Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Институт повышения квалификации
и переподготовки кадров

Кафедра «Информатика»

# ВЕРСТКА WEB-СТРАНИЦ

## КУРС ЛЕКЦИЙ
## по одноименной дисциплине
## для слушателей специальности 1-40 01 74
## «Web-дизайн и компьютерная графика»
## заочной формы обучения

Гомель 2013

УДК 004.738.52(075.8)
ББК 32.973.202я73
Т46

*Рекомендовано к изданию научно-методическим советом*
*факультета автоматизированных и информационных систем*
*ГГТУ им. П. О. Сухого*
(*протокол № 7 от 25.02.2013 г.*)

Рецензент: проф. д-р техн. наук, проф. каф. «Информационные технологии»
ГГТУ им. П. О. Сухого *И. А. Мурашко*

**Тихоненко, Т. В.**

Т46 Верстка WEB-страниц : курс лекций по одноим. дисциплине для слушателей специальности 1-40 01 74 «Web-дизайн и компьютерная графика» заоч. формы обучения / Т. В. Тихоненко. – Гомель : ГГТУ им. П. О. Сухого, 2013. – 172 с. Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://library.gstu.by. – Загл. с титул. экрана.

Данное издание представляет собой курс лекций по теоретическому и практическому освоению технологий создания Web-страниц – языка разметки гипертекста HTML и каскадных таблиц стилей CSS. Уделено внимание правилам использования графики и звука при создании web-страниц, изложены принципы формирования карт-изображений, фреймов и пользовательских форм. Приведены примеры верстки сайтов с использованием стилей. В последней главе рассказывается об основных этапах размещения, поддержки и раскрутки сайта в Интернете.

Для слушателей специальности 1-40 01 74 «Web-дизайн и компьютерная графика» заочной формы обучения ИПК и ПК.

**УДК 004.738.52(075.8)**
**ББК 32.973.202я73**

**Содержание**

1 ВВЕДЕНИЕ И ОСНОВНЫЕ ПОНЯТИЯ ..... 5
1.1 Понятие Всемирной паутины. Основные сведения о языках разметки: HTML, XML, XHTML ..... 5
1.2 Понятия web-сервера, web-сайта, web-страницы, и их отличия ..... 8
1.3 Что такое тег? Типы тегов ..... 9
1.4 Структура и привила оформления HTML-документа ..... 10
1.5 Основные элементы форматирования текста. Элементы блочные и строчные ..... 12
1.6 Понятие элементов и атрибутов ..... 17
1.7 Специальные символы языка HTML ..... 21
1.8 Вопросы для самоконтроля ..... 23
2 ССЫЛКИ И ИЛЛЮСТРАЦИИ ..... 25
2.1 Механизмы адресации на ресурсы в Internet. Понятие ссылки ..... 25
2.2 Элемент <a>…</a> и его атрибуты ..... 26
2.3 Размещение иллюстраций на web-странице. Типы файлов иллюстраций ..... 27
2.4 Элемент IMG и его атрибуты ..... 31
2.5 Навигационные карты ..... 34
2.6 Вопросы для самоконтроля ..... 40
3 СПИСКИ И ТАБЛИЦЫ ..... 41
3.1 Структурирование информации на web-странице при помощи списков. Типы списков ..... 41
3.2 Таблица и ее элементы ..... 44
3.3 Правила задания атрибутов для таблицы и ее ячеек. Объединение ячеек ..... 45
3.4 Верстка при помощи таблиц. Вложенные таблицы ..... 48
3.5 Приемы использования таблиц на web-странице ..... 50
3.6 Преимущества и недостатки табличной верстки ..... 53
3.7 Вопросы для самоконтроля ..... 55
4 ФРЕЙМЫ И ФОРМ ..... 56
4.1 Фреймы и их описание на языке HTML. Задание логики взаимодействия фреймов ..... 56
4.2 Типичные достоинства и недостатки фреймовой структуры ..... 64
4.3 Форма и ее элементы. Методы отправки информации из полей формы ..... 65
4.4 Использование табличных функций при создании форм на примере анкеты ..... 75
4.5 Вопросы для самоконтроля ..... 77
5 КАСКАДНЫЕ ТАБЛИЦЫ СТИЛЕЙ (CSS) ..... 78

5.1 Основные цели и задачи CSS. Способы добавления стилей на web-страницу ..... 78
5.2 Основные понятия и определения. Грамматика языка стилей..... 82
5.3 Создание стилей и классов. Применение стилей и классов на web-странице..... 84
5.4 Единицы измерения размеров ..... 88
5.5 Применение селекторов к элементам на web-странице ..... 93
5.6 Декоративные возможности CSS: форматирование текста, формирование рамок и отступов ..... 118
5.7 Наследование, каскадирование и специфичность ..... 125
5.8 Позиционирование элементов на странице и управление моделью элемента..... 131
5.9 Вопросы для самоконтроля..... 135
6 ИСПОЛЬЗОВАНИЕ СТИЛЕЙ ПРИ СОЗДАНИИ САЙТА..... 136
6.1 Верстка на CSS. Создание блока ..... 136
6.2 Создание текста в три колонки..... 137
6.3 Современная верстка сайта при помощи CSS..... 139
6.4 Приемы макетирования web-страницы с использованием стилей .. 143
7 ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ HTML И CSS. ..... 150
7.1 Размещение мультимедийной информации на web-странице ..... 150
7.2 Нестандартные теги HTML..... 153
7.3 Зачем нужен DOCTYPE?..... 153
7.4 Вопросы для самоконтроля..... 155
8 РАЗМЕЩЕНИЕ САЙТА НА СЕРВЕРЕ И ЕГО ПОДДЕРЖКА ..... 156
8.1 Описание метаинформации сайта с помощью элемента META..... 156
8.2 Размещение сайта в интернете. Вопросы хостинга..... 159
8.3 Работа с FTP-клиентом..... 161
8.4 Поддержка сайта ..... 165
8.5 Способы раскрутки сайта..... 166
8.6 Вопросы для самоконтроля..... 171
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ..... 172

# 1 Введение и основные понятия

## 1.1 Понятие Всемирной паутины. Основные сведения о языках разметки: HTML, XML, XHTML

Основные концепции Всемирной паутины появилась в стенах Европейского совета по ядерным исследованиям в 1989 году. Над созданием этой концепции работал знаменитый британский учёный Тим Бернерс-Ли, который также работал над созданием протокола HTTP, языка HTML и идентификаторов URI.

Под Всемирной паутиной (World Wide Web – WWW) понимают распределенную систему, предоставляющую доступ к связанным между собой документам, расположенным на различных компьютерах, подключенных к Интернету. Эти электронные документы описываются с помощью специальных технологических правил, которые составляются на языке гипертекстовой разметки **HTML** (HyperText Markup Language, язык гипертикстовой разметки). Данный язык является основой всех размещенных в Интернете документов и выступает в роли некоего фундамента, на базе которого реализуются прочие сетевые программные технологии, призванные в конечном итоге повысить общую привлекательность, эффективность и интерактивность носителей информационных данных в Сети. HTML был сформированы на основе уже существующего стандарта **SGML** (Standard Generalized Markup Language, стандартный язык обобщенной разметки) в 1991 году.

Начиная с момента своего возникновения, разработкой спецификации языка HTML стала заниматься специально созданная организация – консорциум **W3C** (World Wide Web Consortium). Основной задачей данной организации являлось составление и принятие технических рекомендаций единого стандарта разметки гипертекстовых документов. Необходимость работы над этим стандартом была обусловлена постоянным ростом популярности Интернета среди пользователей, в рамках которого производители браузеров выдвигали свои предложения по улучшению правил описания гипертекстовых данных. Деятельность консорциума W3C по сути призвана регулировать и контролировать развитие и совершенствование языка гипертекстовой разметки HTML, учитывая потребности сферы интернет-технологий и компаний-разработчиков, работающих на рынке браузеров. Производители браузеров постоянно предлагают технологические инновации в спецификацию языка, часть которых получает одобрение консорциума W3C. Остальная часть может, тем не менее, внедряться в программную платформу выпускаемых браузеров, что на практике

вызывает проблемы несовместимости электронных документов при их просмотре браузерами разных моделей и версий.

В 1993 году компанией Mosaic Communications был разработан браузер с графическим интерфейсом пользователя Mosaic, от которого в 1994 году стартовал браузер Netscape. Затем в августе 1995 года в состав операционной системы Windows 95 компании Microsoft впервые был включен интернет-браузер Internet Explorer 1.0. Разработчики Microsoft поставили Internet Explorer на более высокий уровень развития, что позволило браузеру-дебютанту составить достойную конкуренцию Netscape. В связи с этим основатели компании Netscape Communications Corporation выпускают новую версию своего браузера, снабдив его не только новым именем Netscape Navigator, но и некоторыми техническими возможностями, тем самым начав знаменитую «войну браузеров» [1]. В настоящее время существует огромное количество браузеров, наиболее популярными среди которых являются Internet Explorer, Mozilla Firefox, Opera, Safari, Chrome и др.

Стандарта HTML 1 никогда не было, первой официальной спецификацией стал стандарт **HTML 2.0**, опубликованный IETF (Internet Engineering Task Force, целевая группа инженерной поддержки Internet) в конце 1994 года. Многие элементы в этой спецификации были заимствованы из существующих реализаций, например, тег <img> уже поддерживался, лидировавшим в те времена браузером Mosaic. Во второй половине 90-х годов язык HTML претерпел множество изменений, последней в череде которых стала спецификация **HTML 4.01** (1999 год). Следующий за версией языка HTML 4.01, стал стандарт **XHTML 1.0** (Extensible HTML, расширенный HTML). Содержание спецификации XHTML 1.0 не сильно отличалось от HTML 4.01 (не было добавлено ни одного нового элемента или атрибута), однако отличался синтаксис языка. HTML давал разработчикам web-страниц значительную свободу в написании элементов и атрибутов, а XHTML стал требовать выполнения особых правил **XML** (Extensible Markup Language, расширяемый язык разметки) – более строгого языка разметки, на котором основаны многие технологии W3C. У строгих правил есть свои преимущества, они заставляют разработчиков использовать единый стиль кодирования. Если прежде теги и атрибуты могли быть в верхнем регистре, нижнем регистре и любой их комбинации, то в XHTML 1.0 все теги и атрибуты должны быть в нижнем регистре. Следующий версией стандарта разметки стал XHTML 1.1. Если XHTML 1.0 был простым HTML переформулированным на XML, то XHTML 1.1 стал настоящим XML, он больше не мог подаваться клиенту как text/html. Однако, если разработчик использует для документа медиа тип XML, то самый популярный на тот момент web-браузер Internet Explorer мог отобразить документ.

В консорциуме W3C решили, что историю языка гипертекстовой разметки HTML нужно закончить на четвертой версии и начали работу над **XHTML 2**. Данный стандарт был призван вести пользователей в новое будущее, основанное на XML. Несмотря на то, что названия стандартов XHTML 2 и XHTML 1 отличаются лишь в одной цифре, они совершенно не похожи. В отличии от XHTML 1, в XHTML 2 нет обратной совместимости с существующим web-контентом и предыдущими версиями HTML, это должен был быть «чистый» язык, не обремененный историей предыдущих спецификаций.

Консорциум W3C формулировал свои стандарты, игнорируя потребности web-разработчиков. Очевидно, что представителей Opera, Apple и Mozilla это не удовлетворяло, что побудило их сформировать свою собственную рабочую группу WHATWG (Web Hypertext Application Technology Working Group), главным редактором которой стал Ян Хиксон. С самого начала WHATWG начал работу над двумя расширениями HTML: web Forms 2.0 и web Apps 1.0, позже они были объединены в одну спецификацию названную **HTML5**. В то же время консорциум W3C продолжал работу над XHTML 2.

В октябре 2006 года, Тим Бернерс-Ли признал в своем блоге, что попытка перейти с HTML на XML не удалась. Через несколько месяцев была создана новая рабочая группа по HTML. Базой для будущей версии HTML стали наработки рабочей группы WHATWG.

Все это сильно запутало ситуацию. Консорциум W3C работает одновременно над двумя различными, несовместимыми типами разметки XHTML 2 и HTML 5 (обратите внимание на пробел перед цифрой пять). В то же время отдельная организация WHATWG, работает над спецификацией HTML5 (без пробела), которая используется как базис одной из спецификаций W3C.

В 2009 году W3C объявило, что работа над XHTML 2 не будет продолжена. В свою очередь разработчики, которым XHTML 1 нравился за более строгий стиль кодирования, опасались, что HTML5 приведет к возвращению небрежного кода. На самом деле это не обязательно, HTML5 может быть строгим или свободным настолько, насколько этого хочет разработчик.

Самый неясный вопрос для разработчиков желающих использовать HTML5 «Когда он будет готов?». В одном из интервью, Ян Хиксон сказал, что HTML5 может получить статус «proposed recommendation» к 2022 году, что вызвало волну недовольства среди web-разработчиков. Однако это возмущение не обосновано. Статус «proposed recommendation» предполагает наличие двух полных реализаций HTML5 и, учитывая объем спецификации, эта дата выглядит даже оптимистичной.

Но даже эта дата не особенно важна для web-разработчиков, самое большое значение имеет то, когда браузеры начнут поддерживать новые возможности. Части стандарта CSS 2.1 были использованы, как только браузеры начинали их поддерживать, если бы пришлось ждать, пока все браузеры будут полностью поддерживать CSS 2.1, то все еще прошлось бы ждать. То же самое с HTML5, не будет такой даты, чтобы можно было сказать, да сегодня язык готов к использованию. Можно начинать использовать части спецификаций, как только появиться их поддержка в браузерах. Отдельные части языка HTML5 уже работают. Следует понимать, что HTML5 – это не совершенно новый стандарт, созданный с нуля, это эволюционное, а не революционное изменение языка разметки, если сейчас вы используете HTML, то вы уже используете HTML5.

## 1.2 Понятия web-сервера, web-сайта, web-страницы, и их отличия

***Сервер*** – это компьютер с установленным на нем специальным программным обеспечением (специальная программа тоже называется сервером, web-сервером или http-сервером), которое отображает web-страницы по запросу клиентской машины, а также выполняет множество других полезных функций и имеющий собственное доменное имя, то есть адрес DNS, отвечающий стандартам Domain Name System. Когда компьютер связывается с сервером и получает от него все необходимые данные, например код web-страницы, он выступает в роли «клиента», а всю систему в этом случае принято называть связкой «клиент-сервер».

***Системой «клиент-сервер»*** называют механизм передачи информации между удаленным компьютером, предоставляющим свои ресурсы в распоряжение пользователей, и пользовательским компьютером, эксплуатирующим эти ресурсы. Сервер должен представлять собой «информационный портал», то есть он является достаточно большим виртуальным пространством, состоящим из множества различных тематических разделов меньшего размера, либо некоторого количества самостоятельных проектов.

А как это происходит? Цифра 1 на рис. 1.1 – это запрос «клиента» на сервер о необходимом документе, адрес которого набирается либо вручную, либо с помощью нажатия по гиперссылке. Цифра 2 означает, что сервер сформировал ответ в виде текста и отдал «клиенту». Дальше пользовательский браузер разбирает этот текст, анализирует его, понимает, какие есть еще файлы, которые используются на этой web-странице. Цифра 3 показывает запрос, который идет для получения этих файлов (картинки, таблицы стилей, флэш и другие отдельные фрагменты). Причем, цифра 3, как показано на рис. 1.1, уходит на тот же самый сервер, но по факту, все эти файлы могут находиться на других серверах. Получив

все части web-страницы, браузер отдает пользователю необходимый документ (цифра 5 на рис. 1.1). Вот основные этапы получения информации из Всемирной паутины.


*Рисунок 1.1 – Организация системы «клиент-сервер»*

Участок сервера, точнее, раздел, полностью посвященный какой-либо теме, называется ***сайтом***.

Текст, созданный с помощью любого текстового редактора, а затем сохраненный в формате HTML, становится ***web-страницей*** (HTML-документом) после добавления в него команд языка HTML.

## 1.3 Понятие тега. Типы тегов

Любой язык разметки состоит из двух основных частей:

1. информация, которую нужно передать (текст, который размечается);
2. данные разметки.

Данные разметки можно узнать по угловым скобкам < и >. Текст между угловыми скобками – это не что иное, как разметка HTML-документа. Такого образа конструкции, называются *теги*. Другими словами, тег – это инструкция браузеру, указывающая способ отображения текста.

Теги могут быть двух видов:

1) <..> одноэлементные (одиночные). Элементы, образованные при помощи одиночных тегов, называют *пустыми*;

2) <...>...</...> парные. Парный тег влияет на текст, с того места, где он употреблен, и до того места, где указан признак окончания его действия. Признаком завершения служит тот же тег, только начинающийся с символа « / » (обратный слеш).

Есть еще специальный вид тега, который в HTML называется *декларация*. Он используется исключительно для служебной информации.

Одним из видов декларации является комментарий. Любой HTML-комментарий должен начинаться с конструкции <!-- и заканчиваться -->. Между ними может находиться любой текст, цифры, символы и прочее, за исключением тегов.

HTML-комментарии позволяют размечать структуру HTML-кода, давая заголовки различным логическим блокам элементов. Впоследствии такая разметка поможет быстро сориентироваться и найти необходимый фрагмент кода.

**Пример 1.1** – Комментарии в HTML-коде

```
<!-- Начало блока новостей-->
    Код блока новостей...
<!-- Окончание блока новостей -->
```

Коме того, в комментариях можно указать информацию об авторском праве и прочие персональные данные, которые будут проиндексированы поисковыми системами.

При всем удобстве включения в HTML-код комментариев необходимо помнить, что просмотреть код электронного документа может практически любой пользователь Интернета, открыв нужную web-страницу в любом текстовом редакторе. Поэтому размещать в комментариях информацию конфиденциального, коммерческого или личного характера не рекомендуется.

**Замечание 1.1.** Вложенность комментариев недопустима.

Более детальную информацию о декларациях мы рассмотрим в главе 8.

## 1.4 Структура и привила оформления HTML-документа

В силу своей специфики HTML-документы должны создаваться на основе некоторых правил. Чтобы обеспечить согласованность HTML-документов с этими правилами, а также сделать возможной проверку их корректности, в HTML-документах должны присутствовать определенные элементы. В языке HTML есть заранее определенный набор таких элементов. Все необходимую информацию об элементах языка HTML можно найти на сайте консорциума http://www.w3.org.

Структура HTML-документа состоит из нескольких правил. Абсолютно в любом языке разметки есть единственный корневой элемент. ***Корневым*** называют элемент, внутри которого находятся все остальные элементы. Этот элемент всегда образуется при помощи открывающего и закрывающего тега. В HTML корневым элементом является <**html**>…</**html**>. Любой HTML-документ должен начинаться тегом

<html> и заканчиваться тегом </html>. Этот тег указывает на то, что данный документ содержит в себе HTML-текст. Большинство браузеров способно распознать HTML-документ и без указания данного тега, тем не менее, пропускать раздел HTML разработчикам не рекомендуется.

Языком HTML предусмотрены еще два обязательных раздела: заголовок и содержание.

**<head> ... </head> -** заголовок HTML-программы. Заголовок также называют головной частью программы, он выполняет функцию рабочего заголовка HTML-документа и является, по сути, «невидимым» (теги, указываемые внутри этого раздела, чрезвычайно важны и могут сильно влиять на внешний вид документа, но сами остаются незаметными глазу пользователя).

**<body> ... </body> -** содержание HTML-программы. Основное содержание страницы, которое отображается браузером, помещается в тег <body> ... </body>. Его также называют телом программы.

Эти три элемента (html, head, body) являются обязательными.

**Замечание 1.2**. Язык HTML не чувствителен к регистру, т.е. все равно какими буквами писать: прописными или строчными (например, BODY эквивалентно body или Body). Однако лучше писать в нижнем регистре, поскольку в HTML5 используется только нижний регистр.

В разделе head может прописываться парный тег <**title**>…</**title**>, предназначенный для указания имени созданного электронного документа. Следует помнить, что под именем документа в данном случае имеется в виду не файловое наименование, а визуальный заголовок HTML-страницы. Указание конструкции <title>…</ title > не является обязательным, однако рекомендуется по ряду причин:

- отсутствие тега названия документа заставит браузер при интерпретации HTML-кода вывести в заголовке окна фразу типа Untitled Document (документ без названия), что не соответствует ни тематике вашего электронного документа, ни его наполнению;
- при попытке добавить созданный HTML-документ (без тега <title>…</title>) в «закладки» браузера пользователю придется самостоятельно вписывать название добавляемой страницы;
- поисковые системы, столкнувшись с безымянной страницей, занесут ее в свои базы данных под заголовком Untitled, что сделает HTML-документ безликим и похожим на миллионы других электронных документов, размещенных в Интернете.

Для создания HTML-документов можно воспользоваться любым текстовым редактором, например Notepad (Блокнот). Все HTML-файлы следует сохранять с расширением .htm или .html, что укажет компьютеру, что файл содержит коды HTML.

Таким образом, любой HTML-документ имеет следующую структуру:

**<html>**
  **<head>**
    <**title**>Заголовок документа</**title**>
  **</ head >**
**<body>**
    …текст документа…
**</ body >**
**</ html>**

После создания HTML-файла, его необходимо просмотреть браузером, например Opera, см. рис. 1.2.


*Рисунок 1.2 – Вид HTML-документа в окне браузера*

## 1.5 Основные элементы форматирования текста. Элементы блочные и строчные

Теперь перейдем к изучению тегов, которые прописываются внутри раздела body. Все элементы внутри раздела body можно разделить на две основные группы: элементы *строчные* (inline) и *блочные* (block). Строчные элементы используются для оформления кусков текста, а блочные для структуры блоков текста на странице. Заметим также, что блочные элементы занимают всю доступную ширину. Внутри структуры строчных элементов блочные элементы разместить нельзя.

В случае необходимости указания специальных свойств отдельному фрагменту текста используются теги *текстовых блоков* <**span**> и <**div**>. Теги  и  являются яркими представителями строчных и блочных элементов соответственно. Поэтому между  и  имеются существенные отличия. Во-первых,  берет начало в области физического форматирования текста, а  является исключительно структурным тегом. Во-вторых,  создает принудительный перенос строки на одну позицию после своего закрывающего тега (в отличие от тега абзаца <p>, который осуществляет

перенос на две позиции), поэтому задавать с его помощью отдельные свойства фрагмента внутри абзаца нельзя. Если такая необходимость возникла, лучше использовать тег , который позволяет назначать новые правила отображения текстовых фрагментов без изменения структуры документа. Обращаем ваше внимание на то, что для тегов  и  обязательно наличие закрывающих тегов  и .

Особенности работы тегов  и  можно увидеть на рис. 1.3.


*Рисунок 1.3 – Блочные и строчные элементы*

Метки вида <**hi**> (i = 1 ÷ 6) описывают *заголовки* шести различных уровней. На рис. 1.4 показано, что заголовок **<h6>** будет минимальным, а **<h1>** - самым большим. Справа на рис. 1.3 изображен вид заголовков различных уровней в окне браузера, а слева HTML-код. Использование заголовков помогает отделить фрагменты текста по смыслу, а также помечать важность этих фрагментов. Особенностью тегов <hi> … </hi> (i = 1 ÷ 6) является то, что они автоматически делают отступ от текстовой части. Заголовки являются блочными элементами, поэтому внутри заголовков могут находиться только строчные элементы.

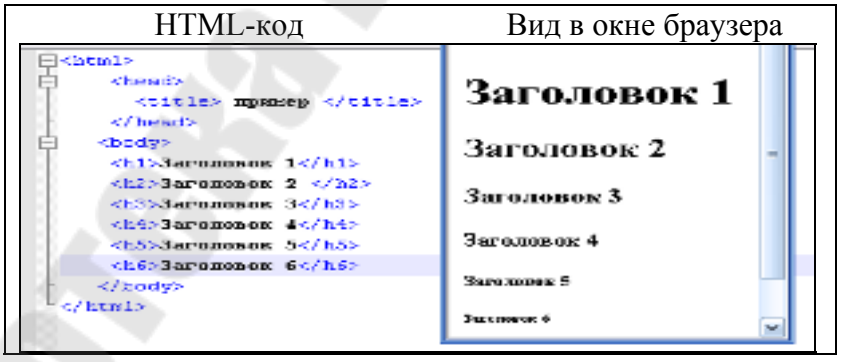



*Рисунок 1.4 – Заголовки различных уровней*

Заголовок h1 лучше делать единственным на странице. Не рекомендуется пропускать уровни заголовков. Если оформление документа начато с заголовка h1, то следующим лучше поставить заголовок h2, потом h3 и т.д. Это улучшит восприятие информации пользователями вашей web-страницы.

Элемент <**hr**> (*горизонтальная линия*) является одиночным пустым и блочным. Этот элемент образует полоску и выполняет функцию визуального разделителя фрагментов. Элемент <hr>, поскольку он является блочным, занимает по ширине все доступное пространство. На рис. 1.5 приведен пример горизонтальной линии.


*Рисунок 1.5 –Горизонтальная линия*

При наборе текста в любом текстовом редакторе для обозначения абзаца используют клавишу <Enter>. Такое действие дает программе команду обособить один фрагмент текста от другого, задав «красную строку». При создании HTML-документа для обозначения *абзаца* (*параграфа*) используется специальный парный тег <**p**>..</**p**>, который разделяет фрагменты текста вертикальным отступом. Простой перевод строки в данном случае не поможет: браузер, интерпретируя код, не воспримет отступ как команду создания абзаца, и при выводе содержания на экран монитора объединит фрагменты текста вместе.

В примере на рис. 1.6 открывающий тег  есть, а закрывающий тег  отсутствует. Однако данная оплошность не влияет на предназначение тега . Для параграфа наличие закрывающего тега не обязательно, поскольку параграфы – блочные элементы, внутри которых могут быть только строчные элементы. При анализе элементов браузер находит открывающий тег , анализируя внутренне содержание, находит опять блочный элемент  и автоматически ставит закрывающий тег.

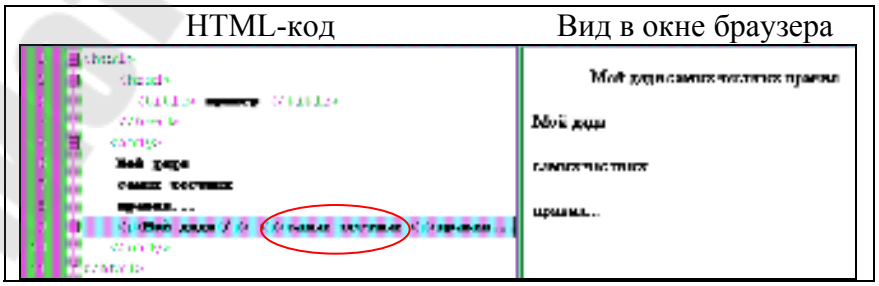



*Рисунок 1.6 – Абзац*

При написании кодов HTML-документов, следует знать, что браузеры динамически форматируют текст. Браузер заполняет текстом все доступное пространство. Если поменять это пространство (уменьшить или увеличить размер окна браузера), то и изменится формирование текста: возникнут новые переносы. Делает это браузер на основании пробельных символов.

Есть три вида пробельных символов: клавиша пробел (пробел), клавиша Tab (табуляция), и клавиша Enter (перенос строки). Ориентируясь на эти пробельные символы, происходит перенос текста браузером. Язык HTML характерен тем, что любое количество идущих подряд пробельных символов, заменяется при анализе строк на один единственный пробел.

Как видно из рисунка 1.6, тег <p> осуществляет перенос текста на две позиции. Если необходимо перейти на новую строку, не прерывая абзаца, лучше использовать строчный пустой элемент <**br**>. Использование тега <br> очень удобно при публикации стихов.

Если вам необходимо отобразить текстовую строку фиксированной длины без переносов, используется парный тег <**nobr**>..</**nobr**>. При использовании данного тега в случае, если длина строки превышает ширину экрана, в нижней части окна браузера появится горизонтальная полоса прокрутки.

Для того чтобы задать блочную цитату, можно использовать блочный элемент <**blockquote**>…</**blockquote**>. Поскольку этот элемент является блочным, внутри его могут располагаться как строчные, так и блочные элементы. Элемент <blockquote>…</blockquote> создает небольшие отступы слева и справа (по умолчанию 40px). Перед цитатой и после ее браузер оставляет по одной пустой строке. Раньше отступы можно было сделать только с помощью этого элемента. Сейчас элемент блочной цитаты используется очень редко. Для форматирования коротких цитат используется элемент <**q**>..</**q**>. Содержимое элемента отображается без разрыва строки. Цитаты, оформленные с помощью элемента <**cite**>…</**cite**>, выделяются курсивом.

Для авторского форматирования используется блочный элемент <**pre**>…</**pre**>, внутри которого могут находиться только строчные элементы. В этом случае все, что расположено между тегами <pre>…</pre> отобразится в окне браузера с сохранением всех введенных разработчиком пробелов, переводов строк и отступов.

С помощью блочного парного тега <**address**>…</**address**> можно задать курсивный текст в виде абзаца. Внутри данного элемента могут находиться только строчные элементы. Этот элемент используется для размещения в HTML-документе контактной информации.

Все элементы непосредственного форматирования текста делятся на две категории: элементы логического форматирования и элементы визуального форматирования.

К элементам *визуального оформления* относятся теги, представленные на рис.1.7.

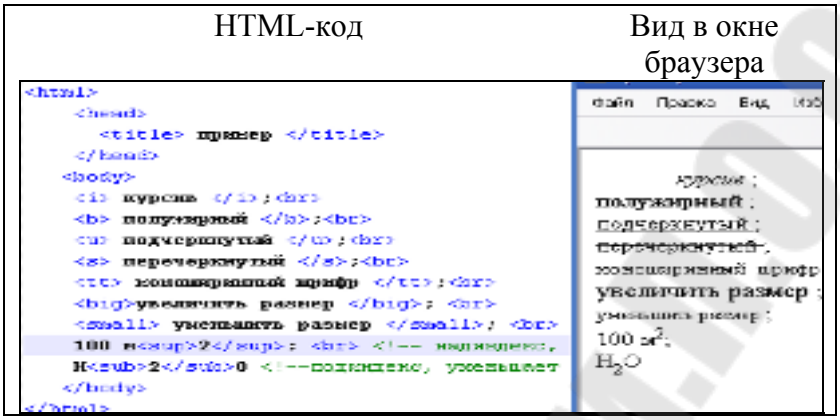



*Рисунок 1.7 – Элементы визуального форматирования*

Данные пары тегов можно вкладывать друг в друга. Однако, необходимо соблюдать порядок вложенности тегов. Например, полужирный курсив будет выглядеть так: Полужирный курсив.

Теги <big>…</big>, <small>…</small>, <sup>…</sup>, <sub>…</sub> нельзя ставить внутри тега <pre>…<pre>, потому, что они изменяют размер шрифта и сдвигают его. Все элементы визуального форматирования, кроме  и , не вошли в HTML5.

К элементам *логического форматирования* относятся следующие элементы:

 – выделение важных фрагментов текста *курсивом*.

 – выделение наиболее важных фрагментов **полужирным**.

Рекомендуется заменять  на , а  на .

<ins> – выделение фрагмента подчеркиванием, когда требуется визуально показать, что текст был вставлен после опубликования документа.

<del> – выделение фрагмента ~~перечеркиванием~~, когда требуется визуально показать, что текст был удален после опубликования документа.

Элементы <ins>, <del> могут выступать и как строчные, и как блочные элементы.

<cite> – выделение цитат *курсивом*.

<code> – отображение фрагментов программного кода `моноширинным шрифтом`.

<kbd> – текст, вводимый с клавиатуры: отображается моноширинным шрифтом.

<var> – название переменных: отображается *курсивом*.

<samp> – выделение нескольких символов моноширинным шрифтом.

<dfn> – определение вложенного термина *курсивом*.

Все элементы логического форматирования вошли в HTML5.

**Важно!** Чтобы лучше ориентироваться в большом коде, его нужно правильно и грамотно форматировать. Открывающие и закрывающие теги должны стоят полвертикали на одном уровне. Внутренние элементы можно сдвигаем с помощью табуляции. Табуляцию легко настроить на два и более пробельных символа. С помощью табуляции можно передвигать несколько строчек одновременно, предварительно их выделив. Нажатие клавиш SHIFT+TAB поможет вернуть строки обратно. Правильное форматирование кода может помочь увидеть, где допущена ошибка, а также снизить процент возможных ошибок.

## 1.6 Понятие элементов и атрибутов

Один и тот же тег может интерпретироваться браузером по-разному. Это зависит дополнительных параметров. Эти параметры называются *атрибутами* (*параметрами*) *тега*. Атрибуты уточняют действие тегов. Каждый тег имеет свой набор допустимых атрибутов. Атрибуты указываются только в открывающем теге. Для одного тега можно использовать несколько атрибутов, разделенных пробелами. Отметим, что имена атрибутов заранее определены, а вот значение атрибуту присваивается заданное или произвольное.

Синтаксис записи тега и соответствующими атрибутами следующий:

**<тег имя_атрибута_1= “значение” …**
**имя_атрибута_n=”значение”>**

Например, <hr **align**=”left” **width**=”50%” **size**=4>.

Кавычки могут быть как одиночные, так и двойные, но только одинаковые. Например, в HTML используются двойные кавычки, в Java Script – только одиночные.

Все атрибуты можно разделить на три основные группы:

1. общие. К ним относятся следующие атрибуты: id, class, lang, dir, title, style. Общие атрибуты могут стоять практически у подавляющего большинства элементов;

2. атрибуты событий. Эта группа атрибутов нужна для того, чтобы вызвать действие в ответ на действие пользователя или на какие-то системные события;

3. уникальные. К этой группе относятся индивидуальные атрибуты некоторых элементов.

**Замечание 1.4.** Корневой элемент <html> атрибутов не имеет.

Рассмотрим некоторые атрибуты для изученных элементов.

Атрибут **align** может применяться только для блочных элементов и имеет следующие значения: left, right, center, justify.

Для заголовков <hi> … </hi> (i – цифра от 1 до 6) атрибут align означает, что его содержимое будет выравниваться по левому краю при align=”left” (по умолчанию), по правому краю при align=”right”, все строки будут по центру при align=”center”, выравнивание произойдет как по левому, так и по правому краю при align=”justify”. Для маленьких кусков текста justify не имеет смысла.

У пустого элемента <hr> есть 4 атрибута: align, noshade, size, width, color.

Для тега <hr> атрибут align означает выравнивание самой горизонтальной линии и принимает следующие значения: right, left, center.

Выравнивание для элемента <hr> имеет смысл только тогда, когда прописан атрибут **width** с заданным значением. Значение атрибута width может быть задано в пикселях или процентах. Например, тег <hr width=15> определяет горизонтальную линию в 15 пикселов. Если значение атрибута width задано в процентах, то ширина линии вычисляется относительно ширины окна браузера. Например, <hr width=”40%”>. По умолчанию, значение атрибут width имеет 100%.

Атрибут **size** задает высоту горизонтальной линии в пикселях. Значение данного атрибута может варьировать от 1 до 175. По умолчанию, это значение равно 2px.

Атрибут **noshade** (не отбрасывать тень) является одиночным и не имеет никакого значения. В разных браузерах использование этого атрибута может привести к различным результатам.

Атрибут **color** определяет цвет горизонтальной линии.

На рис. 1.8 предложена коллекция горизонтальных линий разной ширины и высоты.

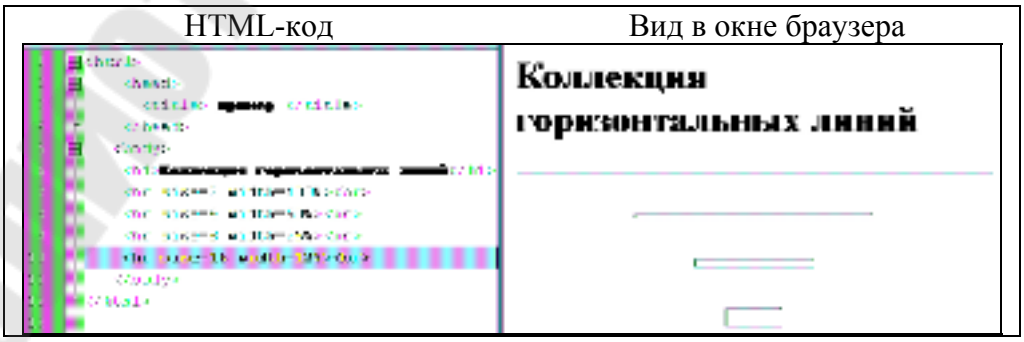



*Рисунок 1.8 – Коллекция горизонтальных линий*

Значение цвета в атрибутах языка HTML может задаваться несколькими способами.

– по шестнадцатеричному значению

Для задания цветов используются числа в шестнадцатеричном коде. Шестнадцатеричная система базируется на числе 16. Цифры будут следующие: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Числа от 10 до 15 заменены на буквы латинского алфавита. Числа больше 15 в шестнадцатеричной системе образуются объединением двух чисел в одно. Например, числу 255 в десятичной системе соответствует число FF в шестнадцатеричной системе. Чтобы не возникало путаницы в определении системы счисления, перед шестнадцатеричным числом ставят символ решетки #, например #666999. Каждый из трех цветов — красный, зеленый и синий — может принимать значения от 00 до FF. Таким образом, обозначение цвета разбивается на три составляющие #rrggbb, где первые два символа отмечают красную компоненту цвета, два средних — зеленую, а два последних — синюю. Допускается использовать сокращенную форму вида #rgb, где каждый символ следует удваивать (#rrggbb). К примеру, запись #fe0 расценивается как #ffee00.

– по названию

Браузеры поддерживают некоторые цвета по их названию. В таблице 1.1 приведен шестнадцатеричный код, название и описание. Более полную таблицу цветов можно найти в книге [2].

Таблица 1.1 – Некоторые стандартные цвета

| **Код** | **Название** | **Описание** |
|---|---|---|
| #000000 | ЧЕРНЫЙ | black |
| #FFFFFF | БЕЛЫЙ | white |
| #FF0000 | КРАСНЫЙ | red |
| #008000 | ЗЕЛЕНЫЙ | green |
| #0000FF | СИНИЙ | blue |
| #808000 | ОЛИВКОВЫЙ | olive |
| #FFFF00 | ЖЕЛТЫЙ | Yellow |
| #C0C0C0 | СЕРЕБРИСТЫЙ | Silver |
| #808080 | СЕРЫЙ | Gray |
| #FF00FF | ФУКСИНОВЫЙ | Fuchsia |
| #000080 | УЛЬТРАМАРИН | Navy |
| #008080 | СИЗЫЙ | Teal |
| #00FF00 | СВЕТЛО-ЗЕЛЕНЫЙ | Lime |
| #800080 | ПУРПУРНЫЙ | Purple |
| #800000 | КАШТАНОВЫЙ | maroon |

– с помощью RGB

Можно определить цвет, используя значения красной, зеленой и синей составляющей в десятичном исчислении. Значение каждого из трех

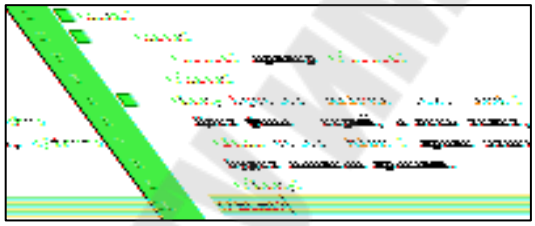
цветов может принимать значения от 0 до 255. Также можно задавать цвет в процентном отношении. Вначале указывается ключевое слово rgb, а затем в скобках, через запятую указываются компоненты цвета, например rgb(255, 0, 0) или rgb(100%, 20%, 20%).

