложенных элементов **source** можно задать несколько источников видео, один из которых будет использоваться:

```
<video width="400" height="300" controls>
    <source src="cats.mp4" type="video/mp4">
    <source src="cats.webm" type="video/webm">
    <source src="cats.ogv" type="video/ogg">
</video>
```

Элемент `source` использует два атрибута для установки источника видео:

- `src`: путь к видеофайлу
- `type`: тип видео (MIME-тип)

Если браузер не поддерживает первый тип видео, то он пытается загрузить второй видеофайл. Если же и тип второго видеофайла не поддерживается, то браузер обращается к третьему видеофайлу.

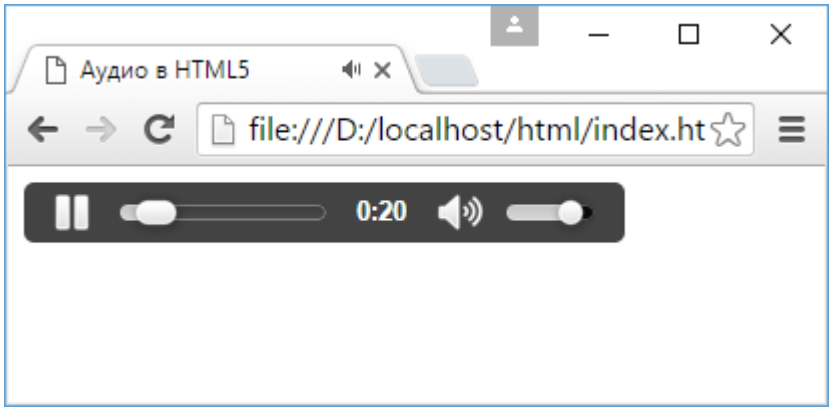
Аудио

Для воспроизведения звука без видео в HTML5 применяется элемент **audio**. Он во многом похож на элемент video. Для настройки элемента audio мы можем использовать следующие его атрибуты:

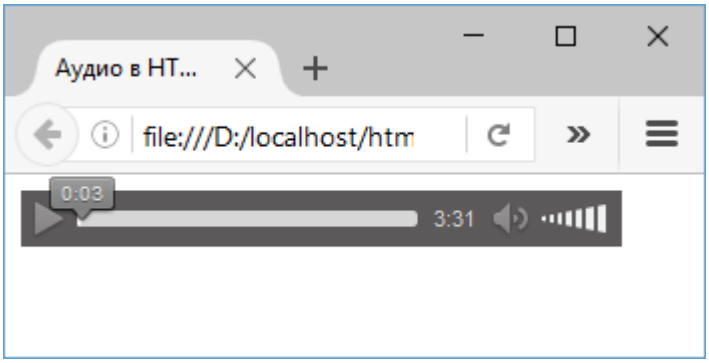
- `src`: путь к аудиофайлу
- `controls`: добавляет элементы управления воспроизведением
- `autoplay`: устанавливает автовоспроизведение
- `loop`: задает повторение аудиофайла
- `muted`: отключает звук по умолчанию
- `preload`: устанавливает режим загрузки файла

Действие всех этих атрибутов будет аналогично их действию в элементе video.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Аудио в HTML5</title>
    </head>
    <body>
        <audio src="mobile_phone.mp3" controls></audio>
    </body>
</html>
```



Опять же в зависимости от браузера внешний вид элементов управления может отличаться. Выше приведен пример для Google Chrome. А в Firefox они будут выглядеть немного иначе:



Поддержка форматов аудио

Ключевым моментом для работы с аудио является поддержка браузером тех или иных форматов. На данный момент подавляющее большинство браузеров поддерживают mp3. Однако если у нас есть неуверенность, что наше аудио в определенном формате будет поддерживаться браузером пользователя, то мы можем использовать вложенный элемент `source` и указать аудио в иных форматах:

