**Раздел 1.**

**Технология создания Web-документов**

* 1. **История возникновения HTML. HTML-редакторы. Структура HTML-документа. Физическое и логическое форматирование текста. Ссылки. Использование графики**
  2. **Списки. Таблицы. Блоки. Изображения-карты**
  3. **Формы. Фреймы. Таблицы стилей**
  4. **История возникновения HTML. HTML-редакторы. Структура HTML-документа. Физическое и логическое форматирование текста. Ссылки. Использование графики**

В этой лекции мы познакомимся с основами языка *HTML* — что это такое, что он делает, его историей вкратце, и как выглядит структура документа *HTML*. Следующие лекции будут рассматривать каждую отдельную часть *HTML* со значительно большей глубиной. Лекция имеет следующую структуру:

* Что такое HTML
* Как выглядит HTML
* История HTML
* Структура документа HTML
* Синтаксис элементов HTML
* Элементы блочного уровня и строковые элементы
* Символьные ссылки
* Физическое и логическое форматирование текста
* Ссылки
* Использование графики

**Что такое HTML**

Большинство настольных приложений, которые читают и записывают файлы, используют специальный формат файлов. Например, Microsoft*Word* понимает файлы ".doc", а Microsoft *Excel* понимает ".xls". Эти файлы содержат инструкции о том, как восстановить документ, когда вы откроете его в следующий раз, каково содержимое документа, и "*метаданные*" статьи, такие как *автор*, дата последней модификации документа, даже такие вещи, как *список* сделанных изменений, чтобы можно было перемещаться между различными версиями.

***HTML*** ("*Язык разметки* гипертекста") является языком описания содержимого документов *Web*. Он использует специальный *синтаксис*, содержащий маркеры (называемые " *элементами* "), которые охватывают текст в документе, чтобы указать, как агенты пользователя должны интерпретировать эту часть документа.

Здесь используется технический термин "агенты пользователя", а не "браузеры *Web*". *Агент* пользователя является любым программным продуктом, который используется для доступа к страницам *Web* от имени пользователей. Здесь необходимо сделать важное различие — все типы программ браузеров настольных компьютеров (*Internet Explorer*, Opera, Firefox, Safari, и т.д.) и *альтернативные* браузеры для других устройств (такие как *Интернет*-канал Wii и браузеры мобильных телефонов, такие как Opera Mini и WebKit на iPhone) являются агентами пользователей, но не все агенты пользователей являются программами-браузерами. Автоматические программы, которые компании Google и Yahoo! используют для индексирования *Web* для использования в своих поисковых системах, также являются агентами пользователей, но ни одно человеческое существо не управляет ими непосредственно.

**Как выглядит HTML**

*HTML* является просто обычным текстовым представлением содержимого и его общего смысла. Например, код заголовка "Что такое *HTML* " выше выглядит следующим образом:

<h2>Что такое HTML</h2>

Часть "<h2>" является маркером (который мы называем " ***тегом*** "), который означает "то, что следует далее, должно рассматриваться, как заголовок второго уровня". "</h2>" является *тегом* для указания, где находится окончание заголовка второго уровня (который мы называем "закрывающий *тег* "). Открывающий *тег*, закрывающий *тег* и все, что между ними называются " ***элементом*** ". Однако многие люди используют термины *элемент* и *тег* взаимозаменяемо, что не совсем строго правильно.

В большинстве браузеров имеется *функция* "Исходный текст" или "Просмотр исходного текста", обычно в *меню* "Вид". Если такая *функция*имеется, выберите ее и уделите некоторое время изучению исходного кода *HTML* страницы.

**История HTML**

Хотя *HTML* значительно изменился с тех пор, большая часть содержимого современного *HTML* реализована в этом первом документе и более половины " *тегов* ", описанных в исходном документе " *Теги* *HTML* " *по* прежнему существуют.

*По* мере того как все больше людей начали создавать *Web*-страницы и *альтернативные* варианты исходной программы браузера, все больше свойств добавлялось в *HTML*. Многие были признаны повсеместно (такие как *элемент* img, используемый для вставки изображения в документ, реализованный впервые в браузере *NCSA* *Mosaic*). Некоторые были более частными и использовались реально только в одном или двух браузерах. Возникла настоятельная потребность в стандартизации — чтобы авторы других программ *Web*-браузеров имели документ (называемый "спецификацией"), который окончательно описывал для них, как выглядит *HTML*, чтобы они могли оценить, пропустили они или нет при реализации некоторые части *HTML*.

*IETF* (*Internet Engineering Task Force*- орган стандартизации, занимающийся взаимодействием в *Интернет*) опубликовал *черновик* предложения*по* стандарту *HTML* в 1993. Он утратил силу, не став стандартом в 1994 г., но стимулировал *IETF* на создание рабочей группы для создания стандарта *HTML*.

В 1995 г. был написан " *HTML* 2.0", использовавший идеи из исходного черновика *HTML*. Дейв Раггетт написал также альтернативное предложение, названное *HTML* +, которое использовалось в качестве основы для многих новых *элементов*, реализованных в браузерах (такие как метод вставки изображений в документы, предложенный впервые в браузере *NCSA* *Mosaic*).

*Черновик* *HTML* 3.0 появился позже в этом же году, но работа над этой версией была прервана, в связи с отсутствием поддержки со стороны производителей браузеров. *HTML* 3.2 потерял многие новые свойства 3.0, и вместо этого принял многие разработки популярных в то время браузеров *Mosaic* и Netscape Navigator.

В 1997 консорциум *W3C* опубликовал версию *HTML* 4.0 в качестве рекомендации, которая включила дополнительные специальные расширения браузеров, но попыталась также рационализировать и очистить *HTML*. Это было сделано, помечая различные *элементы* как исключенные — что означает, что *элементы* устарели и хотя они еще существуют в этой версии, они будут удалены в последующем. Это должно было стимулировать лучшее и более семантическое использование *HTML* в документах Версия *HTML* 4.01 была опубликована в 1999 г., и с некоторыми исправлениями опечаток в 2001 г. Это самая последняя версия *HTML*, хотя в настоящее время готовится *черновик* версии *HTML* 5.

В 2000 г., консорциум *W3C* опубликовал также спецификацию *XHTML* 1.0, которая была реструктуризацией *HTML*, чтобы сделать его действительным документом *XML*.

**Структура документа HTML**

Минимальный возможный *действительный* документ *HTML* будет выглядеть примерно следующим образом:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

<title>Example page</title>

</head>

<body>

<h1>Hello world</h1>

</body>

</html>

Документ начинается с *элемента* типа документа, или doctype . Он описывает, какой тип *HTML* будет использоваться — чтобы агенты пользователя могли определить, как интерпретировать документ, и решить, следует ли он тем правилам, которым собирался следовать *по* своему заявлению.

После этого можно видеть открывающей *тег* *элемента* html. Это *оболочка* вокруг всего документа. Закрывающий *тег* html является последним объектом в любом документе *HTML*.

Внутри *элемента* html имеется *элемент* head. Он является оболочкой, содержащей информацию о документе (*метаданные*). Это описано более подробно в следующей лекции. Внутри head находится *элемент* title, который определяет заголовок "*Example* page" в панели *меню*.

После *элемента* head следует *элемент* body, который является оболочкой, содержащей реальное содержимое страницы — в данном случае только *элемент* заголовка первого уровня ( h1 ), который содержит текст "Hello world". И это, собственно, и есть весь документ.

Как можно видеть, *элементы* часто содержат другие *элементы*. *Тело документа* всегда будет содержать множество вложенных друг в друга *элементов*.

*Разделы* страницы создают общую структуру документа, и будут содержать подразделы. Они будут содержать *заголовки*, параграфы, списки и т.д. Параграфы могут содержать *элементы*, которые создают ссылки на другие *элементы*, цитаты, выделения и т.д. Больше об этих *элемента х* будет сказано в дальнейшем.

**Синтаксис элементов HTML**

Как вы уже видели, базовый *элемент* в *HTML* состоит из двух маркеров вокруг блока текста. Существуют *элементы*, которые не являются оболочкой для текста, и почти в каждом случае *элементы* могут содержать подэлементы (как html содержит head и body в примере выше).

*Элементы* могут также иметь *атрибуты*, которые могут модифицировать поведение *элемента* и вводить дополнительное *значение*.

<div id="masthead">

<h1>The Basics of

<abbr title="Hypertext Markup Language">HTML</abbr>

</h1>

</div>

В этом примере *элемент* div (раздел страницы, способ разбиения документов на логические блоки) имеет добавленный *атрибут* id, для которого задано *значение* masthead. *Элемент* div содержит *элемент* h1 (заголовок первого, или самого важного уровня), который в свою*очередь* содержит некоторый текст. Часть этого текста упакована в *элемент* abbr (который используется для определения расширения сокращений), который имеет *атрибут* title, *значение* которого задано как Hypertext Markup Language.

Многие *атрибуты* в *HTML* являются общими для всех *элементов*, хотя некоторые являются специфическими для данного *элемента* или*элементов*. Они всегда имеют форму ключевое\_слово="значение". *Значение* должно быть помещено в одиночные или двойные кавычки (в некоторых ситуациях кавычки могут отсутствовать, но это не слишком хорошо с точки зрения предсказуемости, понимания и ясности — необходимо всегда заключать значения в кавычки).

*Атрибуты* и их возможные значения определяются в основном спецификациями *HTML* (<http://www.w3.org/TR/html401/index/attributes.html>) - нельзя создать свои собственные *атрибуты*, не сделав код *HTML* недействительным, так как это может путать агентов пользователей и вызывать проблемы правильной интерпретации *Web*-страницы. Единственными реальными исключениями являются *атрибуты* id и class — их значения полностью под вашим контролем, так как они предназначены для добавления в документы вашего собственного значения и семантики.

*Элемент* внутри другого *элемента* называют "потомком" этого *элемента*. Поэтому в примере выше abbr является потомком h1, который сам является потомком div. И наоборот, div будет называться "предком" *элемента* h1. Концепция предок/*потомок* является важной, так как она формирует основу *CSS* и активно используется в JavaScript.

**Элементы блочного уровня и строковые элементы**

Имеется две основные категории *элементов* в *HTML*, которые соответствуют типам контента и структуре, которую представляют эти *элементы*— *элементы* блочного уровня и *строковые* *элементы*.

Блочный уровень означает более высокий уровень *элемента*, обычно информирующий о структуре документа. *Элементы* блочного уровня можно представлять как *элементы*, которые начинаются с новой строки, отрываясь от того, что было перед этим. Распространенными блочными *элементами* являются параграфы, пункты списка, *заголовки* и таблицы.

*Строковые* *элементы* содержатся внутри структурных *элементов* блочного уровня и охватывают только части текста документа, а не целые области. *Строковый* *элемент* не приводит к появлению в документе новой строки, они являются *элементами*, которые появляются в параграфе текста. Распространенными *строковыми* *элементами* являются гипертекстовые ссылки, выделенные слова или фразы и краткие цитаты.

**Символьные ссылки**

Последним вопросом, который стоит упомянуть о документе *HTML*, является использование специальных символов. В *HTML* символы <, > и& являются специальными. Они начинают и заканчивают части документа *HTML*, а не представляют символы меньше, больше и амперсанд.

Одной из первых ошибок, которую может сделать *автор* в *Web*, является использование символа амперсанда в документе и получение в связи с этим чего-то неожиданного. Например, *запись* "stones&pounds" может в действительности появиться в некоторых браузерах как "stones£s".

Это связано с тем, что литеральная строка "&pound;" является в действительности символьной ссылкой в *HTML*. *Символьная ссылка* является способом включения в документ символа, который трудно или невозможно ввести с помощью клавиатуры, или в кодировке конкретного документа.

Символ амперсанда ( & ) вводит ссылку, а точка с запятой ( ; ) заканчивает ее. Однако, многие агенты пользователя могут быть достаточно снисходительны к ошибкам *HTML*, таким как отсутствие точки с запятой, и интерпретировать "&pound" как символьную ссылку. Ссылки могут быть либо числами (числовые ссылки) или сокращенными словами (объектные ссылки).

Реальный амперсанд должен вводиться в документ как "&amp;", что является объектной ссылкой символа, или как "&#38;", что является числовой ссылкой. Полную таблицу символьных ссылок можно найти на сайте

**Физическое и логическое форматирование.**

Трудно сказать определённо, как ведут себя поисковые роботы, и какие алгоритмы они включают, когда встречают на страницах Вашего сайта различные **стили форматирования** текста (выделение курсивом, жирный шрифт и т.д.). Определённо можно сказать только то, что хоть в современных браузерах мы и видим одинаковый результат при использовании разных тегов физического и **логического форматирования**, но задумывались они для разных целей. Проще говоря, **физическое форматирование** предназначено для физического выделения текста, т.е. для красоты. Логическое форматирование указывает браузерам (и должно указывать поисковикам) какой тип текста или данных находится в выделенном фрагменте. Это может быть программный код, переменная, цитата, аббревиатура или др. Кроме того, вид отображения текста при выделении тегами **логического форматирования,** можно задать произвольно, прописав это в стилях.

**Физическое форматирование.**

Тег <b>, тег <i>, тег <u>.

Тег <b> отображает выделенный фрагмент текста в полужирном начертании.

Тег <i> отображает выделенный фрагмент текста в виде курсива.

Тег <u> отображает выделенный фрагмент текста с подчёркиванием.

Допустимо использовать эти теги совместно, но только не забывайте о правилах вложенности.

Например: <b> <i> <u> Правила вложенности </u> </i> </b>.

**Логическое форматирование**.

Тег <strong>, тег <em>.

Тег <strong> применяется для выделения значимых фрагментов текста и подчёркивает его важность. Если не производилось перенастройки стилей, в браузерах отображается полужирным начертанием (аналогично тегу <b>).

Тег <em> также используется для выделения важных моментов при форматировании текста. В начертании аналогичен тегу физического форматирования <i> - курсив.

Примеры использования тегов <b>, <i>, <u>, <strong> и<em>.

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  
<html lang="ru">  
<head>  
<meta content="text/html; charset=windows-1251" http-equiv="content-type">  
<title>Физическое и логическое форматирование</title>  
</head>  
<body>  
<p>Обычный текст<br>  
<b>Физическое форматирование - полужирное начертание</b><br>  
<i>Физическое форматирование - курсив</i><br>  
<u>Физическое форматирование - подчёркивание</u><br>  
<strong>Логическое форматирование - важный фрагмент выделен полужирным начертанием</strong><br>  
<em>Логическое форматирование - важный фрагмент выделен курсивом</em>  
</body>  
</html>

Ещё теги логического форматирования.

### Тег <abbr>.

Тег <abbr> применяется для описания аббревиатур. В атрибуте title можно указать расшифровку.

Например.

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=WINDOWS-1251">  
<title>Тег ABBR</title>  
</head>  
<body>  
<abbr title="Hyper text Markup Language - язык разметки гипертекста">HTML</abbr>  
</body>  
</html>

### Тег <acronym>.

Тег <acronym> применяется для выделения сокращений, которые произносятся как единое слово. Например. ЦУМ или ВУЗ. Также имеет атрибут title.

### Тег <cite>, Тег <q>.

В тег <cite> и тег <q> помещайте текст, который Вы хотите процитировать, чтобы поисковый робот не уличил Вас в плагиате. При использовании тега <q> кавычки проставляются автоматически.

Пример использования тега <cite> и тега<q> с переопределённым стилем.

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=WINDOWS-1251">  
<title>Тег Q</title>  
<style type="text/css">  
Q {  
font-family: Times, serif; /\* Шрифт с засечками \*/  
font-style: italic; /\* Курсивное начертание текста \*/  
color: green; /\* Цвет текста \*/  
}  
</style>  
</head>  
<body>  
<p>Михаил Булгаков: <cite>"Это был ни с чем по прелести не сравнимый запах только что отпечатанных денег".</cite>  
<p>Михаил Булгаков: <q>Никогда и ничего не просите, никогда и ничего, особенно у тех, кто сильнее Вас, сами все предложат и сами все дадут</q>.</p>  
</body>  
</html>

**Тег <code>.**

Тег <code> применяется для отображения программного кода.

### Тег <dfn>.

Тег <dfn> используется, если в тексте встречается определение, т.е. Вы описываете какое-то понятие впервые.

### Тег <kbd>.

Тег <kbd> следует использовать, если Вы хотите предложить пользователю ввести какую-то информацию, нажать клавишу или комбинацию клавиш.

### Тег <samp>.

Тег <samp> используют для вывода результата работы программы.

На этом теги логического форматирования не заканчиваются, но мы рассмотрели, как я считаю, самые необходимые, которые следует применять для эффективной оптимизации текстов на страницах Ваших Интернет-ресурсов.

### Что такое ссылки?

Ссылки являются частью *Web*-сайта (созданного обычно с помощью *HTML*, но не всегда), которые указывают на другие ресурсы — другие документы *HTML*, *текстовые файлы*, файлы *PDF*, и т.д. Существуют ссылки, по которым *браузер* должен переходить автоматически, созданные с помощью элементов *link* (вы уже встречали некоторые из них в предыдущих лекциях — они используются для импорта файлов *CSS* в документ *HTML*), и существуют также ссылки, которые *пользователь* не обязан активировать. Они называются ***анкерами***, и их можно добавлять в документ с помощью элемента a.

### Анатомия анкерной ссылки

Любой *строковый элемент* в документе можно превратить в анкерную ссылку, добавляя вокруг него элемент a. Например, в следующем документе *HTML* текст *Yahoo Developer Network* превращен в ссылку (linkexample.*html*).