Цвет и фон страницы выбираются по желанию дизайнера, но нужно учитывать и тот факт, что от выбора зависит визуальное восприятие всего сайта. Не стоит стремиться затемнить все, так как текст удобно читать, если его хорошо видно. Для вставки цвета фона в строку с тегом <body> нужно добавить атрибут **bgcolor** и указать его значение – название цвета или его шестнадцатеричный вид. Фон страницы станет красным, если прописать одну из следующих строк.

<body **bgcolor**=”red”> <!-- использовано название цвета-->

<body **bgcolor**=”#FF0000”> <!--использован шестнадцатеричный вид цвета-->

Для задания цвета текста в HTML есть специальный атрибут **text**,



который может применяться только в теге <body>.

*Рисунок 1.9 – Атрибуты тегов <body> и <font>*

Тег <body bgcolor=”silver” **text**=”red”> задаст цвет фона HTML-страницы серым, а цвет текста красным, см. рисунок 1.9.

Для определения свойства шрифта в HTML есть специальный элемент **<font>…</font>**. Этот элемент имеет 3 атрибута: color, size, face.

Синтаксис элемента font следующий:

**<font color=”значение_1” size=”значение_2” face=”значение _3”> любой текст </font>**.

Чтобы написать текст красным цветом, необходимо вставить перед ним парный тег <font>…</font> и присвоить его атрибуту **color** значение red или #FF0000 (см. рисунок 1.7):

<font **color**="red">Красный текст</font> или <font **color**="#FF0000">Красный текст</font>.

Вторым атрибутом элемента **<font>…</font>** является размер шрифта – **size**. Параметр этого атрибута может быть описан либо абсолютной, либо относительной величиной. Абсолютная величина подразумевает использование в качестве параметра целого числа, указывающего на высоту шрифта в пунктах. Относительная же величина, обозначаемая целым числом со знаком плюс или минус (например, +2 или

-1), - это количество пунктов, которые следует прибавить или отнять от размера шрифта, используемого браузером (по умолчанию 12pt). Так, запись <font **size**=”+1”> говорит о том, что размер помеченного таким образом текста будет на один пункт больше, чем обычный текст документа. Кроме того, размер шрифта можно записывать целыми числами от 1 до 7 (самый мелкий шрифт имеет размер 1, а самый большой – 7; size=”3” – значение шрифта по умолчанию).

С помощью атрибута **face** тега <font> можно менять стиль написания (гарнитуру) шрифта, например:

<font **face**=”Times New Roman”> Текст с начертанием Times New Roman </font>.

Следует помнить, что если на компьютере посетителя вашей web-страницы не установлен шрифт с указанным Вами названием, то браузер использует свой стандартный шрифт. Вы можете задать несколько возможных шрифтов, тогда в случае отсутствия первого шрифта браузер будет использовать второй, если нет второго, использует третий и т.д. Например: <font **face**=”Comic Sans MS, Courier New, Courier”>.

Существует 5 основных семейств шрифтов:

1. Шрифты с засечками (семейство serif). Например, Georgia, Times New Roman, Palatino и др.;
2. Рубленные шрифты или шрифты без засечек (семейство sans-serif). Например, Arial, Helvetica, Verdana, Tahoma и др.;
3. Моноширинные шрифты (семейство monospase). Например, Courier, Courier New, Lucida Sans, Consolas и др.;
4. Fantasy;
5. Курсив (italic).

## 1.7 Специальные символы языка HTML

К специальным символам HTML относятся символы, не входящие в состав стандартных ASCII-кодов. Их реализация в HTML-документах возможна при помощи отдельных кодовых конструкций или числовых комбинаций.

К специальным символам относятся: знак авторского права и зарегистрированной торговой марки, значки иностранных валют и математические символы, дробные числа и элементы HTML-форматирования, буквы иностранных алфавитов и многое другое. Например, знак доллара ($) можно легко ввести с помощью соответствующей клавиши, не опасаясь за корректность отображения данного символа в любых кодировках, моделях и версиях браузеров. А вот отобразить таким же образом значок английской денежной единицы – фунта стерлингов – вряд ли удастся.

Условно все специальные символы HTML можно разделить на три большие группы:

1. *Символы, отображающие элементы HTML-форматирования*. Часто бывает необходимо отобразить в браузере символы, используемые в спецификации языка HTML: теги (< >), знак амперсанда (&) и т. д. Если необходимо в браузере отобразить сам код, нужно все левые угловые скобки в коде заменить на *<*, а правые — на *>*. После замены всех символов специальными кодовыми конструкциями первоначальный фрагмент кода примет вид, как на рис. 1.10.

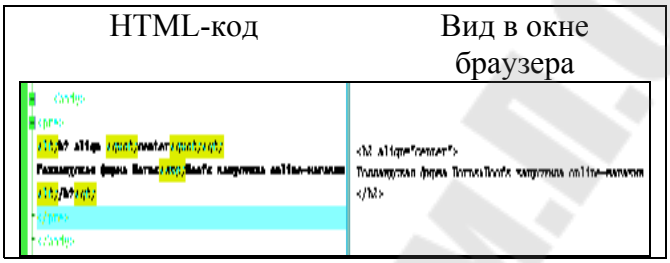



*Рисунок 1.10 – Пример оформления фрагмента кода*

2. *Символы оформления документа*. К так называемым символам оформления HTML-документов относятся значки авторского права, зарегистрированной торговой марки, символы иностранных валют, математические обозначения («плюс-минус», «умножить» и др.) и пр. Приведенный на рис. 1.11 пример, может быть получен за счет вставки в код документа следующих конструкций:

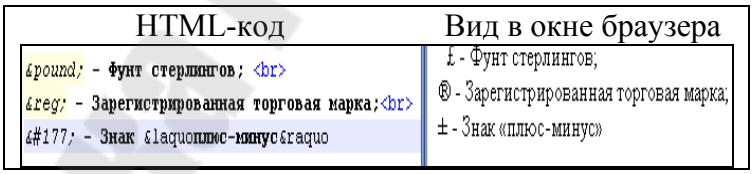



*Рисунок 1.11– Некоторые символы оформления HTML-документов*

3. *Буквы иностранных алфавитов.* Когда создается HTML-документ на русском или английском языке, никаких затруднений с отображением букв в браузере не возникает. Трудности появляются, если необходимо разработать электронный документ на языке, алфавит которого содержит некоторые буквы, чье начертание характерно только лишь для данного языка (например, португальский).

Выйти из такого положения можно несколькими способами.

а) добавить в раскладку клавиатуры новый язык, включить в используемом текстовом редакторе поддержку нужного языка и, выбрав на

Панели задач значок подключения языка, спокойно набирать текст. Однако в этом случае текст будет некорректно интерпретирован браузером.

b) прибегнуть к услугам визуальных web-редакторов, которые включают мультиязыковую поддержку. Однако последующая программная конвертация исходного текста в формат HTML может привести, во-первых, к некорректному отображению его на web-странице некоторыми моделями и версиями браузеров, а во-вторых — к генерации совершенно ненужного и бесполезного кода, только увеличивающего конечный размер файла.

c) использовать специальные кодовые или числовые конструкции, предназначенные для корректной визуализации нестандартных букв иностранного алфавита. Например, чтобы браузер правильно отобразил слова на португальском языке, показанном на рис. 1.12, в код документа нужно вписать следующую конструкцию:

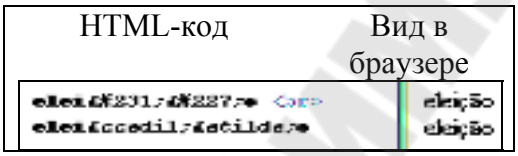



*Рисунок 1.12 – Символы португальского языка*

В результате никаких проблем с кросс-браузерной совместимостью и «утяжеленным» размером HTML-файла.

Полный перечень всех специальных символов, а также их кодовые и числовые конструкции можно найти в [2].

## 1.8 Вопросы для самоконтроля

1. Когда и кем был создан язык HTML?
2. Назовите наиболее популярные браузеры?
3. Дайте определение понятиям гипертекста, web-страницы и web-сайта.
4. Опишите структуру тега. Назовите виды тегов. Приведите примеры.
5. Какова структура HTML-документа?
6. Опишите структурные теги (<html> , <head> , <title>, <body>).
7. Дайте характеристику и назовите атрибуты тега <body>.
8. Каков порядок создания HTML-документа?
9. Дайте характеристику тегов <h1>...</h1> ÷ <h6>...</h6>, <nobr> … </nobr> и ... .
10. Дайте характеристику тегов <br> , <hr>, <pre>...</pre>.

11. Назовите элементы визуального и логического форматирования. Какие из них вошли в спецификацию HTML5?
12. Дайте характеристику тега <font>…</font>.
13. Опишите три основные категории специальных символов.

# 2 Ссылки и иллюстрации

## 2.1 Механизмы адресации на ресурсы в Internet. Понятие ссылки

Миллионы электронных документов, которые размещены в Интернете, похожи по тематике и ориентированы на одну и ту же пользовательскую аудиторию. Эти документы связаны между собой с помощью гипертекстовых ссылок, по нажатию на которые происходит переход от одного документа к другому. Успешный переход по ссылке возможен в двух случаях:

– ресурс, на который ссылается документ, существует. Этот фактор является объективным, т.к. разработчик электронного документа, однажды поставив в нем ссылку на внешний ресурс, может и не знать о том, что этот ресурс, например, прекратил свое существование;

– структура гиперссылки верна с точки зрения HTML. Этот фактор относится к разряду субъективных, так как только от создателя HTML-документа может зависеть, сумеет ли посетитель перейти по ссылке или она составлена неверно.

Чтобы последнего не произошло, рассмотрим структуру и правила описания гипертекстовых ссылок.

*Ссылка* представляет собой логическую связь одного фрагмента (элемента) документа с другим фрагментом в том же самом или в другом документе. Ссылка характеризуется двумя точками, называемыми закладками. Различают начальную и конечную закладки.

*Начальная закладка* устанавливается на конкретном HMTL-элементе и определяет точку, из которой задается ссылка. Начальные закладки можно размещать в тексте, таблицах, списках изображениях, фреймах. Начальная закладка выделяется на странице подчеркиванием и цветом.

*Конечная закладка* определяет точку назначения ссылки и может относиться не только к конкретному HTML-элементу, но и к программе или документу в целом.

В HTML различают внутренние и внешние гиперссылки. *Внутренние ссылки* осуществляют переход в пределах одного и того же документа. *Внешние ссылки* обеспечивают переходы к другим документам. Внешние ссылки и называют *гиперссылками*.

Текстовые гиперссылки можно разделить на четыре условных категории:

а) ссылки на документы,
б) ссылки на разделы,
в) ссылки на адрес электронной почты,
г) ссылки на файловые объекты.

## 2.2 Элемент <a>…</a> и его атрибуты

Как уже отмечалось ранее, с помощью ссылок осуществляется связь текста или картинки с другими гипертекстовыми документами. Текст ссылки, как правило, выделяется цветом и (или) оформляется подчеркиванием. Ссылка создается с помощью тега <a>…</a> и его атрибутов и имеет следующую конструкцию:

**<a href ="**URL**" target ="**Окно**" title ="**Подсказка**">Название ссылки</a>**

Элемент <a>…</a> имеет следующие атрибуты:

***href*** – принимает значение URL (Uniform Resource Locator, унифицированный локатор ресурсов) – адрес любого файла в Интернете. Может быть абсолютными, то есть указывается полный адрес странички (например, http://tut.by/index.html) и относительным, указывается файл относительно текущего (например, index.html).

***target*** – определяет, в каком окне загрузить гиперссылку. Может иметь значения:

*_top* – загружает гиперссылку на всем пространстве окна браузера (если до этого существовало разбиение на фреймы, то оно исчезнет).

*_blank* – загружает гиперссылку в новом окне браузера.

*_self* – загружает содержимое страницы, в окно, которое содержит эту ссылку (используется по умолчанию).

*_parent* – загружает содержимое страницы, заданной ссылкой, в окно, являющееся непосредственным владельцем набора фреймов.

***title*** – задает текст подсказки, который будет появляться при наведении мышки на гиперссылку. Параметр не обязательный.

Для примера создадим ссылку в документе menu.html на заранее созданный документ index.html (см. пример 2.1). Предполагается, что оба документа находятся в одной папке.

**Пример 2.1.**

```
<p align="center">
    <a href="index.html" target="self" title="Пример
ссылки">Ссылка</a>
</p>
```

Щелкнув на ссылку, откроется другой документ, в данном случае index.html.

Вы можете также создать ссылку на e-mail, в это случае нужно прописать следующее:

**<a href="mailto:po4ta@tut.by">**po4ta@tut.by**</a>**

Почтовая гиперссылка имеет несколько параметров (не обязательных):

&subject - тема письма;

&body - текст вашего сообщения;
&cc - копии письма через запятую;
&bcc - скрытые копии письма через запятую;
Атрибут title="Выпадающая подсказка" ставиться по желанию и располагается отдельно от параметров почтовой ссылки:

<a **href**="mailto:po4ta@tut.by &**subject**=Письмо &**body**=text
&**cc**=copy@tut.by &**bcc**=hidden_copy
@tut.by" **title**="Пример почтовой гиперссылки">Почта</a>

Иногда возникает непобедимость сделать *ссылку на определенное место* в том же или в другом документе. Чтобы нажав по какой-нибудь ссылке можно было попасть в определенное место данного документа, необходимо создать *закладки*.

Ссылка на закладку в том же документе имеет следующий вид:

<a href="**#Имя закладки**">Название раздела</a>.

А так выглядит ссылка на закладку в другом документе:

<a href="**Имя документа#Имя закладки**">Название раздела</a>

Сама закладка будет такой:

<a **name**="**Имя закладки**"> …текст… </a> или
<тег **id**="**Уникальное имя закладки**">…текст…</тег>

Щелкнув на фразу «Название раздела» пользователь будет попадать на определенную вами закладку.

Для того, чтобы при наведении на ссылку курсором и при клике она меняла свой цвет, в тег <body> нужно добавить еще несколько параметров: text – цвет текста, link – цвет ссылки, vlink – цвет пройденой ссылки, alink – цвет активной ссылки, когда подводиться к ней курсор.

Например, <body **text**="black" **link**="blue" **vlink**="purple" **alink**="red">

Данные атрибуты определяют свойства для всего документа. Поместив такой код в HTML-документ, уже не надо будет назначать каждый раз цвет текста и цвет ссылок, т.к везде он будет таким, каким определен в теге <body>.

## 2.3 Размещение иллюстраций на web-странице. Типы файлов иллюстраций

На данный момент практически все браузеры широко поддерживают три формата: GIF, JPEG, PNG. Рассмотрим каждый из них.

**Формат GIF** (Graphic Interchange Format, формат обмена графикой), созданный в 1987 году, является первым форматом, который появился для обмена медийной информации. Изображение, кодированное в формат GIF, всегда содержит от 2 до 256 цветов. Такая палитра цветов называется *индексированная* или *фиксированная*.

*Отрицательные стороны формата GIF*.

Если изображение содержит цвета, принадлежащие индексированной палитре, то цвета будут передаваться как есть, без искажений. Однако, если цвета картинки выступают за границу 256 возможный, то графический редактор, подсчитав по своим гистограммам наиболее часто встречаемые цвета, выделит из них 256, а все остальные создаст с помощью подмешивания. Это означает, что отдельные пиксели на вашем изображении будут использовать базовую палитру, а при отдалении, как будто бы смешиваясь, они передают «недостающие» цвета. Если, например, в составе рисунка имеется участок, состоящий из нескольких сходных полутонов, к примеру, голубого, светло-голубого и темно-голубого цвета, они будут кодированы одним оттенком – голубым. В документах HTML формат GIF используется только для отображения так называемой бизнес-графики: диаграмм, логотипов, кнопок, разделительных линий и других элементов оформления страницы. Таким образом, GIF способен работать только с 256 цветами, для полноцветных изображений формат GIF не используют.

*Положительные стороны формата GIF.*

Положительной стороной формата GIF является возможность мультипликации, которая была выявлена в начале 1990-х годах. Анимационный GIF является обыкновенным графическим файлом. Изюминка в том, что подобный файл состоит из нескольких изображений, которые через тот или иной обозреватель Internet последовательно выводятся на экран пользователя.

Чтобы создать анимационную картинку, вам необходимо сначала создать картинки, из которых будет состоять результирующий файл. Эти картинки можно сделать, например, в Adobe Photoshop, а собрать их вам поможет одна из следующих программ:

- GIF Construction Set 32;
- Microsoft GIF Animator.

Растровая графика никакой другой формат пока не поддерживает, кроме формата GIF. Формат mng только разрабатывается и о его поддержке говорить пока не приходится. Поэтому GIF – это единственный формат на сегодняшний день, которые поддерживает мультипликацию.

Формат GIF обладает еще одной необыкновенной возможностью. В данном формате есть возможность указывать любому пикселю состояние прозрачности либо непрозрачности. Использование прозрачных пикселей хорошо тем, что, накладывая изображение с прозрачными областями на любой фон, последний будет просвечивается сквозь эти размытые области. Такая функция необходима, например, при размещении картинок неправильной геометрической формы на странице со сложным фоновым рисунком, когда корректно «подогнать» части изображений друг к другу не представляется возможным. Минусом является то, что графический

редактор в размытые области подмешивает цвет, который называется matte. По умолчанию это может быть белый цвет. Поэтому в любом графическом редакторе, при создании картинки GIF с размытыми краями, обязательно нужно указать цвет matte. Какой цвет matte используется можно увидеть, если положить изображение на другой фон.

Каждый формат имеет в своей основе какой-то *алгоритм сжатия*. Если бы не нужно было сжимать информацию, то изображения можно было передавать в формате bmp, где каждый пиксель описывается тремя байтами. Алгоритм сжатия формата GIF предполагает лучшую работу изменения цвета по вертикали. Если строки монотонные и изменение цвета происходит по вертикали, то формат GIF позволяет очень хорошо оптимизировать такие изображения. Как только возникает градиент горизонтали или наклонной, то тут же размер изображения увеличивается, и могут появляться артефакты (дефекты), особенно при наклонном градиенте. Поэтому надо четко понимать, где нужно использовать формат GIF, а где лучше не использовать.

Одном из свойств формата GIF является его особенность, названная разработчиками «*interlace*», или, «*черезстрочность*». Она позволяет загружать картинку с сервера в клиентский браузер не целиком, а частями следующим образом: сначала на экране отображаются первая, пятая и десятая строки, составляющие изображение, затем – вторая, шестая и одиннадцатая и т. д. Таким образом, для пользователя создается иллюзия постепенной загрузки графического элемента: картинка как бы медленно «проявляется» на странице.

**Формат JPEG** (Join Photographic Experts Group, объединенная группа экспертов в области фотографии) – это графический стандарт, созданный на основе одноименного алгоритма сжатия, изображений с потерей качества, кодирующего не идентичные элементы, а межпиксельные интервалы. Этот формат работает с матрицей 8x8 пикселей, то есть разбивает изображение на квадраты со стороной 8 пикселей и начинает сжимать. Особенностью формата JPEG является то, что он всегда искажает картинку. Невозможно создать картинку формата JPEG с качеством как у оригинала. Искажение происходит при каждом сжатии. Например, если изображение в хорошем качестве сохранить в формате JPEG с максимальным качеством, потом снова сохранили в формате JPEG с максимальным качеством, то через 15-20 раз получится нечто очень страшное. Поэтому рекомендуется изображения в формате JPEG делать из очень хороших исходных файлов.

Формат JPEG не охватывает всю палитру RGB целиком, однако он позволяет включать много цветов. Данный формат был разработан для передачи полноцветных изображений в качестве наиболее приемлемом для человеческого глаза. Формат JPEG оптимален для передачи

фотографических изображений, а также картинок с большим количеством полутонов и цветовых переходов. Максимальное число цветов, которое может содержать изображение в формате JPEG, достигает 16 миллионов.

Формат JPEG обладает таким параметром, как качество. Фактически с помощью этого параметра происходит управление коэффициентом сжатия. Чем сильнее сжимаете, тем качество хуже и тем меньший размер будет иметь файл изображения.

Существует несколько разновидностей стандарта JPEG.

Формат *JPEG Baseline Optimized* основан на более совершенном алгоритме компрессии изображений. Недостатком данной реализации JPEG является то, что она не распознается целым рядом часто используемых приложений.

*Progressive JPEG* был оптимизирован специально для представления графики во Всемирной сети, изображения в этом формате сжимаются чуть лучше, чем в стандартном JPEG, но хуже, чем в JPEG Baseline Optimized. Отличительная особенность Progressive JPEG – возможность сохранять графику в чересстрочном режиме, как это реализовано в стандарте GIF.

**Формат PNG** (Portable Network Graphics, портативная сетевая графика) – это формат, который поддерживает «черезстрочность» не только по горизонтали, но и по вертикали. Данный формат есть в описании на сайте консорциума http://www.w3.org/Graphics/PNG/. Рассмотрим его разновидности.

*PNG-8* – это формат с глубиной цвета 8 бит. Для описания пикселей используются те же 256 цветов. Формат PNG-8 разрабатывался как замена GIF. Крупные ресурсы уже перешли на PNG-8, формат GIF ими уже не используется.

*PNG-24*. Глубина цвета 24 бита означает, что на каждый пиксель используется описание 3-х цветовых каналов по 8 бит. Это позволяет передавать цвета «как есть». Формат PNG-24 в отличии от JPEG не искажает качество изображения. Однако, не теряя в качестве изображения, происходят увеличения размера файла. Изображение, сохраненное в формате PNG-24, «весит» больше, чем изображение в формате JPEG. Формат PNG-24 используется там, где нельзя искажать качество картинки. Что лучше использовать JPEG или PNG-24 зависит от конкретной ситуации.

*PNG-32* – это наиболее часто используемый формат, который имеет следующие возможности:

1. Передача полноцветных изображений.
2. Использование полупрозрачности.
3. Использование альфа-канала для прозрачности (т.е. PNG-32 – это PNG-24 плюс альфа-канал).

Альфа-канал можно использовать не только в PNG-32, но и в PNG-8. В этом и есть принципиальное отличие PNG-8 от GIF. Однако такую возможность вам дадут далеко не все редакторы. Например, Photoshop этого вам сделать не позволит. Photoshop – прекрасная программа для растровой графики, но для web эта программа показывает результаты очень посредственные. Поэтому для картинок, которые вы будите использовать на сайтах, используйте редакторы которые предназначены для web, например, GIMP (GNU Image Manipulation Program) для ОС Linux, Adobe Fireworks для ОС Windows.

Форматы PNG-32 и PNG-8 в Internet Explorer версии 6 и ниже не поддерживаются.

При сохранении изображений GIF и JPEG на диск в файлы записывается определенное количество «лишних» данных, таких, например, как невидимые текстовые комментарии, ссылки на разработчиков стандарта и соответствующего программного обеспечения и т. д. Все эти дополнения увеличивают физический размер файла и не несут никакой полезной нагрузки. Чтобы сократить время загрузки иллюстраций в клиентский браузер, используют специальные методы оптимизации изображений с использованием специальных программ – *оптимизаторов графики*.

## 2.4 Элемент IMG и его атрибуты

Картинку можно вставить как объект на web-странице, что осуществляется с помощью элемента <**img**>. Данный элемент относят к числу особенных, которые еще называют строчными элементами с замещающим контентом (содержимым). К таким элементам относятся: img, iframe, object, embed. Эти элементы ведут себя как строчные, но внутри них отображается посторонний внешний контент. Браузер, анализируя код, как только «встретит» элемент <img>, создает область для строчного элемента, в который загружается внешний файл - картинка. Чтобы показать (загрузить) картинку необходимо прописать следующую строку:

<**img src**="путь_к_файлу">

Например, <**img src**="img.gif"> или <**img src**="pic/img.gif">, или <**img src**="http://site.ru/pic/img.gif">.

Далее, браузер при загрузке картинки определяет ее истинные размеры, на которые и «раздвигает» выделяемую строчную область. Если необходимо, чтобы браузер сразу выделял область нужного размера, то указывают атрибуты width и height.

Например, <img src="img.gif" **width**="100" **height**="150">

Браузер будет «вписывать» изображение в определенные разработчиком размеры, даже если картинка имеет размеры больше, чем выделенная область.

Поскольку изображение – это строчный элемент, т.е. его можно считать просто большой буквой, то такое изображение можно вставлять внутрь гиперссылок, тем самым делая его кликабельным. Для этого вместо названия ссылки нужно прописать графический элемент (См. пример 2.2). В этом случае вокруг изображения появится рамка в 3 пикселя (по умолчанию), цвета, определенного для гиперссылок.

**Пример 2.2.**

```
<a href="URL" target="окно">
    <img src="путь_к_рисунку" width="ширина" height="высота"
title="подсказка">
</a>
```

В примере 2.2. атрибут title может быть только у рисунка.

Обратим свое внимание на подписи к изображениям. Здесь имеются следующие особенности:

1. Ссылки на изображения могут иметь неправильный адрес. Например, поставлен лишний пробел или просто изображение удалено с сервера;
2. Соединение может быть прервано;
3. Для незрячих пользователей изображения загружаются с помощью голосового браузера.

Во всех этих случаях используется атрибут *alt* (от англ. alternative, альтернатива).

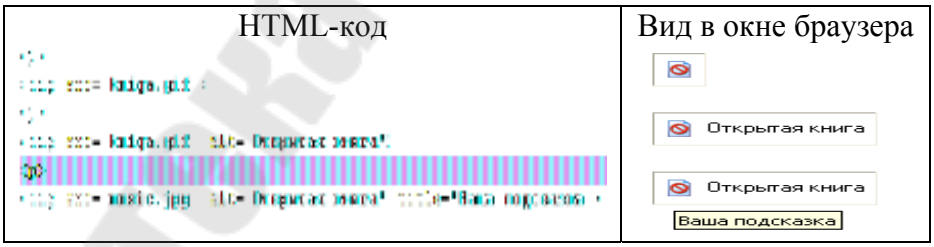



*Рисунок 2.1 – Использование атрибута alt*

На рис. 2.1 представлен случай, когда изображение не загрузилось, но область под него была сформирована. Чтобы пользователь представлял, что на этом месте должна быть картинка разработчики ставят атрибут alt, где пишут описание своего изображения. Для голосовых браузеров этот текст альтернативного представления просто читается. Если навести курсором мыши на рисунок, то появится описание картинки. В нашем случае это будет фраза – «Открытая книга». Если параметр alt не задавать,

описания не будет. Рекомендуется всегда задавать описание картинкам, т.к. есть пользователи, которые «бродят» по интернету с отключенной графикой.

Для любых элементов, не обязательно для изображений, есть необходимость показывать при наведении какой-то поясняющий текст. Например, если необходимо сделать меню в виде графических кнопок, а места для их подписи мало. В таком случае, подсказки можно делать с помощью специального атрибута *title*. Этот атрибут можно ставить практически в любом элементе на странице, кроме раздела head. Атрибут title используется очень часто. Напомним, что атрибут alt нужен для альтернативного представления, а title для подсказок.

Рассмотрим вопрос расположения изображений относительно текста. На рис. 2.2 представлен самый простой случай, при котором рядом с рисунком располагается только одна строчка текста.

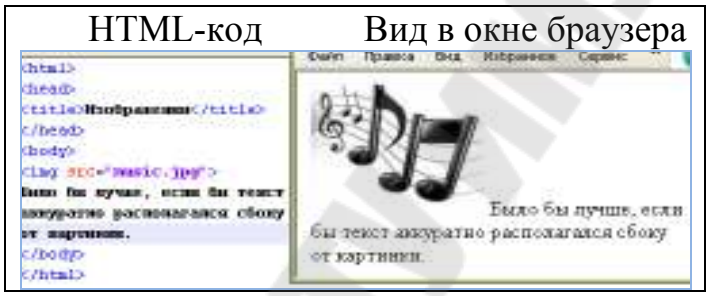



*Рисунок 2.2 – Расположение изображений относительно текста*

Как же сделать так, чтобы текст располагался весь рядом с изображением, а не только одна его строчка? В этом случае поможет атрибут align: <img src="pr1.png" **align**="**left**"> - это означает, что картинка будет прижата к левому краю экрана, а текст будет обтекать ее справа. Чтобы сделать наоборот (картинка справа, текст слева) надо прописать: <img src="pr1.png" **align**="**right**">.

Атрибут align у картинок может принимать и другие значения. Текст может располагаться внизу картинки (по умолчанию) - <img src="pr1.png" **align**="**bottom**">, посередине - <img src="pr1.png" **align**="**middle**"> , и вверху - <img src="pr1.png" **align**="**top**"> .

Кроме параметра align существует еще несколько атрибутов:

- атрибут *vspace*. Например, <img src="pr1.png" vspace="10"> - задает расстояние между текстом и рисунком (по вертикали).
- параметр *hspace*. Например, <img src="pr1.png" hspace="30"> - тоже задает расстояние между текстом и рисунок (по горизонтали).

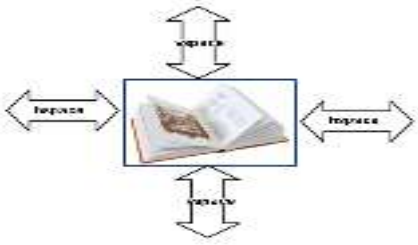


*Рисунок 2.3 – Графическое представление атрибутов hspace и vspace*

Изображение можно использовать не только как элементы, но и как заливку в качестве фона. В современной верстке картинки являются фактически основой для визуального оформления страницы. Никаких оформительских элементов с помощью img на страницах нет. Поэтому фоновые картинки имеют очень важный момент. Для применения картинок в качестве фона используются специальный атрибут background элемента body:

<body **background** =”путь_к_файлу”>.

Заливка фоновой картинки начинается с левого верхнего угла, и начинает повторяться по двум осям, по оси x и по y.

## 2.5 Навигационные карты

Уже говорилось об изображениях и о том, как изображение сделать ссылкой. Пользователи интернета знают, что можно сделать так, чтобы при нажатии на разные области (части) одного и того же изображения, можно было попадать на разные страницы, это называется – *навигационная карта*.

Навигационные карты задаются элементом <**map**>…</**map**>. Тэг <map> включает в себя тэг(и) <area>, которые определяют геометрические области внутри карты и ссылки, связанные с каждой областью (т.е. то куда вы попадете после нажатия). Конструкция элементов, при создании кликабельных областей может быть следующей:

```
<map>
   <area ...>
   <area ...>
   ...
   <area ...>
</map>
```

Кликабельные области бывают трех различных форм:

**Rect** – прямоугольная форма, координаты задаются в пикселях и записываются через запятую x1, y1, x2, y2. (x1,y1) – координаты левого верхнего угла изображения и (x2,y2) – правый нижний угол.

**Circle** – форма окружности, через запятую задаются координаты центра окружности (x,y) и величина радиуса R.

**Poly** – многоугольник, координаты задаются последовательно для каждой точки. Заканчиваются той же точкой из которой вышли.

Рассмотрим эти области подробнее.

*Прямоугольники.* Для начала нам нужно изображение. Возьмем изображение, представленное на рис. 2.4.

*Рисунок 2.4 – Прямоугольная клиткабельная область*

Картинка простая - всего лишь серый квадрат. Представьте, что черный прямоугольник, нарисованный на нем, - это изображение какой-то кнопки, а серое - какой-то сложный фон. Чтобы не создавать сложную таблицу, вы решили не резать картинку на много частей и не вычленять кнопки, Вы решили поступить проще - создать навигационную карту, где область прямоугольника будет ссылкой.

Итак, геометрические области и то, куда пользователь попадет при нажатии на них, определяет тэг <area>, естественно, при помощи своих параметров. Это параметры shape и coords.

Параметр shape - определяет форму области (будет ли она прямоугольником (shape="rect"), кругом (shape="circle") или многоугольником (shape="poly")). Сейчас работаем с прямоугольной областью, поэтому:

```
<map>
   <area shape="rect">
</map>
```

Параметр coords - определяет координаты (положение нашей геометрической формы). Число координат и порядок их значений зависят от выбранной нами формы (см. рисунок 2.5).

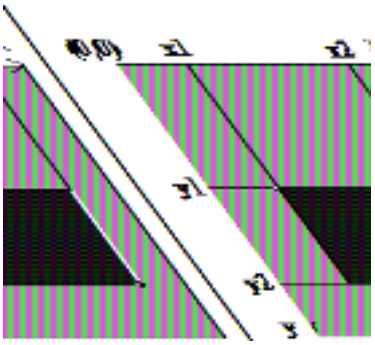


*Рисунок 2.5 – Определение координат прямоугольной области*

Отсчет ведется от левого верхнего угла картинки, считайте его началом координат (0;0). Если работаем с прямоугольной областью, то нужны координаты верхнего-левого и нижнего-правого углов области. Порядок записи координат для параметра coords следующий: <area shape="rect" **coords**="**x1,y1,x2,y2**">.

В нашем примере у прямоугольника координаты такие: x1=83, y1=122, x2=177, y2=185.

Значит, код будет выглядеть следующим образом:

```
<map>
   <area shape="rect" coords="83,122,177,185">
</map>
```

Теперь пропишем, куда будет ссылаться наша область, для этого нам понадобится уже знакомый параметр href:

```
<map>
   <area href="drugoy_document.html" shape="rect"
coords="83,122,177,185">
</map>
```

Однако этого все еще не достаточно, чтобы картинка стала ссылкой, нужно еще указать имя карты и связать ее с картинкой.

У тэга <map> есть параметр name - имя карты, назовем карту - karta1:

```
<map name="karta1">
   <area href="drugoy_document.html" shape="rect"
coords="83,122,177,185">
</map>
```

Для того, чтобы связать карту с картинкой, надо использовать атрибут usemap="#имя_карты" для картинки:

```
<img src="blackrect.gif" usemap="#karta1">
```

*…текст…*

```
<map name="karta1">
   <area href="drugoy_document.html" shape="rect"
coords="83,122,177,185">
</map>
```

*Круги.* Для создания круглой области нужны будут координаты ее центра (x, y) и длинна радиуса R в пикселях. Порядок записи следующий: <area shape="circle" coords="x, y, R">.

Работать будем с геометрической областью, изображенной на рис. 2.6.

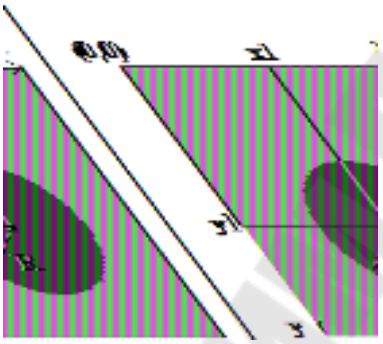


*Рисунок 2.6 – Округлая клиткабельная область*

В нашем случае координаты для круга будут такие: x=46, y=48; а длинна радиуса - R=35. Запишем:

```
<map>
   <area shape="circle" coords="46, 48, 35">
</map>
```

Теперь, когда самое главное записано, пропишем имя карты, куда она ссылается и привяжем карту к рисунку:

```
<img src="mapcircle.gif" usemap="#karta2">
...текст....
<map name="karta2">
   <area href="drugoy_document_2.html" shape="circle" coords="46, 48,
35">
</map>
```

Для карты можно прописать alt для каждой области:

```
<img src="mapcircle.gif" usemap="#karta2">
...текст....
<map name="karta2">
   <area href="drugoy_document_2.html" shape="circle" coords="46, 48,
35" alt="круг">
</map>
```

Теперь при наведении на область будет всплывать подсказка.

*Многоугольники.* Указывая точки (координаты углов), они как бы соединяются, и можно получить очень разнообразные фигуры (рис. 2.7). Используя значение атрибута **shape** poly, можно делать самые разнообразные области, от скромного треугольника до шикарной звезды.

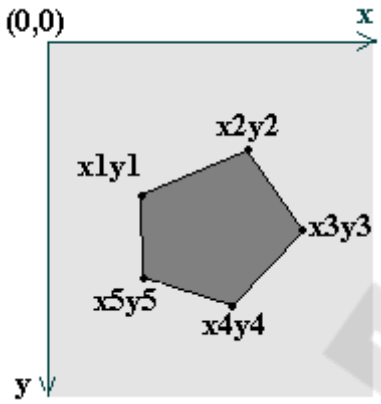



*Рисунок 2.7 – Определение координат многоугольной области*

Зададим тип области:

```
<map>
   <area shape="poly">
</map>
```

Координаты пишутся по следующему принципу: <area shape="poly" **coords="x1,y1,x2,y2,...,xN,yN**">.

Расшифровывается это так: "координаты первого угла (x1,y1), координаты второго угла (x2,y2), еще много углов и их координат (...), координаты последнего угла (xN,yN)". Т.е. для пятиугольника запись полностью будет выглядеть так: <area shape="poly" **coords="x1,y1,x2,y2,x3,y3,x4,y4,x5,y5"**>.

Теперь подставим реальные значения координат в код:

```
<map>
   <area shape="poly" coords="168,9,232,29,200,97,223,129,153,119">
</map>
```

Дальше уже прописываем ссылку, имя карты, и привязываем карту к рисунку (это везде по одному и тому же принципу):

```
<img src="mappoly.gif" usemap="#karta3">
...текст....
<map name="karta3">
   <area href="drugoy_document_3.html" shape="poly"
coords="168,9,232,29,200,97,223,129,153,119">
</map>
```

При создании кликабельных областей могут возникнуть некоторые нюансы:

1. Можно одновременно использовать разные области, например круг и многоугольник:

```
<img src="mappoly.gif" usemap="#karta3">
...текст....
<map name="karta3">
  <area href="drugoy_document_3.html" shape="circle" coords="46,48,35,">
  <area href="drugoy_document_3.html" shape="poly" cords = "168, 9, 232, 29, 200, 97, 223, 129, 153, 119" >
</map>
```

2. Области могут пересекаться. В этом случае при нажатии на область пересечения приоритет имеет область, которая указана первой (т.е. пользователь пойдет на страницу, куда она ссылается). Используя две перекрывающиеся окружности можно создать фигуру, изображенную на рисунке 2.8.

*Рисунок 2.8 – Перекрывающиеся окружности*

```
<img src="combo_map.jpg" border="0" usemap="#my_map4" />
<map name="my_map4">
        <area href="index.html" shape="circle" coords="111, 107, 40" alt="Две окружности" />
        <area href="index.html" shape="circle" coords="75, 76, 20" alt="Две окружности" />
</map>
```

3. Что бы не мучиться с картами, можно найти на просторах интернета специальную программу, с помощью которой без труда можно расчертить карту и не прописывать все вручную. Это может сэкономить время.

