Министерство образования Республики Беларусь

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

Институт повышения квалификации
и переподготовки кадров

Кафедра «Информатика»

Д. А. Литвинов

ОСНОВЫ WEB-ПРОГРАММИРОВАНИЯ

КУРС ЛЕКЦИЙ
по одноименному курсу
для слушателей специальности
1-40 01 73 «Программное обеспечение
информационных систем»
заочной формы обучения

Гомель 2015

УДК 004.738.1(075.8)
ББК 32.973-018.2я73
Л64

Рекомендовано научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого
(*протокол № 5 от 29.12.2014 г.*)

Рецензенты: зав. каф. «Информационные технологии» ГГТУ им. П. О. Сухого
канд. физ.-мат. наук, доц. *К. С. Курочка*

Литвинов, Д. А.

Л64 Основы web-программирования : курс лекций по одноим. курсу для слушателей специальности 1-40 01 73 «Программное обеспечение информационных систем» заоч. формы обучения / Д. А. Литвинов. – Гомель : ГГТУ им. П. О. Сухого, 2015. – 145 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: https://elib.gstu.by. – Загл. с титул. экрана.

Курс лекций предназначен для обучения проектированию и разработки Web-сайтов. Описаны языки HTML и CSS, применяемые для создания содержимого и представления Web-страниц. Рассматриваются основных подходы к созданию Web-страниц, задания макетов сайтов, приемы создания интерактивных компонентов сайтов.

Для слушателей специальности 1-40 01 73 «Программное обеспечение информационных систем» заочной формы обучения ИПК и ПК.

УДК 004.738.1(075.8)
ББК 33.973-018.2я73

Содержание

HTML. Язык разметки гипертекста

1. Глобальные компьютерные сети

1.1 Интернет. Основные понятия, принципы функционирования

Интернет (Internet) – глобальная сеть, объединяющая компьютеры и сети, имеющие различную архитектуру, системное программное обеспечение и т.д. Для передачи информации из одного вида сетей в другой используются шлюзы (**gateway**) – устройства, служащие для объединения сетей с различными протоколами обмена. На стыке сетей располагаются маршрутизаторы (**router**) – устройства, определяющие маршруты пакетов.

Сеть Интернет построена на основе протокола **TCP/IP** (Transmission Control Protocol/Internet Protocol) состоящего из двух главных сетевых протоколов:

- **IP** (Internet Protocol) – межсетевой протокол (протокол маршрутизации, транспортный протокол). Определяет формат пакетов, формат адресов компьютеров сети, маршрут пакета, правила обработки пакетов маршрутизаторами и компьютерами сети.
- **TCP** (Transmission Control Protocol) – протокол контроля передачи данных. Обеспечивает надежность передачи данных и сборку всех пакетов в единое сообщение.

Помимо базовых протоколов существуют прикладные протоколы (высокоуровневые), отвечающие за функционирование служб Интернета (HTTP, FTP, SMTP, POP3, IMAP и др.).

Основным проколом Интернет является протокол передачи гипертекста HTTP (Hypertext Transfer Protocol) предназначенный для передачи гипертекстовых документов от сервера к клиенту. Сервер HTTP (Web сервер) находится в состоянии ожидания соединения со стороны клиента по порту 80 TCP, а клиент HTTP (Web браузер) является инициатором соединения. Все, к чему может получить доступ пользователь – называют ресурсами сети. Каждый ресурс имеет уникальный для адрес, называемый универсальным идентификатором ресурса URI (Universal Resource Identifier). В самом общем случае URI выглядит следующим образом:

protocol://user:password@host:port/path/file?parameters#fragment

Отдельные поля URI имеют следующее назначение:

- *protocol* - протокол, посредством которого получают доступ к ресурсу;
- *user* – имя пользователь;
- *password* - пароль для аутентификации;
- *host* - IP-адрес или имя сервера, на котором расположен ресурс;
- *port* - номер порта, на котором работает сервер, предоставляющий доступ к ресурсу;
- *path* - путь к файлу, содержащему ресурс;
- *file* - файл, содержащий ресурс;
- *parameters* - параметры для обработки ресурсом-программой;
- *fragment* - точка в файле, начиная с которой следует отображать ресурс.

1.2 Адресация в Интернете

Адреса есть у каждого компьютера работающего в сети – *цифровой адрес (IP-адрес)*. Компьютерам, постоянно работающим в сети, присваивается постоянный IP-адрес. Такие компьютеры называют **хостами**. Компьютерам, работающим в сеансовом режиме, IP-адрес присваивается на время работы в сети (один из свободных адресов). IPv4-адрес – это цифровой адрес, состоящий из четырех десятичных чисел [0 - 255], отделенных друг от друга точками (например, 155.240.100.23).

IPv6 (англ. Internet Protocol version 6) – новая версия протокола IP, призванная решить проблемы, с которыми столкнулась предыдущая версия (IPv4) при её использовании в интернете, за счёт использования длины адреса 128 бит вместо 32. После того, как адресное пространство в IPv4 закончится, два стека протоколов – IPv6 и IPv4 – будут использоваться параллельно. Пакеты протокола IPv6 состоят из управляющей информации необходимой для доставки пакета адресату и полезных данных которые требуется переслать. Управляющая информация делится на содержащуюся в основном фиксированном заголовке и содержащуюся в одном из необязательных дополнительных заголовках.

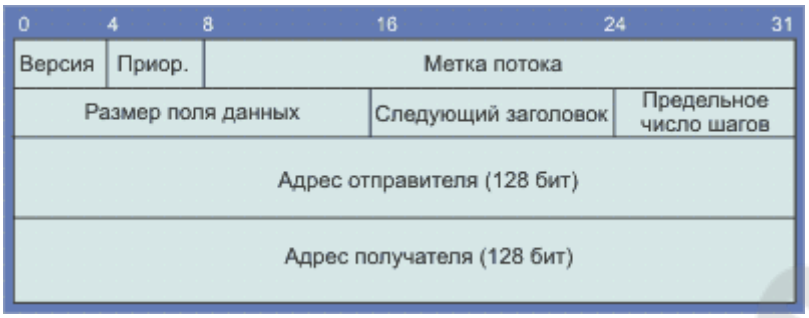


Рисунок 1.1 - Формат заголовка пакета IPv6

Человеку цифровые адреса неудобны, поэтому кроме цифровых адресов используются и символические адреса (например, www.tut.by). Одному IP-адресу соответствует один символический адрес. Чтобы символические адреса не повторялись, они регистрируются в ассоциации InterNIC.

Для преобразования символического адреса в цифровые и обратно, существуют специальные серверы – *DNS (Domain Name Server)*. Символические адреса строятся на основе иерархической системы, называемой *доменной*. **Домен** – группа хостов, объединенная по определенному признаку и имеющая одно имя. Домены первого уровня формируются по *территориальному* признаку (by – Белорусия, ru – Россия, su – СССР и др.) или *функциональному* признаку (gov – правительственные организации, mil – военные организации, edu – образовательные организации, com – коммерческие организации и др.). Домены второго уровня группируют хосты по территории или принадлежности одной организации.

Пример: **www.fais.gstu.by**
by – домен 1 уровня;
gstu.by – домен 2 уровня;
fais.gstu.by – домен 3 уровня.

1.3 Архитектура web-приложений

С появлением высокопроизводительных серверов, сетевого оборудования и высокоскоростных каналов связи стала реальностью организация корпоративных вычислительных сетей. Корпоративные сети объединены во всемирную глобальную сеть - Internet. Одним из крупнейших достижений Internet стала "всемирная паутина" - WWW (World Wide Web или просто Web). WWW представляет собой множество независимых, но взаимосвязанных серверов.

Согласно REC-html40-971218 – стандарту языка **HTML** 4.0 (RFC – Resource for Comments, так называются основные документы консорциума W3, специфицирующие технологии Internet), Web – это сеть информационных ресурсов, в которой для доступности этих ресурсов наиболее широкой аудитории используется три механизма:

– Единая схема именования ресурсов для поиска последних в Web - URI.
– Протокол для доступа к ресурсам через Web - HTTP.
– Гипертекст для перемещения по ресурсам - HTML.

Под Web-технологиями будем понимать всю совокупность средств для организации WWW. Поскольку в каждом сеансе взаимодействуют две стороны - сервер и клиент, Web-технологии естественно разделяются на две группы - технологии стороны сервера (server-side) и технологии стороны клиента (client-side). Клиентская обычно представляет собой Web – браузер, а серверная WEB – сервер. Взаимодействие между клиентом и сервером Web осуществляется путём обмена HTTP сообщениями. Веб-обозрева́тель, бра́узер (browser) - программное обеспечение для просмотра веб-сайтов, их обработки, вывода и перехода между страницами (Internet Explorer, Opera, Mozilla FireFox, Chrome, Safari и др.). Веб-сервер - это сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы, в виде HTML-страниц, изображений, файлов, медиа-потока или другими данными. Веб-сервером называют как программное обеспечение, выполняющее функции веб-сервера, так и непосредственно компьютер, на котором это программное обеспечение работает (Apache, IIS). Наиболее популярным Веб-сервером является Apache 85%, рынка.

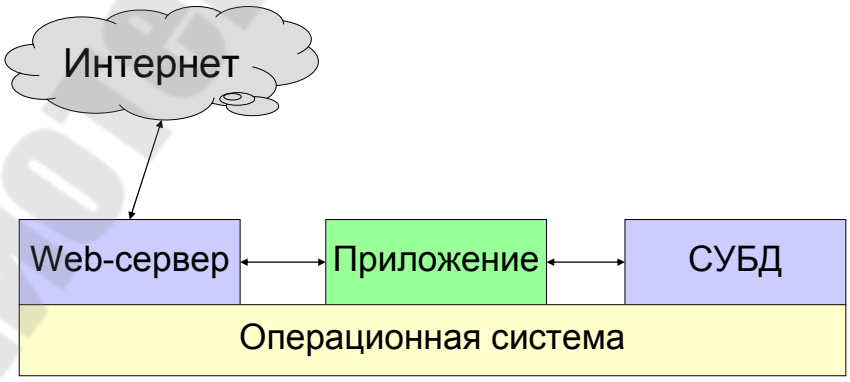


Рисунок 1.2 - Схема функционирования web-приложений на стороне сервера

Преимущества web:

- Независимость программных и аппаратных платформ на стороне клиента и сервера;
- низкие требования к программным и аппаратным средствам на стороне клиента;
- упрощение администрирования и обновления информации;
- упрощение разработки.

Недостатки web:

- относительно большое время отклика;
- недостаток интерактивности.

1.4 Современные WEB - технологии

Web технологии, сегодня, позволяют создавать Интернет проекты самого разного типа, сложности, а также целевой направленности. Они, постоянно совершенствуются и развиваются. Выделим основные технологии, применяющиеся для создания сайтов или других Интернет проектов.

Клиентские технологии:

- **HTML** – основной язык создания Web-страниц, используется для форматирования и разметки.
- **DHTML** (динамический HTML) - это набор средств, которые позволяют создавать интерактивные Web-страницы без увеличения загрузки сервера. DHTML построен на объектной модели документа (Document Object Model, DOM), которая обеспечивает динамический доступ к содержимому документа, его структуре и стилям. Каждый элемент Web-страницы является объектом, который можно изменять. DOM не определяет новых тэгов и атрибутов, а просто обеспечивает возможность программного управления всеми тэгами, атрибутами и каскадными листами стилей (CSS).
- **CSS** (Cascading Style Sheets) - позволяют осуществлять сложное форматирование используя каскадные таблицы стилей.
- **JavaScript** - используются для написания сценариев для активных HTML-страниц. Встраивается непосредственно в исходный текст HTML-документа и интерпретируется браузером по мере загрузки документа. С помощью JavaScript можно динамически изменять текст загружаемого HTML-документа и реагировать на со-

бытия, связанные с действиями посетителя или изменениями состояния документа или окна.

- **Macromedia Flash** – создание высококачественной интерактивной анимации, основанная на выполнении кода в клиентском приложении.
- **Java** - позволяет придавать Интернет странице интерактивность, создавать многочисленные активные элементы. Приложения Java компилируются в специальный байт-код и могут работать на любой виртуальной Java-машине (JVM) вне зависимости от компьютерной архитектуры.
- **ActiveX** - элементы управления ActiveX представляют собой динамически загружаемые библиотеки, выполняющиеся в адресном пространстве браузера. С помощью элементов управления ActiveX, как и посредством Java-аплетов, можно реализовать любую функциональность. Имеют большую функциональность и уровень доступа к локальным ресурсам. Элементы управления ActiveX применяются главным образом в интрасетях.
- **XML** (eXtensible Markup Language) - расширяемый язык разметки. Основное внимание в XML сосредоточено на данных. В XML структурная разметка данных и представление данных строго разделены.
- **XSLT** (eXtensible Stylesheet Language Transformations) - расширяемый язык преобразования листов стилей. Язык XSLT служит транслятором, с помощью которого можно свободно модифицировать исходный текст. Представляет собой универсальный язык хранения и передачи данных. Область применения XSLT широка - от электронной коммерции до беспроводного Web.
- **Ajax** (Asynchronous Javascript And XML «Асинхронные Javascript и XML») - в стандартном веб-приложении обработкой всей информации занимается сервер, а браузер отвечает только за взаимодействие с пользователем, передачу запросов и вывод поступившего HTML. Ajax дополнительный посредник, определяющий какие запросы можно обработать "на месте", а за какими необходимо обращаться на сервер.

Серверные технологии:

- **CGI** (Common Gateway Interface) – это спецификация обмена данными между прикладной программой, выполняемой по запросу пользователя, и HTTP-сервером, который данную программу

запускает. Обычно представляет собой исполняемый файл, выполняемый на стороне сервера.

- **PHP** – серверный язык создания сценариев. Конструкции на языке PHP встраиваются в HTML документ для придания странице интерактивности и интерпретируется специальным серверным модулем при обращении к странице. Результат работы внедряется в HTML документ, на место сценария.
- **ASP** (Active Server Pages) - технология создания веб-приложений и веб-сервисов от компании Майкрософт, с внедренными в них фрагментами кода, выполняемыми на стороне IIS сервера (Internet Information Server) ISAPI-библиотекой. Внедренный фрагмент кода замещается результатом его выполнения, а полученная таким образом динамическая страница передается в пользовательский браузер.
- **JSP** (Java Server Pages) – технология, позволяющая создавать содержимое, которое имеет как статические компонетнты (HTML, XML), так и динамические JSP элементы. Весь код страницы транслируется в java-код сервлета с помощью компилятора JSP страниц Jasper, и затем компилируется в байт-код виртуальной машины java (JVM). Контейнеры сервлетов, способные исполнять JSP страницы, написаны на языке Java. JSP является платформонезависимой, переносимой и легко расширяемой технологией для разработки веб-приложений. Назначение аналогично ASP. Основная идея которой – однократная компиляция Java-кода (сервлета) при первом обращении к нему, выполнение методов этого сервлета и помещение результатов выполнения этих методов в набор данных, отправляемых в браузер. Разработчик – Sun.

2. Язык гипертекстовой разметки страниц HTML

2.1 Язык гипертекстовой разметки страниц HTML

Основу WWW составляет гипертекстовый документ, создаваемый с помощью языка разметки гипертекстовых документов. **Гипертекст** – это расширенный текст, элементы которого могут храниться на различных ресурсах в сети, связь с которыми осуществляется с помощью гиперссылок.

HTML (Hyper Text Markup Language) - стандартный язык разметки документов во Всемирной паутине. Язык HTML интерпретируется браузерами и отображается в виде документа, в удобной для человека форме. HTML является описательным языком. В состав языка входят развитые средства для создания различных уровней заголовков, шрифтовых стилей, различные списки, таблицы, иллюстраций, аудио- и видеофрагментов и др. Современная версия языка 5.0.

Основной единицей языка HTML является тег. Теговая модель описывает документ как совокупность контейнеров, каждый из которых начинается и заканчивается тегами. Общая структура контейнера:

```
<"имя тега" "список атрибутов">
    содержание контейнера
</"имя тега">
```

Большинство тегов спарены т.е. за открывающим тегом следует соответствующий закрывающий тег, а между ними содержится текст или другие теги. В таких случаях два тега и часть документа, отделенная ими, образуют блок, называемый HTML элементом. Некоторые теги являются элементами HTML сами по себе, и в рамках спецификации HTML для них соответствующий конечный тег необязателен, однако для спецификации XHTML закрывать тег обязательно для валидности WEB документа.

Пример:

```
<I>Текст курсивом</I>
<p>обычный текст <b>Текст жирный</b> обычный текст </p>
<BR> разрыв строки <BR /> разрыв строки
```

Для большинства тегов определяется множество возможных атрибутов, однако атрибутов может и не быть. Конечные *теги* никогда не содержат атрибутов. Общий вид задания атрибута:

```
имени атрибута = значения атрибута
```

Пример: Задание таблицы шириной 570 пикселов, с выравниванием по центру, ширина бордюра 5 пикселов.

```
<TABLE WIDTH=570 ALIGN=center BORDER=5>
```

Кроме тегов, элементами HTML являются специальные символы CER (Character Entity Reference). Спецсимволы могут задаваться трехзначным кодом – **&#nnn** или именем элемента – **&имя**.

Таблица 2.1 – Специальные символы

Числовой код	Имя символа	Символ	Описание
&#034;	&quot;	"	Кавычка
&#038;	&amp;	&	Амперсанд
&#060;	&lt;	<	Меньше
&#062;	&gt;	>	Больше
			Неразрывный пробел
&#162;	&cent;	¢	Цент
&#163;	&pound;	£	Фунт
&#164;	&curren;	¤	Валюта
&#165;	&yen;	¥	Йена
&#168;	&uml;	¨	Умляут
&#169;	&copy;	©	Копирайт
&#171;	&laquo;	«	Левая угловая кавычка
&#174;	&reg;	®	Торговая марка
&#177;	&plusmn;	±	Плюс или минус
&#187;	&raquo;	»	Правая угловая кавычка

Все теги HTML по их назначению и области действия можно разделить на следующие основные группы:

- определяющие структуру документа;
- задающие оформление элементов документа;
- гипертекстовые ссылки и закладки;
- формы для организации диалога;
- вызов программ.

2.2 Структура HTML-документа

HTML-документ – представляет собой контейнер, начинающийся с тега <HTML> и заканчивается тегом </HTML>. Также документ включает два вложенных контейнера: заголовка документа (HEAD) и тела документа (BODY). Таким образом любой документ на языке HTML имеет следующею базовую структуру предоставления информации:

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

```
    <meta http-equiv="content-type" content="text/html; charset=utf-8">
    <title> название </title>
  </head>
  <body>
    содержание документа
  </body>
</html>
```

Рассмотрим основные теги подробнее.

2.2 Задание типа HTML-документа

Элемент <!**DOCTYPE**> предназначен для указания типа текущего документа. Это необходимо, чтобы браузер понимал, как следует интерпретировать веб-страницу, поскольку HTML существует в нескольких версиях, кроме того, имеется XHTML, похожий на HTML, но различающийся с ним по синтаксису. Существует несколько видов <!**DOCTYPE**>, они различаются в зависимости от версии языка, на которого ориентированы. В таблице 2.2 приведены основные типы документов с их описанием.

Таблица 2.2 – Основные типы документов

DOCTYPE	**Описание**
HTML 4.01	
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">	Строгий синтаксис HTML.
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">	Переходный синтаксис HTML.
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">	В HTML-документе применяются фреймы.
HTML 5	
<!DOCTYPE html>	Для всех документов.
XHTML 1.0	
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">	Строгий синтаксис XHTML.
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">	Переходный синтаксис XHTML.
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"	Документ написан на XHTML и содержит фреймы.

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">	
XHTML 1.1	
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">	Разработчики XHTML 1.1 предполагают, что он постепенно вытеснит HTML. Никакого деления на виды это определение не имеет, синтаксис один и подчиняется четким правилам.

2.3 Основные элементы заголовка HTML-документа

К основным элементам HTML-разметки заголовка относятся:

1. ***HEAD*** (элемент разметки *HEAD*);
2. ***TITLE*** (заглавие документа);
3. ***BASE*** (базовый URL);
4. ***META*** (метаинформация);
5. ***LINK*** (общие ссылки);
6. ***STYLE*** (описатели стилей);
7. ***SCRIPT*** (скрипты).

Элемент разметки **HEAD** содержит заголовок HTML-документа. Данный элемент разметки не является обязательным. Атрибутов у тега нет.

TITLE служит для задания имени документу. Атрибутов у тега нет. Синтаксис контейнера TITLE в общем виде выглядит следующим образом:

```
<TITLE>название документа</TITLE>
```

Элемент <**BASE**> не выводит никакого контента и выполняет исключительно служебную функцию – позволяет указать базовый URL, относительно которого будут устанавливаться другие адреса, например, для изображений и ссылок. Также <**base**> задаёт значение атрибута **target**, которое по умолчанию применяется ко всем ссылкам.

Пример. Задание базового адреса для документа.

```
<html>
 <head>
  <meta charset="utf-8">
  <title>Адреса</title>
  <base href="http://htmlbook.ru/">
 </head>
 <body>
  <p><img src="example/images/figure.jpg" alt=""></p>
```

```
 <p><a href="example/1.html">Ссылка</a></p>
 </body>
</html>
```

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. Атрибуты :

- **charset** – Задает кодировку документа.
- **content** – Устанавливает значение атрибута, заданного с помощью name или http-equiv.
- **http-equiv** – Предназначен для конвертирования метатега в заголовок HTTP.
- **name** – Имя метатега, устанавливает его предназначение.

Возможные применения:

1. Если в заголовке HTTP-сообщения указать оператор ***refresh*** то по истечении заданного интервала времени (атрибут **CONTENT**) браузер загружает документ, определенный атрибутом URL данного оператора. Таким образом можно реализовывать смену страниц через заданный интервал времени (слайд-шоу) или обновление.

Пример. Загрузка документа Page1.html через 1 сек:

```
<META HTTP-EQUIV="Refresh"  CONTENT="1;  URL=Page1.html">
```

Обновление текущей страницы каждые 3 сек:

```
<META HTTP-EQUIV="Refresh" CONTENT="3" >
```

2. Если в заголовке HTTP-сообщения указать оператор **Content-type**, с помощью атрибута **CHARSET** можно задать типа кодировки документа, с атрибутом **CONTENT** тип документа.

Приме. Задание кодировки документа

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

3. Заголовок HTTP – **Cache-Control** позволят управлять кэшированием документа и хранением документа на стороне клиента.

Пример. Запрещение кэширования:

```
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">
```

Запрещение хранение документа после пересылки:

```
<META HTTP-EQUIV="Cache-Control" CONTENT="no-store">
```

Так же можно задать время последней модификации (**Last-Modified**) имеет смысл только для статических страниц или дату истечения актуальности документа (**Expire**).

4. Описание поискового образа документа. Описания документа используется два **META**-тега. Один определяет список ключевых слов (**keywords**), а второй – реферат (**description**) (краткое содержание документа), который отображается в качестве пояснения к ссылке на документ в отчете *поисковой машины* о выполненном запросе.

Пример. Описание поискового образа документа

```
<META NAME="description" http-equiv="description"
content=" Канал Eurosport: все о спорте в режиме online, новости спорта: футбол, хоккей, теннис, баскетбол, бокс; Eurosport;">
<META NAME="keywords" HTTP-EQUIV="keywords"
CONTENT=" спорт новости спорта канал весь спорт спортивные игры футбол бокс хоккей баскетбол все спорте eurosport евроспорт ">
```

При индексировании можно указывать одновременно и атрибут **NAME**, и атрибут **HTTP-EQUIV** с одинаковыми значениями. Это связано с тем, что одни роботы индексирования анализируют содержание **META**-элемента по атрибуту **NAME**, а другие – по атрибуту **HTTP-EQUIV**.

Элемент разметки **LINK** используется для загрузки CSS – стилевых таблиц. Более подробно будет рассмотрен в разделе о стилевых таблицах.

Пример. Загрузка стилей из файла style.css

```
<LINK REL=stylesheet href="../css/style.css" TYPE="text/css">
```

SCRIPT предназначен для описания скриптов, может содержать ссылку на программу или ее текст на определенном языке (обычно **JavaScript**). Скрипты могут располагаться во внешнем файле и связываться с любым HTML-документом. Более подробно будет рассмотрен при изучении языка **JavaScript**. Синтаксис:

```
<script type="тип">  ...</script>
<script type="тип" src="URL"></script>
```

Пример. Загрузка скрипта из файла script.js

```
<SCRIPT TYPE="text/javascript" SRC=../js/script.js>
```

2.4 Теги форматирования HTML-документа

Теги тела документа предназначены для управления отображением информации и задания структуры документа. Описание тела HTML документа начинается с тега **BODY**. Тег определяет видимую часть документа. В этом разделе располагается вся содержательная часть документа (текст статьи, фотографии, формы для заполнения, другие объекты). Тег имеет ряд необязательных атрибутов:

- **bgcolor**= цвет – устанавливает цвет фона документа;
- **background** = «имя файла» – указывает на url-адрес изображения фона документа (файл с расширением *.gif или *.jpeg);
- **text** = цвет– устанавливает цвет текста документа;
- **link** =цвет – цвет гипертекстовых ссылок;
- **vlink**=цвет – цвет открытых гипертекстовых ссылок;
- **alink**=цвет – цвет "активных" гипертекстовых ссылок;

Пример. Задание атрибутов тега BODY.

```
<body background="fon.jpg" bgcolor="silver" text="blue" link="#44C1DC" vlink="#721CA4">
```

Цвет можно задавать как текстом, так и в формате RGB формате (см. таблицу 2.3).