```
<audio width="400" height="300" controls>
    <source src="mobile_phone.mp3" type="audio/mpeg">
    <source src="mobile_phone.m4a" type="audio/aac">
    <source src="mobile_phone.ogg" type="audio/ogg">
</audio>
```

Как и в случае с элементом `video`, здесь у элемента source устанавливается атрибут `src` с ссылкой на файл и атрибут `type` - тип файла.

Media API. Управление видео из JavaScript

Вместе с новыми элементами audio и video в HTML5 был добавлен новый API в JavaScript для управления этими элементами. С помощью кода JavaScript мы можем получить элементы video и audio (как и любой другой элемент) и использовать их свойства. В JavaScript эти элементы представлены объектом **HTMLMediaElement**, который с помощью свойств, методов и событий позволяет управлять воспроизведением аудио и видео. Отметим наиболее важные свойства, которые могут нам пригодиться для настройки этих элементов:

- **playbackRate**: устанавливает скорость воспроизведения. По умолчанию равно 1
- **src**: возвращает название воспроизводимого ресурса, если он установлен в коде html элемента
- **duration**: возвращает длительность файла в секундах
- **buffered**: возвращает длительность той части файла, которая уже буферизирована и готова к воспроизведению
- **controls**: устанавливает или возвращает наличие атрибута `controls`. Если он установлен, возвращается `true`, иначе возвращает `false`
- **loop**: устанавливает или возвращает наличие атрибута `loop`. Если он установлен, возвращается `true`, иначе возвращает `false`
- **muted**: устанавливает или возвращает наличие атрибута `muted`
- **preload**: устанавливает или возвращает наличие атрибута `preload`
- **volume**: устанавливает или возвращает уровень звука от 0.0 до 1.0
- **currentTime**: возвращает текущее время воспроизведения

Отдельно для элемента `video` мы можем использовать ряд дополнительных свойств:

- **poster**: устанавливает или возвращает атрибут `poster`
- **height**: устанавливает или возвращает атрибут `height`
- **width**: устанавливает или возвращает атрибут `width`
- **videoWidth, videoHeight**: для элемента video возвращают ширину и высоту видео

Следует также отметить два метода, с помощью которых мы можем управлять воспроизведением:

- **play()**: начинает воспроизведение
- **pause()**: приостанавливает воспроизведение

Основные события элементов video и audio:

- **canplaythrough**: это событие срабатывает после загрузки страницы, если браузер определит, что он может воспроизводить это видео/аудио
- **pause**: событие срабатывает, когда воспроизведение мультимедиа приостанавливается, и оно переводится в состояние "paused"
- **play**: событие срабатывает, когда начинается воспроизведение файла
- **volumechange**: срабатывает при изменении уровня звука мультимедиа
- **ended**: срабатывает при окончании воспроизведения
- **timeupdate**: срабатывает при изменении времени воспроизведения
- **error**: генерируется при возникновении ошибки
- **loadeddata**: срабатывает, когда будет загружен первый фрейм видеофайла
- **loadedmetadata**: срабатывает после загрузки метаданных мультимедиа (длительность воспроизведения, размеры видео и т.д.)
- **seeking**: срабатывает, когда пользователь начинает перемещать курсор по шкале воспроизведения для перемещения к новому месту аудио- или видеофайла

- **seeked**: срабатывает, когда пользователь завершил перемещение к новому месту на шкале воспроизведения

Теперь используем некоторые из этих свойств, событий и методов для управления элементом video:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Аудио в HTML5</title>
        <style>
            .hidden{
                display:none;
            }
            #playBtn {
                border: solid 1px #333;
                padding: 5px;
                cursor: pointer;
            }

        </style>
    </head>
    <body>
        <video width="400" height="300">
            <source src="cats.mp4" type="video/mp4">
            <source src="cats.webm" type="video/webm">
            <source src="cats.ogv" type="video/ogg">
        </video>
        <div id="controls" class="hidden">
            <a id="playBtn">Play</a>
            <span id="timer">00:00</span>
            <input type="range" step="0.1" min="0" max="1" value="0" id="volume" />
        </div>
        <script>
        // получаем все элементы
        var videoEl = document.getElementsByTagName('video')[0],
            playBtn = document.getElementById('playBtn'),
            vidControls = document.getElementById('controls'),
            volumeControl = document.getElementById('volume'),
            timePicker = document.getElementById('timer');

        // если браузер может воспроизводить видео удаляем класс
        videoEl.addEventListener('canplaythrough', function () {
            vidControls.classList.remove('hidden');
            videoEl.volume = volumeControl.value;
        }, false);
        // запускам или останавливаем воспроизведение
        playBtn.addEventListener('click', function () {
            if (videoEl.paused) {
                videoEl.play();
            } else {
                videoEl.pause();
            }
```

```
49          }, false);
50
51          videoEl.addEventListener('play', function () {
52
53              playBtn.innerText = "Pause";
54          }, false);
55
56          videoEl.addEventListener('pause', function () {
57
58              playBtn.innerText = "Play";
59          }, false);
60
61          volumeControl.addEventListener('input', function () {
62
63              videoEl.volume = volumeControl.value;
64          }, false);
65
66          videoEl.addEventListener('ended', function () {
67              videoEl.currentTime = 0;
68          }, false);
69
70          videoEl.addEventListener('timeupdate', function () {
71              timePicker.innerHTML = secondsToTime(videoEl.currentTime);
72          }, false);
73
74          // рассчет отображаемого времени
75          function secondsToTime(time){
76
77              var h = Math.floor(time / (60 * 60)),
78                  dm = time % (60 * 60),
79                  m = Math.floor(dm / 60),
80                  ds = dm % 60,
81                  s = Math.ceil(ds);
82              if (s === 60) {
83                  s = 0;
84                  m = m + 1;
85              }
86              if (s < 10) {
87                  s = '0' + s;
88              }
89              if (m === 60) {
90                  m = 0;
91                  h = h + 1;
92              }
93              if (m < 10) {
94                  m = '0' + m;
95              }
96              if (h === 0) {
97                  fulltime = m + ':' + s;
98              } else {
99                  fulltime = h + ':' + m + ':' + s;
100             }
101             return fulltime;
102         }
103         </script>
```

```
104         </body>
105     </html>
```