4. Существует еще одна область – default – оставшаяся часть, которая не попала в области, описанные с помощью элемента <area>, но тоже

может стать кликабельной. Атрибут default прописывается в конце всех элементов <area> и не поддерживается в Internet Explorer.

5. Иногда возникнет ситуация, когда необходимо вычесть одну область из другой или, другими словами, сделать в области «отверстие».

*Рисунок 2.9 – Некликабельная круглая область в центре*

Элемент nohref позволяет сделать часть области некликабельной.

Например, сделаем кольцо внутри, которого будет некликабельная область.

<area shape=”circle” cords=”50, 30, 15” nohref>

<area shape=”circle” cords=”50, 30, 35” href=http://yandex.ru title=”Привет”>

## 2.6 Вопросы для самоконтроля

1. Какие существуют типы ссылок?
2. Опишите атрибуты элемента <a>…</a>.
3. Как сделать картинку ссылкой?
4. Какие Вы знаете типы файлов иллюстраций.
5. Опишите атрибуты элемента <img>.
6. Что такое навигационные карты и для чего они используются?
7. Какие теги используются для создания кликабельных областей?
8. Опишите основные виды кликабельных областей.
9. Для чего нужны элементы default и nohref?

# 3 Списки и таблицы

## 3.1 Структурирование информации на web-странице при помощи списков. Типы списков

Язык HTML предлагает несколько механизмов создания списков. В каждом списке должен быть один или несколько элементов списков. Списки могут содержать:

- неупорядоченную информацию;
- упорядоченную информацию;
- определения.

*Неупорядоченные списки* создаются с помощью элемента **<ul>…</ul>**, внутри которого может находиться не менее одного элемента <li>…</li>:

```
<ul>
   <li>первый элемент;</li>
   <li>второй элемент;</li>
   <li>третий элемент.</li>
</ul>
```

Для того, чтобы управлять внешним видом маркера, необходимо применить атрибут **type** со значением “circle” или “square”.

Например, <ul **type**=”**circle**”> - маркер в виде окружности,

<li **type**=”**square**”> - маркер в виде квадрата.

На рис. 3.1 приведен пример HTML-кода неупорядоченного списка с разными видами маркера.

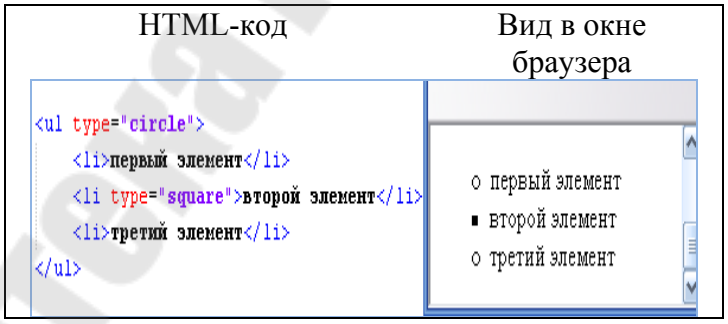



*Рисунок 3.1 – Неупорядоченный список*

*Упорядоченный список*, создаваемый с помощью элемента **<ol>…</ol>**, может содержать информацию, в которой важен порядок, например, рецепт:

1. Тщательно смешать сухие ингредиенты.
2. Влить жидкость.
3. Смешивать 10 минут.

4. Выпекать в течение часа при температуре 300 градусов.

Реализация в HTML упорядоченного списка представлена на рисунке 3.2.

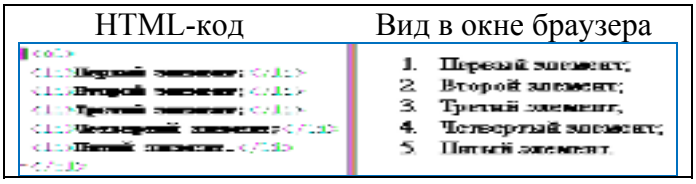



*Рисунок 3.2 – Упорядоченный список*

Если необходимо начать нумерацию не с первого номера, а, например, с 5, то используйте атрибут **start** со значение “5” (рис.3.3).

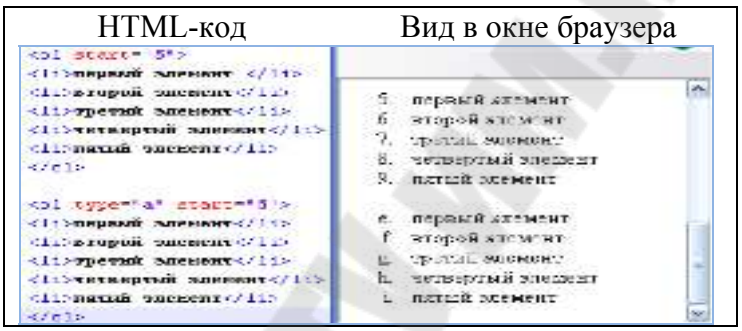



*Рисунок 3.3 – Упорядоченные списки*

Для управления внешним видом нумерованного списка, необходимо применить атрибут **type** со значениями, приведенными в таблице 3.1.

**Таблица 3.1**- Типы нумераций упорядоченного списка

| **Начальный тег** | **Вид номера на экране** |
|---|---|
| <ol> | Нумерация выполняется арабскими цифрами (1, 2, 3, …) |
| <ol type=“1”> | Нумерация выполняется арабскими цифрами (1, 2, 3, …) |
| <ol type=“A”> | Нумерация выполняется прописными буквами (A, B, C, …) |
| <ol type=“a”> | Нумерация выполняется строчными буквами (a, b, c, …) |
| <ol type=“I”> | Нумерация выполняется большими римскими цифрами (I, II, III, …) |
| <ol type=“i”> | Нумерация выполняется малыми римскими цифрами (i, ii, iii, …) |
| <ol type=“5”> | Что получится? |

*Списки определений*, создаваемые с помощью элемента **<dl>…</dl>**, могут содержать ряд пар термин/определение (хотя списки определений могут иметь и иные применения). Определение состоит из термина, определенного тегом <dt>…</dt>, и его описания, определенного тегом <dd>…</dd>.

Все эти элементы – блочные. Внутри элемента dt может стаять только строчный контент. Внутри элемента <dd>…</dd> можно ставить как строчные, так и блочные элементы. Элемент <dd>…</dd> визуально автоматически делает отступ с левой стороны на 40 пикселей, тем самый он отделяет термин от описания. Внутри элемента <dl>…</dl> элементы <dt>…</dt> и <dd>…</dd> могут стоять в любом порядке и в любом количестве. Других элементов в элементе <dl>…</dl> быть не может.

Например, список определений можно использовать в рекламе изделия:

**Низкая цена**

Новая модель этого изделия существенно дешевле предыдущей!

**Проще работа**

Мы изменили изделие, так что с ним теперь легко работать!

**Безопасно для детей**

Вы можете оставить своих детей в комнате, и изделие не причинит им вреда (не гарантируется).

На языке HTML он определяется следующим образом:

<**dl**>
<**dt**>Низкая цена</dt>
<**dd**> Новая модель этого изделия существенно дешевле предыдущей!</dd>
<**dt**>Проще работа</dt>
<**dd**>Мы изменили изделие, так что с ним теперь легко работать!</dd>
<**dt**>Безопасно для детей </dt>
<**dd**> Вы можете оставить своих детей в комнате, и изделие не причинит им вреда (не гарантируется).</dd>
</**dl**>

Списки могут быть *вложенными* (смешанными). Разные типы списков можно использовать вместе, как в следующем примере (рисунок 3.3), где список определений содержит неупорядоченный список (новость дня) и упорядоченный список (новость ночи).


*Рисунок 3.4 – Смешанный список*

Не стоит использовать списки для создания отступов в тексте. Это делается с помощью каскадных таблиц стилей (CSS, см. главу 5).

## 3.2 Таблица и ее элементы

При создании сайтов, таблицы используются очень часто. Используя таблицы, можно создавать такие эффекты, как верстка в несколько колонок, применение эффектов состыковки картинки и фона, тонкие линии на всю ширину или высоту странички и т.д.

В устройстве таблицы легче разобраться на примере (рис. 3.5).

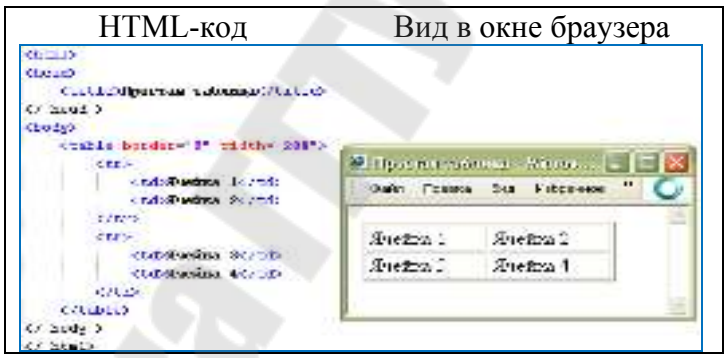



*Рисунок 3.5 – Простая таблица*

В данном примере на рис. 3.5 создана таблица с фиксированной шириной (width="200" пикселей), но лучше использовать проценты, т.к. в этом случае размер таблицы будет изменяться в зависимости от размера окна браузера.

Таблица начинается открывающимся тегом **<table>** и завершается закрывающимся </**table**>. В примере на рис. 3.5 таблица из двух строк и двух столбцов, вместе образующих 4 ячейки. Границы таблицы заданы border="2".

Рассмотрим создание простой таблицы поэтапно.

Сначала зададим две строки таблицы.

<table>

```
        <tr></tr>
        <tr></tr>
</table>
```

Теперь в каждой строке зададим по два столбца (ячейки):

```
<table>
        <tr>
                <td>…</td>
                <td>…</td>
        </tr>
        <tr>
                <td>…</td>
                <td>…</td>
        </tr>
</table>
```

Для начала рекомендуем нарисовать желаемую таблицу на листе бумаге, чтобы все наглядно видеть.

## 3.3 Правила задания атрибутов для таблицы и ее ячеек. Объединение ячеек

Тег **<table>** может включать следующие атрибуты:

***width*** **-** определяет ширину таблицы в пикселях или процентах, по умолчанию ширина таблицы определяется содержимым ячеек.

***border*** **-** устанавливает толщину рамки. По умолчанию таблица рисуется без рамки.

***bordercolor*** **-** устанавливает цвет окантовки.

***bgcolor*** **-** устанавливает цвет фона для всей таблицы.

***background*** **-** заполняет фон таблицы изображением.

***cellspacing*** **-** определяет расстояние между рамками ячеек таблицы в пикселях.

***cellpadding*** **-** определяет расстояние в пикселях между рамкой ячейки и текстом.

***align*** **-** определяет расположение таблицы в документе. По умолчанию таблица прижата к левому краю страницы. Допустимые значения атрибута align: *left* (слева), *center* (по центру страницы) и *right* (справа).

***frame*** - управляет внешней окантовкой таблицы, может принимать следующие значения:

*void* - окантовки нет (значение по умолчанию).
*above* - только граница сверху.
*below* - только граница снизу.
*hsides* - границы сверху и снизу.

*vsides* - только границы слева и справа.
*lhs* - только левая граница.
*rhs* - только правая граница.
*box* - рисуются все четыре стороны.
*border* - также все четыре стороны.

***rules -*** управляет линиями, разделяющими ячейки таблицы. Возможные значения атрибута rules:

*none* - нет линий (значение по умолчанию).
*groups* - линии будут только между группами рядов.
*rows* - только между рядами.
*cols* - только между колонками.
*all* - между всеми рядами и колонками.

Таблица может включать заголовок, который располагается между тегами <**caption**>…</**caption**>. Он должен быть непосредственно после тега <table>. К заголовку возможно применение атрибута **align**, определяющего его положение относительно таблицы со следующими значениями:

*top* - значение по умолчанию, заголовок над таблицей по центру.
*left* - заголовок над таблицей слева.
*right* - заголовок над таблицей справа.
*bottom* - заголовок под таблицей по центру.

Строки таблицы начинаются открывающимся тэгом <**tr**> и завершаются закрывающимся </**tr**>, а каждая ячейка таблицы начинается тегом <**td**> и завершается </**td**>. Данные теги могут иметь такие атрибуты:

***align*** - устанавливает горизонтальное выравнивание текста в ячейках строки. Может принимать значения:

*left* – выравнивание влево;
*center* – выравнивание по центру;
*right* – выравнивание вправо.

***valign*** - устанавливает вертикальное выравнивание текста в ячейках строки. Принимает допустимые значения: *top* – выравнивание по верхнему краю;

*center* – выравнивание по центру (по умолчанию);
*bottom* – выравнивание по нижнему краю.

***Bgcolor*** - устанавливает цвет фона строки или ячейки.
***background*** - заполняет фон строки или ячейки изображением.

Следующие атрибуты могут применяться только *для ячеек*.
***width*** - определяет ширину ячейки в пикселях.
***height*** - определяет высоту ячейки в пикселях.

***nowrap*** - присутствие этого атрибута показывает, что текст должен размещаться в одну строку.

***background*** - заполняет фон ячейки изображением.

Кроме этого, любая ячейка таблицы может быть определена не тегами **<td>…</td>**, а тегами **<th>…</th>.** Текст внутри тегов **<th>…</th>** будет выделен полужирным шрифтом и отцентрирован. Однако, в настоящее время, тег <th> уже устарел, поскольку для работы с оформлением текста применяется CSS.

Если ячейка пустая, то вокруг нее рамка не рисуется. Если рамка все же нужна вокруг пустой ячейки, то в нее надо ввести символьный объект ** ** (от англ. non-breaking space, неразрывающий пробел). Ячейка по-прежнему будет пуста, но рамка вокруг нее будет (  - обязательно должен набираться строчными буквами и закрываться точкой с запятой).

Теги, устанавливающие шрифт (<b>, <i>, <font size="значение">, <font color="значение">), необходимо повторять для каждой ячейки.

Подробнее остановимся на атрибутах colspan и rowspan.

***Colspan*** - определяет количество столбцов, на которые простирается данная ячейка, а ***rowspan*** - количество рядов. Эти параметры могут принимать значение от 2 и более, т.е. наша ячейка может растягиваться на два и более столбца (ряда). Теперь, обратимся к примерам.

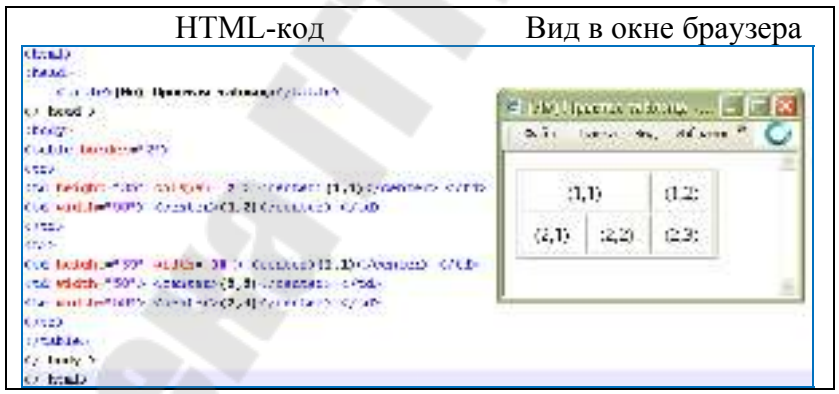



*Рисунок 3.6 – Использование атрибута colspan*

В таблице на рис. 3.6 использован параметр colspan = 2, прописанный для ячейки (1,1), где первая цифра - это номер ряда, а вторая - номер столбца (т.е. (1,2) - первый ряд, второй столбец и т.д.).

Обратите внимание, на то, что параметр width для ячейки (1,1) в примере на рис. 3.6 не указан. Если необходимо задать этот атрибут, то в данном примере для ячейки (1,1) его надо было бы прописать равным 100 пикселям, т.к. все-таки ячейка (1,1) длиннее других в два раза.

И второе на что следует обратить внимание, в рассмотренном примере на рис. 3.6 нет ячейки (1,3), т.е. в первом ряду всего лишь две

ячейки, т.к. ячейка (1,1) равна сама по себе двум ячейкам по длине (что мы и указали параметром colspan). Если вы прописали ячейку (1,3), тогда у вас получилась бы такая «таблица», как на рис. 3.7.

| | | |
|---|---|---|
| (1,1) | (1,2) | (1,3) |
| (2,1) | (2,2) | (2,3) |

*Рисунок 3.7 – Ошибочное построение таблицы*

Теперь, рассмотрим параметр rowspan. Принцип действия тут тот же. Попробуйте самостоятельно прописать код для таблицы, что представлена на рис. 3.8.

| | | |
|---|---|---|
| (1,1) | (1,2) | (1,3) |
| (2,1) | (2,2) | |

*Рисунок 3.8 – Использование атрибута rowspan*

## 3.4 Верстка при помощи таблиц. Вложенные таблицы

Изначально, назначение тега <table> сводилось к размещению на странице структурированных данных. На рис. 3.9 приведен пример классического применения структуры <table>.

Хронология фильмов Гарри Поттер

| | | |
|---|---|---|
| 1 | Гарри Поттер и философский камень | 2001 |
| 2 | Гарри Поттер и Тайная Комната | 2004 |
| 3 | Гарри Поттер и Узник Азкабана | 2004 |
| 4 | Гарри Поттер и Кубок огня | 2005 |
| 5 | Гарри Поттер и Орден Феникса | 2007 |
| 6 | Гарри Поттер и Принц Полукровка | 2009 |
| 7 | Гарри Поттер и Дары смерти. Часть 1 | 2010 |
| 8 | Гарри Поттер и Дары смерти. Часть 2 | 2011 |

*Рисунок 3.9 – Пример использования таблицы*

По мере усложнения сайтов этот тег превратился в инструмент для разметки каркаса web-страниц. Способ верстки, использующий тег <table> называется *табличный*. При помощи этого тега можно быстро построить достаточно сложный каркас страницы.

Построение каркаса страницы (верстка) осуществляется путем вложения таблиц друг в друга. Сначала создается основная таблица, затем, в ее ячейках создаются другие таблицы, в которых в свою очередь, также

можно создать таблицы. Таким образом, можно построить достаточно сложный каркас. Приведем пример создания каркаса страницы (для краткости приведем только содержание раздела <body>).

**Пример 3.1** – Вложенные таблицы

```
<table border="0" cellspacing="0" cellpadding="0" width="100%">
    <tr>
        <td>Блок заголовка</td>
    </tr>
    <tr>
        <td>
            <table width="100%">
                <tr>
                    <td width="15%">Меню</td>
                    <td>Контент</td>
                </tr>
            </table>
        </td>
    </tr>
    <tr>
        <td>Нижний блок</td>
    </tr>
</table>
```

Таким образом, создана страница, состоящая из четырех блоков: блока заголовка, блока меню, блока контента и нижнего блока.

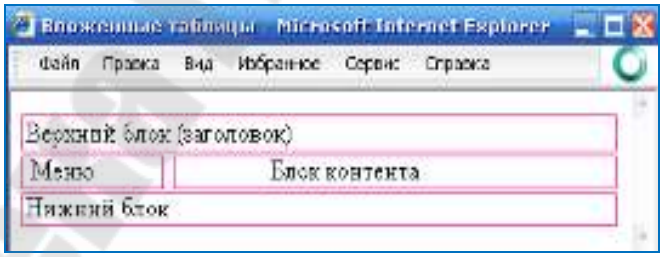



*Рисунок 3.10 – Пример табличной верстки*

Каждый из созданных в примере 3.1 блоков, может быть также разбит на блоки при помощи таблиц.

Рассмотрим еще один вид верстки. *Комбинированной*, называется верстка с одновременным использованием тегов-контейнеров и конструкции <table>. Например, код страницы, которая была создана при помощи табличной верстки в примере 3.1, можно упростить, используя комбинацию тегов  и конструкции <table>.

**Пример 3.2** – Комбинированная верстка

```
<div>Блок заголовка</div>
```

```
<div>
    <table width="100%">
        <tr>
            <td width="15%">Меню</td>
            <td>Контент</td>
        </tr>
    </table>
</div>
<div>Нижний блок</div>
```

В результате будет достигнут тот же результат (см. рисунок 3.10), но с меньшими затратами.

## 3.5 Приемы использования таблиц на web-странице

Благодаря вложенным таблицам гораздо проще управлять содержимым страницы сайта.

Создайте с помощью одной большой таблицы разметку своей страницы. Теперь у вас возникла необходимость поместить в уже размеченную страницу, например, подменю сайта. В этом вам поможет еще одна таблица, вложенная в ячейку исходной таблицы (выделено прямоугольником в левой части рис. 3.11).

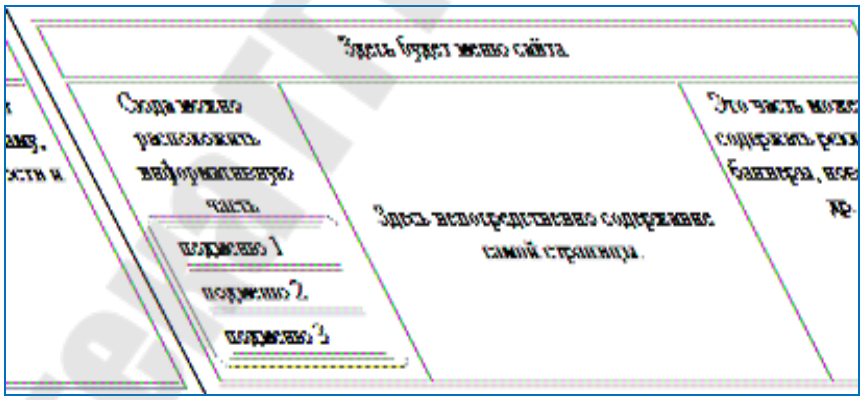


*Рисунок 3.11 – Макетирование сайта с помощью вложенных таблиц*

В примере 3.3 приведен код таблицы, представленной на рис. 3.11.

**Пример 3.3** – Код макета сайта

```
<table border="1" width="600" height="200" align="center">
    <tr>
        <td colspan="3" valign="top" height="30"><Center>Здесь
        будет меню сайта</Center></td>
    </tr>
    <tr align="center">
```

```
<td valign="top" width="150" height="170">Сюда можно
расположить информативную часть <br>
    <table border="1" width="120">
        <tr>
            <td>подменю 1</td>
        </tr>
        <tr>
            <td>подменю 2</td>
        </tr>
        <tr>
            <td>подменю 3</td>
        </tr>
    </table>
</td>
<td width="300">Здесь непосредственно содержание
самой страницы.</td>
<td valign="top" width="150">Это часть может содержать
рекламу, баннеры, новости и др.</td>
</tr>
</table>
```

Вложенная таблица ни чем не отличается от простой таблицы. Для начала выбирается ячейка, куда будет вкладываться таблица, затем пишется код нужной таблицы.

В данном примере использовались атрибуты width и height как для самой таблицы, так и для ее ячеек, размеры заданы в пикселях. Ширина самой таблицы 600 пикселей, высота - 200. Для ячеек задана ширина 150, 300, 150, т.е. ширина всех ячеек в сумме получается 600, а высота ячеек была задана 30 и 170, в итоге их сумма равна высоте всей таблицы. А для вложенной таблицы ширина самой таблицы задана 120 пикселей.

Механизм HTML дает возможность создания таблиц разной сложности. Можно создать таблицу со многими вложенными таблицами, раскрасить их в разные цвета радуги и т.д. все это не сложно.

Сложность будет заключаться в том, чтобы то, что вы сверстали выглядело бы под разными браузерами одинаково. Опытные web мастера стараются просматривать сверстанную ими страницу под разными типами браузеров, так как у каждого браузера есть свои недостатки и капризы. Например, вы создали таблицу для разметки своей страницы и решили вложить в нее вложенную таблицу, фон которой должен отличаться от фона основной таблицы. Конечно, можно использовать атрибут bgcolor, но не все так просто, если Internet Explorer отображает атрибут bgcolor для вложенных таблиц, то некоторые другие браузеры просто отказываются отображать этот атрибут для вложенных таблиц, поэтому приходится

изыскивать другие пути, используя атрибут background для вложенной таблицы.

На примере это будет выглядеть как на рис. 3.12.

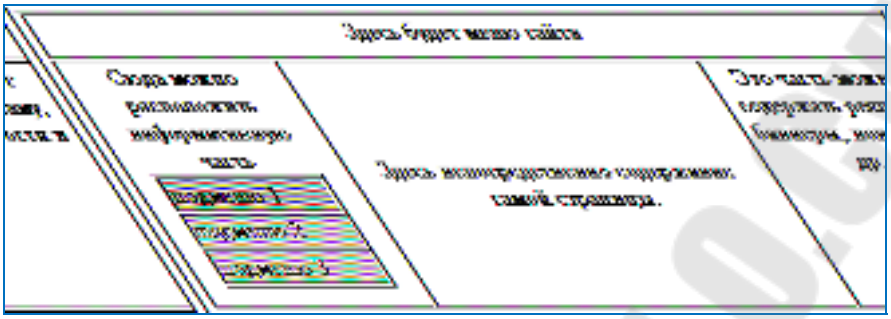


*Рисунок 3.12 – Использование атрибута background для вложенной таблицы*

Фон ячеек вложенной таблицы задан атрибутом **background** со значением Fon1.jpg, Fon2.jpg и Fon3.jpg соответственно для каждой ячейки подменю. В качестве картинки использован квадрат 1x1 пикселя определенного цвета.

Итак, пробуем создать web-страницу. Для начала пишем код страницы и таблицы, с помощью которой мы делаем разметку этой страницы. Размер таблицы можно задать в процентах относительно окна браузера: width="100%".

Следующим шагом будет наполнение страницы содержимым, вставка вложенных таблиц, удаление рамок вокруг таблиц. В итоге может получиться web-страничка, представленная на рис. 3.13:

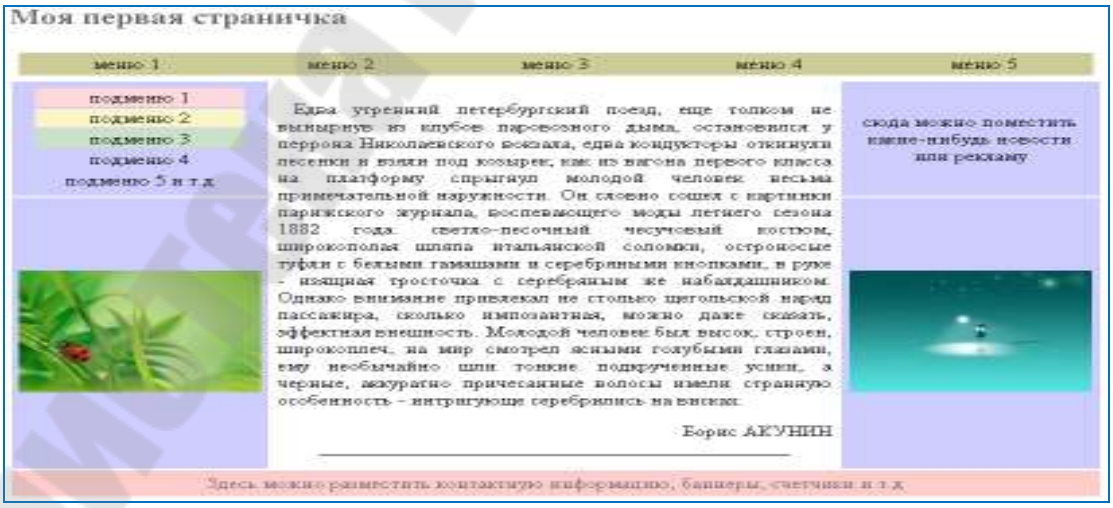



*Рисунок 3.13 – Пример создания web странички с помощью табличной верстки*

## 3.6 Преимущества и недостатки табличной верстки

Табличная верстка в настоящее время не пользуется популярностью. Многие ресурсы перешли на верстку с помощью слоев. Однако есть и те, кто по-прежнему отдает предпочтение табличной верстке. Попытаемся объективно разобраться, что же представляют собой таблицы, где их следует применять, а где не стоит. Сразу следует оговорить, что спор вокруг таблиц происходит только в том случае, когда они используются для верстки. Если дело касается хранения табличных данных, то не возникает сомнений, что таблицы именно для этой цели и предназначены.

***Преимущества таблиц***

Таблицы предлагали достаточно простые методы для размещения разных элементов на web-странице и, при отсутствии конкурентов, довольно долго применялись для верстки. Таблица выступает в роли невидимой модульной сетки, относительно которой добавляется текст, изображения и другие элементы. Далее рассмотрены плюсы таблиц.

1. Возможность *создания колонок.* Таблицы хорошо выступают в качестве многоколонной модульной сетки, каждая ячейка представляет собой отдельную колонку. Это позволяет легко создавать двух- и трехколончатый макет документа (см. рис. 3.13). При изменении размера окна браузера, колонки сохраняют свой исходный вид, а не переносятся как слои друг под друга. К тому же высота разных колонок при использовании таблиц остается одинаковой, независимо от объема их содержимого.

2. *Создание «резинового» макета.* Таблицы удачно подходят для «резинового» макета, ширина которого привязана к ширине окна браузера. Благодаря тому, что размер таблицы можно задавать в процентах, она занимает все отведенное ей свободное пространство. Также можно регулировать и высоту содержимого. Например, если текста немного, то «подвал» страницы может висеть в ее середине. Параметрами таблицы можно отрегулировать это так, что при небольшом тексте «подвал» плотно прилегает к нижнему краю окна браузера, независимо от размеров окна.

3. *«Склейка» изображений.* Рисунки часто разрезают на отдельные фрагменты, а затем собирают их вновь в одно целое, выкидывая одни фрагменты или заменяя их другими изображениями. Это требуется для различных дизайнерских изысков вроде создания эффекта перекатывания, анимации или уменьшения объема файлов. Таблицы позволяют легко обеспечить «склейку» нескольких рисунков в одно изображение. Каждая картинка помещается в определенную ячейку, параметры таблицы при этом устанавливаются такими, чтобы не возникло стыков между отдельными ячейками.

4. Создание *фоновых рисунков.* В ячейки таблицы разрешается добавлять фоновый рисунок, в зависимости от размеров ячейки он может повторяться по горизонтали, вертикали или сразу в двух направлениях. За счет этого приема на странице создаются декоративные линии, рамки самого разнообразного вида, добавляется тень под элементом.

5. *Выравнивание элементов.* Содержимое ячеек можно одновременно выравнивать по горизонтали и по вертикали, за счет чего расширяются возможности по размещению элементов относительно друг друга и на странице в целом.

6. *Особенности браузеров.* Браузеры достаточно вольно толкуют некоторые параметры CSS, поэтому создание универсального кода с применением слоев может стать настоящей головной болью для разработчиков. В этом смысле таблицы отображаются в разных браузерах практически одинаково, поэтому создание web-страниц упрощается.

***Недостатки таблиц***

Несмотря на описанные достоинства таблиц, у них есть и определенные недочеты, которые порой заставляют искать другие способы верстки.

*1. Долгая загрузка.* Если таблица велика по высоте, может пройти достаточно много времени, прежде чем мы увидим нужную информацию. Существуют и способы обхода этого свойства, в частности, разбиение одной большой таблицы на несколько таблиц поменьше, а также использование стилевого свойства table-layout.

*2. Слишком большой код.* Таблицы содержат сложную иерархическую структуру вложенных тегов. В некоторых случаях для достижения желаемого результата приходится вкладывать одну таблицу внутрь другой, а это также влияет на размер кода, который не принимает непосредственного участия в отображении web-страницы.

*3. Плохая индексация поисковиками.* За счет того, что текст располагается в отдельных ячейках таблицы, в коде он может находиться достаточно далеко друг от друга. Такая раздробленность информации, а также значительная вложенность тегов затрудняет правильное индексирование страницы поисковыми системами. Как результат документ не попадает в первую десятку выдачи запроса по ключевым словам, хотя вполне может и заслуживать это.

*4. Нет разделения содержимого и оформления.* В идеале HTML-код должен содержать только теги с указанием стилевого класса или идентификатора. А все оформление вроде цвета текста и положения элемента выносится в CSS и модифицируется отдельно. Такое разделение позволяет независимо править код страницы и менять вид отдельных ее элементов. Хотя к таблицам стиль легко добавляется, но обилие «лишних»

тегов не позволяет действительно просто и удобно управлять видом отдельных компонентов страницы. К тому же не все параметры таблиц имеют свой стилевой синоним, поэтому в любом случае приходится обращаться к коду web-страницы и править его.

*5. Несоответствие стандартам.* В последнее время стандарты HTML и CSS прочно засели в умах web-разработчиков. Этому способствует развитие XHTML и XML, которые более «жестко» относятся к коду документа, появление новых версий браузеров, придерживающихся спецификации, и мода на верстку слоями.

Что же говорит спецификация относительно таблиц? А говорит она, что таблицы в первую и последнюю очередь нужны для размещения табличных данных. Все остальные способы использования таблиц осуждаются.

## 3.7 Вопросы для самоконтроля

1. Какие Вы знаете типы списков?
2. Элементы ol, ul, dd и их атрибуты.
3. Как управлять видом маркеров в неупорядоченном списке?
4. Элемент table и его атрибуты.
5. Для чего нужны и как правильно использовать атрибуты colspan и rowspan?
6. Основные приемы создания вложенных таблиц?
7. Что такое табличная верстка? Как она используется на web-страницах?
8. Какие Вы знаете недостатки табличной верстки?

# 4 Фреймы и форм

## 4.1 Фреймы и их описание на языке HTML. Задание логики взаимодействия фреймов

Существуют различные приложения, не обязательно web-страницы, где окно разделено на несколько областей, в каждую из которых загружен какой-то документ (см. рис. 4.1, 4.2). Причем эти области ведут себя независимо.

*Фреймы* – способ организации структуры приложения, при котором оно дробится на ряд отдельных составляющих и «собирается» в главном окне из нескольких независимых или вложенных окон. Слово «фрейм» можно понимать как слово «окно».

При таком представлении каждый компонент страницы является самостоятельным документом и встраивается в ту область экрана, которая задается тегом **<frameset>**.


*Рисунок 4.1 - Пример создания web странички с помощью фреймов*

*Рисунок 4.2 - Пример создания учебника с помощью фреймов*

Несмотря на то, что сайты с фреймами встречаются все реже, изучение HTML было бы неполным без рассмотрения темы о фреймах. К тому же фреймы в каком-то смысле заняли свою нишу и применяются для систем администрирования и справки. Там, где недостатки фреймов не имеют особого значения, а преимущества наоборот, активно востребованы.

Для создания фрейма используется тег <frameset>, который заменяет тег <body> в документе и применяется для разделения экрана на области. Внутри данного тега находятся теги <frame>, которые указывают на HTML-документ, предназначенный для загрузки в область (рис. 4.3).


*Рисунок 4.3 - Пример разделения окна браузера на два фрейма*

При использовании фреймов необходимо как минимум три HTML-файла: первый определяет фреймовую структуру и делит окно браузера на две части, а оставшиеся два документа загружаются в заданные окна. Количество фреймов не обязательно равно двум, может быть и больше, но никак не меньше двух, иначе вообще теряется смысл применения фреймов.

Рассмотрим этапы создания фреймов на основе страницы, продемонстрированной на рис. 4.3. Понадобится три файла: index.html — определяет структуру документа, menu.html — загружается в левый фрейм и content.html — загружается в правый фрейм. Из них только index.html отличается по структуре своего кода от других файлов (пример 4.1).

**Пример 4.1** – Файл index.html

```
<html>
        <head>
                <title>Фреймы</title>
        </head>
 <frameset cols="100,*">
        <frame src="menu.html" name="MENU">
        <frame src="content.html" name="CONTENT">
 </frameset>
</html>
```

В данном примере 4.1 окно браузера разбивается на две колонки с помощью атрибута cols, левая колонка занимает 100 пикселов, а правая —

оставшееся пространство, заданное символом звездочки. Ширину или высоту фреймов можно также задавать в процентном отношении, наподобие таблиц.

В теге <frame> задается имя HTML-файла, загружаемого в указанную область с помощью атрибута src. В левое окно будет загружен файл, названный menu.html (пример 4.2), а в правое — content.html (пример 4.3). Каждому фрейму желательно задать его уникальное имя, чтобы документы можно было загружать в указанное окно с помощью атрибута name.

**Пример 4.2 -** Файл menu.html

```
<html>
        <head>
            <title>Меню сайта</title>
        </head>
<body>
        <p>МЕНЮ</p>
 </body>
</html>
```

**Пример 4.3** – Файл content.html

```
<html>
        <head>
            <title>Содержание сайта</title>
        </head>
<body>
        <p>СОДЕРЖАНИЕ</p>
</body>
</html>
```

Рассмотрим более сложный пример уже с тремя фреймами (рис. 4.4).

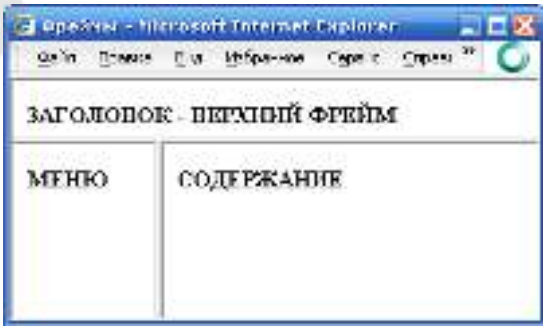



*Рисунок 4.4 - Разделение страницы на три фрейма*

В данном случае опять используется тег <frameset>, но два раза, причем один тег вкладывается в другой. Горизонтальное разбиение создается через атрибут rows, где для разнообразия применяется процентная запись (пример 4.4).

**Пример 4.4 -** Три фрейма

```
<html>
    <head>
        <title>Фреймы</title>
    </head>
    <frameset rows="25%,75%">
        <frame src="top.html" name="TOP" scrolling="no" noresize>
        <frameset cols="100,*">
            <frame src="menu.html" name="MENU">
            <frame src="content.html" name="CONTENT">
        </frameset>
    </frameset>
</html>
```

Как видно из примера 4.4, контейнер <frameset> с атрибутом rows вначале создает два горизонтальных фрейма, но вместо второго фрейма подставляется еще один <frameset>, который повторяет уже известную структуру из примера 4.1. Чтобы не появилась вертикальная полоса прокрутки, и пользователь не мог самостоятельно изменить размер верхнего фрейма, добавлены атрибуты scrolling="no" и noresize.

В обычном HTML-документе при переходе по ссылке, в окне браузера текущий документ заменяется новым. При использовании фреймов схема загрузки документов отличается от стандартной. Основное отличие — возможность загружать документ в выбранный фрейм из другого фрейма. Для этой цели используется атрибут target тега <a>. В качестве значения используется имя фрейма, в который будет загружаться документ, указанный атрибутом name (пример 4.5).