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

<meta http-equiv="content-type" content="text/html; charset=utf-8">

<title>Link Example</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<h1>A link to the YDN</h1>

<p><a href="http://developer.yahoo.com">Yahoo Developer Network</a></p>

</body>

</html>

Посетители, активирующие эту ссылку (щелкая на ней мышью или активируя ее с помощью клавиатуры или голоса в некоторых случаях) покинут текущий *сайт* и перейдут на YDN. Имеются дополнительные изменения, происходящие в самой ссылке, и мы увидим их позже, когда будем говорить о стилевом оформлении ссылки.

*Анкер* имеет несколько атрибутов, которые можно использовать:

* ***href*** - ресурс, на который указывает *анкер* (внешний файл или ID *анкера* ).
* id - ID *анкера*, если *анкер* является целью, а не ссылкой.
* *title* - дополнительная информация о внешнем ресурсе.

Давайте рассмотрим сначала наиболее важные атрибуты, а затем поговорим о том, что можно сделать, чтобы облегчить посетителям восприятие ссылок.

### Ссылка или цель? Атрибуты id и href

Элемент a может играть несколько ролей в зависимости от заданных атрибутов. Наиболее часто используется *атрибут* *href* , который определяет *ресурс*, на который указывает *ссылка*. Этот *атрибут* может содержать различные значения:

* URL, который может задаваться в той же папке (help.html), относительно текущей папки (например, "../../help/help.html" ), абсолютно, относительно корня сервера (например, "/help/help.html" ) или вообще на другом сервере (например, "http://wait-till-i.com", или "ftp://ftp.opera.com/", или "http://developer.yahoo.com/yui" ).
* *Идентификатор фрагмента* или имя *анкера* с предшествующей решеткой (например, "#menu" ). Это является указателем на цель внутри того же самого документа.
* Смесь обоих - можно соединиться прямо с разделом другого документа, указывая атрибут *href* с URL, за которым следует *идентификатор фрагмента*(например, "http://developer.yahoo.com/yui/#cheatsheets" ).

Любое из этих значений делает элемент a ссылкой, так как он указывает куда-то в другое *место*. С другой стороны *атрибут* id сделает его на странице *анкером*— позицией, на которую указывает другая *ссылка*. Это создает небольшую путаницу, так как в обоих случаях используется элемент *анкер* ( a ). Чтобы легче запомнить, представляйте это следующим образом: *атрибут* id делает ссылку *анкером*, и можно использовать его для соединения с определенными разделами документа. Следующий код *HTML* содержит примеры всех различных типов ссылок (linkexamples.*html*):

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

<meta http-equiv="content-type" content="text/html; charset=utf-8">

<title>Different Link Example</title>

<link rel="stylesheet" href="linkexamplestyles.css">

</head>

<body>

<h1>Different Link examples</h1>

<h2>Example of in-page navigation with fragment identifiers, links and anchors</h2>

<div id="nav">

<ul id="toc">

<li><a href="#sec1">Section One</a></li>

<li><a href="#sec2">Section Two</a></li>

<li><a href="#sec3">Section Three</a></li>

<li><a href="#sec4">Section Four</a></li>

<li><a href="#sec5">Section Five</a></li>

</ul>

</div>

<div id="content">

<div>

<h2><a id="sec1">Section #1</a></h2>

<p><a href="#toc">Back to menu</a></p>

</div>

<div>

<h2><a id="sec2">Section #2</a></h2>

<p><a href="#toc">Back to menu</a></p>

</div>

<div>

<h2><a id="sec3">Section #3</a></h2>

<p><a href="#toc">Back to menu</a></p>

</div>

<div>

<h2><a id="sec4">Section #4</a></h2>

<p><a href="#toc">Back to menu</a></p>

</div>

<div>

<h2><a id="sec5">Section #5</a></h2>

<p><a href="#toc">Back to menu</a></p>

</div>

</div>

<h2>Some other link examples</h2>

<ul>

<li><a href="http://developer.yahoo.com">Yahoo Developer Network</a></li>

<li><a href="http://dev.opera.com/articles/view/the-freelancing-business-part-1-pricing/#marketing">Tips on marketing yourself</a></li>

<li><a href="ftp://get.opera.com/pub/opera/win/">Download different Opera versions</a></li>

<li><a href="http://farm1.static.flickr.com/56/188791635\_0b8bdd808d.jpg?v=0">Photo of my book</a></li>

</ul>

</body>

</html>

Листинг Примеры всех различных типов ссылок

Откройте этот *файл* в браузере и поэкспериментируйте с ним. Вы обнаружите, что активация любой из ссылок в первом списке вызывает переход в соответствующий раздел документа. Это связано с тем, что они связаны одним идентификатором фрагмента — первая *ссылка* в списке, например, имеет *атрибутhref* со значением #sec1, что совпадает со значением ID ссылки в первом элементе h2 контента. Это все, что нужно сделать, чтобы соединить два анкерных элемента в документе — использовать то же самое *значение* в атрибуте ссылки *href* с предшествующим символом решетки, если это *значение* присутствует в атрибуте id. Можно также заметить, что URL в строке адреса в браузере изменился, и показывает теперь в конце *идентификатор фрагмента*, что означает, что посетители могут сделать закладку для этого раздела или отправить ссылку другому человеку через *e-mail*, чтобы точно направить их к соответствующему разделу.

Однако, если активировать любую из ссылок "*Back* to *menu*", происходит то же самое. Как это возможно? Идентификатором фрагмента может быть любой элемент с атрибутом ID. Повторим вкратце:

* анкерные ссылки могут иметь *идентификатор фрагмента* в качестве значения атрибута href - этот *идентификатор фрагмента* должен начинаться со знака решетки ( # ).
* При активации ссылка выполняет переход к любому элементу HTML с id, имеющим это значение. Значения ID на каждой странице должны быть уникальными.
* ID следуют определенным соглашениям об именах. Самое важное, что они должны начинаться с алфавитно-цифрового символа и не должны содержать пробелов.

Это охватывает *меню* и различные *разделы* в примере, но как насчет других ссылок? Если проверить их, то можно увидеть, что они указывают на различные цели — одна переходит на другой *сайт*, другая выводит фотографию, а третья показывает определенный раздел другой *Web*-страницы (находимую при переходе к указанному *ID*). Если все это работает, то отлично — но как быть, если вы или ваш *браузер* не можете понять некоторые из этих ресурсов?

Самое важное, что нужно помнить о ссылках, состоит в том, что они являются существенной частью отношений с посетителями. Они считают, что когда вы предлагаете им ссылку, они могут перейти по ней и получить хорошую, подходящую информацию. Если ссылки не работают, так как указанный *ресурс* недоступен, или представлен в формате, который посетитель не может использовать, то вы не оправдали надежду и теряете *доверие*. Не позволяйте такому происходить.

#### Предоставление дополнительной информации с помощью атрибута title

Как почти для любого другого элемента HTML в элемент a можно добавить атрибут *title* , чтобы предоставить некоторую дополнительную информацию. Браузеры показывают так называемую всплывающую подсказку, когда посетители проводят курсором своей мыши над ссылкой. Эта всплывающая подсказка сообщает информацию о ссылке. Например, можно дать небольшое введение в содержание и расположение присоединенного по ссылке документа (titleexample.html):

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

<meta http-equiv="content-type" content="text/html; charset=utf-8">

<title>Adding extra information with a title attribute</title>

<link rel="stylesheet" href="linkexamplestyles.css">

</head>

<body>

<h1>Adding extra information with a title attribute</h1>

<ul>

<li>Find more information on the <a title="The Yahoo Developer Network is the main hub

for all the developer tools Yahoo offers, including the Yahoo User Interface Library (YUI)

and the Design Patterns repository"

href="http://developer.yahoo.com">Yahoo Developer Network</a>.</li>

</ul>

</body>

</html>

Однако нельзя надеяться, что посетители обладают достаточным терпением и точной координацией, чтобы полагаться на это средство для критически важной информации. Пользователи с недостатками зрения, которые вообще не могут видеть страницу, скорее всего не смогут добраться до этой информации. Хотя считыватели экрана имеют возможность читать атрибуты *title* для конечного пользователя, она выключена по умолчанию — именно поэтому никогда не нужно использовать атрибут *title* для критически важной информации о ссылке. Критически важной может быть следующая информация:

* Ссылки на ресурсы, отличные от HTML, такие как файлы PDF, изображения, видео, звуковые файлы или загрузки.
* Уход с текущего сайта и соединение с другим сервером (внешние и внутренние ссылки).
* Соединение с документом, который открывается в другом фрейме или всплывающем окне.

#### Соединение с ресурсами, отличными от HTML - не заставляйте людей гадать

Может быть очень досадно, когда после щелчка на ссылке браузер не знает, что делать с тем, на что указывает эта ссылка. Однако достаточно часто можно видеть Web-сайты со ссылками на изображения, документы PDF, и видео, которые не предупреждают об этом посетителей. Видео особенно часто становится причиной для отказа браузеров. Более того, ресурс может быть большим по объему (PDF файл размером 20 Мбайтов, например), который посетитель мог бы предпочесть загрузить, а не открывать в браузере, или вообще не обращаться к этому документу.

Одним из важнейших факторов успеха Web-продукта является исключение для людей необходимости догадываться, что произойдет, когда они выполнят какое-то действие, и четко сообщить им вместо этого, какой результат будет иметь их действие. В случае ссылок для исключения разочарования достаточно сообщить посетителям, с каким ресурсом соединяется ссылка. Вот несколько примеров (linkingnonhtml.html):

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

<meta http-equiv="content-type" content="text/html; charset=utf-8">

<title>Linking non-HTML resources</title>

<link rel="stylesheet" href="linkexamplestyles.css">

</head>

<body>

<h1>Linking non-HTML resources</h1>

<ul>

<li>Find more information on the <a href="http://developer.yahoo.com">Yahoo Developer Network site (external)</a></li>

<li>Download the <a href="http://www.wait-till-i.com/stuff/JavaScript-DOM-Cheatsheet.pdf">Dom Cheatsheet (PDF, 85KB)</a></li>

<li>Pick and <a href="ftp://get.opera.com/pub/opera/win/">download different Opera versions from their FTP (external)</a></li>

<li>Check out a <a href="http://farm1.static.flickr.com/56/188791635\_0b8bdd808d.jpg?v=0">Photo of my book (JPG, 200KB)</a></li>

</ul>

</body>

</html>

Предоставляя такую информацию о файле ссылки и его размере, вы оставляете принятие решения о том, что с ним делать, посетителям, а не предполагаете, что они имеют определенные настройки браузера или установленное программное обеспечение. Если соединить это с хорошим стилевым оформлением, то можно даже заставить это выглядеть симпатичным и интуитивно понятным. Если вы хотите быть совершенно безопасным, предложите также раздел справки, который поясняет, что означают различные форматы файлов, и где можно взять программное обеспечение, необходимо для их вывода.

### Стилевое оформление ссылки

Bы пока еще не добрались в этом курсе до *CSS*, но будет полезно рассмотреть в этом месте, что способ представления ссылок очень важен, и что имеется несколько различных состояний ссылки, которые надо учитывать. Состояниями (которые позже свяжутся с псевдо селекторами *CSS* — это звучит угрожающе, но это не сложно) являются:

* *link* - состояние по умолчанию - оно определяет, как должны выглядеть ссылки в определенной части документа. По умолчанию непосещенные ссылки имеют синий цвет.
* visited - стиль ссылки, которая уже была посещена раньше (онa может уже находиться в кеше браузера). По умолчанию посещенные ссылки имеют пурпурный цвет.
* hover - стиль ссылки, когда курсор мыши находится над ней.
* active - стиль ссылки, когда она активируется, т.е. когда выполняется соединение с другим сайтом; это также стиль последней активированной ссылки, когда вы снова возвращаетесь к документу в своем браузере.

### Использование графики в HTML

Для того чтобы вставить в *Web*-страницу изображение, необходимо либо нарисовать его, либо взять уже готовое. В любой программе рисования можно создать простое изображение и сохранить его в нужном формате. Если *программа* этот формат не поддерживает, необходимо преобразовать *файл* в требуемый формат. Существует множество программ, предназначенных для преобразования одного графического формата в другой. Позаимствовать же *картинки* можно из различных программных пакетов или с других *Web*-страниц в *Internet*, содержащих библиотеки свободного доступа художественных изображений. Когда *браузер* выводит *Web*-страницу с изображением, соответствующий графический *файл* временно хранится в памяти компьютера. В большинстве браузеров есть *команда*, позволяющая сохранить *файл* на локальном диске. Существует также множество других вариантов получения графических файлов.

Изображения могут нести определенную информацию, да и просто придают *Web*-странице привлекательный вид. Приведем наиболее распространенные случаи применения изображений:

* логотип компании на деловой странице;
* графика для рекламного объявления;
* различные рисунки;
* диаграммы и графики;
* художественные шрифты;
* подпись автора страницы;
* применение графической строки в качестве горизонтальной разделительной линии;
* применение графических маркеров для создания красивых маркированных списков.

Теперь рассмотрим как вставить изображение в *Web*-страницу. Тегом *HTML*, который заставляет *браузер* выводить изображение, является <***IMG*** > с обязательным атрибутом SRC (SouRCe, источник). *Имя файла* представляет собой имя выводимого графического файла. Замыкающего тега не требуется.

Пример вставки изображения:

<IMG SRC="image.gif" ALT="ИЗОБРАЖЕНИЯ">

Изображения на *Web*-странице могут использоваться в качестве гипертекстовых ссылок, как и обычный текст. Читатель щелкает на изображении и отправляется на другую страницу или переходит к другому изображению. Для обозначения изображения как гипертекстовой метки используется тот же *тег* <A>, что и для текста, но между <A> и </A> вставляется *тег* изображения < *IMG* > :

<A HREF="адрес файла или изображения"><IMG SRC="image.gif"></A>

При этом изображение, используемое в качестве гипертекстовой ссылки, обводится дополнительной рамкой.

#### Атрибуты и их аргументы

Тег изображения имеет один обязательный атрибут SRC и необязательные: ALT, ALIGN, USEMAP, HSPACE, VSPACE, BORDER, WIDTH,HEIGHT.

##### Атрибут SRC

Указывает файл изображения и путь к нему; изображение должно быть загружено в браузер и размещено в том месте документа, где расположен тег изображения.

##### Атрибут ALT

Позволяет указать текст, который будет выводиться вместо изображения браузерами, неспособными представлять графику. В некоторых случаях при недостаточной пропускной способности линий связи пользователи отключают отображение графики. Наличие названий вместо картинок облегчает восприятие Web-страниц в таком режиме.

##### Атрибут АLIGN

Определяет положение изображения относительно окружающего его текста. Возможные значения аргумента — [ "top" | "middle" |"bottom" ] (соответственно, "вверху", "посередине", "внизу").

**ALIGN="top"** выравнивает верх изображения по верхнему краю самого высокого элемента в строке окружающего текста.

**ALIGN="middle"** выравнивает центр изображения по базовой линии строки окружающего текста.

**ALIGN="bottom"** выравнивает нижний край изображения по базовой линии строки окружающего текста.

Кроме основных значений атрибута ALIGN="ключевое слово" существует еще ряд аргументов, которые расширяют возможности взаимного размещения графики и текста. Рассмотрим их подробнее.

Дополнительные возможные значения аргумента — [ "left" | "right" | "top" | "texttop" | "middle" | "absmiddle" |"baseline" | "bottom" | "absbottom" ].

**ALIGN="left"** определяет огибаемое текстом изображение. Изображение располагается вдоль левой границы документа, а последующие строки текста огибают его справа.

**ALIGN="right"** определяет огибаемое текстом изображение. Изображение располагается вдоль правой границы документа, а последующие строки текста огибают его слева.

**ALIGN="top"** выравнивает верх изображения по верхнему краю самого высокого элемента в строке окружающего текста точно так же, как при использовании стандартного набора атрибутов.

**ALIGN="texttop"** выравнивает верх изображения по верхнему краю самого высокого текстового символа в строке окружающего текста. Действие этого аргумента в большинстве случаев, но не всегда, подобно действию аргумента ALIGN="top".

**ALIGN="middle"** выравнивает центр изображения по базовой линии строки окружающего текста точно так же, как при использовании стандартного набора атрибутов.

**ALIGN="absmiddle"** выравнивает центр изображения по центру строки окружающего текста.

**ALIGN="baseline"** выравнивает нижний край изображения по базовой линии строки окружающего текста, то есть производит такое же действие, как и ALIGN="bottom".

**ALIGN="bottom"** выравнивает нижний край изображения по базовой линии строки окружающего текста точно так же, как при использовании стандартного набора атрибутов.

**ALIGN="absbottom"** выравнивает нижний край изображения по нижнему краю строки окружающего текста.

##### Атрибут USEMAP

Если присутствуют атрибут USEMAP и тег <MAP>, изображение становится чувствительной картой, или "графическим меню". Если щелкнуть кнопкой мыши на активной области изображения, для которого определен атрибут USEMAP, произойдет гипертекстовый переход к информационному ресурсу, установленному для этой области. Более подробно этот вопрос будет рассматриваться в следующем разделе.

##### Атрибут BORDER

Целочисленное значение аргумента определяет толщину рамки вокруг изображения. Если значение равно нулю, рамка отсутствует. Чтобы не вводить пользователей в заблуждение, не стоит задействовать BORDER=0 в изображениях, которые представляют собой часть элемента якоря, поскольку рисунки, применяемые в качестве гиперссылок, обычно выделяются цветной рамкой.

##### Атрибут HSPACE

Целочисленное значение этого атрибута задает горизонтальное расстояние между вертикальной границей страницы и изображением, а также между изображением и огибающим его текстом.

##### Атрибут VSPACE

Целочисленное значение этого атрибута задает вертикальное расстояние между строками текста и изображением.