Таблица 2.3 – Таблица цветов HTML

Предопределенное имя	Шестнадцатеричное значение	Цвет
aliceblue	#F0F8FF	
antiquewhite	#FAEBD7	
aqua	#00FFFF	
aquamarine	#7FFFD4	
azure	#F0FFFF	
beige	#F5F5DC	
bisque	#FFE4C4	
black	#000000	
blanchedalmond	#FFEBCD	
blue	#0000FF	
blueviolet	#8A2BE2	
brown	#A52A2A	
burlywood	#DEB887	

cadetblue	#5F9EA0	
chartreuse	#7FFF00	
chocolate	#D2691E	
coral	#FF7F50	
cornflowerblue	#6495ED	
cornsilk	#FFF8DC	
crimson	#DC143C	
cyan	#00FFFF	
darkblue	#00008B	
darkcyan	#008B8B	
darkgoldenrod	#B8860B	
darkgray	#A9A9A9	
darkgreen	#006400	
darkkhaki	#BDB76B	
darkmagenta	#8B008B	
darkolivegreen	#556B2F	
darkorange	#FF8C00	
darkorchid	#9932CC	
darkred	#8B0000	
darksalmon	#E9967A	
darkseagreen	#8FBC8F	
darkslateblue	#483D8B	
darkslategray	#2F4F4F	
darkturquoise	#00CED1	
darkviolet	#9400D3	
deeppink	#FF1493	
deepskyblue	#00BFFF	
dimgray	#696969	
dodgerblue	#1E90FF	
firebrick	#B22222	
floralwhite	#FFFAF0	
forestgreen	#228B22	
fuchsia	#FF00FF	
gainsboro	#DCDCDC	
ghostwhite	#F8F8FF	

gold	#FFD700	
goldenrod	#DAA520	
gray	#808080	
green	#008000	
greenyellow	#ADFF2F	
honeydew	#F0FFF0	
hotpink	#FF69B4	
indianred	#CD5C5C	
indigo	#4B0082	
ivory	#FFFFF0	
khaki	#F0E68C	
lavender	#E6E6FA	
lavenderblush	#FFF0F5	
lawngreen	#7CFC00	
lemonchiffon	#FFFACD	
lightblue	#ADD8E6	
lightcoral	#F08080	
lightcyan	#E0FFFF	
lightgoldenrodyellow	#FAFAD2	
lightgreen	#90EE90	
lightgrey	#D3D3D3	
lightpink	#FFB6C1	
lightsalmon	#FFA07A	
lightseagreen	#20B2AA	
lightskyblue	#87CEFA	
lightslategray	#778899	
lightsteelblue	#B0C4DE	
lightyellow	#FFFFE0	
lime	#00FF00	
limegreen	#32CD32	
linen	#FAF0E6	
magenta	#FF00FF	
maroon	#800000	
mediumaquamarine	#66CDAA	
mediumblue	#0000CD	

mediumorchid	#BA55D3	
mediumpurple	#9370DB	
mediumseagreen	#3CB371	
mediumslateblue	#7B68EE	
mediumspringgreen	#00FA9A	
mediumturquoise	#48D1CC	
mediumvioletred	#C71585	
midnightblue	#191970	
mintcream	#F5FFFA	
mistyrose	#FFE4E1	
moccasin	#FFE4B5	
navajowhite	#FFDEAD	
navy	#000080	
oldlace	#FDF5E6	
olive	#808000	
olivedrab	#6B8E23	
orange	#FFA500	
orangered	#FF4500	
orchid	#DA70D6	
palegoldenrod	#EEE8AA	
palegreen	#98FB98	
paleturquoise	#AFEEEE	
palevioletred	#DB7093	
papayawhip	#FFEFD5	
peachpuff	#FFDAB9	
peru	#CD853F	
pink	#FFC0CB	
plum	#DDA0DD	
powderblue	#B0E0E6	
purple	#800080	
red	#FF0000	
rosybrown	#BC8F8F	
royalblue	#4169E1	
saddlebrown	#8B4513	
salmon	#FA8072	

sandybrown	#F4A460	
seagreen	#2E8B57	
seashell	#FFF5EE	
sienna	#A0522D	
silver	#C0C0C0	
skyblue	#87CEEB	
slateblue	#6A5ACD	
slategray	#708090	
snow	#FFFAFA	
springgreen	#00FF7F	
steelblue	#4682B4	
tan	#D2B48C	
teal	#008080	
thistle	#D8BFD8	
tomato	#FF6347	
turquoise	#40E0D0	
violet	#EE82EE	
wheat	#F5DEB3	
white	#FFFFFF	
whitesmoke	#F5F5F5	
yellow	#FFFF00	
yellowgreen	#9ACD32	

Тег <**H1**> ... </**H1**> – <**H6**> ... </**H6**> описывает заголовки шести различных уровней. Заголовок первого уровня – самый крупный, шестого уровня, естественно – самый мелкий. Тег имеет ряд необязательных атрибутов:

ALIGN=CENTER | RIGHT | LEFT – задает выравнивание заголовка.

Пример. Заголовок первого уровня, с выравниванием по центру

```
<H1 ALIGN=CENTER>Выравнивание заголовка по центру </H1>
```

Теги <**P**> ... </**P**> – описывает абзац. Тег является блочным элементом, абзац текста всегда начинается с новой строки. Тег имеет ряд необязательных атрибутов:

ALIGN=CENTER | RIGHT | LEFT | justify – задает выравнивание заголовка.

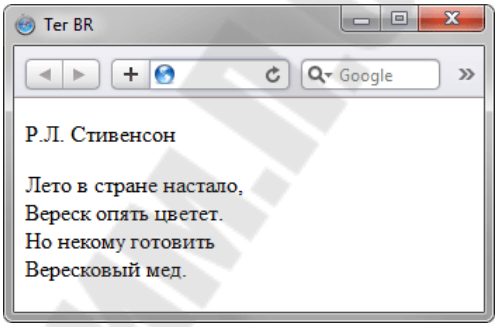
Пример. Абзац с выравниванием текста по ширине.

```
<P ALIGN= justify > выравнивание абзаца по ширине </P>
```

Тег <**BR**> – устанавливает перевод строки в том месте, где этот тег находится. В отличие от тега абзаца <**p**>, использование тега <**br**> не добавляет пустой отступ перед строкой. Если текст, в котором используется перевод строки, обтекает плавающий элемент, то с помощью атрибута **clear** тега <**br**> можно сделать так, чтобы следующая строка начиналась ниже элемента.

Пример.

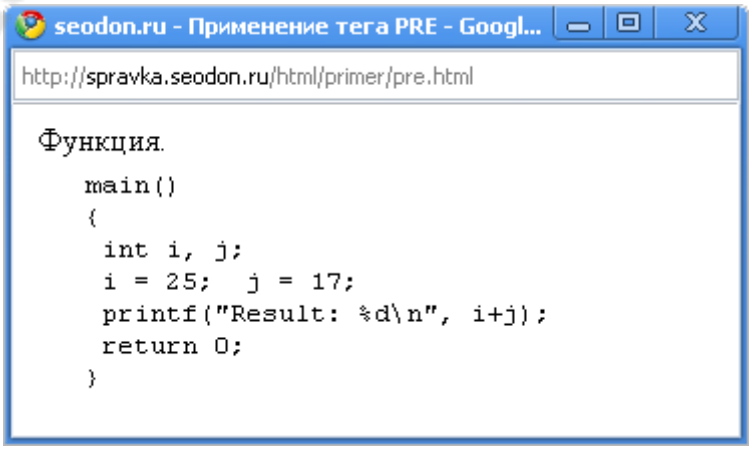
```
<body>
    <p>Р.Л. Стивенсон</p>
    <p>Лето в стране настало,<br>
     Вереск опять цветет.<br>
     Но некому готовить<br>
     Вересковый мед.</p>
 </body>
```



Тег <**PRE**> - определяет блок предварительно форматированного текста. Такой текст отображается обычно моноширинным шрифтом и со всеми пробелами между словами. По умолчанию, любое количество пробелов идущих в коде подряд, на веб-странице показывается как один. Тег <**pre**> позволяет обойти эту особенность и отображать текст как требуется разработчику. Внутри контейнера <**pre**> допустимо применять любые теги кроме следующих: <**big**>, <**img**>, <**object**>, <**small**>, <**sub**> и <**sup**>.

Пример.

```
<body>
 <p>Функция.</p>
 <pre>
  main()
  {
   int i, j;
   i = 25;  j = 17;
   printf("Result: %d\n", i+j);
   return 0;
  }
 </pre>
</body>
```



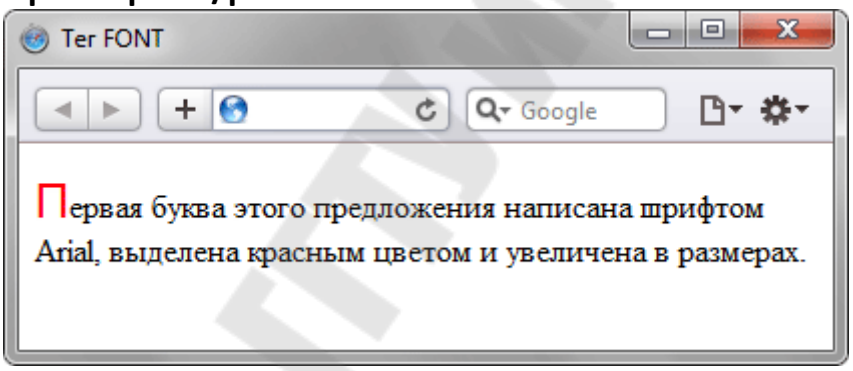
Тег <**FONT**>…</**FONT**> – представляет собой контейнер для изменения характеристик шрифта, таких как размер, цвет и гарниту-

ра. Хотя этот тег до сих пор поддерживается всеми браузерами, он считается устаревшим и от его использования рекомендуется отказаться в пользу стилевых таблиц. Атрибуты:

– **color=цвет** – Устанавливает цвет текста;
– **Face= «Гарнитура»** – Определяет гарнитуру шрифта: шрифты с засечками (антиквенные), типа **Times**; **sans-serif –** рубленные шрифты (шрифты без засечек или гротески), типа – **Arial**; **cursive –** курсивные шрифты; **fantasy –** декоративные шрифты; **monospace** – моноширинные шрифты, ширина каждого символа в таком семействе одинакова.;
– **Size=X** – Задает размер шрифта в условных единицах от 1 до 7. Размер по умолчанию 3.

Пример. Задание буквицы

```
<p><font size="5" color="red" face="Arial">П</font>ервая буква этого предложения  написана шрифтом Arial, выделена красным цветом и увеличена в размерах.</p>
```



Тег <**DIV**> – является блочным элементом и предназначен для выделения фрагмента документа с целью изменения вида содержимого. Как правило, вид блока управляется с помощью стилей. Закрывающий тег обязателен. Атрибуты:

– **align="center | left | right | justify"** – Задает выравнивание содержимого тега.
– **Title** – Добавляет всплывающую подсказку к содержимому.

Пример. Задание блока.

```
<div align=" center " title="ПОДСКАЗКА">
Текст, таблицы, изображения. Выравнивание по центру </div>
```

Дополнительные теги управляющие формой отображения и характеризующие описываемую информацию приведены в таблицах 2.4 и 2.5.

Таблица 2.4 – Теги, управляющие формой отображения

Тег	Значение
<I>...</I>	Курсив (Italic)
<B>...</B>	Усиление (Bold)
<TT>...</TT>	Телетайп
<U>...</U>	Подчеркивание
<S>...</S>	Перечеркнутый текст
<BIG>...</BIG>	Увеличенный размер шрифта
<SMALL>...</SMALL>	Уменьшенный размер шрифта
_{...}	Подстрочные символы
^{...}	Надстрочные символы

Таблица 2.5 – Теги, характеризующие тип информации

Тег	Значение
<EM>...</EM>	Типографское усиление
<CITE>...</CITE>	Цитирование
<STRONG>...</STRONG>	Усиление
<CODE>...</CODE>	Код программ
<SAMP>...</SAMP>	Последовательность литералов
<KBD>...</KBD>	Пример ввода символов с клавиатуры
<VAR>...</VAR>	Переменная
<DFN>...</DFN>	Определение
<Q>...</Q>	Текст, заключенный в двойные кавычки

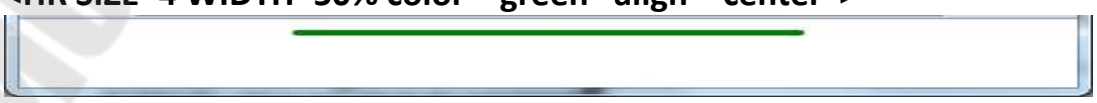
2.5 Графические объекты HTML-документа

Тег <**hr**> – рисует горизонтальную линию, которая по своему виду зависит от используемых параметров, а также браузера. Тег относится к блочным элементам, линия всегда начинается с новой строки, а после нее все элементы отображаются на следующей строке. Тег имеет ряд необязательных атрибутов:

– **ALIGN=CENTER | RIGHT | LEFT** – задает выравнивание;
– **width= xx** – ширина линии в процентах/пикселях (точках);
– **size=X** – толщина линии в пикселях;
– **color=цвет** – цвет линии.

Пример. Задание линии

```
<HR SIZE=4 WIDTH=50% color="green" align="center">
```



Тег <**marquee**>...</**marquee**> – создает бегущую строку на странице. На самом деле содержимое контейнера <**marquee**> не ограничивается строками и позволяет перемещать (скролировать) любые элементы веб-страницы – изображения, текст, таблицы, элементы форм и т.д. Перемещение можно задать не только по горизонтали, но и вертикали, в этом случае указываются размеры области, в которой будет происходить движение. Первоначально тег <**marquee**> был предназначен только для браузера Internet Explorer, но современные версии других браузеров также понимают и поддерживают этот тег.

Этот тэг может иметь следующие атрибуты:

- **behavior="alternate | scroll | slide"** - Задает тип движения содержимого контейнера <**marquee**>;
- **bgcolor = "значение"**– Цвет фона.
- **direction="down | left | right | up"** – Указывает направление движения содержимого контейнера;
- **height= "значение"** – Высота области прокрутки;
- **hspace= "значение"** – Горизонтальные поля вокруг контента;
- **loop = "значение"**– Задает, сколько раз будет прокручиваться содержимое;
- **scrollamount= "значение"** – Скорость движения контента;
- **scrolldelay= "значение"** – Величина задержки в миллисекундах между движениями;
- **vspace = "значение"**– Вертикальные поля вокруг содержимого;
- **width= "значение"** – Ширина области прокрутки.

Пример. Бегущая строка красного цвета, направление скролинга справо налево, число циклов прокрутки 2.

```
<marquee width=70% direction="left" loop="2" bgcolor="red">
Это бегущая строка </marquee>
```

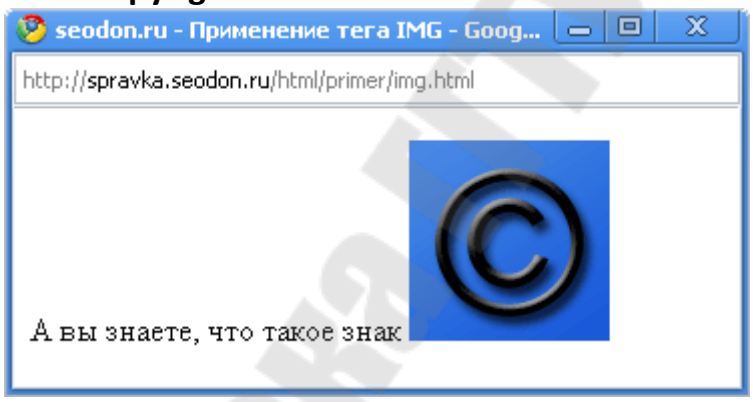
Тег <**IMG**> предназначен для отображения на веб-странице изображений в графическом формате GIF, JPEG или PNG. Этот тег имеет единственный обязательный параметр **src**, который определяет адрес файла с картинкой. При этом вокруг изображения отображается рам-

ка. Рисунки также могут применяться в качестве ссылок. Тег имеет ряд атрибутов:

- **align="bottom | left | middle | right | top"** – Определяет как рисунок будет выравниваться по краю и способ обтекания текстом. Наиболее популярные параметры – left и right, создающие обтекание текста вокруг изображения
- **border="толщина рамки»** – Толщина рамки вокруг изображения.
- **height="высота"** – Высота изображения.
- **width="ширина»** – Ширина изображения.
- **hspace="отступ по горизонтали»** – Горизонтальный отступ от изображения до окружающего контента.
- **vspace="отступ по вертикали"** – Вертикальный отступ от изображения до окружающего контента.
- **src="URL"** – Путь к графическому файлу.

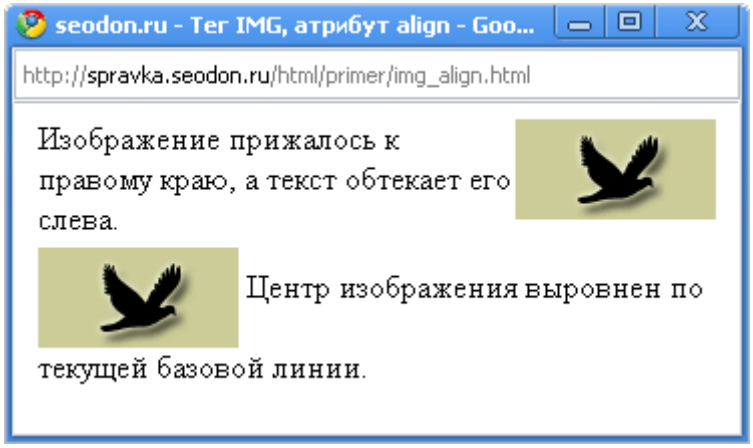
Пример. Вставка изображения в документ.

```
<img src="images/copyright.png" width="100" height="100"
alt="Знак Copyright">
```



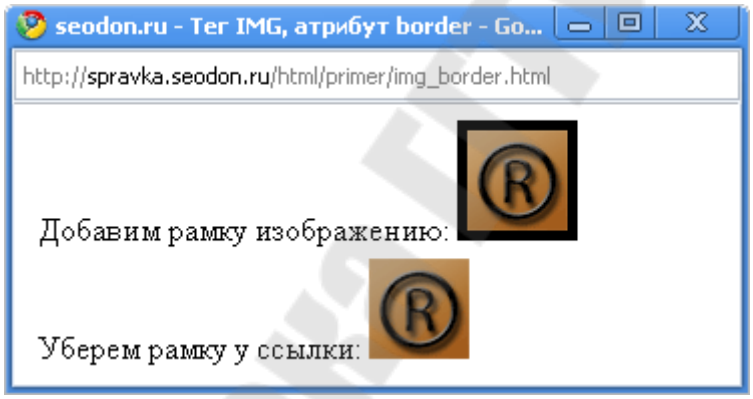
Управление выравниванием изображения.

```
<p><img src="images/orel.gif" align="right" width="100" height="50"
alt="Орел"> Изображение прижалось к правому краю, а текст обтекает
его слева.</p>
<p><img src="images/orel.gif" align="middle" width="100" height="50"
alt="Орел"> Центр изображения выровнен по текущей базовой
линии.</p>
```



Задание рамки вокруг изображения.

```
<p>Добавим рамку изображению:
  <img src="images/registered.png" border="5" width="50" height="50"
alt="Registered"></p>
<p>Уберем рамку у ссылки:
  <a href="files/registered.html">
   <img src="images/registered.png" border="0" width="50" height="50"
alt="Registered"> </a>
</p>
```



Самыми распространенными графическими форматами в Web являются **GIF, PNG** и **JPEG**.

GIF - формат графики без потери качества за счет уменьшения количества цветов на рисунке. Максимум цветов может быть 256, чего обычно хватает для сохранения логотипов, схем, надписей и контрастных рисунков. Формат **GIF** поддерживает прозрачность, но прозрачность только одной градации: нельзя сделать пиксель прозрачным на 20% или на 77%. Он может быть или прозрачным, или непрозрачным. Также, **GIF** позволяет создавать файлы анимации. Для этого создается последовательность кадров, которая объединяется в один

файл. После загрузки анимированного **GIF** кадры последовательно отображаются с задержкой на указанное для каждого кадра количество миллисекунд.

PNG (англ. portable network graphics) – растровый формат хранения графической информации, использующий сжатие без потерь по алгоритму **Deflate**. Графический формат **PNG** разрабатывался как альтернатива защищенному патентом и не поддерживающему переменную прозрачность формату **GIF**. Формат **PNG** сочетает лучшие качества форматов для интернета **JPEG** и **GIF**: сжимать изображение, оставляя только нужные цвета, при этом обеспечивая четкость и контрастность для графики. В большинстве случаев изображение в формате **PNG** занимает меньше места, чем оно же в **JPEG** (кроме фотографий) или в **GIF** (кроме крошечных изображений с несколькими цветами). Изображения в **PNG** можно сохранять, используя или 8 бит (PNG-8) или 24 бита (PNG-24).

Результаты сохранения картинки в **PNG-8** и **GIF** мало чем отличается. Оба формата альфа-прозрачность не поддерживают. Обычно файл в **PNG-8** весит чуть меньше, чем в **GIF**, если нужно сжимать области постоянного цвета с сохранением четкости линий в больших картинках. Т.е. сохранять в **PNG-8** лучше логотипы, иллюстрации с текстом, карикатуры, схемы и т.п.

PNG-24, как и формат **JPEG**, позволяет сохранять насыщенную графику, градиенты, изображение не теряет яркости, оттенков и четкости. В нем можно хранить и фотографии, но в **JPEG** они обычно занимают меньше места. Формат **PNG-24**, как и форматы GIF и PNG-8, сохраняет четкость деталей в штриховой графике, логотипах и иллюстрациях с текстом. Кроме того, **PNG-24** поддерживает альфа-прозрачность, т.е. прозрачность может быть частичной. Браузеры Internet Explorer младше 7й версии не поддерживают частичную прозрачность в **PNG-24**.

JPEG – для изображений фотографического качества. Качество изображения зависит от степени сжатия файла: чем больше ужат файл, тем хуже картинка, но зато меньше размер самого файла. Потому хранение фотографий в JPEG – это компромисс между качеством изображения и его размером. На сильно ужатом файле этого формата появляются размытые области, теряется контрастность. Качество картинки теряется с каждым новым сохранением файла в JPEG, потому хранить промежуточные результаты работы нужно в формате, который не влияет на качество изображения.

Активное изображение (**image maps**) – это изображение чувствительные к щелчкам мыши, с активными областями, которые ссылаются на другие страницы или ресурсы. Процесс создания активного изображения состоит из двух этапов. Сначала на картинке необходимо определить области, которые нужно сделать активными, а потом соотнести их со ссылками на другие URL. Границы активных областей задаются координатами углов прямоугольника и многоугольника или центра и радиуса круга. Для задания активного изображения используется тег <**MAP**> и внутренний задающие области тег <**AREA**>. Синтаксис:

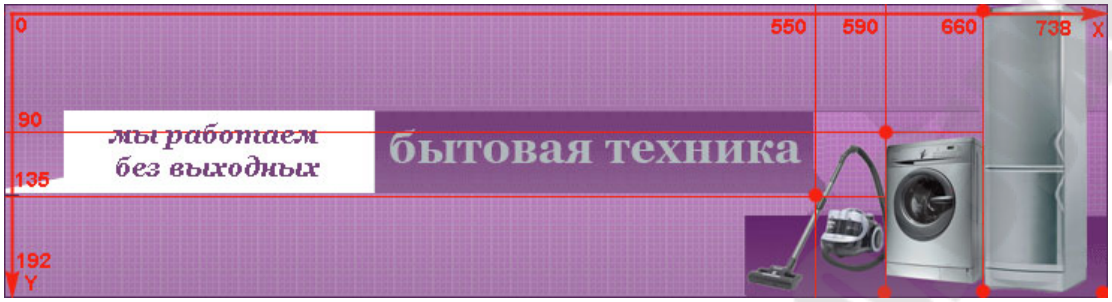
```
<map name="имя">
 <area атрибуты>
 <area атрибуты>
……..
</map>
```

Тег <**map**> служит контейнером для элементов <area>, которые определяют активные области для карт-изображений. Каждый элемент <**area**> определяет активные области изображения, которые являются ссылками. Тег <**area**> задает форму области, ее размеры, устанавливает адрес документа, на который следует сделать ссылку, а также имя окна или фрейма, куда браузер будет загружать документ. Основные атрибуты тега <**area**>:

- **alt ="текст"** – Альтернативный текст для области изображения.
- **coords="координата 1, координата 2, координата 3, ..."** – Координаты активной области. Зависят от ее вида.
- **href="URL"** – Задает адрес документа, на который следует перейти.
- **nohref** – Область без ссылки на другой документ.
- **target ="имя окна"** – Имя окна или фрейма, куда браузер будет загружать документ.(**_blank** –в новое окно браузера.**_self** –в текущее окно.)
- **SHAPE= rect | poly | circle** – Форма области. Способы задания координат для разных типов областей:
  - ***SHAPE="rect" COORDS="левый x, верхний y, правый x, нижний y";***
  - ***SHAPE="circle" COORDS="центр x, центр y, радиус";***
  - ***SHAPE="poly" COORDS="x1,y1,x2,y2,x3,y3,...";***

Пример. Задание активного изображения.

Ширина нашей картинки 738 пикселов, а высота - 192 пиксела. Проведем линии по границам наших областей и примерно определим координаты.



Зададим найденные координаты в теги <**area**>.

```
<body>
<img src="map.gif" usemap="#map" border="0">
<map name="map">
   <area shape="rect" coords="550,135, 590,192" href="#" alt="пылесосы">
   <area shape="rect" coords="591,90, 660,192" href="#" alt="стиральные
   машины">
   <area shape="rect" coords="661,0, 730,192" href="#"
   alt="холодильники">
</map>
</body>
```

2.6 Задание ссылок

Тег **<A>** является одним из важных элементов HTML и предназначен для создания ссылок. В зависимости от присутствия атрибутов **name** или **href** тег **<A>** задает ссылку или якорь. Якорь – закладка внутри страницы, которую можно указать в качестве цели ссылки. При использовании ссылки, которая указывает на якорь, происходит переход к закладке внутри веб-страницы. Допускается переход к якорю и на другой странице.