**Пример 4.5 –** Ссылка на другой фрейм

```
<html>
    <head>
        <title>Фреймы</title>
    </head>
<frameset cols="100,*">
    <frame src="menu2.html" name="MENU">
    <frame src="content.html" name="CONTENT">
</frameset>
</html>
```

В приведенном примере фрейму присваивается имя CONTENT. Чтобы документ загружался в указанный фрейм, используется конструкция target="CONTENT", как показано в примере 4.6.

**Пример 4.6 –** Содержимое файла menu2.html

```
<html>
    <head>
        <title>Навигация по сайту</title>
    </head>
<body>
    <p>МЕНЮ</p>
    <p><a href="text.html" target="CONTENT">Текст</a></p>
</body>
</html>
```

Имя фрейма должно начинаться с цифры или латинской буквы. В качестве зарезервированных имен используются следующие:

*_blank* – загружает документ в новое окно;

*_self* – загружает документ в текущий фрейм;

*_parent* – загружает документ во фрейм, занимаемый родителем, если фрейма-родителя нет значение действует также, как top;

*_top* – отменяет все фреймы и загружает документ в полное окно браузера.

Граница между фреймами отображается по умолчанию и, как правило, в виде трехмерной линии. Чтобы ее скрыть используется атрибут frameborder тега <frameset> со значением 0. Однако в браузере Opera граница хоть и становится в этом случае бледной, все же остается. Для этого браузера требуется добавить **framespacing**="0". Таким образом, комбинируя разные атрибуты тега <frameset>, получим универсальный код, который работает во всех браузерах. Линия при этом показываться никак не будет (пример 4.6).

**Пример 4.6 –** Убираем границу между фреймами

```
<html>
    <head>
        <title>Фреймы</title>
    </head>
<frameset cols="100,*" frameborder="0" framespacing="0">
    <frame src="menu.html" name="MENU">
    <frame src="content.html" name="CONTENT">
</frameset>
</html>
```

Следует отметить, что атрибуты frameborder и framespacing не являются валидными и не соответствуют спецификации HTML.

Если граница между фреймами все же нужна, в браузере она рисуется по умолчанию, без задания каких-либо атрибутов. Можно, также, задать цвет рамки с помощью атрибута **bordercolor**, который может применяться в тегах <frameset> и <frame>. Цвет указывается по его названию или шестнадцатеричному значению (пример 4.7), а толщина линии управляется атрибутом border. Браузер Opera игнорирует этот атрибут и обычно отображает линию черного цвета.

**Пример 4.7 –** Изменение цвета границы

```
<html>
    <head>
        <title>Фреймы</title>
    </head>
<frameset cols="100,*" bordercolor="#000080" border="5">
    <frame src="menu.html" name="MENU">
    <frame src="content.html" name="CONTENT">
</frameset>
</html>
```

Атрибуты bordercolor и border тега <frameset> также не являются валидными и не признаются спецификацией HTML.

В данном примере линия между фреймами задается синего цвета толщиной пять пикселов. Линии различается по своему виду в разных браузерах, несмотря на одинаковые параметры (рис. 4.5).

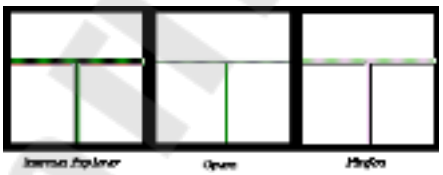


*Рисунок 4.5 - Вид границы между фреймами в разных браузерах*

Браузер Opera никак не изменяет цвет границы между фреймами, Internet Explorer устанавливает широкую границу практически сплошного цвета, а Firefox границу отображает в виде набора линий.

По умолчанию размеры фреймов можно изменять с помощью курсора мыши, наведя его на границу между фреймами. Для блокировки возможности изменения пользователем размера фреймов следует воспользоваться атрибутом **noresize** тега <frame> (пример 4.8).

**Пример 4.8 –** Запрет на изменение размера фреймов

```
<html>
    <head>
        <title>Фреймы</title>
    </head>
 <frameset cols="100,*">
```

```
        <frame src="menu.html" name="MENU" noresize>
        <frame src="content.html" name="CONTENT">
 </frameset>
</html>
```

Атрибут noresize не требует никаких значений и используется сам по себе. Для случая двух фреймов этот атрибут можно указать лишь в одном месте. Естественно, если у одного фрейма нельзя изменять размеры, то у близлежащего к нему размеры тоже меняться не будут.

Если содержимое фрейма не помещается в отведенное окно, автоматически появляются полосы прокрутки для просмотра информации. В некоторых случаях полосы прокрутки нарушают дизайн web-страницы, поэтому от них можно отказаться. Для управления отображением полос прокрутки используется атрибут **scrolling** тега <frame>. Он может принимать два основных значения: yes – всегда вызывает появление полос прокрутки, независимо от объема информации и no – запрещает их появление (пример 4.9).

**Пример 4.9 –** Полоса прокрутки

```
<html>
        <head>
                <title>Фреймы</title>
        </head>
<frameset cols="100,*">
        <frame src="menu.html" name="MENU" noresize scrolling="no">
        <frame src="content.html" name="CONTENT">
</frameset>
</html>
```

При выключенных полосах прокрутки, если информация не помещается в окно фрейма, просмотреть ее будет сложно. Поэтому scrolling="no" следует использовать осторожно.

Если атрибут scrolling не указан, то полосы прокрутки добавляются браузером только по необходимости, в том случае, когда содержимое фрейма превышает его видимую часть.

Разговор о фреймах будет неполным без упоминания плавающих фреймов. Так называется фрейм, который можно добавлять в любое место web-страницы. Еще одно его название – встроенный фрейм, он называется так из-за своей особенности встраиваться прямо в тело web-страницы. На рис. 4.6 приведен пример такого фрейма.

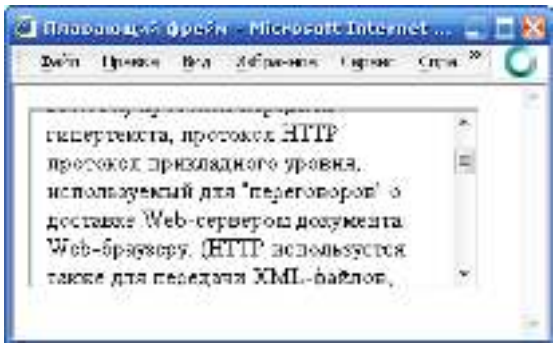



*Рисунок 4.6 - Плавающий фрейм на web-странице*

Во фрейм можно загружать HTML-документ и прокручивать его содержимое независимо от остального материала на web-странице. Размеры фрейма устанавливаются самостоятельно согласно дизайну сайта или собственных предпочтений.

Создание плавающего фрейма происходит с помощью тега <iframe>, он имеет обязательный атрибут src, указывающий на загружаемый во фрейм документ (пример 4.10).

**Пример 4.10 -** Использование тега <iframe>

```
<html>
    <head>
        <title>Плавающий фрейм</title>
    </head>
<body>
    <p><iframe src="text.html" width="300" height="120"></iframe></p>
</body>
</html>
```

В данном примере ширина и высота фрейма устанавливается через атрибуты width и height. Сам загружаемый во фрейм файл называется text.html. Заметьте, что если содержимое не помещается целиком в отведенную область, появляются полосы прокрутки.

Еще одно удобство плавающих фреймов состоит в том, что в него можно загружать документы по ссылке. Для этого требуется задать имя фрейма через атрибут name, а в теге <a> указать это же имя в атрибуте target (пример 4.11).

**Пример 4.11 –** Загрузка документа во фрейм

```
<html>
    <head>
        <title>Плавающий фрейм</title>
    </head>
<body>
    <p><a href="rgb.html" target="color">RGB</a>
```

```
<a href="cmyk.html" target="color">CMYK</a>
<a href="hsb.html" target="color">HSB</a></p>
<p><iframe src="model.html" name="color" width="100%" height="200"></iframe></p>
</body>
</html>
```

В данном примере добавлено несколько ссылок, они открываются во фрейме с именем color.

## 4.2 Достоинства и недостатки фреймовой структуры

Поскольку вокруг фреймов существует много разговоров об их необходимости, далее приведем достоинства и недостатки фреймов, чтобы можно было самостоятельно решить стоит ли их использовать на своем сайте.

***Достоинства фреймовой структуры***:

*Простота*

Разграничить web-страницу на две части можно с помощью фреймов. Обычно одна из частей содержит навигацию по сайту, а другая его контент. Механизм фреймов позволяет открывать документ в одном фрейме, по ссылке, нажатой в совершенно другом фрейме. Такое разделение web-страницы на составляющие интуитивно понятно и логически обусловлено.

*Быстрота*

Размещение на одной странице и навигации и содержания увеличивает объем каждой страницы и в сумме может существенно повлиять на объем загружаемой с сайта информации. А так как фреймы используют разделение информации на части, страницы с ними будут загружаться быстрее.

*Размещение*

Фреймы предоставляют возможность размещать информацию точно в нужном месте окна браузера.

*Изменение размеров областей*

Можно изменять размеры фреймов «на лету», чего не позволяет сделать традиционная верстка HTML.

*Загрузка*

Загрузка web-страницы происходит только в указанное окно, остальные остаются неизменными. С помощью языка JavaScript можно осуществить одновременную загрузку двух и более страниц во фреймы.

***Недостатки фреймовой структуры***:

*Навигация*

Пользователь зачастую оказывается на сайте, совершенно не представляя, куда он попал, потому что всего лишь нажал на ссылку, полученную в поисковой системе. Чтобы посетителю сайта было проще разобраться, где он находится, на каждую страницу помещают название сайта, заголовок страницы и навигацию. Фреймы, как правило, нарушают данный принцип, отделяя заголовок сайта от содержания, а навигацию от контента. Представьте, что вы нашли подходящую ссылку в поисковой системе, нажимаете на нее, а в итоге открывается документ без названия и навигации. Чтобы понять, где мы находимся или посмотреть другие материалы, придется редактировать путь в адресной строке, что в любом случае доставляет неудобство.

*Плохая индексация поисковыми системами*

Поисковые системы плохо работают с фреймовой структурой, поскольку на страницах, которые содержат контент, нет ссылок на другие документы.

*Внутренние страницы нельзя добавить в «Закладки»*

Фреймы скрывают адрес страницы, на которой находится посетитель, и всегда показывают только адрес сайта. По этой причине понравившуюся страницу сложно поместить в закладки браузера.

*Несовместимость с разными браузерами*

Параметры фреймов обладают свойством совершенно по разному отображаться в различных браузерах. Причём противоречие между ними настолько явное, что одни и те же параметры интерпретируются браузерами совершенно по-своему.

*Непрестижность*

Этот недостаток носит скорее идеологический характер. Сайты с фреймами считаются несолидными, а их авторы сразу выпадают из разряда профессионалов, которые никогда не используют фреймы в своих работах. Исключение составляют чаты, где без фреймов обойтись можно, но достаточно хитрыми методами, а с помощью фреймов создавать чаты достаточно просто.

Надо отметить, что некоторые приведённые недостатки вполне обходятся. Так, с помощью скриптов можно сделать, что открытый в браузере отдельный документ формируется со всей фреймовой структурой. Поисковые системы также уже лучше индексируют фреймовые документы, чем это было несколько лет назад.

## 4.3 Форма и ее элементы. Методы отправки информации из полей формы

На различных сайтах можно встретить элементы, позволяющие передать куда-либо информацию и что-то получать в ответ. Подобный принцип реализован в многочисленных электронных конференциях,

досках объявлений, поисковых системах, сайтах требующих авторизации или регистрации и т.д. На языке HTML единственное, что позволит вводить, обрабатывать и передавать информацию – это формы.

*Формой* называется объединение логически связанных элементов управления в HTML-документе, с помощью которых осуществляется ввод, обработка и передача данных от HTML-документа интерактивным элементам сайта, например сценариям. Поместив в форму какие-либо значения, посетитель сервера нажимает на соответствующую кнопку, после чего введенная им информация передается скрипту, который принимает управление процессом обработки данных.

Формы образуются с помощью элемента **<form>…</form>**. Этот элемент блочный, внутри него можно размещать любой контент (хоть блочный, хоть строчный) кроме других форм.

На рис. 4.7 представлены основные элементы формы.

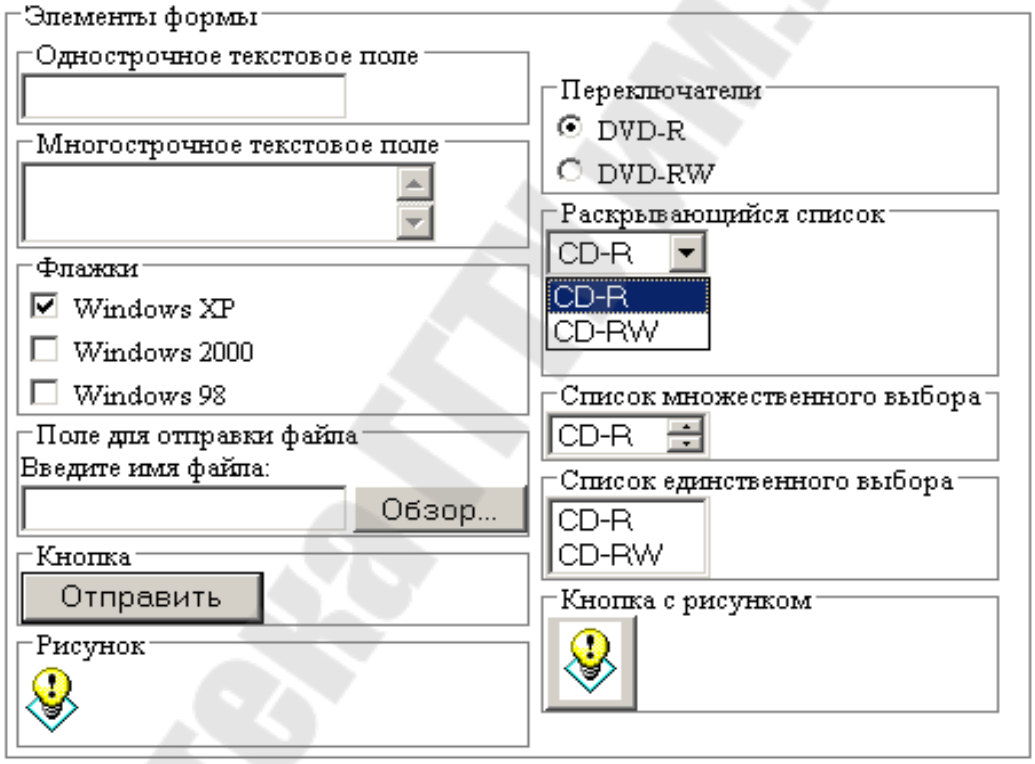



*Рисунок 4.7 –Основные элементы формы*

Формат задания формы следующий:

<**form name**="имя_формы" **action**="URL" **enctype**="тип_кодировки" **method**="метод_передачи_данных" **target**="значение">

Содержание формы

</**form**>

В качестве параметра атрибута **action** в кавычках указывается строка вызова скрипта, который использует данная форма (при нажатии кнопки),

например, "http://www.gstu.by/имя_сценария.php". Если в качестве значения атрибута **action** указать адрес электронной почты, например, **action**=”mailto:gstu@gstu.by”, тогда браузер автоматически отправит результаты, введенные в форму, по указанному адресу. Для корректной интерпретации данных используется параметр **enctype**=”text/plain”.

Если же атрибут **action** отсутствует, то в качестве значения **action** подставляется URL самого документа. В этом случае, после отправки формы текущая страница перезагружается, возвращая все элементы формы к их значениям по умолчанию.

Значение атрибута **method** устанавливает метод передачи данных из формы на сервер: "GET" с помощью стандартного интерфейса HTTP или "POST" – по каналам электронной почты.

Передача данных происходит в один этап адресной строкой при задании значения "GET". Пары «имя=значение» присоединяются в этом случае к адресу после вопросительного знака и разделяются между собой амперсандом (&). Метод GET используется для передачи данных с небольших форм с короткими полями. При задании значения "POST" передача данных происходит, как минимум, в два этапа: браузер устанавливает связь с сервером, указанным в атрибуте action; затем отдельной передачей происходит посылка дополнительных данных. Переданные данные не отображаются в командной строке. Метод POST используется для передачи данных форм, имеющих много длинных полей.

Атрибут **enctype** устанавливает тип для данных, отправляемых вместе с формой. Обычно не требуется определять значения параметра **enctype**, однако если используется поле для отправки файла (input type=file), то следует задать параметр enctype =”multipart/form-data”.

Атрибут **target** определяет окно, в которое будет загружаться итоговая web-страница. Значения этого атрибута такие же, как и при задании фреймовой структуры.

Содержание формы описывается тегом <**input**>, запись которого в общем виде следующая:

<input **type**="тип_элемента" **name**="имя" **value**="строка" **checked** **size**="целое_число" **maxlength**="целое_число" **align**="значение" **scr**="URL" **tabindex**="значение">

Атрибуты тега <input>:

**type** – задает тип элемента формы;

**name** – задает уникальное имя для каждого элемента формы;

**value** – указывает первоначальное значение текущего поля;

**checked** – устанавливает выделенный объект из нескольких в случае, если значением атрибута **type** является radio или checkbox;

**size** – определяет размер текстового поля в символах;

**maxlength** – определяет максимально возможную длину текстового поля в символах для полей ввода текста;
**align** – определяет положение элементов формы на web-странице;
**src** – используется совместно с атрибутом **type**=image и задает URL нужного изображения;
**tabindex** – позволяет установить порядок перемещения фокуса по элементам формы при нажатии клавиши Tab.

*Однострочным текстовым полем* называется поле ввода или поле редактирования, которое
предназначено для ввода пользователем строки текста.

Формат записи:
<input **type**="text" **name**="имя_поля" **value**="начальный текст, содержащийся в поле" **size**="ширина поля" **maxlength**="максимальное количество вводимых символов">

Например, <input type=text size=40 name=user_name value=”Введите Ваше имя”>

*Поле для ввода пароля* – это обычное текстовое поле, вводимый текст в котором отображается звездочками.

Формат задания:
<input **type**=password **name**="имя_поля" **value**="начальный текст, содержащийся в поле" **size**="ширина поля" **maxlength**="максимальное количество вводимых символов">

*Скрытые поля* не отображаются на странице и встраиваются в HTML-файл, когда необходимо передать серверу техническую информацию. Скрытые поля служат доступной альтернативой файлам *cookies* – специальным файлам, в которых сохраняются индивидуальные настройки пользователя и позволяющим, например, восстановить последнее состояние формы при повторном посещении пользователем содержащей эту форму страницы.

Формат задания:
<input **type**=hidden **name**="имя_поля" **value**="текст, содержащийся в поле">

Например, <input type=hidden name="form1" value="d3375-535-8412">

*Многострочные текстовые поля* используются для передачи текста большого размера.

Формат записи:

<**textarea name**="имя элемента" **rows**="целое число" **cols**="целое число" **wrap**=значение **disabled readonly**>

Текст по умолчанию

</**textarea**>

Атрибуты тега <textarea>:

**rows** и **cols** – указывают соответственно максимально допустимое количество строк вводимого текста и символов в строке. В случае, если набираемый пользователем текст не умещается в видимую часть текстового контейнера, по краям поля появляются вертикальные и горизонтальные полосы прокрутки.

**wrap** – управляет переносом слов и имеет следующие значения:

*off* – запрет автоматического переноса; при этом сохраняются переносы, определенные пользователем;

*virtual* – перенос слов при отображении браузером, а серверу введенные данные передаются одной строкой;

*physical* – сохраняется перенос слов как при отображении браузером, так и при передаче серверу.

**disabled** – блокирует доступ и изменение текстового поля. Поле отображается серым цветом, недоступно для активации пользователем, и не может получить фокус. Состояние этого поля можно изменять с помощью скриптов.

**readonly** – текстовое поле недоступно для изменения пользователем, в него не допускается вводить новый текст или модифицировать существующий, оно не может получить фокус. Поле отображается обычным цветом.

Например,

```
<textarea name="message" rows=25 cols=40>
        Введите текст сообщения
</textarea>
```

*Поле выбора файлов* создает на экране кнопку, при нажатии на которую появляется Проводник Windows, позволяющий присоединить к отсылаемым на сервер данным любой файл с локального компьютера пользователя (рис.4.8). Рядом с кнопкой отображается небольшое текстовое поле, куда автоматически заносится имя отсылаемого файла и путь к нему на локальном диске. Поле выбора файлов работает корректно лишь при методе пересылки Post и формате кодировки multipart/form-data.

Формат задания:

<input **type**=file **name**="имя" **size**="ширина поля" **maxlength**="максимальная длина текста">

**Пример 4.12** – Выбор файла

```
<html>
```

```
        <head>
            <title>Выбор файла</title>
        </head>
<body>
        <form enctype="multipart/form-data" action="URL"
method=POST>
            Введите имя файла:<br>
            <input type=file name=myfile> <br>
            <input type=submit value="Отправить">
        </form>
</body>
</html>
```


*Рисунок 4.8 – Поле выбора файла*

*Флажки* используют, когда необходимо выбрать два или более варианта из предложенного списка (рис. 4.9). Элемент представляет собой простую форму выбора, принимающую одно из двух состояний: «отмечено» – «не отмечено». Несколько флажков могут объединяться в группу, которая будет отвечать набору параметров выбора. Данный элемент оперирует с булевыми переменными**,** то есть переменными, каждая из которых может принимать значение true или false.

Формат задания:

<input **type**=checkbox **name**="имя флажка или группы флажков" **value**="значение становленного флажка" **checked title**="всплывающая подсказка">

По умолчанию начальное положение флажка считается не установленным. Чтобы задать начальное положение установленного флажка, надо дополнить его атрибутом **checked**.

Значением установленного флажка является строка, заданная атрибутом **value**.

**Пример 4.13 –** Задание группы флажков

```
<input type=checkbox name=system value="WXP" checked>Windows
XP<br>
<input type=checkbox name=system value="W2000">Windows
2000<br>
<input type=checkbox name=system value="W98">Windows 98<br>
```

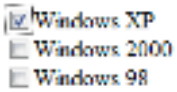


*Рисунок 4.9 – Флажки*

В рассмотренном примере 4.12 надписи рядом с флажками созданы как простой текст. Для того чтобы флажок устанавливался не только щелчком непосредственно по квадратику флажка, но и щелчком по надписи (рис. 4.10), необходимо связать надпись с флажком с помощью тега label, в котором содержится ссылка на связанный элемент управления с помощью атрибута for. Этому атрибуту ставится в соответствие идентификатор id.

**Пример 4.14** – Связывание надписи с флажком

<input type=checkbox **id**=WindowsXP name=system value="WXP" checked>

<**label for**=WindowsXP>Windows XP</label><br>

<input type=checkbox **id**=Windows2000 name=system value="W2000">

<**label for**=Windows2000> Windows 2000</label><br>

<input type=checkbox **id**=Windows98 name=system value="W98">

<**label for**=Windows98> Windows 98</label><br>

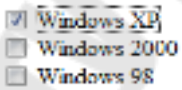



*Рисунок 4.10 – Связывание надписи с флажком*

*Переключатели (кнопки выбора или радио-кнопки)* применяется в случае, когда какая-либо логическая переменная может принимать только одно значение из множества возможных.

Формат задания:

<input **type**=radio **name**="имя переключателя или группы" **value**="значение установленного переключателя" **checked title**="всплывающая подсказка">

Все элементы **radio** одной группы обозначаются одним и тем же значением атрибута **name**. Использование радио-кнопок требует явного указания значений атрибута **value**, одна из кнопок должна быть выделена атрибутом **checked**. Если атрибут **checked** не присвоен ни одному из переключателей группы, браузер при загрузке установит по умолчанию первый переключатель. При обработке формы на сервере будет отправлено значение установленного переключателя.

Например:

<input type=radio name=DVD value="DVDR" checked> DVD-R<br>

<input type=radio name=DVD value="DVDRW" checked> DVD-RW<br>

Как и в случае флажков, надписи можно связать с соответствующими переключателями так, чтобы каждый переключатель устанавливался при щелчке по надписи. Для связывания каждому переключателю должен быть присвоен уникальный идентификатор, а все переключатели должны образовывать группу с определенным именем.

**Пример 4.15 -** Связывание надписи с переключателем:

```
<input type=radio id=disk1 name=DVD value="DVDR" checked>
<label for=disk1>DVD-R</label><br>
<input type=radio id=disk2 name=DVD value="DVDRW" checked>
<label for=disk2> DVD-RW</label><br>
```

*Кнопки* - это элементы управления, которые используются для представления формы (кнопка submit), сброса данных формы (кнопка reset), создания эффектов для кнопки (кнопка button).

Кнопку можно создать двумя способами:

1. Использование тега <input>.

Формат задания:

<input **type**=button **name**="имя кнопки" **value**="надпись на кнопке">

2. Использование тега <button>. На таких кнопках можно размещать любые элементы HTML, в том числе изображения и таблицы и изменять вид кнопки.

Формат задания:

```
<button>
        Надпись или изображение
</button>
```

В HTML предусмотрены два типа кнопок, которые создаются без использования значения button. Это кнопки специального назначения:

Подача запроса (submit) и сброс (reset).

Кнопка **submit** предназначена для запуска процедуры передачи формы на сервер.

Формат задания:

<input **type**=submit **name**="имя кнопки" **value**="надпись на кнопке">

*или*

<button **type**=submit> Надпись на кнопке </button>

Если атрибут value отсутствует, то кнопка по умолчанию имеет надпись «Подача запроса». В версиях Internet Explorer 4.0 и выше кнопка submit может работать как кнопка по умолчанию, то есть она активизируется при нажатии клавиши Enter. В форме можно применять несколько кнопок submit.

Кнопка **reset** предназначена для приведения формы в начальное положение (сброс всех введенных данных).

Формат задания:

<input **type**=reset **name**="имя кнопки"  **value**="надпись на кнопке">
*или*
<button **type**=reset> Надпись на кнопке </button>

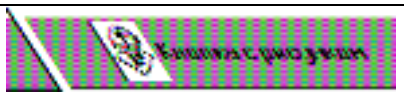
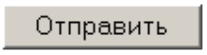
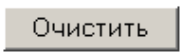
Если атрибут value отсутствует, то кнопка по умолчанию имеет надпись «Сброс».

*Кнопка с изображение* создает кнопку отсылки, аналогичную элементу **submit**, но с использованием графического изображения. Обычно применяется в случаях, когда стандартная серая прямоугольная кнопка «не вписывается» в дизайн сайта.

Формат записи:
<input **type**=image **name**="имя кнопки" **scr**="URL изображения" **value**="надпись на кнопке">

В этом случае тег <input> может содержать все атрибуты тега img.

**Таблица 4.1 –** Примеры задания кнопок

| Описание | Вид в окне браузера |
|---|---|
| *1. Обычная кнопка*<br><input type=button name=press value="Нажми меня!"> |  |
| *2. Обычная кнопка*<br><Button><br>Кнопка с текстом<br></Button> |  |
| *3. Кнопка с рисунком*<br><Button><br><img src="tips.gif" align=absmiddle><br>Кнопка с рисунком<br></Button> |  |
| *4. Кнопка submit*<br><input type=submit value="Отправить"> |  |
| *5. Кнопка reset*<br><input type=reset value="Очистить"> |  |
| *6. Кнопка с изображением*<br><input type=image src="tips.gif"> |  |

*Списки* представляют пользователю список вариантов для выбора. Существует три типа списков:

1. *раскрывающийся список*, представляющий собой однострочное поле т треугольной стрелкой, которая раскрывает список;

2. *поле-список*, в котором на экран выводится заданное число строк; для просмотра всех срок список может быть снабжен полосой прокрутки;
3. *список со множественным выбором*, позволяющий благодаря полосе прокрутки просматривать все позиции списка и выбирать одновременно несколько позиций.

Формат записи:

<**select name**="имя списка" **size**="целое число" **multiple**>
    <**option** value="значение" selected>
        Пункт 1
    </option>
</select>

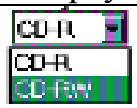
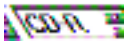
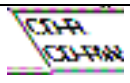
Атрибуты тега <select>:

**multiple –** включает режим выбора нескольких элементов из списка, т.е. определяет список со множественным выбором;

**size –** устанавливает высоту списка. Если значение size=1, то список становится раскрывающимся. При добавлении параметра multiple список отображается как «крутилка»;

**selected –** делает текущий элемент списка выделенным.

**Таблица 4.2** - Примеры задания списка

| Описание | Вид в окне браузера |
|---|---|
| *1. Раскрывающийся список*<br><select name=CD><br><option value=1>CD-R</option><br><option value=2>CD-RW</option><br></select> |  |
| *2. Список множественного выбора*<br><select multiple size=1><br><option value=1>CD-R</option><br><option value=2>CD-RW</option><br></select> |  |
| *3. Список единственного выбора*<br><select size=2><br><option value=1>CD-R</option><br><option value=2>CD-RW</option><br></select> |  |

Чтобы страница имела законченный вид, элементы формы должны быть распределены по *группам*. Формат задания группы:

```
<fieldset>
    <legend>
        Легенда группы элементов
    </legend>
    …
</fieldset>
```

Теги <legend> вводят надпись, которая помещается в разрыв рамки, обрамляющей группу (рис. 4.11).

**Пример 4.16. –** Использование тега <legend>

```
<fieldset>
 <legend>Флажки</legend>
 <input type=checkbox checked>Windows XP<br>
 <input type=checkbox>Windows 2000<br>
 <input type=checkbox>Windows 98<br>
</fieldset>
```


*Рисунок 4.11 – Легенда*

## 4.4 Использование табличных функций при создании форм на примере анкеты

Рассмотрим пример использования структуры таблиц при создании анкеты web-разработчика. Вид формы в окне браузера представлен на рисунке 4.12

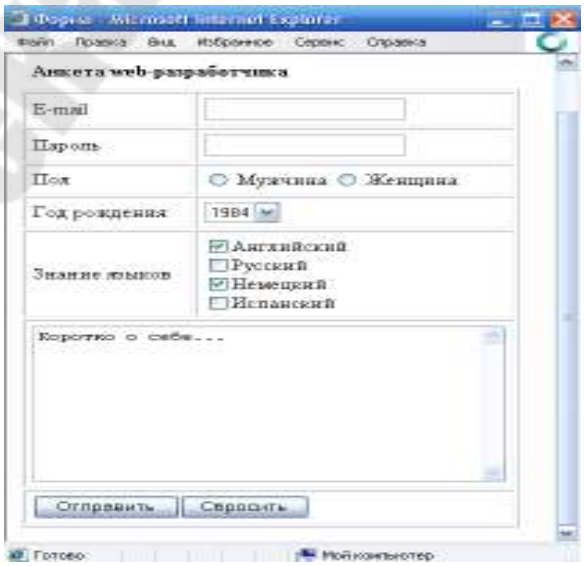



*Рисунок 4.12 – Вид формы в окне браузера*

Далее распишем HTML-код данной формы.

```
<html>
  <head> <title>Форма</title></head>
<body>
<form action="">
        <table border="1" cellpadding="5">
        <caption align="left"><b> Анкета web-разработчика </b></caption>
                <tr>
                        <td>E-mail</td>
                        <td><input type="text" name="email"></td>
                </tr>
                <tr>
                        <td>Пароль</td>
                        <td><input type="password" name="pass"></td>
                </tr>
                <tr>
                        <td>Пол</td>
                        <td>
                        <input type="radio" name="gender" id="g1" value="1">
                        <label for="g1">Мужчина</label>
                        <input type="radio" name="gender" id="g2" value="2">
                        <label for="g2">Женщина</label>
                        </td>
                </tr>
                <tr>
                        <td>Год рождения</td>
                        <td>
                        <select name="year" size="1">
                        <option value="1984" selected>1984</option>
                        <option value="1985">1985</option>
                        <option value="1986">1986</option>
                        <option value="1987">1987</option>
                        <option value="1988">1988</option>
                        <option value="1989">1989</option>
                        </select>
                        </td>
                </tr>
                <tr>
                        <td>Знание языков</td>
                        <td>
                        <input type="checkbox" name="languadg" value="Inglish"
checked>Английский<br>
                        <input type="checkbox" name="languadg" value="Rushen">Русский<br>
                        <input type="checkbox" name="languadg" value="Doich"
checked>Немецкий<br>
                        <input type="checkbox" name="languadg" value="Spain">Испанский<br>
                        </td>
                </tr>
                <tr>
                        <td colspan="2">
                        <textarea name="description" cols="40" rows="10">Коротко о себе...
</textarea>
                        </td>
```

```
                </tr>
                <tr>
                        <td colspan="2">
                        <input type="submit" value="Отправить">
                        <button type="reset">Сбросить</button>
                        </td>
                </tr>
        </table>
</form>
</body>
</html>
```

## 4.5 Вопросы для самоконтроля

1. Что такое фреймы и зачем они нужны?
2. Каким образом создаются фреймы на языке HTML?
3. Перечислите недостатки фреймовой структуры?
4. Что такое форма и для чего она нужна?
5. Перечислите основные элементы формы.
6. Перечислите методы отправки информации из полей формы.
7. Опишите элемент input и его атрибуты (type, name, value, checked, size, maxlength, align, src).
8. Как создать многострочное текстовое поле?
9. Перечислите атрибуты тега <textarea>…</textarea>.
10. Как создать поле выбора файлов?
11. Назовите способы создания флажков и переключателей.
12. Перечислите виды кнопок и опишите форматы их записи.

# 5 Каскадные таблицы стилей (CSS)

## 5.1 Основные цели и задачи CSS. Способы добавления стилей на web-страницу

HTML лишь первый этап в процессе обучения созданию сайтов. Следующим шагом является изучение стилей или CSS (Cascading Style Sheets, каскадные таблицы стилей).

Стандарты листов стилей для web были разработаны консорциумом W3C в 1995-96. Слово «*каскадные*» означает, что листы стилей позволяют создавать иерархию стилевых свойств, согласно которой локальный стиль отменяет глобальный стиль.

*CSS-код* – это список написанных особым образом инструкций для браузера, который «говорит» как и где отображать элементы web-страницы. Другими словами, стили представляют собой набор параметров, управляющих видом и положением элементов web-страницы. Для наглядности рассмотрим примеры.

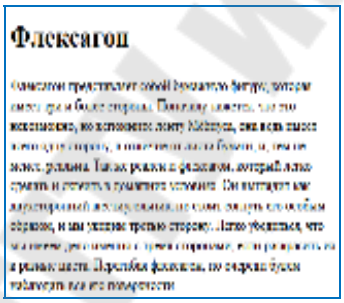



*Рисунок 5.1 – Web-страница, созданная только на HTML*

На рис. 5.1 представлена обычная web-страница, оформленная без всяких изысков. Тот же самый документ, но уже с добавлением стилей приобретает совершенно иной вид (рис. 5.2).

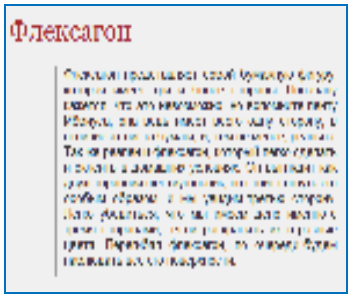



*Рисунок 5.2 – Web-страница, созданная на HTML и CSS*

В самом коде HTML никаких изменений не произошло и единственное добавление – это строка

<link **rel**="stylesheet" **type**="text/css" **href**="**style.css**" >,

которая ссылается на внешний файл с описанием стилей под именем style.css.

В файле style.css как раз и описаны все параметры оформления таких тегов как <body>, <h1> и <p>. Сами теги в коде HTML пишутся как обычно. На файл со стилем можно ссылаться из любого места web-документа, что приводит в итоге к сокращению объема повторяющихся данных. А благодаря разделению кода и оформления повышается гибкость управления видом документа и скорость работы над сайтом.

Для добавления стилей на web-страницу существует несколько способов, которые различаются своими возможностями и назначением. Далее рассмотрим их подробнее.

***Связанные стили***

При использовании связанных стилей описание селекторов и их значений располагается в отдельном файле, как правило, с расширением css, а для связывания документа с этим файлом применяется тег <link>. Данный тег помещается в контейнер <head>, как показано в примере 5.1.

**Пример 5.1** – Подключение связанных стилей

```
<html>
   <head>
      <title>Стили</title>
      <link rel="stylesheet" type="text/css" href="style.css">
      <link rel="stylesheet" type="text/css" href="http://gstu.by/main.css">
   </head>
 <body>
    <h1>Заголовок</h1>
    <p>Текст</p>
 </body>
 </html>
```

Значения атрибутов тега <link> - rel и type остаются неизменными независимо от кода, как приведено в данном примере 5.1. Значение href задает путь к CSS-файлу, он может быть задан как относительно, так и абсолютно. Заметьте, что таким образом можно подключать таблицу стилей, которая находится на другом сайте.

Содержимое файла style.css подключаемого посредством тега <link> приведено в примере 5.2.

**Пример 5.2** – Файл со стилем style.css

```
H1 {
    color: #001180;
    font-size: 200%;
```

```
    font-family: Arial, Verdana, sans-serif;
    text-align: center;
  }
P {
     padding-left: 20px;
  }
```

Как видно из примера 5.2, файл со стилем не хранит никаких данных, кроме синтаксиса CSS. В HTML-документ содержит только ссылку на файл со стилем, т.е. таким способом в полной мере реализуется принцип разделения кода и оформления сайта. Поэтому использование связанных стилей является наиболее универсальным и удобным методом добавления стиля на сайт.

***Глобальные стили***

При использовании глобальных стилей свойства CSS описываются в самом документе и располагаются в заголовке web-страницы. По своей гибкости и возможностям этот способ добавления стиля уступает предыдущему, но также позволяет хранить стили в одном месте, в данном случае прямо на той же странице с помощью контейнера <**style**>, как показано в примере 5.3.

**Пример 5.3** – Использование глобального стиля

```
<html>
  <head>
    <title>Глобальные стили</title>
    <style type="text/css">
      h1 {
          font-size: 120%;
          font-family: Verdana, Arial, Helvetica, sans-serif;
          color: #334466;
           }
    </style>
  </head>
 <body>
    <h1> Здесь любой текст.</h1>
 </body>
</html>
```

В примере 5.3 определен стиль тега <h1>, который затем можно повсеместно использовать на данной web-странице.

***Внутренние стили***

Внутренний или встроенный стиль является расширением для одиночного тега используемого на web-странице. Для определения стиля

используется атрибут **style**, а его значением выступает набор стилевых правил (пример 5.4).

**Пример 5.4** – Использование внутреннего стиля

```
<html>
   <head>
      <title>Внутренние стили</title>
   </head>
 <body>
    <p style="font-size: 120%; font-family: monospace; color: #cd66cc">Пример текста</p>
</body>
</html>
```

В примере 5.4 стиль тега <p> задается с помощью атрибута style, в котором через точку с запятой перечисляются стилевые свойства.

Внутренние стили рекомендуется применять на сайте ограниченно или вообще отказаться от их использования. Дело в том, что добавление таких стилей увеличивает общий объем файлов, что ведет к повышению времени их загрузки в браузере, и усложняет редактирование документов для разработчиков.