Вначале в коде JavaScript мы получаем все элементы. Затем, если браузер поддерживает видео и может его воспроизвести, то обрабатываем событие `canplaythrough`, устанавливая уровень звука и удаляя класс hidden:

```
1   videoEl.addEventListener('canplaythrough', function () {
2       vidControls.classList.remove('hidden');
3       videoEl.volume = volumeControl.value;
4   }, false);
```

Чтобы запустить воспроизведение, нам надо обработать нажатие ссылки Play:

```
1   playBtn.addEventListener('click', function () {
2       if (videoEl.paused) {  // если видео остановлено, запускаем
3           videoEl.play();
4       } else {
5           videoEl.pause();
6       }
7   }, false);
```

Обрабатывая события запуска и остановки воспроизведения, мы можем изменять надпись на ссылке:

```
1   videoEl.addEventListener('play', function () {
2
3       playBtn.innerText = "Pause";
4   }, false);
5
6   videoEl.addEventListener('pause', function () {
7
8       playBtn.innerText = "Play";
9   }, false);
```

Обрабатывая событие `input`, которое возникает при изменении значения ползунка, мы можем синхронизировать изменение ползунка и громкость видео:

```
1   volumeControl.addEventListener('input', function () {
2
3       videoEl.volume = volumeControl.value;
4   }, false);
```

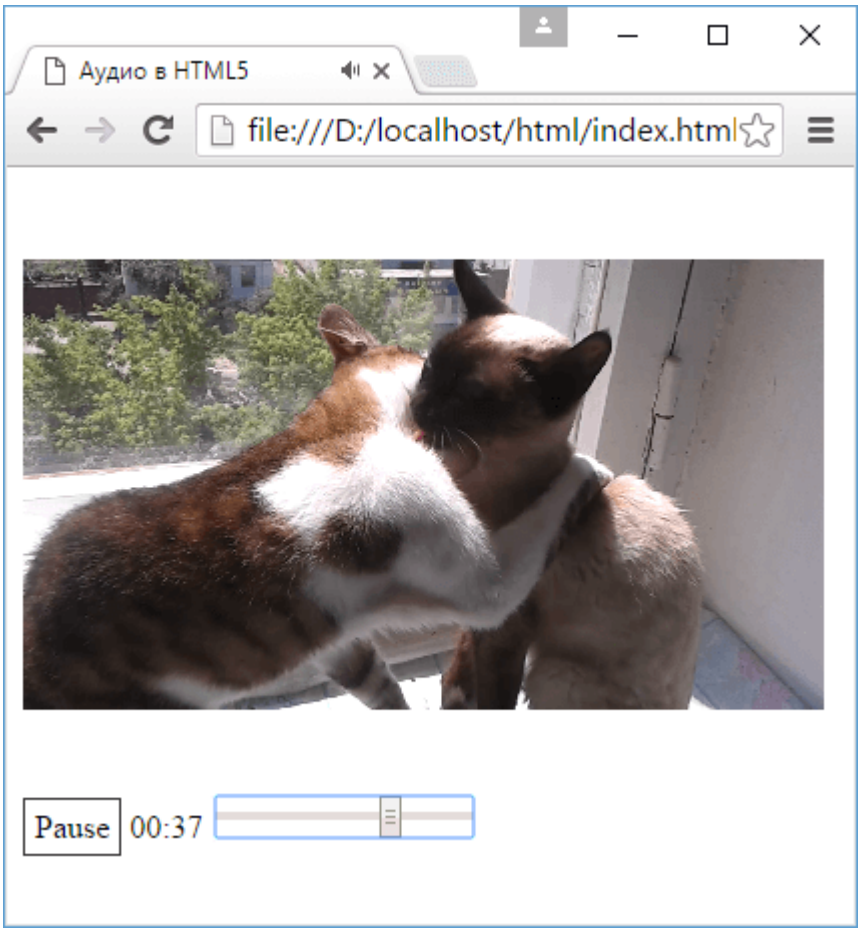
Обработка события `ended` позволит сбросить время воспроизведения:

```
1   videoEl.addEventListener('ended', function () {
2
3       videoEl.currentTime = 0;
4   }, false);
```

А обработчик события `timeupdate` позволит динамически изменять показатель времени воспроизведения:

```
videoEl.addEventListener('timeupdate', function () {
    timePicker.innerHTML = secondsToTime(videoEl.currentTime);
}, false);
```

Для форматирования строки времени применяется вспомогательная функция `secondsToTime`. В итоге мы получим следующие элементы воспроизведения:



Ну и подобным образом можно добавить другие элементы, например, шкалу воспроизведения, какие-то другие кнопки.