Для создания ссылки необходимо:

– задать ссылку (объект на который необходимо нажать);
– указать адрес документа, на который следует перейти.

Адрес ссылки может быть абсолютным и относительным. Абсолютные адреса работают везде, независимо от имени сайта или веб-страницы. Относительные ссылки, построены относительно текущего документа или корня сайта. Синтаксис:

```
<a href="URL">...</a>
<a name="идентификатор">...</a>
```

Атрибуты:

- **href="URL"** – Задает адрес документа, на который следует перейти. Поскольку в качестве адреса ссылки может использоваться документ любого типа, то результат перехода по ссылке зависит от конечного файла. Так, архивы будут сохраняться на локальный диск. По умолчанию новый документ загружается в текущее окно браузера,
- **name="закладка"** – Устанавливает имя якоря внутри документа.
- **target ="имя окна"** – Имя окна или фрейма, куда браузер будет загружать документ (**_blank** – в новое окно браузера, **_self** – текущее окно);
- **title="текст"** – Добавляет всплывающую подсказку к тексту ссылки.

Пример. Использование тега <**A**>

Задание ссылки с использованием изображения:

```
<p><a href="images/my.jpg">Посмотрите на мою фотографию!</a></p>
```

Задание ссылки с использованием текста:

```
<p><a href="tip.html">Как сделать такое же фото?</a></p>
```

Пример. Использование параметра href

```
<a href="my.html">Относительная ссылка</a>
<a href="http://www.name.ru/html/my.html ">Абсолютная ссылка</a>
```

Пример. Задание закладок.

Вначале следует задать в соответствующем месте якорь и установить его имя параметром **name** тега **<A>**. Имя ссылки на закладку начинается символом #, после чего идет название закладки. Можно также делать ссылку на закладку, находящуюся в другой веб-странице и даже другом сайте. Для этого в адресе ссылки надлежит указать ее адрес и в конце добавить символ решетки # и имя закладки.

```
<body>
 <p><a name="top"></a></p>                    /*задание якоря*/
 <p>Здесь много-много текста.
  Прокручивай его вниз. </p>

 <p><a href="#top">Наверх</a></p>             /*ссылка на закладку*/
</body>
```

Пример. Открытие ссылки в новом окне

```
<p><a href="new.html" target="_blank">Открыть в новом окне</a></p>
```

Пример. Создание всплывающей подсказки

```
<p><a href="zoo.html" title="Рисунки  животных ...">Рисунки</a></p>
```

2.7 Структурированные данные. Списки

Списки позволяют упорядочить и систематизировать различную информацию и представить ее для посетителя в удобном виде. Списки в HTML могут быть трех разновидностей: маркированные списки, нумерованные списки и списки определений. Рассмотрим задание списков.

Тег <**UL**> задает маркированный список. Каждый элемент списка должен начинаться с тега <**LI**>. Синтаксис:

```
<ul type= "disc | circle | square" >
    <li>элемент маркированного списка</li>
    <li>элемент маркированного списка</li>
    …
</ul>
```

Атрибуты тега:

– **type= "disc | circle | square"** – Устанавливает вид маркера списка. Значения параметра type и получаемый вид показан в таблице 2.6. По умолчанию значение атрибута – «disc».

Таблица 2.6 – Примеры маркированного списка

Код	Пример

`<ul type=" disc ">`
`<li>Чебурашка</li>`
`<li>КрокодилГена</li>`
`<li>Шапокляк</li>`
`</ul>`
	• Чебурашка
	• Крокодил Гена
	• Шапокляк

`<ul type="circle">`
`<li>Чебурашка</li>`
`<li>КрокодилГена</li>`
`<li>Шапокляк</li>`
`</ul>`
	o Чебурашка
	o Крокодил Гена
	o Шапокляк

<ul type=" square "> ** - Чебурашка ** ** - КрокодилГена ** ** - Шапокляк ** ** **	▪ Чебурашка ▪ Крокодил Гена ▪ Шапокляк

Тег **<OL>** задает нумерованный список. Каждый элемент списка должен начинаться с тега **<LI>**. Синтаксис:

```
<ol type="A | a | I | i | 1" start="число" >
    <li>элемент маркированного списка</li>
    <li>элемент маркированного списка</li>
    <li>элемент маркированного списка</li>
    …
</ol>
```

Атрибуты тега:

– **type= " A | a | I | i | 1"** – вид и тип нумерации. В качестве маркеров могут быть следующие значения: заглавные латинские буквы; строчные латинские буквы; заглавные римские цифры; строчные римские цифры; арабские цифры. Значения параметра type и получаемый вид показан в таблице 2.7. По умолчанию значение атрибута – «1»;

– **start="число"** – параметр устанавливает номер, с которого будет начинаться список.

Таблица 2.7 – Примеры нумерованного списка

Код	Пример
**<o type="A"> … **	A. Чебурашка B. Крокодил Гена C. Шапокляк
**<ol type="a"> … **	a. Чебурашка b. Крокодил Гена c. Шапокляк
**<ol type="I"> … **	I. Чебурашка II. Крокодил Гена III. Шапокляк
**<ol type="i"> … **	i. Чебурашка ii. Крокодил Гена iii. Шапокляк
**<ol type="1"> … **	1. Чебурашка

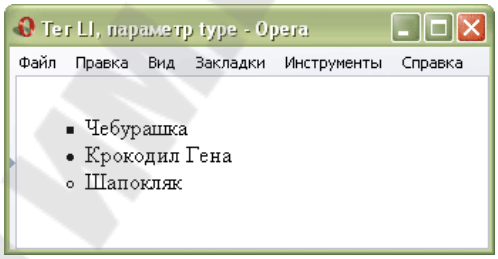
	2. Крокодил Гена 3. Шапокляк

Задание элементов списка используется тег **<LI>**. Атрибуты тега:

- **type= "disc | circle | square"** – устанавливает вид маркера для ненумерованных списков;
- **type= " A | a | I | i | 1"** – вид и тип нумерации маркера для нумерованных списков;
- **value="число"** – Параметр устанавливает номер текущего элемента списка. Только для нумерованных списков.

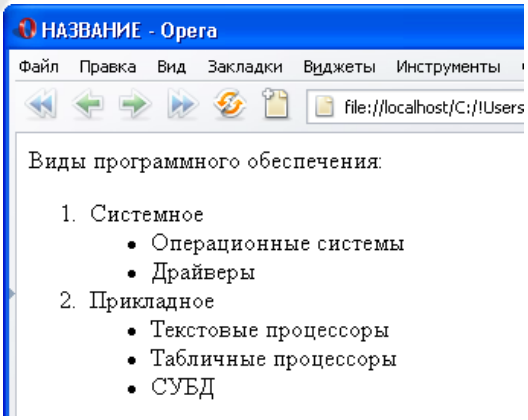
Пример. Задать простой ненумерованный список с различными маркерами для его элементов.

```
<ul>
<li type="square">Чебурашка</li>
<li>Крокодил Гена</li>
<li type="circle">Шапокляк</li>
</ul>
```



Пример. Задать вложенный нумерованный список с ненумерованными подсписком.

```
<p> Виды программного
обеспечения:
<ol>
 <li> Системное
 <ul>
    <li> Операционные системы
    <li> Драйверы
 </ul>
 <li> Прикладное
 <ul>
    <li> Текстовые процессоры
    <li> Табличные процессоры
    <li> СУБД
 </ul>
</ol>
</p>
```

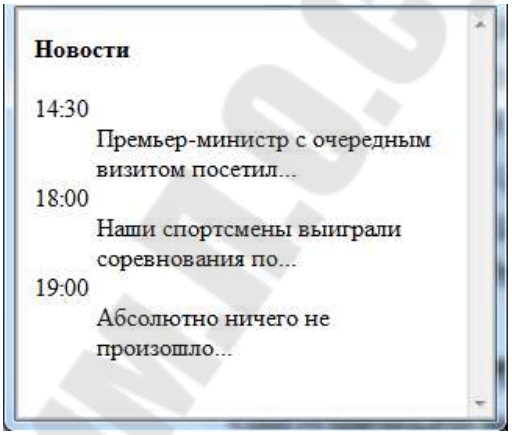


Для создания списка терминов и их определений используется тег <**DL**>. Определяемый термин задается тегом <**DT**>, а его определение тегом <**DD**>. Синтаксис:

```
<DL><DT>Термин</DT> <DD>Определение</DD></DL>
```

Пример. Задание определения

```
<h4>Новости</h4>
<dl>
<dt>14:30</dt>
<dd>Премьер-министр с
    очередным визитом
    посетил...</dd>
<dt>18:00</dt>
<dd>Наши спортсмены выиграли
    соревнования по...</dd>
<dt>19:00</dt>
<dd>Абсолютно ничего не
    произошло...</dd>
</dl>
```



2.8 Структурированные данные. Таблицы

В настоящее время таблицы в HTML используются, в основном, для форматирования и оформления страниц, хотя и первоначальное их назначение как метода представления информации не утратило своего значения. Таблицы дают широчайшие возможности для оформления интернет-страниц. Рассмотрим подробнее их использование.

Элемент **<TABLE>** служит контейнером для элементов, определяющих содержимое таблицы. Любая таблица состоит из строк и ячеек, которые задаются с помощью тегов **<TR>** и **<TD>**. Таблица начинается с тэга <**table**> и заканчивается тэгом </**table**>. Общая структура таблицы:

```
<table>
…
</table>
```

Тэг <**table**> может включать несколько атрибутов:

– **align="left | center | right"** – устанавливает расположение таблицы по отношению к полям документа.

- **background="URL"** – Определяет изображение, которое будет использоваться в качестве фонового рисунка таблицы.
- **bgcolor="цвет"** – Устанавливает цвет фона таблицы.
- **width="значение"** – Задает ширину таблицы. Ее можно задать в пикселях (например, WIDTH=400) или в процентах от ширины страницы (например, WIDTH=80%). Если общая ширина содержимого превышает указанную ширину таблицы, то браузер будет пытаться «втиснуться» в заданные размеры за счет форматирования текста. В случае, когда это невозможно, например, в таблице находятся изображения, параметр width будет проигнорирован, и новая ширина таблицы будет вычислена на основе ее содержимого.
- **border="толщина"** – устанавливает ширину внешней рамки таблицы и ячеек в пикселях (например, BORDER=4). Если атрибут не установлен, таблица показывается без рамки.
- **bordercolor="цвет"** – Устанавливает цвет рамки таблицы.
- **cellspacing="значение"** – устанавливает расстояние между рамками ячеек таблицы в пикселях (например, CELLSPACING=2).
- **cellpadding="значение"** – устанавливает расстояние между рамкой ячейки и текстом в пикселях (например, CELLPADDING=10).
- **cols="число"** – задает количество столбцов в таблице, помогая браузеру в подготовке к ее отображению. Без этого параметра таблица будет показана только после того, как все содержимое таблицы будет загружено в браузер и проанализировано.
- **rules="значение"** – Сообщает браузеру, где отображать границы между ячейками. Толщина границы и ее цвет указывается с помощью параметров border и bordercolor. По умолчанию рамка рисуется вокруг каждой ячейки, образуя тем самым сетку. Допустимые значения:
  - **all** – Линия рисуется вокруг каждой ячейки таблицы;
  - **cols** – Линия отображается между колонками;
  - **none** – Все границы скрываются;
  - **rows** – Граница рисуется между строками таблицы, созданных через тег <TR>.

Для задание контейнера для строк таблицы служит тег **<TR>**. Общая структура:

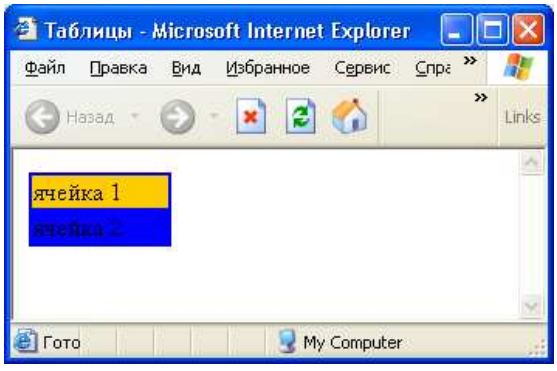
```
<table>
<tr>.....</tr>
<tr>.....</tr>
...
</table>
```

Атрибуты тега:

- **bgcolor="цвет"** – Устанавливает цвет фона строки таблицы;
- **bordercolor="цвет"** – Устанавливает цвет рамки вокруг строки. Рамка показывается, когда установлен параметр border с ненулевым значением у тега <TABLE>.
- **align="left | center | right | justify**" – Задает выравнивание содержимого ячеек строки по горизонтали. Выравнивание осуществляется для всех ячеек в пределах одной строки.
- **valign="top | middle | bottom | baseline"** – Устанавливает вертикальное выравнивание содержимого ячеек в строке. По умолчанию содержимое ячейки располагается по ее вертикали в центре – middle. Допустимые значения:
  - **top** – Выравнивание содержимого ячеек по верхнему краю строки;
  - **middle** – Выравнивание по середине;
  - **bottom** – Выравнивание по нижнему краю.

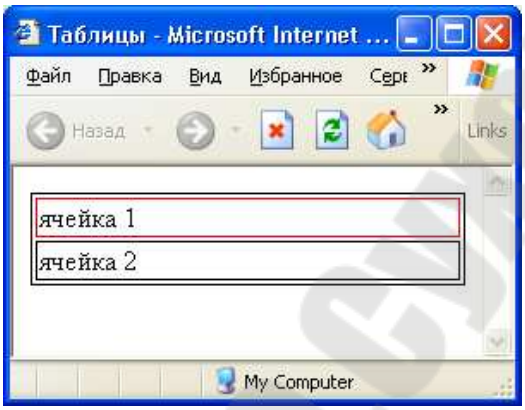
Пример. Задать таблицу с синим фоном. Для первой строки установить золотой цвет.

```
<table bgcolor="blue" width="30%">
  <tr bgcolor="#ffcc00">
   <td> ячейка 1 </td>
  </tr>
  <tr>
   <td> ячейка 2 </td>
  </tr>
 </table>
```



Пример. Задать таблицу с черным цветом сетки. Для первой строки установить красный цвет сетки.

```
<table bordercolor="black"
      border="1" width="100%">
   <tr bordercolor="red">
    <td> ячейка 1 </td>
   </tr>
   <tr>
    <td> ячейка 2 </td>
   </tr>
</table>
```



Для задание ячеек таблицы предназначен тег **<TD>**. Контейнер должен размещаться внутри тега **<TR>**.

```
<table>
<tr>  <td>ячейка 1, 1-строки </td> ….. <td> ячейка n, 1-строки </td> </tr>
<tr>  <td>ячейка 1, 2-строки </td> ….. <td> ячейка n, 2-строки </td> </tr>
…
</table>
```

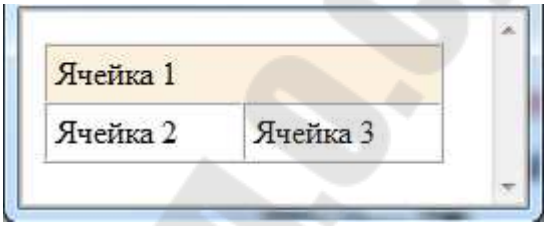
Атрибуты тега:

- **align="left | center | right | justify"** – Задает выравнивание содержимого ячейки по горизонтали.
- **valign="top | middle | bottom | baseline"** – Устанавливает вертикальное выравнивание содержимого ячейки.
- **background="URL"** – Определяет изображение, которое будет использоваться в качестве фонового рисунка таблицы.
- **bgcolor="цвет"** – Устанавливает цвет фона ячейки.
- **bordercolor="цвет"** – Устанавливает цвет рамки вокруг ячейки.
- **colspan="число"** – Устанавливает число ячеек, которые должны быть объединены по горизонтали.
- **rowspan="число"** – Устанавливает число ячеек, которые должны быть объединены по вертикали.
- **height="значение"** – Браузер сам устанавливает высоту таблицы и ее ячеек исходя из их содержимого. Однако при использовании параметра height высота ячеек будет изменена. Здесь возможны два варианта. Если значение height меньше, чем содержимое ячейки, то этот параметр будет проигнорирован. В случае, когда установлена высота ячейки, превышающая ее содержимое, добавляется пустое пространство по вертикали
- **width="значение"** – Задает ширину ячейки. Если общая ширина содержимого превышает указанную ширину ячейки, то браузер

будет пытаться «втиснуться» в заданные размеры за счет форматирования текста. В случае, когда это невозможно, например, в ячейке находятся изображения, параметр width будет проигнорирован, и новая ширина ячейки будет вычислена на основе ее содержимого.

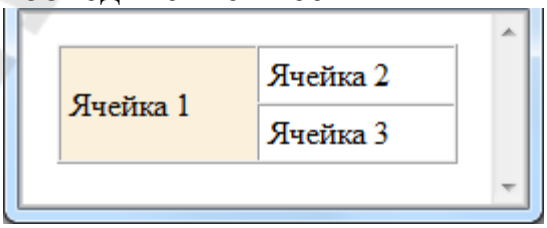
Пример. Таблица с горизонтальным объединение ячеек

```
<table width="200" border="1"
cellpadding="4" cellspacing="0">
 <tr>
  <td colspan="2" bgcolor="#FBF0DB"
    >Ячейка 1</td>
  </tr>
 <tr>
  <td>Ячейка 2</td>
  <td>Ячейка 3</td>
 </tr>
</table>
```



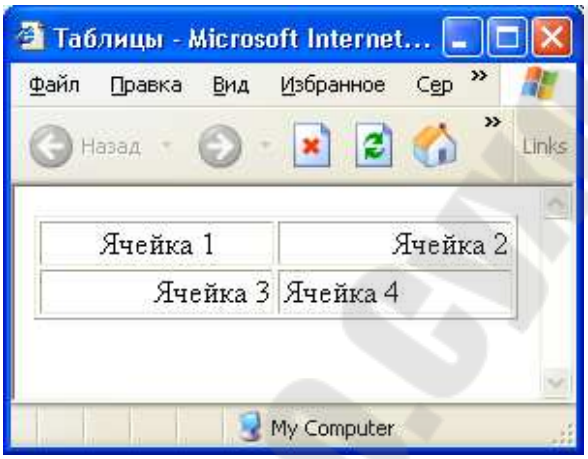
Пример. Таблица с вертикальным объединение ячеек

```
<table width="200" border="1"
 align="center"
   cellpadding="4" cellspacing="0">
  <tr>
   <td rowspan="2" bgcolor="#FBF0DB"
     >Ячейка 1</td>
   <td>Ячейка 2</td>
  </tr>
  <tr>
   <td>Ячейка 3</td>
  </tr>
</table >
```



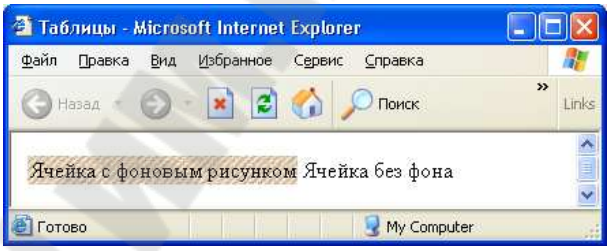
Пример. Задать таблицу шириной 200 пикселей. Для первой строки первая ячейка выравнивание по центру, для второй по правому краю, вторая строка – все ячейки по правому краю, последняя по левому.

```
<table width=”250” border=”1”>
  <tr>
   <td align=”center”>Ячейка 1</td>
   <td align=”right”>Ячейка 2</td>
  </tr>
  <tr align=”right”>
   <td>Ячейка 3</td>
   <td align=”left”>Ячейка 4</td>
  </tr>
</table >
```



Пример. Задать таблицу в которой одна ячейка с фоновым рисунком, а другая без.

```
 <table>
  <tr>
   <td background="fon.jpg">Ячейка
      с фоновым рисунком</td>
   <td>Ячейка без фона</td>
  </tr>
 </table>
 </table> >
```



Пример. Задать таблицу следующего вида.

Пара	День	
	Понедельник	Вторник
8:00-9:30	Пение	Рисование
	Информатика	Математика
9:40-11:10	Философия	История
	Физика	Химия

```
<table width="75%" border="2" cellspacing="0" cols="3" bgcolor="gold"
     align="center" bordercolor="black" >
  <tr align="center">
   <td rowspan="2">Пара</td>
   <td colspan="2">День</td>
   </tr>
  <tr align="center">
   <td>Понедельник</td>
   <td>Вторник</td>
```

```
 </tr>
 <tr>
  <td rowspan="2" bgcolor="red">8:00-9:30</td>
  <td>Пение</td>
  <td>Рисование</td>
 </tr>
 <tr>
  <td>Информатика</td>
  <td>Математика</td>
 </tr>
 <tr>
  <td rowspan="2" bgcolor="red">9:40-11:10</td>
  <td>Философия</td>
  <td>История</td>
 </tr>
 <tr>
  <td>Физика</td>
  <td>Химия</td>
 </tr>
</table>
```

Пара	День	
	Понедельник	Вторник
8:00-9:30	Пение	Рисование
	Информатика	Математика
9:40-11:10	Философия	История
	Физика	Химия

3. Язык гипертекстовой разметки страниц HTML. Формы

Формы используются для организации взаимодействия пользователя с Web-сервером через Internet. Браузер собирает, если необходимо, кодирует предоставленные данные и отправляет их серверу или по заданному адресу электронной почты. Сервер передает данные поддерживающей программе или приложению, которое обрабатывает информацию и генерирует ответ, обычно на HTML. Сервер посылает

ответ броузеру клиента, если связь осуществляется по электронной почте, информация просто попадает в почтовый ящик.

3.1 Задание формы

Форма может располагаться в любом месте тела документа, между тегом <**form**> и его закрывающим тегом </**form**>. Внутри формы располагаются теги задающие элементы интерфейса **input**, **select**, **textarea**. Атрибуты тега.

Атрибут action - задает **URL** приложения, которое должно получить и обработать данные формы.

Пример. Задание скрипта обработки данных формы.

Передаёт значения формы скрипту **update**, расположенному в каталоге **cgi-bin** на вебсервере **mysite**.

```
<form action="http://www.mysite/cgi-bin/update">...</form>
```

URL-адрес файла обработчика, написанной на языке Perl:

```
<form action="http://mysite.com/scripts/myprogram.pl">
```

URL-адрес скрипта на asp-файл (Active Server Page):

```
<form action="www.anyserver.ru/mypage.asp">
```

URL-адрес скрипта на PHP " файл:

```
<form action="www.anyserver.ru/mypage.php">
```

Для отправки данных по электронной почте по указанному адресу:

```
<form action="mailto:my@server.ru">
```

Атрибут method - устанавливает метод, посредством которого браузер передает серверу для обработки данные формы. Существуют два метода **POST** и метод **GET** (по умолчанию). Следуя методу **POST**, браузер передает данные в два шага: во-первых, браузер устанавливает связь с обрабатывающим форму сервером, адрес которого указан в атрибуте **action**, и, во-вторых, когда связь установлена, посылает данные серверу в отдельном сеансе связи. Метод **GET**, напротив, соединяется с обрабатывающим форму сервером и отправляет данные за один шаг.

Метод **GET** используют при передаче небольших объемов данных. **POST** применяется при передаче больших объемов данных и сообщений электронной почты, данные находиться в теле запроса.

Маленькие формы с небольшим числом коротких полей обычно посылаются методом **GET**. Для отправки форм, с большим числом

полей или полей содержащих длинные текстовые записи, используется метод **POST**.

Если защищенность данных представляется важной, лучше использовать **POST**. **GET** добавляет параметры формы прямо к URL запросу, где они могут быть легко перехвачены сетевыми анализаторами пакетов или выделены из регистрационного журнала сервера. При передаче методом **POST,** можно воспользоваться шифровкой параметров, передаваемых в отдельном сеансе связи с сервером.

Атрибут **enctype** – метод кодирования данных передаваемых на сервер. Возможные значения:

– **application/x-www-form-urlencoded** – преобразует все пробелы в значениях формы в знаки «плюс» (+), символы, отличные от латинских букв и цифр, – в знак процента (%), за которым следуют две шестнадцатеричные цифры, являющиеся ASCII кодом символа, а разрывы строк в многострочных данных – в **%0D%0A**. Для каждого элемента формы генерируется своя пара **имя=значение**. При использовании метода **get** такие пары разделяются амперсандом (**&**). Например, вот что браузер посылает серверу после того, как пользователь заполнил форму с двумя полями ввода, имена которых **name** и **address**. В первом поле только одна строка текста, тогда как второе содержит несколько строк ввода:

name=Lena+Ivanova&address=Gomel+Sovetskaya+187

– **multipart/form_data** - используется с формами, содержащими поле выбора файла, который загружает пользователь. Кодировка включает поля формы в виде отдельных частей. У каждого поля в результирующем пакете есть соответствующий сегмент, отделенный от сегментов соседей стандартным разделителем. В каждом сегменте одна или несколько заголовочных строк определяют имя поля, за которым следуют одна или несколько строк, содержащих его значение. Эта кодировка более сложная и длинная по сравнению с **application/x_www_form_urlencoded**. Поэтому она применяется, только при передаче метолом **post**.

----------146931364513459
Content Disposition: form data; name="name"
Lena Ivanova
----------146931364513459
Content Disposition: form data; name="address"
Gomel Sovetskaya 187
----------146931364513459

– **text/plain** - применяется при посылке форм по адресу электронной почты. Закодированными в этом формате остаются только символы возврата каретки и перевода строки в многострочных полях ввода.

3.2 Элементы пользовательского интерфейса

Элементы пользовательского интерфейса могут использоваться в HTML – документах как элементы формы (контейнера <**form**>), если предполагается передача содержащиеся в них данные на сервер, так и сами по себе, как элемент интерфейса.