Все описанные методы использования CSS могут применяться как самостоятельно, так и в сочетании друг с другом. В этом случае необходимо помнить об их иерархии. Первым всегда применяется внутренний стиль, затем глобальный стиль и в последнюю очередь связанный стиль. В примере 5.5 применяется сразу два метода добавления стиля в документ.

**Пример 5.5** – Сочетание разных методов подключения стилей

```
<html>
   <head>
      <title>Подключение стиля</title>
      <style type="text/css">
            h1 {
               font-size: 120%;
               font-family: Arial, Helvetica, sans-serif;
               color: green;
               }
      </style>
   </head>
 <body>
    <h1 style="font-size: 36px; font-family: Times, serif; color: red">Заголовок 1</h1>
      <h1>Заголовок 2</h1>
```

```
</body>
</html>
```

В примере 5.5 заголовок 1 задается красным цветом размером 36 пикселов с помощью внутреннего стиля, а заголовок 2 задается зеленым цветом через таблицу глобальных стилей.

## 5.2 Основные понятия и определения. Грамматика языка стилей

Как уже было отмечено ранее, стилевые правила записываются в своем формате, отличном от HTML. Основным понятием выступает *селектор* – это некоторое имя стиля, для которого добавляются параметры форматирования. В качестве селектора выступают теги, классы и идентификаторы. Общий способ записи имеет следующий вид:

```
Селектор {
        свойство1: значение;
        свойство2: значение;
        свойство3: значение;
          }
```

Вначале пишется имя селектора, например, h1, это означает, что все стилевые параметры будут применяться к тегу <h1>, затем ставятся фигурные скобки, в которых записывается стилевое свойство, а его значение указывается после двоеточия. Стилевые свойства разделяются между собой точкой с запятой, в конце этот символ тоже ставится.

Форма записи правил зависит от желания разработчика, поскольку CSS не чувствителен к регистру, переносу строк, пробелам и символам табуляции. Так, в примере 5.6 показаны две разновидности оформления селекторов и их правил.

**Пример 5.6** – Использование стилей

```
<html>
  <head>
    <title>Заголовки</title>
    <style type="text/css">
      h1 {
          color: #a6780a;
          font-weight: normal;
           }
      h2 { color: olive; border-bottom: 2px solid black; }
    </style>
  </head>
 <body>
        <h1>Заголовок 1</h1>
        <h2>Заголовок 2</h2>
```

```
 </body>
</html>
```

В данном примере свойства селектора h2 записаны в одну строку, а для селектора h1 каждое свойство находится на отдельной строке. В первом случае легче отыскивать нужные свойства и править их по необходимости, но при этом незначительно возрастает объем данных за счет активного использования пробелов и переносов строк. Так что в любом случае способ оформления стилевых параметров зависит от разработчика.

Далее приведены некоторые правила, которые необходимо знать при описании стиля.

*Правило 1.* Для селектора допускается добавлять каждое стилевое свойство и его значение по отдельности, как это показано в примере 5.7.

**Пример 5.7** – Расширенная форма записи

```
td { background: olive; }
td { color: white; }
td { border: 1px solid black; }
```

Очевидно, что такая запись не очень удобна. Приходится повторять несколько раз один и тот же селектор, да и легко запутаться в их количестве. Поэтому пишите все свойства для каждого селектора вместе. Указанный набор записей в таком случае получит следующий вид (пример 5.8). Такая форма записи более наглядная и удобная в использовании.

**Пример 5.8** – Компактная форма записи

```
td {
     background: olive;
     color: white;
     border: 1px solid black;
   }
```

*Правило 2.* Если для селектора вначале задается свойство с одним значением, а затем то же свойство, но уже с другим значением, то применяться будет то значение, которое в коде установлено ниже (пример 5.9).

**Пример 5.9** – Разные значения у одного свойства

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

В данном примере для селектора p цвет текста вначале задается зеленым, а затем красным. Поскольку значение red расположено ниже, то оно в итоге и будет применяться к тексту.

На самом деле такой записи лучше вообще избегать и удалять повторяющиеся значения. Но подобное может произойти не явно,

например, в случае подключения разных стилевых файлов, в которых содержатся одинаковые селекторы.

*Правило 3.* У каждого свойства может быть только соответствующее его функции значение. Например, для color, который устанавливает цвет текста, в качестве значений недопустимо использовать числа.

Комментарии нужны, чтобы делать пояснения по поводу использования того или иного стилевого свойства, выделять разделы или писать свои заметки. Комментарии позволяют легко вспоминать логику и структуру селекторов, и повышают разборчивость кода. Вместе с тем, добавление текста увеличивает объем документов, что отрицательно сказывается на времени их загрузки. Поэтому комментарии обычно применяют в отладочных или учебных целях, а при выкладывании сайта в сеть их стирают.

Любой CSS-комментарий начинается с конструкции /* и заканчивается конструкцией */.

**Пример 5.10** – Комментарии в CSS-файле

```
/*
Стиль для сайта mysite.by
*/
div {
      width: 200px; /* Ширина блока */
      margin: 10px; /* Поля  вокруг элемента */
      float: left; /* Обтекание по правому краю */
    }
```

Как следует из примера 5.10, комментарии можно добавлять в любое место CSS-документа, а также писать текст комментария в несколько строк. Вложенные комментарии недопустимы.

## 5.3 Создание стилей и классов. Применение стилей и классов на web-странице

В качестве *селектора* может выступать любой тег HTML, для которого определяются правила форматирования, такие как: цвет, фон, размер и т.д. Правила задаются в следующем виде.

**Тег { свойство1: значение; свойство2: значение; ... }**

Вначале указывается имя тега, оформление которого будет переопределено, заглавными или строчными символами не имеет значения. Внутри фигурных скобок пишется стилевое свойство, а после двоеточия – его значение. Набор свойств разделяется между собой точкой с запятой и может располагаться как в одну строку, так и в несколько (пример 5.21).

**Пример 5.11** – Изменение стиля тега абзаца

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-
8">
      <title>Селекторы тегов</title>
      <style type="text/css">
            P {
               text-align: justify; /* Выравнивание по ширине */
               color: green; /* Зеленый цвет текста */
               }
      </style>
   </head>
 <body>
       <p>Ваш текст…</p>
</body>
</html>
```

В данном примере изменяется цвет текста и выравнивание текста абзаца. Стиль будет применяться только к тексту, который располагается внутри контейнера <p>.

Следует понимать, что хотя стиль можно применить к любому тегу, результат будет заметен только для тегов, которые непосредственно отображаются в контейнере <body>.

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

**Тег.Имя класса { свойство1: значение; свойство2: значение; ... }**

Внутри стиля вначале пишется желаемый тег, а затем, через точку пользовательское имя класса. Имена классов должны начинаться с латинского символа и могут содержать в себе символ дефиса (-) и подчеркивания (_). Использование русских букв в именах классов недопустимо. Чтобы указать в коде HTML, что тег используется с определенным классом, к тегу добавляется атрибут class="Имя класса" (пример 5.12).

**Пример 5.12** – Использование классов

```
<html>
   <head>
      <title>Классы</title>
      <style type="text/css">
            P { /* Обычный абзац */
```

```
                text-align: justify; /* Выравнивание текста по ширине
*/
            }
        P.cite { /* Абзац с классом cite */
            color: navy; /* Цвет текста */
            margin-left: 20px; /* Отступ слева */
            border-left: 1px solid navy; /* Граница слева от текста
*/
            padding-left: 15px; /* Расстояние от линии до текста */
            }
    </style>
  </head>
 <body>
    <p>Слово «каскадные» означает, что листы стилей позволяют
создавать
    иерархию стилевых свойств, согласно которой локальный
стиль отменяет глобальный стиль.</p>
    <p class="cite">CSS-код – это список инструкций для браузера,
– как и где
    отображать элементы web-страницы, написанный особым
образом.</p> </body>
</html>
```

Результат примера 5.12 показан на рис. 5.3.

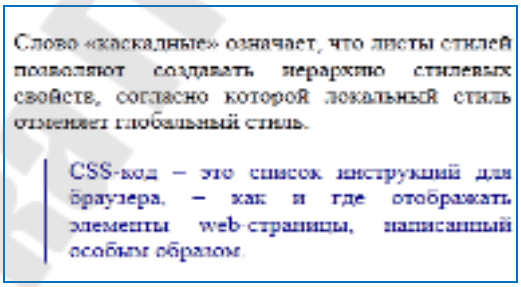



*Рисунок 5.3 - Вид текста, оформленного с помощью стилевых классов*

Первый абзац выровнен по ширине с текстом черного цвета (этот цвет задается браузером по умолчанию), а следующий, к которому применен класс с именем cite — отображается синим цветом и с линией слева.

Можно, также, использовать *классы и без указания тега*. Синтаксис в этом случае будет следующий.

**.Имя класса { свойство1: значение; свойство2: значение; ... }**

При такой записи, класс можно применять к любому тегу (пример 5.13).

**Пример 5.13** – Использование классов

```
<html>
   <head>
      <title>Классы</title>
      <style type="text/css">
            .gost {
                  color: green; /* Цвет текста */
                  font-weight: bold; /* Жирное начертание */
                   }
            .term {
                  border-bottom: 1px dashed red; /* Подчеркивание под текстом */
                  }
      </style>
   </head>
<body>
      <p>Согласно <span class="gost">ГОСТ 12.1.003-83 ССБТ &amp;quot;Шум. Общие
      требования безопасности &amp;quot;</span>, шумовой характеристикой рабочих
      мест при постоянном шуме являются уровни звуковых давлений в децибелах
      в октавных полосах. Совокупность таких уровней называется
      <b class="term">предельным спектром</b>, номер которого численно равен
      уровню звукового давления в октавной полосе со среднегеометрической
      частотой 1000&amp;nbsp;Гц.
</p>
</body>
</html>
```

Результат применения классов к тегам  и <b> показан на рис. 5.4.

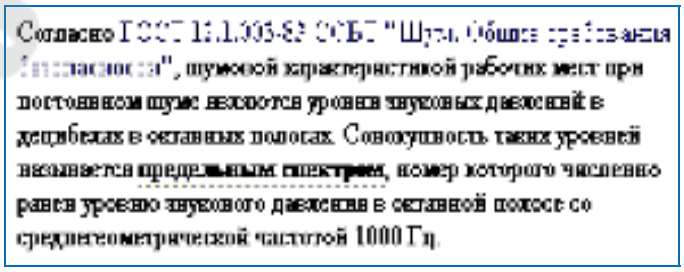



*Рисунок 5.4 - Вид тегов, оформленных с помощью классов*

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

## 5.4 Единицы измерения размеров

Все многообразие значений стилевых свойств может быть сведено к определенному типу: строка, число, проценты, размер, цвет, адрес или ключевое слово. Рассмотрим их подробнее.

***Строки***

Любые строки необходимо брать в двойные или одинарные кавычки. Если внутри строки требуется оставить одну или несколько кавычек, то можно комбинировать типы кавычек или добавить перед кавычкой слеш.

Допустимы следующие строки:

1. 'Гостиница "Турист"' – применяются одинарные кавычки, а слово «Турист» взято в двойные кавычки;
2. "Гостиница 'Турист'" – применяются двойные кавычки, а слово «Турист» взято в одинарные кавычки;
3. "Гостиница \"Турист\"" – используются только двойные кавычки, но внутренние экранированы с помощью слеша.

***Числа***

Значением может выступать целое число, содержащее цифры от 0 до 9 и десятичная дробь, в которой целая и десятичная часть разделяются точкой (пример 5.14).

**Пример 5.14** – Числа в качестве значений

```
<html>
    <head>
        <title>Числа</title>
        <style type="text/css">
                p {
                    font-weight: 700; /* Жирное начертание */
                    line-height: 1.5; /* Межстрочный интервал */
                    }
        </style>
    </head>
 <body>
     <p>Пример текста</p>
 </body>
</html>
```

Если в десятичной дроби целая часть равна нулю, то ее можно не писать. Запись .1 и 0.1 равнозначна.

***Проценты***

Процентная запись обычно применяется в тех случаях, когда надо изменить значение относительно родительского элемента или когда размеры зависят от внешних условий. Так, ширина таблицы 100% означает, что она будет подстраиваться под размеры окна браузера и меняться вместе с шириной окна (пример 5.15).

**Пример 5.16** – Процентная запись

```
<html>
   <head>
      <title>Ширина в процентах</title>
      <style type="text/css">
            table {
                  width: 100%; /* Ширина таблицы в процентах */
                  background: #f0f0f0; /* Цвет фона */
                  }
      </style>
   </head>
 <body>
    <table>
      <tr><td>Содержимое таблицы</td></tr>
    </table>
 </body>
</html>
```

Проценты не обязательно должны быть целым числом, допускается использовать десятичные дроби, вроде значения 56.8%, но не всегда.

***Размеры***

Для задания размеров различных элементов, в CSS используются абсолютные и относительные единицы измерения. Абсолютные единицы не зависят от устройства вывода, а относительные единицы определяют размер элемента относительно значения другого размера.

*Относительные единицы* обычно используют для работы с текстом, либо когда надо вычислить процентное соотношение между элементами. В табл. 5.1 перечислены основные относительные единицы.

**Таблица 5.1** – Относительные единицы измерения

| Единица | Описание |
|---|---|
| Em | Размер шрифта текущего элемента |
| Ex | Высота символа x |
| Px | Пиксел |
| % | Процент |

Единица em это изменяемое значение, которое зависит от размера шрифта текущего элемента (размер устанавливается через стилевое

свойство font-size). В каждом браузере заложен размер текста, применяемый в том случае, когда этот размер явно не задан. Поэтому изначально 1em равен размеру шрифта, заданного в браузере по умолчанию или размеру шрифта родительского элемента. Процентная запись идентична em, в том смысле, что значения 1em и 100% равны.

Единица ex определяется как высота символа «x» в нижнем регистре. На ex распространяются те же правила, что и для em, а именно, он привязан к размеру шрифта, заданного в браузере по умолчанию, или к размеру шрифта родительского элемента.

Пиксел это элементарная точка, отображаемая монитором или другим подобным устройством, например, смартфоном. Размер пиксела зависит от разрешения устройства и его технических характеристик. В примере 5.16 показано применение пикселов и em для задания размера шрифта.

**Пример 5.16** – Использование относительных единиц

```
<html>
   <head>
      <title>Относительные единицы</title>
      <style type="text/css">
            h1 { font-size: 30px; }
            p { font-size: 1.5em; }
      </style>
   </head>
 <body>
    <h1>Заголовок размером 30 пикселов</h1>
    <p>Размер текста 1.5 em</p>
 </body>
</html>
```

Результат данного примера показан ниже (рис. 5.3).

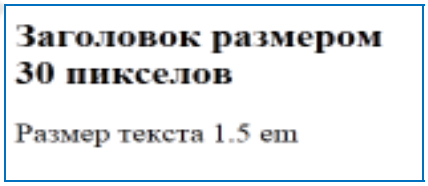



*Рисунок 5.5. - Размер текста при различных единицах*

*Абсолютные единицы* применяются реже, чем относительные и обычно при работе с текстом. В табл. 5.2 перечислены основные абсолютные единицы.

**Таблица 5.2** - Абсолютные единицы измерения

| Единица | Описание |
|---|---|
| In | Дюйм (1 дюйм равен 2,54 см) |
| Cm | Сантиметр |
| Mm | Миллиметр |
| Pt | Пункт (1пункт равен 1/72 дюйма) |
| Pc | Пика (1 пика равна 12 пунктам) |

Самой, пожалуй, распространенной единицей является пункт, который используется для указания размера шрифта. Хотя мы привыкли измерять все в миллиметрах и подобных единицах, пункт, пожалуй, единственная величина из не метрической системы измерения, которая используется у нас повсеместно. И все благодаря текстовым редакторам и издательским системам. В примере 5.17 показано использование пунктов и миллиметров.

**Пример 5.17** – Использование абсолютных единиц

```
<html>
   <head>
      <title>Абсолютные единицы</title>
      <style type="text/css">
            h1 { font-size: 25pt; }
            p { margin-left: 40mm; }
      </style>
   </head>
<body>
    <h1>Заголовок размером 25 пункта</h1>
    <p>Сдвиг текста вправо на 40 миллиметров</p>
</body>
</html>
```

Результат использования абсолютных единиц измерения показан ниже (рис. 5.6).

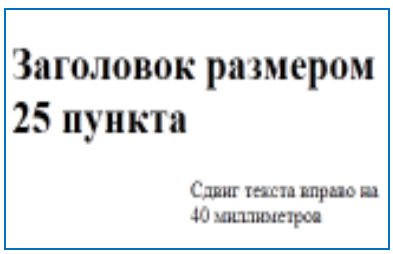



*Рисунок 5.6 - Размер текста при различных единицах*

При установке размеров обязательно указывайте единицы измерения, например width: 30px. В противном случае браузер не сможет

показать желаемый результат, поскольку не понимает, какой размер Вам требуется. Единицы не добавляются только при нулевом значении (margin: 0).

***Цвет***

Цвет в CSS можно задавать тремя способами: по шестнадцатеричному значению, по названию и в формате RGB (см. раздел 1.6).

В примере 5.18 представлены различные способы задания цветов элементов web-страниц.

**Пример 5.18** – Представление цвета

```
<html>
   <head>
      <title>Цвета</title>
      <style type="text/css">
            body { background-color: #C0C0C0; } /* Цвет фона web-страницы */
            h1 {background-color: RGB(29, 160, 200); } /* Цвет фона под заголовком */
            p { background-color: maroon; /* Цвет фона под текстом абзаца */
               color: white; /* Цвет текста */
               }
      </style>
   </head>
  <body>
      <h1>Институт повышения квалификации и переподготовки кадров</h1>
      <p>По окончании учёбы слушателям выдаётся диплом государственного образца о переподготовке на уровне высшего образования.</p>
  </body>
</html>
```

Результат данного примера показан на рис. 5.7.

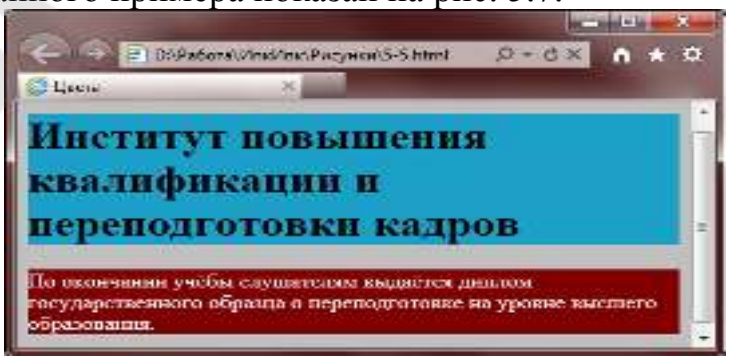



*Рисунок 5.7 - Цвета на web-странице*

***Адреса***

Адреса (URI, Uniform Resource Identifiers, унифицированный идентификатор ресурсов) применяются для указания пути к файлу, например, для установки фоновой картинки на странице. Для этого применяется ключевое слово url(), внутри скобок пишется относительный или абсолютный адрес файла. При этом адрес можно задавать в необязательных одинарных или двойных кавычках (пример 5.19).

**Пример 5.19** – Адрес графического файла

```
<html>
   <head>
      <title>Добавление фона</title>
      <style type="text/css">
            body { background: url('http://webimg.ru/images/156_1.png') no-repeat; }
            div {
               background: url(images/warning.png) no-repeat;
               padding-left: 20px;
               margin-left: 200px;
                }
      </style>
   </head>
 <body>
      <div>Внимание, запрашиваемая страница не найдена!</div>
 </body>
</html>
```

В примере 5.19 в селекторе body используется абсолютный адрес к графическому файлу, а в селекторе div — относительный.

***Ключевые слова***

В качестве значений активно применяются ключевые слова, которые определяют желаемый результат действия стилевых свойств. Ключевые слова пишутся без кавычек.

Правильно: p { text-align: right; }

Неверно: p { text-align: "right"; }

## 5.5 Применение селекторов к элементам на web-странице

***Селектор ID***

*Идентификатор* (называемый также «ID селектор») определяет уникальное имя элемента, которое используется для изменения его стиля и обращения к нему через скрипты.

Синтаксис применения идентификатора следующий.

**#Имя идентификатора { свойство1: значение; свойство2: значение; ... }**

При описании идентификатора вначале указывается символ решетки (#), затем идет имя идентификатора. Оно должно начинаться с латинского символа и может содержать в себе символ дефиса (-) и подчеркивания (_). Использование русских букв в именах идентификатора недопустимо. В отличие от классов идентификаторы должны быть уникальны, иными словами, встречаться в коде документа только один раз.

Обращение к идентификатору происходит аналогично классам, но в качестве ключевого слова у тега используется параметр id, значением которого выступает имя идентификатора (пример 5.20). Символ решетки при этом уже не указывается.

**Пример 5.20** – Использование идентификатора

```
<html>
  <head>
    <title>Идентификаторы</title>
    <style type="text/css">
      #help {
      position: absolute; /* Абсолютное позиционирование */
      left: 160px; /* Положение элемента от левого края */
      top: 50px; /* Положение от верхнего края */
      width: 225px; /* Ширина блока */
      padding: 5px; /* Поля вокруг текста */
      background: #f0f0f0; /* Цвет фона */
      }
    </style>
  </head>
<body>
  <div id="help">
  Идентификатор (называемый также «ID селектор») определяет уникальное имя элемента, которое используется для изменения его стиля и обращения к нему через скрипты.
  </div>
</body>
</html>
```

В данном примере определяется стиль тега  через идентификатор с именем help (рис. 5.8).

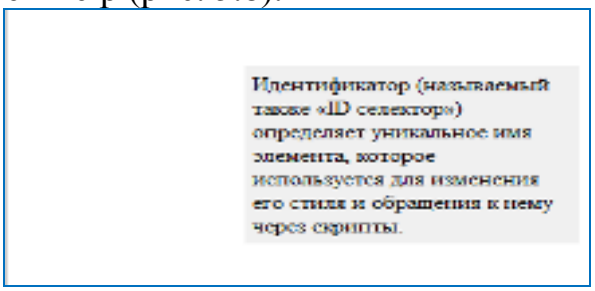



*Рисунок 5.8 - Результат применения идентификатора*

Как и при использовании классов, идентификаторы можно применять к конкретному тегу. Синтаксис при этом будет следующий.

**Тег#Имя идентификатора { свойство1: значение; свойство2: значение; ... }**

Вначале указывается имя тега, затем без пробелов символ решетки и название идентификатора. В примере 5.21 показано использование идентификатора применительно к тегу <p>.

**Пример 5.21** – Применение идентификатора совместно с тегом

```
<html>
   <head>
      <title>Идентификаторы</title>
      <style type="text/css">
            p {
               color: green; /* Зеленый цвет текста */
               font-style: italic; /* Курсивное начертание текста */
              }
            p#opa {
               color: red; /* Красный цвет текста */
               border: 1px solid #666; /* Параметры рамки */
               background: #eee; /* Цвет фона */
               padding: 5px; /* Поля вокруг текста */
            }
      </style>
   </head>
<body>
   <p>Обычный параграф</p>
    <p id="opa">Параграф необычный</p>
</body>
</html>
```

Результат примера 5.21 показан на рис. 5.9.

*Рисунок 5.9 - Вид текста после применения стиля*

В данном примере вводится стиль для тега <p> и для такого же тега, но с указанием идентификатора opa.

***Селекторы атрибутов***

Многие теги различаются по своему действию в зависимости от того, какие в них используются атрибуты. Например, тег <input> может создавать кнопку, текстовое поле и другие элементы формы всего лишь за счет изменения значение атрибута type. При этом добавление правил стиля к селектору INPUT применит стиль одновременно ко всем созданным с помощью этого тега элементам. Чтобы гибко управлять стилем подобных элементов, в CSS введены селекторы атрибутов. Они позволяют установить стиль по присутствию определенного атрибута тега или его значения.

Рассмотрим несколько типичных вариантов применения таких селекторов.

*Простой селектор атрибута* устанавливает стиль для элемента, если задан специфичный атрибут тега. Его значение в данном случае не важно. Синтаксис применения такого селектора следующий.

[**атрибут**] { Описание правил стиля }

**Селектор**[**атрибут**] { Описание правил стиля }

Стиль применяется к тем тегам, внутри которых добавлен указанный атрибут. Пробел между именем селектора и квадратными скобками не допускается.

В примере 5.22 показано изменение стиля тега <q>, в том случае, если к нему добавлен атрибут title.

**Пример 5.22** – Вид элемента в зависимости от его атрибута

```
<html>
   <head>
      <title>Селекторы атрибутов</title>
      <style type="text/css">
         q {
         font-style: italic; /* Курсивное начертание */
         quotes: "\00AB" "\00BB"; /* Меняем вид кавычек в цитате */
         }
        q[title] {
        color: red; /* Цвет текста */
        }
      </style>
   </head>
 <body>
       <p>Продолжая известный закон Мерфи, который гласит: <q>Если неприятность может случиться, то она обязательно случится</q>, можем ввести свое наблюдение: <q title="Из законов Фергюссона-Мержевича">После того, как web-страница будет корректно отображаться в одном браузере,
```

выяснится, что она неправильно показывается в другом</q>.</p>
</body>
</html>

Результат примера показан на рис. 5.10.

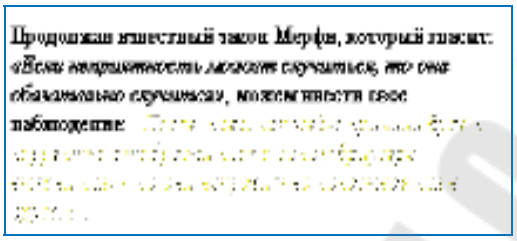


*Рисунок 5.10 - Изменение стиля элемента в зависимости от применения атрибута title*

В данном примере меняется цвет текста внутри контейнера <q>, когда к нему добавляется title. Обратите внимание, что для селектора q[title] нет нужды повторять стилевые свойства, поскольку они наследуются от селектора q.

*Атрибут со значением* устанавливает стиль для элемента в том случае, если задано определенное значение специфичного атрибута. Синтаксис применения следующий:

[**атрибут**="**значение**"] { Описание правил стиля }

**Селектор**[**атрибут**="**значение**"] { Описание правил стиля }

В первом случае стиль применяется ко всем тегам, которые содержат указанное значение. А во втором — только к определенным селекторам.

В примере 5.23 показано изменение стиля ссылки в том случае, если тег <a> содержит атрибут target со значением _blank. При этом ссылка будет открываться в новом окне и чтобы показать это, с помощью стилей добавляем небольшой рисунок перед текстом ссылки.

**Пример 5.23** – Стиль для открытия ссылок в новом окне

```
<html>
   <head>
      <title>Селекторы атрибутов</title>
      <style type="text/css">
            a[target="_blank"] {
                  background: url(images/blank.png) 0 6px no-repeat; /*
Параметры фонового рисунка */
                  padding-left: 15px; /* Смещаем текст вправо */
                  }
      </style>
   </head>
<body>
```

```
        <p><a href="1.html">Обычная ссылка</a> |
        <a href="link2" target="_blank">Ссылка в новом окне</a></p>
    </body>
    </html>
```

Результат примера показан ниже (рис. 5.11).

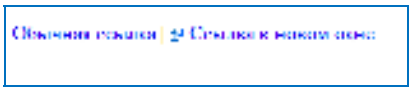


*Рисунок 5.11 - Изменение стиля элемента в зависимости от значения target*

*Значение атрибута начинается с определенного текста* устанавливает стиль для элемента в том случае, если значение атрибута тега начинается с указанного текста. Синтаксис применения следующий:

**[атрибут^="значение"]** { Описание правил стиля }

**Селектор[атрибут^="значение"]** { Описание правил стиля }

В первом случае стиль применяется ко всем элементам, у которых значение атрибута начинаются с указанного текста. А во втором — только к определенным селекторам. Использование кавычек не обязательно.

Предположим, что на сайте требуется разделить стиль обычных и внешних ссылок — ссылки, которые ведут на другие сайты. Чтобы не добавлять к тегу <a> новый класс, воспользуемся селекторами атрибутов. Внешние ссылки характеризуются добавлением к адресу протокола, например, для доступа к гипертекстовым документам используется протокол HTTP. Поэтому внешние ссылки начинаются с ключевого слова http://, его и добавляем к селектору A, как показано в примере 5.24.

**Пример 5.24** – Изменение стиля внешней ссылки

```
<html>
    <head>
        <title>Селекторы атрибутов</title>
        <style type="text/css">
            a[href^="http://"] {
                font-weight: bold /* Жирное начертание */
                }
        </style>
    </head>
<body>
    <p><a href="1.html">Обычная ссылка</a> |
    <a href="http://htmlbook.ru" target="_blank">Внешняя ссылка на сайт gstu.by</a></p>
</body>
</html>
```

Результат примера показан ниже (рис. 5.12).

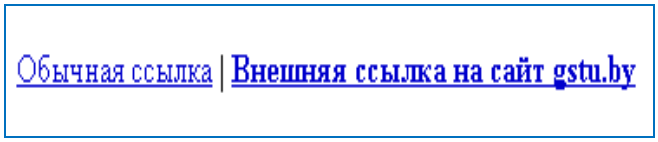



*Рисунок 5.12 - Изменение стиля для внешних ссылок*

В данном примере внешние ссылки выделяются жирным начертанием. Также можно воспользоваться показанным в примере 5.24 приемом и добавлять к ссылке небольшое изображение, которое будет сообщать, что ссылка ведет на другой сайт.

*Значение атрибута оканчивается определенным текстом* устанавливает стиль для элемента в том случае, если значение атрибута оканчивается указанным текстом. Синтаксис применения следующий.

[**атрибут**$="**значение**"] { Описание правил стиля }

**Селектор**[**атрибут**$="**значение**"] { Описание правил стиля }

В первом случае стиль применяется ко всем элементам, у которых значение атрибута завершается заданным текстом. А во втором — только к определенным селекторам.

Таким способом можно автоматически разделять стиль для ссылок на сайты домена ru и для ссылок на сайты других доменов вроде com, как показано в примере 5.25.

**Пример 5.25 -** Стиль для разных доменов

```
<html>
  <head>
    <title>Селекторы атрибутов</title>
    <style type="text/css">
      a[href$=".ru"] { /* Если ссылка заканчивается на .ru */
        background: url(images/ru.png) no-repeat 0 6px; /* Добавляем фоновый рисунок */
        padding-left: 12px; /* Смещаем текст вправо */
        }
      a[href$=".com"] { /* Если ссылка заканчивается на .com */
        background: url(images/com.png) no-repeat 0 6px;
        padding-left: 12px;
        }
    </style>
  </head>
<body>
    <p><a href="http://www.yandex.com">Yandex.Com</a> |
```

```
<a href="http://www.yandex.ru">Yandex.Ru</a></p>

</body>
</html>
```

В данном примере содержатся две ссылки, ведущие на разные домены — com и ru. При этом к каждой такой ссылке с помощью стилей добавляется своя фоновая картинка (рис. 5.13). Стилевые свойства будут применяться только для тех ссылок, атрибут href которых оканчивается на «.ru» или «.com». Заметьте, что добавив к имени домена слэш (http://www.yandex.ru/) или адрес страницы (http://www.yandex.ru/fun.html), мы изменим тем самым окончание и стиль применяться уже не будет. В этом случае лучше воспользоваться селектором, у которого заданный текст встречается в любом месте значения атрибута.


*Рисунок 5.13 - Добавление картинки к ссылкам*

*Значение атрибута содержит указанный текст.* Возможны варианты, когда стиль следует применить к тегу с определенным атрибутом, при этом частью его значения является некоторый текст. При этом точно не известно, в каком месте значения включен данный текст — в начале, середине или конце. В подобном случае следует использовать такой синтаксис:

[**атрибут***="**значение**"] { Описание правил стиля }

**Селектор**[**атрибут***="**значение**"] { Описание правил стиля }

В примере 5.30 показано изменение стиля ссылок, в атрибуте href которых встречается слово «gstu».

**Пример 5.26** – Стиль для разных сайтов

```
<html>
   <head>
      <title>Селекторы атрибутов</title>
      <style type="text/css">
            [href*="gstu"] { background: yellow; } /* Желтый цвет фона */
      </style>
   </head>
<body>
      <p><a href="http://www.gstu.by/science/">Наука</a> |
      <a href="http://education.gstu.by">Учебный портал</a> |
```

```
        <a href="http://webimg.ru">Графика для Веб</a></p>
</body>
</html>
```

Результат данного примера показан на рис. 5.14

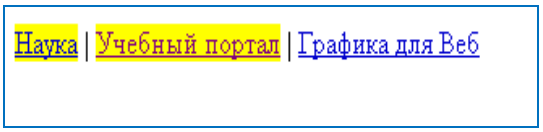



*Рисунок 5.14 - Изменение стиля для ссылок, в адресе которых встречается «gstu»*

*Одно из нескольких значений атрибута.* Некоторые значения атрибутов могут перечисляться через пробел, например имена классов. Чтобы задать стиль при наличии в списке требуемого значения применяется следующий синтаксис:

[**атрибут**~="**значение**"] { Описание правил стиля }

**Селектор**[**атрибут**~="**значение**"] { Описание правил стиля }

Стиль применяется в том случае, если у атрибута имеется указанное значение или оно входит в список значений, разделяемых пробелом (пример 5.27).

**Пример 5.27** – Стиль в зависимости от имени класса

```
<html>
    <head>
      <title>Блок</title>
      <style type="text/css">
            [class~="block"] h3 { color: green; }
      </style>
    </head>
<body>
    <div class="block tag">
      <h3>Заголовок</h3>
    </div>
</body>
</html>
```

В данном примере зеленый цвет текста применяется к селектору h3, если имя класса у слоя задано как block. Отметим, что аналогичный результат можно получить, если использовать конструкцию *= вместо ~=.

*Дефис в значении атрибута.* В именах идентификаторов и классов разрешено использовать символ дефиса (-), что позволяет создавать значащие значения атрибутов id и class. Для изменения стиля элементов, в значении которых применяется дефис, следует воспользоваться следующим синтаксисом.

[**атрибут**|="**значение**"] { Описание правил стиля }

**Селектор**[**атрибут**|="**значение**"] { Описание правил стиля }

Стиль применяется к элементам, у которых атрибут начинается с указанного значения или с фрагмента значения, после которого идет дефис (пример 5.31).

**Пример 5.28** – Дефисы в значениях

```
<html>
    <head>
        <title>Блок</title>
        <style type="text/css">
            div[class|="block"] {
                background: #306589; /* Цвет фона */
                color: #acdb4c; /* Цвет текста */
                padding: 5px; /* Поля */
                }
            div[class|="block"] a { color: #fff; } /* Цвет ссылок */
        </style>
    </head>
<body>
      <div class="block-menu-therm">
            <h2>Термины</h2>
            <div class="content">
                <ul class="menu">
                    <li><a href="t1.html">Буквица</a></li>
                    <li><a href="t2.html">Выворотка</a></li>
                    <li><a href="t3.html">Выключка</a></li>
                    <li><a href="t4.html">Интерлиньяж</a></li>
                    <li><a href="t5.html">Капитель</a></li>
                    <li><a href="t6.html">Начертание</a></li>
                    <li><a href="t7.html">Отбивка</a></li>
                </ul>
            </div>
      </div>
</body>
</html>
```

В данном примере имя класса задано как block-menu-therm, поэтому в стилях используется конструкция |="block", поскольку значение начинается именно с этого слова и в значении встречаются дефисы.

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

[**атрибут1**="**значение1**"][**атрибут2**="**значение2**"] { Описание правил стиля }

**Селектор**[**атрибут1**="**значение1**"][**атрибут2**="**значение2**"] { Описание правил стиля }.

***Псевдоклассы***

Псевдоклассы определяют динамическое состояние элементов, которое изменяется со временем или с помощью действий пользователя, а также положение в дереве документа. Примером такого состояния служит текстовая ссылка, которая меняет свой цвет при наведении на нее курсора мыши. При использовании псевдоклассов браузер не перегружает текущий документ, поэтому с помощью псевдоклассов можно получить разные динамические эффекты на странице.

Синтаксис применения псевдоклассов следующий:

**Селектор**:**Псевдокласс** { Описание правил стиля }

Вначале указывается селектор, к которому добавляется псевдокласс, затем следует двоеточие, после которого идет имя псевдокласса. Допускается применять псевдоклассы к именам идентификаторов или классов (A.menu:hover {color: green}), а также к контекстным селекторам (.menu A:hover {background: #fc0}). Если псевдокласс указывается без селектора впереди (:hover), то он будет применяться ко всем элементам документа.

Условно все псевдоклассы делятся на три группы:

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

Рассмотрим подробнее все псевдоклассы.

*Псевдоклассы, определяющие состояние элементов*

К этой группе относятся псевдоклассы, которые распознают текущее состояние элемента и применяют стиль только для этого состояния.

**:active**

Происходит при активации пользователем элемента. Например, ссылка становится активной, если навести на нее курсор и щелкнуть мышкой. Несмотря на то, что активным может стать практически любой элемент web-страницы, псевдокласс :active используется преимущественно для ссылок.

**:link**

Применяется к непосещенным ссылкам, т.е. таким ссылкам, на которые пользователь еще не нажимал. Браузер некоторое время сохраняет историю посещений, поэтому ссылка может быть помечена как посещенная хотя бы потому, что по ней был зафиксирован переход раньше.

Запись A {...} и A:link {...} по своему результату равноценна, поскольку в браузере дает один и тот же эффект, поэтому псевдокласс :link можно не указывать. Исключением являются якоря, на них действие :link не распространяется.

**:focus**

Применяется к элементу при получении им фокуса. Например, для текстового поля формы получение фокуса означает, что курсор установлен в поле, и с помощью клавиатуры можно вводить в него текст (пример 5.29).

**Пример 5.29 –** Применение псевдокласса focus

```
<html>
   <head>
        <title>Псевдоклассы</title>
        <style type="text/css">
              inpute:focus {
                    color: red; /* Красный цвет текста */
                    }
        </style>
   </head>
<body>
     <form action="">
          <p><input type="text" value="Черный текст"></p>
          <p><input type="text" value="Черный текст"></p>
     </form>
</body>
</html>
```

Результат примера показан ниже (рис. 5.15). Во второй строке находится курсор, поэтому текстовое поле получило фокус.

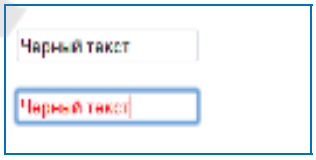



*Рисунок 5.15 - Изменение стиля текста при получении фокуса*

В данном примере в текстовом поле содержится предварительный текст, он определяется значением атрибута value тега <input>. При щелчке по элементу формы происходит получение полем фокуса, и цвет текста меняется на красный. Достаточно щелкнуть в любом месте страницы (кроме текстового поля, естественно), как произойдет потеря фокуса и текст вернется к первоначальному черному цвету.

Результат будет виден только для тех элементов, которые могут получить фокус. В частности, это теги <a>, <input>, <select> и <textarea>.

**:hover**

Псевдокласс :hover активизируется, когда курсор мыши находится в пределах элемента, но щелчка по нему не происходит.

**:visited**

Данный псевдокласс применяется к посещенным ссылкам. Обычно такая ссылка меняет свой цвет по умолчанию на фиолетовый, но с помощью стилей цвет и другие параметры можно задать самостоятельно (пример 5.30).

**Пример 5.30** – Изменение цвета ссылок

```
<html>
   <head>
        <title>Псевдоклассы</title>
        <style type="text/css">
              a:link { color: #036; }/* Цвет непосещенных ссылок */
              a:visited { color: #606; } /* Цвет посещенных ссылок */
              a:hover { color: #f00; } /* Цвет ссылок при наведении на них курсора мыши */
              a:active { color: #ff0; } /* Цвет активных ссылок */
        </style>
   </head>
<body>
      <p>
        <a href="1.html">Ссылка 1</a> |
        <a href="2.html">Ссылка 2</a> |
        <a href="3.html">Ссылка 3</a>
         </p>
</body>
</html>
```

Результат примера показан на рис. 5.16.

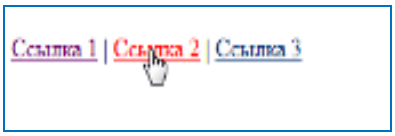



*Рисунок 5.16 - Вид ссылки при наведении на нее курсора мыши*

В данном примере показано использование псевдоклассов совместно со ссылками. При этом имеет значение порядок следования псевдоклассов. Вначале указывается :visited, а затем идет :hover, в противном случае посещенные ссылки не будут изменять свой цвет при наведении на них курсора.

Селекторы могут содержать более одного псевдокласса, которые перечисляются подряд через двоеточие, но только в том случае, если их действия не противоречат друг другу. Так, запись A:visited:hover является корректной, а запись A:link:visited — нет. Впрочем, если подходить формально, то валидатор CSS считает правильным любое сочетание псевдоклассов.

Браузер Internet Explorer 6 и младше позволяет использовать псевдоклассы :active и :hover только для ссылок. Начиная с версии 7.0 псевдоклассы в этом браузере работают и для других элементов.

Псевдокласс :hover не обязательно должен применяться к ссылкам, его можно добавлять и к другим элементам документа. Так можно создать таблицу, строки которой меняют свой цвет при наведении на них курсора мыши. Это достигается за счет добавления псевдокласса :hover к селектору TR.

*Псевдоклассы, имеющие отношение к дереву документа*

К этой группе относятся псевдоклассы, которые определяют положение элемента в дереве документа и применяют к нему стиль в зависимости от его статуса.

**:first-child**

Применяется к первому дочернему элементу селектора, который расположен в дереве элементов документа. Чтобы стало понятно, о чем речь, разберем небольшой код (пример 5.31).

**Пример 5.31** – Использование псевдокласса :first-child

```
<html>
  <head>
    <title>Псевдоклассы</title>
    <style type="text/css">
      b:first-child { color: red; } /* Красный цвет текста */
    </style>
  </head>
<body>
    <p><b>Динамический HTML</b> построен на объектной <b>модели</b>, которая расширяет традиционный <b>статический</b> HTML-документ.</p>
    <p><b>Таблицы стилей CSS</b> документа могут быть <b>изменены</b> в любое время.</p> </body>
</html>
```

Результат примера показан ниже (рис. 5.17).

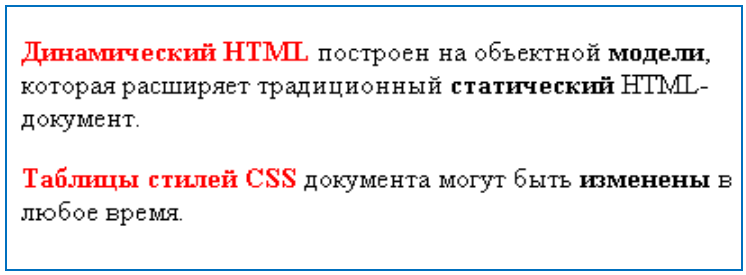



*Рисунок 5.17 - Использование псевдокласса first-child*

В данном примере псевдокласс :first-child добавляется к селектору B и устанавливает для него красный цвет текста. Хотя контейнер <b> встречается в первом абзаце три раза, красным цветом будет выделено лишь первое упоминание, т.е. текст «Динамический HTML». В остальных случаях содержимое контейнера <b> отображается черным цветом. Со следующим абзацем все начинается снова, поскольку родительский элемент поменялся. Поэтому фраза «Таблицы стилей CSS» также будет выделена красным цветом.

Браузер Internet Explorer поддерживает псевдокласс :first-child начиная с версии 7.0.

Псевдокласс :first-child удобнее всего использовать в тех случаях, когда требуется задать разный стиль для первого и остальных однотипных элементов. Например, по правилам типографики красную строку для первого абзаца текста не устанавливают, а для остальных абзацев добавляют отступ первой строки. С этой целью применяют свойство text-indent с нужным значением отступа. Но чтобы изменить стиль первого абзаца и убрать для него отступ потребуется воспользоваться псевдоклассом :first-child.

***Псевдоэлементы***

Псевдоэлементы позволяют задать стиль элементов не определенных в дереве элементов документа, а также генерировать содержимое, которого нет в исходном коде текста.

Синтаксис использования псевдоэлементов следующий.

**Селектор**:**Псевдоэлемент** { Описание правил стиля }

Вначале следует имя селектора, затем пишется двоеточие, после которого идет имя псевдоэлемента. Каждый псевдоэлемент может применяться только к одному селектору, если требуется установить сразу несколько псевдоэлементов для одного селектора, правила стиля должны добавляться к ним по отдельности, как показано ниже.