##### Атрибуты WIDTH и HEIGHT

Оба атрибута задают целочисленные значения размеров изображения по горизонтали и по вертикали соответственно. Это позволяет уменьшить время загрузки страницы с графикой. Браузер сразу отводит рамку для изображения и продолжает загружать текст на страницу. Пока загружается графика, пользователь может начать читать текст. Определить размер изображения нетрудно, для этого достаточно воспользоваться любой программой просмотра графических файлов, например ACDSee или графическим редактором Corel PhotoPaint или Adobe Photoshop. Откройте файл в графическом редакторе и определите размер *картинки* в пикселах. В теге изображения следует указать ширину и высоту *картинки*.

<IMG SRC="image.gif" ALT="изображение" WIDTH="100" HEIGHT="200" HSPACE="10" VSPACE="10"

BORDER="2" ALIGN="left">

### Форматы графических файлов

Самыми распространенными графическими форматами в *Web* являются *GIF* и *JPEG*. ***GIF*** — наиболее подходящий формат для обмена изображениями между системами. Архивы с изображениями в формате *GIF* можно найти на многих серверах *Internet*. Данный формат поддерживают многие графические приложения, в том числе все программы просмотра графики World Wide *Web*.

Однако у этого формата есть одно серьезное ограничение: он не поддерживает изображения с глубиной цвета больше восьми *бит* на *пиксел*. Обычно восьми *бит* на пиксель оказывается достаточно для контурных изображений типа комиксов и рисунков, где используется ограниченное количество цветов, или для небольших картинок, где для цветопередачи хватает 256 оттенков. Однако для больших изображений фотографического качества больше подходит формат *JPEG*.

Формат *GIF* использует один из лучших алгоритмов сжатия *LZW*, который изначально не предназначался специально для графики. Он не очень подходит для работы с двухцветными (черно-белыми) или фотографическими изображениями.

С развитием аппаратного обеспечения, поддерживающего высокое разрешение и богатую цветовую гамму, графические файлы значительно увеличились в размерах. Профессиональные художники теперь, как правило, работают с файлами, содержащими 10 и более *мегабайт* данных на каждое изображение. Даже пользователи с более скромными запросами подчас имеют дело с изображениями 640 на 480 пикселей в 256 цветах (а это более 300 *килобайт*). Кроме того, многие сейчас начинают работать с полноцветными изображениями 1024 на 768 пикселей (это более 2,3 *мегабайт* данных). Так как высококачественные изображения встречаются все чаще, ограничения, накладываемые традиционными методами сжатия (например, *LZW*), становятся все более ощутимыми.

Для поиска оптимального способа сжатия изображений фотографического качества две международные организации по стандартизации,*International* *Telecommunications* *Union* (*ITU*, Международный союз телекоммуникаций) и *International* Organization for Standartization (*ISO*, Международная организация по стандартизации), создали *Joint* Photographic Experts *Group* ( ***JPEG***, объединенная экспертная *группа* по фотографии). С тех пор сокращение " *JPEG* " используется как название этой техники сжатия. Кроме того, оно входит в названия некоторых использующих ее файловых форматов.

Имя *JPEG* указывает на метод сжатия, но не на формат файла. На самом деле метод сжатия *JPEG* используют как многочисленные мало различающиеся форматы, зачастую известные, например " *JPEG* ", так и единичные радикально отличающиеся форматы, такие как *TIFF* и Quick*Time*. К счастью, все же большинство форматов, известных под именем " *JPEG* ", очень похожи, и, скорее всего, у вас не возникнет с этим проблем, однако знать о возможных осложнениях не помешает.

Формат *JPEG* отличается от других графических форматов прежде всего тем, что он использует метод сжатия "с потерями". *JPEG* частично идентифицирует и удаляет ту информацию, которая несущественна для восприятия изображения. В результате *JPEG* может достигать высокого уровня сжатия без заметных потерь в качестве изображения.

Метод сжатия "с потерями" имеет много реализаций. *JPEG* достигает существенного сжатия за счет отбрасывания той графической информации, которая обычно не проявляется в реальных изображениях. Однако при сжатии с помощью *JPEG* изображений с четкими контурами линии начинают заметно "дрожать". Так, например, если изображение содержит какие-либо подписи, подобный эффект может возникнуть вокруг символов. Этот эффект можно свести к минимуму, задав очень высокие значения параметра качества, однако при этом нельзя достичь приемлемых показателей сжатия.

Так как *JPEG* предполагает сжатие с потерями, при создании файлов необходимо быть внимательным. Большинство программ, создающих такие файлы, позволяют задавать *значение* параметра качества изображения. Обычно оно варьирует от нуля до ста. Нижние значения позволяют при сжатии *JPEG* отбрасывать больше информации, в результате чего получаются файлы меньшего размера. В свою *очередь*, высокие значения ограничивают *количество информации*, которой можно пренебречь во время сжатия.

Одна из наиболее распространенных ошибок заключается в интерпретации значения параметра качества от нуля до ста как процента сохраняемых данных. Чтобы развеять это заблуждение, некоторые современные программные продукты *JPEG* просто используют шкалу "лучшее сжатие" — "лучшее качество".

Хитрость заключается в том, чтобы при наименьшей величине параметра качества получить изображение без видимого его ухудшения. Лучше начинать со средних значений и внимательно оценивать результат. Если вы отмечаете некоторое ухудшение, попробуйте увеличить *значение* параметра, если нет — попытайтесь его уменьшить. При просмотре изображения обращайте внимание на следующие моменты: четкость очертаний и углов, например, вокруг текста, или *контур* детали изображения, выделяющейся на общем фоне. Часто бывает заметно, что *контур* "смазан" или линия "дрожит".

Сжатие *JPEG* использует мозаику размером восемь на восемь пикселей. Если задаются слишком низкие значения качества, ее границы могут стать заметны. Если у вас уже есть изображения в *GIF* или другом восьмиразрядном формате, возможно, вы захотите попробовать конвертировать их в *JPEG*. Несмотря на то, что иногда это все же приводит к уменьшению необходимого для хранения файлов пространства, в большинстве случаев *игра* не стоит свеч. Если вы все же хотите попытаться, сначала выясните, сколько цветов использует изображение *GIF*. Если в нем только 64 цвета, то *конверсия* вряд ли себя оправдает, так как изображение с такой бедной цветовой палитрой не имеет тех плавных цветовых переходов, которые хорошо сжимает *JPEG*. В результате вы просто ухудшите качество изображения, не освободив места.

Одна из серьезных проблем конверсии изображений *GIF* в *JPEG* заключается в том, что изображения в формате *GIF*, лимитированные набором из 256 (или менее) цветов, часто используют клиширование (*dithering*) и полутона (*halftoning*), в результате чего пиксели двух цветов смешиваются для получения эффекта третьего тона. В результате использования этой техники образуются шаблоны, крайне плохо сжимаемые с помощью *JPEG*. Отдельные программы позволяют усреднять значения этих шаблонов, "смягчая" таким образом изображение до преобразования, в результате чего сжатие с помощью *JPEG* оказывается более эффективным.

**1.2 Списки. Таблицы. Блоки. Изображения-карты**

Списки используются для объединения в группу *связанных объектов* информации, чтобы облегчить их ассоциирование друг с другом и чтение. В современной разработке *Web* списки являются основополагающими элементами, часто используемыми для навигации, а также обычного контента.

Списки хорошо подходят со структурной точки зрения, так как они помогают создать хорошо структурированный, более доступный, и простой в обслуживании документ. Они полезны также с чисто практических соображений — они предоставляют дополнительные элементы для соединения со стилями *CSS* для целого *множества* стилей оформления.

В этой лекции будут рассмотрены различные типы списков, доступных в *HTML*, когда и как их нужно использовать, и как применять некоторые базовые стили.

### Три типа списков

В *HTML* имеется три типа списков:

* **Неупорядоченный список** - используется для объединения в группу множества *связанных объектов* без определенного порядка.
* **Упорядоченный список** - используется для объединения в группу множества *связанных объектов* в определенном порядке.
* **Список определений** - используется для вывода пар имя/значение, таких как термины и их *определения*, или время и события.

Каждый из них имеет определенное назначение - они не являются взаимозаменяемыми!

#### Неупорядоченные списки

*Неупорядоченные* списки, или маркированные списки, используются, когда множество объектов может быть размещено в любом порядке. Примером является список покупок:

* молоко
* хлеб
* сливочное масло
* кофейные зерна

Однако все эти объекты являются частью одного списка, можно разместить объекты в любом порядке и список будет по-прежнему иметь смысл:

* хлеб
* кофейные зерна
* молоко
* сливочное масло

Можно использовать CSS для изменения маркера на один из нескольких используемых по умолчанию стилей, использовать свое собственное изображение, или даже вывести список без маркеров - мы посмотрим, как сделать это, немного позже в этой лекции, и расширим немного дальше в будущей лекции.

##### Разметка неупорядоченного списка

*Неупорядоченные* списки используют одну пару тегов <ul></ul>, охватывающих множество пар тегов <li></li>:

<ul>

<li>хлеб </li>

<li>кофе в зернах </li>

<li>молоко </li>

<li>масло </li>

</ul>

#### Упорядоченные списки

*Упорядоченные* списки, или нумерованные списки, используются для вывода списка объектов, которые необходимо разместить в определенном порядке. Примером могут быть кулинарные инструкции по приготовлению, которые должны быть выполнены в определенном порядке рецепта:

1. Собрать ингредиенты
2. Смешать ингредиенты
3. Поместить ингредиенты в форму для выпечки
4. Выпекать в духовке в течение часа
5. Вынуть из духовки
6. Дать постоять десять минут
7. Подать

Если список позиций переставить в другом порядке, то информация больше не будет иметь смысл:

1. Собрать ингредиенты
2. Выпекать в духовке в течение часа
3. Вынуть из духовки
4. Подать
5. Поместить ингредиенты в форму для выпечки
6. Дать постоять десять минут
7. Смешать ингредиенты

*Упорядоченные* списки могут выводиться с помощью одной из нескольких цифровых или алфавитных систем — то есть с буквами или числами. По умолчанию в большинстве браузеров используются десятичные числа, но имеется большее количество возможностей.

* Буквы
  + Буквы ascii нижнего регистра (a, b, c…)
  + Буквы ascii верхнего регистра (A, B, C…)
  + Классические греческие буквы нижнего регистра: ( \alpha, \beta, \gamma …)
* Числа
  + Десятичные числа (1, 2, 3…)
  + Десятичные числа с ведущим нулем (01, 02, 03…)
  + Римские числа в нижнем регистре (i, ii, iii…)
  + Римские числа в верхнем регистре (I, II, III…)
  + Традиционная грузинская нумерация (an, *ban*, *gan*…)
  + Традиционная армянская нумерация (mek, yerku, yerek…)

Здесь также можно использовать CSS для изменения стиля списков.

##### Разметка упорядоченного списка

*Упорядоченные* списки используют одну пару тегов <ol></ol>, охватывающих множество пар тегов <li></li>:

<ol>

<li>Собрать ингредиенты</li>

<li>Смешать ингредиенты</li>

<li>Поместить ингредиенты в форму для выпечки </li>

<li>Выпекать в духовке в течение часа </li>

<li>Вынуть из духовки</li>

<li>Дать постоять десять минут </li>

<li>Подать</li>

</ol>

##### Начало упорядоченных списков с числа, отличного от 1

Можно создать *упорядоченный* список, нумерация которого начинается с числа отличного от 1 (или i, или I, и т.д.). Это делается с помощью атрибута start, который получает числовое значение, даже если используется CSS для изменения нумерации списка на алфавитную или римскую с помощью свойства *list-style-type*. Это будет полезно, если имеется единый список объектов, но вы хотите разбить список некоторыми примечаниями, или некоторой другой подходящей информацией. Например, можно было бы сделать это с предыдущим примером:

<ol>

<li>Собрать ингредиенты </li>

<li>Смешать ингредиенты </li>

<li>Поместить ингредиенты в форму для выпечки </li>

</ol>

<p class="note">Прежде чем поместить ингредиенты в форму для выпечки,

нагрейте духовку до 180 градусов по Цельсию /350 градусов по Фаренгейту,

чтобы быть готовым к следующему шагу </p>

<ol start="4">

<li> Выпекать в духовке в течение часа </li>

<li> Вынуть из духовки </li>

<li> Дать постоять десять минут </li>

<li> Подать </li>

</ol>

Это приводит к следующему результату:

1. Собрать ингредиенты
2. Смешать ингредиенты
3. Поместить ингредиенты в форму для выпечки

Прежде чем поместить ингредиенты в форму для выпечки, нагрейте духовку до 180 градусов по Цельсию /350 градусов по Фаренгейту, чтобы быть готовым к следующему шагу

1. Выпекать в духовке в течение часа
2. Вынуть из духовки
3. Дать постоять десять минут
4. Подать

Отметим, что этот атрибут является фактически исключенным в самой последней версии спецификации *HTML*, что означает, что страницы не пройдут валидацию при использовании строгого doctype. Это может показаться странным, так как этот атрибут имеет смысл, и нет эквивалента в CSS. Это показывает, что валидация *HTML* является идеальной целью для достижения, но не всегда абсолютно конечной целью. Кроме того, есть и еще один спасительный момент — атрибут start больше не является исключенным в спецификации *HTML*5 (документ об отличиях *HTML* 5 от *HTML* 4 подтверждает это). Если вы хотите использовать такую возможность на строгой странице *HTML* 4, и она должна обязательно пройти валидацию, можно сделать это с помощью CSS Counters

#### Списки определений

Списки *определений* соединяют определенные объекты и их *определения* в списке. Например, если вы хотите добавить *определения* к объектам в списке покупок, можно сделать это следующим образом:

молоко

Белый жидкий молочный продукт.

хлеб

Продукт питания, получаемый выпечкой из муки.

сливочное масло

желтый твердый молочный продукт.

кофейные зерна

Семена плодов некоторых кофейных деревьев.

Каждое *определение* и термин является группой *определения* (или группой имя-значение). Можно иметь любое требуемое количество групп *определений*, но в каждой группе должны быть как минимум один термин и как минимум одно *определение*. Нельзя иметь термин без *определения* или *определение* без термина.

Можно ассоциировать более одного термина с одним *определением*, или наоборот. Например, термин "кофе" может иметь несколько значений, и можно показать их одно за другим:

кофе

напиток, приготовленный из жареных, молотых кофейных зерен

чашка кофе

встреча, во время которой пьют кофе

цвет от среднего до темно коричневого

Альтернативно можно иметь более одного термина с одним *определением*. Это используется, чтобы показать вариации термина, которые все имеют одно значение:

содовая

шипучка

газировка

кола

Сладкий, насыщенный углекислым газом, напиток

Списки *определений* отличаются от других видов списков, так как они используют определяемые термины и описания *определений* вместо объектов списка.

Поэтому списки *определений* используют одну пару элементов <dl></dl>, охватывающую группы тегов <dt></dt> и <dd></dd>. Необходимо располагать парами как минимум одну группу тегов <dt></dt> с одной группой <dd></dd> ; теги <dt></dt> должны всегда быть первыми по порядку.

Простой список *определений* одного термина с одним *определением* будет выглядеть следующим образом:

<dl>

<dt>Термин </dt>

<dd>Определение термина </dd>

<dt>Термин </dt>

<dd>Определение термина </dd>

<dt>Термин </dt>

<dd>Определение термина </dd>

</dl>

Который выводится следующим образом:

Термин

Определение термина

Термин

Определение термина

Термин

Определение термина

В этом примере мы соединяем более одного термина с *определением* и наоборот:

<dl>

<dt>Термин </dt>

<dd>Определение термина </dd>

<dt>Термин </dt>

<dt>Термин </dt>

<dd>Определение которое применяется к обоим предыдущим терминам </dd>

<dt>Термин, который может иметь оба следующие определения </dt>

<dd>Одно определение термина </dd>

<dd>Другое определение термина </dd>

</dl>

Что будет представлено следующим образом:

Термин

Определение термина

Термин

Термин

Определение которое применяется к обоим предыдущим терминам

Термин, который может иметь оба следующие определения

Одно определение термина

Другое определение термина

Обычно не принято связывать несколько терминов с одним *определением*, но полезно знать, что это возможно, если возникнет такая необходимость.

### Выбор типа списка

При принятии решения об использовании определенного типа списка, обычно можно решить это, задавая два простых вопроса:

1. Определяются термины (или соединяются другие пары имя/значение)?
   * Если да, используйте список *определений*.
   * Если нет, не используйте список *определений* - перейдите к следующему вопросу.
2. Важен ли порядок элементов списка?
   * Если да, используйте *упорядоченный* список.
   * Если нет, используйте *неупорядоченный* список.

### Различие между списками HTML и текстом

Можно задаться вопросом, в чем различие между списком *HTML* и каким-то текстом с маркерами или числами, написанными вручную. Существует несколько преимуществ использования списка *HTML*:

* Если потребуется изменить порядок элементов списка в упорядоченном списке, то вы просто перемещаете их в коде *HTML*. Если числа будут написаны вручную, то придется все просмотреть и изменить число каждого элемента, чтобы исправить порядок — что достаточно скучно, по крайней мере!
* Использование списка *HTML* позволяет правильно оформить стиль списка. Если используется просто большой текст, то окажется значительно труднее оформить стиль отдельных элементов каким-либо более-менее полезным образом.
* Использование списка *HTML* создает для контента подходящую семантическую структуру, а не просто "спискообразный" визуальный эффект. Это имеет важные преимущества, так как позволяет считывателям экрана сообщить пользователям с недостатками зрения, что они читают список, а не просто прочитать путаную смесь текста и чисел.