Для задания элементов управления формы используется тег <**INPUT**>. С помощью него, можно создавать элементы различных типов: поля ввода данных, кнопки и переключатели и т.д. Тип элемента определяется атрибутом **type**. Общие атрибуты тега <**input**>:

Кроме рассмотренных ранее, имеет следующие атрибуты:

– **name** – имя (идентификатор) элемента. Обязательно если предполагается использовать элемент в составе формы для передачи значения атрибута **value** на сервер;
– **disabled** – делает элемент недоступным пользователю;
– **accesskey** – определяет клавишу быстрого доступа к элементу (в сочетании с клавишей <**Alt**>); **Например:<input type= "button" value="OK" accesskey="Y">**;
– **tabindex** – целое число, определяющее порядок перехода к элементу по клавиши <**TAB**>.

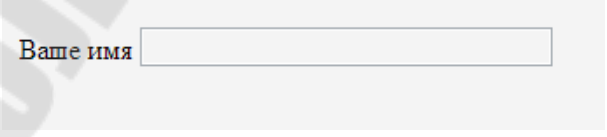
Рассмотрим подробнее элементы управления.

Текстовое поле ввода данных: **type= text**. Дополнительные атрибуты:

– **maxlength** – максимальное количество вводимых символов;
– **size** – ширина поля, выраженная в количестве одновременно видимых символов;
– **value** – значение (содержимое поля).

Пример. Задать поле ввода имени. Максимальная длина имени 35 символов.

Ваше имя <INPUT TYPE="text" NAME="Name" SIZE="35">

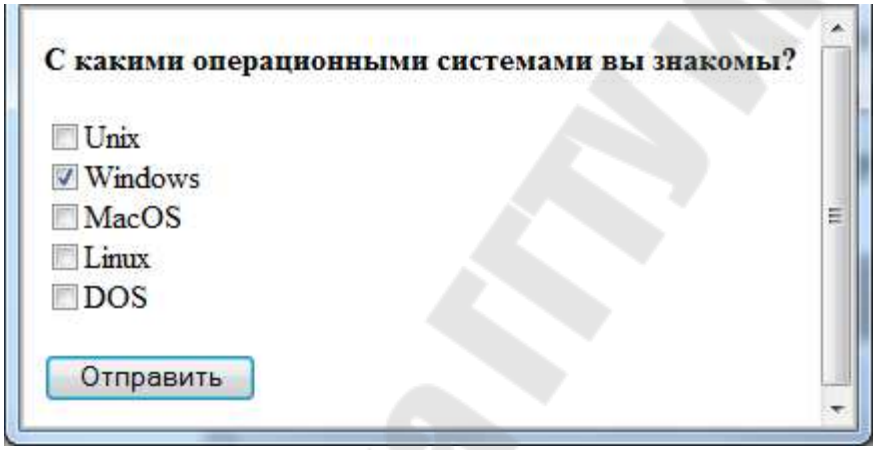


Переключатель типа флажок: **type= checkbox**. Дополнительные атрибуты:

– **value** – значение, ассоциированное с флажком;
– **checked** – определяет, помечен ли заранее переключатель.

Пример. Задать форму выбора операционной системы, с отправкой данных скрипту обработки – "**input.php**". Ос **Windows** – выбрана по умолчанию.

```
<form method="post" action="input.php">
<p><b>С какими операционными системами вы знакомы?</b></p>
<p><input type="checkbox" name="option1" value="a1">Unix<Br>
<input type="checkbox" name="option2" value="a2" checked>Windows<Br>
<input type="checkbox" name="option3" value="a3">MacOS<Br>
<input type="checkbox" name="option4" value="a4">Linux<Br>
<input type="checkbox" name="option5" value="a5">DOS</p>
<p><input type="submit" value="Отправить"></p>
</form>
```

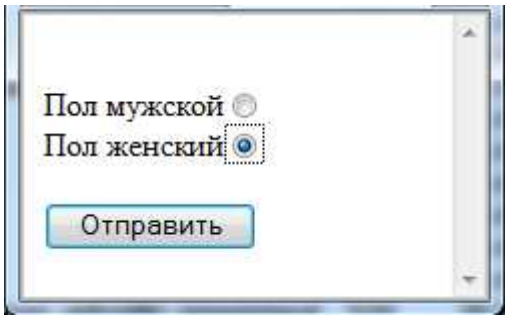


Переключатель выбора одного варианта: **type = radio**. Дополнительные атрибуты:

– **value** – значение, ассоциированное с радиокнопкой;
– **checked** – определяет, помечен ли заранее переключатель.

Пример. Задать форму выбора пола.

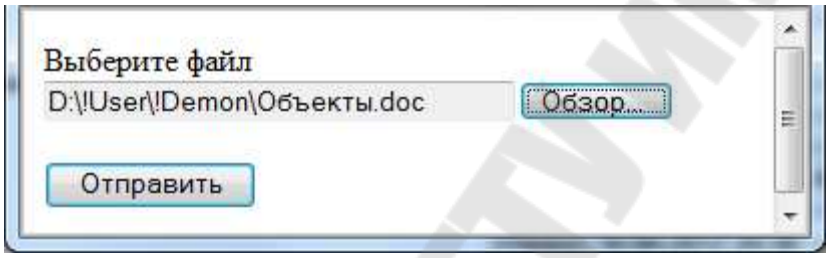
```
<form method="post" action="input.php">
   <BR>Пол мужской<INPUT NAME="Пол" TYPE=radio VALUE="Мужской">
   <BR>Пол женский<INPUT NAME="Пол" TYPE=radio VALUE="Женский">
   <p><input type="submit" value="Отправить"></p>
</form>
```



Поле ввода имени файла, для отсылки на сервер: **type=file**. Атрибуты как у тестового поля. Поле содержит кнопку «Обзор» для открытия диалогового окна выбора файла.

Пример. Форма выбора файла для отсылки на сервер.

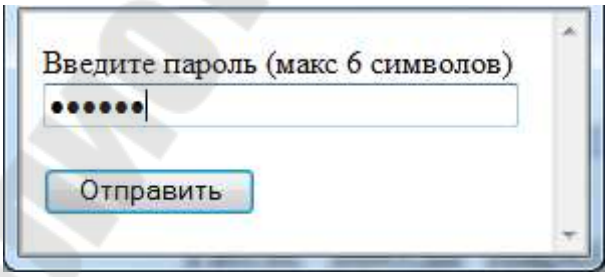
```
<form method="post" action="input.php">
    Выберите файл <INPUT  TYPE="file" NAME="Name" SIZE="35">
    <p><input type="submit" value="Отправить"></p>
</form>
```



Поле ввода пароля: **type= password**. Обычное текстовое поле, в котором все вводимые символы отображаются как символ *. Атрибуты как у тестового поля.

Пример. Форма ввода пароля.

```
<form method="post" action="input.php">
    Введите пароль (макс 6 символов)<INPUT TYPE="password"
    NAME="Name" SIZE="35" maxlength="6">
    <p><input type="submit" value="Отправить"></p>
</form>
```

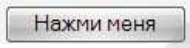


Задание кнопки: **type= button**. Дополнительные атрибуты:

– **value** – надпись, отображаемая на кнопке. По умолчанию на кнопке нет надписи.

Пример. Задание кнопки

```
<INPUT TYPE=button VALUE="Нажми меня">
```

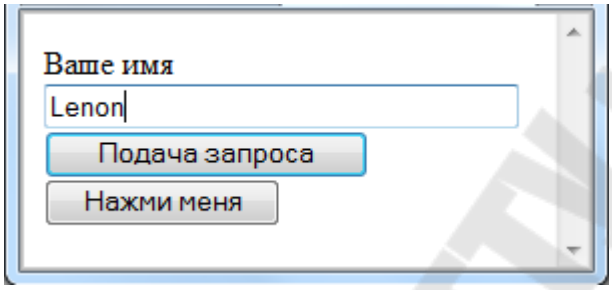


Кнопка отправки данных формы на сервер: **type= submit.** Дополнительные атрибуты:

– **value –** строка, отображаемая на кнопке. По умолчанию на кнопке находится надпись «**Подача запроса**» или «**Submit**».

Пример. Задание кнопки отправки данных.

```
<form method="post" action="input.php">
    Ваше имя <INPUT  TYPE="text" NAME="Name" SIZE="35">
    <INPUT TYPE=submit>
    <INPUT TYPE=submit VALUE="Нажми меня">
</form>
```



Графическая кнопка отправки данных формы на сервер: **type= image.** При нажатии мышью по изображению, браузер дополнительно передает координаты точки клика на сервер. Дополнительные атрибуты:

– **src="URL"** – адрес графического файла.

Пример. Задание графической кнопки отправки данных.

```
<form action="http://www.ya.ru/input.php">
    Ваше имя <INPUT  TYPE="text" NAME="Name" SIZE="35">
    <INPUT TYPE=image  src="pic.png" >
</form>
```

Строка передаваемая на сервер:

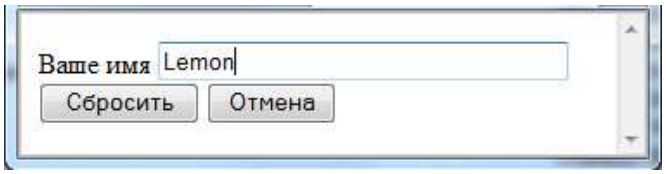
```
http://www.ya.ru/input.php?Name=lenon&x=65&y=18
```

Кнопка для сброса данных формы в первоначальное значение: **type= reset.** Дополнительные атрибуты:

– **value** – строка, отображаемая на кнопке. По умолчанию на кнопке находится надпись «**Сбросить**» или «**Reset**».

Пример. Задание графической кнопки отправки данных.

```
<form action="input.php">
    Ваше имя <INPUT  TYPE="text" NAME="Name" SIZE="35">
    <INPUT TYPE="reset">
    <INPUT TYPE="reset" VALUE="Отмена">
</form>
```



Скрытое поле: **type= hidden.** Позволяет включить в отправляемую форму значения атрибутов **NAME** и **VALUE**, которые пользователь изменить не может. Такие поля полезны для передачи специальных данных например идентификатора или даты заполнения формы.

Полный набор атрибутов элементов формы приведен на рисунке 3.1.

Тег формы или тип поля ввода <input>	Accept	accesskey	align	alt	border	cols	checked	disabled	maxlength	multiple	name	notab	onBlur	onChange	onClick	onFocus	onSelect	readonly	rows	size	src	tabindex	taborder	usemap	value	wrap
button		•						•			×	•	•		•	•						•	•		×	
checkbox		•					•	•			×	•			•			•				•	•		×	
file	•	•						•	•		×	•	•	•	•	•		•		•		•	•		•	
hidden											×														×	
image		•	•	•	•			•			•	•			•						×	•	•	•		
password		•						•	•		×	•	•	•	•	•	•	•		•		•	•		×	
radio		•					•	•			×	•			•			•				•	•		×	
reset		•						•				•			•							•	•		•	
submit		•						•			•	•			•							•	•		•	
text		•						•	•		×	•	•	•	•	•	•	•		•		•	•		•	
<button>		•						•			×		•		•	•						•			•	
<select>								•		•	×		•	•	•	•				•		•				
<textarea>		•				•		•			×		•	•	•	•	•	•	•			•				•

Рисунок 3.1 – Атрибутов элементов формы

3.3 Задание кнопок. Тег BUTTON

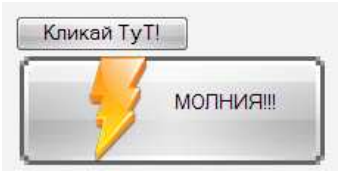
Для создания кнопки можно использовать контейнерный тег <**button**>. Текст, изображение и другие видимые элементы, вставленные в этот тег, отображаются на кнопке. Размер кнопки устанавливается автоматически в зависимости от размеров элементов, вставленных в тег. Тег <**button**> имеет следующие атрибуты:

- **disabled** – делает элемент недоступным пользователю;
- **accesskey** – определяет клавишу быстрого доступа к элементу (в сочетании с клавишей <**Alt**>).

Кнопка, заданная тегом <**button**>, может использоваться в формах, однако, поскольку тег не имеет атрибутов **value** и **name** его содержимое не может быть отправлено на сервер с помощью кнопки **submit**. Обычно используется совместно со скриптами для организации интерфейса.

Пример. Задание кнопки.

```
<form action="input.php">
    Ваше имя <INPUT  TYPE="text" NAME="Name" SIZE="35">
<button>Кликай ТуТ! </button> <br>
<button> <img src="light.png" align=center> МОЛНИЯ!!! </button>
```



3.4 Раскрывающийся список. Тег SELECT

Тег <**SELECT**> используется для создания раскрывающегося списка. Это контейнерный тег, внутри которого используются теги <**OPTION**>, предназначенные для задания элементов списка. Тег имеет следующие атрибуты:

- **align** – горизонтальное выравнивание списка; возможны значения;
- **absbottom** – выравнивание нижней границы списка по нижней границе текущей строки;
- **absmiddle** – выравнивание середины списка по середине текущей строки;
- **baseline** – выравнивание нижней границы списка по базовой линии текущей строки;

- **bottom** – то же, что и baseline;
- **top** – верхняя граница списка выравнивается по самому высокому элементу текущей строки;
- **texttop** – верхняя граница списка выравнивается по самому высокому текстовому элементу текущей строки;
- **left** – список располагается у левого края окна; текст н другие элементы обтекают его справа;
- **right** – список располагается у правого края окна; текст и другие элементы обтекает его слева;
- **accesskey** – определяет клавишу быстрого доступа к раскрывающемуся списку; Например, **<select accesskey="S">;**
- **name** – имя (идентификатор) списка;
- **disabled** – делает список недоступным пользователю;
- **multiple** –возможность выбора из списка одновременно нескольких элементов;
- **size** – количество одновременно видимых элементов списка; по умолчанию – 1, Если это число больше 1, то список снабжается полосой прокрутки;
- **tabindex** – целое число, определяющее порядок перехода к элементу с помощью клавиши <**TAB**>.

Тег <**option**> имеет следующие атрибуты:

- **selected** – обозначает выбранный (выделенный) элемент списка;
- **value** – значение, ассоциированное с элементом списка.

Пример. Задание списков.

Простой список:

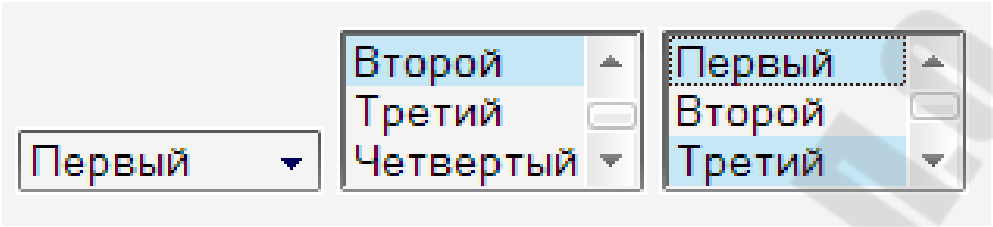
```
<select name="List1">
 <option value=1>Первый
 <option value=2>Второй
 <option value=3>Третий
 <option value=4>Четвертый
</select>
```

Список с заданным количеством видимых элементов:

```
<select name="List2" size="3">
 <option value=1>Первый
 <option value=2 selected>Второй
 <option value=3>Третий
 <option value=4>Четвертый
</select>
```

Список с возможностью одновременного выбора нескольких элементов:

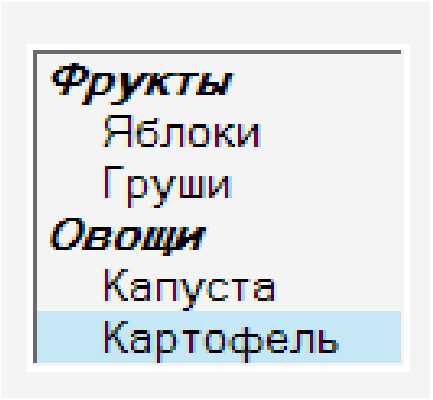
```
<select name="List2" size="3" multiple>
 <option value=1>Первый
 <option value=2 selected>Второй
 <option value=3>Третий
 <option value=4>Четвертый
</select>
```



Для создания иерархических списков внутри тега <**SELECT**> можно использовать контейнерный тег <**optgroup**>, в который заключаются теги <**option**> объединяемых в одну группу. Атрибут **label** тега <**optgroup**> - это невыбираемый элемент списка, выполняющий роль заголовка группы элементов. При этом выбираемые элементы группы, заданные посредством тега <**option**>, отображаются с левым отступом.

Пример. Задание иерархического списка.

```
<select name="List1" size=6>
<optgroup label="Фрукты">
 <option value=1>Яблоки
 <option value=2>Груши
</optgroup>
<optgroup label="Овощи">
 <option value=3>Капуста
 <option value=4 selected>Картофель
</optgroup>
</select>
```



3.5 Текстовая область. Тег TEXTAREA

Контейнерный тег <**textarea**> позволяет задавать текстовую область. Это прямоугольная область с полосами прокрутки. Внутри размещается многострочный текст, который можно сделать как редактируемым, так и только для просмотра. Текстовую область используют как большое поле ввода данных. Важной особенностью тег

<**textarea**> является то, что все теги, заключенные в него, не выполняются браузером, а отображаются как обыкновенный текст. Тег имеет следующие атрибуты:

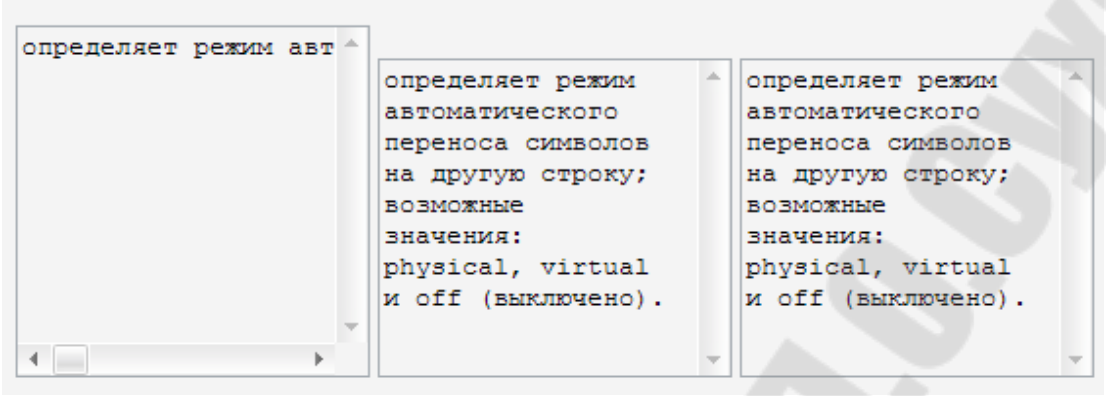
- **align** – горизонтальное выравнивание текстовой области (значения как у SELECT);
- **cols** – количество символьных позиций по горизонтали;
- **rows** – количество строк, в окне текстовой области;
- **readonly** – устанавливает текстовую область в режим просмотра;
- **name**– имя (идентификатор) текстовой области;
- **tabindex** – целое число, определяющее порядок перехода к элементу с помощью клавиши <**TAB**>;
- **disabled** – делает список недоступным пользователю;
- **wrap** – определяет режим автоматического переноса символов на другую строку; возможные значения: **physical**, **virtual** и **off** (выключено). По умолчанию включен автоматический перенос посимвольно.

Если атрибуту **wrap** присвоено значение **virtual**, текст переносится в области ввода только для представления пользователю, но передается серверу, как если бы никаких автопереносов строки не было, за исключением тех случаев, когда нажималась клавиша <**Enter**>. Если атрибуту **wrap** присвоено значение **physical**, текст переносится в окне и отправляется серверу, как если бы пользователь его именно так и набрал. Это самый полезный способ употребления атрибута **wrap**, поскольку текст отправляется именно в том виде, в каком пользователь видит его в области ввода.

Пример. Задание текстовой области с различными вариантами переносов.

```
<textarea rows=10 cols=20 name=text1 wrap=off>
определяет режим автоматического переноса символов на другую строку; возможные значения: physical, virtual и off (выключено).
</textarea>

<textarea rows=10 cols=20 name=text1 wrap=virtual >
определяет режим автоматического переноса символов на другую строку; возможные значения: physical, virtual и off (выключено).
</textarea>

<textarea rows=10 cols=20 name=text1 wrap=physical>
```

определяет режим автоматического переноса символов на другую строку; возможные значения: physical, virtual и off (выключено).
</textarea>



CSS. Каскадные таблицы стилей
4. Назначение стилевых таблиц

Собственные средства HTML (теги и их атрибуты) выполняют две основные роли: поддержку структуры документа (состав и взаимосвязи элементов) и определение внешнего вида визуальных элементов. Идея разделения описания внешнего вида документа от элементов, определяющих его структуру, воплотилась в технологии, называемой **каскадными таблицами стилей** (Cascading Style Sheets, **CSS**). Таблица стилей, действует подобно шаблону форматирования, и может быть разработана отдельно от HTML-документа, а затем применена к нему. Изменяя содержимое таблицы стилей, можно изменять внешний вид HTML-документов, не затрагивая их структуры и информационного содержания. Одна и та же таблица стилей может применяться к нескольким документам, и, наоборот, к одному и тому же документу может быть применено несколько таблиц стилей. В последнем случае браузер учитывает приоритеты таблиц и по определенным правилам разрешает возникающие конфликты, в результате чего таблицы выстраиваются неким каскадом.

Кроме технологичности стилизации HTML-документов, CSS обеспечивают еще две важные вещи: произвольное позиционирование элементов и создание визуальных эффектов, таких как полупрозрачность и трансформации графических изображений и текстов.

4.1 Встраивание таблиц стилей в HTML-документ

Для применения каскадной таблицы стилей к HTML-документу необходимо ее связать с ним или встроить в него. Это можно сделать четырьмя способами:

1. Вставка непосредственно в заголовок HTML-документа. Правила таблицы стилей заключаются в контейнерный тег **<style>.**

```
<head>
    <style>
        H1 {color: blue;  font-size: 24pt}
        b {font-style: italic}
    </style>
</head>
```

2. Вставка непосредственно тег виде строки описания в атрибуте **style.**

```
<h1 style="color: blue;  font-size:24pt">
```

3. Импорт - вставка таблицы стилей из внешнего файла. Файл таблицы стилей является текстовым файлом с расширением **css**. Оператор **@import** используется **перед другими** правилами таблицы стилей в контейнере **<style>** или в **css**-файле.

```
<style>
    @import: url(http://www.myserver.ru/css/mystyle.css)
</style>
```

4. Связывание с таблицей стилей в внешнем файле с помощью ссылки задаваемой тегом **<link>**, который помещаемого в контейнер **<head>**. Общий вид:

```
<link rel=stylesheet type=text/css href="URL-адрес файла ">
```

Пример.

```
<head>
    <link rel=stylesheet type=text/css
    href="http:// www.myserver.ru/css/mystyle.css ">
</head>
```

Таблица стилей, вставленная с помощью тега **<style>,** действует на элементы только текущего HTML-документа, в котором этот тег находится. Если ту же таблицу необходимо применить и к другим документам, то ее код придется повторить в соответствующих HTML-страницах. При этом возрастает общий объем файлов сайта, а также трудозатраты в случае необходимости изменить его стиль. Чтобы избежать этого, используют импорт или связывание таблиц из внешних **css**-файлов. Когда требуется изменить параметры стилей для отдельных элементов (например, их оформление), используют атрибут **style.** Возможно также комбинирование всех способов встраивания таблиц.

При использовании внешних стилей авторы поступают совершенно по-разному. Одни хранят все стили сайта в одном файле, другие – в нескольких. Например, один файл содержит стили для всего сайта и присутствует на всех страницах, другой – дополнительно подключается только к определенным разделам, третий – к определенным подразделам и т.д.

В записях таблиц стилей можно задавать комментарии которые задаются символами /* и */. Тег **<style>** имеет следующие атрибуты:

type - для каскадных таблиц стилей всегда имеет значение **text/css;**

media - определяет тип устройства вывода. Браузеры обычно используют следующие значения: **screen** (экран), **print** (печать) и **all** (все). Можено создать стили отдельно для отображения документа на экране монитора и для вывода на печать.

4.2 Типы селекторов

CSS состоит из **правил**, а каждое правило – из **селектора** и **блока объявлений**. Блок объявлений содержит CSS-свойства, определяющие отображение элемента веб-страницы в браузере. Селектор отвечает за выбор этого самого элемента. Селектор служит для однозначной идентификации HTML элемента средствами CSS. Он позволяет выбирать именно тот элемент (или группу элементов), который нужен. С помощью **простых** селекторов можно выбирать:

- все объекты – универсальный селектор;
- объекты определенного типа;
- объекты с заданным классом;
- объект с определенным идентификатором;
- объекты с определенными характеристиками – селекторы атрибутов.

Объединяя простые селекторы можно выбирать объекты по более сложным правилам:

- объекты, находящиеся внутри какого-то объекта – селектор потомка;
- объекты, непосредственно вложенные в какой-то объект – дочерний селектор;
- объект, расположенный после другого объекта – сестринский селектор.

Также существуют селекторы псевдоклассов и псевдоэлементов. Они позволяют назначать стили элементам, которые зависят не только от разметки, но и от состояния документа.

4.2.1 Универсальный селектор

Предназначен для выбора всех элементов. Стили, указанные для универсального селектора применяются ко всем элементам сразу.