```
.foo:first-letter { color: red }
.foo:first-line {font-style: italic}
```

Псевдоэлементы не могут применяться к внутренним стилям, только к таблице связанных или глобальных стилей.

Далее перечислены все псевдоэлементы, их описание и свойства.

**:after**

Применяется для вставки назначенного контента после элемента. Этот псевдоэлемент работает совместно со стилевым свойством content, которое определяет содержимое для вставки. В примере 5.32 показано использование псевдоэлемента :after для добавления текста в конец абзаца.

**Пример 5.32 –** Применение псевдоэлемента :after

```
<html>
   <head>
      <title>Псевдоэлементы</title>
      <style type="text/css">
         P.new:after {
               content: " - ДА!"; /* Добавляем после текста абзаца
*/
               }
      </style>
   </head>
<body>
   <p class="new">Ловля льва в пустыне с помощью метода золотого сечения.</p>
    <p>Метод ловли льва простым перебором.</p>
</body>
</html>
```

Результат примера показан на рис. 5.18.

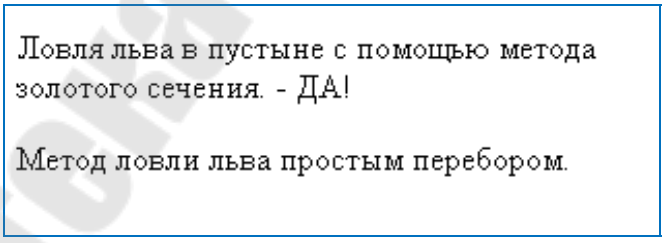



*Рисунок 5.18 - Добавление текста к абзацу с помощью :after*

В данном примере к содержимому абзаца с классом new добавляется дополнительное слово, которое выступает значением свойства content.

Псевдоэлементы :after и :before, а также стилевое свойство content не поддерживаются браузером Internet Explorer до седьмой версии включительно.

**:before**

По своему действию :before аналогичен псевдоэлементу :after, но вставляет контент до элемента. В примере 5.33 показано добавление маркеров своего типа к элементам списка посредством скрытия стандартных маркеров и применения псевдоэлемента :before.

**Пример 5.33** – Использование псевдоэлемента :before

```
<html>
  <head>
    <title>Псевдоэлементы</title>
    <style type="text/css">
      ul {
        padding-left: 0; /* Убираем отступ слева */
        list-style-type: none; /* Прячем маркеры списка */
        }
      li:before { content: "\20aa ";} /* Добавляем перед элементом списка символ в юникоде */
    </style>
  </head>
<body>
    <ul>
      <li>Чебурашка</li>
      <li>Крокодил Гена</li>
      <li>Шапокляк</li>
      <li>Крыса Лариса</li>
    </ul>
</body>
</html>
```

Результат примера показан ниже (рис. 5.19).

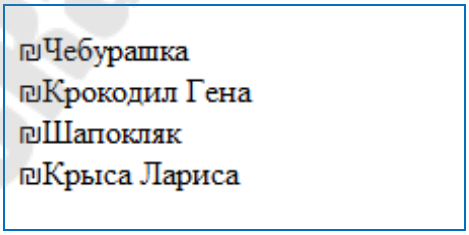



*Рисунок 5.19 - Изменение вида маркеров с помощью :before*

В данном примере псевдоэлемент :before устанавливается для селектора li, определяющего элементы списка. Добавление желаемых символов происходит путем задания значения свойства content. Обратите внимание, что в качестве аргумента не обязательно выступает текст, могут применяться также символы юникода.

**:first-letter**

Определяет стиль первого символа в тексте элемента, к которому добавляется. Это позволяет создавать в тексте буквицу и выступающий инициал.

Буквица представляет собой увеличенную первую букву, базовая линия которой ниже на одну или несколько строк базовой линии основного текста. Выступающий инициал — увеличенная прописная буква, базовая линия которой совпадает с базовой линией основного текста.

Рассмотрим пример создания выступающего инициала. Для этого требуется добавить к селектору P псевдоэлемент :first-letter и установить желаемый стиль инициала. В частности, увеличить размер текста и поменять цвет текста (пример 5.34).

**Пример 5.34 –** Использование псевдоэлемента :first-letter

```
<html>
  <head>
    <title>Псевдоэлементы</title>
    <style type="text/css">
                    p {
                    font-family: Arial, Helvetica, sans-serif; /* Гарнитура шрифта основного текста */
                    font-size: 90%; /* Размер шрифта */
                    color: black; /* Черный цвет текста */
                    }
              p:first-letter {
                    font-family: 'Times New Roman', Times, serif; /* Гарнитура шрифта первой буквы */
                    font-size: 200%; /* Размер шрифта первого символа */
                    color: red; /* Красный цвет текста */
                     }
    </style>
  </head>
<body>
    <p>Институт повышения квалификации и переподготовки кадров является структурным
    подразделением Учреждения образования «Гомельский государственный технический университет имени П.О. Сухого», действующим на условиях полного хозяйственного расчёта и самофинансирования без права юридического лица.</p>
    <p>Деятельность по повышению квалификации и переподготовки кадров в университете осуществляется
```

с 2 мая 1997 года, когда был создан факультет повышения квалификации и переподготовки кадров. </p>

</body>

</html>

Результат примера показан ниже (рис. 5.20).

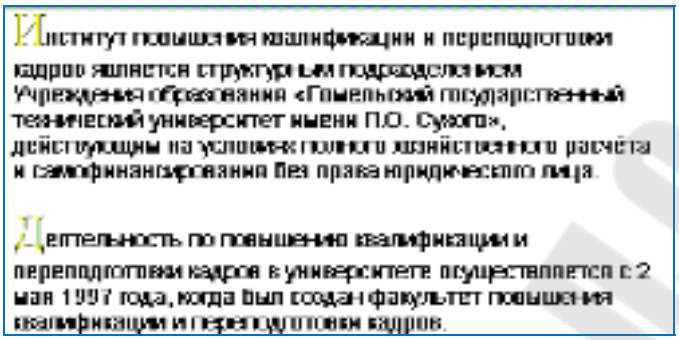



*Рисунок 5.20 - Создание выступающего инициала*

В данном примере изменяется шрифт, размер и цвет первой буквы каждого абзаца текста.

**:first-line**

Определяет стиль первой строки блочного текста. Длина этой строки зависит от многих факторов, таких как используемый шрифт, размер окна браузера, ширина блока, языка и т.д.

К псевдоэлементу :first-line могут применяться не все стилевые свойства. Допустимо использовать свойства, относящиеся к шрифту, изменению цвет текста и фона, а также: clear, line-height, letter-spacing, text-decoration, text-transform, vertical-align и word-spacing.

В примере 5.35 показано использование псевдоэлемента :first-line применительно к абзацу текста.

**Пример 5.35 –** Выделение первой строки текста

```
<html>
   <head>
      <title>Псевдоэлементы</title>
      <style type="text/css">
            p:first-line {
                  color: red; /* Красный цвет текста */
                  font-style: italic; /* Курсивное начертание */
                  }
      </style>
   </head>
<body>
      <p> Институт повышения квалификации и переподготовки кадров является структурным подразделением
```

Учреждения образования «Гомельский государственный технический университет имени П.О. Сухого», действующим на условиях полного хозяйственного расчёта и самофинансирования без права юридического лица.

</body>

</html>

Результат примера показан на рис. 5.21.

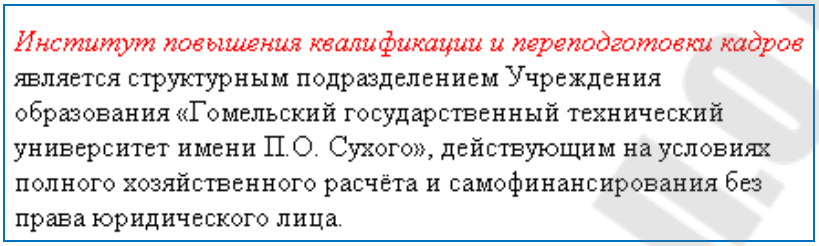



*Рисунок 5.21 - Результат применения псевдоэлемента :first-line*

В данном примере первая строка выделяется красным цветом и курсивным начертанием. Обратите внимание, что при изменении ширины окна браузера, стиль первой строки остается постоянным, независимо от числа входящих в нее слов.

Из рассмотренных нами селекторов можно составлять комбинации.

***Контекстный селектор***

При создании web-страницы часто приходится вкладывать одни теги внутрь других. Чтобы стили для этих тегов использовались корректно, помогут селекторы, которые работают только в определенном контексте. Например, задать стиль для тега <b> только когда он располагается внутри контейнера . Таким образом можно одновременно установить стиль для отдельного тега, а также для тега, который находится внутри другого.

Контекстный селектор состоит из простых селекторов разделенных пробелом. Так, для селектора тега синтаксис будет следующий.

**Тег1 Тег2** { ... }

В этом случае стиль будет применяться к Тегу2 когда он размещается внутри Тега1, как показано ниже.

<Тег1>

<Тег2> ... </Тег2>

</Тег1>

Использование контекстных селекторов продемонстрировано в примере 5.36.

**Пример 5.36** – Контекстные селекторы

<html>

<head>

```
        <title>Контекстные селекторы</title>
        <style type="text/css">
            p b {
                font-family: Times, serif; /* Семейство шрифта */
                font-weight: bold; /* Жирное начертание */
                color: navy; /* Синий цвет текста */
                }
        </style>
    </head>
<body>
        <div><b>Жирное начертание текста</b></div>
        <p><b>Одновременно жирное начертание текста и выделенное цветом</b></p>
</body>
</html>
```

В данном примере показано обычное применение тега <b> и этого же тега, когда он вложен внутрь абзаца <p>. При этом меняется цвет и шрифт текста, как показано на рис. 5.22.

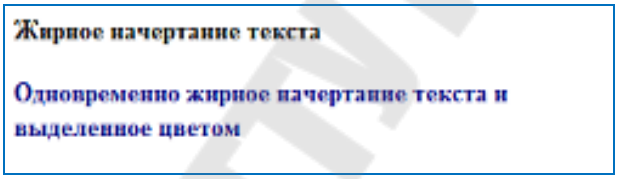



*Рисунок 5.22 - Оформление текста в зависимости от вложенности тегов*

Не обязательно контекстные селекторы содержат только один вложенный тег. В зависимости от ситуации допустимо применять два и более последовательно вложенных друг в друга тегов.

Более широкие возможности контекстные селекторы дают при использовании идентификаторов и классов. Это позволяет устанавливать стиль только для того элемента, который располагается внутри определенного класса.

***Дочерний селектор***

Дочерним называется элемент, который непосредственно располагается внутри родительского элемента. Чтобы лучше понять отношения между элементами документа, разберем небольшой код (пример 5.37).

**Пример 5.37** - Вложенность элементов в документе

```
<html>
    <head>
        <title>Псевдоклассы</title>
```

```
    </head>
  <body>
     <div class="main">
       <p><em>Псевдоклассы, определяющие состояние элементов</em>, распознают текущее
       состояние элемента и применяют стиль только для этого состояния.</p>
       <p><strong><em>Псевдоклассы, имеющие отношение к дереву документа</em></strong>,
       определяют положение элемента в дереве документа и применяют к нему стиль в
       зависимости от его статуса.</p>
     </div>
  </body>
  </html>
```

В данном примере применяется несколько контейнеров, которые в коде располагаются один в другом. Нагляднее это видно на дереве элементов, так называется структура отношений тегов документа между собой (рис. 5.23).


*Рисунок 5.23 - Дерево элементов*

На рис. 5.23 в удобном виде представлена вложенность элементов и их иерархия. Здесь дочерним элементом по отношению к тегу  выступает тег <p>. Вместе с тем тег  не является дочерним для тега , поскольку он расположен в контейнере <p>.

Вернемся теперь к селекторам. Дочерним селектором считается такой, который в дереве элементов находится прямо внутри родительского элемента. Синтаксис применения таких селекторов следующий.

**Селектор 1** > **Селектор 2** { Описание правил стиля }

Стиль применяется к Селектору 2, но только в том случае, если он является дочерним для Селектора 1.

Если снова обратиться к примеру 5.37, то стиль вида p > em { color: red } будет установлен для первого абзаца документа, поскольку тег 

находится внутри контейнера <p>, и не даст никакого результата для второго абзаца. А все из-за того, что тег  во втором абзаце расположен в контейнере , поэтому нарушается условие вложенности.

По своей логике дочерние селекторы похожи на селекторы контекстные. Разница между ними следующая. Стиль к дочернему селектору применяется только в том случае, когда он является прямым потомком, иными словами, непосредственно располагается внутри родительского элемента. Для контекстного селектора же допустим любой уровень вложенности. Чтобы стало понятно, о чем идет речь, разберем следующий код (пример 5.38).

**Пример 5.38** – Контекстные и дочерние селекторы

```
<html>
   <head>
      <title>Дочерние селекторы</title>
      <style type="text/css">
            div i { /* Контекстный селектор */
                  color: green; /* Зеленый цвет текста */
                  }
            p > i { /* Дочерний селектор */
                  color: red; /* Красный цвет текста */
                  }
      </style>
   </head>
<body>
      <div>
            <p><i>Одна из первых задач web-дизайнера</i> заключается в разработке  схемы с
            <i>картой сайта</i>. Затем он должен создать набор макетов.</p>
      </div>
</body>
</html>
```

Результат данного примера показан на рис. 5.24.

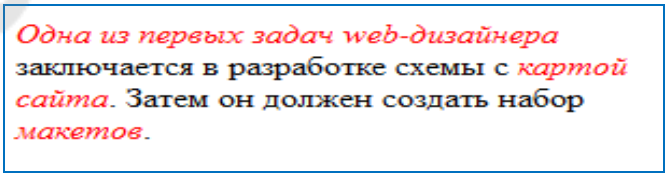



*Рисунок 5.24 - Цвет текста, заданный с помощью дочернего селектора*

На тег  в примере действуют одновременно два правила: контекстный селектор (тег  расположен внутри ) и дочерний селектор (тег  является дочерним по отношению к ). При этом правила являются равносильными, поскольку все условия для них выполняются и не противоречат друг другу. В подобных случаях применяется стиль, который расположен в коде ниже, поэтому курсивный текст отображается красным цветом. Стоит поменять правила местами и поставить DIV I ниже, как цвет текста изменится с красного на зеленый.

Заметим, что в большинстве случаев от добавления дочерних селекторов можно отказаться, заменив их контекстными селекторами. Однако использование дочерних селекторов расширяет возможности по управлению стилями элементов, что в итоге позволяет получить нужный результат, а также простой и наглядный код.

Удобнее всего применять указанные селекторы для элементов, которые обладают иерархической структурой — сюда относятся, например, таблицы и разные списки.

***Соседний селектор***

Соседними называются элементы web-страницы, когда они следуют непосредственно друг за другом в коде документа. Рассмотрим несколько примеров отношения элементов.

 Наша мама мыла большую раму.

В этом примере тег  является дочерним по отношению к тегу , поскольку он находится внутри этого контейнера. Соответственно  выступает в качестве родителя .

 Наша мама мыла <var>большую</var> раму.

Здесь теги <var> и  никак не перекрываются и представляют собой соседние элементы. То, что они расположены внутри контейнера , никак не влияет на их отношение.

 Наша мама  мыла  большую раму, почерневшую от времени <tt> и мух </tt>.

Соседними здесь являются теги  и , а также  и <tt>. При этом  и <tt> к соседним элементам не относятся из-за того, что между ними расположен контейнер .

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

**Селектор 1 + Селектор 2** { Описание правил стиля }

Пробелы вокруг плюса не обязательны, стиль при такой записи применяется к Селектору 2, но только в том случае, если он является соседним для Селектора 1 и следует сразу после него.

В примере 5.39 показана структура взаимодействия тегов между собой.

**Пример 5.39** – Использование соседних селекторов

```
<html>
  <head>
    <title>Соседние селекторы</title>
    <style type="text/css">
          b + i {
                color: red; /* Красный цвет текста */
                }
    </style>
  </head>
<body>
     <p>WEB-<b>дизайнер</b> должен создать набор <i>макетов</i>.</p>
     <p>Эти макеты должны отображать <i>содержание</i> и <i> навигационные элементы </i>
     каждой страницы.</p>
</body>
</html>
```

Результат примера показан на рис. 5.25.

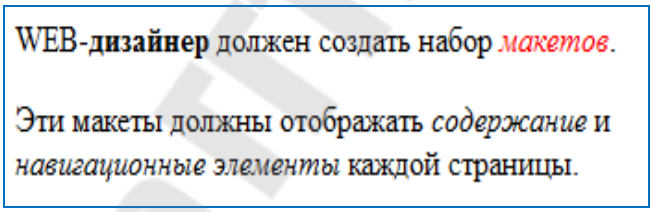



*Рисунок 5.25 - Выделение текста цветом при помощи соседних селекторов*

В данном примере происходит изменение цвета текста для содержимого контейнера <i>, когда он располагается сразу после контейнера <b>. В первом абзаце такая ситуация реализована, поэтому слово «consectetuer» в браузере отображается красным цветом. Во втором абзаце, хотя и присутствует тег <i>, но по соседству никакого тега <b> нет, так что стиль к этому контейнеру не применяется.

***Универсальный селектор***

Иногда требуется установить одновременно один стиль для всех элементов web-страницы, например, задать шрифт или начертание текста. В этом случае поможет универсальный селектор, который соответствует любому элементу web-страницы.

Для обозначения универсального селектора применяется символ звездочки (*) и в общем случае синтаксис будет следующий.

* { Описание правил стиля }

В некоторых случаях указывать универсальный селектор не обязательно. Так, например, записи *.class и .class являются идентичными по своему результату.

В примере 5.40 показано одно из возможных приложений универсального селектора — выбор шрифта и размера текста для всех элементов документа.

**Пример 5.40** – Использование универсального селектора

```
<html>
   <head>
      <title>Универсальный селектор</title>
      <style type="text/css">
            * {
               font-family: Arial, Verdana, sans-serif; /* Рубленый шрифт для текста */
               font-size: 96%; /* Размер текста */
              }
      </style>
   </head>
<body>
      <p> Универсальный селектор помогает установить одновременно один стиль для всех элементов web-страницы.</p>
</body>
</html>
```

Заметим, что аналогичный результат можно получить, если в данном примере поменять селектор * на BODY.

## 5.6 Декоративные возможности CSS: форматирование текста, формирование рамок и отступов

Приступим к изучению правил css. Начнем с правил, которые относятся к шрифтам.

*Font-family: семейства шрифта | тип шрифта;*

Задает описание шрифта. Аналогом в HTML является атрибут face элемента font (см. .

Например, font-family: Arial, Geneva, Helvetica, sans-serif.

Если название шрифта состоит из нескольких слов, разделенных пробелами, например, Times New Roman, то в отличие от HTML необходимо это название заключить в кавычки.

*Font-size: величина | %;*

Задает величину шрифта, которую можно задавать в em, ex и px. Другое значение – это проценты (%). Вы можете написать 2em, что будет означать увеличение шрифта в два раза. Все правила, которые начинаются с приставки font, наследуются. Это означает, что все элементы на web-странице уже получают какой-то размер шрифта, который можно изменить прописав, например, h1{font-size:2em;}. Последнее правило, следует понимать так: элементу h1 «пришел» размер шрифта, такой же как и всем, а вы его увеличили в 2 раза. Заметим, что правила font-size:2em; и font-size:200%; работают одинаково. Проценты берутся от того размера, который «пришел» к элементу.

Еще есть 7 размеров, которые коррелируют с атрибутом size:

1. Абсолютные величины: xx-small, x-small, small, medium, large, x-large, xx-large.
2. Относительные величины: larger, smaller.

*Font-width: normal | bold | bolder | lighter | число от 100 и до 900;*

Данное правило задает начертание шрифта по жирности.

Отметим, что 400=normal, 700=bold. Пока работают только первые два значения font-width.

*Font-style: normal | italic | oblique;*

Это правило задает наклон текста нормальный, курсив или наклон прямого текста. Поясним, что в некоторых шрифтах нет курсивов и поэтому у них возможен только наклонный вариант. Например, шрифт Tahoma имеет только наклон. Это означает, что если браузер не находит наклонного начертания у шрифта, то он его сам «наклоняет».

*Font-variant : normal | small-caps;*

Font-variant : small-caps; - означает, что строчные буквы свое начертание меняют на заглавные, а размер остается такой же. Вид для русского текста становится страшным. Поэтому такое правило, встречается очень редко.

*Line-height : normal | величина | %;*

Это правило задает высоту линии шрифта.

Есть пространство, симметрично откладываемое вверх и вниз, которое служит для того, чтобы строки текста между собой не слиплись. Это пространство называется размер линии (line-height). Соотношение размера шрифта и высоты линии для каждого шрифта уникально. Для таких шрифтов как Helvetica, Arial соотношение шрифта используется 120%. Высота линии на 20% выше, чем высота шрифта. Для Times New Roman – это 125 процентов. Для того, чтобы вернуть к исходному

значению мы применяем правило line-height:normal;, а правило line-height:200%; - задает соотношение линии и высоты шрифта 200%, тем самым делает строки редкими.

Можно использовать все 6 правил в одном:

*Font: font-style font-variant font-weight font-size font-family*

Простые правила перечисляются через запятую и порядок их важен. Среди этих пяти правил, два последних (font-size и font-family) обязательны! Без них написать правила font нельзя. Если пропустить 3 первых правила, то браузер поставит значения по умолчанию (normal). Правило line-height задается через слеш (/).

Например, font: bold 10/12px Arial;

Рассмотрим правила, которые относятся к тексту.

*Text-align : left | right | center | justify;*

Работает для блочных элементов и служит для выравнивания контента внутри.

*Text-decoration : none | overline (|) underline (|) line-through;*

Выделяет текст горизонтальной линией, где

*none* – не выделять текст линией;

*overline* – рисовать горизонтальную линию над текстом;

*underline* – рисовать горизонтальную линию под текстом;

*line-through* – перечеркнуть текст;

Если, например, написать a{text-decoration:none;}, то это правило уберет подчеркивание у ссылки.

*Text-indent : величина | %;*

Позволяет задавать отступ красной строки. Работает для блочных элементов.

Например, правило text-indent:50%; задает красную строку с середины поля. Можно задавать и отрицательные значения.

*Text-transform : none | capitalize | uppercase | lowercase;*

Это трансформация текста. Когда вы получаете макет сайта, в котором используется заглавное начертание всех заголовков, то не торопитесь включать capslock и набирать текст заглавными буквами. Используйте обычный набор. Потому, что есть возможности трансформации этого содержания.

Например,

text-transform : uppercase; - любой текст делает заглавными буквами.

text-transform : lowercase; - любой текст делает строчными буквами.

text-transform : capitalize; - каждое слово начинается с большой буквы.

Рассмотрим промежутки между символами и словами.

*letter-spacing : normal |величина;*

Данное правило задает промежуток между символами.

*word-spacing : normal |величина;*

Данное правило задает промежуток между словами.

*normal* – это обычное состояние.

*величина* – это значение в em, ex и px. Эти значения могут быть положительные (тем самым буквы или слова будут реже) и отрицательные (символы или слова будут сдавливаться).

*Vertical-align : baseline | sub | super | top | text-top | middle | bottom | text-bottom | величина | %;*

Это правило задает вертикальное выравнивание строчных элементов. По умолчанию vertical-align:baseline; это выравнивание текста по базовой линии.

*sub* – надиндекс;

*super* – подиндекс;

*text-top* – выравнивание по верхней границе текстовой строки;

*text-bottom* - выравнивание по нижней границе текстовой строки;

*middle* - выравнивание по середине текстовой строки;

*top* – для ячеек таблицы выравнивание по верхней границе;

*bottom* - для ячеек таблицы выравнивание по нижней границе;

В ячейках таблицы vertical-align означает выравнивание контента по границам ячейки.

*величина* – это значения в px, em и ex. Может принимать как положительные так и отрицательные значения.

*White-space:normal | pre | nowrap;*

Данное правило задает пробельные символы.

*normal* – в строчном контенте браузер пробельные символы сокращает и делает по ним перенос. В блочном контенте он их просто игнорирует;

*pre* – это иметация элемента pre в HTML. Напомним, что пробельные символы не сокращаются, а остаются как есть, никакого автоматического переноса не происходит;

*nowrap* – пробельные символы сокращаются, но для автоматического переноса не используются.

Рассмотрим правила, позволяющие задать фоновые цвета.

*Background-color: цвет | transparent;*

Задает фоновый цвет. По умолчанию все элементы на странице прозрачные (*transparent*).

*Background-image : none | url;*

Позволяет задать фоновую картинку. По умолчанию – *none* – нет фоновой картинки. Если необходимо задать картинку, то надо прописать следующее правило: background-image:url (picture.jpg);.

Рассмотрим понятие блочной модели.

У каждого элемента, согласно блочной модели есть 4 области (рис 5.26):

1. Область контента (белый прямоугольник). Именно в ней располагается внутреннее содержание какого-то элемента (текст, другие элементы).
2. Между этим содержимым и рамкой располагается расстояние, которое называется padding (внутренний отступ).
3. Сама рамка - border (красного цвета).
4. Margin (внешний отступ) – область, которая определяет взаимоотношения элемента с другими. Кроме этого она может влиять и на размеры элемента в определенных случаях.

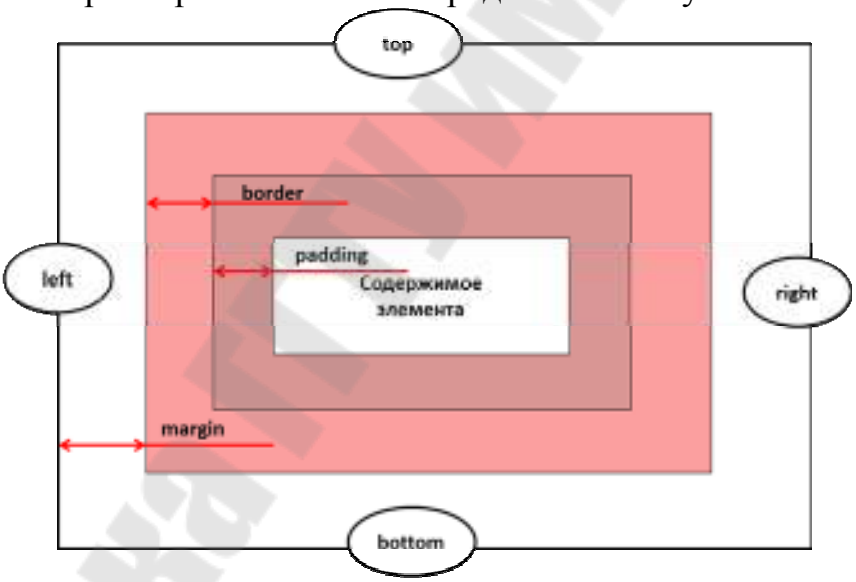



*Рисунок 5.26 – Блочная модель*

С каждой из этих областей связаны свои правила. Внешняя область – это правило margin. Отметим, что эта область всегда прозрачная. Рамка – это правило border. Внутренний отступ – правило padding. И область контента – это правила width и height, которые определяют размеры области с контентом.

Область **margin** имеет четыре стороны: top, bottom, right, left, что дает возможность указать для этих четырех сторон, по отдельности отступы следующими правилами:

*Margin-top : auto | величина | %;*

*Margin-bottom : auto | величина | %;*

*Margin-right : auto | величина | %;*

*Margin-left : auto | величина | %;*

*auto* – расстояние будет рассчитывать сам браузер автоматически. По горизонтали оно почти всегда равно 0.

По вертикали auto всегда 0.

*величина* измеряется в em, ex и px. Величина может иметь и отрицательное значение, это позволяет элементам «заходить» друг на друга.

У любого элемента есть понятие «контейнер», то есть это область, в которой этот элемент находится и относительно которой он берет размеры (позиционируется).

Если внутреннему элементу назначить margin в процентах, то этот процент берется относительно ширины области, в которой он лежит. Это очень часто используется в блочной верстке.

Есть возможность объединить все четыре правила в одно сборное правило margin:

*Margin: Margin-top Margin-right Margin-bottom Margin-left;*

В это правило через пробелы ставим четыре значения в строгой последовательности, начиная сверху по часовой стрелки, обходя все четыре стороны. Такая последовательность работает для любых правил, где есть четыре стороны.

Например, margin : 10px 20px 20px 30px;

Однако, на практике встречается не только четыре цифры, но и две и три и даже одна.

Можно написать только top, right и bottom. Это значит, что left будет такой же как и right. Если top и bottom равны и right и left равны, то можно написать два значения, top и right. Если написана одна цифра, то все значения одинаковые. Однако, если по горизонтали значения разные, то придется писать 4 цифры.

Рассмотрим внутренний отступ **padding**.

*Padding-top : величина | %;*

*Padding-right : величина | %;*

*Padding-bottom : величина | %;*

*Padding-left : величина | %;*

*Величина* задается в em, ex, px. В отличие от margin внутренний отступ бывает только положительным. Максимум, что можно сделать, это контент к рамке приблизить вплотную.

Проценты (%) работают так же как и для margin. Рассчитывается от ширины области, в которой находится элемент.

Padding по умолчанию равен 0. Все 4 значения можно записать в одно единственное правило. С помощью такого же принципа как и в правиле margin.

*Padding : Padding-top Padding-right Padding-bottom Padding-left;*

Рамки – правило **border.**

Рамки имеют три стилистических правила. Правила border можно задавать словами, а можно величиной (em, ex, xp).

*Border-width : величина* | (*thin*|*medium*|*thick*); - medium – средний (по умолчанию), thin – тонкий, thick – толстый.

*Border-color : цвет*; - указание цвета рамки. По умолчанию цвет такой же как у элемента. Если у элемента цвет зеленый, то и рамка будет зеленой.

*Border-style : none* | *dotted* | *dashed* | *solid* | *double* | *groove* | *ridge* | *inset* | *outset* ;

*none* – рамку не рисовать;
*dotted* – прерывистая линия (точка с пунктиром);
*dashed* - штриховая линия;
*solid* – сплошная линия;
*double* – двойная линия;
*groove* – имитация выпуклости рамки;
*ridge* – имитация выпуклости рамки;
*inset* – имитация того как элемент вжат в страницу;
*outset* – имитация того как элемент отжат от страницы.

Можно написать для каждой стороны комбинированное правило.

Border-top : (width | color | style);
Border-right : (width | color | style);
Border-bottom : (width | color | style);
Border-left : (width | color | style);

Например, нарисуем рамку одинаковую со всех сторон, только сверху она будет другого цвета.

border : border-width border-style border-color;
Border : 1px solid black;
Border-top : 1px solid blue;

Осталось рассмотреть область с контентом (**содержимое элемента**).

*Width: auto* | *величина* | *%;*
*Height : auto* | *величина* | *%;*

Эти два правила задают границы контента. По умолчанию эти правила принимают значения auto: браузер самостоятельно определяет высоту и ширину любых элементов. Для элементов уровня блок ширина auto.

Проценты ширины берутся относительно ширины области, в которой находится этот элемент. А проценты высота элемента берется от высоты области, в которой находится элемент.

Если одновременно задать width, height, padding, border, то эти значения складываются.

## 5.7 Наследование, каскадирование и специфичность

***Наследование.***

*Наследованием* называется перенос правил форматирования для элементов, находящихся внутри других. Такие элементы являются дочерними, и они наследуют некоторые стилевые свойства своих родителей, внутри которых располагаются.

Разберем наследование на примере таблицы. Особенностью таблиц можно считать строгую иерархическую структуру тегов. Вначале следует контейнер <table> внутри которого добавляются теги <tr>, а затем идет тег <td>. Если в стилях для селектора table задать цвет текста, то он автоматически устанавливается для содержимого ячеек, как показано в примере 5.41.

**Пример 5.41** – Наследование параметров цвета

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
        <title>Наследование</title>
        <style type="text/css">
            table{
                color: red; /* Цвет текста */
                background: #333; /* Цвет фона таблицы */
                border: 2px solid red; /* Красная рамка вокруг таблицы */
                }
        </style>
    </head>
<body>
    <table cellpadding="4" cellspacing="0">
        <tr>
            <td>Ячейка 1</td>
            <td>Ячейка 2</td>
        </tr>
        <tr>
            <td>Ячейка 3</td>
            <td>Ячейка 4</td>
        </tr>
```

```
        </table>
    </body>
    </html>
```

Таблица в этом случае примет вид как на рис. 5.3.


*Рисунок 5.27 – Наследование параметров цвета*

В примере 5.41 для всей таблицы установлен красный цвет текста, поэтому в ячейках он также применяется, поскольку тег <td> наследует свойства тега <table>. При этом следует понимать, что не все стилевые свойства наследуются. Так, border задает рамку вокруг таблицы в целом, но никак не вокруг ячеек. Аналогично не наследуется значение свойства background. Тогда почему цвет фона у ячеек в данном примере темный, раз он не наследуется? Дело в том, что у свойства background в качестве значения по умолчанию выступает transparent, т.е. прозрачность. Таким образом, цвет фона родительского элемента «проглядывает» сквозь дочерний элемент.

Чтобы определить, наследуется значение стилевого свойства или нет, требуется заглянуть в справочник по свойствам CSS (www.w3c.org в разделе CSS 2.1 (Appendix F. Full property table)).

Наследование позволяет задавать значения некоторых свойств единожды, определяя их для родителей верхнего уровня. Допустим, требуется установить цвет и шрифт для основного текста. Достаточно воспользоваться селектором body, добавить для него желаемые свойства, и цвет текста внутри абзацев и других текстовых элементов поменяется автоматически (пример 5.42).