С другой стороны: текст и списки не являются одним и тем же. Использование текста вместо списка требует больше работы, и может создавать проблемы для читателей документа. Поэтому, если в документе требуется *список*, то необходимо использовать правильный *список* *HTML*.

### Вложенные списки

*Элемент списка* может содержать другой *целый* *список* — он называется "вложенным" списком. Это может быть полезно для таких вещей как *таблица* контента, такой как в начале лекции:

1. Глава один

1. Раздел один

2. Раздел два

3. Раздел три

2. Глава два

3. Глава три

Ключевым моментом, который надо помнить, является то, что вложенный *список* должен относиться к одному конкретному элементу списка. Чтобы отразить это в коде, вложенный *список* помещается внутрь этого элемента списка. Код для приведенного выше списка выглядит следующим образом:

<ol>

<li>Глава один

<ol>

<li>Раздел один </li>

<li>Раздел два </li>

<li>Раздел три </li>

</ol>

</li>

<li>Глава два </li>

<li>Глава три </li>

</ol>

Отметим, что вложенный *список* начинается после элемента <li> и текста, содержащего *список* элемента ("Глава один"); а заканчивается перед элементом </li>, содержащего *список* элемента. Вложенные списки часто формируют основу для навигационного *меню* *Web*-сайта, так как они являются удобным способом *определения* структуры *Web*-сайта.

Теоретически можно вложить любое количество списков, хотя на практике это может внести путаницу при слишком глубоком вложении списков. Для очень больших списков может быть лучше разбить содержимое на несколько списков с заголовками, или даже разбить на отдельные страницы.

### Пошаговый пример

Давайте рассмотрим пошаговый пример, чтобы собрать все это вместе. Рассмотрим следующий *сценарий*:

Мы создаем небольшой *Web*-*сайт* для Кулинарной школы. На основной странице мы покажем *список* классифицированных рецептов, связанный ссылками со страницами рецептов. Каждая страница рецепта перечисляет требуемые ингредиенты, замечания по этим ингредиентам и метод приготовления. Тремя категориями являются "Cakes" (включающая рецепты для "Plain *Sponge*", "Chocolate Cake" и "Apple Tea Cake"); "Biscuits" (включающая рецепты "ANZAC Biscuits", "*Jam* Drops" и "Melting *Moments*"); и "Quickbreads" (включающая рецепты для "Damper" и "Scones"). Клиенту все равно в каком порядке будут выводиться категории и рецепты, он просто хочет, чтобы люди понимали, какие позиции являются категориями, а какие являются рецептами.

Давайте пройдем через процесс создания этого сайта. В этом разделе мы займемся созданием разметки, а также добавим некоторое стилевое оформление для списков. Стилевое оформление не будет рассматриваться подробно до лекции о стилевом оформлении списков в дальнейшем в этой серии.

#### Разметка основной страницы

Создайте правильно сформированную страницу *HTML*, включая doctype, элементы *html* , head и body, и сохраните его как stepbystep-main.html. Добавьте основной заголовок ( h1 ) " *HTML* Cooking School", и подзаголовок ( h2 ) "Recipes":

<h1>HTML Cooking School</h1>

<h2>Recipes</h2>

Имеется три категории рецептов для представления, и порядок не важен - для этого больше всего подходит *неупорядоченный* список, поэтому добавим на страницу следующий код:

<h2>Recipes</h2>

<ul>

<li>Cakes</li>

<li>Biscuits</li>

<li>Quickbreads</li>

</ul>

Отступ для элементов li делает код удобнее для чтения, но не является обязательным.

Теперь нужно добавить рецепты в качестве подпунктов, например, "Plain *Sponge*", "Chocolate Cake" и "Apple Tea Cake" являются частью категории "Cakes". Для этого необходимо в каждой позиции списка создать вложенный список. Так как порядок неважен, то снова подходит *неупорядоченный* список. Чтобы упростить материал для учебного руководства, все рецепты можно соединить с одной страницей рецептов:

<h2>Recipes</h2>

<ul>

<li>Cakes

<ul>

<li><a href="stepbystep-recipe.html">Plain Sponge</a></li>

<li><a href="stepbystep-recipe.html">Chocolate Cake</a></li>

<li><a href="stepbystep-recipe.html">Apple Tea Cake</a></li>

</ul>

</li>

<li>Biscuits

<ul>

<li><a href="stepbystep-recipe.html">ANZAC Biscuits</a></li>

<li><a href="stepbystep-recipe.html">Jam Drops</a></li>

<li><a href="stepbystep-recipe.html">Melting Moments</a></li>

</ul>

</li>

<li>Breads/quickbreads

<ul>

<li><a href="stepbystep-recipe.html">Damper</a></li>

<li><a href="stepbystep-recipe.html">Scones</a></li>

</ul>

</li>

</ul>

### Заключение

Теперь вы должны иметь четкое понимание трех различных типов списков в *HTML*. Используя пошаговый пример вы должны были создать все три типа списков и узнать, как вкладывать списки в элементы списка.

Когда вы узнали, как правильно использовать списки *HTML*, то, вероятно, обнаружите, что используете их все время. Существует много контента в *Web*, который должен помещаться в списки, но был просто вставлен в базовый элемент с несколькими тегами переноса строки. Эта располагающая к лени практика, которая создает значительно больше проблем, чем решает — поэтому не делайте этого! Вы должны всегда создавать семантически правильные списки, чтобы облегчить людям чтение создаваемых *Web*-сайтов. Это значительно лучшая практика для каждого, и не в последнюю *очередь* для себя, когда понадобится обслуживать сайты в дальнейшем.

### Средства описания таблиц в HTML

*По* мере развития *WWW* стало ясно, что средств, которые заложены в НТМL, недостаточно для качественного отображения различного типа документов. Недостатком НТМL было отсутствие в его составе средств отображения *таблиц*. Для этой цели обычно использовался предформатированный текст (*тег* <PRE> ), в котором *таблица* обрисовывалась символами АSСII. Но такая форма представления *таблиц* была недостаточно высокого качества и выбивалась из общего стиля документа. После введения *таблиц* в *HTML* у Web-мастеров появился не просто инструмент для размещения текстовых и числовых данных, а мощное средство дизайна для размещения в нужном месте экрана графических образов и текста.

#### Создание таблиц в HTML

Для описания ***таблиц*** используется тег <ТАВLЕ> . Тег <ТАВLЕ>, как и многие другие, автоматически переводит строку до и после *таблицы*.

##### Создание строки таблицы - тег <ТR>

Тег <ТR> (Таble Row, строка *таблицы* ) создает строку *таблицы*. Весь текст, другие теги и атрибуты, которые требуется поместить в одну строку, должны размещаться между тегами <ТR></ТR>.

##### Определение ячеек таблицы - тег <ТD>

Внутри строки *таблицы* обычно размещаются ячейки с данными. Каждая ячейка, содержащая текст или изображение, должна быть окружена тегами <ТD></ТD>. Число тегов <ТD></ТD> в строке определяет число ячеек ([открыть](http://www.intuit.ru/EDI/06_02_17_3/1486333223-7320/tutorial/100/objects/5/files/example1.htm))

<HTML>

<BODY>

<H1 ALIGN=center>Таблица</H1>

<CENTER>

<TABLE BORDER>

<TR>

<TD COLSPAN=3>Если в таблице два

тега TR, то в ней две строки.</TD>

</TR>

<TR>

<TD>Если в строке три тега TD,</TD>

<TD>то в ней</TD>

<TD>три столбца.</TD>

</TR>

</TABLE>

</CENTER>

</BODY>

</HTML>

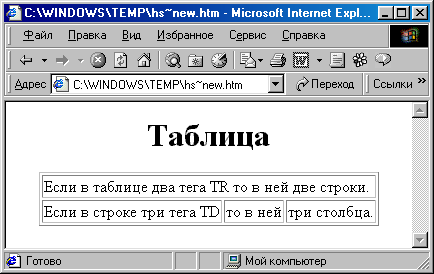


Рис.Использование таблиц в дизайне

##### Заголовки столбцов таблицы - тег <ТН>

Заголовки для столбцов и строк *таблицы* задаются с помощью тега заголовка <ТН></ТН> (Таblе Неаder, заголовок *таблицы* ). Эти теги подобны <ТD></ТD>. Отличие состоит в том, что текст, заключенный между тегами <ТН></ТН>, автоматически записывается жирным шрифтом и по умолчанию располагается посередине ячейки. Центрирование можно отменить и выровнять текст по левому или правому краю. Если воспользоваться <ТD></ТD> с тегом <В> и атрибутом <АLIGN=center>, текст тоже будет выглядеть как заголовок. Однако следует иметь в виду, что не все браузеры поддерживают в *таблицах* жирный шрифт, поэтому лучше задавать заголовки *таблиц* с помощью <ТН>.

<HTML>

<BODY>

<TABLE BORDER>

<TR>

<TH>Заголовок центрирован по умолчанию

</TH>

<TH COLSPAN=2>Заголовок может объединять

столбцы</TH>

</TR>

<TR>

<TH>Заголовок может быть расположен

перед столбцами</TH>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

</TR>

<TR>

<TH ROWSPAN=3>Заголовок может объединять

строки</TH>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

</TR>

<TR>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

</TR>

<TR>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

</TR>

</TABLE>

</BODY>

</HTML>

##### Использование заголовков таблицы - тег <САРТIОN>

Тег <CAPTION> позволяет создавать заголовки *таблицы*. По умолчанию заголовки центрируются и размещаются либо над ( <САРТION АLIGN=top> ), либо под *таблицей* ( <САРТION ALIGN=bottom> ). Заголовок может состоять из любого текста и изображений. Текст будет разбит на строки, соответствующие ширине *таблицы*. Иногда тег <САРТION> используется для подписи под рисунком. Для этого достаточно описать *таблицу* без границ.

<HTML>

<BODY>

<TABLE BORDER>

<CAPTION ALIGN=top>Заголовок над таблицей

</CAPTION>

<TR>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

</TR>

</TABLE>

<TABLE BORDER>

<CAPTION ALIGN=bottom>Заголовок под таблицей

</CAPTION>

<TR>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

</TR>

</TABLE>

</BODY>

</HTML>

##### Атрибут NOWRAP

Обычно любой текст, не помещающийся в одну строку ячейки *таблицы*, переходит на следующую строку. Однако при использовании атрибута NOWRAP с тегами <ТН> или <ТD> длина ячейки расширяется настолько, чтобы заключенный в ней текст поместился в одну строку.

##### Атрибут СОLSPAN

Теги <ТD> и <ТН> модифицируются с помощью атрибута СОLSPAN (Column Span, соединение столбцов). Если вы хотите сделать какую-нибудь ячейку шире, чем верхняя или нижняя, можно воспользоваться атрибутом СОLSPAN, чтобы растянуть ее над любым количеством обычных ячеек.

<HTML>

<BODY>

<CENTER>

<TABLE BORDER=3>

<TR>

<TD>Если вы хотите сделать какую-нибудь

ячейку шире, чем верхняя или нижняя,

</TD>

<TD>можно воспользоваться атрибутом

СОLSPAN=2, </TD>

</TR>

<TR>

<TD BGCOLOR=white COLSPAN=2>чтобы

растянуть ее над любым количеством

обычных ячеек.</TD>

</TR>

</TABLE>

</CENTER>

</BODY>

</HTML>

##### Атрибут ROWSPAN

Атрибут ROWSPAN, используемый в тегах <ТD> и <ТН>, подобен атрибуту СОLSPAN=, только он задает число строк, на которые растягивается ячейка. Если вы указали в атрибуте ROWSPAN=s число, большее единицы, то соответствующее количество строк должно находиться в растягиваемой ячейке. Внизу *таблицы* ее поместить нельзя.

##### Атрибут WIDТН

Атрибут WIDТН применяется в двух случаях. Можно поместить его в тег <ТАВLЕ>, чтобы задать ширину всей *таблицы*, а можно использовать в тегах <ТD> или <ТН>, чтобы задать ширину ячейки или группы ячеек. Ширину можно указывать в пикселах или в процентах. Например, если вы задали в теге <ТАВLЕ WIDTH=250>, вы получите *таблицу* шириной 250 пикселей независимо от размера страницы на мониторе. При задании WIDТН=50% в теге <ТАВLЕ> *таблица* будет занимать половину ширины страницы при любом размере изображения на экране. Так что, указывая ширину *таблицы* в процентах, имейте в виду, что если у пользователя узкая область просмотра, ваша страница может выглядеть несколько странно. Если вы пользуетесь пикселами, и *таблица* оказывается шире области просмотра, внизу появится полоса прокрутки для перемещения вправо и влево по странице. В зависимости от поставленных задач и тот, и другой способ задания ширины *таблицы* может оказаться полезным.

<HTML>

<BODY>

<TABLE BORDER WIDTH=100%>

<TR>

<TD ALIGN=center>Текст или данные -

ширина 100%</TD>

</TR>

</TABLE>

или<BR>

<TABLE BORDER WIDTH=50%>

<TR>

<TD ALIGN=center>Текст или данные -

ширина 50%</TD>

</TR>

</TABLE>

или<BR>

<TABLE BORDER WIDTH=200>

<TR>

<TD ALIGN=center>Текст или данные -

ширина 200 пикселов</TD>

</TR>

</TABLE>

или<BR>

<TABLE BORDER WIDTH=100>

<TR>

<TD ALIGN=center>Текст или данные -

ширина 100 пикселов</TD>

</TR>

</TABLE>

</BODY>

</HTML>

##### Применение пустых ячеек

Если ячейка не содержит данных, она не будет иметь границ. Если требуется, чтобы у ячейки были границы, но не было содержимого, необходимо поместить в нее что-то, что не будет видно при просмотре. Можно воспользоваться пустой строкой <ВR>. Можно даже задать пустые столбцы, определив их ширину в пикселах или относительных единицах и не введя в полученные ячейки никаких данных. Это средство может оказаться полезным при размещении на странице текста и *графики*.

##### Атрибут СЕLLРАDDING

Данный атрибут определяет ширину пустого пространства между содержимым ячейки и ее границами, то есть задает поля внутри ячейки.

<HTML>

<BODY>

<CENTER>

<TABLE BORDER CELLPADDING=20>

<TR>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

</TR>

<TR>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

</TR>

</TABLE>

<BR>

<TABLE BORDER CELLPADDING=0>

<TR>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

</TR>

<TR>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

</TR>

</TABLE>

</CENTER>

</BODY>

</HTML>

##### Атрибуты АLIGN и VALIGN

Теги <ТR>, <ТD> и <ТН> можно модифицировать с помощью атрибутов ALIGN и VALIGN. Атрибут АLIGN определяет выравнивание текста и *графики* по горизонтали, то есть по левому или правому краю, либо по центру. Горизонтальное выравнивание может быть задано несколькими способами:

**ALIGN=blееdleft** прижимает содержимое ячейки вплотную к левому краю.

**ALIGN=left** выравнивает содержимое ячейки по левому краю с учетом отступа, заданного атрибутом СЕLLPADDING.

**АLIGN=сеnter** располагает содержимое ячейки по центру.

**АLIGN=right** выравнивает содержимое ячейки по правому краю с учетом отступа, заданного атрибутом СЕLLPADDING.

<HTML>

<BODY>

<TABLE BORDER WIDTH=100%>

<TR>

<TD ALIGN=left>Текст или данные</TD>

<TD ALIGN=center>Текст или данные</TD>

<TD ALIGN=right>Текст или данные</TD>

</TR>

<TR>

<TD ALIGN=right>Текст или данные</TD>

<TD ALIGN=center>Текст или данные</TD>

<TD ALIGN=left>Текст или данные</TD>

</TR>

<TR>

<TD ALIGN=right>Текст или данные</TD>

<TD ALIGN=right>Текст или данные</TD>

<TD ALIGN=right>Текст или данные</TD>

</TR>

<TR>

<TD ALIGN=center>Текст или данные</TD>

<TD ALIGN=center>Текст или данные</TD>

<TD ALIGN=center>Текст или данные</TD>

</TR>

<TR>

<TD ALIGN=left>Текст или данные</TD>

<TD ALIGN=left>Текст или данные</TD>

<TD ALIGN=left>Текст или данные</TD>

</TR>

</TABLE>

</BODY>

</HTML>

Атрибут VALIGN осуществляет выравнивание текста и *графики* внутри ячейки по вертикали. Вертикальное выравнивание может быть задано несколькими способами:

**VALIGN=top** выравнивает содержимое ячейки по ее верхней границе.

**VALIGN=middle** центрирует содержимое ячейки по вертикали.

**VALIGN=bottom** выравнивает содержимое ячейки по ее нижней границе.

<HTML>

<BODY>

<CENTER>

<TABLE BORDER WIDTH=90%>

<TR>

<TD WIDTH=100>Атрибут VALIGN осуществляет

выравнивание текста и графики внутри

ячейки по вертикали.</TD>

<TD VALIGN=top>верх,</TD>

<TD VALIGN=middle>середина,</TD>

<TD VALIGN=bottom>низ.</TD>

</TR>

<TR VALIGN=top>

<TD VALIGN=top>Выравнивает содержимое

ячейки по ее верхней границе.</TD>

<TD>верх,</TD>

<TD>верх,</TD>

<TD>верх.</TD>

</TR>

<TR VALIGN=middle>

<TD VALIGN=middle>Центрирует содержимое

ячейки по вертикали.</TD>

<TD>середина,</TD>

<TD>середина,</TD>

<TD>середина.</TD>

</TR>

<TR VALIGN=bottom>

<TD VALIGN=bottom>Выравнивает содержимое

ячейки по ее нижней границе.</TD>

<TD>низ,</TD>

<TD>низ,</TD>

<TD>низ.</TD>

</TR>

</TABLE>

</CENTER>

</BODY>

</HTML>

##### Атрибут BORDER

В теге <ТАВLЕ> часто определяют, как будут выглядеть рамки, то есть линии, окружающие ячейки *таблицы* и саму *таблицу*. Если вы не зададите рамку, то получите *таблицу* без линий, но пространство под них будет отведено. Того же результата можно добиться, задав <ТАВLЕ ВОRDER=0>. Иногда хочется сделать границу потолще, чтобы она лучше выделялась. Можно для привлечения внимания к рисунку или тексту задать исключительно жирные границы. При создании вложенных *таблиц* приходится делать для разных *таблиц* границы различной толщины, чтобы их легче было различать.