Обычно применяется для сброса зависящих от браузера начальных значений стилей (в частности, отступов).

```
* {
        padding: 0;
        margin: 0;
}
```

4.2.2 Селектор типа

Предназначен для выбора всех элементов определенного типа. Стили применяются ко всем элементам указанного типа не зависимо от уровня вложенности. Применяется для задания общих, для всех элементов определенного типа, стилей. Общий вид:

```
Имя тега { описание }
```

Пример. Для всех абзацев (тег p) установить размер шрифта 12px.

```
p {font-size: 12px;}
```

4.2.3 Селектор класса

Предназначен для выбора всех элементов по имени класса (по значению атрибута **class**). Стили применяются к любым тегам с соответствующим классом. Важно учитывать, что в отличие от названий HTML-тегов, в названиях классов различаются большие и маленькие буквы. То есть *class="active"* и *class="Active"* – это совсем разные классы. Общий вид:

```
.Имя класса { описание }
```

Пример. Задать зеленый цвет текста в любых тегах с классом **active**.

```
.active {color: #0f0; }
```

Комбинируя селектор класса и селектор типа, можно объединить их свойства.

Пример. Задать зеленый цвет текста только для абзацев с классом **active**.

```
p.active {color: #0f0; }
```

HTML позволяет задавать в качестве значения атрибута **class** список разделенных пробелами названий (порядок следования не имеет значения). Другими словами, один элемент может иметь сразу несколько классов:

```
<div class="panel hint active"></div>
```

Стили, относящиеся к каждому из перечисленных классов будут, объединяясь, применяться к этому элементу. IE6 не понимает такой записи.

Пример. Правило распространяется только на элементы, в списке классов которых встречаются (в любом порядке) и класс **popup** и класс **active**.

```
.popup.active { color: #0f0;}
```

Концепция классов, наиболее часто применяется при верстке. Например, верстать страницу, используя в качестве контейнеров теги div, задавая им определенные классы (в соответствии с функциональным назначением):

```
<div class="header">
    <!--элементы шапки сайта-->
</div>
<div class="sideBar">
    <!--элементы панели меню-->
</div>
<div class="content">
    <!--основное содержимое-->
</div>
```

4.2.4 Селектор идентификатора

Предназначен для выбора элемента по уникальному идентификатору (значению атрибута id). Позволяет задать стили конкретному HTML-элементу. Общий вид:

```
#Имя идентификатора { описание }
```

Пример. Задать зеленый цвет текста в любом теге с id="active"

```
#active { color: #0f0; }
```

Комбинируя селектор идентификатора и селектор типа, можно объединить их свойства. Учитывая, что id на странице должен быть уникальным, смысла в таком комбинировании не много. Как правило, идентификаторы применяются там, где предполагается работа скриптов. Например, в формах или в каких-то динамических элементах. Так же с помощью id можно подчеркнуть уникальность элемента, его присутствие в единственном экземпляре.

Пример. Задать зеленый цвет только для элемента списка с данным id.

```
li#active { color: #0f0; }
```

4.2.5 Селектор атрибутов

Предназначен для выбора всех элементов по наличию или значению заданного атрибута. IE6 не понимает этот селектор. Способы задания:

Element[attrName]{ описание } – элементы с атрибутом
[attrName] { описание } – любой элемент с атрибутом

Пример. Выделим жирным шрифтом все элементы, для которых задана всплывающая подсказка:

```
[title] {font-weight: bold;}
```

Element [attrName=attrValue]{ описание } – элемент с заданным значением атрибута

Пример. Зададим цвет фона только для тех **input**, значение атрибута **type** которых в равно "**submit**" (для кнопки отправки формы).

```
input[type="submit"] {background: #0f0;}
```

Element[attrName~=attrValue] { описание } – элементы, заданный атрибут которых в перечне значений имеет определенное слово.

Пример. Зададим обтекание для тех элементов **div**, в перечне значений атрибута **class**, которых присутствует "**sideBar**" (вот это-то и есть, фактически, селектор класса):

```
div[class~="sideBar"] { float: left; }
<!--Будут выбраны элементы:-->
<div class="sideBar"><div>
<div class="sideBar userBar"><div>
<div class="contentPanel sideBar"><div>
<!--Не будет выбран элемент:-->
<div class="sideBarInner"><div>
```

Element[attrName|=attrValue] { описание } –элементы, заданный атрибут которых равен определенному значению или начинается с этого значения, после которого идет дефис.

Пример. Зададим стили для всех элементов, язык которых задан в атрибутах, как английский (en, en-us, en-gb, en-au и т.д.):

```
[lang|="en"] { какие-то стили }
<!--Будут выбраны элементы:-->
```

```
<div lang="en"><div>
<div lang="en-us"><div>
<p lang="en-au"><p>
<!--Не будут выбраны элементы:-->
<div lang="ru"><div>
<div lang="english"><div>
```

CSS3 несколько расширяет возможности выбора по конкретным значениям атрибутов и дополняет их еще тремя вариантами.

Element[attrName^=attrValue] – все элементы, заданный атрибут которых начинается с определенного значения.

Пример. Зададим стили для всех элементов, атрибут title которых начинается со слова "Hint":

```
[title^="Hint"] { какие-то стили }
<!--Будут выбраны элементы:-->
<div title="Hint. Did you know that…"><div>
<div title="Hint…"><div>
<span title="Hinting…"><span>
<!--Не будет выбран элемент:-->
<div title="Important Hint…"><div>
```

Element[attrName$=attrValue] – все элементы, заданный атрибут которых оканчивается на определенное значение.

Пример. Зададим стили для всех элементов img, с расширением gif:

```
<!--Будет выбран элемент:-->
<img src="/images/user.gif" height="64px" width="64px" alt="аватарка"/>
<!--Не будет выбран элемент:-->
<img src="/images/user.png" height="64px" width="64px" alt="аватарка"/>
```

Element[attrName*=attrValue] – все элементы, заданный атрибут которых содержит определенную подстроку.

Пример. Зададим стили для всех ссылок, в адресах которых есть подстрока "xiper.net"

```
a[href*="xiper.net"] { какие-то стили }
<!--Будут выбраны элементы:-->
<a href="http://www.xiper.net/">Главная</a>
<a href="xiper.net">Это тоже главная</a>
<a href="http://www.xiper.net/learn/css/">Уроки CSS</a>
<!--Не будет выбран элемент:-->
```

```
<a href="www.xipernet">Ошибка в адресе!</a>
```

Для любых, из перечисленных выше селекторов можно сделать выбор сразу по нескольким атрибутам. **Например**:

```
img[width][height][alt] {
эти стили применятся только к изображениям, у которых есть все три перечисленных атрибута
}
a[title][href*="xiper.net"] {
эти стили применятся только к ссылкам, в адресах которых есть подстрока "xiper.net" и для которых задан атрибут title
}
```

4.2.6 Селектор псевдоклассов

Предназначен выбор всех элементов по определенному псевдоклассу. **Псевдокласс** – это фантомный класс, наличие которого зависит от состояния элемента или структуры документа в целом. В определенных условиях у HTML-элемента автоматически появляется/пропадает некий класс. Названия этих классов-призраков и условия их появления известны заранее и это позволяет применять к таким элементам различные стили (т.е. менять их оформление).

Набор псевдоклассов в CSS3 значительно расширен, но, к сожалению, большая их часть не поддерживается IE, даже до 9-ой версии (у других браузеров, традиционно поддержка значительно лучше). В CSS2 определены 4 группы псевдоклассов (часть из которых перекочевала из CSS1):

1. **псевдокласс first-child** – задает стиль для элемента, являющегося первым потомком своего родителя.

Пример. Убираем маркер у первого элемента списка:

```
li:first-child{list-style: none; }
```

2. **Псевдоклассы ссылок** – В этой категории два псевдокласса **:link** и **:visited**. Позволяют задавать стили для элементов, являющихся ссылками и для тех ссылок, на которые пользователь уже нажимал.

Пример. Задаем цвет для посещенных ссылок:

```
a:visited {color: #80bd34; }
```

3. **Динамические псевдоклассы** – применяются к элементам в зависимости от действий пользователя. CSS2 определяет три таких псевдокласса – :**hover**, :**active** и :**focus**:

- :**hover** – элемент, над которым находиться указатель мыши. Как только указатель уходит за пределы элемента, стили, заданные этим псевдоклассом, отменяются.
- :**active** – элемент, активированный пользователем (например, ссылка или кнопка в момент щелчка).
- :**focus** – элемент, которому принадлежит фокус ввода.

Динамические псевдоклассы часто применяются для «оживления» странички – создания эффекта реакции элементов на действия пользователя без применения скриптов.

Пример. При наведении мыши на ссылку убираем подчеркивание:

```
a:hover { text-decoration: none; }
```

4. **Псевдокласс :lang** – используется для выбора элементов по их языку. Код языка указывается в круглых скобках. При этом, согласно спецификации, информация о языке может быть получена и нескольких источников, таких как атрибут **lang**, метатег **Content-language**, информация из протокола (**HTTP**-заголовок).

Пример. Все французские надписи отобразим красным цветом:

```
p:lang(fr){ color: red; }
```

При желании можно объединять селекторы псевдоклассов. Заданные таким образом стили будут применяться только к элементам, у которых будут все перечисленные псевдоклассы.

Пример. Скроем маркер первого элемента списка, а при наведении на него мышки – отобразим:

```
li:first-child{ list-style: none; }
li:first-child:hover{ list-style:disc; }
```

4.2.7 Селекторы псевдоэлементов

Предназначены для настройки стилей псевдоэлементов, а в некоторых случаях и их генерации. Псевдоэлементы – это условные фиктивные элементы, которые являются частью существующих элементов или вводятся в документ дополнительно.

Селектор псевдоэлементов с точки зрения синтаксиса – это фактически селектор псевдоклассов. Разница только в выбираемых элементах. В CSS2 определено 4 псевдоэлемента.

1. **Псевдоэлемент :first-letter** – позволяет задавать стили для первой буквы внутри элемента.

Пример. Увеличить первую букву абзацев:

```
p:first-letter { font-size: 120%;}
```

2. **Псевдоэлемент :first-line** – применяет стиль ко всей первой строке.

3. **Псевдоэлементы :after и :before** – позволяет вставить сгенерированный на ходу контент после или до определенного элемента. Для того, чтобы задать вставляемый текст используется специальное свойство **content**.

Пример. Реализуем FAQ в виде списка определений. При этом автоматически генерируем слова «вопрос» и «ответ»(в исходном HTML их не будет, а на страничке они отобразятся):

HTML

```
<dl>
    <dt>Тут будет какой-то вопрос</dt>
    <dd>А тут ответ</dd>
</dl>
```

CSS

```
dt::first-line{ content: "Вопрос: "; }
dd:before{ content: "Ответ: "; }
```

В CSS-селекторе псевдоэлемент должен обязательно идти последним. Например, такая запись не сработает, так как после псевдокласса идет еще и span:

```
p:first-line span{ color: #80bd34; }
```

4.2.8 Составные селекторы. Комбинаторы

Составные селекторы очень удобны, так как они позволяют выбирать элементы на основании их размещения в дереве документа. То есть при выборе можно, например, манипулировать такими понятиями как «является потомком», «следует за», «лежит непосредственно внутри». Если учесть, что вид и количество простых селекторов, связываемых комбинаторами, никто не ограничивает, то становится понятно, какой мощный инструмент вложили в наши руки разработчики

CSS. Используя составные селекторы можно, допустим, задать стили для ссылки, на которую навели мышкой, внутри абзаца, который расположен сразу за заголовком h1. Для комбинации селекторов используются символы представленные в таблице 4.1.

Таблица 4.1 – Табличка комбинаторов

Обозначение	Название соответствующего селектора	Версия CSS
пробел	селектор потомка	2
>	селектор дочерних элементов	2
+	селектор сестринского элемента	2
~	селектор обобщенных родственных элементов	3

4.2.9 Селектор потомка

Предназначены для выбора заданных потомков определенных элементов. Общий вид:

```
Селектор1 Селектор2 { описание }
```

Пример. Зададим цвет текста для тех элементов span, которые находятся внутри элементов p.

```
p span { color: #333; }
```

Селектор выбирает всех потомков не зависимо от их уровня вложенности. Например, применим вышестоящий CSS к такому коду:

```
<p>Черный текст <span>серый текст</span> черный текст </p>
<span>Черный текст</span>
<p>Черный текст <a href="#">Черный текст <span>серый текст</span>
</a>черный текст</p>
```

На span, который находится внутри ссылки тоже будут распространятся заданные стили.

Цепочка селекторов потомка не ограничена двумя селекторами. **Например**, зададим цвет всем элементам span, среди предков которых есть элемент a, который в свою очередь вложен в элемент div:

```
div a span { color: #333; }
```

Из всех составных селекторов селектор потомков употребляется чаще всего. С помощью него можно разграничить область влияния стилей.

Например: Задать стили который применяется только для всех li внутри контейнера с классом mainMenu.

```
.mainMenu li { float: left; }
```

Не нужно злоупотреблять вложенностью селекторов. Чем проще селекторы, тем быстрее их обработка браузером.

4.2.10 Селектор дочерних элементов

Предназначены для выбора дочерних элементов. Селектор дочернего элемента выбирает потомков только первого уровня – то есть непосредственно вложенные элементы. Общий вид:

```
Селектор1 > Селектор2 { описание }
```

Пример. Зададим отступ слева для списка, непосредственно вложенного в элемент с классом content (на списки второго уровня вложенности эти правила не подействуют):

```
.content > ul {margin-left: 20px;}
```

Применяется, как правило, для уменьшения объема CSS, там, где нужно разделить стили непосредственно вложенных элементов и элементов второго (и более) уровня вложенности. Например, при оформлении вложенных списков (допустим, выпадающих меню). Еще одно применение – специфическое оформление элементов с известными заранее родителями/

4.2.11 Селектор сестринского элемента

Предназначены для выбора элемента, расположенного непосредственно за другим заданным элементом и имеющего с ним общего родителя. Общий вид:

```
Селектор1 + Селектор2 { описание }
```

Пример. Выделить жирным текстом элемент span, следующий сразу после заголовка h1 (при условии, что у них есть общий родитель):

```
h1 + span { font-weight: bold;}
```

Этот span будет жирным.

```
<h1>Очень полезная статья</h1>
<span>Тут указан, например, автор. Жирный текст</span>
<p>Далее идет текст статьи</p>
```

Этот span не будет выделен жирным – между ним и h1 имеется дополнительный элемент.

```
<h1>Очень полезная статья</h1>
Далее идет текст статьи.
```

```
<span>Мы с h1 сестринские?</span>
```

4.2.12 Селектор обобщенных родственных элементов

Предназначен для выбора элемента (элементов), идущего после другого заданного элемента и имеющего с ним общего родителя. В отличие от сестринского селектора, который выбирает только элемент идущий непосредственно за заданным, селектор обобщенных родственных элементов выбирает все элементы, которые идут после заданного. Общий вид:

```
Селектор1 ~ Селектор2 { описание }
```

Пример. Радать отступ красной строки всем абзацам, которые следуют после h1:

```
h1 ~ p { text-indent: 20px; }
```

4.3 Иерархия стилей

Наследование – это механизм с помощью которого стили применяются не только к самим элементам, но и к их потомкам. При написании CSS может возникнуть ситуация, когда для одного и того же элемента подходит сразу несколько правил. **Например**:

```
div {color: #0f0; font-weight: bold;} /* зеленый цвет */
.box {font-weight: normal;}
body {color: #f00; } /* красный цвет */
}
/* будет синий зеленый но не жирные */
<div class="box">Какого же я цвета?</div>
```

И первое и второе правила подходят для нашего элемента div (в первом случае мы выбираем его, используя селектор типа, во втором – селектор класса). Более того, третье правило тоже нам подходит. Поскольку div наверняка лежит внутри тега body и, согласно вышеприведенному определению, должен унаследовать значение его свойства color.

Текст внутри тега div, не может быть одновременно трех цветов. Должны существовать правила, согласно которым точно рассчитывается, какое из определений будет применено. Такие правила называют – правила каскадирования.

Для отображения элемента браузер должен:

1. Найти все правила, селектор которых подходит для выбора данного элемента.
2. Разделить все объявления из найденных правил на группы согласно важности и источника:
    a. важные объявления читателя;
    b. важные объявления автора;
    c. обычные объявления автора;
    d. обычные объявления читателя;
    e. объявления агента пользователя.
3. В пределах групп провести сортировку объявлений по специфичности.
4. Для равных специфичностей отсортировать по очередности расположения.

После остается выбрать победившие объявления и сформировать из них правило, применяющееся к данному конкретному элементу.

Если очень нужно, можно пометить какое нибудь объявление, как важное (**important**). Такое объявление будет считаться заведомо победившим

```
div {
    color: #00f !important;  /* важное объявление - сразу победа! */
}
```

4.4 Единицы измерения

Единицы измерения CSS используются для указания размеров различных элементов. Есть абсолютные и относительные единицы измерения. Абсолютные единицы не зависят от устройства вывода, а относительные единицы определяют размер элемента относительно значения размера, используемого в родительском элементе. Ниже представлена таблица и подробное описание каждой единицы измерения CSS.

Абсолютные единицы:

1. **Миллиметр - mm, сантиметр - cm и дюйм – in**. Само собой разумеется, что это абсолютные единицы измерения. Один cm = 0.39370in, 1in = 2.54cm и 10mm = 1cm. Компьютерные дисплеи плохо вычисляют данные единицы измерения, таким образом у этих величин ограниченное применение и обычно их используют при указании размера для вывода страниц на печать.

2. **Пиксели – px**. Пиксель это маленькая точка на экране. Пиксели определяют размер элемента. Использование пикселей дает вам точный контроль над размером элемента, позволяя вам точно вычислить его ширину и высоту, это будет полезным для точной разметки дизайна страницы. Однако, есть несколько минусов использования пикселей, вы должны быть осведомлены о них:

Во-первых, установка размера шрифта с помощью пиксельных единиц не позволяет пользователю изменять размер шрифта с помощью настроек в браузере. Если размер шрифта 12 пикселей, он всегда будет иметь высоту 12 точек, независимо от того, что пользователь установил размер шрифта по умолчанию в браузере. Так что если вы решили указать размер шрифтов, следует подумать об использовании другой единицы измерения.

Во-вторых, когда речь заходит о печатных средствах массовой информации, пиксели не имеют реального значения. При разработке документа для печати, устройство печати должно будет само догадываться о том, что вы имели ввиду с точки зрения физических размеров. Хотя обычно можно просмотреть документ и внести изменения перед печатью.

3. **Точки - pt и пики – pc.** Эти единицы измерения чаще всего используются для указания именно размера шрифта, например, в обычном блокноте, мы часто указываем размер текста равный 14, это значение как раз и есть единицы измерения pt.

Точка (1pt) равна 1/72 дюйма, в то время как пика (1pc) равна 1/6 дюйма (1pc = 12pt). Документы, предназначенные для печати будут иметь возможность сообщить устройству именно тот размер шрифта, который следует использовать при печати. Цифровым дисплеям, однако, придется самим догадываться, как конвертировать эти единицы в пиксели, и нет никаких реальных универсальных способов узнать, как это будет сделано. Поскольку точки (pt) были использованы с первых дней Интернета, большинство браузеров автоматически могут установить соотношения между пикселями и точками, но это по сути неправильно. Помните, что небольшие дисплеи сегодня могут иметь высокое разрешение, так что определить то, насколько большой «дюйм на экране» будет практически невозможно на устройствах. Для указания размера элементов на веб-страницах следует избегать использования этих единиц.

Относительные единицы:

1. **Процент - %.** Самая простая единица измерения, это процент (%), она не имеет напрямую никакого отношения к размеру шрифта или элемента в целом и может быть использована в комбинации с другими единицами измерения указывающими величину. Размер установленный в процентах, напрямую зависит от размера родительского элемента, например размер шрифта задается относительно размера шрифта родительского элемента, также высота и ширина выражается в процентах относительно высоты и ширины родительского элемента.

2. **Вычисляемая x-высота – ex.** Единица измерения ex используется достаточно редко. В качестве основы для размера 1ex используется высота символа "x" в нижнем регистре указанного шрифта. Большинство браузеров не поддерживает эту единицу измерения должным образом, и она не рекомендуется для использования в документах, предназначенных для браузеров.

3. **Вычисляемая единица – em.** Em является относительной единицей измерения. Один em равен 16px. Если em используется для определенного элемента, то за 1em принимается размер шрифта его родителя.

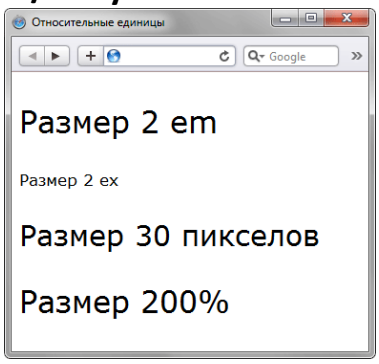
Примеры использования едениц измерения.

a	Описание	Пример
%	проценты	**p{border: 5%;}**
in	дюйм	**p{word-spacing: 2in;}**
cm	сантиметры	**div {margin-left: 2cm;}**
mm	миллиметры	**p{font-size: 15mm;}**
em	1em равен текущему размеру шрифта. Если текущей размер шрифта составляет 12pt, тогда 2em будет равен 24 pt	**p{letter-spacing: 7em;}**
ex	1ex равен высоте символа "x" в нижнем регистре указанного шрифта	**p{line-height: 3ex;}**
pt	1pt равен 1/72 дюйма	**body{font-size: 18pt;}**
pc	1pc равен 12pt	**p{font-size: 20pc;}**
px	пиксель - это маленькая точка на экране	**p{padding: 25px;}**

Пример. Использование относительных единиц

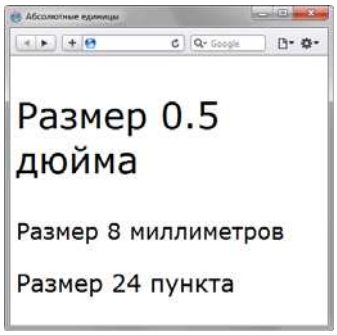
```
<head>
<style type="text/css">
  .em, .ex, .px, .percent {
   font-family: Verdana, Arial, sans-serif;
  }
```

```
  .em { font-size: 2em; }
  .ex { font-size: 2ex; }
  .px { font-size: 30px; }
  .percent { font-size: 200%; }
 </style>
</head>
<body>
 <p class="em">Размер 2 em</p>
 <p class="ex">Размер 2 ex</p>
 <p class="px">Размер 30 пикселов</p>
 <p class="percent">Размер 200%</p>
</body>
```



Пример. Использование абсолютных единиц

```
<style type="text/css">
  .in, .mm, .pt {
   font-family: Verdana, Arial, sans-serif;
  }
  .in { font-size: 0.5in; }
  .mm { font-size: 8mm; }
  .pt { font-size: 24pt; }
 </style>
</head>
<body>
 <p class="in">Размер 0.5 дюйма</p>
 <p class="mm">Размер 8 миллиметров</p>
 <p class="pt">Размер 24 пункта</p>
</body>
```



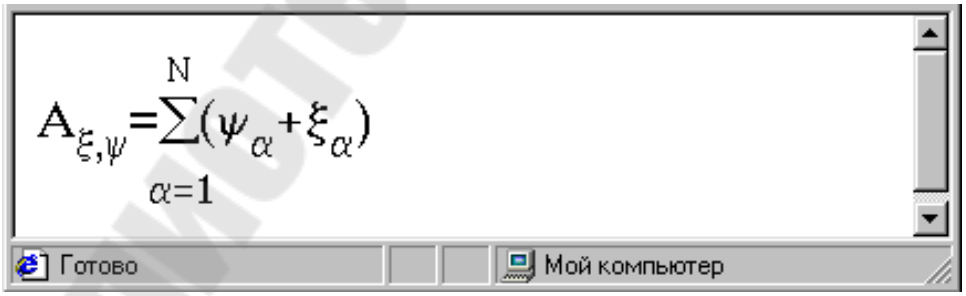
4.5 Описание шрифтов

Для описания шрифтов с помощью CSS применяются следующие стилевые параметры:

1. **font-family** - определяет список семейств шрифтов. Обычно задаются несколько похожих шрифтов в порядке предпочтения на случай, если в компьютере пользователя нет нужного шрифта При указании имени группы шрифтов, как показано выше, браузер подбирает подходящий для отображения шрифт данной группы из имеющегося набора шрифтов. Если название шрифта состоит из нескольких слов, то оно заключается в кавычки. Список шрифтов желательно завершить родовым именем шрифта: **serif, sans-serif, cursive, fantasy** или **monospace.** Например, для шрифта Times родовым является **serif,** для Helvetica – **sans-serif,** для Courier – **monospace.**

Пример. Прямое указание гарнитуры шрифта

```
<SPAN STYLE="font-family:symbol; padding-left:65px;">N</SPAN>
<BR>
<SPAN STYLE="font-family:symbol; font-size:24px;">A
<SUB>x,y</SUB>=e(y<SUB>a</SUB>+x<SUB>a</SUB>) </SPAN> <BR>
<SPAN STYLE="font-family:symbol; padding-left:60px; font-size:20px;">a=1
</SPAN>
```



2. **font-size** – размер шрифта. Значение может быть задано различными способами:

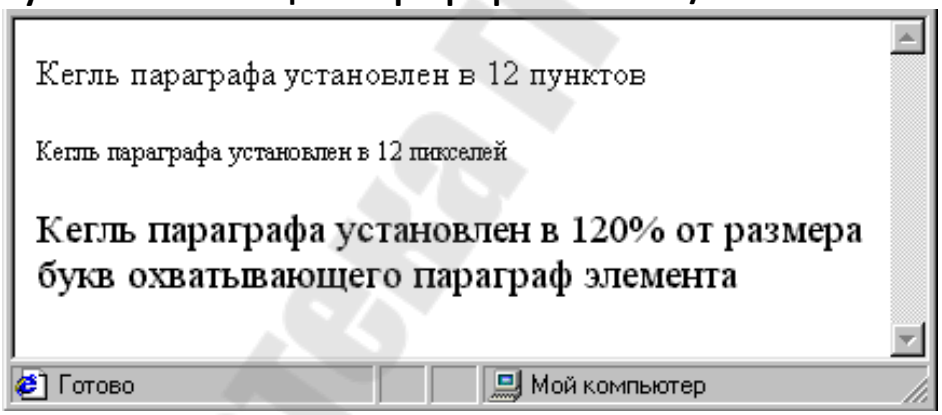
– абсолютный размер, задаваемый с помощью ключевых слов**: xx-small, x-small, medium, large, x-large, xx-large**. Эти значения

представляют индексы таблицы размеров шрифтов, поддерживаемой браузером. По умолчанию используется значение **medium** (средний);

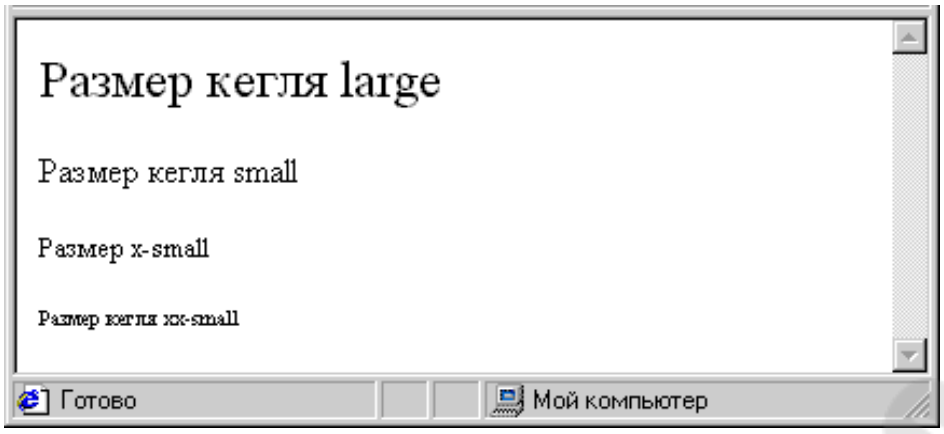
– относительный размер, задаваемый с помощью ключевых слов: **larger** (больше) и **smaller** (меньше). Эти значения интерпретируются относительно таблицы размеров шрифтов браузера и размера шрифта элемента-родителя. Например, если элемент-родитель имеет шрифт размером **medium**, то значение **larger** для элемента-потомка сделает размер шрифта равным **large** (большой). В CSS1 масштабирующий множитель равен 1,5, в CSS2 – 1,2. Размер шрифта относительно элемента-родителя можно задавать и в процентах;
– размер, задаваемый в абсолютных единицах длины (обычно pt=0,35мм).