**Пример 5.42** – Параметры текста для всей web-страницы

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Наследование</title>
    <style type="text/css">
        body {
            font-family: Arial, Helvetica, sans-serif; /* Гарнитура шрифта */
            color: navy; /* Синий цвет текста */
            }
    </style>
```

```
  </head>
<body>
  <p>Цвет текста этого абзаца синий.</p>
 </body>
</html>
```

В примере 5.42 рубленый шрифт и цвет текста абзацев устанавливается с помощью селектора body. Благодаря наследованию уже нет нужды задавать цвет для каждого элемента документа в отдельности. Однако бывают варианты, когда требуется все-таки изменить цвет для отдельного контейнера. В этом случае придется переопределять нужные параметры явно, как показано в примере 5.43.

**Пример 5.43** – Изменение свойств наследуемого элемента

```
<!doctype html  >
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-
8">
    <title>Наследование</title>
    <style type="text/css">
        body {
        font-family: Arial, Helvetica, sans-serif; /* Гарнитура
шрифта */
        color: navy; /* Синий цвет текста */
        }
        P.red {
        color: maroon; /* Темно-красный цвет текста */
        }
    </style>
  </head>
<body>
    <p>Цвет текста этого абзаца синий.</p>
    <p class="red">А у этого абзаца цвет текста уже другой.</p>
</body>
</html>
```

В данном примере 5.43 цвет первого абзаца наследуется от селектора body, а для второго установлен явно через класс с именем red.

***Каскадирование***

Аббревиатура CSS расшифровывается как Cascading Style Sheets (каскадные таблицы стилей), где одним из ключевых слов выступает

«каскад». Под каскадом в данном случае понимается одновременное применение разных стилевых правил к элементам документа — с помощью подключения нескольких стилевых файлов, наследования свойств и других методов. Чтобы в подобной ситуации браузер понимал, какое в итоге правило применять к элементу, и не возникало конфликтов в поведении разных браузеров, введены определенные приоритеты.

Ниже приведены приоритеты браузеров, которыми они руководствуются при обработке стилевых правил. Чем выше в списке находится пункт, тем ниже его приоритет, и наоборот.

Стиль браузера.

Стиль пользователя.

Стиль автора.

Стиль автора с добавлением !important.

Стиль пользователя с добавлением !important.

Самым низким приоритетом обладает *стиль браузера*. Оформление, которое по умолчанию применяется к элементам web-страницы браузером. Это оформление можно увидеть в случае «голого» HTML, когда к документу не добавляется никаких стилей. Например, заголовок страницы, формируемый тегом <H1>, в большинстве браузеров выводится шрифтом с засечками размером 24 пункта.

*Стиль пользователя*.

Это стиль, который может включить пользователь сайта через настройки браузера. Такой стиль имеет более высокий приоритет и переопределяет исходное оформление документа. В браузере Internet Explorer подключение стиля пользователя делается через меню Сервис > Свойство обозревателя > Кнопка «Оформление», как показано на рис. 5.28.

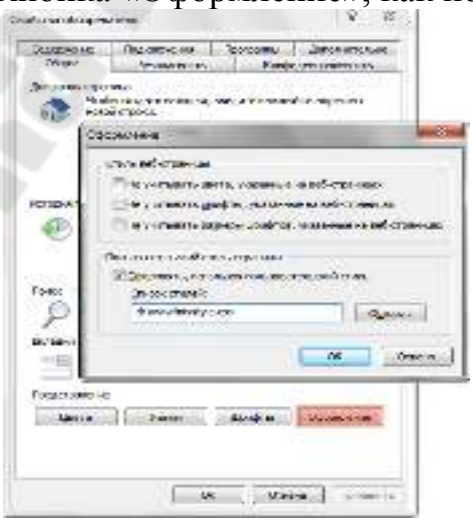


*Рисунок 5.28 - Подключение стиля пользователя в браузере Internet Explorer*

В браузере Opera аналогичное действие происходит через команду Инструменты > Общие настройки > Вкладка «Расширенные» > Содержимое > Кнопка «Параметры стиля» (рис. 5.29).

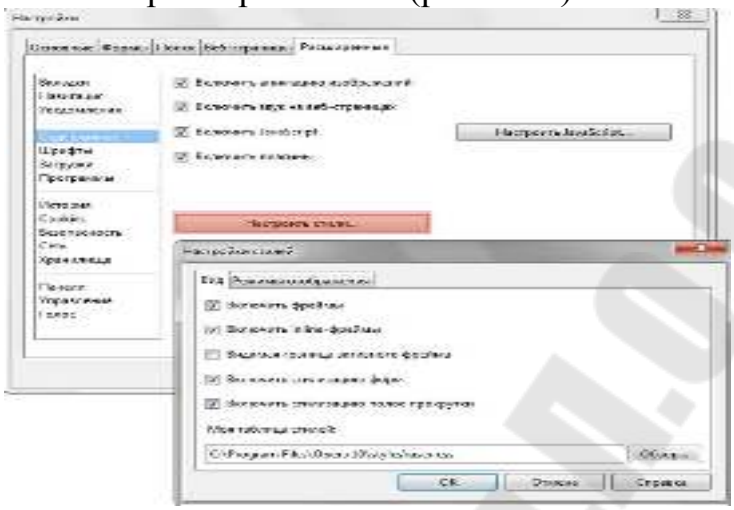


*Рисунок 5.29 - Подключение стиля пользователя в браузере Opera*

*Стиль автора*

Стиль, который добавляет к документу его разработчик.

*!important*

Ключевое слово !important играет роль в том случае, когда пользователи подключают свою собственную таблицу стилей. Если возникает противоречие, когда стиль автора страницы и пользователя для одного и того же элемента не совпадает, то !important позволяет повысить приоритет стиля или его важность, иными словами.

При использовании пользовательской таблицы стилей или одновременном применении разного стиля автора и пользователя к одному и тому же селектору, браузер руководствуется следующим алгоритмом.

!important добавлен в авторский стиль — будет применяться стиль автора.

!important добавлен в пользовательский стиль — будет применяться стиль пользователя.

!important нет как в авторском стиле, так и стиле пользователя — будет применяться стиль автора.

!important содержится в авторском стиле и стиле пользователя — будет применяться стиль пользователя.

Синтаксис применения !important следующий.

Свойство: значение !important

Вначале пишется желаемое стилевое свойство, затем через двоеточие его значение и в конце после пробела указывается ключевое слово !important.

Повышение важности требуется не только для регулирования приоритета между авторской и пользовательской таблицей стилей, но и для повышения специфичности определенного селектора.

**Специфичность**

Если к одному элементу одновременно применяются противоречивые стилевые правила, то более высокий приоритет имеет правило, у которого значение специфичности селектора больше. Специфичность это некоторая условная величина, вычисляемая следующим образом. За каждый идентификатор (в дальнейшем будем обозначать их количество через a) начисляется 100, за каждый класс и псевдокласс (b) начисляется 10, за каждый селектор тега и псевдоэлемент (c) начисляется 1. Складывая указанные значения в определенном порядке, получим значение специфичности для данного селектора.

```
*              {} /* a=0 b=0 c=0 -> специфичность = 0   */
li             {} /* a=0 b=0 c=1 -> специфичность = 1   */
li:first-line  {} /* a=0 b=0 c=2 -> специфичность = 2   */
ul li          {} /* a=0 b=0 c=2 -> специфичность = 2   */
ul ol+li       {} /* a=0 b=0 c=3 -> специфичность = 3   */
ul li.red      {} /* a=0 b=1 c=2 -> специфичность = 12  */
li.red.level   {} /* a=0 b=2 c=1 -> специфичность = 21  */
#t34           {} /* a=1 b=0 c=0 -> специфичность = 100 */
#content #wrap {} /* a=2 b=0 c=0 -> специфичность = 200 */
```

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

Если два селектора имеют одинаковую специфичность, то применяться будет тот стиль, что определен в коде ниже.

В примере 15.44 показано, как влияет специфичность на стиль элементов списка.

**Пример 15.44.** Цвет списка

```
<!doctype html  >
<html>
        <head>
                <meta  http-equiv="Content-Type"  content="text/html; charset=utf-8">
                <title>Список</title>
                <style type="text/css">
                        #menu ul li {
                        color: green;
                         }
                        .two {
                        color: red;
                        }
                </style>
```

```
    </head>
<body>
    <div id="menu">
        <ul>
            <li>Первый</li>
            <li class="two">Второй</li>
            <li>Третий</li>
        </ul>
    </div>
</body>
</html>
```

В данном примере цвет текста списка задан зеленым, а второй пункт списка с помощью класса two выделен красным цветом. Вычисляем специфичность селектора #menu ul li — один идентификатор (100) и два тега (2) в сумме дают значение 102, а селектор .two будет иметь значение специфичности 10, что явно меньше. Поэтому текст окрашиваться красным цветом не будет. Чтобы исправить ситуацию, необходимо либо понизить специфичность первого селектора, либо повысить специфичность второго (пример 5.45).

**Пример 5.45 –** Изменение специфичности

```
/* Понижаем специфичность первого селектора */
 ul li {...} /* Убираем идентификатор */
 .two  {...}

/* Повышаем специфичность второго селектора */
 #menu ul li {...}
 #menu  .two {...} /* Добавляем  идентификатор */
 #menu ul li {...}
 .two { color:  red !important; } /* Добавляем !important */
```

Добавление идентификатора используется не только для изменения специфичности селектора, но и для применения стиля только к указанному списку. Поэтому понижение специфичности за счет убирания идентификатора применяется редко, в основном, повышается специфичность нужного селектора.

## 5.8 Позиционирование элементов на странице и управление моделью элемента

При работе с элементами, предполагается, что элементы на web-странице будут идти в той же последовательности, в которой они стоят в коде. Это называется *нормальный поток*. Разделяют два правила, которые этот нормальный поток могут нарушить.

Это такие правила как:

1. *float : none | left | right;*
2. *position : static | absolute | relative;*

Рассмотрим эти правила подробнее.

Правило ***float*** определяет плаванье. Как только элементу поставить правило float: left|right;, то этот элемент становится плавающим, что означает, что э элементом происходит следующее:

- элемент становится блочным;
- плавающий элемент свои размеры определяет в соответствии со своим контентом. Это значит, что если в плавающем элементе контента нет, а ширина и высота были заданы, то он просто «схлопнится»;
- блочные элементы плавающие игнорируют (исключением являются таблицы), а строчные элементы их обтекают;
- плавающие элементы в схемах с margin colabs не участвуют.

Можно сделать так, чтобы блочные элементы, идущие в коде за плавающими, учитывали эти плавающие элементы. Для этого воспользуемся специальным правилом

*clear : none | left | right | both;*

С помощью этого правила блочные элементы учитывают (слева, справа, с обеих сторон) плавающий элемент.

По умолчанию float и clear имеют значения none.

Плавающие элементы встают максимально высоко и максимально в левую сторону или в правую в зависимости от правила float. А поскольку плавающие элементы друг друга учитывают, то это позволяет поставить их в ряд.

Правило ***position*** тоже серьезно меняет взаимоотношения элементов. Элемент с позиционированием можно двигать относительно каких-то областей.

*Position : static;* - это правило по умолчанию стоит у всех элементов. Применение этого правила, дает возможность поставить все элементы друг за другом.

*Position : relation;* - это относительное позиционирование. В этом случае нормальный поток не нарушается. Однако с помощью задания смещения (top, left, right, bottom), мы можем смещать этот элемент относительного своего положения.

*top : auto | величина | %;*
*left : auto | величина | %;*
*right : auto | величина | %;*
*bottom : auto | величина | %;*

Плавающие элементы реализуются для колонок.

*Position : absolute;* - позволяет позиционировать элемент относительно каких-то других элементов на странице.

Для строчных элементов ширина, высота и вертикальный margin игнорируются.

Следует запомнить несколько свойств:

– Любой элемент с Position : absolute; становится блочным.
– Его размеры определяются его контентом.
– Ни один элемент на странице, кроме корневого, его не учитывает.

У правила position есть еще одно значение – ***fixed***. По своему действию это значение близко к absolute, но в отличие от него привязывается к указанной свойствами left, top, right и bottom точке на экране и не меняет своего положения при прокрутке веб-страницы. Браузер Firefox вообще не отображает полосы прокрутки, если положение элемента задано фиксированным, и оно не помещается целиком в окно браузера. В браузере Opera хотя и показываются полосы прокрутки, но они никак не влияют на позицию элемента.

Следующие правила используют для динамических эффектов.

*Правило* ***display*** *: none* | *block* | *inline* | *list-item* по умолчанию есть у любого элемента. Это тип представления элемента. Например, для элементов div, p, hi, display находится в значении block. Для таких элементов как span, em, strong правило display имеет значение inline. Для элементов li – display: list-item;. Значение none говорит о том, что элемент не будет показываться вообще.

*Правило* ***visibility*** *: hidden* | *visible* | *inherit;* - это видимость нашего элемента.

visibility : hidden; – элемент не показывается, но место под его резервируется.

visibility : visible; – значение по умолчанию.

*Правило* ***overflow*** *: auto* | *scroll* | *visible* | *hidden;*. Переполнение контентом возникает тогда, когда заданные размеры не соответствуют размерам содержания.

overflow : visible; – значение по умолчанию. Использование этого правила, приведет к тому, что все что выйдет за пределы контента будет показано и не будет влиять на другие элементы.

overflow : hidden; – контент, который не вместился, скроется и у пользователя отсутствует возможность просмотреть его.

overflow : scroll; – появятся полосы прокрутки, которые позволят увидеть весь текст.

overflow : auto; – полосы прокрутки появляются сами в зависимости от размеров контента. Если контент вмещается в область, то никаких полос прокрутки добавлено не будет.

*Правило **z-index** : auto | величина | inherit;*

В качестве величины используются целые числа (положительные, отрицательные и ноль). Чем больше значение, тем выше находится элемент по сравнению с теми элементами, у которых оно меньше. При равном значении z-index, на переднем плане находится тот элемент, который в коде HTML описан ниже. Хотя спецификация и разрешает использовать отрицательные значения z-index, но такие элементы не отображаются в браузере Firefox до версии 2.0 включительно. Кроме числовых значений применяется auto — порядок элементов в этом случае строится автоматически, исходя из их положения в коде HTML и принадлежности к родителю, поскольку дочерние элементы имеют тот же номер, что их родительский элемент. Значение inherit указывает, что оно наследуется у родителя.

*Правило **clip** : auto | rect (top right bottom left);*

Определяет ту часть элемента, которую вы собираетесь показать. В статических страницах это правило используется только при сложной верстке. Сразу появляется возможность для элементов с position : absolute | fixed; показать часть элемента.

В качестве значения clip выступает некоторая область. Планировалось, что таких областей будет много, однако на практике используется только прямоугольная область - rect. Например, clip : rect (10px 20px 30px 10px); Все значения вычисляются относительно самой верхней левой точки области, в которой выделяем необходимую часть.

*Правило **cursor**:* [*url('путь к курсору'),*] | [ *auto* | *crosshair* | *default* | *e-resize* | *help* | *move* | *n-resize* | *ne-resize* | *nw-resize* | *pointer* | *progress* | *s-resize* | *se-resize* | *sw-resize* | *text* | *w-resize* | *wait* | *inherit* ]

Устанавливает форму курсора, когда он находится в пределах элемента. Вид курсора зависит от операционной системы и установленных параметров. Прежде чем воспользоваться возможностью переделать вид курсора, решите, а будет ли он использоваться к месту. Многих пользователей подобные изменения могут ввести в заблуждение, когда, например, вместо традиционной руки, появляющейся при наведении на ссылку, возникает нечто другое. В большинстве случаев, лучше оставить все как есть.

По умолчанию у всех видимых элементах используется правило cursor : auto;

Бывают и другие значения.

*Cursor: crosshair;* - перекрёстная линия, оптический прицел. Данное правило используется там, где нужно четко попадать в какие-то координаты.

*Cursor: pointer;* - указатель-рука, которая является своеобразной подсказкой, что можно кликнуть. При наведении на элемент, у которого прописано правило cursor:pointed; курсор будет меняться на изображение руки.

*Cursor:help;* - вместо курсора появится курсов со знаком вопроса. Показывает, что пользователь может кликнуть на элемент и появится всплывающая подсказка.

*Cursor:wait;* - курсор в виде песочных часов.

*Cursor:move;* - подсказка, что пользователь может передвигать элемент.

*Cursor:text;* - вид вертикальной черты при наведении на текст. Означает, что текст можно выделить и что-то с ним сделать.

Есть значения, которые говорят, как наш курсор расположить относительно сторон света. Всего 8 направлений: ***cursor****: e-resize* | *n-resize* | *ne-resize* | *nw-resize* | *s-resize* | *se-resize* | *sw-resize* | *w-resize;*.

## 5.9 Вопросы для самоконтроля

1. Какие три типа CSS Вы знаете?
2. В каком месте Web-страницы записываются встроенные CSS?
3. Где располагается описание внешних CSS?
4. В каком месте страницы описываются внедренные CSS?
5. Какое основное достоинство каскадных таблиц стилей?
6. Какое расширение должен иметь файл с описанием внешних CSS?
7. Для чего используются классы?
8. Какие программы используются для автоматизации создания таблиц стилей?
9. Какова зона действия встроенных CSS?
10. Какова зона действия внедренных CSS?
11. Какова зона действия внешних CSS?
12. Как с помощью CSS создать псевдографическое изображение?
13. Что такое стиль?
14. Почему таблицы стилей называют каскадными?
15. Какой тип CSS следует использовать, если предполагается изменение формата сразу нескольких страниц?
16. Какой из трех типов CSS следует считать наименее перспективным?
17. Как выполнить группировку селекторов?

# 6 Использование стилей при создании сайта

## 6.1 Верстка на CSS. Создание блока

В этом разделе вы узнаете об одном из способов создания стандартного блока на основе CSS, состоящего из «шапки» и основной части.

Для этого потребуется выполнить ряд простых действий:

1) Для удобства вынесем CSS в отдельный файл (см. раздел 5.1).

2) Вставьте в этот файл код, приведенный в примере 6.1.

3) Чтобы увидеть работу стилей, создадим в той же директории html-файл с кодом как в примере 6.2.

4) Сохраняем и смотрим, что получилось.

Для удобства чтения и лучшего понимания кода, мы прописываем пояснения к его командам в комментариях.

**Пример 6.1** – Создание блока на CSS (css файл)

```
/* Задание стилей всего блока */
#block{
        width: 250px; /* Задание ширины блока */
        }

/* Задание стилей заголовка */
.head {
        text-align:center; /* Выравнивание заголовка по центру блока */
        color: #fff; /* Задание цвета заголовка (тут - белый) */
        background-color: #0274b0; /* Задание цвета фона (тут - синий)
*/
        border: 2px solid #ffba00; /* Задание сплошной границы блока
шириной в 2 пикселя и её цвета */
        font-size: 15px; /* Задание размера шрифта заголовка */
        font-weight:bold; /* Задание полужирного начертания шрифта
*/
        padding: 7px 0 7px 0; */ Задание верхнего и нижнего отступов
текста заголовка от границ блока */
        }

/* Задание стилей основного блока */
.body {
         color:#333; /* Задание цвета текста */
        background-color: #d2efff; /* Задание цвета фона */
        border: 2px solid #ffba00; /* Задание сплошной границы блока
шириной в 2 пикселя и её цвета */
```

```
border-top-style: none; /* Удаление верхней границы блока */
font-size: 12px;  /* Задание размера шрифта */
padding: 5px; /* Задание отступа в 5 пикселей со всех сторон */
}
```

**Пример 6.2** – Создание блока на CSS (html-файл)

```
<html>
<head>
    <link rel="stylesheet" href="style.css" type="text/css" />
</head>
<body>
    <div id="block">
        <div class="head">
            ЗАГОЛОВОК БЛОКА
        </div>
        <div class="body">
        ОСНОВНОЙ БЛОК.<br>
        Текст блока….
        </div>
    </div>
</body>
</html>
```

Если ошибок при написании кода вами допущено не было, то в окне браузера вы увидите, блок представленный на рис. 6.1.

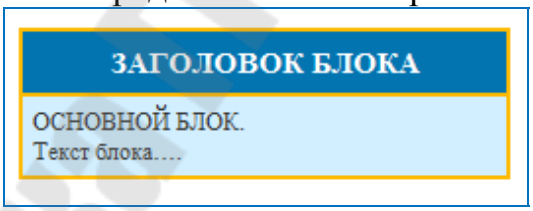



*Рисунок 6.1 – Стандартный блок на CSS*

## 6.2 Создание текста в три колонки

В данном разделе поговорим об одном из самых простых способов вёрстки в несколько столбцов, без использования таблиц. Порядок действий тут такой же как и в разделе 6.1. Здесь мы лишь приведем содержание css-файла (пример 6.3) и html-файла (пример 6.4).

**Пример 6.3** – Создание текста в три колонки (css-файл)

```
/* Задание стилей всего блока */
#body {
    width:900px; /* Установка ширины блока в 900 пикселей */
    }
```

```
/* Задание стилей левого столбца */
#left {
        float:left; /* Установка обтекания */
        width:300px; /* Установка ширины столбца */
        background:#aeddff; /* Установка фонового цвета */
        height: 300px; /* Установка высоты столбца */
        }

/* Задание стилей правого столбца */
#right {
        float:right; /* Установка обтекания */
        width:300px; /* Установка ширины столбца */
        background:#ffecbe; /* Установка фонового цвета */
        height: 300px; /* Установка высоты столбца */
        }

/* Задание стилей среднего столбца */
#center {
        margin-left:300px; /* Установка отступа */
        margin-right:300px; /* Установка отступа */
        background:#beffc0; /* Установка фонового цвета */
        height: 300px; /* Установка высоты столбца */
        }
```

**Пример 6.4** - Создание текста в три колонки (html-файл)

```
<html>
        <head>
                <link rel="stylesheet" href="style.css" type="text/css" />
        </head>
<body>
        <div id="body">
                <div id="left">Текст в левом столбце</div>
                <div id="right">Текст в правом столбце</div>
                <div id="center">Текст в среднем столбце</div>
        </div>
</body>
</html>
```

Способ довольно прост, тем не менее эффективен и гибок. Он допускает любое форматирование столбцов и их содержания.

В браузере перед вами появится вот такой вид, рис. 6.2.

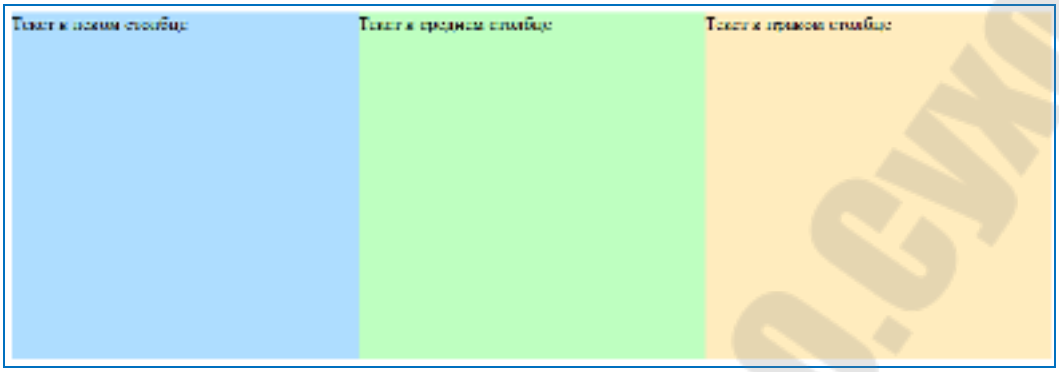


*Рисунок 6.2 – Создание три колонки*

В последующих двух разделах поговорим о различных способах создания шаблонов сайтов.

## 6.3 Современная верстка сайта при помощи CSS

Начнём с самого простого, стандартного типа шаблонов, рис. 6.3.

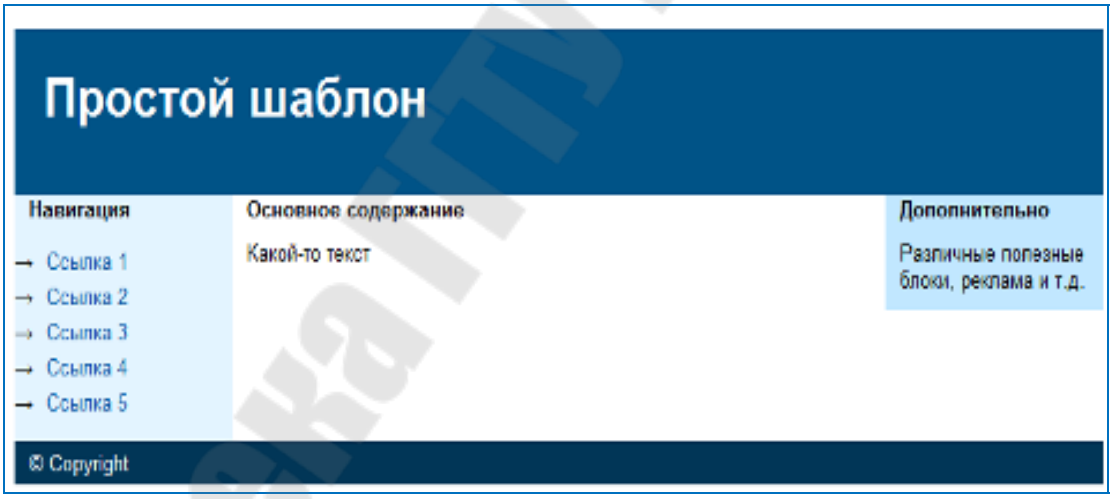



*Рисунок 6.3 – Пример шаблона сайта*

К особенностям шаблона на рис. 6.3 относятся:
- наличие 3-х колонок;
- стандартные поля заголовков;
- простое меню;
- отсутствие таблиц;
- отсутствие графики.

Этот шаблон будет очень удобен при изучении блочной вёрстки.

Как и раньше в примерах 6.5 и 6.6 приведем содержание css и html файлов соответственно с комментариями команд кода.

**Пример 6.5** – Простой шаблон (css-файл)

```
/* Задание стилей всего шаблона */
body {
        font: 80% Arial;
        text-align:center;
        }

/* Задание стилей новой строки */
p {margin:0 10px 10px;}

/* Задание стилей ссылок */
a {
    padding:5px;
    text-decoration:none;
    color:#0053a1;
}

/* Задание стилей ссылок при наведении */
a:hover {
    text-decoration:underline;
    color:#067a00;
}

/* Задание стилей блока заголовка */
div#header {
    background-color:#005387;
    color:#fff;
    height:80px;
    line-height:80px;
    padding-left:20px;
}

/* Задание стилей всего шаблона */
div#all {
    text-align:left;
    width:750px;
    margin:0 auto;
}
```

```
/* Задание стилей навигации */
div#navigation {
    background:#e3f4ff;
    float:left;
    width:150px;
    margin-left:-750px;
}

/* Задание стилей списка */
div#navigation ul {
    margin:15px 0;
    padding:0;
    list-style-type:none;
}

/* Задание стилей элемента списка */
div#navigation li{margin-bottom:5px;}

/* Задание стилей правого столбца */
div#extra {
    background:#c1e7ff;
    float:left;
    width:150px;
    margin-left:-150px;

}

/* Задание стилей "подвала" */
div#footer {
    background-color:#013657;
    color:#fff;
    clear:left;
    height:25px;
}

div#footer p {
    margin:0;
    padding:5px 10px;
}

/* Задание стилей всего шаблона */
div#templ {
```

```
    float:left;
    width:100%;
}

/* Задание стилей центрального столбца */
div#content {margin: 0 150px;}
```

**Пример 6.6** – Простой блок (html-файл)

```
<html>
        <head>
                <title>Шаблон сайта</title>
                <link rel="stylesheet" type="text/css" href="style.css" />
        </head>
<body>
<div id="all">
<div id="header"><h1>Простой шаблон </h1></div>
  <div id="templ">
    <div id="content">
      <p><strong>Основное содержание</strong></p>
<p>Какой-то текст</p>
    </div>
  </div>
  <div id="navigation">

    <p><strong>Навигация</strong></p>
    <ul>
      <li>&#8594; <a href="#">Ссылка 1</a></li>
      <li>&#8594; <a href="#">Ссылка 2</a></li>
      <li>&#8594; <a href="#">Ссылка 3</a></li>
      <li>&#8594; <a href="#">Ссылка 4</a></li>
      <li>&#8594; <a href="#">Ссылка 5</a></li>
    </ul>
  </div>
  <div id="extra">
    <p><strong>Дополнительно</strong></p>
    <p>Различные полезные блоки, реклама и т.д.</p>
  </div>
  <div id="footer">
    <p>© Copyright</p>
  </div>
</div>
</body>
```

</html>

## 6.4 Приемы макетирования web-страницы с использованием стилей

В этом разделе продолжим изучать способы создания простых шаблонов сайтов. Однако теперь приведем пример шаблона несколько сложнее и интереснее первого. Этот шаблон будет выполнен также без использования таблиц. Особенность создаваемого шаблона будет то, что он не имеет столбцов. Структуру шаблона создают блоки (div). На рис. 6.4 приведен макет будущего шаблона сайта.

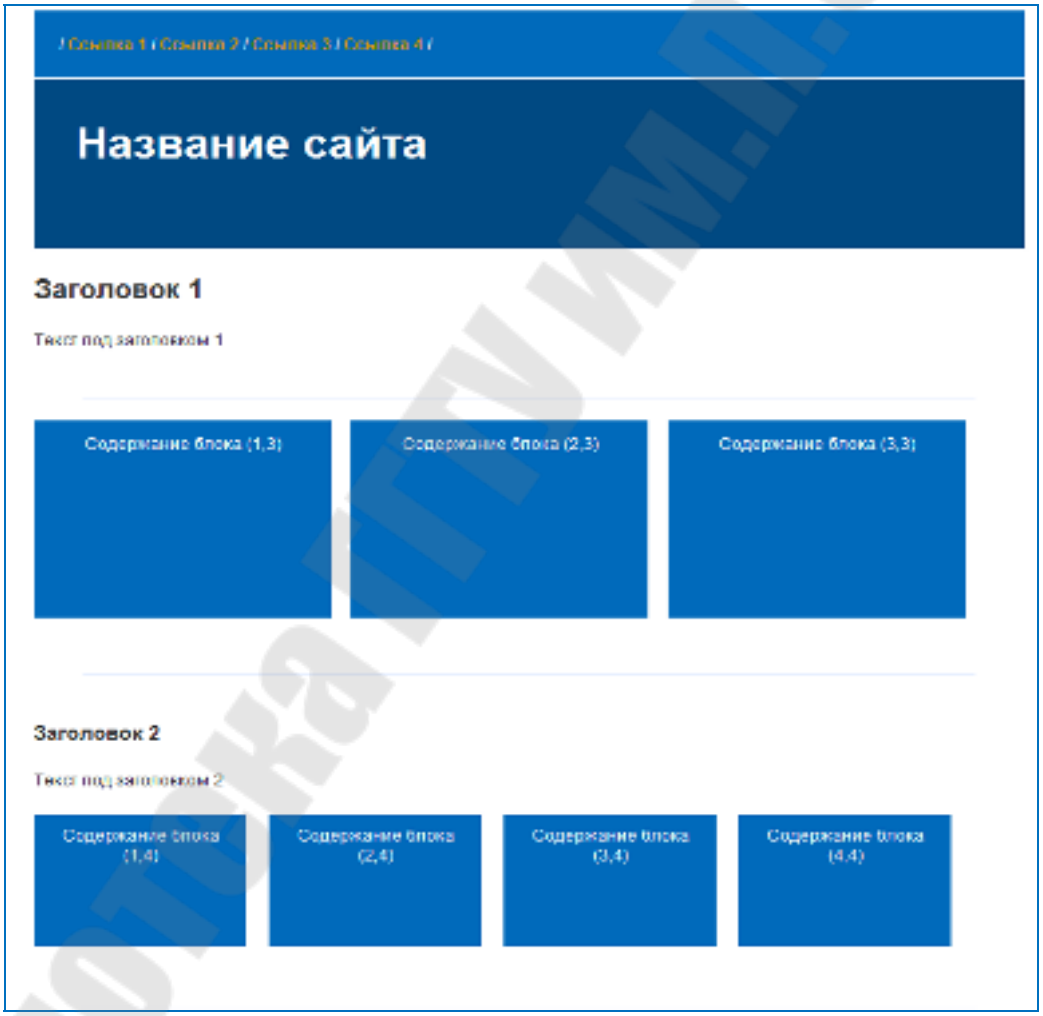



*Рисунок 6.4 – Макет сайта на CSS*

По макету на рис.6.4 прекрасно видно, что шаблон состоит из 11 блоков содержания и одного общего блока, стили которых предстоит задать. Постараемся последовательно, шаг за шагом, вместе с вами создать шаблон будущего сайта. Для этого представим вам html и css коды не

сразу, как они будут выглядеть окончательно, а постепенно дописывая его «части».

Вначале необходимо создать html файл, пример 6.7.

**Пример 6.7** – Создание общего блока

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
    charset=windows-1251">
        <link rel="stylesheet" type="text/css" href="style.css" />
        <title>Шаблон 2</title>
    </head>
<body>
    <div id="site">
```

Теперь нам надо задать стили для всего шаблона, его ссылок, линий и т.д. Запишем в CSS файл следующий код, пример 6.8.

**Пример 6.8** - Задание стилей общего блока

```
/* Задание стилей страницы */
html, body {
    margin:5px;
    color:#333;
    text-align:center;
    font-family:Arial;
    font-size:12px;
    }

/* Задание стилей всего шаблона */
#site {
    margin:0 auto;
    padding:10px;
    width:720px;
    text-align:left;
    }

#site h2 {
    font-size:20px;
    }

#site h3 {
    font-size:15px;
```

```
        }

/* Задание стилей ссылок всего шаблона */
a {
   color:#e39601;
   font-weight:bold;
   text-decoration:none;
}

a:hover {
        color:#fff;
        }

/* Задание стилей горизонтальных линий */
hr {
   margin:10px 0;
   border:0;
   width:90%;
   background-color:#bcd2ff;
   color:#bcd2ff;
   height:1px;
   clear:both;
   text-align:center;
}
```

Итак, созданы стили всего шаблона (общего блока). Теперь приступаем к созданию блока верхнего меню.

Продолжим html файл следующим кодом:

**Пример 6.9** – Добавление навигации

```
<div id="nav">
    / <a href="#">Ссылка 1</a> / <a href="#">Ссылка 2</a> / <a
    href="#">Ссылка 3</a> /
    <a href="#">Ссылка 4</a> /
</div>
```

А в CSS файл запишем такие стили:

**Пример 6.10** - Задание стилей блока навигации

```
#nav {
   background:#006abb;
   padding:18px;
```

```
    color:#fff;
    font-weight:bold;
}
```

Итак, блок навигации готов. Продолжим шаблон блоком заголовка. Для этого - запишем в html файл следующий код:

**Пример 6.11** – Создание блока заголовка

```
<div id="header">
        <h1>Название сайта</h1>
</div>
        <h2>Заголовок 1</h2>
<p>Текст под заголовком 1</p>
<hr />
```

Теперь запишем в CSS файл такие строки:

**Пример 6.12** - Задание стилей блока заголовка

```
#header {
   height:130px;
   padding:30px;
   background-color:#004982;
   border-top:1px solid #fff;
   color:#fff;
}
```

Пришло время для того, чтобы создать первые три блока содержания. Это довольно просто. Запишем в html файл такой код:

**Пример 6.13** – Создание первых трех блоков содержания

```
<div id="set1">
        <div id="cell_1">
                Содержание блока (1,3)
        </div>

        <div id="cell_2">
                Содержание блока (2,3)
        </div>

        <div id="cell_3">
                Содержание блока (3,3)
        </div>
```

```
</div>
<hr />
<h3>Заголовок 2</h3>
<p>Текст под заголовком 2</p>
```

Далее пишем в CSS файл такие стили:

**Пример 6.14** – Задание стилей блоков столбцов

```
/*Задание стилей блоков столбцов*/
#set1 {
  height:175px;
  color:#fff;
  text-align:center;
}

#set2 {
  height:130px;
  color:#fff;
  text-align:center;
}

/* Задание стилей блока 1 (3 столбца) */
#cell_1 {
   padding:10px;
   float:left;
   width:210px;
   background-color:#006abb;
   height:150px;
}

#cell_2 {
   margin:0 14px;
   padding:10px;
   float:left;
   width:210px;
   background-color:#006abb;
   height:150px;
}

#cell_3 {
   padding:10px;
   float:left;
```

```
    width:210px;
    background-color:#006abb;
    height:150px;
}
```

Важно заметить, что мы задали сразу ширину обоих строк блоков (см. #set1 и #set2 соответственно).

Аналогично с первой строкой блоков – создадим вторую и закроем наш html файл.

**Пример 6.15** – Создание второй строки блоков

```
<div id="set2">
    <div id="box_1">
        Содержание блока (1,4)
    </div>

    <div id="box_2">
        Содержание блока (2,4)
    </div>

    <div id="box_3">
        Содержание блока (3,4)
    </div>

    <div id="box_4">
        Содержание блока (4,4)
    </div></div>
    <hr />
</div>
</body>
</html>
```

Затем допишем в CSS файл следующие строки:

**Пример 6.16** - Задание стилей блока 2 (4 столбца)

```
#box_1 {
    padding:8px;
    float:left;
    width:150px;
    background-color:#006abb;
    height:100px;
}
```

```
#box_2 {
   margin-left:16px;
   margin-right:8px;
   padding:8px;
   float:left;
   width:150px;
   background-color:#006abb;
   height:100px;
}

#box_3 {
   margin-left:8px;
   margin-right:16px;
   padding:8px;
   float:left;
   width:150px;
   background-color:#006abb;
   height:100px;
}

#box_4 {
   padding:8px;
   float:left;
   width:150px;
   background-color:#006abb;
   height:100px;
}
```

# 7 Дополнительные возможности HTML и CSS.

## 7.1 Размещение мультимедийной информации на web-странице

Под мультимедиа, в первую очередь, понимают аудио и видео. Мультимедиа в приложении к web-дизайну – это аудио- и видеоролики, размещенные на web-страницах.

До недавних пор разместить на web-странице аудио- или видеоролик можно было только с помощью громадного HTML-кода и дополнительных программ. Но сейчас, с появлением HTML 5 и поддерживающих его (хотя бы частично) браузеров, потребуется всего один тег, не сложнее уже знакомого нам тега "img"!

Мы будем рассматривать работу с мультимедиа исключительно средствами HTML 5. Устаревшие способы (в частности, тег "object") описаны не будут.

Форматов мультимедийных файлов существует не меньше, чем форматов файлов графических. Это такое форматы как: wav, ogg, mp4, quicktime (один из первых). Как и в случае с интернет-графикой, браузеры поддерживают далеко не все мультимедийные форматы, а только немногие. Всю необходимую информацию о форматах можно найти в [3].