##### Атрибут CELLSPACING

Атрибут СЕLLSPACING определяет ширину промежутков между ячейками в пикселах. Если этот атрибут не указан, по умолчанию задается величина, равная двум пикселам. С помощью атрибута СЕLLSPACING= можно размещать текст и *графику* там, где вам нужно. Если вы хотите оставить пустое место, можно вписать в ячейку пробел.

<HTML>

<BODY>

<CENTER>

<TABLE BORDER CELLSPACING=20>

<TR>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

</TR>

<TR>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

</TR>

</TABLE>

<TABLE BORDER CELLSPACING=10>

<TR>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

</TR>

<TR>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

</TR>

</TABLE>

<TABLE BORDER CELLSPACING=0>

<TR>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

</TR>

<TR>

<TD>Текст или данные</TD>

<TD></TD>

<TD>Текст или данные</TD>

</TR>

</TABLE>

</CENTER>

</BODY>

</HTML>

##### Атрибут BGCOLOR

Данный атрибут позволяет установить цвет фона. В зависимости от того, с каким тегом (сTABLE, TR, TD ) он применяется, цвет фона может быть установлен для всей *таблицы*, для строки или для отдельной ячейки. Значением данного атрибута является RGB-код или стандартное название цвета.

<HTML>

<BODY>

<CENTER>

<TABLE BORDER BGCOLOR=yellow>

<TR BGCOLOR=blue>

<TD>Текст или данные</TD>

<TD BGCOLOR=red>Текст или данные

</TD>

<TD>Текст или данные</TD>

</TR>

<TR BGCOLOR=green>

<TD>Текст или данные</TD>

<TD>Текст или данные</TD>

<TD BGCOLOR=lime>Текст или данные

</TD>

</TR>

</TABLE>

</CENTER>

</BODY>

</HTML>

##### Атрибут BACKGROUND

Данный атрибут задает фоновое изображение для *таблиц*. Применим к тегам TABLE и TD. Его значением является URL файла с фоновым изображением. Применение этого атрибута рассматривается ниже.

#### Использование таблиц в дизайне страницы

*Таблицы* хороши тем, что при желании можно сделать их границы невидимыми. Это позволяет с помощью тега <ТАВLЕ> красиво размещать на странице текст и *графику*. Пока тег <ТАВLЕ> остается единственным мощным средством форматирования в HTML. Дизайнеры Web-страниц сейчас обладают практически той же свободой в отношении использования "пустого пространства", что и создатели печатных страниц. *Таблицы* лучше всего помогают отойти от иерархического размещения текста на Web-страницах.

Если браузер поддерживает *таблицы*, он обычно правильно отображает наиболее интересные эффекты, полученные с их помощью ([открыть](http://www.intuit.ru/EDI/06_02_17_3/1486333223-7320/tutorial/100/objects/5/files/example2.htm))

<HTML>

<BODY>

<CENTER>

<TABLE CELLPADDING="10" CELLSPACING="0"

BORDER="16">

<TR>

<TD ALIGN="center">

<H2>Интернет-Университет

Информационных Технологий</H2>

<H3>Добро пожаловать!</H3>

<TABLE BORDER WIDTH="100%">

<TR>

<TD ALIGN="center"><I>Учебный курс

"Основы Web-технологий"</I></TD>

</TR>

</TABLE>

</TD>

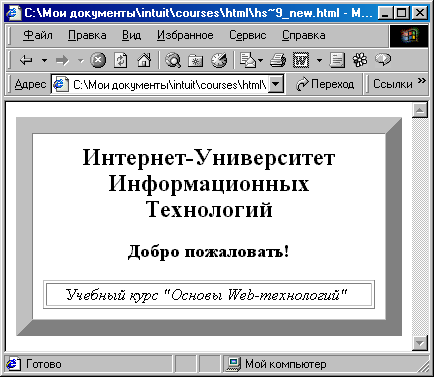
</TR>

</TABLE>

</CENTER>

</BODY>

</HTML>



**Рис.** Использование таблиц в дизайне

##### Создание разноцветных таблиц

Есть несколько способов раскрасить *таблицу*, в основном они зависят от используемого браузера.

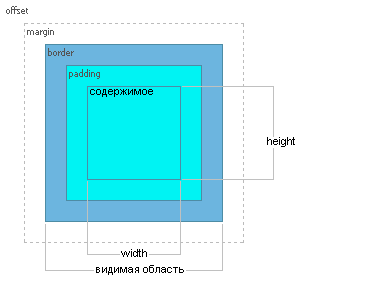
Цветные границы в Netscape Navigator. Вы не только можете окружить *таблицу* красивой рамкой, но еще и задать для нее цвет, отличный от цветов текста и фона. Создайте простой серый GIF (или любой GIF, который вы хотели бы иметь в качестве фона) и определите его в теге <ВODY> как фон страницы. Затем задайте цвет фона страницы. В результате ваш тег <ВОDY> будет выглядеть примерно так:

<BODY ВАСКGROUND="image.gif" ВGCOLOR="#FF0000">

Вы создали двойной фон - GIF и заданный цвет. В результате фоновый цвет будет виден на всех границах *таблиц* и горизонтальных линиях ( <НR> ). Вне зависимости от того, является ваш фоновый GIF серым или нет, цветные линии и границы *таблиц* будут заметно выделяться. Если фоновый GIF устроен не слишком сложно, время загрузки страницы возрастет лишь немного.

**Блоки**

Методика форматирования Web-страниц и размещения на них содержимого в отведенном для этого пространстве описывается в CSS в виде пространственной модели (Box Formatted Model). Согласно этой модели, пространство, в терминологии создателей CSS - бокс (box), в котором размещено содержимое или часть содержимого Web-страницы (тексты, изображения), можно представить в виде объекта, обладающего набором специфических характеристик. Необходимо подчеркнуть, что боксом считается не страница или ячейка таблицы, а лишь пространство, отведенное для содержимого или определенной его части. Формально каждый бокс имеет ядро, представляющее собой содержательную часть страницы или ячейки таблицы, и набор окружающих ядро элементов, характеристики которых определяют внешний вид бокса. Под последними понимаются поля отступа, границы и т. д.



Ширина бокса состоит из суммы ширин содержимого, отступов (padding), границы (border) и величины полей (margin). Параметры, определяющие размеры каждого из этих элементов, задаются отдельно в зависимости от желания авторов Web-страниц. Имеющиеся на странице боксы составляют блок.

***Отступы*** регулируются следующими **свойствами**:

|  |  |
| --- | --- |
| padding-top  padding-left  padding-bottom  padding-right | Задают размеры отступов соответственно сверху, слева, снизу и справа. Значение можно задавать в единицах измерения длины и в процентах от ширины блока. |
| padding | В этом случае все отступы одинаковы |

***Поля*** регулируются аналогично **свойствами**:

|  |  |
| --- | --- |
| margin-top  margin-left  margin-bottom  margin-right | Задают размеры полей соответственно сверху, слева, снизу и справа. Значение можно задавать в единицах измерения длины и в процентах от ширины блока. |
| margin | Все поля одинаковой ширины |

***Границы*** определяют свойством *border*.

|  |  |
| --- | --- |
| border-top-color  border-left-color  border-bottom-color  border-right-color  border-color | Задают цвет рамки для верхней, левой, нижней, правой или всех границ одновременно. Значением являются единицы определения цвета. Если цвет не указан, но указан стиль границы, то используется цвет самого элемента. |
| border-top-width  border-left-width  border-bottom-width  border-right-width  border-width | Задают ширину рамки для верхней, левой, нижней, правой или всех границ одновременно. Значением являются единицы длины. Кроме значений длины можно использовать три фиксированных значения:  thin-тонкая рамка  medium-рамка средней толщины  thick-толстая рамка |
| border-top-style  border-left-style  border-bottom-style  border-right-style  border-style | Задают стиль (вид) границы для верхней, левой, нижней, правой или всех границ одновременно. Возможны следующие значения: |
| border-top  border-left  border-bottom  border-right | Эти обобщенные свойства задают свойства границ верхней, левой, нижней или правой, если они одинаковые. |

Разрешается использовать одно, два, три или четыре значения, разделяя их между собой пробелом. Эффект зависит от количества аргументов:

|  |  |
| --- | --- |
| 1  2  3  4 | Стиль рамки будет задан для всех сторон элемента.  Первый аргумент устанавливает стиль верхней и нижней границы, второй аргумент — левой и правой.  Первый аргумент задает стиль верхней границы, второй — одновременно левой и правой стороны, а третий — нижней границы.  Поочередно устанавливается стиль верхней, правой, нижней и левой границы. |

# *Обводка* **определяется свойством** ***outline***

Универсальный атрибут, одновременно устанавливающий цвет, стиль и толщину внешней границы на всех четрырех сторонах элемента. В отличие от линии, задаваемой через border, атрибут outline не влияет на положение блока и его ширину.

|  |  |
| --- | --- |
| outline-color | Задает цвет линии в любом допустимом для CSS формате |
| outline-style | Стиль линии |
| outline-width | Толщина границы |

Возможна сокращенная запись, указывающая вид границы и другие свойства CSS. В данном случае запись параметров выглядит следующим образом:

<div style="border: solid thick green">Зеленая граница</div>

Как видно из примера, три свойства, задающие стиль границы, её ширину и цвет, свелись к одному - border, а три соответствующих значения пишутся через пробел.

**Свойства списков**

Таблицы стилей позволяют управлять отображением списков: задавать гарнитуру, размер и цвет шрифта, выбирать вид маркеров.

|  |  |  |  |
| --- | --- | --- | --- |
| list-style-type | Задают маркеры для списков. Часто используемые значения:   |  |  | | --- | --- | | * Square * circle * disc   1. decimal | i. lower-roman  I. upper-roman  a. lower-alpha  A. upper-alpha  none | |
| list-style-image | Задает маркер в виде картинки (изображения). Например:  {list-style-image:url(“marker.gif”)} |
| list-style-position | Определяет, как будет размещаться маркер относительно текста. Допускаются значения:  outside – маркер вынесен за границу элемента списка  inside – маркер обтекается текстом |

**Другие группы стилевых свойств**

|  |  |
| --- | --- |
| display | Определяет отображение элементов документа. Используемые значения:  none – отключает вывод элемента на экран  block – позволяет преобразовать строчный элемент в блок  inline - позволяет преобразовать абзац в строчный элемент  list-item – заставляет «вести» элемент как список |
| cursor | Определяет форму указателя мыши, которую он принимает при наведении на форматируемый элемент. Используемые значения:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | default |  | n-resize |  | se-resize | |  | pointer |  | s-resize |  | sw-resize | | css_cursor_crosshair | crosshair |  | e-resize |  | nw-resize | |  | move |  | w-resize |  | ne-resize | |  | help |  | text |  | progress |   **Примечание**: внешний вид курсора может изменяться в зависимости от операционной системы, браузера и темы оформления. |

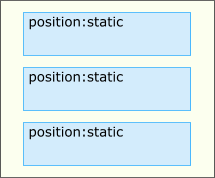
**ПОЗИЦИОНИРОВАНИЕ**

В HTML существует, по большому счету, четыре способа раскладывать блоки по странице: прямой поток, позиционирование, float'ы и таблицы. К сожалению, ни один из них не дает полного набора средств, которыми можно было бы сверстать сложные макеты. Поэтому используются они обычно все вместе, да еще и с изрядной долей приёмов для обхода несовместимостей в реализациях стандарта.

Суть позиционирования очень проста: любой элемент можно расположить в любом месте страницы, задав ему точные координаты. Это может быть не только <div>,возможно позиционировать хоть <b>, если очень захочется.

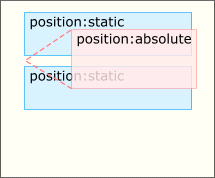
Сложности начинаются тогда, когда необходимо делать растянутые колонки, панели и прочее. Другими словами - завязывать положение и размеры боксов друг на друга.

Существуют четыре способа позиционирования боксов:

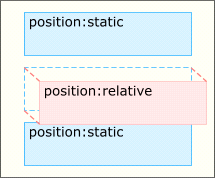


**Static**

Это способ используется по умолчанию, можно сказать, отстутствие какого бы то ни было специального позиционирования, а просто выкладывание блоков одного за другим сверху вниз. Этот порядок как раз и есть прямой поток.

**Absolute**

Бокс с абсолютным позиционированием располагается по заданным координатам, а из того места, где он должен был бы быть, он удаляется, и в этом месте сразу начинают раскладываться следующие боксы. Говорят, что он "исключается из потока".

**Relative**

Такой бокс можно сдвинуть относительно того места, где он был бы в потоке, но при этом из потока он не исключается, а продолжает занимать там свое место. То есть сдвигается со своего места он только визуально, а положение всех боксов вокруг него никак не меняется.

**Fixed**

Ведет себя так же, как absolute, но при этом он не скролится вместе с остальной страницей.

#### Карты изображений

Наряду с использованием изображений в качестве иллюстраций к тексту или элементов оформления Web-страницы, в HTML предусмотрена возможность создания карт изображений (imagemap), отдельные области которого могут являться гиперссылками на другие разделы или страницы Web-сайта. В общем виде это соответствует использованию изображения в качестве гиперссылки с тем отличием, что на одной карте изображений можно создать несколько несовпадающих областей, и соответственно, гиперссылок.

Применение таких карт изображений удобно для создания географических справочных систем, путеводителей, карт погоды. Также карты изображений находят широкое применение при создании сложных графических меню.

**КАРТЫ-ИЗОБРАЖЕНИЯ**

В HTML предусмотрены два варианта обработки карт изображений: обработка карты изображения непосредственно браузером и передача на сервер координат указателя мыши для дальнейшей обработки.

##### Карты изображений, обрабатываемые браузером

Автономные (обрабатываемые браузером) карты изображений описываются с помощью атрибута usemap тега <img>:

<img src="figure\_1.png" usemap="#coordinates">

Где "figure\_1.png" - имя файла, содержащего изображение, "#coordinates" - ссылка на часть HTML-документа, описывающего координатные области.

Координатные области карты изображений определяются при помощи тегов <map> и <area>.

Тег <area> описывает координаты отдельной области изображения, параметры которой и адрес гиперссылки задаются при помощи атрибутов shape, coords и href.

Атрибут shape определяет форму области-ссылки. По умолчанию ему присваивается значение shape="rect" - прямоугольник. Также область может быть описана в форме окружности ( shape="circle" ) или многоугольника ( shape="poly" ).

Атрибут coords определяет размеры области. В зависимости от типа размечаемой области может меняться значение этого атрибута. Так для shape="rect" указываются две пары координат (левого верхнего и правого нижнего угла прямоугольника) в пикселях. Для shape="circle" указываются координаты центра окружности и ее радиус, а для shape="poly" последовательно указываются координаты вершин многоугольника.

Атрибут href задает URL-адрес ссылки для перехода в случае щелчка мышью на выбранной области.

Заданные при помощи тегов <area> координатные области изображения ограничиваются при помощи тегов <map> - </map>.

В следующем примере задана карта изображений, на которой в соответствии с изображением выделены прямоугольная, круглая и треугольная координатная области.

<html>

<body>

<img src="figure\_1.png" width="297" height="210" border="0"

alt="Карта изображений" usemap="#coordinates">

<map name="coordinates">

<area shape="rect" coords="130,194,270,130" href="rect.html" alt="Прямоугольник">

<area shape="circle" coords="73,90,40" href="circle.html" alt="Круг">

<area shape="poly" coords="168,109,280,109,223,13" href="polygon.html"

alt="Полигон (треугольник)">

</map>

</body>

</html>

Для корректной работы следующего примера необходимо создать в рабочей папке файлы rect.html, circle.html, polygon.html. Например, файл rect.html:

<html>

<body>

Вы выбрали прямоугольник

</body>

</html>

Обратите внимание, что атрибут alt (заменяющий текст) может использоваться как для всего изображения, так и для каждой его области.

В следующем примере координатные области пересекаются, обратите внимание, что для карт изображений действует принцип, согласно которому при пересечении двух или более координатных областей активной будет область, описанная первой. В данном примере есть определенное несоответствие, при щелчке мышью на окружности в области пересечения ее с прямоугольником загрузится файл rect.html.

<html>

<body>

<img src="figure\_2.png" width="297" height="210" border="0" usemap="#coordinates">

<map name="coordinates">

<area shape="rect" coords="84,173,224,109" href="rect.html" alt="Прямоугольник">

<area shape="circle" coords="130,110,40" href="circle.html" alt="Круг">

<area shape="poly" coords="105,138,217,138,161,42" href="polygon.html"

alt="Полигон (треугольник)">

</map>

</body>

</html>

В следующем примере порядок описания координатных областей приведен в соответствие с изображением на экране.

<html>

<body>

<img src="figure\_2.png" width="297" height="210" border="0" usemap="#coordinates">

<map name="coordinates">

<area shape="circle" coords="130,110,40" href="circle.html" alt="Круг">

<area shape="poly" coords="105,138,217,138,161,42" href="polygon.html"

alt="Полигон (треугольник)">

<area shape="rect" coords="84,173,224,109" href="rect.html" alt="Прямоугольник">

</map>

</body>

</html>

##### Карты изображений, обрабатываемые на сервере

Карты изображений, обрабатываемые на сервере, описываются с помощью атрибута ismap тега <img>, располагающегося между тегами <a> - </a>, например:

<a href="tryhtml\_ismap.htm">

<img src="figure\_1.png" ismap>

</a>

Где "tryhtml\_ismap.htm" – файл на сервере, описывающий координатные области изображения.