Пример. Задание размера шрифта.

```
<P STYLE="font-size:12pt;">
Кегль параграфа установлен в 12 пунктов</P>
<P STYLE="font-size:12px;">
Кегль параграфа установлен в 12 пикселов</P>
<P STYLE="font-size:120%;">
Кегль параграфа установлен в 120% от размера
букв охватывающего параграф элемента</P>
```




```
<P STYLE="font-size:large;">Размер кегля large</P>
<P STYLE="font-size:small;">Размер кегля small</P>
<P STYLE="font-size:x-small;">Размер кегля x-small</P>
<P STYLE="font-size:xx-small;">Размер кегля xx-small</P>
```



3. **font-weight** – жирность шрифта. Возможные значения: **normal**, **bold** (жирный), **bolder** (жирнее), **lighter** (светлее) или одним из девяти целых чисел от 100 до 900. Ключевому слову **normal** соответствует числовое значение 400; ключевое слово **bold** задает обычный жирный (полужирный) шрифт и соответствует числовому значению 700. Следует иметь в виду, что в текущем семействе может и не быть шрифта с заданной степенью жирности. В этом случае можно лишь гарантировать, что шрифт с большим значением параметра **font-weight** будет не светлее, чем шрифт с меньшим значением этого параметра.

4. **font-style** – стиль шрифта. Возможные значения: **normal** (нормальный прямой), **italic** (курсив) и **oblique** (наклонный). По умолчанию применяется значение **normal**. Если уже имеется готовый шрифт, соответствующий заданному стилю, то он будет применен. В противном случае текущий шрифт будет изменен программным способом.

5. **font-variant** – вариант стиля шрифта. Возможные значения: **normal** (принимается по умолчанию) и **small-caps**. Значение **normal** не влияет на отображение шрифта. Значение **small-caps** заменяет строчные буквы прописными, но делает их несколько меньшими по размеру, чем прописные буквы текущего шрифта.

Пример. Задания свойств шрифта

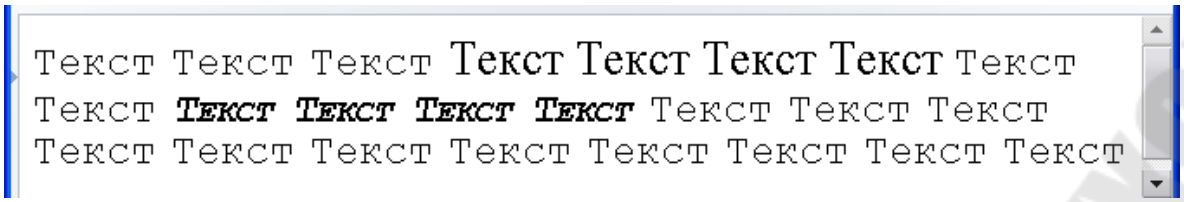
```
<html>
<head>
<style>
   #MyType1 {font-family: "Times New Roman", serif; font-size: larger}
   #MyType2 {font-weight: bolder; font-style: oblique; font-variant:small-
   caps}
</style>
```

```
<body style="font-family:fantasy; font-size:18pt">
   <p>Текст Текст Текст <span id=MyType1>Текст Текст Текст Текст
   </span>Текст Текст <span id=MyType2>Текст Текст Текст Текст
   </span>Текст Текст Текст Текст Текст Текст Текст Текст Текст </p>
</body>
</html>
```

6. **font** – позволяет установить сразу несколько свойств шрифта в одном определении: **font-style, font-variant, font-weight, font-size, line-height, font-family**. Значения этих параметров указываются через пробел в том порядке, в котором они были перечислены. Допустимо не указывать первые три параметра, что соответствует значению **normal**, принятому для них по умолчанию. Если задается высота строки (**line-height**), то ее значение отделяется от размера шрифта (**font-size**) прямым слэшем (/). Если список семейств шрифтов (**font-family**) содержит более одного элемента, то последние отделяются друг от друга запятыми.

Пример. Задания свойств шрифта одним определением.

```
<html>
<head>
<style>
   #MyType1 {font: larger  "Times New Roman"}
   #MyType2 {font: oblique  small-caps  bolder 18pt}
</style>
<body style="font-family:fantasy; font-size:18pt">
<p>Текст Текст Текст <span id=MyType1>Текст Текст Текст Текст
</span>Текст Текст <span id=MyType2>Текст Текст Текст Текст
</span>Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст
</p>
</body>
</html>
```



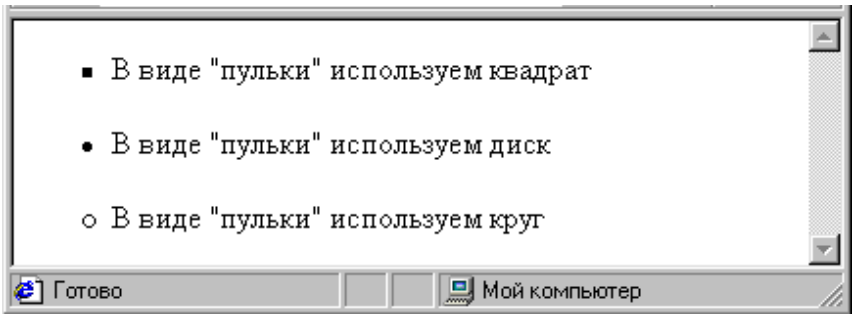
4.6 Оформление списков

При отображении списков CSS позволяет управлять формой и изображением "пулек" (bullets) списка. Параметры задания списка:

1. **list-style-image** - определяет **URL**-адрес графического изображения, используемого в качестве маркера для элемента списка (**none** по умолчанию);
2. **list-style-position** - указывает положение маркеров элементов списка: внутри или вне "**тела**" списка. Принимает значения: **inside** (внутри) или **outside** (значение по умолчанию).;
3. **list-style-type** - определяет вид маркера элементов списка. Принимает значения: **none**, **circle** (окружность), **disk** (затемненный круг), **square** (квадрат), **decimal** (десятичные номера), **lower-alpha** (строчные буквы), **upper-alpha** (прописные буквы), **lower-roman** (маленькие римские цифры), **upper-roman** (большие римские цифры). Значением по умолчанию является **disk**;
4. **list-style** – позволяет задать значения сразу всех или некоторых параметров, описанных выше. Значения указываются через пробел в следующем порядке: **вид__маркера положение URL-адрес**.

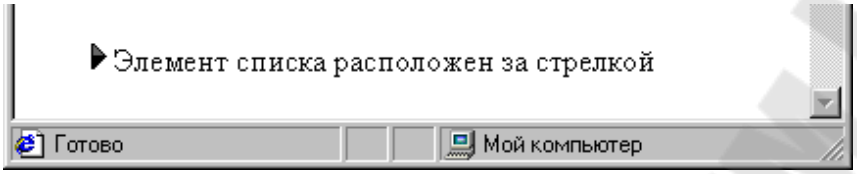
Пример. Задание списков

```
<UL STYLE="list-style-type:square;">
   <LI>В виде "пульки" используем квадрат
</UL>
<UL STYLE="list-style-type:disc;">
   <LI>В виде "пульки" используем диск
</UL>
<UL STYLE="list-style-type:circle;">
   <LI>В виде "пульки" используем круг
</UL>
```



Пример. Задание маркеров списков

```
<UL STYLE="list-style-image:url(barrow.gif);">
   <LI>Элемент списка расположен за стрелкой
</UL>
```



4.7 Задание цвета и фона

Цвет текста определяется с помощью параметра **color**, а цвет фона элемента – с помощью параметра **background-color**. По умолчанию цвет фона элемента определен как прозрачный (**transparent**). Значения этих параметров задаются именем цвета или числовым представлением в системе **RGB**.

Числовое представление цвета допускает следующие варианты:

– шестнадцатеричное число (**#ffff00**);

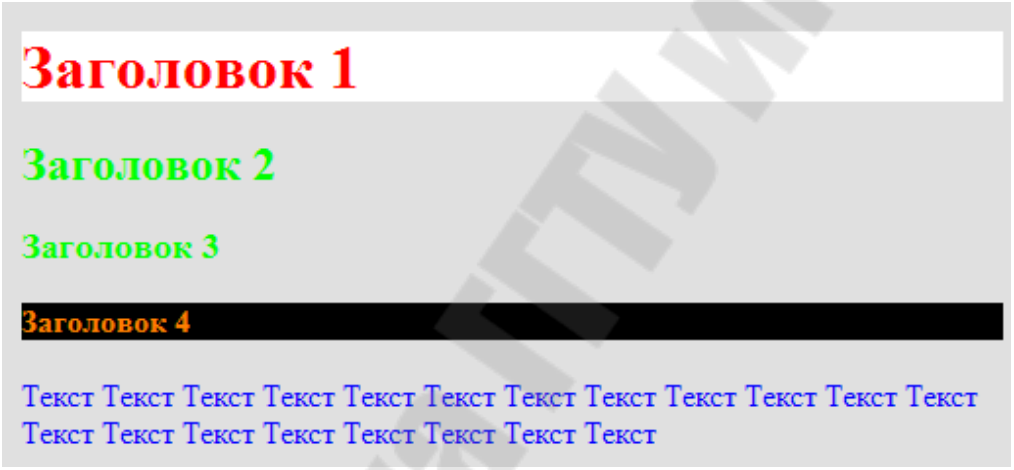
– тройка целых десятичных чисел, каждое из которых представляет яркость одной из **RGB**-составляющих цвета в диапазоне от **0** до **255**. (**rgb(255,255,0)**);

– тройка вещественных чисел со знаками **%**, каждое из которых представляет яркость одной из **RGB**-составляющих цвета в диапазоне от **0** до **100%**. **rgb**(**100%, 100%, 0%**).

Пример. Способы задания цвета.

```
<html>
<head>
<style>
   /* текст синий, фон серый */
   body {color: blue; background-color:#e0e0e0}
   /* заголовок 1-го уровня красный на белом фоне */
   h1 {color: #ff0000; background-color:white}
```

```
    /* заголовок 2-го уровня зеленый на сером фоне */
    h2 {color: rgb(0,  255,  0)}
    /* заголовок 3-го уровня зеленый на сером фоне */
    h3 {color: rgb(0,  255,  0)}
    /* заголовок 4-го уровня оранжевый на черном фоне */
    h4 {color: rgb(100%,  50%,  0%); background-color: black}
</style>
<body>
    <h1> Заголовок 1 </h1>
    <h2> Заголовок 2 </h2>
    <h3> Заголовок 3 </h3>
    <h4> Заголовок 4 </h4>
    <p>Текст Текст Текст Текст Текст Текст Текст Текст Текст
    Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст </p>
</body>
</html>
```



В качестве фона элемента или всего документа, можно использовать изображение из файла по указанному URL-адресу. При этом можно задать способ заполнения, позиционирование изображения, а также указать, должно ли оно перемещаться при прокрутке документа, с помощью следующих параметров:

1. **background-image** - принимает в качестве значения url (URL-адрес изображения) или none (отсутствие изображения).

Пример. Задание фонового изображения

```
<style>
    body {background-image: url(/img/picture.jpg)}
</style>
```

2. **background-repeat** - определяет способ заполнения области элемента изображением. Возможны следующие значения:

– **repeat** - заполнение по горизонтали и по вертикали (по умолчанию);
– **repeat-x** - заполнение по горизонтали;
– **repeat-y** - заполнение по вертикали;
– **no-repeat** - изображение выводится в единственном экземпляре.

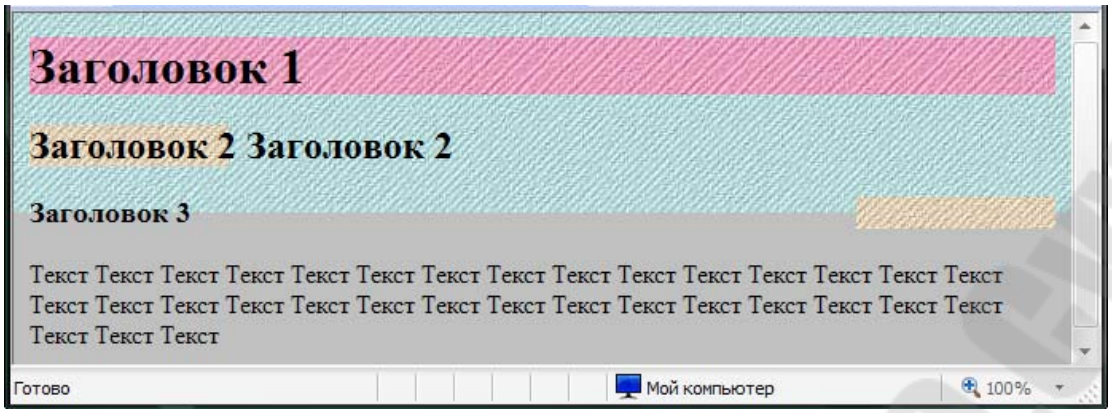
3. **background-attachment** - определяет, будет ли фоновое изображение прокручиваться при пролистывании документа. Значением по умолчанию является **scroll** (будет прокручиваться). Для фиксации изображения используется значение **fixed**;

4. **background-position** – определяет начальное положение фонового изображения с помощью двух значений (горизонтальной и вертикальной координат), разделенных запятой. Значения задаются в виде процентов, в абсолютных значениях длины или в виде ключевых слов: **top**, **center**, **bottom**, **left**, **right**. Значения по умолчанию: **left**, **top**;

5. **background** - позволяет установить значения рассмотренных выше свойств: **background-color, background-image, background-repeat и background-attachment**. Эти значения указываются через пробел. Значение **transparent** означает отсутствие фона.

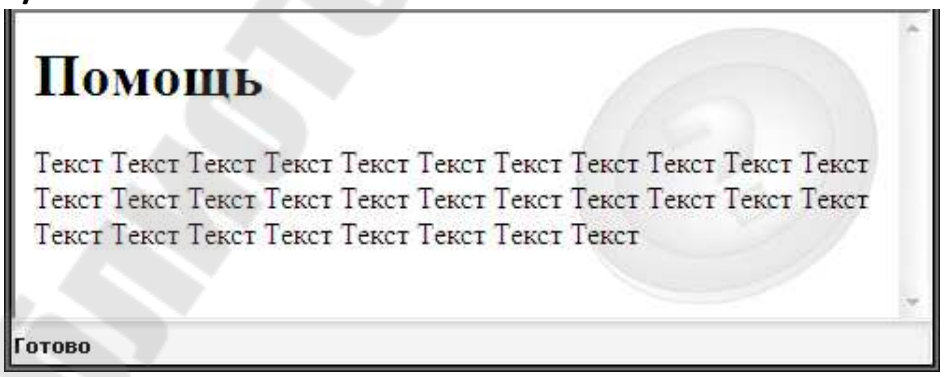
Пример. Способы задания фона

```
<html>
<head>
<style>
   h1{background-image: url(23.jpg);}
   h2{background-image: url(22.jpg); background-repeat:no-repeat;}
   h3{background-image: url(22.jpg); background-repeat:no-repeat;
   background-position:100% 0%}
   body {background: silver url(26.jpg) repeat-x}
</style>
<body>
   <h1>Заголовок 1</h1>
   <h2>Заголовок 2 Заголовок  2</h2>
   <h3>Заголовок 3</h3>
   <p>Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст
   Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст
   Текст Текст Текст Текст Текст Текст Текст </p>
</body>
</html>
```



Пример. Использования фона.

```
<html>
<head>
<style type="text/css">
     BODY {
      background:
       white                 /* Цвет фона */
       url(help2.gif)        /* Путь к файлу с рисунком фона */
       right top                    /* Положение в правом верхнем углу */
       no-repeat                    /* Не повторять рисунок */
       fixed                        /* Зафиксировать фон */
     }
</style>
</head>
<body>
    <h1>Помощь</h1><p>Текст Текст Текст Текст Текст Текст Текст Текст
    Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст
    Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст </p>
</body>
</html>
```



4.8 Блочная модель документа. Размеры, поля, отступы, границы

Блочные элементы (блоки текста или **box**) позволяют оперировать с текстом в терминах прямоугольников, которые этот текст занимает. При этом блок текста становится элементом дизайна страницы с теми же свойствами, что и картинка, таблица или прямоугольная область приложения. Блок текста обладает свойствами: высоты (**height**), ширины (**width**), границы (**border**), отступа (**margin**), набивки (**padding**), произвольного размещения (**float**), управления обтеканием (**clear**) (рисунок 4.1).

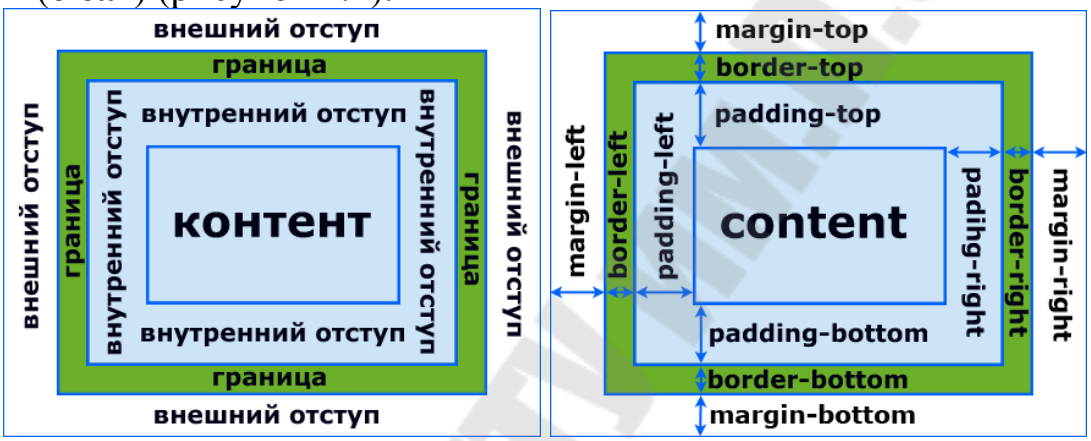


Рисунок 4.1 – Блочная модель документа (**Padding** – отступ, набивка или внутренний отступ, **Margin** – поле или внешний отступ)

Размеры отступов, границ и полей можно задавать по своему усмотрению. Однако по умолчанию они имеют значение 0, т. е. не видны и не занимают место. Параметры **height** и **with** определяют размеры содержимого элемента, отступы задают положение содержимого относительно рамки, внешние поля позволяют позиционно отделить содержимое элемента с рамкой от других элементов документа.

Для полей и отступов можно задать размеры 4-х сторон с помощью следующих параметров:

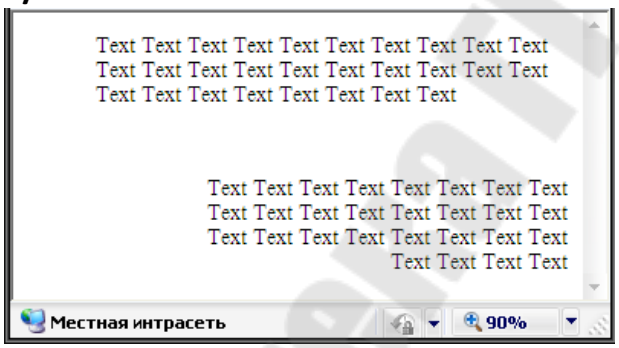
– **margin-top, margin-right, margin-left, margin-bottom** – размеры полей;
– **padding-top, padding-right, padding-left, padding-bottom** – размеры отступов.

Значения этих параметров могут быть заданы числом с указанием единиц измерения или ключевым словом **auto**.

Кроме того, можно использовать параметры **margin** и **padding** с перечислением через пробел размеров всех четырех сторон в следующем порядке: **top**, **right**, **left**, **bottom**. Если указано только одно значение, то оно будет задавать размер всех сторон. Если указаны два или три значения, то остальные будут равны размерам соответствующих противоположных сторон.

Пример. Задание поле и отступов.

```
<html>
<style>
   P.1 {margin-left:50px;margin-right:5px; margin-top:15px;
        margin-bottom:50px; padding:0px;text-align:left; }
   P.2 {padding-left:100px;text-align:right;}
</style>
<body>
   <p class=1>
   Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text
   Text Text Text Text Text Text Text Text Text Text Text Text Text Text </p>
   <p class=2>
   Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text
   Text Text Text Text Text Text Text Text Text Text Text Text Text Text </p>
</body>
</html>
```



Для описания границ блоков применяются следующие атрибуты:

1. размеры границы (рамки): **border-top-width, border-right-width, border-left-width, border-bottom-width**. Значения можно задавать в абсолютных единицах (**%** - нельзя) или ключевые слова: **thin** (тонкая), **medium** (средняя), **thick** (толстая), по умолчанию - **medium**. Также можно использовать параметр **border-width** с перечислением размеров всех четырех сторон.

2. цвет сторон границы: **border-top-color, border-right-color, border-left-color, border-bottom-color**. Параметр **border-color** можно использовать для задания сразу всех или только некоторых сторон;

3. стиль границы: **border-top-style, border-right-style, border-left-style, border-bottom-style**. Может принимать значения: **none, dotted, dashed, solid, double, groove, ridge, inset, outset**. Согласно спецификации **CSS1**, может быть задан для каждой из границ блока. Указание типа линии границы поддерживается не всеми браузерами.

Параметр **border-style** можно использовать для задания стиля сразу всех или только некоторых сторон границы. Если стиль границы не задан в явном виде, другие ее параметры не будут действовать. Задание стиля (типа) границы в явном виде необходимо, чтобы она была отображена (по умолчанию граница не видна).

Для описания границы нет необходимости указывать в стиле все атрибуты. Существует сокращенная запись атрибутов по границам в отдельности:

атрибут: ширина_линии тип_линии цвет_линии

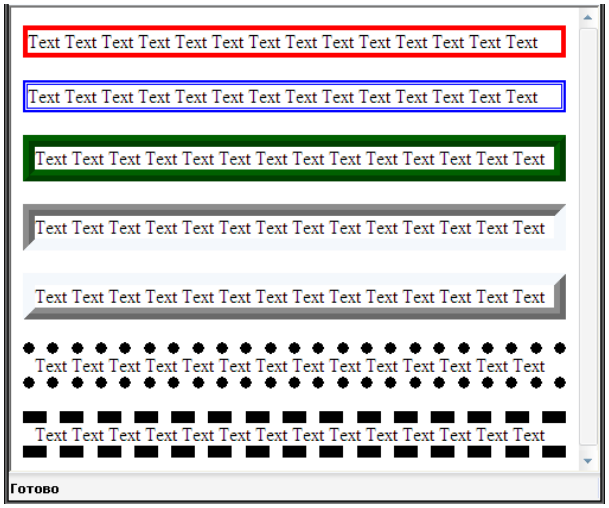
Пример. Задать верхнюю линию границы блока – пунктир, красный цвет.

```
P { border-top:1px dotted red; }
```

Пример. Задание границ

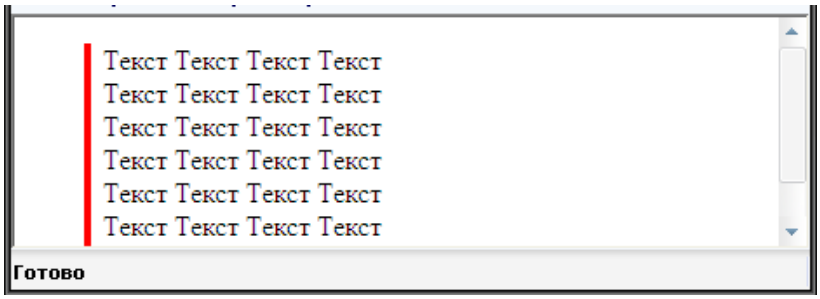
```
<p style="border-style:solid; border-width:4px; border-color:red">
Text Text Text Text Text Text Text Text Text Text Text Text </p>

<p style="border-style:double;  border-color:blue">
Text Text Text Text Text Text Text Text Text Text Text Text </p>

<p style="border-style:ridge; border-width:10px;  border-color:green">
Text Text Text Text Text Text Text Text Text Text Text Text </p>

<p style="border-style:inset; border-width:10px">
Text Text Text Text Text Text Text Text Text Text Text Text </p>

<p style="border-style:outset; border-width:10px">
Text Text Text Text Text Text Text Text Text Text Text Text </p>

<p style="border-style:dotted; border-width:10px">
Text Text Text Text Text Text Text Text Text Text Text Text </p>
```

```
<p style="border-style:dashed; border-width:10px">
Text Text Text Text Text Text Text Text Text Text Text Text </p>
```



Пример. Выделение абзацев

```
<html>
<head>
<style type="text/css">
     #vline {
      border-left: 4px solid red;
      margin-left: 30px;          /* Отступ от края окна до линии */
      margin-right: 50%;          /* Отступ от края окна до линии */
      padding-left: 7px           /* Поле от линии до текста */
     }
</style>
</head>
<body>
    <div id=vline>
    Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст
    Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст
    Текст
    </div>
</body>
</html>
```



4.9 Блочная модель документа. Позиционирование, обтекание, управление видимостью

В CSS существует, по большому счету, четыре способа раскладывать боксы по странице: прямой поток, позиционирование, плавающие блоки, таблицы. Ни один из них не дает полного набора средств, которыми можно было бы сверстать что-нибудь реальное.

