

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

А.М. ГОСТИН, А.Н. САПРЫКИН

ИНТЕРНЕТ-ТЕХНОЛОГИИ

Часть 1

Учебное пособие

Рязань 2016

УДК 004.43

Интернет-технологии. Часть 1: учеб. пособие / А.М. Гостин, А.Н. Сапрыкин; Рязан. гос. радиотехн. ун-т. Рязань, 2016. 64 с.

Приведены основные сведения о языках HTML 5 и CSS 3. Рассмотрены основные принципы создания содержимого и представления Web-страниц.

Предназначено для бакалавров и магистров очной, очно-заочной и заочной форм обучения направления 09.00.00 – "Информатика и вычислительная техника".

Табл. 1. Ил. 25. Библиогр.: 5 назв.

Язык HTML, HTML-теги, веб-страница, гипертекст, каскадные таблицы стилей, CSS

Печатается по решению редакционно-издательского совета Рязанского государственного радиотехнического университета.

Рецензент: кафедра САПР вычислительных средств Рязанского государственного радиотехнического университета (зав. кафедрой В.П. Корячко)

Г о с т и н Алексей Михайлович
С а п р ы к и н Алексей Николаевич

Интернет-технологии

Редактор Р.К. Мангутова

Корректор С.В. Макушина

Подписано в печать 30.05.16. Формат бумаги 60х84 1/16.

Бумага писчая. Печать трафаретная. Усл. печ. л. 4,0.

Тираж 20 экз. Заказ .

Рязанский государственный радиотехнический университет.

390005, Рязань, ул. Гагарина, 59/1.

Редакционно-издательский центр РГРТУ.

©Рязанский государственный
радиотехнический университет, 2016

1. Обзор стандарта HTML 5

1.1. Языки гипертекстовой разметки

SGML (*Standard Generalized Markup Language* – стандартный обобщённый язык разметки) – метаязык, на котором можно определять язык разметки для документов.

От SGML произошли языки разметки HTML и XML. HTML – это приложение SGML, а XML – это подмножество SGML, разработанное для упрощения процесса машинного разбора документа.

HTML (*HyperText Markup Language* – язык гипертекстовой разметки) – стандартный язык разметки документов в сети Интернет. Большинство веб-страниц содержат описание разметки на языке HTML. Язык HTML интерпретируется браузерами, а полученный в результате интерпретации форматированный текст отображается на экране монитора компьютера или мобильного устройства.

В настоящее время W3C принята версия HTML 5. Стандарт был официально рекомендован к применению W3C в 2014 году, но еще с 2013 года всеми браузерами оперативно осуществлялась поддержка, а разработчиками – использование рабочего стандарта HTML 5. Целью разработки HTML 5 является улучшение поддержки мультимедиа-технологий, сохранение обратной совместимости, удобочитаемости кода для человека и простоты анализа HTML страниц для парсеров.

В стандарте HTML 5 вводится понятие семантической верстки, удобной для более эффективной последующей стилизации страницы средствами CSS, а также для индексации страниц поисковыми системами. В дополнение к этому HTML 5 устанавливает API, который может быть использован совместно с языком JavaScript, убраны устаревшие теги, добавлен элемент холст с методами рисования 2D, контроль над проигрыванием медиафайлов, управление хранилищем веб-данных, реализация технологии *drag-and-drop* применительно к элементам страницы DOM, регистрация обработчиков MIME типов, возможность применения микроразметки и др.

С принятием стандарта HTML 5 наконец появилась возможность полного разделения содержания, представления и поведения веб-страниц. Эта возможность получила реализацию в новой концепции разработки Web 2.0.

XML (*eXtensible Markup Language*) – расширяемый язык разметки, является подмножеством SGML, рекомендован W3C. Спецификация XML описывает XML-документы и частично

описывает поведение XML-процессоров – программ, читающих XML-документы и обеспечивающих доступ к их содержимому. XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком, с подчёркиванием нацеленности на использование в Интернете.

Пример корректного XML документа:

```
<?xml version="1.0" encoding="utf-8"?>
<book>
<title>Введение в Perl</title>
<authors>Рэндал Шварц, Том Кристианен</authors>
  <publisher>BNV Group</publisher>
  <year>1999</year>
  <content>Содержание</content>
</book>
```

В настоящее время XML широко используется для межсетевого обмена данными в веб-сервисах, в протоколе SOAP, в качестве содержания документов открытого формата ODF, для хранения структурированных данных в файловых хранилищах, в системных конфигурационных файлах и др.

Синтаксис HTML

Документ на языке HTML представляет собой набор элементов, заключенных в угловые скобки, – тегов. Теги могут быть одинарными, не имеющими содержимого, и парными (контейнерными), имеющими содержимое. Перед именем завершающего контейнерного тега ставится символ / (слеш).

Пример одиночного тега:

```
<br>
```

Пример контейнерного тега:

```
<p>Загадки разума</p>
```

Контейнерные теги могут включать в себя другие вложенные теги:

```
<p><strong>Что в имени тебе моем?</strong></p>
```

Каждый тег может содержать атрибуты и через символ = (равенство) их значения, заключенные в кавычки:

```
<p class="pod">Александр Великий</p>
```

Атрибуты являются свойствами, предоставляющими дополнительную информацию о теге.

Имена тегов и их атрибутов специфицированы в стандарте HTML и соответствуют общим правилам идентификаторов: имена содержат латинские буквы, цифры, символы тире и подчеркивания; каждое имя начинается с латинской буквы. Прописные и заглавные буквы в именах не различаются.

Если тег имеет несколько атрибутов, то они отделяются друг от друга пробелами:

```
<meta http-equiv="content-type" content="text/html">
```

Если атрибут не имеет содержания, то символ = (равенство) и пустые кавычки опускаются:

```
<td nowrap>Пиррова победа</td>
```

Универсальные атрибуты:

- `id` – уникальный идентификатор элемента на веб-странице;
- `class` – имя класса стилевой таблицы;
- `src` – источник встраиваемого содержимого;
- `href` – ссылка на связанный ресурс.

Запрещается использовать угловые скобки `<` и `>`, а также кавычки вне HTML разметки. При необходимости использования данных символов в тексте они должны замещаться на специальные сущности `<`, `>` и `"`; соответственно. Список наиболее часто употребляемых сущностей приведен в таблице.

Символ	Сущность	Символ	Сущность	Символ	Сущность
<code><</code>	<code>&lt;</code>	«	<code>&laquo;</code>	®	<code>&reg;</code>
<code>></code>	<code>&gt;</code>	»	<code>&raquo;</code>	©	<code>&copy;</code>
<code>"</code>	<code>&quot;</code>	—	<code>&ndash;</code>	<i>пробел</i>	<code>&nbsp;</code>
<code>'</code>	<code>&amp;</code>	—	<code>&mdash;</code>	<i>код UTF-8</i>	<code>&#1234;</code>

Для указания комментариев в тексте HTML документа используется метатег `<!-- текст комментариев -->`. Внутри комментариев разрешается добавлять другие теги. Содержимое комментариев браузером не отображается.

Структура HTML документа

Типовая структура HTML документа:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Заголовок страницы</title>
  </head>
  <body>
    ... содержание страницы ...
  </body>
</html>
```

Согласно спецификации HTML 5 первым тегом документа является метатег `<!DOCTYPE html>`, после которого размещается сам HTML документ. Данный метатег представляет собой команду для корректного отображения документа в браузере.

Тег `<html>` является главным корневым тегом веб-страницы, он содержит два контейнерных тега `<head>` и `<body>`, задающих заголовок страницы и ее содержание. Заголовок `<head>` содержит служебную часть, которая содержит различные теги, определяющие тип и формат содержимого для браузера (метатеги), а также ссылки на различное подгружаемое содержимое: каскадные стилевые таблицы CSS, скрипты со сценарием поведения Javascript и др. В данном примере указано минимальное содержимое веб-страницы в HTML 5: тег `<meta>` с помощью атрибута `charset` задает кодировку документа; тег `<title>` задает заголовок окна. Тело документа `<body>` представляет собой непосредственно выводимое содержимое на экран браузера.

1.2. Элементы оформления текста

Семантическая верстка страницы

Стандартом HTML 5 предусмотрены семантические теги для верстки содержимого страницы. Структура сайта (см. рисунок 1) может состоять из нескольких отдельных частей, которые могут быть представлены соответствующими контейнерными тегами: шапка сайта `<header>`, главное верхнее меню `<nav>`, боковая панель `<aside>`, содержимое `<article>` и подвал `<footer>`. Содержимое, в свою очередь, может состоять из одной или нескольких секций `<section>`,

содержащих заголовки, абзацы, списки и другие структурные HTML элементы. Элементы `<section>` могут быть вложенными.



Рис. 1. Структура сайта по стандарту HTML 5

Также в HTML 5 может использоваться семантический тег `<main>`, задающий основное содержимое страницы. На странице может быть только один тег `<main>`, при этом он не должен располагаться внутри элементов `<article>`, `<aside>`, `<footer>`, `<header>` или `<nav>`.

Основное отличие семантической верстки от физической состоит в том, что теги семантической верстки никаким образом не влияют на изначальное представление HTML страницы в браузерах – если разметка не стилизована средствами CSS, то она не отображается. Впоследствии для каждого такого элемента можно определить свой собственный стиль отображения, не прибегая к описанию именных классов и уникальных стилей в каскадных таблицах CSS.

Таким образом, использование семантической верстки HTML 5 позволяет значительно упростить представление отдельных элементов страницы и улучшить читабельность HTML кода.

Заголовки

HTML предлагает шесть заголовков разного уровня (см. рисунок 2), которые показывают относительную важность секции, расположенной после заголовка. Так, элемент `<h1>` представляет собой наиболее важный заголовок первого уровня, а `<h6>` служит для обозначения наименее важного заголовка шестого уровня.

Пример заголовка:

```
<h1>Заголовок первого уровня</h1>
```

В браузере заголовки отображаются шрифтом жирного начертания от `<h1>` к `<h6>` в сторону уменьшения размера. Заголовок `<h4>` отображается размером текущего текста.

Элементы `<h1>`,...,`<h6>` относятся к контейнерным элементам, они всегда начинаются с новой строки, как и элементы, следующие за ними. До и после заголовка добавляется абзацный интервал.

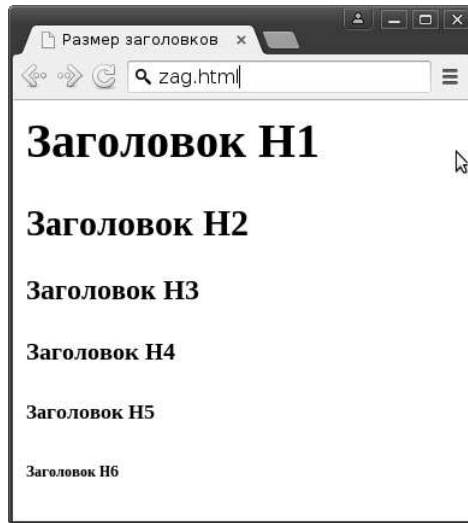


Рис. 2. Пример заголовков различного уровня

Абзацы

Основной структурной единицей текста является абзац, выделяемый в контейнерный тег `<p>`. До и после абзаца добавляется абзацный интервал.

По умолчанию содержимое абзаца отображается в браузере выровненным по левому краю экрана (родительского контейнерного тега). Тип выравнивания можно установить с помощью атрибута `align`, принимающего одно из четырех значений:

- `left` – выравнивание содержимого абзаца слева;
- `right` – выравнивание содержимого абзаца справа;
- `center` – выравнивание содержимого абзаца по центру;
- `justify` – выравнивание содержимого абзаца по ширине.

Атрибут `align` является устаревшим. Вместо использования атрибута `align` рекомендуется устанавливать отдельный стиль для тега `<p>` в каскадной таблице CSS.

Блочные элементы

Блочные элементы характеризуются тем, что они занимают всю доступную ширину, высота элемента определяется его содержимым, и каждый элемент начинается с новой строки.

Для оформления текста в HTML имеется несколько блочных элементов. Все эти элементы представляют собой контейнеры:

- `<div>` – представляет собой универсальный контейнер, служащий для блочной верстки. В отличие от тега `<p>` при отображении элемента браузером отсутствуют абзацные интервалы. Стилизация определяется средствами CSS.
- `<blockquote>` – предназначен для выделения длинных цитат внутри документа. Текст, обозначенный этим тегом, традиционно отображается как выровненный по ширине абзац с отступами слева и справа (около 40 пикселей), с абзачными интервалами снизу и сверху.
- `<address>` – предназначен для оформления контактных данных. Текст отличается от оформления абзаца `<p>` курсивным начертанием.
- `<pre>` – задает блок исходного отформатированного текста. Такой текст отображается обычно моноширинным шрифтом, со всеми пробелами между словами. Обычно при отображении HTML страницы в браузере любое количество пробелов идущих подряд, а также символов перевода строки отображается как один пробел. Тег `<pre>` позволяет обойти эту особенность и отображать исходное форматирование текста.
- `<hr>` – задает декоративную горизонтальную разделительную черту. Данный элемент не имеет завершающего тега и является одиночным.

Строчные элементы

Строчные элементы занимают ширину и высоту, определяемую содержимым, и могут размещаться внутри абзацев и других блочных элементов. Как правило, строчные элементы служат для выделения и оформления текста. Большинство строчных элементов представляют собой контейнеры:

- `` – представляет собой универсальный контейнер строчного содержимого. Стилизация определяется средствами CSS.
- `` – выделение текста (также ``). Содержимое тега при просмотре в браузере выделяется полужирным шрифтом.
- `` – акцентирование текста (также `<i>`). Содержимое тега при просмотре в браузере выделяется курсивом.
- `<q>` – содержимое элемента в браузере отображается в кавычках.
- `<cite>` – оформление цитаты. Содержимое тега при просмотре в браузере выделяется курсивом.
- `
` – разрыв строки. Данный элемент не имеет завершающего тега и является одиночным. Отображение последующего за ним текста в браузере переносится на следующую строку.

Пример:

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

Списки

Стандарт HTML позволяет представлять два вида списков: нумерованных и маркированных.

Нумерованный список задается контейнерным тегом ``. Списки состоят из произвольного количества вложенных элементов ``, также представляющих собой контейнеры. При отображении в браузере каждый элемент списка будет иметь абзацный отступ и нумерацию. Необязательный атрибут `start` задает начальное число, с которого начинается нумерация (по умолчанию с 1). Для установки нумерации отдельного элемента списка `` может использоваться атрибут `value`.

Пример нумерованного списка:

```
<ol>
  <li>Первый элемент</li>
  <li>Второй элемент</li>
  <li>Третий элемент</li>
</ol>
```

Результат выполнения примера представлен на рисунке 3.

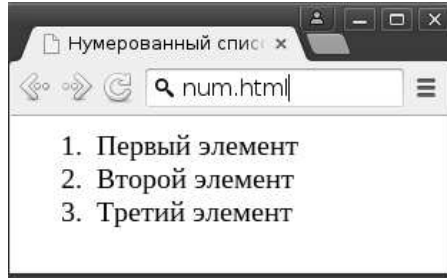


Рис. 3. Пример нумерованного списка

Маркированный список задается контейнерным тегом ``, содержащим элементы списка ``. При отображении в браузере каждый элемент списка будет иметь абзацный отступ и предваряться маркером.

Пример маркированного списка:

```
<ul>
  <li>Первый элемент</li>
  <li>Второй элемент</li>
  <li>Третий элемент</li>
</ul>
```

Результат выполнения примера представлен на рисунке 4.

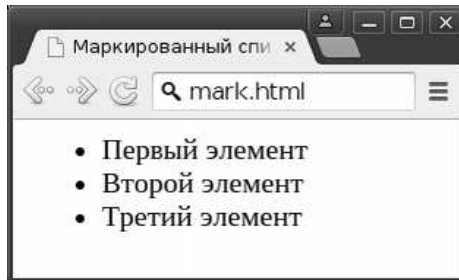


Рис. 4. Пример маркированного списка

Список определений задается контейнерным тегом `<dl>`. Элементами списка определений являются пары тегов `<dt>`, определяющих термины, и тегов `<dd>`, определяющих разъяснения этих терминов. При отображении в браузере термины выравниваются по левому краю, а разъяснения выводятся с новой строки с небольшим отступом.

Пример списка определений:

```
<d1>
  <dt>TCP/IP</dt>
  <dd>Сетевой протокол</dd>
  <dt>Apache</dt>
  <dd>Веб-сервер</dd>
</d1>
```

Результат выполнения примера представлен на рисунке 5.

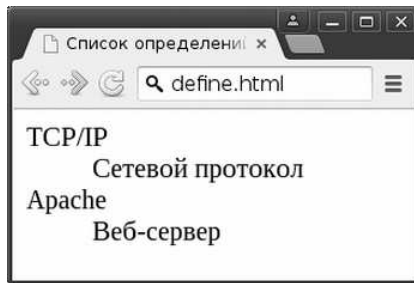


Рис. 5. Пример списка определений

В HTML разрешается использовать вложенные списки. Для этого все теги внутреннего списка вкладываются в элемент `` внешнего списка. При отображении вложенных списков в браузере выводится только текущий уровень нумерации элементов. Каждый уровень списка отображается со своим отступом.

Пример вложенного списка:

```
<ol>
  <li>Верстка
    <ol start="2">
      <li>Табличная</li>
      <li>Влочная</li>
    </ol>
  </li>
  <li value="4">Гиперссылки</li>
</ol>
```

Результат выполнения примера представлен на рисунке 6.

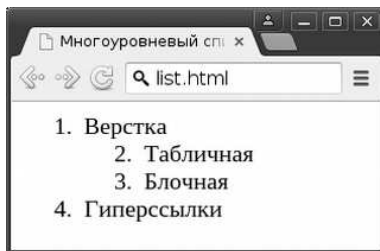


Рис. 6. Пример вложенного списка

Гиперссылки

Гиперссылки в HTML представляются строчным контейнерным тегом `<a>`. По умолчанию гиперссылки отображаются в браузере подчеркнутым текстом синего цвета. Посещенные гиперссылки отображаются сиреневым цветом. При наведении указателя на гиперссылку он принимает вид указательного пальца. При активации гиперссылки происходит переход по заданному URL адресу и открытие в окне браузера документа, находящегося по этому адресу.

Атрибуты:

- `href` – URL адрес документа, на который следует перейти;
- `target` – цель гиперссылки, указывающая имя окна или фрейма, куда будет загружаться документ. Для загрузки в новое окно браузера указывается значение `target="_blank"`;
- `download` – вывод диалогового окна с предложением загрузки или сохранения документа (работает в Chrome, Firefox, Opera, Microsoft Edge).

В качестве гиперссылок могут использоваться как абсолютные, так и относительные URL адреса. При использовании абсолютной адресации в URL рекомендуется указывать протокол. Основные типы URL, используемые при отображении HTML страниц в браузерах:

- `http://htmlbook.ru/test.html` – протокол передачи гипертекста (*HyperText Transfer Protocol*);
- `https://htmlbook.ru/test.html` – защищенный протокол передачи гипертекста с использованием шифрования (*HyperText Transfer Protocol Security*);
- `ftp://htmlbook.ru/file.zip` – файловый протокол передачи данных (*File Transfer Protocol*);
- `file://d:/example/test.zip` – адрес файла на локальном компьютере;

- `mailto:ivanov@gmail.com` – адрес электронной почты;
- `skype:ivanov?call` – вызов абонента по адресу Skype.

Относительные адреса позиционируются от папки размещения текущей страницы на веб-сервере:

```
<a href="example/test.html">Относительная ссылка </a>
<a href="http://htmlbook.ru/example/test.html">
```

```
Абсолютная ссылка</a>
```

Переход по гиперссылке будет зависеть от типа загружаемого документа. Например, архивные файлы с расширением `zip` или `rar` будут сохраняться на локальный диск:

```
<a href="test.zip">Тестовый архив</a>
```

1.3. Таблицы

Таблица в HTML представляет собой отдельный блочный элемент `<table>`. В составе таблицы можно выделить отдельные структурные элементы:

- `<caption>` – заголовок;
- `<thead>` – начальный колонтитул;
- `<tbody>` – содержание;
- `<tfoot>` – конечный колонтитул.

Все эти элементы являются опциональными и, за исключением тега `<caption>`, могут включать одну или несколько строк таблицы `<tr>`. Если в таблице содержание и колонтитулы не указаны, то строки `<tr>` могут включаться непосредственно в содержание тега `<table>`.

Каждая строка таблицы `<tr>` может включать одну или несколько ячеек. Ячейки могут быть представлены двумя тегами:

- `<td>` – ячейка;
- `<th>` – заголовок столбца.

По умолчанию текст заголовка столбца отображается в браузере полужирным шрифтом, выровненным по центру; текст простой ячейки – выровненным по левому краю; текст заголовка – выровненным по центру. Размер таблицы по умолчанию определяется ее содержанием, границы таблицы не отображаются, по вертикали содержимое всех ячеек с выравниванием по центру. На рисунке 7 показана структура таблицы в HTML.

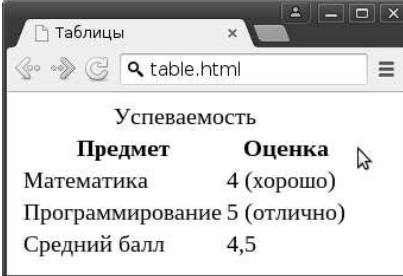


Рис. 7. Структура таблицы в HTML

Пример таблицы:

```
<table>
  <caption>Успеваемость</caption>
  <thead>
    <tr>
      <th>Предмет</th>
      <th>Оценка</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Математика</td>
      <td>4 (хорошо)</td>
    </tr>
    <tr>
      <td>Программирование</td>
      <td>5 (отлично)</td>
    </tr>
  </tbody>
  <tfoot>
    <tr>
      <td>Средний балл</td>
      <td>4,5</td>
    </tr>
  </tfoot>
</table>
```

Результат выполнения примера представлен на рисунке 8.



Предмет	Оценка
Математика	4 (хорошо)
Программирование	5 (отлично)
Средний балл	4,5

Рис. 8. Пример таблицы

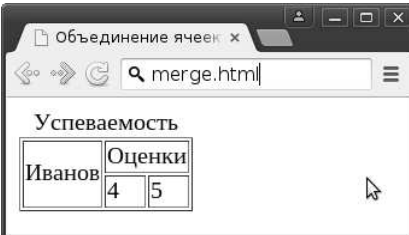
Атрибут `border` тега таблицы `<table>` задает ширину внешней границы таблицы в пикселах (по умолчанию, 0) и позволяет сделать ее отображаемой. Данный атрибут является устаревшим, рекомендуется использовать каскадную таблицу стилей CSS.

Атрибут `colspan` тегов `<td>` и `<th>` задает объединение соседних ячеек по горизонтали (число колонок). Атрибут `rowspan` задает объединение соседних ячеек по вертикали (число строк).

Пример таблицы с объединенными ячейками:

```
<table border="1">
  <caption>Успеваемость</caption>
  <tr>
    <td rowspan="2">Иванов</td>
    <td colspan="2">Оценки</td>
  </tr>
  <tr>
    <td>4</td>
    <td>5</td>
  </tr>
</table>
```

Результат выполнения примера представлен на рисунке 9.



Иванов	Оценки
4	5

Рис. 9. Пример таблицы с объединенными ячейками

Таблицы могут иметь произвольную вложенность друг в друга – на этом основан так называемый табличный дизайн страницы. Размеры таблиц, а также отдельных строк и столбцов, стиль отображения границ, выравнивание содержимого ячеек и другие свойства задаются с помощью стилизации CSS.

1.4. Графика

Форматы графических изображений

В HTML страницы могут быть внедрены графические файлы трех основных форматов: GIF, PNG и JPEG.

Формат GIF (*Graphics Interchange Format*) – разработан в 1987 году, используется для хранения элементов оформления HTML страниц. Изображения GIF поддерживают кодирование только 256 цветов, что в большинстве случаев является недостаточным для полноцветного представления. В настоящий момент считается устаревшим. Достоинства формата GIF: использование алгоритма сжатия LZW без потерь качества; малый размер файлов и высокая скорость их загрузки; поддержка прозрачности; возможность анимирования изображений.

Формат PNG (*Portable Network Graphics*) – разработан в 1996 году на замену устаревшего формата GIF, используется для хранения полноцветных изображений, содержащих текст, чертежи и схемы (тонкие линии). Достоинства формата PNG: использование алгоритма сжатия *Deflate* без потерь качества; поддержка прозрачности; использование полноцветной палитры *TrueColor* (16М цветов, 24 бит/пиксел). Анимирование изображений невозможно.

Формат JPEG (*Joint Photographic Expert Group*) – разработан в 1993 году для хранения полноцветных изображений в фотографии. Достоинства формата JPEG: использование высокоэффективного алгоритма сжатия с усреднением, позволяющего сохранять без видимых потерь качества полноцветные фотографические изображения в относительно малых по размеру файлах. Прозрачность и анимирование изображений не поддерживаются. Формат плохо подходит для хранения изображений, содержащих текст, схемы и тонкие линии.

Вставка изображения на страницу

Для вставки изображения на страницу используется одиночный тег ``. Обязательный атрибут `src` указывает URL изображения. Изображение будет вставлено на место размещения тега в тексте

HTML страницы. Тег `` является встроенным строчным элементом.

Некоторые используемые атрибуты:

- `alt` – обозначение альтернативного текста, выводимого на месте изображения в браузере в случае отключения загрузки изображений. Рекомендуются атрибут.
- `title` – текст подсказки, всплывающей при наведении указателя на изображение.

Пример:

```

```

В данном примере изображение содержится в файле *flowers.jpg* в каталоге `images` на веб-сервере. Изображение загружается и выводится на экран браузера при отображении страницы.

Изображение также может быть содержимым гиперссылки. В этом случае активация перехода происходит при щелчке мыши на изображении:

```
<a href="description.html"></a>
```

Для определения холста изображения используется контейнерный тег семантической разметки `<figure>`, в который могут включаться теги изображения ``, подписи `<figcaption>` и другие теги. Тег `<figcaption>` содержит текст подписи и также является контейнерным. Пример:

```
<figure>
  
  <figcaption>Цветы</figcaption>
</figure>
```

Дополнительное оформление фона и стиля страницы с помощью изображений, а также выравнивание, обтекание текстом, изменение размера изображений рекомендуется осуществлять с помощью таблиц CSS.

1.5. Формы и элементы управления

Для взаимодействия клиента с веб-сервером в HTML используются клиентские формы. Форма, имеющая поля ввода и другие элементы управления, заполняется клиентом с помощью браузера, проверяется обработчиком Javascript на стороне клиента и затем отправляется на веб-сервер, где управление передается

отдельному серверному приложению, написанному на одном из серверных скриптовых языков, например PHP. Взаимодействие клиента с сервером осуществляется с помощью сетевого протокола HTTP (*HyperText Transfer Protocol*).

Для создания формы используется контейнерный тег `<form>`. Атрибут `action` указывает URL адрес серверного приложения. Если данные формы обрабатываются и отправляются на сервер сценарием Javascript, то в качестве значения атрибута `action` можно указать пустой URL адрес: `action="#"`.

Стандартные атрибуты формы:

- `name` – имя формы. Используется для идентификации формы в Javascript.
- `method` – метод HTTP отправки формы на сервер. Атрибут может принимать значения `GET` или `POST`. Если атрибут не задан, используется метод `GET`. В случае использования метода `GET` данные формы отправляются на сервер в строке с URL адресом серверного приложения после знака `?` (вопрос) в виде пар: `имя=значение`, разделенных символом `&` (амперсанд), и передаются управляющей программе. Имена определяются атрибутами `name` элементов управления, а значения вводятся пользователем или устанавливаются по умолчанию с помощью атрибутов `value`. Национальный алфавит и специальные символы кодируются в формате %HH, представляющем собой шестнадцатеричный код UTF-8.

Пример:

```
http://htmlbook.ru/handler.php?nick=%ED%FF&page=5
```

В случае использования метода *POST* данные формы также передаются управляющей программе, но не в строке URL, а в содержании HTTP запроса:

```
http://htmlbook.ru/handler.php  
nick=%C2%E0%ED%FF&page=5
```

- `enctype` – способ кодирования данных формы. Атрибут может принимать значения:
 - ♦ `application/x-www-form-urlencoded` – для передачи данных, закодированных в UTF-8;
 - ♦ `multipart/form-data` – для передачи некодированных бинарных данных (файлов);
 - ♦ `text/plain` – для передачи некодированных текстовых данных.

Если атрибут не задан, используется первый способ кодирования `application/x-www-form-urlencoded`.

Метки

Меткой является вспомогательный контейнерный тег `<label>`. Метки служат для связи дополнительной текстовой информации с элементами формы. Существует два способа связывания элемента формы и метки:

- использование атрибута `id` элемента формы и указание его имени в качестве атрибута `for` элемента `<label>`;
- элемент формы помещается внутрь контейнера `<label>`.

Элементы управления

К элементам управления формы относятся поля ввода, скрытые поля, поля для отправки файлов, области редактирования, выпадающие списки, переключатели, флажки и кнопки. Каждый элемент управления имеет свой вид отображения в браузере.

Поля ввода, скрытые поля, поля для отправки файлов, переключатели, флажки и кнопки создаются с помощью одиночного тега `<input>`. Тип элемента управления определяется значением атрибута `type`.

Для группировки элементов формы используется контейнерный тег `<fieldset>`. Такая группировка облегчает работу с формами, содержащими большое число данных. Элемент `<fieldset>` отображается браузером в виде рамки. Рамка может включать заголовочную надпись, определяемую содержимым вложенного контейнерного тега `<legend>`. Пример:

```
<fieldset>
  <legend>Владение языками</legend>
  <input type="checkbox">русский
  <input type="checkbox">английский
</fieldset>
```

Текстовое поле ввода

Предназначение: ввод символов с клавиатуры.

Пример:

```
<input name="family" type="text">
```

Результат выполнения примера представлен на рисунке 10.




Рис. 10. Текстовое поле ввода

Некоторые используемые атрибуты:

- `value` – значение поля по умолчанию.
- `size` – ширина поля в моноширных символах.
- `maxlength` – максимальное количество вводимых символов.
- `autofocus` – устанавливает автофокус (атрибут без содержания).
- `form` – имя формы. Используется для привязки отдельного элемента к заданной форме.
- `disabled` – блокирует доступ и изменение элемента (атрибут без содержания).
- `readonly` – блокирует изменение элемента (атрибут без содержания).
- `placeholder` – задает текст подсказки, выводимой в поле.
- `required` – проверяет заполненность вводимого поля (атрибут без содержания). В случае если поле незаполнено, в браузер выводится предупреждающее сообщение.
- `pattern` – устанавливает регулярное выражение в соответствии с ECMA-262 (*ECMAScript 2015 Language Specification*) для фильтрации вводимого текста. В браузере вводится только текст, разрешенный шаблоном.

Примеры регулярных выражений: `\d` – одна цифра; `\w` – одна латинская буква; `\s` – один пробел; `[A-Z]` – один символ из диапазона; `\D` – не цифра; `\S` – не пробел; `^` – начало строки; `$` – конец строки; `.` – один любой символ; `.*` – любое количество любых символов; `^\S+` – первое слово; `\S+$` – последнее слово; `\d{3}` – три цифры; `\d{3,6}` – от трех до шести цифр; `\.` – десятичная точка; `^[^0]` – не ноль; `(one|two)` – фраза *one* или *two* и т.д.

Поле ввода пароля

Предназначение: маскировка вводимых символов браузером.

Пример:

```
<input name="passw" type="password">
```

Результат выполнения примера представлен на рисунке 11.



Рис. 11. Поле ввода пароля

Обратите внимание, что несмотря на маскировку символов браузером, при отправке данных на сервер значения полей ввода паролей передаются по протоколу HTTP в незашифрованном виде. Для передачи конфиденциальной информации используйте шифрование трафика SSL (*Secure Socket Layer*) и протокол HTTPS.

Виды и значения остальных атрибутов совпадают с текстовым полем ввода.

Скрытое поле ввода

Предназначение: передача не отображаемых браузером данных на сервер. Может включать атрибут `value`, содержащий предустановленное значение.

Пример:

```
<input name="userid" type="hidden" value="135">
```

Текстовая кнопка

Предназначение: связывает событие `onclick` элемента с обработчиком Javascript. Используется для предварительной обработки данных формы перед отправкой их на сервер.

Пример:

```
<input name="btn" type="button" value="Ввод">
```

Результат выполнения примера представлен на рисунке 12.



Рис. 12. Текстовая кнопка

Некоторые используемые атрибуты:

- `value` – текст кнопки, отображаемый браузером;
- `onclick` – функция обработчика события Javascript.

Кнопка отправки данных

Предназначение: отправка данных формы на сервер.

Пример:

```
<input name="btn" type="submit">
```

Результат выполнения примера представлен на рисунке 13.

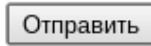


Рис. 13. Кнопка отправки данных

Атрибуты те же, что и у обычной кнопки.

Кнопка сброса

Предназначение: возвращение данных формы в первоначальное состояние.

Пример:

```
<input name="btn" type="reset">
```

Результат выполнения примера представлен на рисунке 14.

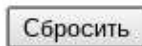


Рис. 14. Кнопка сброса

Атрибуты те же, что и у обычной кнопки.

Флажок выбора опций

Предназначение: установка и снятие флажка опции. Используется при выборе нескольких элементов из предложенных. Если при отправке формы флажок не установлен, данные элемента не передаются. Пример использования:

```
<label>Выберите теплые цвета:
  <input name="opt1" type="checkbox">красный
  <input name="opt2" type="checkbox">желтый
</label>
```

Результат выполнения примера представлен на рисунке 15.

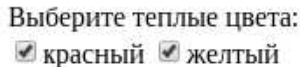


Рис. 15. Флажок выбора опций

Некоторые используемые атрибуты:

- `value` – значение элемента. Если этот атрибут отсутствует, то при выборе опции элемент принимает значение `on`.

- `checked` – делает флажок установленным по умолчанию.
- `disabled` – блокирует доступ к элементу (атрибут без содержания). Независимо от состояния элемента данные не передаются.
- `required` – проверяет наличие установленного флажка (атрибут без содержания).

Переключатель

Предназначение: выбор одного из нескольких предложенных элементов из группы. Все элементы переключателя должны иметь одинаковое имя `name`. Если при отправке формы переключатель не установлен, его данные не передаются.

Пример использования:

```
<label>Выберите верный ответ:
  <input name="opt" type="radio" value="1">да
  <input name="opt" type="radio" value="0">нет
</label>
```

Результат выполнения примера представлен на рисунке 16.

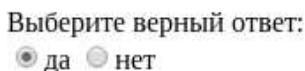


Рис. 16. Переключатель

Атрибуты те же, что и у флажка выбора опций. Атрибут `checked` может быть только у одного элемента из группы.

В случае если форма содержит несколько независимых переключателей, элементы в них группируются по одноименному атрибуту `name`.

Поле для отправки файлов

Предназначение: прикрепление содержания файла к отправляемой форме. Форма должна отправляться на сервер с помощью метода `POST`, способ кодирования данных должен быть установлен как `multipart/form-data`. При нажатии на кнопку *"Обзор"* в браузере открывается диалог выбора файла. После выбора файла его имя появляется в поле ввода, а содержимое прикрепляется к форме и подготавливается для отправки на сервер.

Пример использования:

```
<input name="attach" type="file">
```


Результат выполнения примера представлен на рисунке 17.

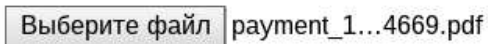


Рис. 17. Поле отправки файла

Дополнительные атрибуты:

- `accept` – устанавливает фильтр на MIME-типы прикрепленных файлов. При нескольких значениях MIME-типы перечисляются через запятую. Возможно обобщение типов, например `image/*` – для всех графических файлов.
- `multiply` – позволяет указывать одновременно несколько файлов в поле (атрибут без содержания).
- `maxlength`, `size` и `value` – аналогичные текстовому полю ввода.

Поле электронной почты

Предназначение: указание адреса электронной почты. Адрес проверяется на корректность ввода.

Пример:

```
<input name="email" type="email">
```

Поле ввода чисел

Предназначение: ввод целых чисел. Содержит кнопки-стрелки вверх-вниз, изменяющие введенное значение на один шаг.

Пример:

```
<input name="num" type="number">
```

Дополнительные атрибуты:

- `value` – значение по умолчанию;
- `min` – минимальное значение;
- `max` – максимальное значение;
- `step` – шаг приращения (по умолчанию 1).

Поле выбора чисел из диапазона

Предназначение: выбор целых чисел из диапазона. Отображает ползунок, изменяющий значение на один шаг.

Пример:

```
<input name="num" type="range">
```

Дополнительные атрибуты:

- `value` – значение по умолчанию;
- `min` – минимальное значение (по умолчанию 0);
- `max` – максимальное значение (по умолчанию 100);
- `step` – шаг приращения (по умолчанию 1).

Область редактирования

Предназначение: ввод многострочного текстового содержимого, предоставление возможности онлайн редактирования. Область образуется с помощью контейнерного тега `<textarea>`.

Пример использования:

```
<textarea name="content">
  Область редактирования
</textarea>
```

Результат выполнения примера представлен на рисунке 18.



Рис. 18. Область редактирования

Дополнительные атрибуты:

- `cols` – ширина текстовой области в символах моноширного текста.
- `rows` – высота текстовой области в строках, отображаемых без прокрутки содержимого.
- `wrap` – режим переноса на следующую строку. Может принимать одно из трех значений: `physical`, `virtual` и `off`.
- `disabled`, `readonly`, `placeholder` и `required` – аналогичные текстовому полю ввода.

Выпадающий список

Предназначение: выбор одного или нескольких элементов из заданного списка. Список задается контейнерным тегом `<select>`, включающим контейнерные элементы `<option>` с текстовым содержимым. При выборе элемента из списка его значение `value` устанавливается для контейнера `<select>` и передается на сервер.

Пример использования:

```
<select name="lang">
  <option value="RU">русский</option>
  <option value="EN">английский</option>
</select>
```

Результат выполнения примера представлен на рисунке 19.

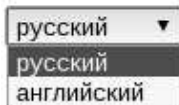


Рис. 19. Выпадающий список

Дополнительные атрибуты тега `<select>`:

- `multiply` – возможность множественного выбора элементов списка (атрибут без содержания). Множественный выбор элементов осуществляется с участием нажатой клавиши `<Ctrl>`.
- `size` – определяет высоту списка в элементах (по умолчанию, `1`).
- `disabled` – блокирует выбор элементов (атрибут без содержания).
- `required` – проверяет, выбран ли какой-либо элемент из списка (атрибут без содержания). В случае если элемент не выбран, в браузер выводится предупреждающее сообщение.
- `value` – значение элемента списка. Атрибут относится к тегу `<option>`. Если атрибут не установлен, то значение выбранного элемента определяется его содержимым.
- `selected` – выделение элемента списка по умолчанию (атрибут без содержания). Атрибут относится к одному или нескольким (в случае множественного выбора) тегам `<option>`.

Элементы списка могут быть сгруппированы в иерархию с помощью специального контейнерного тега `<optgroup>`. Атрибут `label` используется для обозначения заголовка группы списка. Заголовок группы отображается выделенным текстом в браузере и является невыбираемым элементом.

Пример использования:

```
<select name="lang">
  <optgroup label="Языки ввода">
    <option value="RU">русский</option>
    <option value="EN">английский</option>
```

```

    </optgroup>
</select>

```

Результат выполнения примера представлен на рисунке 20.

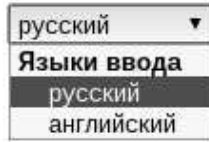


Рис. 20. Выпадающий список с группировкой элементов

Комбинированный список

Предназначение: выбор при наборе в текстовом поле. Список задается контейнерным тегом `<datalist>`, включающим контейнерные элементы `<option>` с текстовым содержимым. С помощью атрибута `id` тега `<datalist>` список связывается с текстовым полем `<input>`, имеющим соответствующий атрибут `list`. Список становится доступным при получении полем фокуса или при наборе текста.

Пример:

```

<input name="lang" list="lang">
<datalist id="lang">
  <option value="RU">русский</option>
  <option value="EN">английский</option>
</datalist>

```

2. Каскадные стилевые таблицы CSS 3

2.1. Синтаксис CSS правил

Каскадной стилевой таблицей CSS (Cascading Style Sheets) называется набор правил – параметров форматирования, которые применяются к элементам документа, чтобы изменить их внешний вид. Стилизация страницы обычно применяется после разработки структуры и наполнения ее содержанием средствами HTML. Параметры форматирования объединяются в правила вида:

```

селектор1, селектор2, ... {
  свойство1: значение1;

```

```
    свойство2: значение2; ...
}
```

Селекторы определяют теги, к которым применяется данное правило. Каждый селектор может применяться к одному или нескольким тегам веб-страницы. CSS правило может определяться тремя разными способами (в порядке уменьшения значимости):

- **Локальный.** Стиль определяется в атрибуте *style* тега. Переопределяет внешние и страничные стили. Правило распространяется только на текущий тег.

Например:

```
<div style="text-align: center">Текст по центру</div>
```

- **Страничный.** Стиль определяется внутри контейнерного тега вида: `<style type="text/css">`, размещаемого в заголовке HTML документа. Переопределяет внешние стили. Правило распространяется на весь HTML документ.

Например:

```
<style type="text/css">
  div { text-align: center; }
</style>
```

Дополнительный атрибут *media* служит для определения класса устройств, к которым применимы данные стили, например:

- ♦ *screen* – экран компьютера (по умолчанию);
- ♦ *print* – принтер;
- ♦ *handheld* – смартфон.
- **Внешний.** Стиль определяется во внешнем CSS файле. Внешний файл подключается в заголовке страницы с помощью тега вида: `<link rel="stylesheet" type="text/css" href="styles.css">`, где *styles.css* – имя внешнего CSS файла.

Определение стилей

При загрузке веб-страницы сначала загружаются внешние CSS файлы и к тегам веб-страницы применяются все определенные в них стили, затем загружаются страничные стили, наконец, последними применяются локальные стили. При этом более специфичные правила переопределяют менее специфичные. Таким образом, все правила, описанные в каскадных таблицах стилей, применяются последовательно ко всем встречаемым тегам, дополняя и переопределяя их свойства. Входящие в правила *селекторы* могут

состоять из имен тегов, классов, идентификаторов, псевдоклассов и псевдоэлементов.

- *Имена тегов* применяют, когда необходимо определить стиль для отдельных тегов веб-страницы. Для этого используется следующий синтаксис правил:

```
тег { свойство: значение; ... }
```

Пример описания CSS стиля:

```
div { text-align: center; } /* Для всех тегов div */
```

При этом структура веб-страницы остается неизменной, а все содержание тегов *div* будет выровнено по центру:

```
<div>Текст по центру</div>
```

- *Группирование селекторов.* Когда на веб-странице одновременно используется множество селекторов, возможно появление повторяющихся стилевых правил. Чтобы не повторять правила для каждого селектора, их можно сгруппировать для удобства представления и сокращения кода.

Для группировки селекторов их наименования перечисляются через запятую:

```
селектор1, селектор2, ... { свойство: значение; ... }
```

Примеры стилей:

```
/* Для всех тегов p и div */
p, div { text-align: center; }
/* Для заголовков */
h1, h2, h3 { font-family: Arial, Helvetica, sans-serif; }
```

- *Классы* применяют, когда необходимо определить стиль для индивидуального элемента веб-страницы или задать разные стили для одного тега. При использовании совместно с тегами синтаксис для классов будет следующий:

```
тег.имя_класса { свойство: значение; ... }
```

Пример стиля:

```
p.cite { color: red; } /* Абзацы с классом cite */
```

Чтобы применить класс к некоторым абзацам веб-страницы, для соответствующих тегов `<p>` необходимо указать атрибут `class`:

```
<p class="cite">Красный цвет текста</p>
```

Можно использовать классы и без указания тега:

```
.имя_класса { свойство: значение; ... }
```

Пример стиля:

```
.cite { margin-left: 20px; } /* Для всех тегов */
```

При такой записи класс можно применять к любому тегу:

```
<div class="cite">Отступ слева 20px</div>
```

Классы удобно использовать, когда нужно применить стиль к разным элементам веб-страницы: ячейкам таблицы, ссылкам, абзацам и др., например:

```
div { background-color: #fff; } /* Цвет фона белый*/
div.odd { background-color: #ccc; } /* Цвет фона серый*/
```

```
<div>белый цвет фона</div>
```

```
<div class="odd">серый цвет фона</div>
```

- К любому элементу можно добавить несколько классов, описываемых отдельными правилами. Такие классы называются *мультиклассами*. В этом случае классы перечисляются в атрибуте class через пробел:

```
<div class="cite odd">Отступ слева и серый цвет
фона</div>
```

Для выделения таких тегов можно описать отдельное правило, указав селекторы классов без пробелов:

```
.cite.odd { margin-left: 20px; background-color: #ccc; }
```

- *Идентификаторы* (ID селекторы) определяют уникальные имена элементов, которые используются для изменения их стилей и обращений к ним посредством Javascript. Синтаксис применения идентификатора следующий:

```
#имя_идентификатора { свойство: значение; ... }.
```

Пример стиля:

```
#help { width: 300px; background-color: gray; }
```

Для применения стиля идентификатора к элементу веб-страницы в соответствующем теге необходимо указать атрибут id:

```
<div id="help">Как поймать льва в пустыне?</div>
```

Так же, как и при использовании классов, идентификаторы можно применять к конкретному тегу:

```
тег#имя_идентификатора { свойство: значение; ... }
```

Пример стиля:

```
div#help { width: 300px; background-color: gray; }
```

- *Контекстные селекторы* используются для выделения тегов из сложной структуры веб-страницы и применения к ним стилизации. Простой контекстный селектор состоит из нескольких селекторов, разделенных пробелом, что соответствует вложенности тегов на веб-странице:

```
селектор1 селектор2 { свойство: значение; ... }
```

Пример стиля:

```
p div { background-color: #ccc; }
```

В этом случае стиль будет применяться ко всем тегам `<div>`, размещенным внутри тегов `<p>`. Уровень вложенности тегов при этом значения не имеет.

- *Соседние селекторы* применяются к непосредственно следующим друг за другом тегам на одном уровне иерархии веб-страницы. Для управления стилями соседних элементов используется символ `+` (плюс) между селекторами:

```
селектор1 + селектор2 { свойство: значение; ... }
```

Соседние селекторы удобно использовать для тех тегов, к которым автоматически добавляются отступы, чтобы самостоятельно регулировать их величину. Например, расстояние между подряд идущими тегами `<h1>` и `<h2>` можно регулировать с помощью соседних селекторов:

```
h1 + h2 { margin-top: -10px; } /* Смещение вверх */
```

- *Родственные селекторы* применяются к любым не подряд идущим слева направо тегам, размещаемым на одном уровне иерархии (имеющим общего родителя). Для управления стилями родственных элементов используется символ `~` (тильда) между селекторами:

```
селектор1 ~ селектор2 { свойство: значение; ... }
```

```
h1 ~ h2 { margin-top: -10px; } /* Смещение всех заголовков 2 вверх */
```

- *Дочерние селекторы* применяются к непосредственно вложенным друг в друга тегам. Для управления стилями дочерних элементов используется символ `>` (больше) между селекторами:

```
селектор1 > селектор2 { свойство: значение; ... }
```



```
ul > li { list-style: none; } /* Убрать маркеры списка
*/
```

- *Селекторы атрибутов* используются для выделения тегов, имеющих специфичные атрибуты:

```
[атрибут] { свойство: значение; ... }
селектор[атрибут] { свойство: значение; ... }
```

Пример стиля:

```
q[title] { color: red; }
```

В результате к тегам `<q>`, имеющим атрибут `title`, применяется красный цвет.

Для выделения любых тегов, имеющих специфичные атрибуты, имя тега можно не указывать:

```
[title] { color: red; } /* Для любого тега с атрибутом
title */
```

Для выделения тегов, имеющих определенные значения атрибутов, используется следующее правило:

```
селектор[атрибут="значение"] { свойство: значение; ... }
```

Например:

```
a[target="_blank"] { color: red; } /* Для ссылки в
новом окне */
```

Также для других видов содержания атрибутов:

- ◆ селектор `[атрибут^="значение"]` – атрибут начинается со значения;
- ◆ селектор `[атрибут$="значение"]` – атрибут заканчивается значением;
- ◆ селектор `[атрибут*="значение"]` – атрибут включает символы значения;
- ◆ селектор `[атрибут~="значение"]` – атрибут включает слово "значение".

Все перечисленные методы можно комбинировать между собой, определяя стиль для элементов, которые содержат два и более атрибута. В подобных случаях квадратные скобки идут подряд, например:

```
[атрибут="значение"][атрибут="значение"] {свойство:
значение;...}
```

- *Универсальный селектор* соответствует любому элементу веб-страницы. Для обозначения универсального селектора применяется символ `*` (звёздочка):

```
* { свойство: значение; ... }. Пример стиля:
```

```
* { margin: 0; padding: 0; } /* Для всех элементов */
```

Псевдоклассы и псевдоэлементы

Псевдоклассы определяют динамическое состояние элементов, которое изменяется с помощью действий пользователя, а также положение в дереве HTML документа. Псевдоклассы чаще всего применяются к кнопкам, гиперссылкам и другим активным элементам. Общий синтаксис:

```
селектор:псевдокласс { свойство: значение; ... }
```

Пример стиля:

```
a:active { color: #f00; } /* Цвет активной ссылки */
```

При использовании псевдоклассов браузер не перегружает текущий документ, это применяется для получения различных динамических эффектов на странице.

Допускается применять псевдоклассы к именам идентификаторов или классов, а также к контекстным селекторам. Если псевдокласс указывается без селектора впереди (:hover), то он будет применяться ко всем элементам документа.

Примеры стилей CSS:

```
a.menu:hover { color: green; }
.menu a:hover { background-color: #fc0; }
: hover { color: yellow; }
```

Условно все псевдоклассы делятся на три группы:

- определяющие состояние элементов;
- имеющие отношение к дереву элементов;
- указывающие язык текста.

1. *Псевдоклассы, определяющие состояние элементов.* К этой группе относятся псевдоклассы, которые распознают текущее состояние элемента и применяют стиль только для этого состояния.

- `:active` – происходит при активации пользователем элемента. Например, ссылка становится активной, если навести на неё курсор и щёлкнуть мышкой. Несмотря на то, что активным может стать практически любой элемент веб-страницы, псевдокласс `:active` используется преимущественно для ссылок.
- `:link` – применяется к непосещенным ссылкам, т. е. таким ссылкам, на которые пользователь ещё не нажимал. Браузер некоторое время сохраняет историю посещений, поэтому ссылка может быть помечена как посещенная хотя бы потому, что по ней был зафиксирован переход ранее.

- Запись `a {...}` и `a:link {...}` по своему результату равноценна, поскольку в браузере даёт один и тот же эффект, поэтому псевдокласс `:link` можно не указывать. Исключением являются якоря, на них действие `:link` не распространяется.
- `:focus` — применяется к элементу при получении им фокуса. Например, для текстового поля формы получение фокуса означает, что курсор установлен в поле и с помощью клавиатуры можно вводить в него текст.
- `:hover` — активизируется при наведении на элемент курсора мыши, но без нажатия на кнопку, например:

```
tr:hover { background-color: #fc0; } /* Изменение
цвета фона строки */
```

- `:visited` — применяется к посещённым ссылкам. Обычно такая ссылка меняет свой цвет по умолчанию на фиолетовый, но с помощью стилей цвет и другие параметры можно задать самостоятельно.

Селекторы могут содержать более одного псевдокласса, которые перечисляются подряд через двоеточие, но только в том случае, если их действия не противоречат друг другу:

```
a:visited:hover
```

2. *Псевдоклассы, имеющие отношение к дереву документа.* К этой группе относятся псевдоклассы, которые определяют положение элемента в дереве документа и применяют к нему стиль в зависимости от его статуса.

- `:first-child` — применяется к первому элементу селектора, который расположен в дереве элементов документа, например:

```
b:first-child { color: red; } /* Термин выделен
красным цветом */
```

Дихотомический метод заключается в разбиении **выделенной** области решений пополам

- `:last-child` — применяется к последнему элементу селектора, расположенного в дереве элементов документа.
- `:only-child` — применяется к первому элементу селектора, расположенного в дереве элементов документа, только если он единственный.

- `:nth-child(n)` — применяется к n -му порядковому элементу селектора, расположенного в дереве элементов документа. Параметр n также может принимать значения `even` (все четные элементы), `odd` (все нечетные элементы) и `a+b`, где a и b — произвольные целые числа. Псевдокласс часто используется при оформлении таблиц.

Пример — задание цвета фона всех четных строк:

```
tr:nth-child(even) { background: #f0f0f0; }
```

- `:not(селектор)` задаёт правила стилей для элементов, которые не содержат указанный селектор.

Пример — задание цвета фона для всех абзацев, кроме именованного с атрибутом `id="main"`:

```
p:not(#main) { background: yellow; }
```

3. *Псевдоклассы, задающие язык текста.* С помощью псевдоклассов можно изменять стиль оформления иностранных текстов, а также некоторые настройки, характерные для разных языков, например вид кавычек в цитатах.

- `:lang` — определяет язык, который используется в документе или его фрагменте. На веб-странице язык устанавливается через атрибут `lang`, он обычно добавляется к тегу `<html>`:

```
Элемент:lang(язык) { свойство: значение; ... }
```

В качестве языка могут выступать следующие значения: `ru` — русский; `en` — английский; `de` — немецкий; `fr` — французский; `it` — итальянский. Пример:

```
q:lang(ru) {
  quotes: "\00AB" "\00BB"; /* Кавычки для русского
  языка */
}
```

4. *Псевдоэлементы.* Позволяют задать стиль элементов, не определенных в дереве элементов документа, а также генерировать содержимое, которого нет в исходном коде текста. Синтаксис использования псевдоэлементов:

```
Селектор::Псевдоэлемент { свойство: значение; ... }
```

Каждый псевдоэлемент может применяться только к одному селектору.

- `::after` — применяется для вставки назначенного контента после содержимого элемента. Этот псевдоэлемент работает совместно со стилевым свойством `content`, которое определяет содержимое для вставки. Например:

```
p.new::after {
  content: " - Новьё!"; /* Добавляет текст после
абзаца */
}
```

- `::before` – аналогичен псевдоэлементу `::after`, но вставляет контент до содержимого элемента. Например:

```
li::before {
  content: "\20aa "; /* Добавляет перед элементом
символ юникода */
}
```

- `::first-letter` – определяет стиль первого символа в тексте элемента, к которому добавляется. Это позволяет создавать в тексте буквицу.
- `::first-line` – определяет стиль первой строки блочного текста. Длина этой строки зависит от многих факторов, таких как используемый шрифт, размер окна браузера, ширина блока, языка и т.д.

Наследование правил и специфичность

Наследованием называется перенос правил форматирования для элементов, находящихся внутри других. Такие элементы являются дочерними, и они наследуют некоторые стилевые свойства своих родителей, внутри которых располагаются. Например, если в стилях для селектора `table` задать цвет текста, то он автоматически устанавливается для содержимого всех ячеек таблицы:

```
table { color: red; }
```

Необходимо отметить, что наследуются не все свойства элементов. Перечень наследуемых свойств элементов приведен в справочнике [5].

Приоритеты наследования (в сторону увеличения):

1. Стиль браузера.
2. Стиль автора.
3. Стиль пользователя.
4. Стиль автора с добавлением `!important`.
5. Стиль пользователя с добавлением `!important`.

Если к одному элементу одновременно применяются противоречивые стилевые правила, то более высокий приоритет имеет правило, у которого значение специфичности селектора больше.

Для определения специфичности используется следующее правило: за каждый идентификатор начисляется 100 баллов, за каждый класс и псевдокласс начисляется 10 баллов, за каждый селектор тега и псевдоэлемент начисляется 1 балл. Складывая указанные значения, получаем значение специфичности для каждого селектора.

Встроенный стиль, добавляемый к тегу через атрибут `style`, имеет специфичность 1000, поэтому всегда перекрывает связанные и глобальные стили. Однако добавление к стилю свойства `!important` перекрывает в том числе и встроенные стили.

Пример:

```
#menu ul li { color: #000; } /* специфичность 102 */
.two { color: red; } /* специфичность 10 */

<div id="menu">
  <ul>
    <li>Первый элемент</li>
    <li class="two">Второй элемент</li>
  </ul>
</div>
```

В данном примере правило `.two` имеет меньшую специфичность и перекрывается первым правилом, следовательно, оба элемента списка будут отображены черным цветом. Для выделения второго элемента можно изменить первое правило, убрав идентификатор и тем самым понизив специфичность:

```
ul li { color: #000; } /* специфичность 2 */
```

Другим способом выделения второго элемента является добавление идентификатора ко второму правилу и повышение его специфичности:

```
#menu .two { color: red; } /* специфичность 110 */
```

Так как стилей может быть много, они могут размещаться в различных местах документа и даже в разных файлах, сопоставить правила бывает непросто. Альтернативным решением является использование свойства `!important`, что сразу приводит к желаемому результату:

```
.two { color: red !important; } /* Добавляем
свойство !important */
```

@-правила используются для подключения дополнительных свойств.

- `@font-face { свойства }` – определение шрифта с заданными свойствами и его загрузка. Имя файла шрифта задается с помощью параметра `src`.

Пример:

```
@font-face {
    font-family: Pompadur; /* Имя шрифта */
    src: url(font/pompadur.ttf); /* Путь к файлу со шрифтом */
}
```

- `@media носитель { }` – определение стиля для заданного типа устройств:

Пример:

```
@media screen { /* Стил ь для отображения в браузере */
    body {
        font-family: Arial, Verdana, sans-serif; /* Рубленый шрифт */
        font-size: 12pt; /* Размер шрифта */
    }
}
```

- `@page :поле { margin: размер }` – определение размеров полей для печати страниц, заданных типом носителя `print`. Параметр `:поле` может иметь одно из трех значений:

- ◆ `:left` – для левых страниц;
- ◆ `:right` – для правых страниц;
- ◆ `:first` – для первой страницы.

Пример:

```
@page :first {
    margin: 1cm; /* Отступы для первой страницы */
}
```

- `@keyframes` или `@-webkit-frames` (для браузера Google Chrome) – определяют ключевые кадры анимации. Общий формат:

```
@keyframes <переменная> { from { стиль } to { стиль } }
@keyframes <переменная> { nn% { стиль } nn% { стиль } },
где
```

- ◆ параметр `<переменная>` связывает ключевой кадр с атрибутом `animation`;

- ♦ параметр `from` задает начальный стиль анимированного элемента (0%),
- ♦ параметр `to` – конечный стиль анимированного элемента (100%).

Начальный и конечный стили могут также устанавливаться в виде процентов.

Пример:

```
@keyframes fadeIn {
  0% { opacity: 0; } /* уровень прозрачности 0 */
  100% { opacity: 1; } /* уровень прозрачности 1 */
}
```

2.2. Стилизация текста

Параметры шрифта

Для стилизации текста могут использоваться как универсальные, так и специальные свойства, задающие характеристики шрифта и текста. По умолчанию все стили шрифта принимают значение `inherit` (наследуемое). Стиль шрифта может быть задан универсальным составным свойством `font`. Упрощенный вид свойства:

```
font: [<наклон>] [<строч>] [<толщина>] [<размер>
[/h_стр]] <имя>
```

Обязательно указывается имя шрифта.

Пример:

```
p { font: bold italic 14pt sans-serif; }
```

Для абзаца значение `bold` устанавливает жирное начертание, `italic` – курсив, размер шрифта `14pt` и семейство шрифтов `sans-serif`. Каждое из этих свойств может быть задано отдельной строкой, например:

```
p { font-style: italic;
  font-weight: bold;
  font-size: 14pt; }
```

Имена шрифтов `<имя>` устанавливаются атрибутом `font-family` в виде списка, разделенного запятыми. Браузер выбирает первый шрифт, и если он отсутствует на компьютере, выбирается следующий из списка. В случае указания семейства шрифтов выбирается один шрифт из семейства: `serif` (с засечками), `sans-serif` (без засечек), `cursive` (рукописные), `fantasy` (декоративные)

и monospace (моноширные). Если имя шрифта содержит пробелы, то оно заключается в кавычки. Пример:

```
h1 { font-family: "Times New Roman", serif }
```

Параметр `<наклон>` устанавливается атрибутом `font-style`. Возможные значения:

- normal (по умолчанию);
- italic (курсив);
- oblique (небольшой наклон).

Параметр `<строч>` устанавливается атрибутом `font-variant`. Возможные значения:

- normal (по умолчанию);
- small-caps - малые строчные буквы.

Толщина шрифта `<толщина>` устанавливается атрибутом `font-weight`. Возможные значения выражаются числами от 100 до 900 по возрастанию степени выделения, а также константами normal (400 по умолчанию) или bold (700). Также можно указать выделение с уменьшением lighter или увеличением bolder относительно толщины текущего шрифта.

Размер шрифта `<размер>` устанавливается атрибутом `font-size`. Размеры шрифтов, а также других элементов CSS можно задать в абсолютных и относительных единицах:

- px – пикселах;
- pt – типографских пунктах;
- cm – сантиметрах;
- mm – миллиметрах;
- em – размере буквы *m* текущего шрифта;
- % – в процентах от размера родительского элемента.

Всего определено 7 стандартных градаций размеров шрифта, определяемых константами: xx-small, x-small, small, medium, large, x-large и xx-large. Также можно указать размер с уменьшением smaller или увеличением larger относительно текущего шрифта.

Высота строки текста `<h_стр>` устанавливается атрибутом `line-height`. При этом можно задать как абсолютное, так относительное значение. Пример двойной высоты строки:

```
p { line-height: 2 }
```

Атрибут стиля `color` задает цвет текста. Цвет в CSS может задаваться в виде имени, полных #nnnnnn или сокращенных #nnn

шестнадцатеричных чисел, а также в виде функций `rgb(r, g, b)` или `rgba(r, g, b, a)`. Параметры `r`, `g`, `b` задают уровни красного, зеленого и синего цветов (от 0 до 255), а также уровни прозрачности `a` (от 0 до 1), например:

```
h1 { color: #f00 } /* Красный цвет. То же: red,
#ff0000, rgb(255,0,0) */
```

Уровень прозрачности также может быть задан отдельно свойством `opacity`:

```
div { opacity: 0.5 }
```

Атрибут `text-decoration` задает дополнительное оформление шрифта:

- `underline` (подчеркивание);
- `line-through` (зачеркивание);
- `none` — отмена.

Пример:

```
a:link { text-decoration: none }
```

Атрибут `text-transform` изменяет регистр символов текста:

- `uppercase` — верхний регистр;
- `lowercase` — нижний регистр;
- `capitalize` — первая заглавная буква;
- `none` — отмена стиля.

Для указания дополнительного расстояния между символами текста используется атрибут `letter-spacing`. Для отображения сжатого текста межсимвольный интервал можно указать отрицательным:

```
.cond { letter-spacing: -2px }
```

Для указания дополнительного расстояния между отдельными словами текста используется атрибут `word-spacing`:

```
.wide { word-spacing: 3px }
```

Для управления разрывом строк используется атрибут `white-space`, задающий правила отображения и переноса пробелов. Возможные значения:

- `normal` — лишние пробелы игнорируются, строки переносятся автоматически;
- `pre` — сохранение форматирования текста;
- `nowrap` — строки не переносятся;

- `pre-wrap` – сохранение форматирования, длинные строки переносятся;
- `pre-line` – сохранение только переводов строк, длинные строки переносятся.

Значение атрибута `break-word` позволяет переносить длинные слова в предложениях.

Параметры абзацев

Для блочных элементов можно задать параметры горизонтального и вертикального выравнивания. Для горизонтального выравнивания текста используется атрибут `text-align`. Возможные значения:

- `left` – по левому краю (по умолчанию);
- `right` – по правому краю;
- `center` – по центру;
- `justify` – по ширине.

Кроме этого, можно задать отступ для красной строки абзаца с помощью атрибута `text-indent`, например:

```
p { text-indent: 1cm }
```

Для вертикального выравнивания элементов используется атрибут `vertical-align`. Он определяет выравнивание текущего фрагмента текста относительно родительского элемента. Возможные значения:

- `baseline` – по базовой линии текста (по умолчанию);
- `top` – по верхней границе;
- `middle` – по центру;
- `bottom` – по нижней границе;
- `text-top` и `text-bottom` – по верхней или нижней границе текста;
- `sub` и `super` – по базовой линии нижнего или верхнего индекса.

Кроме текстовых фрагментов вертикальное выравнивание может применяться к вложенным в контейнеры `inline` элементам: изображениям, содержимому таблиц и т.д.

Для оформления тени у текста можно использовать атрибут `text-shadow`:

```
text-shadow: <цвет> <смещ_x> <смещ_y> [<размытие>]
```

Горизонтальное и вертикальное смещение может быть задано отрицательным числом, например:

```
h1 { text-shadow: gray 2px -2px 1px }
```

Параметры фона

Фон можно указывать для фрагмента текста, блочного элемента, таблицы, строки или ячейки таблицы или для всей страницы. Для задания фона элемента обычно указывается цвет или фоновое изображение.

Атрибут `background-color` устанавливает цвет фона. Значение атрибута `transparent` устанавливает прозрачный фон. По умолчанию цвет страницы задается браузером. Пример задания желтых букв на синем фоне:

```
body { color: yellow; background-color: blue }
```

Атрибут `background-image` устанавливает фоновое изображение. Возможные значения атрибута: `none` (по умолчанию) или `url(URL_адрес_изображения)`. Форматы GIF и PNG обеспечивают поддержку прозрачности.

Пример:

```
body {
  background-color: yellow;
  background-image: url("images/fon.png")
}
```

В данном случае через прозрачный фон изображения будет просвечивать желтый цвет, установленный атрибутом `background-color`.

В случае если изображение меньше, чем элемент страницы, в браузере оно будет повторяться по вертикали и по горизонтали начиная с левого верхнего угла элемента. Параметры повторения изображения задаются с помощью атрибута `background-repeat`. Возможные значения:

- `repeat` (с повторением);
- `no-repeat` (без повторения);
- `repeat-x` (повторение по горизонтали);
- `repeat-y` (повторение по вертикали).

Для указания начальной позиции изображения в элементе используется атрибут `background-position`:

```
background-position: <смещ_x> [<смещ_y>]
```

Значение `смещ_x` указывает смещение изображения относительно элемента по горизонтали в указанных единицах измерения. Также может принимать значения:

- left (слева);
- center (по центру);
- right (справа).

Значение `смещ_у` указывает смещение изображения относительно элемента по вертикали в указанных единицах измерения. Также может принимать значения:

- top (сверху);
- center (по центру);
- bottom (снизу).

В случае указания только горизонтального смещения по вертикали изображение выравнивается по центру. Пример:

```
p { background-position: 1cm top }
```

В данном примере фон абзаца фиксируется на 1 см от левого края и выравнивается по верхней границе элемента.

Для управления прокруткой фонового изображения используется атрибут `background-attachment`. Возможные значения:

- scroll (по умолчанию) – браузер прокручивает фоновое изображение вместе с содержимым страницы;
- fixed – фон фиксируется и не прокручивается.

Параметры списков

Для задания вида маркеров или нумерации пунктов списка используется атрибут `list-style-type`. Возможные значения для маркированных списков:

- disk – маркер в виде символа круга (по умолчанию);
- circle – маркер в виде символа окружности;
- square – маркер в виде квадрата.

Возможные значения для нумерованных списков:

- decimal – десятичная нумерация (по умолчанию);
- decimal-leading-zero – десятичная с начальным нулем;
- lower-alpha и lower-latin – строчные латинские буквы;
- upper-alpha и upper-latin – заглавные латинские буквы;
- lower-roman и upper-roman – строчные и заглавные римские цифры.

Для отмены маркировки и нумерации списка используется значение `none`.

Для задания графического изображения вместо маркера списка используется атрибут `list-style-image` со значением `url (URL_адрес_изображения)`.

Пример:

```
ul { list-style-image: url("images/ball.png") }
```

2.3. Табличная и блочная верстка

Текстовая и фреймовая верстка

На заре Интернет, до 1995 года, использовалась только *текстовая* верстка. HTML страницы представляли собой набор текстовых абзацев с редкими вкраплениями изображений и таблиц. Навигация осуществлялась с помощью гиперссылок, размещаемых в начале и в конце каждой страницы. Такие страницы достаточно быстро загружались в браузере, но при размещении большого объема информации быстро стали неудобными. Элементы навигации потребовалось размещать таким образом, чтобы они всегда оставались доступными, а обновлялось бы только содержимое.

В 1995 году в браузере Netscape Navigator появилась поддержка *фреймов*. При использовании фреймовой верстки экран браузера делился на две или три части и в каждую из них загружался отдельный HTML документ, содержащий, например, логотип сайта, боковое меню или содержимое – контент. Достоинство использования фреймов – увеличение скорости загрузки страниц, так как в части фреймов содержимое не обновлялось. В качестве недостатка можно отметить громоздкость и негибкость конструкции, невозможность изменения заданной разбивки страницы.

Табличная верстка

С ростом скорости Интернет и появлением средств стилизации необходимость во фреймовой верстке отпала и в начале XXI века его полностью заменила *табличная* верстка. Пример табличного дизайна – оформление сайта в виде газетной полосы. Текст разбивается на две или три колонки и стилизуется. Для позиционирования ячеек таблиц используется фиксированная или плавающая верстка. При этом активно используются вложенные таблицы с невидимыми границами.

Фиксированная табличная верстка обеспечивается с помощью фиксации размера таблиц и ячеек. Пример фиксированной табличной верстки:

```

<table>
  <tr colspan="2">
    <td height="80">Заголовок</td>
  </tr>
  <tr>
    <td width="250">Меню</td>
    <td width="500">
      <table border="0">
        <tr>
          <td width="200">Статьи</td>
          <td width="200">Новости</td>
        </tr>
      </table>
      Контент</td>
    </tr>
  </table>

```

Результат выполнения примера представлен на рисунке 21.

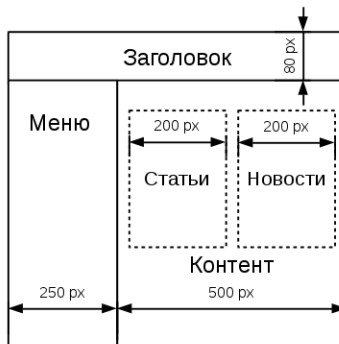


Рис. 21. Фиксированная табличная верстка

Достоинства фиксированной верстки:

- сайт одинаково выглядит во всех браузерах;
- можно позиционировать содержимое как угодно, даже разрезать изображения и загружать их в разные ячейки таблицы.

Недостаток фиксированной верстки – при использовании на экранах с большим разрешением остается много свободного места, а на маленьких экранах, наоборот, может появиться прокрутка. Альтернативой фиксированной верстки, лишенной этого недостатка, явилась плавающая верстка.

Плавающая табличная верстка обеспечивается с помощью фиксации размера таблиц в процентах и лишь некоторых ячеек в пикселах, например высоты логотипа и ширины меню. При этом размеры оставшихся ячеек автоматически раздвигаются до размера родительского элемента, который определяется размером окна браузера. Плавающая табличная верстка многие годы являлась стандартом сайтостроения, пока не была вытеснена более современной блочной версткой. Пример плавающей табличной верстки:

```
<table width="100%">
  <tr colspan="2">
    <td height="80">Заголовок</td>
  </tr>
  <tr>
    <td width="250">Меню</td>
    <td>
      <table border="0" width="100%">
        <tr>
          <td>Статьи</td>
          <td width="200">Новости</td>
        </tr>
      </table>
      Контент</td>
    </tr>
  </table>
```

Результат выполнения примера представлен на рисунке 22.



Рис. 22. Плавающая табличная верстка

Недостатки табличного дизайна:

- большой объем и время загрузки страниц, содержащих большое количество однотипных табличных тегов;

- сложный и избыточный табличный код плохо поддается редактированию и индексации поисковыми роботами.

Блочная верстка

С развитием стилизации и стандарта CSS появилась возможность использовать *блочную* верстку практически для всех элементов страниц. Это существенно упростило структуру сайтов и позволило отделить содержание страниц от их представления. Основным структурным элементом блочной верстки являются контейнерные элементы `div`. С помощью стилей обеспечивается выравнивание и позиционирование контейнеров относительно друг друга.

Вышеуказанный пример, оформленный с помощью блочной верстки:

```
header { height: 80px; } /* высота 80 px */
.menu { width: 250px; float: left; } /* ширина 250 px,
отекание слева */
.content { padding-left: 250px; } /* отступ слева 250 px
*/
.news { width: 200px; float: right; } /* ширина 200 px,
отекание справа */
.articles { padding-right: 200px; } /* отступ справа 200
px */
```

```
<body>
  <header>Заголовок</header>
  <div class="menu">Меню</div>
  <div class="content">
    <div class="news">Новости</div>
    <div class="articles">Статьи</div>
  </div>
</body>
```

Параметры размеров

Для задания размеров блочных элементов используются атрибуты `width` и `height`, задающие соответственно ширину и высоту элемента. По умолчанию ширина и высота элемента определяются его содержимым (значение `auto`). Можно задать относительное значение размера в процентах от родительского элемента. Если размер родительского элемента не задан, то он вычисляется от размера окна браузера. Например:

```
.header { height: 10% }
```

При задании относительных размеров элементов может возникнуть необходимость указать для них минимальные или максимальные возможные размеры. Для указания минимальной ширины и высоты используются атрибуты `min-width` и `min-height`. Для указания максимальной ширины и высоты используются атрибуты `max-width` и `max-height`. Значения данных атрибутов также могут быть как абсолютными, так и относительными

Параметры размещения

Местоположение блочных элементов относительно друг друга определяется атрибутом `float`. Он задает способ выравнивания текущего блока относительно родительского элемента (см. рисунок 23). Возможные значения:

- `none` – блочные элементы следуют друг за другом сверху вниз, без обтекания (по умолчанию);
- `left` – блок выравнивается по левому краю родительского элемента, остальное содержимое обтекает его справа;
- `right` – блок выравнивается по правому краю родительского элемента, содержимое обтекает его слева. Данный атрибут применяется к нескольким элементам для организации плавающей верстки, когда элементы выстраиваются по горизонтали.

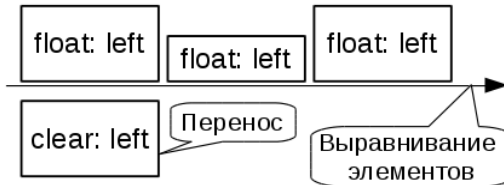


Рис. 23. Способы выравнивания текущего блока относительно родительского элемента

Если необходимо явно указать, что текущий элемент будет располагаться ниже других, имеющих установленный атрибут `float`, к такому элементу применяется атрибут `clear`. Возможные значения:

- `none` – выравнивание не меняется (по умолчанию);
- `left` – текущий элемент располагается ниже всех элементов, имеющих значение `float: left`;
- `right` – текущий элемент располагается ниже всех элементов, имеющих значение `float: right`;

- `both` – текущий элемент располагается ниже всех элементов, имеющих установленное значение `left` или `right` атрибута `float`.

Пример стиля для тега `<footer>`:

```
footer { clear: both }
```

Параметры переполнения

В случае когда содержимое не помещается в установленный размер элемента, возникает переполнение. Поведение отображения содержимого при переполнении определяется атрибутом `overflow`. Возможные значения:

- `visible` – автоматически увеличится высота элемента (по умолчанию);
- `hidden` – не помещающееся содержимое элемента будет обрезано. Высота элемента останется прежней;
- `scroll` – элемент будет иметь полосы прокрутки для отображения непомогающегося содержимого;
- `auto` – если возникнет переполнение содержимого, то появятся полосы прокрутки.

Необходимо отметить, что атрибут `overflow` устанавливается, когда высота элемента `height` указана в абсолютных единицах. При использовании относительных размеров высоты переполнение не работает.

Атрибуты `overflow-x` и `overflow-y` задают поведение отображения при переполнении содержимого элемента по горизонтали или вертикали соответственно. Значения этих атрибутов аналогичны `overflow`.

Параметры отступов

В стандарте CSS используются два вида отступов (см. рисунок 24):

- внутренний отступ, между содержимым и внешней границей элемента;
- внешний отступ, между элементом и соседними блочными элементами.

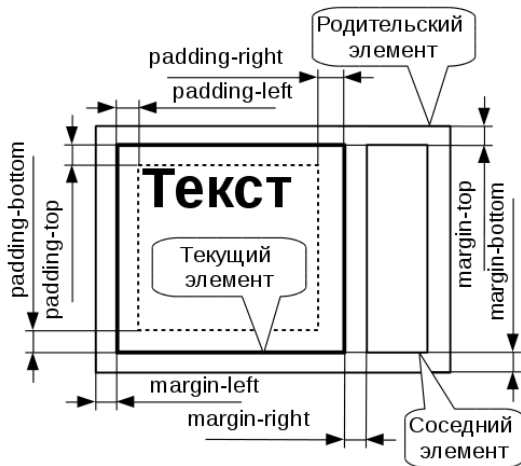


Рис. 24. Отступы в CSS

Величина внутренних отступов определяется свойством padding:

```
padding: <отступ1> [<отступ2>] [<отступ3>] [<отступ4>]
```

Если задан один параметр <отступ1>, то он применяется ко всем сторонам элемента; если два параметра, то первый указывает верх-низ, второй – правую-левую стороны; если три – первый указывает верх, второй – правую-левую стороны, третий – низ; если четыре параметра – первый указывает верх, второй – правую сторону, третий – низ, четвертый – левую сторону. Пример:

```
.indented { padding: 0cm 2cm 2cm 2cm }
```

Атрибуты внутренних отступов могут быть указаны отдельно: padding-left, padding-top, padding-right, padding-bottom. Пример:

```
.indented { padding-left: 2cm }
```

Величина внешних отступов определяется свойством margin:

```
margin: <отступ1> [<отступ2>] [<отступ3>] [<отступ4>]
```

Параметры внешних отступов атрибута margin аналогичны параметрам внутренних отступов padding. Пример задания отступов сверху и снизу:

```
h1 { margin: 5mm 0mm }
```

Атрибуты внешних отступов могут быть указаны отдельно: `margin-left`, `margin-top`, `margin-right`, `margin-bottom`.

Например:

```
h1 { margin-top: 5mm }
```

Необходимо отметить, что для таблицы внешние отступы ячеек задаются отдельным атрибутом `border-spacing` вида:

```
border-spacing: <отступ1> [<отступ2>]
```

Если задан один параметр `<отступ1>`, то он применяется ко всем сторонам ячейки таблицы; если два параметра, то первый указывает отступ слева-справа, второй – верх-низ.

Параметры рамок

Для задания рамки вокруг блочного элемента используется атрибут `border`:

`border: <толщина> <стиль> <цвет>`, например:

```
td { border: 1px solid black }
```

Толщина рамки `<толщина>` задается атрибутом `border-width`, имеющим вид:

```
border-width: <толщина1> [<толщина2>] [<толщина3>]
[<толщина4>]
```

Параметры толщины атрибута `border-width` применяются к границам рамки в последовательности, аналогичной параметрам отступов. Значения толщины могут задаваться одной из констант:

- `thin` – тонкая;
- `medium` – средняя;
- `thick` – толстая или одним из размерных значений CSS.

Пример задания тонкой границы рамки сверху и снизу:

```
h1 { border-width: thin 0 }
```

Атрибуты толщины границы также могут быть указаны отдельно:

- `border-left-width` – для левой границы;
- `border-top-width` – для верхней границы;
- `border-right-width` – для правой границы;
- `border-bottom-width` – для нижней границы.

Например, для нижней границы рамки:

```
h1 { border-bottom-width: 2px } /* толщина 2px */
```

Стиль рамки <стиль> задается атрибутом `border-style`, имеющим вид:

`border-style: <стиль1> [<стиль2>] [<стиль3>] [<стиль4>]`

Параметры стиля атрибута `border-style` применяются к границам рамки в последовательности, аналогичной параметрам отступов. Значения стиля могут задаваться одной из констант:

- `none` или `hidden` – рамка отсутствует (по умолчанию);
- `solid` – сплошная линия;
- `dotted` – пунктирная линия;
- `dashed` – штриховая линия;
- `double` – двойная линия;
- `groove` и `ridge` – вдавленная или выпуклая 3D линия;
- `inset` и `outset` – вдавленная или выпуклая 3D плоскость.

Пример задания выпуклой 3D границы рамки вокруг всего элемента:

```
h1 { border-style: ridge }
```

Атрибуты стиля границы также могут быть указаны отдельно:

- `border-left-style` – для левой границы;
- `border-top-style` – для верхней границы;
- `border-right-style` – для правой границы;
- `border-bottom-style` – для нижней границы.

Например, для нижней границы рамки:

```
h1 { border-bottom-style: solid } /* сплошная линия */
```

Цвет рамки <цвет> задается атрибутом `border-color`, имеющим вид:

`border-color: <цвет1> [<цвет2>] [<цвет3>] [<цвет4>]`

Параметры цвета атрибута `border-color` применяются к границам рамки в последовательности, аналогичной параметрам отступов. Пример:

```
h1 { border-color: lightgreen } /* светло-зеленый
цвет */
```

Атрибуты цвета границы также могут быть указаны отдельно:

- `border-left-color` – для левой границы;
- `border-top-color` – для верхней границы;
- `border-right-color` – для правой границы;
- `border-bottom-color` – для нижней границы.

Например, для нижней границы рамки:

```
h1 { border-bottom-color: #0F0 } /* светло-зеленый
цвет */
```

Радиус закругления углов рамки указывается с помощью атрибута `border-radius`, имеющего вид:

```
border-radius: <r1> [<r2>] [<r3>] [<r4>]
```

Параметры радиусов атрибута `border-radius` применяются к границам рамки в последовательности, аналогичной параметрам отступов, начиная с верхнего левого угла. Каждый параметр может быть как числом, указывающим радиус круга, так и числовой дробью, указывающей размеры полуосей эллипса. Размеры могут быть заданы как в размерных единицах CSS, так и в процентах от ширины блока. Пример:

```
h1 { border-radius: 2px/3px } /* радиус углов рамки
2 на 3 пиксела */
```

Параметры выделения

Для задания выделения блочного элемента используется атрибут `outline`. Выделение представляет собой границу, окружающую данный элемент. В отличие от рамки, выделение не увеличивает размер элемента. Параметры выделения такие же, как и параметры рамок.

`outline: <толщина> <стиль> <цвет>`, например:

```
p { outline: thin dotted black }
```

Для задания параметров выделения также можно использовать отдельные свойства: `outline-width`, `outline-color`, `outline-style`. Параметры свойств выделения аналогичны параметрам свойств рамок.

Параметры таблиц

Для выравнивания содержимого ячеек таблиц по горизонтали и по вертикали используются ранее рассмотренные атрибуты `text-align` и `vertical-align`. В отличие от блоков, при установке атрибута `vertical-align` для ячеек таблицы обычно используются три типа выравнивания содержимого:

- `top` – по верхней границе;
- `middle` – по центру;
- `bottom` – по нижней границе.

Для установки внутренних отступов для содержимого ячеек используется атрибут `padding` или его производные: `padding-left`, `padding-top`, `padding-right` и `padding-bottom`.

Для задания внешних отступов между границами ячеек используется атрибут `border-spacing`. Атрибут `margin` задает внешние отступы для таблицы в целом. Аналогично атрибут `align` задает выравнивание таблицы в целом как блочного элемента относительно родительского элемента (или страницы). Например, выравнивание таблицы по центру задается стилем:

```
table { align: center }
```

Для задания границ ячеек используются атрибут `border` или его производные. Необходимо отметить, что атрибут `border` может применяться отдельно как к таблице в целом, так и для отдельных ячеек. В этом случае каждая ячейка заключается в отдельную рамку. Для объединения рамок ячеек в таблице используется специальный атрибут `border-collapse`. Допустимые значения: `separate` – отдельная рамка (по умолчанию), `collapse` – объединение рамок ячеек в одну сетку.

Для задания размера ячеек используются стандартные атрибуты ширины и высоты: `width` и `height`. Эти же атрибуты применимы и к таблице в целом.

По умолчанию размеры ячеек и таблицы в целом определяются их содержимым. Для изменения этого поведения применительно к таблице используется атрибут `table-layout`. Возможные значения: `auto` – размер таблицы и ячеек определяется содержимым (по умолчанию), `fixed` – размеры таблицы и ячеек будут установлены фиксированными.

Если при задании фиксированных размеров элемента его содержимое не будет умещаться, то возникнет переполнение. Поведение отображения содержимого при переполнении ячеек таблицы задается при помощи вышеуказанных атрибутов `overflow`, `overflow-x` и `overflow-y`.

Атрибут `caption-side` устанавливает местоположение заголовка таблицы относительно его содержания. Возможные значения: `top` – заголовок сверху (по умолчанию), `bottom` – заголовок снизу. Атрибут применяется к тегу `<table>`.

Атрибут `empty-cells` указывает поведение при отображении пустых ячеек таблицы. Возможные значения: `show` – пустые ячейки выводятся на экран вместе с их оформлением, `hide` – пустые ячейки

не будут выводиться на экран. Атрибут может применяться также к таблице целиком.

2.4. Видимость и эффекты

Параметры курсора

Форма и вид курсора, отображаемого при наведении на определенный элемент страницы, устанавливаются с помощью атрибута `cursor`. Наиболее часто используемые значения:

- `auto` – устанавливаемое автоматически браузером (по умолчанию);
- `default` – указатель стрелка;
- `none` – курсор не отображается;
- `help` – стрелка с вопросительным знаком;
- `pointer` – указующий перст (используется при наведении на гиперссылку);
- `progress` – стрелка с песочными часами (используется при ожидании);
- `wait` – песочные часы (используется при ожидании);
- `text` – текстовый курсор (используется при вводе текста).

Параметры и вид отображения

Вид отображения элемента устанавливается с помощью атрибута `display`. Это многоцелевое свойство, которое определяет, как элемент должен быть показан в документе. Оно устанавливает, будет ли элемент блочным, строчным, строчно-блочным и т.д. Возможные значения:

- `inline` – строчный элемент (по умолчанию для строчных элементов);
- `block` – блочный элемент (по умолчанию для блочных элементов);
- `none` – не отображается, место под него не резервируется;
- `inline-block` – строчно-блочный элемент, комбинирует внутреннее поведение блочного и внешнее поведение строчного элемента;
- `list-item` – элемент списка;
- `table` – таблица;
- `inline-table` – таблица с внешним поведением строчного элемента;
- `table-caption` – заголовок таблицы;

- `table-column` — столбец таблицы;
- `table-row` — строка таблицы;
- `table-cell` — ячейка таблицы;
- `table-header-group` — секция заголовка таблицы;
- `table-row-group` — секция содержания таблицы;
- `table-footer-group` — секция подвала таблицы;
- `table-column-group` — группа столбцов таблицы.

Наиболее часто используется свойство `none`, позволяющее делать элемент невидимым:

```
.hidden { display: none }
```

Значения атрибута `display`, начинающиеся с `table-`, позволяют использовать теги `<div>` в качестве элементов таблицы.

Параметры отображения элемента также могут устанавливаться с помощью атрибута `visibility`. Применяются ко всем элементам страницы. Возможные значения:

- `visible` — видимый (по умолчанию);
- `hidden` — скрытый;
- `collapse` — скрывает заданные строки и столбцы таблицы.

Позиционирование

Позиция блочного элемента на странице обеспечивается его поведением, заданным с помощью атрибута `position`. Возможные значения:

- `static` — блок непозиционируемый (по умолчанию);
- `absolute` — абсолютно позиционируемый, место на странице не выделяется. Если родительский элемент позиционирован, то текущий элемент позиционируется от него, иначе — от края экрана;
- `fixed` — подобный абсолютному позиционированию, но не меняет позицию элемента при прокрутке содержимого родительского элемента;
- `relative` — относительно позиционируемый, положение элемента устанавливается относительно его исходного места, место под элемент выделяется.

Для установки позиции элемента используются атрибуты `top`, `left`, `right` и `bottom`, устанавливающие расстояние от верхней, левой, правой или нижней границы текущего элемента до соответствующей границы родительского элемента или экрана. Значение `auto` вышеуказанных атрибутов не изменяет положение

текущего элемента. Пример – левый верхний угол элемента с `id="search"` на экране имеет координаты (200, 100):

```
#search { position: absolute; left: 200px; top: 100px }
```

Перекрытие и область видимости

По умолчанию позиционированный элемент, определенный позже, перекрывает элементы, определенные раньше него. Это поведение можно изменить с помощью атрибута `z-index`. Значение атрибута устанавливается целым числом, определяющим порядок перекрытия. Элементы с большим значением `z-index` перекрывают элементы с меньшим значением. Значение `auto` определяет порядок перекрытия по умолчанию. Пример:

```
#search { z-index: 2 }
```

Область видимости позиционированного элемента определяется атрибутом `clip` (см. рисунок 25), задающим прямоугольник маски и имеющим вид:

```
clip: rect(<верх>,<право>,<низ>,<лево>)
```

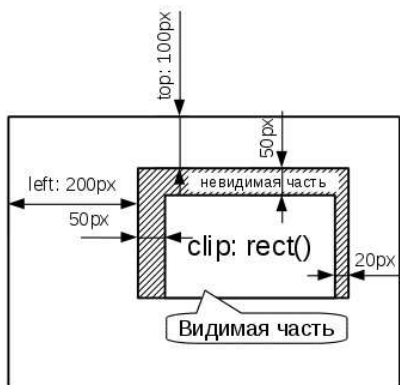


Рис. 25. Область видимости позиционированного элемента

Параметры `<верх>`, `<право>`, `<низ>` и `<лево>` определяют расстояние от верхней, правой, нижней и левой границы текущего элемента до области маски.

Прямоугольник маски задает видимую область элемента. Значение `auto` атрибута `clip` убирает маску и делает видимым весь элемент. Пример:

```
#search { position: absolute;
  left: 200px; top: 100px
  clip: rect(50px, 20px, 0px, 50px)
}
```

Градиенты

Функция `linear-gradient()` добавляет линейный градиент к фону элемента. Она выступает значением свойства `background-image` или `background`. Вид:

```
background-image: linear-gradient( <направл>,
<цветн>, <цветк> );
```

Параметры `<цветн>` и `<цветк>` задают начальный и конечный цвет градиента с необязательной позицией цвета относительно оси градиента в процентах.

Параметр `<направл>` задает направление градиента. Направление может задаваться с помощью угла наклона `deg`, отсчитываемого от вертикали по часовой стрелке, например `45deg`, или с помощью позиции, определяемой константой `to` и комбинациями `left`, `right`, `top` и `bottom`, например `to top` (или `0deg`) – снизу вверх; `to left` (или `270deg`) – справа налево; `to top right` (или `45deg`) – от левого нижнего угла к правому верхнему.

Пример:

```
background-image: linear-gradient( to top, #fefcea,
#f1da36 );
```

Трансформации

Трансформации позволяют выполнять над элементом различные геометрические преобразования: масштабирование, переносы, повороты и др. Для этого используется атрибут `transform` (для браузера Google Chrome и Android используется атрибут `webkit-transform`, для браузера Internet Explorer – атрибут `-ms-transform`). Вид атрибута `transform`:

```
transform: <функция>
```

Параметр `<функция>` определяет функцию трансформации. Для отмены трансформации используется значение `none`. Виды функций трансформации:

- `rotate(<угол>)` – поворот элемента на заданный угол по часовой стрелке относительно точки привязки, заданной дополнительным атрибутом `transform-origin`. Обычно значения атрибута определяются координатами, задаваемыми в виде двух чисел или констант: `left`, `center`,

right — по горизонтали и top, center, bottom — по вертикали. Например:

```
.box { transform-origin: left, top; /* левый
верхний угол элемента */
transform: rotate(45deg); } /* поворот на 45
градусов */
```

- scale(sx [,sy]) — масштабирование элемента по горизонтали и вертикали. Значения sx и sy задаются в относительных единицах — текущий размер элемента принимается за 1. Функции scaleX(sx) и scaleY(sy) задают отдельное масштабирование элемента по горизонтали или вертикали;
- skewX(<угол>) и skewY(<угол>) — наклон элемента на заданный угол;
- translate(dx [,dy]) — перенос элемента по горизонтали и вертикали на заданное значение dx и dy. Функции translateX(dx) и translateY(dy) задают отдельный перенос элемента по горизонтали или вертикали. Значения dx и dy могут быть отрицательными:

```
.box { transform: translate(-2px) } /* смещение
влево на 2px */
```

Анимация по кадрам

К элементам страниц можно применить вид анимации по ключевым кадрам. Для этого используется атрибут animation (для браузера Google Chrome используется атрибут -webkit-animation), который работает в паре с правилом @keyframes (для браузера Google Chrome используется правило @-webkit-keyframes). Общий вид атрибута animation:

```
animation: <переменная> [<время>] [<функция>]
[<задержка>] [<итер>][<направление>] [<режим>]
[<состояние>]
```

Кроме параметра <переменная>, каждый из этих параметров может быть в любой комбинации установлен с помощью отдельных атрибутов: animation-duration, animation-timing-function, animation-delay, animation-iteration-count, animation-direction, animation-fill-mode и animation-play-state.

Параметр <переменная> связывает анимацию с правилом @keyframes:

```
.fadeIn { animation: fadeIn 3s; }
```

```
@keyframes fadeIn { from { opacity: 0; } to {
opacity: 1; } }
```

Таким образом, элемент со стилем `.fadeIn` будет изменять свою прозрачность в течение 3 секунд.

Параметр `<время>` задается атрибутом `animation-duration` и определяет время действия цикла анимации в секундах `s` или миллисекундах `ms`, например `animation-duration: 3s` — три секунды.

Параметр `<функция>` задается атрибутом `animation-timing-function` и определяет временную функцию анимирования. Возможные значения:

- `ease` — ускорение в середине (по умолчанию);
- `ease-in` — постепенное ускорение;
- `ease-out` — постепенное замедление;
- `ease-in-out` — медленное начало и конец;
- `linear` — линейная скорость;
- `step-start` и `step-end` — начальное или конечное состояние без анимации;
- `steps(n, start или end)` — ступенчатая функция с заданным числом шагов;
- `cubic-bezier(a, b, c, d)` — кривая Безье.

Параметр `<задержка>` задается атрибутом `animation-delay` и определяет время ожидания перед выполнением анимации (по умолчанию 0). Время задается в секундах `s` или миллисекундах `ms`.

Параметр `<итер>` задается атрибутом `animation-iteration-count` и определяет число итераций (1 по умолчанию). Бесконечное число итераций задается константой `infinite`.

Параметр `<направление>` задается атрибутом `animation-direction` и определяет направление движения анимации. Возможные значения:

- `normal` — с начала до конца и возврат в исходное состояние (по умолчанию);
- `alternate` — с начала до конца и плавный возврат в исходное состояние;
- `reverse` — с конца до начала;
- `alternate-reverse` — с конца до начала и плавный возврат в исходное состояние.

Параметр `<режим>` задается атрибутом `animation-fill-mode` и определяет состояние элемента после окончания анимации. Возможные значения:

- none – стили не применяются, по умолчанию элемент будет находиться в исходном состоянии;
- forwards – к элементу применяется стиль последнего ключевого кадра;
- backwards – к элементу применяется стиль первого ключевого кадра.

Параметр <состояние> задается атрибутом `animation-play-state` и определяет состояние анимации. Возможные значения: `running` – проигрывать анимацию (по умолчанию); `paused` – приостановить анимацию.

Анимировать можно различные свойства стилей: прозрачность, размер, цвет, положение элементов. Это придает элементам страницы динамичность и современный вид.

Анимация переходов

При наведении курсора мыши можно использовать анимацию переходов. Обычно при использовании псевдокласса `:hover` изменение происходит мгновенно, атрибут `transition` позволяет задать продолжительность и способ перехода (для Android используется атрибут `-webkit-transition`). Общий вид:

```
transition: <свойство> [<время>] [<функция>]
[<задержка>]
```

Кроме того, каждый из параметров может быть в любой комбинации установлен с помощью отдельных атрибутов: `transition-property`, `transition-duration`, `transition-timing-function`, `transition-delay`.

Параметр <свойство> задается атрибутом `transition-property` и определяет свойство элемента, подлежащего анимации, например: `top`. Значение `none` не задает ни одно свойство, значение `all` относится ко всем свойствам. В случае анимации нескольких свойств они перечисляются через запятую.

Параметр <время> задается атрибутом `transition-duration` и определяет время действия перехода аналогично атрибуту `animation-duration`.

Параметр <функция> задается атрибутом `transition-timing-function` и определяет временную функцию перехода аналогично атрибуту `animation-timing-function`.

Параметр <задержка> задается атрибутом `transition-delay` и определяет время ожидания перед выполнением перехода аналогично атрибуту `animation-delay`. Пример анимации перехода для стиля меню:

```
.menu {  
  position: absolute;  
  top: -100px;  
  transition: top 2s;  
}  
.menu:hover { top: 0; }
```


Библиографический список

1. Дронов В.А. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов. – СПб.: БХВ-Петербург, 2011. – 416 с.
2. Дунаев В.В. HTML, скрипты и стили. – 3-е изд. – СПб.: БХВ-Петербург, 2011. – 816 с.
3. Прохоренок Н.А., Дронов В.А. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера. – СПб.: БХВ-Петербург, 2010. – 912 с.
4. Хоган Б. HTML 5 и CSS 3. Веб-разработка по стандартам нового поколения. – СПб.: Питер, 2012. – 270 с.
5. Справочник по HTML и CSS. [Электронный ресурс]. URL: [http:// http://htmlbook.ru/](http://htmlbook.ru/) (дата обращения: 01.03.2016).

Оглавление

1. Обзор стандарта HTML 5.....	1
1.1. Языки гипертекстовой разметки.....	1
1.2. Элементы оформления текста	4
1.3. Таблицы	12
1.4. Графика.....	15
1.5. Формы и элементы управления	16
2. Каскадные стилевые таблицы CSS 3.....	26
2.1. Синтаксис CSS правил	26
2.2. Стилизация текста	38
2.3. Табличная и блочная верстка	44
2.4. Видимость и эффекты.....	55
Библиографический список	63