При использовании серверной обработки карт изображений на сервер передаются координаты курсора относительно изображения. В следующем примере при перемещении курсора по поверхности изображения в строке состояния вместе с именем файла обработчика отображаются координаты курсора.

<html>

<body>

<a href="tryhtml\_ismap.htm">

<img src="figure\_1.png" width="297" height="210" border="0" alt="Карта

изображений" ismap>

</a>

</body>

</html>

Работая с картами изображений необходимо помнить, что при отключении отображения графики (например, при плохих каналах связи) работа с ними практически невозможна. При помощи атрибутов alt можно задать имена выделенным областям, но ориентироваться по такой карте изображений будет достаточно трудно, поэтому необходимо использовать карты изображения только в тех случаях, когда их присутствие целесообразно, например, в географических справочных системах, учебных системах, где необходима наглядность, и т.п. В этом случае признаком хорошего тона считается создание текстового меню, дублирующего ссылки на карте изображении.

**1.3 Формы. Фреймы. Таблицы стилей**

### HTML-формы

***Формы*** были созданы и используются в WWW для получения отклика пользователя на предоставленную информацию и сбора данных о пользователе. После заполнения пользователем *формы* и запуска процесса ее обработки, *информация* из нее попадает к программе, работающей на сервере. Простота использования тега <MAILTO:> в *формах* позволяет даже владельцам небольших страниц получать отклик от своих читателей. Для обработки большого количества откликов используются программы, поддерживающие Common *GatewayInterface* (*CGI*) и расположенные на сервере, в *адрес* которого поступают отклики. Таким образом *пользователь* может интерактивно взаимодействовать с Web-сервером через *Internet*.

### Задание формы — элемент FORM

Элемент FORM обозначает документ как *форму* и определяет границы использования других тегов, размещаемых в *форме*. *Тег* <FORM>определяется последовательностью тегов *<INPUT>* , размещенных внутри пары <FORM> и </FORM>. В *форме* используется как метод ( method ), так и действие ( action ) для описания обработки данных, вводимых пользователем в *форму*. Метод ( GET или POST ) определяет, как должны обрабатываться *входные данные* из *формы*, а действие указывает на *URI* (*Uniform* *Resource* *Identifier*) программы, ответственной за обработку этих данных.

<FORM METHOD=post

ACTION=mailto:yourname@your.email.address>

#### Определение элементов управления формы — тег <INPUT>

Данный тег используют для определения области внутри *формы*, куда вводятся данные. Он формирует *поле для ввода информации*пользователем. Это может быть текстовое поле, опция, изображение или кнопка. Вид *поля ввода* определяется значением атрибута TYPE.

##### Атрибут TYPE=text

Когда пользователю необходимо ввести небольшое количество текста (одну или несколько строк), используется тег *<INPUT>* , и атрибут TYPE устанавливается в значение text. Это значение принято по умолчанию и указывать его необязательно. Кроме того, задается атрибут NAME для определения наименования переменной *поля*.

Ваше имя <INPUT NAME=Name SIZE=35>

Имеется еще три дополнительных атрибута, которые можно использовать. Первый называется MAXLENGTH, он ограничивает число символов, вводимых пользователем в текущее *поле*. По умолчанию данное число не ограничено. Вторым атрибутом является SIZE, определяющий размер видимой на экране области, занимаемой текущим *полем*. Значение по умолчанию определяется типом браузера. Если значение MAXLENGTH больше, чем SIZE, браузер будет прокручивать данные в окне. Последним из дополнительных атрибутов является атрибут VALUE, обеспечивающий начальное значение *поля ввода*.

##### Атрибут TYPE=checkbox

Для создания независимых флагов в *формах* HTML используется тег *<INPUT>* со значением атрибута TYPE=checkbox. В зависимости от содержания *формы* пользователь может отметить несколько флагов. Когда *форма* использует тег *<INPUT>* со значением атрибута CHECKBOX, в нем должны присутствовать и атрибуты NAME, и VALUE. Атрибут NAME указывает на наименование данного *поля (флага) ввода*. В атрибуте VALUE будет содержаться значение *поля*.

<BR>Россия<INPUT NAME="Страна" TYPE=checkbox

VALUE="Россия">

Страны СНГ<INPUT NAME="Страна" TYPE=checkbox

VALUE="СНГ">

В некоторых случаях необходимо инициализировать данный флаг, как уже отмеченный. В таких случаях тег *<INPUT>* должен содержать атрибут CHECKED.

##### Атрибут TYPE=radio

В некоторых случаях требуется организовать выбор одного из нескольких возможных значений. Для создания *формы* ввода при выборе пользователем одного значения из нескольких возможных необходимо использовать тег *<INPUT>* с атрибутом TYPE=radio. Когда в *форме* применяется данный атрибут, в теге *<INPUT>* должны быть указаны атрибуты NAME и VALUE. Атрибут NAME указывает наименование соответствующего *поля* (кнопки). Атрибут VALUE содержит значение *поля*.

<BR>Пол мужской<INPUT NAME="Пол" TYPE=radio

VALUE="Мужской">

Пол женский<INPUT NAME="Пол" TYPE=radio

VALUE="Женский">

##### Атрибут TYPE=image

В зависимости от содержимого *формы* может случиться так, что пользователю потребуется щелкнуть мышью на изображении, чтобы завершить работу с *формой*. Для этого программисты используют тег *<INPUT>* с атрибутом TYPE=image. Когда пользователь щелкает мышью по изображению, браузер сохраняет координаты соответствующей точки экрана. Далее он "обрабатывает" введенную в *форму*информацию. Когда *форма* использует атрибут image, тег *<INPUT>* должен содержать также атрибуты NAME и SRC. NAME указывает наименование *поля ввода формы*. Атрибут SRC содержит URI файла — источника изображения. Атрибут ALIGN является дополнительным и используется аналогично тому же атрибуту тега <IMG>.

<BR>Выберите точку<INPUT TYPE=image NAME=point

SRC=image.gif>

##### Атрибут TYPE=password

Если в *форме* требуется организовать ввод пароля, то атрибут TYPE можно установить в значение password ( TYPE=password ). Используя данный тип, можно организовать ввод пароля без вывода на экран составляющих его символов. При этом следует помнить, что введенные данные передаются по незащищенным каналам связи и могут быть перехвачены.

<BR>Подпись<INPUT NAME=login>Пароль

<INPUT TYPE=password NAME="Слово">

##### Атрибут TYPE=reset

Когда пользователь заполняет *форму*, ему может потребоваться начать все сначала. На такой случай существует кнопка Reset, по которой пользователь может щелкнуть мышью, чтобы вернуться к первоначальным значениям *полей*. Когда пользователь выбирает данную кнопку, *форма* восстанавливает первоначальные значения всех элементов, в которых присутствует атрибут TYPE=reset. Для создания кнопки Reset используется тег *<INPUT>* с атрибутом TYPE=reset. Браузер в свою очередь будет выводить изображение данной кнопки. Если в *форме*используется атрибут reset, тег *<INPUT>* может дополнительно содержать атрибут VALUE. Данный атрибут определяет надпись на изображении кнопки.

<INPUT TYPE=reset VALUE="Очистить форму">

##### Атрибут TYPE=submit

Используя *форму* HTML для ввода информации от пользователя, необходимо обеспечить пользователю возможность завершить ввод данных. Для этого используется тег *<INPUT>* с атрибутом TYPE=submit. Браузер, в свою *очередь, выводит* данный элемент, как кнопку, по которой пользователь может щелкнуть, чтобы завершить процесс редактирования. Когда в *форме* используется тег *<INPUT>* с атрибутом submit, данный элемент может содержать два дополнительных атрибута: NAME и VALUE. Атрибут NAME хранит название переменной *поля* в вашей *форме*. Атрибут VALUE — определяет значение элемента формы, которое будет отправлено на сервер или получено с помощью клиентских скриптов.

<BR><INPUT TYPE=submit

VALUE="Отправить сообщение">

##### Атрибут TYPE=hidden

Скрытые *поля*. Добавление в тег *INPUT* атрибута TYPE=hidden позволит включить в отправляемую *форму* значения атрибутов NAME и VALUE, которые пользователь изменить не может. Такие метки полезны при наличии нескольких *форм* для дальнейшей обработки данных

#### Создание многострочных областей ввода текста — тег <TEXTAREA>

В зависимости от типа *формы* может потребоваться организовать ввод большого количества текста. В таких случаях используется тег ***<TEXTAREA>*** для создания текстового поля из нескольких строк. Данный тег использует три атрибута: COLS, NAME и ROWS.

##### Атрибут COLS

Указывает (число символов) число колонок, содержащихся в текстовой области.

##### Атрибут NAME

Определяет наименование поля.

##### Атрибут ROWS

Задает количество видимых строк текстовой области.

<BR><TEXTAREA NAME="тема" COLS="38" ROWS="3">

</TEXTAREA>

#### Использование списков в форме — тег <SELECT>

Когда формы HTML становятся более сложными, в них часто включают списки с прокруткой и выпадающие *меню*. Для этого используют тег ***SELECT*** с атрибутом TYPE=select . Для определения списка пунктов используют тег *<OPTION>* . Тег *<SELECT>* поддерживает три необязательных атрибута: MULTIPLE, NAME и SIZE.

##### Атрибут MULTIPLE

Позволяет выбрать более чем одно наименование.

##### Атрибут NAME

Определяет наименование объекта.

##### Атрибут SIZE

Определяет число видимых пользователю пунктов списка. Если в *форме* установлено значение атрибута SIZE=1, то браузер выводит на экран список в виде выпадающего *меню*. В случае SIZE > 1 браузер представляет на экране обычный список.

В *форме* может использоваться тег *<OPTION>* только внутри тега *<SELECT>* . Эти теги поддерживают два дополнительных атрибута: SELECTED и VALUE.

##### Атрибут SELECTED

Используется для первоначального выбора значения элемента по умолчанию.

##### Атрибут VALUE

Указывает на значение, возвращаемое *формой* после выбора пользователем данного пункта. По умолчанию значение *поля* равно значению тега *<OPTION>*.

<BR>Выбор

<SELECT NAME="Выбор">

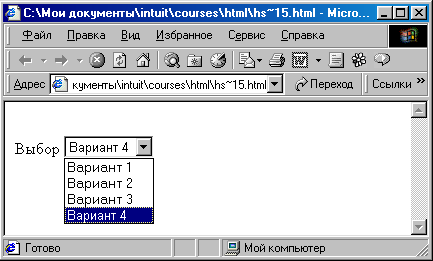
<OPTION>Вариант 1

<OPTION>Вариант 2

<OPTION VALUE="Вариант 3">Вариант 3

<OPTION SELECTED>Вариант 4

</SELECT>



**Рис.** Выпадающее меню

### Фреймы

Один из способов выдать сразу несколько файлов *HTML* на экран пользователя - это открыть несколько окон браузера. Другой *путь* состоит в том, чтобы разбить окно на несколько разделов. Эти *разделы* называются **фреймами** или кадрами. В каждом фрейме показывается свой *HTML*-документ. Каждый *фрейм* может иметь свои полосы прокрутки, ссылки, графические изображения и т. д. Фреймы могут функционировать независимо или влиять друг на друга, используя ссылки, указывающие на другие фреймы.

#### Контейнер <FRAMESET>

Web-страница, которая разделена на фреймы, называется документом группы фреймов. Документы группы фреймов содержат стандартный заголовок, задаваемый тегом HEAD, но в отличии от стандартных HTML-документов, они не содержат тега BODY. Вместо него используется контейнер (т. е. парный тег) <FRAMESET>, который применяется для определения строк и столбцов отдельных фреймов, каждый из которых обозначается тегом <FRAME>.

Если включить контейнер BODY в документ, где используется контейнер FRAMESET, то кадры будут проигнорированы программой просмотра, и информация, содержащаяся в документах, задаваемых тегами <FRAME>, не будет выведена. Будет показана только информация, содержащаяся в контейнере BODY.

Внутри контейнера <FRAMESET> ... </FRAMESET> могут располагаться только теги <FRAME> или другие контейнеры FRAMESET.

#### Определение параметров кадров

Тег <FRAMESET> имеет два главных атрибута: ROWS и COLS, задающих разбиение на строки и столбцы соответственно. Ниже приведен вид контейнера FRAMESET:

<FRAMESET ROWS="список\_значений" COLS="список\_значений">

...

</FRAMESET>

Можно определить любое число рядов и столбцов; необходимым условием является указание хотя бы одного из атрибутов ROWS или COLS.

Кадр не может быть единственным: если вы определили единственный ряд и единственный столбец, то программа просмотра проигнорирует контейнер FRAMESET, и экран останется пустым. Если определены по крайней мере два ряда или два столбца, другой атрибут может быть опущен (ему по умолчанию будет присвоено значение, равное 100%).

Значение атрибута ROWS или COLS представляет собой строку, содержащую список значений в пикселах, процентах или относительных единицах, разделенных запятыми. Количество рядов или столбцов кадров определяется числом этих значений.

**Пример**

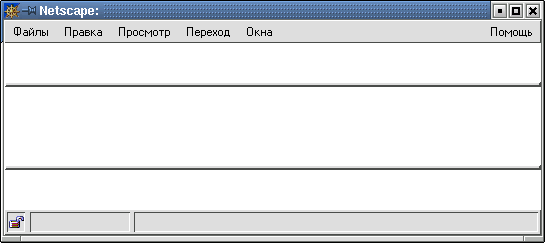
Для задания сетки кадров из трех рядов, высота которых 100, 240 и 140 пикселов соответственно, используйте тег

<FRAMESET ROWS="100,240,140">

Задание высоты ряда в пикселах, однако, является плохим стилем, так как при этом не учитывается тот факт, что окна браузеров могут иметь самую разную величину. В абсолютных единицах стоит указывать размеры кадра лишь для размещения небольших изображений, в остальных же случаях лучше пользоваться относительными величинами.

**Пример**

Тег <FRAMESET ROWS="25%, 50%, 25%>" задаст три кадра, размером по 25%, 50% и 25% от высоты окна браузера.

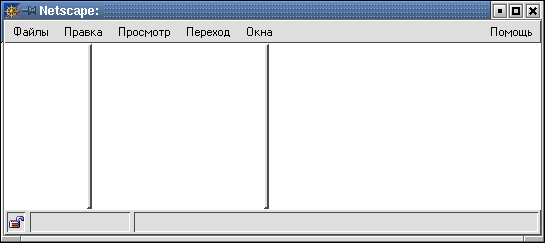


Относительно точности указания размеров фреймов в процентах можно не беспокоиться: если сумма значений не равна 100%, то масштаб кадров будет пропорционально изменен.

Задание параметров кадров в относительных единицах выглядит примерно так:

<FRAMESET COLS="\*,2\*,3\*">

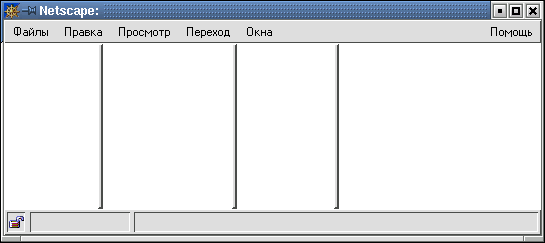
Символ \* обозначает пропорциональное деление окна программы просмотра. В данном примере окно будет разделено на три вертикальных кадра, первый из которых будет иметь ширину в 1/6, второй - в 2/6 (или 1/3) и третий - в 3/6 (или 1/2) от ширины окна браузера. Единица при указании относительных значений может быть опущена.



Указание значений атрибутов ROWS и COLS может быть и смешанным, включающим любое сочетание *абсолютных размеров*, процентных отношений и относительных значений, например,

<FRAMESET COLS="100,25%,\*,2\*">

Здесь первому кадру присвоено абсолютное значение в 100 пикселов по ширине, второму - 25% от ширины окна. Оставшееся пространство делится между третьим и четвертым кадрами в пропорции 1 к 2.



Приоритеты в указаниях значений атрибутов таковы: в первую очередь (слева направо) отводится место для кадра с абсолютным значением, затем - для кадра со значением в процентах, и в последнюю очередь - для кадров с относительными величинами.

Если вы пользуетесь *абсолютными величинами* в атрибутах ROWS и COLS, не делайте такие кадры большими - ведь они должны поместиться в окно браузера любого размера. Совместно с такими кадрами для лучшей балансировки рекомендуется использовать хотя бы один кадр, определенный в процентах или в относительных величинах.

При определении обоих атрибутов получается сетка кадров, например, тег

<FRAMESET ROWS="\*,2\*,\*" COLS="2\*,\*">

задает сетку из трех рядов и двух столбцов. В данном примере первый и последний ряды занимают по 1/4, второй ряд - половину от высоты окна. Первый столбец занимает 2/3, а второй - 1/3 ширины окна браузера.

#### Тег <FRAME>

Тег <FRAME> определяет отдельный кадр. Он должен располагаться внутри контейнера FRAMESET.

**Пример**

<FRAMESET ROWS="\*,2\*">

<FRAME>

<FRAME>

</FRAMESET>

Заметьте, что этот тег не является контейнером и, в отличие от тега <FRAMESET>, не имеет закрывающего тега.

Число тегов <FRAME> **обязательно** должно быть равно числу кадров, определенных в теге <FRAMESET>. В нашем примере определено два кадра, поэтому контейнер содержит соответственное количество тегов <FRAME>. Пока кадры ничем не заполнены.