Суть позиционирования очень проста: любой бокс можно расположить в любом месте страницы, задав ему точные координаты. Именно любой, а не только <**div**>. Существуют четыре способа позиционирования боксов.

Статическое (**Static**), отсутствие какого бы то ни было специального позиционирования, а просто выкладывание боксов одного за другим сверху вниз – прямой поток. Способ по умолчанию.

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

Фиксированное (**Fixed**). Схоже с **absolute**, но при этом он не скролится вместе с остальной страницей.

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

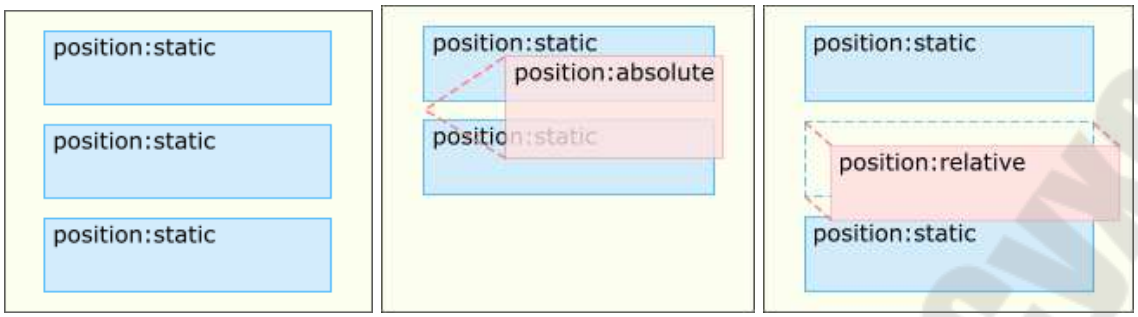


Рисунок 5.2 – Способы позиционирования боксов

Чтобы расположить бокс абсолютно, ему надо задать нужный тип позиционирования и координаты:

```
#somebox {
 position:absolute;
 left:100px; top:100px;
 bottom:100px; right:100px;
}
```

Координаты означают расстояние бокса от краев содержащего блока. Любая из координат необязательна. В случае, если координаты не задают вертикального или горизонтального положения, то оно остается таким же, какое было бы без позиционирования. То есть в случае, когда у нас есть два произвольных бокса один за другим "box1" и "box2", то по вертикали он останется прямо под первым боксов, а по горизонтали будет отстоять от левого края на 150 пикселов:

```
<div id="box1"> первый </div>
<div id="box2"> второй </div>
и второй мы позиционируем так:
#box2 {
 position:absolute;
 left:150px;
}
```

Рассмотрим относительно каких границ двигают бокс координатные свойства.

Страница начинает раскладываться в своеобразный перевернутый "стакан", начинающийся от верха окна, ограниченный с боков и бесконечно продолжающийся вниз. Если все блоки статические, то они так в этот стакан и раскладываются один за другим. Если в этом потоке появляется позиционированный бокс, то его координаты вычисляются от сторон этого самого стакана.

Позиционированный бокс внутри себя, также создает такой же стакан, и все боксы, находящиеся у него внутри позиционируются уже относительно него, а не относительно окна.

Любой позиционированный (не **static**) бокс создает внутри себя такой стакан в терминологии CSS называемый – **содержащий блок** (**containing block**).

Бокс с **position:fixed** – это, разновидность того же абсолютного позиционирования с разницей в том, что при скролинге окна эти боксы остаются на месте. Этот эффект широко используется на страницах веб-приложений для всяческих прилипающих блоков меню и тулбаров. Особенность – если фиксированный бокс не влезет в окно, то доскролиться до него будет уже нельзя.

У абсолютного позиционирования есть очень полезное свойство: им можно задавать размеры боксы по их внешним границам. При абсолютном позиционировании проблема добавления отступов, рамки и границ, решается указанием координат противоположных сторон одновременно:

```
#somebox {
 position:absolute;
 top:0; left:0; right:0;
 margin:20px; padding:20px;
}
```

Этот бокс с заданными левой и правой координатами будет точно касаться боковых сторон своего **содержащего блока**, какой бы ширины тот ни был, а margin'ы и padding'и будут откладываться **внутрь** бокса.

Это свойство неоценимо при создании раскладок веб-приложений, где неперекрывающиеся боксы должны занимать весь экран по определенной сетке.

Рисунок 4.3 – Пример абсолютного позиционирования боксов

Главная проблема абсолютного позиционирования состоит в том, что координаты и размеры бокса можно задать только относительно содержащего блока, в котором он лежит. Что часто неудобно.

Пример - абсолютным позиционированием нельзя сделать самую традиционную раскладку: заголовок, содержимое любой высоты в несколько колонок и нижний блок. Обычно получается такое – рисунок 4.4

Рисунок 4.4 – Пример абсолютного позиционирования боксов

Видны две проблемы:

1. Колонки не получается выровнять по высоте, потому что колонки друг в друге не лежат, и в CSS нет средств сказать "высота как вот у другого бокса.
2. Нижний блок проваливается за колонки, потому что они изымаются из потока, и нижний блок по высоте не толкают. И его нельзя абсолютно позиционировать под самой высокой колонкой, потому что в CSS нет средств сказать "верх под тем другим боксом".

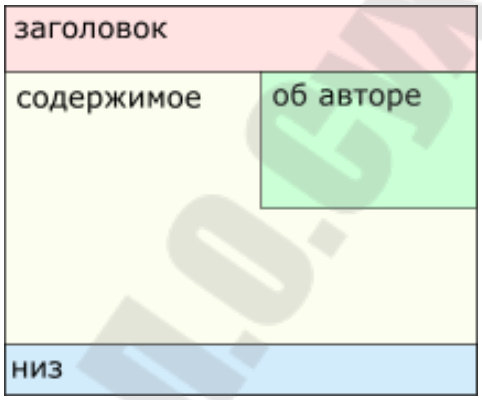
Относительное позиционирование похоже на абсолютное, но к нему зачем-то добавляется эффект: бокс продолжает занимать место в потоке.

Чаще же всего **position:relative** используют вообще без задания смещений. В этом случае он ведет себя как обычный статический бокс, но поскольку он все таки не статический, то он создает **внутри** себя **содержащий блок**, тот самый, относительно которого будут позиционироваться боксы **внутри него**.

Пример. Пусть у нас есть три блока: "заголовок", "содержимое" и "низ", а внутри "содержимого" лежит блок "об авторе". Высота заголовка нам точно не известна. Боксы статические, идут один за дру-

гим, и какая бы высота у заголовка ни была, содержимое будет начинаться прямо под ним. Блок об авторе расположить так, чтобы он был точно в правом верхнем углу содержимого.

```
<div id="header">Заголовок</div>

<div id="contents"> Содержимое
  <div id="author"> Автор </div>
</div>

<div id="footer">Низ</div>
```



1. вариант

```
#author {
 position:absolute;
 top:0; right:0;
}
```

Содержащим блоком для #**author** сейчас является все окно, и блок об авторе уедет поверх заголовка. Точно поставить ему расстояние от верха тоже нельзя, потому что размер заголовка у нас может быть разный.

2. вариант

Сделать блок содержимого тоже абсолютным, тогда он станет содержащим блоком. Но тогда он сам выдернется из потока и низ прижмется прямо к заголовку.

3. вариант

Сделать блок содержимого с относительным позиционированием (**position:relative).** В этом случаи он никуда не денется из потока, но в то же время станет содержащим блоком, и "об авторе" расположится в его правом верхнем углу.

Обтеканием блока текста другим текстом управляют два атрибута CSS: **float** и **clear**. Атрибут **float** определяет блок как "плавающий" и может принимать значения:

– **left** – блок прижат к левой границе охватывающего элемента;

– **right** – блок прижат к правой границе охватывающего элемента;

– **both** – текст может обтекать блок с обеих сторон.

Как и позиционирование, плавающие блоки используется для того, чтобы двигать боксы. Но в отличие от позиционирования, которым можно двигать боксы практически произвольно, все, что может float – это сдвинуть элемент к одной из сторон потока, правой или ле-

вой. При этом сам бокс и следующие за ним в потоке приобретают интересное поведение:

1. Плавающий бокс смещается по горизонтали и прилипает к одной из сторон родителя;
2. Плавающий бокс перестает раздаваться на всю ширину родительского бокса-контейнера (как это происходит с блоками в потоке). С его не прижатой к родителю свободной стороны появляется свободное место;
3. Следующие за ним блочные боксы подтягиваются вверх и занимают его место, как если бы плавающего бокса в потоке не было;
4. Строчные же боксы внутри подвинувшихся наверх блоков начинают обтекать плавающий бокс со свободной стороны.

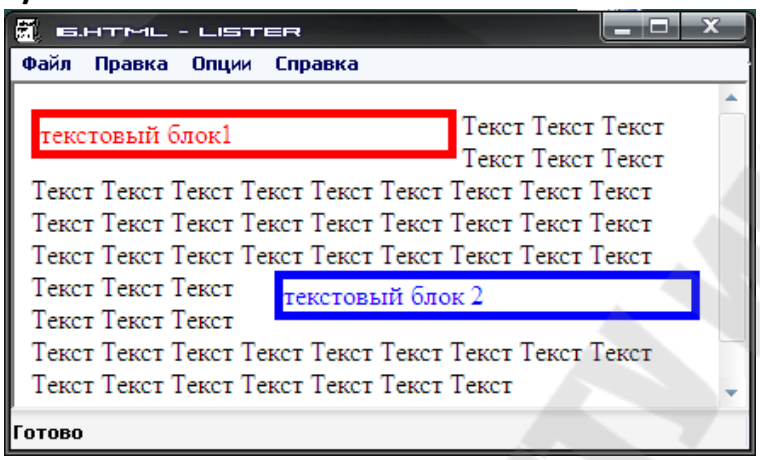
Сама коробка блока, следующего за плавающим блоком, подлезает под него и занимают всю ширину потока, а вот текст внутри этого блока смещается в сторону и обтекает плавающий.

Сдвинутые в одну сторону плавающие блоки будут пытаться уместиться сбоку от предыдущего, с его свободной стороны. И только если ему там не будет достаточно места, тогда сместится ниже и будет пытаться уместиться уже там.

Пример. Задать два плавающих блока прижатых к разным границам охватывающего элемента.

```
<html>
<head>
<style type="text/css">
 .col1 {
  border-style:solid;
  border-width: 5px;
  width: 250px;
  color: red;
  float:left
}
 .col2 {
  border-style:solid;
  border-width: 5px;
  width: 250px;
  color: blue;
  float: right
}
</style>
```

```
</head>
<body>
   <p><div class=col1>  <p>текстовый блок1</p> </div>
   Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст
   Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст   Текст Текст
   Текст Текст Текст Текст Текст Текст Текст <div class=col2>  <p>текстовый
   блок 2</p> </div> Текст Текст Текст Текст Текст Текст Текст Текст Текст
   Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст Текст
   Текст</p>
</body>
</html>
```

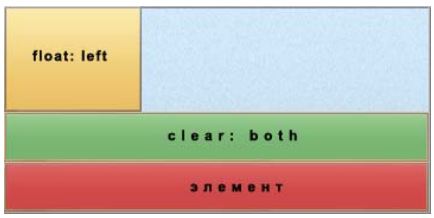


Второй атрибут описания стилей **clear** устанавливает, с какой стороны элемента запрещено его обтекание другими элементами. Если задано обтекание элемента с помощью свойства **float**, то **clear** отменяет его действие для указанных сторон. Допустимые значения:

- **none** — обтекание разрешено с обоих сторон
- **left** — обтекание запрещено слева
- **right** — обтекание запрещено справа
- **both** — обтекание запрещено с обоих сторон
- **inherit** — наследует значение от родителя

Пример. Запрет обтекания вокруг элемента другими элементами

```
.block {clear: both; }
```



Случается, что содержимое элемента выходит за его границы. Например, графическое изображение или текст могут не поместиться полностью в содержащем его контейнере (<**div**>, <**span**> и др). Свойство **overflow** управляет отображением содержания блочного элемента, если оно целиком не помещается и выходит за область заданных размеров. Может принимать следующие значения:

– **visible** – если содержимое и выйдет за пределы отведенного ему места, он все равно будет показан полностью; это значение принято по умолчанию;
– **hidden** – выступающие за границы элемента-контейнера части содержимого будут обрезаны;
– **auto** – добавляет полосы прокрутки, если содержимое выходит за границы элемента-контейнера;
– **scroll** – добавляет полосы прокрутки в любом случае.

Пример. Управление отображением содержания блочного элемента

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>overflow</title>
  <style>
   .layer {
    overflow: scroll; /* Добавляем полосы прокрутки */
    width: 300px; /* Ширина блока */
    height: 150px; /* Высота блока */
    padding: 5px; /* Поля вокруг текста */
    border: solid 1px black; /* Параметры рамки */
   }
  </style>
 </head>
 <body>
    <div class="layer">
    <h2>Duis te feugifacilisi</h2>
    <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diem
    nonummy nibh euismod tincidunt ut lacreet dolore magna aliguam erat
    volutpat. Ut wisis enim ad minim veniam, quis nostrud exerci tution
    ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.</p>
```

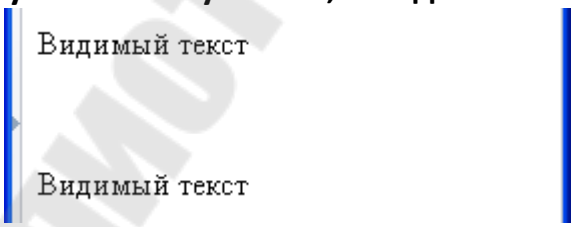
```
    </div>
 </body>
</html>
```

Элементы, по умолчанию видимые в окне браузера, можно сделать невидимыми с помощью параметров **visibility** и **display**. Обычно они используются в сценариях для динамического управления видимостью элементов. Иногда в документе требуется использовать элемент, но не показывать его. Например, можно загрузить в браузер таблицу с данными, которые будут как-то использоваться сценарием, но пользователь не должен их видеть. Параметр **visibility** принимает следующие значения:

– **visible** – элемент виден;
– **hidden** – элемент не виден (скрыт), однако занимает на странице место, т. е. элемент прозрачен;
– **inherit** – элемент виден, если его родительский элемент отображается.

Пример. Управление видимостью элементов.

```
<p style="visibility:visible;">Видимый текст</p>
<p style="visibility:hidden;">Невидимый текст</p>
<p style="visibility:visible;">Видимый текст</p>
```




```
<p style="visibility:hidden"> НЕвидимый текст
<b style="visibility:inherit"> НЕвидимый текст</p>
```

```
<p style="visibility:hidden">НЕвидимый текст
<b style="visibility:visible ">Видимый текст</p>
```

Параметр **display** – многоцелевое свойство, которое определяет, как элемент должен быть показан в документе. Параметр **display**:**none** делает элемент не только невидимым, но и не занимающим место.

5. HTML5

5.1 Описание языка HTML5

HTML5 – это пятая версия основного языка разметки веб-страниц, разработка которой началась еще в 2007 году. В настоящее время **спецификация HTML5** находится в стадии разработки, закончить которую планируют в 2014 году. Язык HTML5 содержит много новых свойств, что делает HTML значительно более мощным и удобным для создания приложений Web. Основные свойства языка HTML5:

- **Новые семантические элементы.** HTML5 содержит новые семантические элементы для разметки страницы, такие как <**nav**>, <**header**>,<**section**>, <**footer**> и <**article**>.**;**
- **Новые свойства форм.** HTML5 предоставляет стандартизованный, простой способ реализации таких свойств, как выбор даты, ползунки и клиентская проверка**;**
- **Собственная поддержка видео и аудио.** HTML5 содержит элементы <**video**> и <**audio**> для простой реализации собственных видео и аудио плееров с помощью только открытых стандартов, и также содержит API, позволяющий легко реализовать индивидуальные элементы управления плеером**;**
- **API рисования на холсте**: Элемент <**canvas**> и соответствующий API позволяют определить на странице область для рисования, и использовать команды JavaScript для рисования линий, фигур и текста, импорта и манипуляций с изображениями и видео, экспорта в различные форматы изображений, и многих других вещей.
- **Сокеты Web**. API позволяет открывать постоянное соединение между сервером и клиентом на определенном порте, и посылать данные в обоих направлениях, пока порт не будет закрыт. Это существенно улучшает эффективность приложений web, так как данные могут непрерывно и аккуратно передаваться между клиентом и сервером без постоянной перезагрузки страницы, и без постоянного опроса сервера, чтобы проверить, нет ли доступных обновлений.
- **Хранилище Web.** Web хранилище HTML5 позволяет хранить значительно больше данных, и делать с ними значительно больше.

– **Web workers**. Общая проблема, встающая перед приложениями web, состоит в том, что их производительность страдает, когда требуется обработать много данных, в связи с тем, что все происходит в одной нити/процессе (только одна последовательность обработки может выполняться в текущий момент). Web Workers смягчают эту проблему, позволяя создавать фоновые процессы для выполнения значительного объема вычислений, позволяя основному процессу продолжить выполнение других задач.
– **Геолокация**. Спецификация геолокации определяет API, который позволяет приложению web получить доступ к данным в любом местоположении, которое стало доступным, например, с помощью средств GPS устройства. Это позволяет добавлять в приложения различные полезные свойства, связанные с местоположением, например, выделить контент, который больше подходит для местоположения.

Все современные браузеры уже поддерживают основные элементы HTML5. Список новых и неподдерживаемых тегов языка представлен в таблице 5.1.

Таблица 5.1 – Список новых и неподдерживаемых тегов HTML5

Тег	Описание
<applet>	**Не поддерживается.** Определяет апплет
<article>	Определяет статью
<aside>	Определяет контент в стороне от основного контента страницы
<audio>	Определяет аудио контент
<basefont>	**Не поддерживается.** Используется вместо CSS для задания шрифта
<canvas>	Определяет графики
	Не поддерживается. Определяет текст по центру
<command>	Определяет командную кнопку
<datagrid>	Определяет данные в упорядоченный список
<datalist>	Определяет выпадающий список
<datatemplate>	Определяет шаблон данных
<details>	Определяет детали элемента
<dialog>	Определяет диалог (разговор)
<dir>	**Не поддерживается.** Определяет список директорий
<eventsource>	Определяет цель события, отправляемого по сер-

	веру
<figure>	Определяет группу медиа-контента, и их подписи
<footer>	Определяет нижний колонтитул для раздела или страницы
<header>	Определяет область заголовка раздела или страницы
<isindex>	**Не поддерживается.** Определяет поисковый индекс в документе
<mark>	Определяет выделенный текст
<meter>	Определяет измерения в течение заранее определенного диапазона
<nav>	Определяет навигационные ссылки
<nest>	Определяет вложенную точку в шаблоне данных
<noframes>	**Не поддерживается.** Определяет секцию, не поддерживающую фрейм
<progress>	Определяет ход выполнения задачи любого рода
<rule>	Определяет правила для обновления шаблонов
<s>	**Не поддерживается**. Определяет зачеркнутый текст
<strike>	**Не поддерживается.** Определяет зачеркнутый текст
<time>	Определяет дату/время
<video>	Определяет видео
<xmp>	**Не поддерживается**. Определяет выровненный текст

Из таблицы видно, что новых тегов достаточно много, некоторые из них уже ориентированы на динамический контент, что дает большое преимущество HTML 5 перед HTML 4.

5.2 Структурные элементы HTML5

HTML4 имеет множество семантических элементов, позволяющих четко определять различные свойства страницы Web, такие как формы, списки, параграфы, таблицы и т.д. Однако он имеет и свои недостатки. Существенно используются элементы <**div**> и <**span**> с различными атрибутами **id** и **class** для определения различных других свойств, таких как навигационные меню, верхние и нижние колонтитулы, основной контент, окна предупреждения, боковые панели, и т.д.

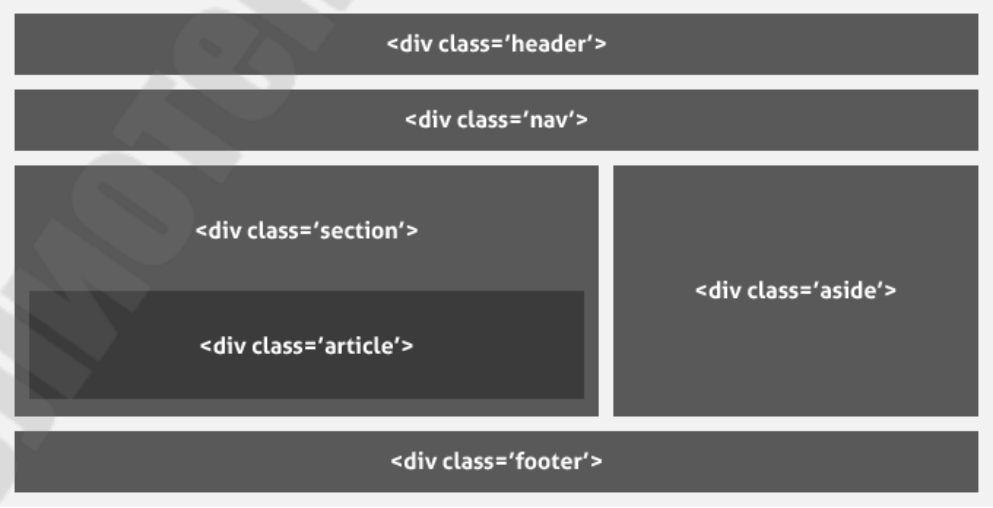
В HTML5 для задания структуры документа используются следующие элементы:

– <**header**>: Используется для верхнего колонтитула сайта.
– <**footer**>: Используется для нижнего колонтитула сайта.
– <**nav**>: Содержит навигационные функции страницы.
– <**article**>: Содержит автономный фрагмент контента, который будет иметь смысл, если используется как позиция RSS, например, новостное сообщение.
– <**section**>: Используется либо для объединения в группу различных статей с различной целью или по различным темам, или для определения различных разделов одной статьи.
– <**time**>: Используется для разметки времени и даты.
– <**aside**>: Определяет блок контента, который связан с основным контентом, но не входит в его основной поток.
– <**hgroup**>: Используется в качестве оболочки скрытия более одного заголовка, если требуется, чтобы учитывался только один заголовок в структуре заголовков страницы.
– <**figure**> и <**figcaption**>: Используется для инкапсуляции рисунка как единого элемента, и содержит, соответственно, подпись для рисунка.

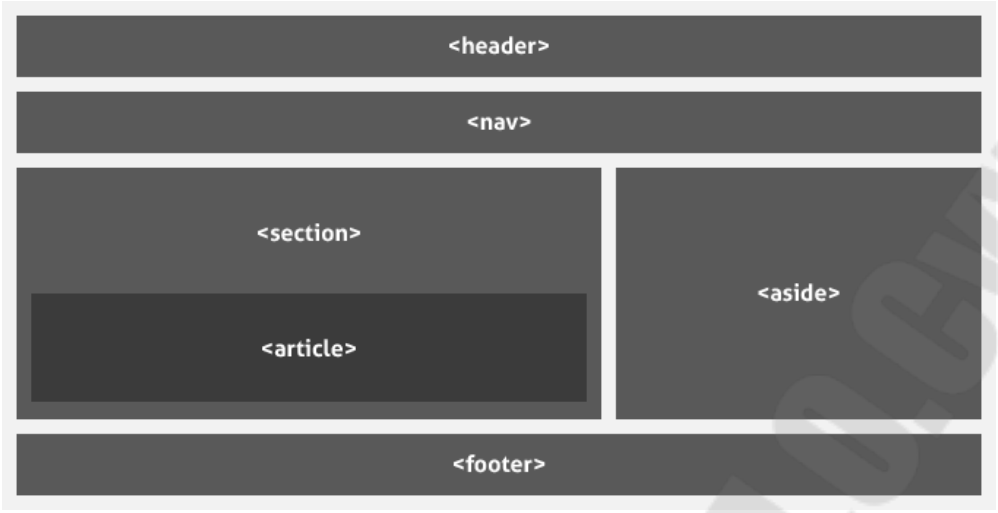
В HTML5 **doctype** стал значительно проще, чем в более старых версиях HTML:

<!DOCTYPE html>

Различия между структурами типичного сайта в HTML4 и HTML5 показаны на рисунке 5.1. HTML5 имеет семантический чистый код, структура более легкая, гибкая и функциональная.



А)



Б)

Рисунок 5.1 – Структура разметки (а – HTML4, б – HTML5)

Назначение элемента <**header**> состоит в создании оболочки вокруг раздела контента, который формирует верхний колонтитул страницы, содержащий обычно логотип компании/графику, заголовок основной страницы, и т.д.

Пример: Задание верхнего колонтитула документа.