*Вставка аудио-ролика*

Для вставки на Web-страницу аудиоролика язык HTML 5 предусматривает парный тег <**audio**>, формат записи которого следующий:

<**audio** src="путь_к_файлу" **autoplay controls** ></**audio**>

Встретив тег <audio>, браузер может поступить следующим образом:

- сразу загрузить и воспроизвести аудиофайл;
- загрузить его без воспроизведения;
- ничего не делать. В этом случае мы можем запустить воспроизведение из web-сценария;
- вывести на web-страницу элементы управления, с помощью которых посетитель запускает воспроизведение аудиофайла, приостанавливает его, прокручивает вперед или назад и регулирует громкость. Все это настраивается с помощью различных атрибутов тега <audio>.

Перед тем как начнем рассматривать атрибуты тега <audio>, напомним, что этот тег блочный. Это означает, что вы не сможем вставить аудио-ролик на web-страницу в качестве части абзаца.

Рассмотрим атрибуты.

*src* – в качестве значения этого атрибута указывается путь к файлу вместе с расширение.

*autoplay* – позволит автоматически воспроизводить аудиоролик. По умолчанию браузер не будет воспроизводить аудио-ролик.

*controls* – добавит элементы управления воспроизведением аудиоролика, которые включают кнопку запуска и приостановки воспроизведения, шкалу воспроизведения и регулятор громкости. По умолчанию аудиоролик никак не отображается на web-странице.

*autobuffer* – позволяет браузеру сразу после загрузки web-страницы загружать файл аудиоролика, что позволит исключить задержку файла перед началом его воспроизведения. Данный атрибут имеет смысл указывать в теге <audio> только в том случае, если там отсутствует атрибут autoplay.

*Вставка видео-ролика*

Для вставки на web-страницу видеоролика предназначен парный тег <**video**>, формат записи которого следующий:

<**video src**="путь_к_файлу" **autoplay controls**
**poster=**"путь_к_файлу"></**video**>

Встретив тег <video>, браузер может поступить следующим образом:

- выведет в том месте web-страницы, где он проставлен, панель просмотра содержимого видеоролика;
- может сразу загрузить и воспроизвести аудиофайл;
- загрузить файл без воспроизведения;
- ничего не делать;
- может вывести на web-страницу элементы управления воспроизведением видеоролика.

Как и тег <audio>, тег <video> создает блочный элемент web-страницы и поддерживает атрибуты *autoplay*, *controls* и *autobuffer*.

Если воспроизведение видеоролика еще не запущено, в панели просмотра будет выведен первый его кадр или вообще ничего (конкретное поведение зависит от браузера). Но мы можем указать графическое изображение, которое будет туда выведено в качестве заставки. Для этого служит атрибут **poster** тега <video>; его значение указывает интернет-адрес нужного графического файла.

Если вы не найдете видеоролика подходящего формата, то можете сами создать его, перекодировав видеоролик, сохраненный в другом формате. Для этого подойдет утилита SUPER ©, которую можно найти по интернет-адресу http://www.erightsoft.com/ SUPER.html. Она поддерживает очень много мультимедийных форматов, в том числе и OGG.

*Дополнительные возможности тегов <AUDIO> и <VIDEO>*

Поскольку набор поддерживаемых мультимедийных форматов у разных браузеров различается, то может случиться так, что аудио- или видеоролик, который мы поместили на web-страницу, окажется каким-то браузером непонятым. Как же поступить в этой ситуации?

Специально для таких случаев HTML 5 предусматривает возможность указать в одном теге <audio> или <video> сразу несколько мультимедийных файлов. Браузер сам выберет из них тот, формат которого он поддерживает.

Если мы собираемся указать сразу несколько мультимедийных файлов в одном теге <audio> или <video>, то должны сделать две вещи.

1. Убрать из тега <audio> или <video> указание на мультимедийный файл, т. е. атрибут src и его значение.

2. Поместить внутри тега <audio> или <video> набор тегов <**source**>, каждый из которых будет определять интернет-адрес одного мультимедийного файла.

Одинарный тег <source> указывает как интернет-адрес мультимедийного файла, так и его тип MIME (подробно о типах MIME смотрите в [3]). Для этого предназначены атрибуты src и type данного тега соответственно, например:

```
<video>
    <source src="film.ogg" type="video.ogg">
    <source src="film.mov" type="video/quicktime">
</video>
```

Данный тег <video> определяет сразу два видеофайла, хранящих один и тот же фильм: один — формата OGG, другой — формата QuickTime. Браузер, поддерживающий формат OGG, загрузит и воспроизведет первый файл, а браузер, поддерживающий QuickTime, — второй файл.

Отметим, что тип MIME видеофайла (и, соответственно, атрибут TYPE тега "source") можно опустить. Но тогда браузеру придется загрузить начало файла, чтобы выяснить, поддерживает ли он формат этого файла. А это лишняя и совершенно ненужная нагрузка на сеть. Так что тип MIME лучше указывать всегда.

Если ваш браузер вообще не поддерживает теги <audio> и <video>, то в таком случае он их проигнорирует и ничего на web-страницу не выведет. Однако вы можете указать текст замены, в котором описать возникшую проблему и предложить какой-либо путь ее решения (например, сменить браузер). Такой текст замены ставят внутри тега <audio> или <video> после всех тегов <source> (если они есть).

## 7.2 Нестандартные теги HTML

Некоторые разработчики браузеров или поисковых систем включают поддержку собственных тегов, которые не входят в спецификацию HTML (См. таблицу 7.1). Использование этих тегов приводит к невалидному коду, а также поддерживаются не всеми браузерами.

**Таблица 7.1** – Нестандартные теги HTML

| Тег | Описание |
|---|---|
| <bgsound> | определяет музыкальный файл, который будет проигрываться на web-странице. |
| <blink> | устанавливает мигание текста. |
| <comment> | добавляет комментарий в код документа. |
| <marquee> | создает бегущую строку на странице |
| <multicol> | задает количество колонок, ширину и расстояние между колонками в многоколоночном тексте |
| <nobr> | уведомляет браузер отображать текст без переносов |
| <noembed> | предназначен для отображения информации на web-странице, если браузер не поддерживает работу с плагинами. |
| <plaintext> | отображает содержимое контейнера «как есть» |
| <spacer> | создает пустое пространство по вертикали или горизонтали |

## 7.3 Зачем нужен DOCTYPE?

Существует множество причин, по которым веб-страница может отображаться некорректно: отсутствие необходимых закрывающих тэгов, неверно отформатированные таблицы, ошибки JavaScript, ошибки в CSS, некорректные запросы к серверу и т.д. Зачастую такие ошибки - следствие того, что разработчик просто не проверяет свою работу, не проводит валидацию кода. В других случаях ошибки - следствие недостатка знаний и опыта при разработке веб-страниц.

Даже если вы сможете избежать все вышеперечисленные типы ошибок, есть еще кое-что, способное нарушить правильность отображения ваших страниц - это игнорирование использования тэга doctype или неправильное его использование.

Doctype сообщает, согласно каким правилам синтаксиса проводить проверку текущего документа. К примеру, для HTML 4.01 и XHTML 1.0 существует по три разных типа doctype, для HTML5 лишь один.

*Строгий doctype*

Применяется строгий синтаксис языка, допускается включать все теги и атрибуты, кроме осуждаемых.

Для HTML 4.01:

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

Для XHTML 1.0:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

Правила синтаксиса зависят от используемой версии, но в целом должны соблюдаться следующие.

1. Осуждается использование следующих тегов: <applet>, <basefont>, , <dir>, <font>, <isindex>, <noframes>, <plaintext>, <s>, <strike>, <u>, <xmp>. Взамен по возможности рекомендуется использовать стили.
2. Текст, изображения и элементы форм нельзя напрямую добавлять в <body>, эти элементы должны обязательно находиться внутри блочных элементов вроде <p> или .
3. Осуждается применение атрибутов target, start (тег <ol>), type (теги <li>, <ol>, <ul>) и др.

Естественно, также должен соблюдаться синтаксис языка — правильное вложение тегов, закрытие тегов, должны присутствовать обязательные теги и др.

*Переходный doctype*

Применяется «мягкий» синтаксис языка, допускается использовать все теги и атрибуты, включая осуждаемые.

Для HTML 4.01:

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

Для XHTML 1.0:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

Цель переходного doctype заключается в постепенном знакомстве с синтаксисом языка. Сразу использовать строгий синтаксис кому-то покажется слишком сложным, для таких людей и предназначен переходный HTML и XHTML. К тому же некоторые привычные вещи в строгой версии оказались под запретом, поэтому переходный doctype покажется панацеей для тех, кто не может отказаться от старых привычек. Речь, к примеру, идёт об атрибуте target, который позволяет открывать ссылку в новом окне. Использование target осуждается, в том числе из-за системы вкладок, ставших стандартом де-факто в браузерах. Одному пользователю нравится открывать ссылку в текущем окне, другому в новой вкладке, а третьему в новом окне. Чтобы поведение ссылки не зависело от прихоти разработчика, атрибут target запрещен при строгом doctype.

*Фреймы*

Применяется во фреймовой структуре в документе, по синтаксису аналогичен переходному doctypeу.

Для HTML 4.01:

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

Для XHTML 1.0:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

Данный doctype применяется только для главного документа, формирующего структуру фреймов с помощью тегов <frameset> и <frame>.

*HTML5*

В HTML5 существует лишь один тип doctype, который переводит браузер в стандартный режим.

```
<!DOCTYPE html>
```

## 7.4 Вопросы для самоконтроля

1. Что такое мультимедиа?
2. Как добавить аудио-ролик на web-страницу?
3. Как добавить видео-ролик на web-страницу?
4. Какие вы знаете дополнительные возможности тегов <audio> и <video>?
5. Назовите нестандартные теги HTML.
6. Что такое doctype? Зачем нужно использовать doctype?
7. Перечислите три типа doctype для HTML 4.01 и XHTML 1.0.
8. Запишите doctype для HTML5.

# 8 Размещение сайта на сервере и его поддержка

## 8.1 Описание метаинформации сайта с помощью элемента META

Тег <**meta**> определяет метатеги, которые используются для хранения информации предназначенной для браузеров и поисковых систем. Например, механизмы поисковых систем обращаются к метатегам для получения описания сайта, ключевых слов и других данных. Разрешается использовать более чем один метатег, все они размещаются в контейнере <head>. Как правило, атрибуты любого метатега сводятся к парам «имя=значение», которые определяются ключевыми словами content, name или http-equiv.

Синтаксис описания метоинформации следующий:

```
<head>
    <meta content="...">
</head>
```

Элемент <meta> закрывающего тега не требует.

**Пример 8.1** – Использование тега <meta>

```
<!doctype html>
    <html>
        <head>
            <title>Тег META</title>
            <meta http-equiv="Content-Language" content="ru">
            <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
            <meta name="GENERATOR" content="Microsoft
FrontPage 4.0">
            <meta name="ProgId"
content="FrontPage.Editor.Document">
        </head>
    <body>
            <p>...</p>
    </body>
    </html>
```

Рассмотрим подробнее атрибуты тега <meta>.

**Charset.**

Задает кодировку документа. Более подробную информацию о кодировках Вы можете найти в [4]. Атрибут charset введен в HTML5 и предназначен для сокращения формы тега <meta>, которая задавала кодировку в предыдущих версиях HTML и XHTML.

Атрибут charset имеет следующий синтаксис: **<meta charset=**"кодировка">.

В качестве значения используется название кодировки, например UTF-8. Никаких значений по умолчанию не имеет.

**Пример 8.2 –** Применение атрибута charset

```
<!doctype html>
    <html>
        <head>
            <meta charset="utf-8">
             <title>Тег META, атрибут charset</title>
        </head>
     <body>
         <p>Документ</p>
    </body>
    </html>
```

**Content.**

Устанавливает значение атрибута, заданного с помощью name или http-equiv. Атрибут content может содержать более одного значения, в этом случае они разделяются запятыми или точкой с запятой.

Атрибут content имеет следующий синтаксис: <**meta content**="...">

Атрибут content является обязательным и в качестве значений выступает строка символов, которую надо взять в одинарные или двойные кавычки. Значений по умолчанию не имеет.

**Пример 8.3 –** Применение атрибута content

```
<!doctype html>
    <html>
        <head>
            <title>Тег META, атрибут content</title>
            <meta http-equiv="Content-Type"
content="text/html; charset=utf-8">
        </head>
    <body>
        <p>...</p>
    </body>
    </html>
```

**Http-equiv.**

Браузеры преобразовывают значение атрибута http-equiv, заданное с помощью content, в формат заголовка ответа http и обрабатывают их, как будто они прибыли непосредственно от сервера.

Атрибут http-equiv имеет следующий синтаксис: <**meta http-equiv**="...">.

В качестве значений может выступать любой подходящий идентификатор. Ниже приведены некоторые допустимые значения атрибута http-equiv:

*Content-Type* задает тип кодировки документа.

*Expires* устанавливает дату и время, после которой информация в документе будет считаться устаревшей.

*Pragma* задает способ кэширования документа.

*Refresh* позволяет загрузить другой документ в текущее окно браузера.

Атрибут http-equiv никаких значений по умолчанию не имеет.

**Пример 8.4 –** Применение атрибута http-equiv

```
<!doctype html>
    <html>
        <head>
            <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
            <title>Тег META, атрибут http-equiv</title>
            <meta http-equiv="expires" content="Sun, 01 Jan 2013 07:01:00 GMT">
        </head>
    <body>
        <p>...</p>
    </body>
    </html>
```

**Name.**

Устанавливает идентификатор метатега для пары «имя=значение». Одновременно использовать атрибуты name и http-equiv не допускается.

Атрибут name имеет следующий синтаксис: <**meta name**="...">.

В качестве значений выбирается любой подходящий идентификатор. Ниже приведены некоторые допустимые значения атрибута name:

*author* позволяет задать имя автора документа.

*description* позволяет задать описание текущего документа.

*keywords* позволет задать список ключевых слов, встречающихся на странице.

Никаких значений по умолчанию атрибут name не имеет.

**Пример 8.5 –** Добавление ключевых слов

```
<!doctype html  >
      <html>
            <head>
                  <title>Тег META, атрибут name</title>
                  <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
                  <meta name="keywords" content="HTML, META,
метатег, тег, поисковая система">
            </head>
      <body>
             <p>...</p>
      </body>
      </html>
```

## 8.2 Размещение сайта в интернете. Вопросы хостинга

Итак, вы создали сайт и думаете о том, как его разместить в интернете. Для этого нужно арендовать для сайта физическое место на сервере, где будут находиться его файлы и приобрести доменное имя или адрес, по которому он будет доступен.

*Хостинг* (англ. hosting) - услуга по предоставлению дискового пространства для физического размещения информации на сервере, постоянно находящегося в сети (как правило, Интернет). Говоря простыми словами, хостинг – это то место, где будет храниться сайт.

Обычно под понятием *услуги хостинга* подразумевают как минимум услугу размещения файлов сайта на сервере, на котором запущено программное обеспечение, необходимое для обработки запросов к этим файлам (web-сервер). Как правило, в услугу хостинга уже входит предоставление места для почтовой корреспонденции, баз данных, DNS, файлового хранилища и т. п., а также поддержка функционирования соответствующих сервисов.

Выделяют 3 причины для заказа хостинга:

1. Для того, чтобы сайт работал, файлы, из которых состоит сайт, должны располагаться на компьютере, имеющем постоянное подключение к сети Интернет;
2. Для обеспечения функционирования сайта необходимо также наличие специального программного обеспечения (web-сервер, сервер баз данных, DNS-сервер, SMTP-сервер);

3. Для обслуживания оборудования и программного обеспечения, поддерживающего работу сайта, необходим высококвалифицированный персонал.

Для переноса своего сайта на хостинг нужно, в первую очередь, зарегистрировать домен (доменное имя, например, mysite.by) и арендовать хостинг. *Регистрация доменного имени* - это услуга по закреплению уникального имени за конкретной организацией или физическим лицом на определенный срок (от 1-го года). Для того, чтобы сайт продолжал быть доступным необходимо оплачивать продление регистрации доменного имени.

Про бесплатный хостинг и бесплатный домен есть масса стаей в Интернете, рекомендуем их прочитать. Однако, если вкратце, то лучше не тратить на них свое время. Как правило, бесплатное сделано не для вашей выгоды.

Регистратор доменов и хостер, это, как правило, разные компании, но хостинг-провайдеры тоже предлагают купить у них доменное имя с небольшой наценкой или большой, или, наоборот, по заниженной цене. Также могут предложить домен в подарок, однако лучше купить все по отдельности.

Для переноса сайта нужно получить доступ на жесткий диск сервера, где он будет храниться. Это так называемый FTP доступ. При перемещении можно использовать специальные программы, как CuteFTP или FileZilla, но можно ограничиться и стандартным Total Commander. Сам процесс переноса представляет собой примерно тоже самое, что и стандартное перемещение файлов и папок на своем компьютере.

Вместе FTP доступом хостинг-провайдер должен сообщить и адреса своих ns серверов. Они понадобятся при регистрации домена, после того как их пропишем по обращению к домену браузер будет обращаться к файлам на хостинге.

Итак, процесс разворачивания сайта в сети Интернет следующий:

1) регистрация имени сайта (доменного имени)
   - выбор подходящего имени сайта, свободного для регистрации
   - оформление заявки
   - получение счета
   - оплата счета
2) выполнение заказа хостинга
   - выбор тарифного плана
   - оформление заказа
   - получение счета
   - оплата услуг по счету

– получение данных для управления хостингом и подробной инструкции, как начать работу

3) размещение сайта на сервере

Размещение сайта на сервере выполняется достаточно просто с помощью панели управления Plesk. Панель управления на русском языке.

## 8.3 Работа с FTP-клиентом

Многие, особенно начинающие пользователи Интернета имеют трудности с **FTP** (File Transfer Protocol, протокол передачи файлов). И это понятно - для работы с FTP в большинстве случаев нужна специальная программа, пользование которой на первый взгляд может показаться сложным.

Всё мы периодически скачиваем из интернета различные программы, музыку, фильмы и другие файлы. Мы редко задумываемся о том, что таится в недрах этого действия, где обитала программа до того, как поселилась в нашем компьютере. Но вдруг мы решаем создать свой сайт, и хостер предоставляет нам какую-то странную возможность загружать его на сервер по ftp. Или друг настраивает на своем компьютере какую-то программу, позволяющую нам скачивать у него файлы через интернет, и приглашает зайти к нему на FTP.

При помощи этого протокола Вы можете подключаться к FTP-серверам и производить различные действия с хранящимися на них файлами и папками: скачивать с сервера на свой ПК, загружать на сервер, создавать, редактировать, переименовывать, удалять, назначать права доступа. Работа с файлами на FTP-сервере во многом напоминает привычные действия с ними на Вашем компьютере.

Примеров использования FTP довольно много: загрузка web-страниц на сервер хостинга, скачивание музыки, фильмов и программ с общедоступных FTP-серверов и т.п.

*FTP-сервер* – это обычный компьютер, на котором установлено специальное программное обеспечение, позволяющее пользователям подключаться к нему и работать с хранящимися на нем файлами и папками подобно тому, как они это делают на своих собственных ПК. Вы можете подключаться к FTP-серверу свободно или по уникальным логину и паролю.

При работе с FTP широко используются два понятия: скачивание и закачивание. Скачивание (по-английски «download») означает процесс сохранения папок и файлов с FTP-сервера на Ваш компьютер. Закачивание (по-английски «upload») – это передача папок и файлов с Вашего компьютера на FTP-сервер.

Обычно каждой папке (реже - файлу) на FTP-сервере назначают права доступа: чтение, запись и выполнение. Чтение означает, что Вы

можете просматривать файл или содержимое папки. Запись позволяет изменять это содержимое. А выполнение дает возможность запускать исполняемые файлы и скрипты на сервере. Вы можете столкнуться с управлением правами доступа, например при разработке web-сайта, когда посетителям нужно запретить доступ в одни каталоги сайта и разрешить выполнение скриптов из других каталогов.

Для подключения к FTP-серверу необходима специальная программа, называемая *FTP-клиент* или *FTP-менеджер*. Простейшими примитивными FTP-клиентами являются браузеры и проводник Windows – для доступа к FTP-серверу в их адресной строке достаточно ввести ftp://имя сервера. web-браузеры и Проводник позволяют просматривать содержимое FTP-серверов и скачивать с них файлы. Однако с помощью браузера Вы не сможете загрузить файл или папку на FTP-сервер, а проводник Windows не поддерживает докачку файлов в случае обрыва связи. Поэтому для работы с FTP лучше использовать специально предназначенные для этого программы. Одни из них заточены для работы только с FTP, другие являются целыми программными комплексами и помимо подключения к FTP-серверам позволяют решать огромное количество повседневных компьютерных задач.

*Настройка FTP-клиента.*

Настройка FTP-клиента во всех файловых менеджерах выполняется примерно одинаково. Различия очень небольшие, и поэтому нет смысла описывать способы настройки каждого из них.

Выполнять настройку FTP клиента мы будем учиться на примере популярного файлового менеджера Total Commander.

Total Commander – программа условно бесплатная. Это означает, что Вы, как клиент, можете пользоваться ей бесплатно неограниченное время. Правда за то, что Вы будете использовать ее бесплатно, Вам придется терпеть маленькое неудобство. При каждом запуске программы нужно будет по подсказке нажать не одну из трех кнопок. Если Вам это надоест, можете за небольшие деньги ее купить.

Ниже описано как настроить FTP-клиент. Приведенные иллюстрации и объяснения помогут Вам настроить ftp- клиент без затруднений.

Для начала нужно запустить программу. На рис. 8.1 Вы видите окно программы Total Commander.

*Рисунок 8.1 - Окно программы Total Commander*

Оно имеет две рабочих области. В какой-нибудь части, например, в правой открываем папку с файлами, которые нужно выгрузить в интернет. Или наоборот, эта папка будет предназначаться для загрузки файлов из Интернета.

В любом случае в этом (правом) окне открыта папка, находящаяся на Вашем компьютере.

В таком случае в левом окне мы будем пытаться выполнить настройку FTP-клиента для установления связи с нужным нам сервером в Интернете.

Сделаем левое окно активным. Для этого нужно нажать мышкой на адресную строку левого окна, как на рис. 8.1 показано стрелкой.

Затем нужно нажать на кнопку "ftp" или нажать сочетание клавиш CTRL+F. На рис. 8.1 кнопка "ftp" обведена кружком. Должно появиться вот окно, как показано на рис.8.2.

*Рисунок 8.2 – Вид окна ftp*

Изначально в программе может быть уже выполнена настройка какого-то FTP соединения, а возможно и не одного. Их можно не трогать,

они не помешают. Мы будем выполнять настройку нового. Для этого нажимаем на кнопку "Добавить…", как показано стрелкой на рис. 8.2. После нажатия должно открыться окно вида как на рис. 8.3.

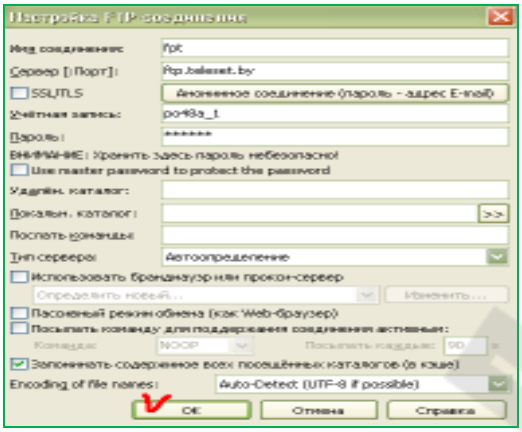



*Рисунок 8.3 – Настройка ftp-соединения*

Рассмотрим, как нужно настроить эти поля (рис.8.3), что они означают и что в них должно находиться.

*Имя соединения.* Здесь вы должны придумать имя для нового соединения, например, "ftp".

*Сервер* [: *Порт*]. В это поле вводим адрес ftp сервера, соединение с которым хотим настроить, например, ftp.teleset.by. Этот адрес вы сможете узнать у своего провайдера.

*Учетная запись.* В это поле вводится логин для доступа к удаленному серверу.

*Пароль.* В целях безопасности поле "Пароль" лучше оставить пустым. В этом случае пароль нужно будет указывать при каждом соединении.

*Удаленный каталог.* В это поле нужно ввести относительный (считая от корневой папки) адрес каталога на ftp сервисе, с которым вы хотите установить связь. Если вам нужно работать с несколькими папками вашего сайта, то для удобства работы можно настроить несколько ftp соединений с разными названиями. В таком случае в это поле будет вноситься относительный адрес конкретной папки. Например, соединение «Корневая папка» будет открывать каталог, в котором находится главная страница сайта, а соединение с названием «Мои фото» будет открывать папку, в которую вы загружаете свои фотографии.

*Посылать команду для поддержания ftp соединения активным.* Практически любой FTP-клиент через какое-то время теряет связь с сервером. Это делается специально в целях безопасности. Если вы хотите, чтобы FTP-соединение долгое время оставалось активным, поставьте галочку в этом поле. В этом случае FTP-клиент будет через указанные

промежутки времени посылать серверу команды, сохраняющие активность установленного соединения. Например, команду NOOP.

Остальные поля в большинстве случаев заполнять не нужно.

Когда поля заполнены, нажимаем кнопку "ОК", рис.8.3.

*Рисунок 8.4 – Соединение с ftp-сервером*

На этом настройка ftp клиента закончена. В окошке появилась строчка с названием созданного нами соединения, рис. 8.4. Для того чтобы воспользоваться созданным соединением, щелкаем по строке левой кнопкой мыши. Теперь, когда нужная строка выделена цветом, нажимаем на кнопку "Соединиться", рис. 8.4.

Разумеется, подключение к интернету должно быть выполнено.

Как видите, настройка FTP-клиента выполняется очень просто.

Если настроить FTP-клиент не удастся, возможно, что настройка не понадобится вообще. Вместо FTP- клиента можно использовать web-сервис.

## 8.4 Поддержка сайта

После размещения сайта в Интернете, его обязательно нужно развивать и поддерживать. Это является неотъемлемой частью существования сайта. Поддержка сайта, его обновление, наполнение это довольно кропотливый труд.

Поддерживая сайта, вы тем самым развиваете и поддерживаете его конкурентоспособность. Ведь Интернет очень динамичен и очень изменчив и требует к себе постоянного внимания.

Старый, давно не обновляемый сайт - это своего рода антиреклама. Человек пришедший на сайт и не получивший конкретный и актуальный ответ на свой запрос, скорее всего просто закроет сайт и продолжит поиск на других сайтах.

*Поддержка сайта* включает в себя:

- добавление, изменение текстов на сайте;
- добавление, изменение картинок, фотографий и прочей графики;
- круглосуточная поддержка работоспособности сайта;
- дополнение всевозможных модулей, компонентов, изменение функциональности сайта;
- управление вашей внутренней почтой;
- ежедневное резервное копирование сайта;
- восстановление вашего сайта;
- включение сайта в схему по перекрестному обмену ссылками (продвижение сайта).

Как видно из списка, поддержка сайта это работа комплексная, включающая в себя все возможные заботы о сайте.

## 8.5 Способы раскрутки сайта

Одна из самых важных задач в работе над web-сайтом это его *поисковое продвижение* или иными словами *раскрутка сайта*. Недостаточно просто создать сайт — необходимо сделать так, чтобы о его существовании знали. Это означает, что ваш сайт должен присутствовать в базе данных поисковой машины (tut.by, GoGo, Яндекс, Google и т.д.). Естественно, чем выше в результатах поиска находится ваш сайт, тем большее количество посетителей (потенциальных потребителей ваших товаров или услуг) он привлекает. Значимыми являются первые 10 позиций, выдаваемых поисковой машиной. Остальная выдача, безусловно, тоже имеет значение, но пользователи заходят на них в 5–6 раз меньше.

Безусловно, о продвижении своего сайта лучше подумать заранее. В идеале об этом подумать еще до создания первой страницы. К сожалению, многие web-мастера, особенно, начинающие, сперва создают сайт, наполняют контентом и лишь потом понимают, сколько работы по переделке тех или иных моментов им предстоит.

Итак, чтобы продвинуть свой сайт по позициям в поисковых системах и привлечь больше посетителей, следует составить подробный план действий и знать, с чего начинать. В принципе, раскрутка любого ресурса строится на трех китах – формирование ключевых запросов, внутренняя оптимизация и внешняя оптимизация.

*Формирование ключевых запросов.*

Система поиска состоит из запросов и ответов-ссылок. Чем больше страниц сайта присутствуют в выдаче популярных поисковиков, тем больше людей приходят на сайт. Естественно, если ресурс присутствует в топе Yandex или Googl по нужным ключам (запросам), то отдача от него будет выше. Как этого добиться? В первую очередь, сформировать эту самую базу запросов.

Сделать это можно с помощью сервиса *wordstat.yandex.ru* в Яндексе. Проанализировав правую и левую колонки, вы можете составить внушительный список ключей, как по высокочастотным запросам, так и по низкочастотникам. Еще один способ – более трудоемкий, но не менее эффективный – посмотреть, кто в топ-10 по данному ключу, и найти среди этих сайтов ресурс с открытой статистикой на поисковые запросы.

Начинающему сайту со скромным бюджетом не стоит замахиваться на популярные коммерческие запросы. На их продвижение тратятся огромные средства. А вот с помощью скромных низкочастотников можно собрать неплохую базу для привлечения целевого трафика. Причем, иногда лучше иметь в активе сотню топовых низкочастотных запросов, нежели один высокочастотный. При откате последнего на более дальние позиции вы сразу потеряете солидный процент посетителей. В то время, как вылет одного низкочастотника пройдет безболезненно.

*Внутренняя оптимизация*

Эту часть работы лучше начать с серьезного аудита сайта. Конечно, лучше, если его проведет опытный специалист. Но для начала можно попробовать различные программы, которые выявят те или иные ошибки в работе ресурса.

Далее переходим к оценке контента. Под нашу базу ключевых запросов должны присутствовать конкретные страницы с определенными статьями. Главная страница сайта обычно оптимизируется под высокочастотный запрос, отражающий основную тематику ресурса. Например, если речь идет о кулинарии, то и оптимизировать главную страницу нужно под этот или близкий к нему общий запрос, а не узкоспециальный – допустим, холодные закуски. Для конкретных ключей существуют внутренние страницы.

На главной странице хорошо бы иметь статью с плавным и ненавязчивым размещением в ней основных ключей. Помните, что поисковики не любят переспамленных страниц. Несколько раз упомянуть запрос в разных падежах, а также озаботиться его присутствием в title – будет достаточно. Тоже самое касается и низкочастотных запросов в статьях. Вообще, старайтесь оформлять страницы как можно лучше – изображения, таблицы, списки, подзаголовки, абзацы – поисковики это любят. И не пренебрегайте заполнением description и keywords.

Очень важной частью внутренней оптимизации является перелинковка страниц. Для наиболее верного решения нужно выбрать страницы, которые собираетесь продвигать и посредством внутренних ссылок с нужными якорями придать им больше веса. Кроме того, указание в статьях ссылок на свои же страницы с актуальной информацией – это увеличение количества визитов и уменьшение показателя отказов, то есть быстрого ухода посетителей с сайта.

*Внешняя оптимизация*

Уже давно прошли те времена, когда веб-мастер мог продвинуть свой сайт по определенным запросам лишь с помощью наличия ключей в статьях. Сегодня вес страницы и ее видимость для поисковых систем во многом определяются количеством внешних ссылок на нее со сторонних ресурсов. Поэтому ищите пути "добыть ссылку".

В сети достаточно много способов получить ссылку бесплатно. Это может быть тематический форум, размещение ссылки на котором принесет не только пользу в плане продвижения запроса, но и новых посетителей. Или комментарии в блогах, анонсы в социальных сетях, а также сервисах закладок. Непременно под каждой статьей поставьте кнопки для автоматического размещения статей в самых популярных социальных сетях с указанием источника.

Кроме всего прочего, внешние ссылки на сайт можно покупать на специальных биржах, заказывать прогон вашего ресурса по каталогам сайтов и доскам объявлений. Правда, эти методы не рекомендуются для новых сайтов, так как слишком быстрый прирост внешних ссылок может вызвать обратную реакцию у поисковых систем, и ваш сайт попадет под пессимизацию.

На самом деле, универсальных рекомендаций для продвижения сайта не существует. Имеются правила и требования, которым должен отвечать любой качественный ресурс. Вытащить откровенно плохую работу архисложно и неоправданно дорого. А вот грамотный сайт – красивый и удобный (будь то магазин или просто тематический журнал) – гораздо легче, так как подобный ресурс всегда "обрастает" естественными ссылками.

И последний совет: постоянно ищите что-то новое – читайте специализированные форумы и блоги, интересуйтесь нововведениями поисковых систем, берите на заметку чужой удачный опыт. И результат не заставит себя ждать.

Перед тем как начинать раскрутку сайта - переведите его на платный хостинг, чтобы потом не пришлось жалеть, что не сделали это раньше! Нельзя сначала раскрутить сайт - а потом переводить его на платный хостинг - тогда вся раскрутка будет напрасной.

Более подробную информацию о раскрутке своего сайта, вы сможете найти в [4].

Для тех, кто ленится читать, предлагаем пошаговую инструкцию самостоятельной раскрутки сайта с нуля.

*1 шаг:* описывается, как правильно оформить страницу (если ваш сайт уже готов, то переходите к шагу 2).

Эффективная раскрутка - это когда ваш сайт отображается в топе поисковых систем, для этого нужно следующее:

1) Название сайта должно соответствовать запросу, по которому его будут искать через поисковые системы.

2) Эта же поисковая фраза должна несколько раз использоваться в тексте и выделяться определенными тегами (подробнее об этом читайте про Seo - продвижение сайта).

Например, если Вы продаете одежду - то называйте сайт: Купить одежду, Где купить одежду, Купить одежду недорого. А если Ваш сайт про заработок в интернете - то называйте: «Заработок в интернете», «Работа в интернете» или «Как заработать в интернете».

*2 шаг:* регистрация сайта в каталогах и социальных закладках.

Хотите регистрируйте сами, а если хотите сэкономить время воспользуйтесь сервисами:

1PC - для регистрации в каталогах. Есть и более дорогостоящие предложения. Бесплатно тоже самое можно сделать с помощью - Addpromo, но на это вы реально потратите очень много времени.

Cheaptop - для регистрации в социальных закладках. Есть и другие более дорогостоящие предложения: регистрация в каталогах сайтов и статей, постинг в доски объявлений - это по вашему усмотрению.

На этом этапе все зависит от того сколько вы готовы потратить на сайт. Но если у вас много свободного времени - то можете все делать самостоятельно. Первый раз, всё можно сделать самостоятельно, но потом поймете – проще заказать.

*3 шаг:* регистрация в поисковых системах.

Вот ссылки для регистрации:

http://webmaster.yandex.ru/ - регистрация в яндекс.

http://www.google.ru/intl/ru/addurl.html - регистрация в гугле.

http://www.rambler.ru/doc/add_site.shtml - регистрация в рамблере.

Можете зарегистрировать и в других поисковых системах, но реально пользователи приходят только с yandex и google.

*4 шаг:* набор ссылочной массы.

Именно с этого момента, на наш взгляд начинается самостоятельная раскрутка сайта.

На этом этапе все зависит от того что у вас за сайт и как вы его используете. Если хотите через сайт продавать какие-нибудь товары или услуги, то набирать ссылочную массу лучше через специальные биржи ссылок. Это, конечно не дешево, но зато даёт самый высокий результат!

Если же вы ещё не знаете, как использовать сайт, как через него зарабатывать, у вас простенький сайт на бесплатном хостинге или вы просто не хотите тратить деньги - тогда можно набирать ссылочную массу бесплатно. Для этого:

Во-первых, пишите про свой сайт на различных форумах - такая самостоятельная раскрутка сайта имеет двойную пользу: Вы получаете ссылочную массу плюс приводите пользователей на сайт через интересные сообщения.

Во-вторых, аналогично с социальными сетями - пишите про ваш сайт и создавайте группы. Обязательно оставляйте ссылки!

В общем, старайтесь оставить как можно больше ссылок на свой сайт! На самом деле достаточно кропотливая работа, можете использовать программу подачи объявлений на бесплатные доски – Addboard.

Установите на сайте блок виджетов. Наиболее популярные приведены на рис. 8.5:

*Рисунок 8.5 – Виджеты*

Сделать это не сложно - виджеты интегрированы почти в каждый конструктор сайтов.

Но если быть похитрее, то можно набрать ссылочную массу очень дешево. Сделать это можно с помощью сайтов на которых можно размещать задания за деньги. Всё просто - даете задание: оставить пост или ссылку на ваш сайт. За одну такую ссылку вы заплатите, например, от 200 до 1000 рублей. За 50000 рублей - можно получить 200 - 350 хороших ссылок. Поверьте такой способ выгоднее всего!

А вообще включайте воображение - самостоятельная раскрутка сайта это интеллектуальный работа – придумайте, что можно ещё сделать, чтобы пользователи оставляли ссылки о вашем сайте!

*5 шаг:* анализ и корректировка.

Это когда некоторые ваши страницы попали в поисковые системы по хорошим запросам, но всё еще находятся во второй десятке! Вот тут уже нужно подключать биржи ссылок - купите 2-3 ссылки с хороших ресурсов и Ваш сайт войдет в топ 10!

Биржи ссылок дают две возможности владельцам сайтов:

1. Заработать на продаже ссылок;
2. Раскрутить сайт в поисковых системах путем покупок ссылок с других сайтов.

Чтобы заработать на биржах ссылок у вас должен быть хороший сайт с высоким индексом цитирования и Pr, на нем должно быть как можно

меньше ссылок с других сайтов. А вот для раскрутки подойдет практически любой. Биржи ссылок в настоящее время самый эффективный инструмент раскрутки сайта.

## 8.6 Вопросы для самоконтроля

1. Для чего нужны метатеги?
2. Опишите атрибуты тега <meta> chartset, content, http-eguiv, name.
3. Что такое хостинг?
4. Опишите процесс размещения сайта в сети Интернет.
5. Что такое ftp-сервер и ftp-клиент?
6. Как настроить ftp-клиент?
7. Что включает в себя поддержка сайта?
8. Какие вы знаете способы раскрутки сайта?
9. В чем разница между внешней и внутренней оптимизацией?

# Список использованных источников

1 Нэш, К. Война Браузеров // эл. журнал: Сети / network world – № 01 – 1997.

2 Петюшкин, А. В. HTML в Web-дизайне // СПб.: БХВ-Петербург – 2004. – 400 с.: ил.

3 Дронов, В. HTML 5, CSS 3 и Web 2. Разработка современных web-сайтов / В. Дронов // СПб.: БХВ-Петербург, 2011. – 416 с.: ил. – (Профессиональное программирование).

4 Загуменов, А.П. Как раскрутить и разрекламировать Web-сайт в сети Интернет / А.П. Загуменов // М.: ДМК Пресс, 2005. – 384 с., ил.

**Тихоненко** Татьяна Владимировна

# ВЕРСТКА WEB-СТРАНИЦ

**Курс лекций**
**по одноименной дисциплине**
**для слушателей специальности 1-40 01 74**
**«Web-дизайн и компьютерная графика»**
**заочной формы обучения**

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 21.10.13.
Рег. № 6Е.
http://www.gstu.by