<FRAME SRC="URL" NAME="имя\_окна"

SCROLLING=yes|no|auto MARGINWIDTH="значение"

MARGINHEIGHT="значение" NORESIZE>

Использовать все атрибуты необязательно. Чаще всего вы будете пользоваться только одним атрибутом - SRC. Строка

<FRAME SRC="URL">

определяет URL-адрес содержимого кадра. Это обычно файл HTML-документа, расположенный в том же каталоге, что и документ, содержащий контейнер FRAMESET, например,

<FRAME SRC="first.html">

Этот документ должен быть полноценным HTML-документом, т. е. содержать все обязательные части (контейнеры HTML, HEAD, BODY и т. д.).

В случае, если программа просмотра не сможет найти указанный файл, кадр не будет построен, и браузер выведет сообщение об ошибке. Если же в теге вовсе не указан атрибут SRC, кадр будет создан и оставлен пустым.

Текст, заголовки, графика и другие элементы не могут напрямую включаться в документ с кадрами. Все они должны вводиться только с помощью указания URL-адреса элемента. Если контейнер FRAMESET содержит "инородное тело", оно будет выведено, а кадры полностью проигнорированы.

Атрибут MARGINWIDTH=n задает размещение слева и справа от содержимого кадра областей свободного пространства высотой по n пикселов, а MARGINHEIGHT=n, соответственно, сверху и снизу.

Значения этих атрибутов всегда должны указываться в абсолютных значениях (пикселах). Так, тег

<FRAME MARGINHEIGHT="5" MARGINWIDTH="7">

создаст внутреннюю рамку на верхней и нижней границах кадра шириной в 5 пикселов, а на левой и правой границе - шириной в 7 пикселов. Внутри этой рамки данные выводиться не будут. Эти атрибуты не имеют ничего общего с рамкой кадра, определяемой браузером, либо задаваемой при помощи атрибута BORDER.

К построенным вами кадрам автоматически добавляются полосы прокрутки, если содержание кадра больше его размера. Иногда это может нарушить эстетику страницы, поэтому в HTML предусмотрен атрибут SCROLLING тега <FRAME>, имеющий следующий формат:

<FRAME SCROLLING="yes|no|auto">

Этот атрибут может принимать одно из трех значений: yes, no и auto. Последнее значение подразумевается по умолчанию, т. е. когда атрибут не определен. Если указано значение yes, полоса прокрутки появится в любом случае, значение no запрещает появление полосы. Определение атрибута SCROLLING, например, может быть следующим:

<FRAME SCROLLING=yes>

По умолчанию размеры кадров могут легко изменяться читателями, однако понятно, что это может сильно нарушить авторский замысел. Поэтому вы, скорее всего, захотите использовать атрибут NORESIZE тега <FRAME>, запрещающий возможность "перекраивания" вашей страницы:

<FRAME NORESIZE>

Этот атрибут не имеет значений. Указав его в одном кадре, вы тем самым запретите изменять размеры и всех смежных кадров. Как правило, этого бывает достаточно, чтобы "закрепить" рамки всех кадров страницы на месте.

Когда вы захватываете рамку кадра мышью, то указатель становится двунаправленной стрелкой, если перемещение рамки возможно. В противном случае, т. е. когда использован атрибут NORESIZE, двунаправленная стрелка не появляется.

Для определения рамки кадра в HTML существуют три атрибута: BORDER, FRAMEBORDER и BORDERCOLOR. Первый из этих атрибутов используется только с тегом <FRAMESET> и устанавливает ширину всех рамок для всех кадров контейнера FRAMESET. Эта величина указывается в пикселах, например,

<FRAMESET BORDER="10">

Если этот атрибут нулевой, то все кадры контейнера будут без рамок. По умолчанию атрибут BORDER имеет значение 5.

Атрибут FRAMEBORDER используется с тегами <FRAMESET> и <FRAME> и может принимать два значения: yes или no. В случае yes рамка имеет трехмерную форму. Если FRAMEBORDER="no", рамка невидима, т. е. она имеет цвет фона окна браузера, устанавливаемого по умолчанию.

По умолчанию атрибут FRAMEBORDER имеет значение yes, т. е. предусматривает наличие трехмерной рамки. Рамка кадра будет невидимой, если значение FRAMEBORDER="no" установлено для всех кадров, смежных с ним.

Атрибут BORDERCOLOR может использоваться с тегами <FRAMESET> и <FRAME>. Ему может быть присвоено стандартное имя цвета или RGB-значение.

**Пример**

FRAMESET BORDERCOLOR="red" ROWS="\*,\*">

<FRAME SRC="first.html" BORDERCOLOR="#FF00FF">

<FRAME SRC="first.html">

</FRAMESET>

Здесь атрибут BORDERCOLOR тега <FRAMESET> ; устанавливает красный цвет рамок ("red"), однако такой же атрибут тега <FRAME>отменяет это значение и определяет цвет рамки первого кадра как фиолетовый. В результате второй кадр, в котором цветовой атрибут не определен, будет иметь часть рамки фиолетовой (на стороне, смежной с первым кадром), а остальную часть рамки - красной.

Если же в двух смежных кадрах определены свои собственные атрибуты BORDERCOLOR, то ни один из них не будет иметь силы. Цвет их рамок будет определяться соответствующим атрибутом контейнера FRAMESET.

#### Организация ссылок

Теперь, когда мы разобрались с методами создания кадров, познакомимся с их главным предназначением - управлением навигацией по сайту.

Для определения имени кадра служит атрибут NAME. Например, строка <FRAME NAME="frame1"> создает кадр с именем **"frame1"**, на который можно сделать гипертекстовую ссылку следующим образом:

<A HREF="putfirst.html" TARGET="frame1">Нажмите сюда,</A>

чтобы перейти на первую страницу

Атрибут TARGET гипертекстовой ссылки содержит имя кадра. При активизации этой ссылки содержимое кадра **frame1**, т. е. файл **first.html**, размещенный в нем при создании, будет заменен файлом **putfirst.html**.

Заметьте, что если атрибут TARGET отсутствует, файл **putfirst.html** будет выведен в том же окне или кадре, где находится указатель ссылки. Атрибут TARGET как раз и предназначен для указания "цели" - кадра, в котором должен быть размещен файл, определенный атрибутом HREF. Этот принцип замены файлов в одном кадре при управлении этим процессом из другого кадра и лежит в основе навигации по сайту.

|  |  |
| --- | --- |
| **Имя** | **Назначение** |
| **blank** | Загружает указанный файл в новое окно без названия |
| **self** | Загружает указанный файл в кадр, откуда делается вызов |
| **parent** | Загружает указанный файл в старший (родительский) кадр сетки кадров; если такой кадр не определен, результат аналогичен действию **\_self** |
| **top** | Загружает указанный файл в полное окно, разрушая всю структуру кадров |

Кадру обязательно нужно присвоить имя, иначе на него нельзя будет ссылаться. Поэтому всем кадрам, содержание которых планируется менять, должны быть даны правильные имена. Имена кадров должны начинаться с алфавитно-цифрового символа. Ваши имена не должны начинаться с символа подчеркивания, так как он является первым символом зарезервированных имен кадров, перечисленных в таблице.

**ТАБЛИЦЫ СТИЛЕЙ**

В то время как *HTML* структурирует документ и сообщает веб-браузеру, какую функцию имеет определенный элемент, *CSS* выдает браузеру инструкции о том, как отобразить определенный элемент - оформление, *размещение* пробелов и *позиционирование*.

*CSS* - одна из широкого спектра технологий, одобренных консорциумом *W3C* и получивших общее название "стандарты *Web*".

|  |  |
| --- | --- |
| начало 1990-х годов | Различные браузеры имели свои стили для отображения веб страниц. HTML развивался очень быстро и был способен удовлетворить все существовавшие на тот момент потребности по оформлению информации, поэтому CSS не получил тогда широкого признания |
| 1994 | Появился термин "каскадные таблицы стилей" |
| 1996 | Консорциумом W3C была издана рекомендация CSS1 |
| 1998 | Консорциумом W3C принята рекомендация CSS2 |
| Сентябрь 2009 | Консорциумом W3C утверждена рабочая версия CSS2.1 |

### Уровень 1 (CSS1)

Это рекомендация *W3C* была принята в 1996 году и откорректирована в 1999 году. Основные возможности, предоставляемых этой рекомендацией:

* Параметры шрифтов. Возможности по заданию гарнитуры и размера шрифта, а также его стиля - обычного, курсивного или полужирного.
* Цвета. Спецификация позволяет определять цвета текста, фона, рамок и других элементов страницы.
* Атрибуты текста. Возможность задавать межсимвольный интервал, расстояние между словами и высоту строки (то есть межстрочные отступы)
* Выравнивание для текста, изображений, таблиц и других элементов.
* Свойства блоков, такие как высота, ширина, внутренние (padding) и внешние (margin) отступы и рамки. Так же в спецификацию входили ограниченные средства по позиционированию элементов, такие как float и clear.

### Уровень 2 (CSS2)

Это рекомендация *W3C* была принята в 1998 году. Она построена на *CSS1* с сохранением обратной совместимости. Среди новых возможностей можно назвать следующие:

* Блочная вёрстка. Появились относительное, абсолютное и фиксированное позиционирование. Позволяет управлять размещением элементов по странице без табличной вёрстки.
* Типы носителей. Позволяет устанавливать разные стили для разных носителей (например монитор, принтер, КПК).
* Звуковые таблицы стилей. Определяет голос, громкость и т. д. для звуковых носителей (например для слепых посетителей сайта).
* Страничные носители. Позволяет, например, установить разные стили для элементов на чётных и нечётных страницах при печати.
* Расширенный механизм селекторов.
* Указатели.
* Генерируемое содержание. Позволяет установить текст или картинку, который будет отображаться до или после нужного элемента.

### Уровень 2.1 ( CSS 2.1 )

Рабочая версия *W3C* от 8 сентября 2009 года. Построена на *CSS2*, в ней исправлен ряд ошибок.

### CSS-верстка

До появления *CSS* оформление веб-страниц осуществлялось непосредственно внутри содержимого документа. Однако с появлением *CSS* стало возможным принципиальное *разделение содержания* *и* *представления документа*. За счёт этого нововведения стало возможным лёгкое применение единого стиля оформления для массы схожих документов, а также быстрое изменение этого оформления.

Использование *CSS* дает следующие преимущества:

* Несколько дизайнов страницы для разных устройств просмотра. Например, на экране дизайн будет рассчитан на большую ширину, во время печати меню не будет выводиться, а на КПК и сотовом телефоне меню будет следовать за содержимым.
* Уменьшение времени загрузки страниц сайта за счет переноса правил представления данных в отдельный CSS-файл. В этом случае браузер загружает только структуру документа и данные, хранимые на странице, а представление этих данных загружается браузером только один раз и кешируется.
* Простота последующего изменения дизайна. Не нужно править каждую страницу, а лишь изменить CSS-файл.
* Дополнительные возможности оформления. Например, с помощью CSS-вёрстки можно сделать блок текста, который остальной текст будет обтекать (например для меню) или сделать так, чтобы меню было всегда видно при прокрутке страницы.

Известны также и недостатки:

* Различное отображение вёрстки в различных браузерах (особенно устаревших), которые по разному интерпретируют одни и те же данные CSS.
* Часто встречающаяся необходимость на практике исправлять не только один CSS-файл, но и теги HTML и код PHP, которые сложным и ненаглядным способом связаны с селекторами CSS, что иногда сводит на нет простоту применения единых файлов стилей и значительно удлиняет время редактирования и тестирования.

*Отображение* элементов реализуется с помощью системы правил, которые определяют, какие элементы *HTML* должны быть дополнительно оформлены, и в каждом правиле перечислятся свойства (например, цвет, размер, *шрифт*, и т.д.) этих элементов *HTML*, которыми они будут манипулировать, какие значения будут для них заданы.

Таким образом, *CSS* не является ни языком программирования, ни языком разметки.

### Определение правил стилей

Базовым конструкцией в *CSS* является правило следующего вида:

селектор {

свойство1: значение;

свойство2: значение;

свойство3: значение;

}

*Селектор* определяет элементы *HTML*, к которым будет применяться правило, определяемые реальным названием элемента, например, body, или другими средствами, такими как значения атрибута class.

*Фигурные скобки* {} содержат пары свойство/*значение*, которые разделяются между собой точкой с запятой; свойства отделяются от своих соответствующих значений двоеточием.

*Свойства* определяют, что вы хотите сделать с выделенными элементами. Они могут задавать, например, цвет элемента, *цвет фона*, позицию, поля, заполнение, тип шрифта, и многое другое.

*Значения* являются теми конкретными характеристиками, которые вы хотите задать каждому *свойству* выделенных элементов. Эти *значения* зависят от *свойства*.

*Свойства*, которые влияют на положение, поля, ширину, высоту и т.д. могут измеряться в *пикселях*, em, *процентах*, *сантиметрах* или других аналогичных единицах измерения.

Рассмотрим конкретный пример:

p {

margin: 10px;

font-family: Times New Roman;

color: green;

}

Это правило выбирает *HTML* элемент p, и для каждого элемента p в документах *HTML*, которые используют этот код *CSS*, будет применяться это правило, если только не будут существовать применяемые к ним более конкретные правила, которые будут переопределять это правило. Данное правило влияет на свойства, которые определяют поля вокруг параграфа (margin), *шрифт* текста в параграфе (font-family), и цвет этого текста (color). Поля задаются размером в 10 пикселей, *шрифт* задается как Times New Roman, а *цвет текста* задается как green.

Все множество подобных правил совместно формируют таблицу стилей. Кроме таких правил в *CSS* существуют и другие конструкции, например *комментарии CSS*, *селекторы объединения в группу*.

### Комментарии CSS.

Комментарии добавляют, помещая их между символами /\* и \*/. Комментарии могут охватывать несколько строк, и в этом случае *браузер* будет игнорировать эти строки:

### Селекторы объединения в группу

Можно также объеднить в группу различные селекторы. Предположим, что вы хотите применить одинаковое оформление к h2 и p, тогда можно было бы написать следующий *CSS*:

h2 {color: red}

p {color: red}

Однако, можно сократить код *CSS*, группируя селекторы вместе с помощью запятой - правила в скобках применяются к обоим селекторам:

h2, p {color: red}

Существует несколько различных *селекторов*, каждый из которых соответствуют различным частям разметки. Тремя наиболее общими селекторами, которые встречаются чаще всего, являются следующие:

|  |  |  |
| --- | --- | --- |
| **Синтаксис** | **Селектор** | **Описание** |
| p {} | селектор элемента | соответствует всем элементам на странице с указанным названием (элементам p, в приведенном выше случае) |
| example{} | селектор класса | соответствует всем элементам, которые имеют атрибут class с указанным значением, так что пример выше будет соответствовать <p class="example">, <li class="example"> или <div class="example">, или любому другому элементу со значением class = "example". |
| #example{} | селектор id | соответствует всем элементам, которые имеют атрибут id с указанным значением, так что пример выше будет соответствовать <p id="example">, <li id="example"> или <div id="example">, или любому другому элементу со значением id = "example". Селекторы id не проверяют название никакого элемента, и можно иметь только один селектор для каждого id в документе HTML - они являются уникальными для каждой страницы. |

### Дополнительные селекторы CSS

С помощью селекторов *CSS*, селекторов элемента, класса и id-селекторов можно реализовать многое, но это, конечно, не все возможные селекторы - существуют другие селекторы, которые позволяют выбирать элементы для стилевого оформления на основе более специфических критериев:

|  |  |
| --- | --- |
| **Селекторы** | **Описание** |
| *Универсальные селекторы* | можно использовать для выбора каждого элемента на странице |
| *Селекторы атрибутов* | позволяют выбирать элементы на основе их атрибутов |
| *Селекторы потомков* | если вы хотите выбрать определенные элементы, которые являются потомками других конкретных элементов |
| Селекторы нижележащих | если вы хотите выбирать определенные элементы, которые являются нижележащими относительно других конкретных элементов (не просто прямыми потомками, но также расположенные ниже в дереве) |
| Селекторы смежных одноуровневых | если вы хотите выбрать только определенные элементы, которые следуют за другими определенными элементами |
| Псевдо-классы | эти селекторы позволяют оформить элементы не на основе того, чем является элемент, но на основе более сложных факторов, скажем, таких как состояние ссылки (если на нее, например, наведен курсор, или она уже была посещена). |
| Псевдо-элементы | эти селекторы позволяют оформить определенные части элементов, а не весь элемент (например, первую букву в этом элементе), они позволяют также вставлять содержимое перед или после определенных элементов |

### Универсальные селекторы

*Универсальные селекторы* выбирают каждый элемент на странице для применения к ним стилей оформления. Например, следующее правило определяет, что для каждого элемента на странице должна быть добавлена сплошная черная граница толщиной 1 пиксель:

{

border: 1px solid #000000;

}

### Селекторы атрибутов элементов

*Селекторы атрибутов* позволяют выбирать элементы на основе содержащихся в них атрибутов. Например, можно выбрать каждый элемент img с атрибутом alt с помощью следующего селектора:

img[alt] {

border: 1px solid #000000;

}

Используя приведенный выше селектор, можно создать черную границу вокруг любого изображения, которое имеет *атрибут* alt, и оформить другие изображения ярко-красной.

Можно выбирать элементы также и по значению атрибута, а не только по названию атрибута. Следующее правило задает все изображения с атрибутом src со значением alert.gif:

img[src="alert.gif"] {

border: 1px solid #000000;

}

Значительно более полезной является возможность выбирать определенные части атрибутов, например, расширения файлов. CSS3 фактически вводит три новых типа *селекторов атрибутов*, которые могут выбирать на основе текстовой строки в значениях атрибутов (в начале, конце, или в любом месте в значении).

### Селекторы потомков элементов

Можно использовать *селектор потомка* для выбора только определенных элементов, которые являются потомками других определенных элементов. Например, следующее правило задает *цвет текста* только тех strong элементов, которые являются потомками элементов h3, как green, но не для других элементов strong:

h3 > strong {

color: green;

}

Селекторы потомоков не поддерживаются в браузере IE 6 (и более младших версиях).

### Селекторы нижележащих элементов

Селекторы нижележащих элементов очень похожи на *селекторы потомков*, за исключением того, что *селекторы потомков* выбирают только непосредственно нижележащих, а селекторы нижележащих выбирают подходящие элементы в любом месте иерархии элементов, а не только непосредственно нижележащих. Рассмотрим пример.

<div>

<em>Привет</em>

<p>и сразу же

<em>Пока</em>.

</p>

</div>

В этом фрагменте элемент div является предком всех других элементов. Он имеет двух потомоков, em и p. Элемент p имеет один элемент *потомок*, еще один em.

### Селекторы смежных одноуровневых элементов

Эти селекторы позволяют выбирать определенный элемент, который следует непосредственно после другого определенного элемента на том же уровне в иерархии элементов. Например, если вы хотите выбрать элементы p, которые следуют непосредственно после элементов h2, но никакие другие элементы p, можно воспользоваться следующим правилом:

h2 + p {

...

}

Селекторы смежных одноуровневых элементов не поддерживаются в браузере IE 6 (и более младших версиях).

### Псевдо-классы

Псевдо-классы используются для обеспечения стилевого оформления не для элементов, а для различных состояний элементов. Наиболее обычным применением, которое можно встретить, является оформление состояний ссылок.

*Список* ниже представляет различные псевдо-классы и описание состояния ссылки, которое они выбирают:

|  |  |
| --- | --- |
| :link | обычное состояние ссылок по умолчанию, когда вы впервые их находите |
| :visited | ссылки, которые вы уже посетили в используемом в данный момент браузере |
| :focus | ссылка (или поле формы, или что-то еще), в которой в данный момент находится курсор клавиатуры |
| :hover | ссылка, на которой в данный момент находится указатель мыши |
| :active | ссылка, на которой в данный момент происходит щелчок. |

Следующие правила *CSS* определяют что:

* по умолчанию ссылки будут синими.
* когда курсор мыши оказывается над ссылкой, используемое по умолчанию подчеркивание ссылки исчезает.
* когда ссылка будет посещена, она станет серой.
* когда ссылка активна, она становится жирной, как дополнительный признак, что что-то сейчас произойдет.

a:link{

color: blue;

}

a:hover{

text-decoration: none;

}

a:visited{

color: gray;

}

a:active{

font-weight: bold;

}

Обратите внимание, если вы не определите эти правила в том же порядке, как они показаны выше, они могут работать не так, как вы ожидаете. Это обусловлено правилом специфичности, заставляющем более поздние правила в таблице стилей переопределять более ранние правила.

Можно выделить *поле ввода*, которое содержит *активный* мигающий *курсор* с помощью следующего правила:

input:focus {

border: 2px solid black;

background-color: lightgray;

}

### Псевдо-элементы

Псевдо-элементы имеют два назначения. Прежде всего, можно использовать их для выбора первой буквы или первой строки текста в заданном элементе. Чтобы создать буквицу в начале каждого параграфа документа, можно использовать следующее правило:

p:first-letter {

font-weight: bold;

font-size: 250%

background-color: red;

}

Чтобы сделать первую строку каждого параграфа жирной, можно использовать следующее правило:

p:first-line {

font-weight: bold;

}

Вторым применением псевдо-элементов является генерация контента с помощью *CSS*, что является более сложной задачей. Можно использовать псевдо-элементы :before или :after для определения того, что содержимое должно быть вставлено перед или после элемента, который вы выбираете. Затем определяют то, что должно быть вставлено. В качестве простого примера можно использовать следующее правило для вставки декоративных изображений после каждой ссылки на страницу:

a:after{

content: " " url(flower.gif);

}

Можно также использовать функцию attr() для вставки значений атрибутов элементов после элемента. Например, можно вставить цель каждой ссылки в документ в скобках с помощью следующего кода:

a:after{

content: "(" attr(href) ")";

}

Такие правила прекрасно подходят для таблиц *стилей печати*, которые являются таблицами стилей, которые автоматически применяются, когда *пользователь* печатает страницу. Преимущество для пользователя состоит в том, что можно скрыть всю навигацию, которую невозможно использовать в печатном материале, и использовать описанную выше технику, чтобы читатель мог видеть адреса *URL*, на которые ссылается страница.

Эти селекторы не поддерживаются в IE 6 (и более младших версиях).

### Сокращенная запись CSS

Можно объединить несколько связанных свойств *CSS* в одно свойство, чтобы сэкономить время и свои усилия.

### Сравнение индивидуальных и сокращенных значений

Рассмотрим следующее правило для полей (сокращения для заполнения и границы работают таким же образом):

div.foo {

margin-top: 1em;

margin-right: 1.5em;

margin-bottom: 2em;

margin-left: 2.5em;

}

Это правило можно записать короче:

div.foo {

margin: 1em 1.5em 2em 2.5em;

}

### Задание менее четырех значений для сокращенного свойства

Сокращенное *значение* может иметь менее четырех значений, согласно приведенному ниже списку. Результаты упорядочены по числу предоставленных значений:

|  |  |
| --- | --- |
| ***Одно* значение применяется ко всем четырем сторонам** | **margin: 2px;** |
| *Первое* значение применяется к верху и низу, *второе* к левому и правому краю | margin: 2px 5px;. |
| *Первое* и *третье* значения применяются к верху и низу соответственно, *второе* значение применяется к левому и правому краю | margin: 2px 5px 1px; |
| Значения применяются к верху, правому краю, низу, и левому краю в соответствии с порядком исходного кода CSS |  |

### Справочник сокращений

|  |  |
| --- | --- |
| Граничные сокращения для различных свойств | Необходимо упомянуть еще, что можно даже задать значения свойств границ (border) только для одной границы элемента следующим образом:  border-left-width: 2px;  border-left-style: solid;  border-left-color: black; |
| Сокращения для некоторых свойств полей (margin), заполнения (padding) и границы (border) | Все это действует таким же образом как было показано выше в разделе "Выбор между одиночным свойством и сокращенным значением". |
| Сокращения для шрифта | С помощью одной строки сокращения можно определить размер шрифта, толщину, стиль, семейство и высоту строки. Например, рассмотрим следующий код:  font-size: 1.5em;  line-height: 200%;  font-weight: bold;  font-style: italic;  font-family: Georgia, "Times New Roman", serif  Можно определить все это с помощью следующей строки:  font: 1.5em/200% bold italic Georgia,"Times New Roman",serif; |
| Сокращение для фона | С помощью одной строки CSS можно определить цвет фона, фоновое изображение, повторение изображения и позицию изображения. Возьмем следующий код:  background-color: #000;  background-image: url(image.gif);  background-repeat: no-repeat;  background-position: top left;  Это можно представить с помощью следующего сокращения:  background:#000 url(image.gif) no-repeat top left; |
| Сокращения для списков | В данном случае аналогичная история со *свойствами списков* позволяет задать на одной строке значения свойств для типа маркера списка, позиции и изображения. Возьмем следующий код:  list-style-type: circle;  list-style-position: inside;  list-style-image: url(bullet.gif) ;  Это эквивалентно следующему:  list-style: circle inside url(bullet.gif); |
|  |  |

### Применение CSS к HTML

Существует три способа применения *CSS* к документу *HTML*:

* строковые;
* вложенные;
* внешние таблицы стилей.

### Строковые стили

Можно применить таблицу стилей к элементу, используя *атрибут* style следующим образом:

<p style="background:blue; color:white;

padding:5px;">Hello</p>

Внутри этого атрибута перечисляются все свойства *CSS* и их значения.

Преимущество строковых стилей состоит в том, что *браузер* будет вынужден использовать эти настройки. Все другие стили, определенные в других таблицах стилей, или даже вложенные в *заголовок документа*, будут переопределены этими стилями.

Серьезна проблема у строковых стилей состоит в том, что они делают обслуживание стилей значительно труднее. Использование *CSS* предназначено для разделения представления документа и его структуры, но строковые стили делают именно обратное - рассеивают правила представления по документу.

Кроме проблем с обслуживанием, вы не получите никаких преимуществ самой в существенной части *CSS*: каскадировании.

### Вложенные стили

Вложенные таблицы стилей помещаются в заголовке документа внутри элемента style, как в следующем примере страницы:

<style type="text/css" media="screen">

p {

color:white;

background:blue;

padding:5px;

}

</style>

Преимущество вложенных таблиц стилей состоит в том, что не нужно добавлять *атрибут* style в каждый *параграф* - можно оформить их все с помощью одного единственного определения. Это означает также, что если потребуется изменить внешний вид всех параграфов, это можно будет сделать в одном единственном месте, однако все это, тем не менее, ограничено одним документом.

### Внешние таблицы стилей

Внешние таблицы стилей означают *размещение* всех определений *CSS* в отдельном файле, сохраняя его с расширением файла *CSS*, и затем применение его к документам *HTML*, используя элемент *link* в заголовке документа. Например:

<link rel="stylesheet" href="styles.css"

type="text/css" media="screen">

Внешние таблицы стилей позволяют хранить все определения оформления в одном единственном файле. Это означает, что можно вносить изменения на всем сайте, изменяя только один *файл*. Веб-*браузер* при этом может загрузить его один раз и затем кешировать для всех других документов, которые на него ссылаются, что уменьшает объем загружаемой информации.

### Импорт таблиц стилей

Существует и другой способ импорта внешних таблиц стилей в файлы *HTML* - оператор @import. Он вставляется во вложенную таблицу стилей, таким же образом, как показанный выше вложенный код *CSS*. *Синтаксис* выглядит следующим образом:

<style type="text/css" media="screen">

@import url("styles.css");

другие операторы импорта или стили CSS.

</style>

Двумя фундаментальными концепциями *CSS* являются *наследование* и *каскадирование*. **Наследование** связано с тем, как элемент в разметке *HTML* наследует свойства своих элементов предков (в которых он содержится) и передает их своим потомкам, в то время как **каскадирование** имеет дело с объявлениями *CSS*, которые применяются к документу, и как конфликтующие правила переопределяют друг друга.

### Наследование

**Наследование** в *CSS* является механизмом, с помощью которого определенные свойства передаются от элемента предка его элементам потомкам.

Используя *наследование* можно, например, определить свойства шрифта для элементов html или body, и они будут унаследованы всеми другими элементами. Можно определить цвета фона и переднего плана для определенного контейнерного элемента, и цвет переднего плана будет автоматически унаследован элементами потомками в этом контейнере.

Каждый элемент в документе *HTML* будет наследовать все наследуемые свойства своего предка, за исключением корневого элемента (html), который не имеет предка.

### Каскадирование

**Каскадирование** - это механизм, который управляет конечным результатом, в случае когда несколько конфликтующих объявлений *CSS* применяются к одному элементу. Имеется три основные концепции, которые управляют порядком, в котором применяются объявления *CSS*:

* Важность
* Специфичность
* Порядок исходного кода

*Важность* является наиболее значимой. Если два объявления имеют одинаковую важность, специфичность правил определяет, какое из них будет применяться. Если правила имеют одинаковую специфичность, то порядок исходного кода управляет результатом.

### Важность

**Важность** объявления *CSS* зависит от того, где оно определено. Конфликтующие объявления будут применяться в следующем порядке, более поздние будут переопределять предыдущие:

* Таблицы стилей агента пользователя
* Обычные объявления в таблицах стиля пользователя
* Обычные объявления в таблицах стиля автора
* Важные объявления в таблицах стиля автора
* Важные объявления в таблицах стиля пользователя

*Таблица стилей агента пользователя* является встроенной таблицей стилей браузера. Каждый *браузер* имеет свои используемые по умолчанию правила, определяющие, как выводить различные элементы *HTML*, если никакой стиль не определен пользователем или дизайнером страницы. Например, непосещенные ссылки будут обычно выводиться синим цветом и подчеркнутыми.

*Таблица стилей пользователя* является таблицей стилей, которую определил *пользователь*. Не все браузеры поддерживают таблицы стилей пользователя, но они могут быть очень полезны, особенно для пользователей с некоторыми типами функциональных недостатков. Например, человек с дислексией может иметь таблицу стилей, которая определяет определенные шрифты и цвета, которые облегчают чтение.

*Таблица стилей автора* является тем, что обычно и называется "*таблица стилей*". Это *таблица стилей*, которую *автор* документа (или, более вероятно, дизайнер сайта) написал и присоединил (или включил).

Для того чтобы превратить обычное объявление в важное за ним необходимо разместить директиву !important. Как можно видеть, важные объявления в таблице стилей пользователя будут перекрывать все остальное, что вполне логично.

Используемое по умолчанию в браузере *представление* будет применяться только в том случае, если эти объявления не переопределены какими-либо правилами таблицы стилей пользователя или таблицы стилей автора, так как *таблица стилей* агента пользователя имеет наименьший приоритет.

### Специфичность

**Специфичность** опеределяют как меру того, насколько конкретным является селектор некоторого правила. Селектор с низкой специфичностью может соответствовать многим элементам (такой как \*, который соответствует каждому элементу в документе), в то время как селектор с высокой специфичностью может соответствовать только одному элементу на страницу (такой как #nav, который соответствует только элементу с id совпадающим с nav).

Специфичность селектора можно легко вычислить. Если два или больше объявлений конфликтуют за данный элемент, и все объявления имеют одинаковую важность, то приоритет в правиле будет иметь объявление с наиболее специфичным селектором.

Специфичность имеет четыре компоненты, которые можно обозначить как a, b, c и d. Компонента a является наиболее разграничивающим, d - наименее.

* Компонента a определяется очень просто: это 1 для объявления атрибута style, иначе это 0.
* Компонента bявляется числом селекторов idв селекторе (тех, которые начинаются *с*#).
* Компонента cявляется числом *селекторов атрибутов*, включая селекторы классов - и псевдо-классов.
* Компонента dявляется числом типов элементов и псевдо-элементов в селекторе.

После небольшого подсчета можно получить строку из этих четырех компонентов, определяющую специфичность для любого правила. Объявления *CSS* в атрибуте style не имеют селектора, поэтому их специфичность всегда будет 1,0,0,0.

В таблице приведены некоторые примеры.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Селектор** | **a** | **b** | **c** | **d** | **Специфичность** |
| h1 | 0 | 0 | 0 | 1 | 0,0,0,1 |
| .foo | 0 | 0 | 1 | 0 | 0,0,1,0 |
| #bar | 0 | 1 | 0 | 0 | 0,1,0,0 |
| html>head+body ul#nav \*.home a:link | 0 | 1 | 2 | 5 | 0,1,2,5 |

Стоит отметить, что соединяющие символы (такие как >, + и *пробел*) не влияют на специфичность селектора. *Универсальный селектор* (\*) также не влияет на специфичность.

Имеется значительная разница в специфичности между селектором id и селектором *атрибута*, который ссылается на *атрибут* id. Хотя они соответствуют одному элементу, они имеют очень разную специфичность. Специфичность #nav будет 0,1,0,0, в то время как специфичность [id="nav"] будет только 0,0,1,0.

### Порядок исходного кода

Если два объявления влияют на один и тот же элемент, имеют одинаковую важность и одинаковую специфичность, то окончательное решение определяет порядок исходного кода. Объявление, которое появляется позже в таблицах стилей будет выигрывать у тех, которые встречаются раньше.

Если имеется единственная внешняя *таблица стилей*, то объявления в конце файла будут переопределять объявления, которые встречаются раньше в файле, если возникает *конфликт*. Конфликтующие объявления могут также возникать в различных таблицах стилей. В этом случае порядок, в котором присоединяются таблицы стилей, включаются или импортируются, управляют тем, какое объявление будет применяться.

Итак, *наследование* позволяет объявлять свойства на высокоуровневых элементах и разрешает этим свойствам распространяться вниз на все элементы потомки. Только некоторые свойства наследуются по умолчанию, но *наследование* можно реализовать принудительно с помощью ключевого слова inherit.

Каскадирование разрешает все конфликты, когда несколько объявлений влияют на данный элемент. Важные объявления будут переопределять менее важные. Среди объявлений с равной важностью специфичность правила управляет тем, какое из них будет применяться. И если все остальное будет совпадать, то порядок исходного кода определяет окончательный выбор.

### Спецификация CSS2

Эта спецификация определяет *Каскадные таблицы Стилей*, уровень 2 (*CSS2*). *CSS2* - это *язык таблиц стилей*, позволяющий авторам и пользователям подключать стили (например, шрифты, пробелы и звуковые сигналы) в структурированные документы (например, документы *HTML* и приложения *XML*). *CSS2* упрощает создание и обслуживание *Web*-сайта путём разделения структуры и стиля представления документов.

*CSS2* построен на основе *CSS1*, и, с очень небольшими исключениями, все таблицы стилей *CSS1* остаются действующими в *CSS2*. *CSS2* поддерживает таблицы для конкретных носителей, так что авторы могут создавать *представление* своих документов для визуальных браузеров, звуковых устройств, принтеров, брайль-устройств, ручных портативных устройств и т.д. Данная спецификация поддерживает также *позиционирование* содержимого, загружаемые шрифты, *отображение* таблиц, возможности интернационализации, автоматические *счётчики* и нумерацию и некоторые свойства, относящиеся к пользовательскому интерфейсу.