```
<header>
 <hgroup>
   <h1>HTML5</h1>
   <h2>Пример верстки сайта </h1>
 </hgroup>
</header>
```

Тег <**hgroup**> позволяет учитывать группу заголовков как один заголовок в структуре документа.

Для размещения контента нижнего колонтитула сайта необходимо использовать тег <**footer**>. Этот контейнер обычно содержит различную информацию, от уведомления об авторских правах и контактной информации, до заявления о доступности, информации о лицензировании и множества других второстепенных ссылок.

Пример: Задание нижнего колонтитула документа.

```
<footer>
 <h3 id="copyright">Copyright and attribution</h3>
</footer>
```

Не существует ограничения на количество колонтитулов. Страница может содержать несколько статей, и иметь для каждой статьи верхний и нижний колонтитул.

Элемент <**nav**> предназначен для разметки навигационных панелей или других конструкций (например, формы поиска), которые направляют вас на различные страницы текущего сайта, или различные области текущей страницы. Другие ссылки, такие как рекламные ссылки, не учитываются. Контейнер <**nav**> может содержать заголовки и другие структурные элементы.

Пример: Задание навигационных панелей.

```
<nav>
 <h2>Contents</h2>
  <ul>
   <li><a href="#Intro">Introduction</a></li>
   <li><a href="#History">History</a>

   <!–другие навигационные ссылки ... -->

  </ul>
</nav>
```

Элемент <**aside**> предназначен для разметки фрагментов контента, которые имеют отношение к основному контенту, но не вписываются явно в основной поток изложения. Другими подходящими кандидатами для элементов <**aside**> являются списки ссылок на внешний связанный контент, справочная информация, цитаты, и боковые панели.

Пример: Разметка боковой панели.

```
<aside>
  <h2><a href="/service/">Услуги DivMotive</a></h2>
  <ul><li>
  <h3><a href="/service/website/">Создание и разработка
  сайтов</a></h3>
  <em>Полностью готовый для работы в сети сайт на базе <abbr
  title="Content Management System">CMS</abbr></em>
  </li>
  <li>
  <h3><a href="/service/design/">Веб-дизайн</a></h3>
```

```
   <em>Оригинальные индивидуальные решения оформления
   сайта</em>
   </li>
   <li>
   h3><a href="/service/markup/">HTML верстка сайтов</a></h3>
   <em>Блочная валидная семантическая <abbr title="HyperText Markup
   Language">HTML</abbr>/<abbr title="Extensible Hypertext Markup
   Language">XHTML</abbr> верстка сайта в строгой технической
   спецификации</em>
   </li>
   </ul>
</aside>
```

Динамический дуэт из <**figure**> и <**figcaption**> был создан для решения проблем с размещением связанных рисунков и подписей к ним. Элемент <**figure**> применяется для, объединения всего контента, из которого составляется один рисунок, будет ли это текст, изображения, SVG, видео, или что-то другое. Элемент <**figcaption**> помещается внутри тега <**figure**>, и содержит описательный заголовок для этого рисунка.

Пример. Простой рисунок, с подписью.

```
<figure>
 <img src="pwei.png" alt="Old poppies logo" />
 <figcaption>
  The old poppies logo, circa 1987.<br />
  <a href="http://www.flickr.com/photos/bobcatrock/317261648/">
   Original picture on Flickr</a>, taken by bobcatrock.
 </figcaption>
</figure>
```

Элемент <**time**> позволяет определить точно выраженное значение даты и времени, которое одновременно понятно человеку и машине. Текст между открывающим и закрывающим тегами может быть любым, подходящим для людей. Полный формат:

ГГГГ-ММ-ДДТчч:мм:сс±чч:мм

- **Год** – задается четырьмя цифрами (1860).
- **Месяц** – две цифры (01 – январь … 12 – декабрь).
- **День** – две цифры от 01 до 31.
- **Час** – две цифры от 00 до 23.

- **Минуты** – две цифры от 00 до 59.
- **Секунды** – две цифры от 00 до 59.
- **Часовой** пояс – часы и минуты с указанием знака плюс или минус.

Пример. Даты выпуска чего либо.

```
<time datetime="1989-03-13">1989</time>
<time datetime="1989-03-13">13th March 1989</time>
<time datetime="1989-03-13">March 13 1989</time>
<time datetime="1989-03-13">My nineteenth birthday</time>
```

Дата внутри атрибута **datetime** представлена в стандарте ISO и является машинно-читаемой датой, поэтому мы получаем двойную выгоду. Также можно добавить время и настройку часового пояса.

Пример. Задание да ты и времени.

```
<time datetime="1989-03-13T13:00">One o'clock in the afternoon, on the 13th of March 1989</time>
<time datetime="1989-03-13T13:00Z-08:00">One o'clock in the afternoon, on the 13th of March 1989</time>
```

Элемент <**article**> предназначен для независимых фрагментов контента, которые будут иметь смысл вне контекста текущей страницы, и могут хорошо объединяться. Такие фрагменты контента включают публикации в блоге, видео и его текстовая запись, новостная история, или одна часть серийной истории. Элемент <**section**>, предназначен для разбиения контента страницы на различные функциональные или тематические области, или разбиения статьи или истории на различные части.

Пример. Описание структуры музыкальной группы.

```
<article>
 <section id="Intro"> <h2>Introduction</h2> </section>
 <section id="History"> <h2>History</h2> </section>
 <section id="Discography"> <h2>Discography</h2> </section>
</article>
```

Пример. Описание структуры

```
<section id="rock">
 <h2>Rock bands</h2>
 <!–Здесь размещается содержательная часть -->
```

```
</section>
<section id="jazz">
  <h2>Jazz bands</h2>
  <!–Здесь размещается содержательная часть -->
</section>
<section id="hip-hop">
  <h2>Hip hop bands</h2>
  <!–Здесь размещается содержательная часть -->
</section>
```

На ряду с новыми элементами задания структуры, старый <**div**> по-прежнему имеет совершенно законное применение. Его следует использовать, когда не существует другого более подходящего доступного элемента для объединения области контента, что часто происходит, когда вы используете элемент только для объединения контента в группу с целью стилевого или визуального оформления. Например, для создания контейнера вокруг всего контента с целью его выравнивания по центру в браузере.

Для проверки документов HTML5 можно использовать валидатор W3C (***http://validator.w3.org/***), который может проверять HTML5, а также большой спектр других разновидностей языков разметки.

5.3 HTML5 и старые браузеры

Старые браузеры естественно не поддерживают HTML5.

Для решения проблемы поддержки необходимо выполнить следующие:

1. Браузеры с поддержкой HTML5 интерпретируют новые теги как блочные элементы. Для того чтобы заставить вести себя правильно в старых браузерах необходимо использовать CSS описания:

```
article, section, aside, hgroup, nav, header, footer, figure, figcaption
{display: block;}
```

2. Internet Explorer не добавляет стиль к элементам, которые не понимает, поэтому предыдущее описание не будет работать в **IE7**, **IE8** (**IE9** поддерживает HTML5). Это можно исправить, если создать фиктивный элемент с помощью **JavaScript**. Для этого включим в <**head**> код создания элементов HTML5. Сам

скрипт заключается в условные комментарии, чтобы выполнялся только для IE версии 8.0 и ниже.

```
<!--[if lt IE 9]>
    <script>
      document.createElement('article');
      document.createElement('section');
      document.createElement('aside');
      document.createElement('hgroup');
      document.createElement('nav');
      document.createElement('header');
      document.createElement('footer');
      document.createElement('figure');
      document.createElement('figcaption');
    </script>
<!--[if lt IE 9]>
```

Также можно воспользоваться общедоступным скриптом написанным Реми Шарпом и распространяемым по лицензии MIT. Для этого достаточно указать на него ссылку:

```
<!--[if lt IE 9]>
    <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js">
    </script>
<![endif]-->
```

5.4 Пример сайта на HTML5

Принцип верстки на HTML5 ничем не отличается от верстки на HTML4. Начнем с HTML разметки. Планируемый макет сайта представлен на рисунке 5.2. Макет содержит: **header**, навигационное меню – **nav**, контент или **section**, внутри снова **section** и **article**, правая часть – **aside**, а снизу **footer**.

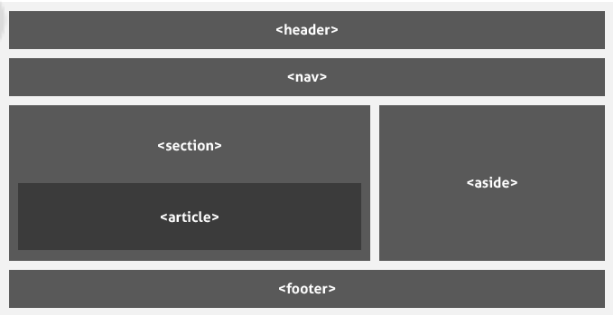


Рисунок 5.2 – Макет сайта

HTML код сайта на HTML5.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Пример верстки сайта на HTML5 от DivMotive.ru</title>
  <!--[if IE]>
  <script>
      document.createElement('header');
      document.createElement('nav');
      document.createElement('section');
      document.createElement('article');
      document.createElement('aside');
      document.createElement('footer');
  </script>
  <![endif]-->
  <style type="text/css">
      /* Здесь будут наши стили */
  </style>
</head>
<body>
<div id="main">
  <header class="h">
      <h1>Пример верстки сайта на HTML5 от DivMotive.ru</h1>
      <i class="tag">&lt;header&gt;</i>
  </header>
  <nav>
      <ul>
          <li><a href="/about/">Об авторе</a></li>
          <li><a href="/service/">Услуги</a></li>
          <li><a href="/clients/">Клиентам</a></li>
          <li><a href="/project/">Проекты</a></li>
          <li><a href="/article/">Статьи</a></li>
          <li><a href="/news/">Новости</a></li>
          <li><a href="/contact/">Контакты</a></li>
      </ul>
      <i class="tag">&lt;nav&gt;</i>
  </nav>
  <section>
      <img class="image" src="img.jpg" alt="" />
      <i class="tag">&lt;section&gt;</i>
  </section>
  <section>
      <section>
          <article>
              <header>
```

```
            <h2>Создание и разработка сайтов</h2>
            <i class="tag">&lt;header&gt;</i>
          </header>
          <p>Компания «<a href="/">DivMotive</a>» предлагает комплекс высококачественных услуг по созданию, разработке и поддержке Интернет-ресурсов в сети на базе <a href="/cms/">Систем управления сайтом</a> (<abbr title="Content Management System">CMS</abbr>).</p>
          <p>К каждому проекту мы подходим индивидуально и выбираем оптимальный вариант для решения поставленных задач. Разработанный нами сайт будет отличаться оригинальным оформлением и функциональностью. При разработке веб-сайтов мы используем самые передовые технологии.</p>
          <i class="tag">&lt;article&gt;</i>
        </article>
        <article>
          <header>
            <h2>Веб-дизайн</h2>
            <i class="tag">&lt;header&gt;</i>
          </header>
          <p>Каждый сайт должен быть по-своему привлекателен. А первое, на что посетитель обращает внимание – это <strong>дизайн</strong>, т.е. все визуальное представление сайта, включая <em>пиктограммы</em> и <em>иконки</em>, <em>шрифты</em>, <em>иллюстрации</em>, и т.д.</p>
          <p>От того, насколько дружественно и качественно <strong>разработан дизайн</strong>, зависит дальнейшее пребывание пользователя на Вашем сайте.</p>
          <i class="tag">&lt;article&gt;</i>
        </article>
        <article>
          <header>
            <h2>HTML верстка сайтов</h2>
            <i class="tag">&lt;header&gt;</i>
          </header>
          <p><em>Верстка макета (HTML верстка)</em> – это процесс формирования веб-страницы, по средствам кода соответствующего языка разметки (<abbr title="HyperText Markup Language">HTML</abbr>, <abbr title="eXtensible Markup Language">XML</abbr> и т.п.), так же состоящей из стилей оформления (<abbr title="Cascading Style Sheets">CSS</abbr>) и подгружаемых картинок и фонов, на которые специальным образом разбивается макет сайта, в соответствии с дизайном, как правило, в формате <abbr title="Photoshop Document">PSD</abbr>.</p>
          <i class="tag">&lt;article&gt;</i>
        </article>
        <i class="tag">&lt;section&gt;</i>
      </section>
      <aside>
        <h2><a href="/service/">Услуги DivMotive</a></h2>
        <ul>
```

```
            <li>
              <h3><a href="/service/website/">Создание и разработка сайтов</a></h3>
              <em>Полностью готовый для работы в сети сайт на базе <abbr
title="Content Management System">CMS</abbr></em>
            </li>
            <li>
              <h3><a href="/service/design/">Веб-дизайн</a></h3>
              <em>Оригинальные индивидуальные решения оформления сайта</em>
            </li>
            <li>
              <h3><a href="/service/markup/">HTML верстка сайтов</a></h3>
              <em>Блочная валидная семантическая <abbr title="HyperText Markup
Language">HTML</abbr>/<abbr title="Extensible Hypertext Markup
Language">XHTML</abbr> верстка сайта в строгой технической спецификации</em>
            </li>
          </ul>
          <br />
          <h2>Наши преимущества</h2>
          <ol>
            <li>самый <a href="/cms/">широкий выбор CMS</a> для реализации любой
идеи;</li>
            <li>создание уникального функционала для сайта;</li>
            <li>подбор наиболее оптимальной <abbr title="Content Management
System">CMS</abbr> для поставленной задачи;</li>
            <li><a href="/service/website/">качественная разработка сайта</a> на
каждом этапе его реализации;</li>
            <li>применение новых технологий в разработке сайта: jQuery, HTML5,
CSS3;</li>
            <li>бесплатные консультации и сопровождение проекта в течение
месяца;</li>
            <li>скидки постоянным клиентам и индивидуальные бонусные
программы.</li>
          </ol>
          <i class="tag">&lt;aside&gt;</i>
      </aside>
      <i class="tag">&lt;section&gt;</i>
  </section>
  <footer>
      Copyright &copy; 2011 <a href="/" rel="copyright">DivMotive</a>. All rights
reserved.
      <i class="tag">&lt;footer&gt;</i>
  </footer>
</div>
</body>
</html>
```

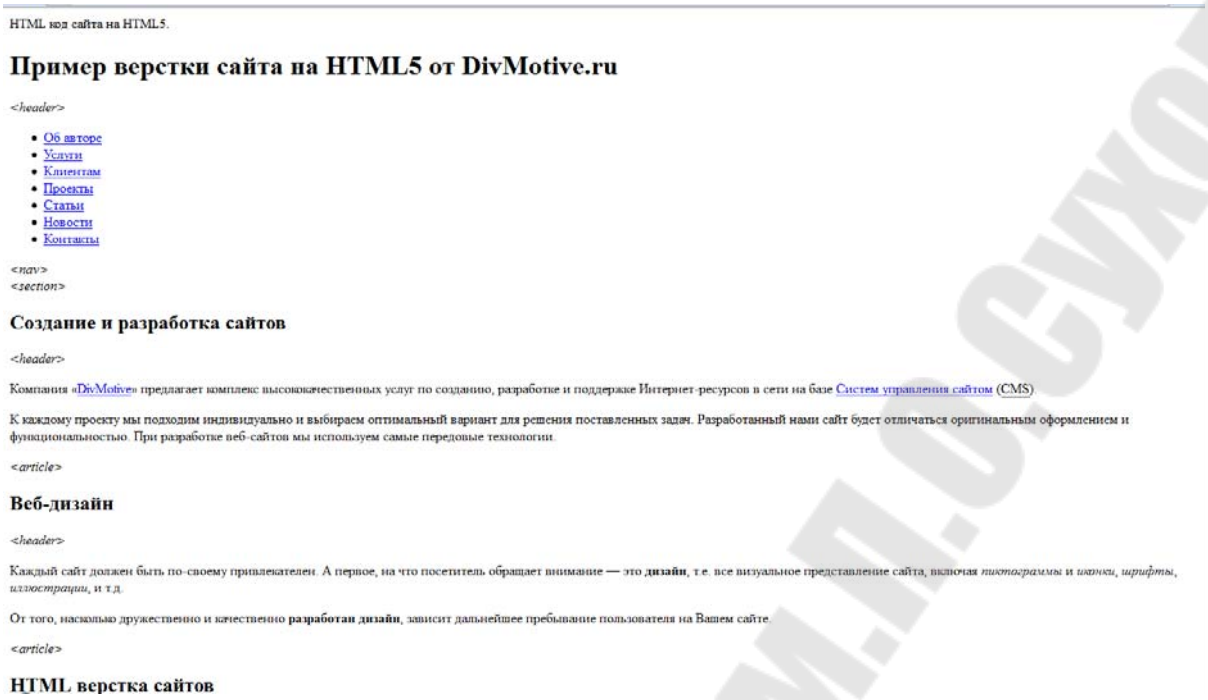


Рисунок 5.3 – Макет сайта без CSS кода

Зададим стилевое оформление – **CSS** код.

```
* {margin:0;padding:0;outline:none} /* обнуляем все элементы */
/* общие стили */
html {font-size:75%;height:100%}
body {font:normal 1em/1.3 arial, helvetica, sans-
serif;color:#333;height:100%;background:#f0f0f0}
a {color:#2b82dc}
a:hover {text-decoration:none}
abbr {border-bottom:1px dotted #2b82dc;cursor:help}
h1, h2, h3, h4, h5, h6 {font-weight:normal;line-height:1;margin:4px 0 12px;color:#2ca9e4}
h1 {font-size:1.88em}
p {text-align:justify;word-spacing:.1ex;line-height:1.6;margin-bottom:1em}
ul {list-style:none}
ol {list-style-position:inside}
header, nav, section, article, aside, footer {display:block;position:relative}
/* выравниваем страницу по центру, делаем отступы и цвет фона */
#main {width:950px;min-height:100%;padding:0 24px;margin:0 auto;background:#fff}
/* пишем класс для шапки - header */
.h {overflow:hidden;height:70px;padding:35px 25px 0}
/* меню навигации */
nav {width:100%;height:32px;background:#2ca9e4}
nav li {float:left;margin:8px 20px 0}
nav a {font-weight:bold;text-decoration:none;color:#fff}
nav a:hover {text-decoration:underline;color:#fff}
/* область основного контента */
section {overflow:hidden}
```

```
/* стиль для картинки */
.image {width:100%;margin:10px 0}
/* внутренняя область контента */
section section {float:left;width:640px}
/* правая часть */
aside {overflow:hidden;width:250px;padding:0 30px}
/* нижняя часть - footer */
footer {height:40px;padding:25px 25px 0;margin:20px 0 0;border-top:1px solid #bbb}
/* стили для выделения областей HTML */
.tag {position:absolute;right:6px;top:2px;font-size:11px;font-style:normal}
header:hover, nav:hover, section:hover, article:hover, aside:hover, footer:hover
{background:#777;color:#fff}
section section:hover, section aside:hover {background:#999}
section article:hover {background:#bbb}
article header:hover {background:#eee;color:#777}
section article .tag {top:12px}
article header .tag {right:70px;top:2px}
aside .tag {top:12px}
```

Рисунок 5.4 – Макет сайта с стилевым оформлением

6. Этапу разработки сайта

Обычно рабочий процесс над проектом представляется следующим образом:

1. Изучение тематики сайта (бриф);
2. Проектирование, разработка концепции (брейншторм);
3. Разработка дизайна;
4. Создание макета сайта;
5. Программирование сайта;
6. Запуск сайта, поисковая оптимизация;
7. Поддержка сайта.

При разработке сайта необходимо как можно больше узнать о целях и задачах, сроках, целевой аудитории, предполагаемом бюджете сайта. По окончании данного этапа должно сложиться четкое понимание того что от вас требуется.

После того определения с целями и задачами, можно переходить к разработке базового макета. Этот этап можно начать с посещения подобных по тематике сайтов. Первая задача – изучить лучшие образцы сайтов по аналогичной или близкой теме и отметить для себя достоинства и недостатки каждого из них. Учитывая опыт конкурентов, можно переходить к брейншторму, в ходе которого нужно определить общую концепцию дизайна, базовые элементы, цвета, шрифты и прочее. Следующая стадия проектирование макета – создания наброска, каркаса. Его можно выполнить на бумаге, можно с помощью специальных программ и сервисов.

Следующим этапом разрабатывается дизайн для созданного макета. Разработка дизайна – это этап определения с формой содержимого сайта, детализация макета. Обычно дизайн создается в Photoshop или другом графическом редакторе. Готовый дизайн должен практически полностью соответствовать конечному продукту (элементы, цвета, картинки, шрифты). Разработка дизайна включает не только основную страницу, но и подстраницы, а возможно и дополнительные версии для мобильных устройств или каких-то других специальных целей.

На этапе создания макета сайта нам нужно «картинку» (детализированный дизайн) превратить в живой сайт. Этот этап зависит от того как, на основе чего и для какой CMS создается код. Обычно у разработчика, уже существуют заготовленные «болванки»

HTML/CSS, поэтому после «нарезки» нужных элементов в Photoshop, процесс разметки может занять не так уж и много времени.

Следующий этап – программирование сайта. На этом этапе выполняется написание скриптов, отладка, тестирование в различных браузерах, валидации и исправления ошибок.

Далее для размещения сайта подбирается хостинг, регистрируется доменное имя. После чего, по ftp сайт закачивается на хостинг и еще раз тщательно тестируется. Можно заниматься созданием сайта сразу на хостинге – в таком случае, во-первых, не придется осуществлять “переезд сайта” с локального компьютера, во-вторых, все тестируется в реальном режиме в реальных условиях (и настройках хостера), в-третьих, заказчик может следить за процессом выполнения работы.

Поддержка сайта включает разработку документации по работе с сайтом (наполнение контентом), либо небольшое обучение заказчика. Так же на этом этапе выполняется сопровождение сайта, обновление модулей, дополнение новых возможностей по требованию заказчика.

6.1 Этапы верстки веб-страниц

Версткой веб-страниц – создание такого HTML-кода, который позволяет размещать элементы веб-страницы (изображения, текст, линии и т.д.) в нужных местах документа и отображать их в окне браузера согласно разработанному макету. При этом следует принимать во внимание ограничения присущие HTML и CSS, учитывать особенности браузеров и знать приемы верстки, которые дают желаемый результат. Верстка это процесс творческий и четких алгоритмов здесь не существует.

Вначале дизайнер готовит макеты веб-страниц в графическом редакторе (например, Adobe Illustrator, Adobe Photoshop), утверждает их у заказчика и передает верстальщику на формирование HTML-кода. Верстальщик получает работу в виде набора рисунков, где каждый из них соответствует макету отдельной страницы со своим дизайном.

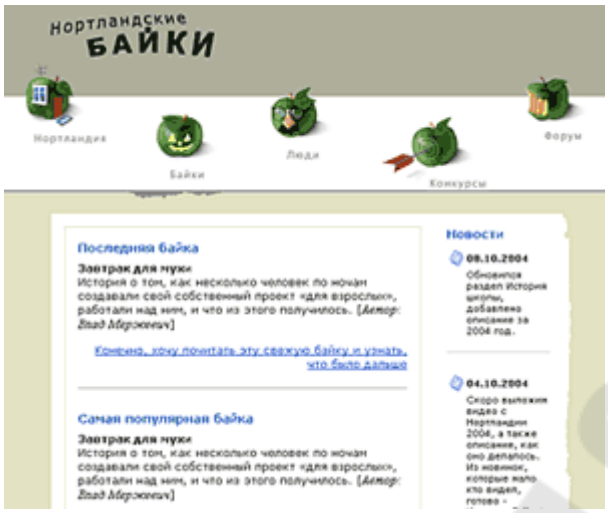


Рисунок 6.1 – Изображение главной страницы

Теперь необходимо проанализировать рисунок и решить, как же его превратить в веб-страницу. Для удобства происходит логическое разбиение картинки на отдельные блоки, с которыми идет дальнейшая работа. Выделим два крупных блока – «шапка» страницы и основной контент. Рассмотрим для начала «шапку», показанную на рисунке 6.2.

Рисунок 6.2 – Изображение «шапки» главной страницы

По задумке дизайнера цветная и белая полоса вверху должны занимать всю ширину веб-страницы, а набор пиктограмм с заголовком сайта выравнивается по центру окна браузера. Положение рисунков относительно друг друга меняться не должно и каждый из них является ссылкой на определенный раздел сайта. С учетом указанных особенностей возможны следующие варианты:

1. Сделать один рисунок и применить к нему карту-изображение;

2. Разрезать изображение на фрагменты и объединить их воедино с помощью таблицы, при этом отдельные фрагменты будут служить ссылкой;
3. Воспользоваться позиционированием элементов.

Каждый из приведенных методов хотя и приводит к нужному результату, но имеет также и свои недостатки.

После того, как первый блок будет готов и воплощен в HTML, можно переходить к работе над следующим блоком. Здесь теперь уже фигурирует текст, поэтому происходит формирование стилевого файла, в котором затронуты следующие факторы:

1. цвет фона веб-страницы;
2. гарнитура основного шрифта, его размер и цвет;
3. размер текста отдельных модулей (новостей, например);
4. цвет, размер и гарнитура шрифта заголовков;
5. параметры горизонтальных линий и рамок.

Затем используя созданный CSS-файл, формируется окончательный HTML-документ главной страницы.

Часто дизайнер готовит макет не только главной страницы, но и остальных разделов, которые несколько отличаются по своему виду от уже проделанной работы (рисунок 6.3).

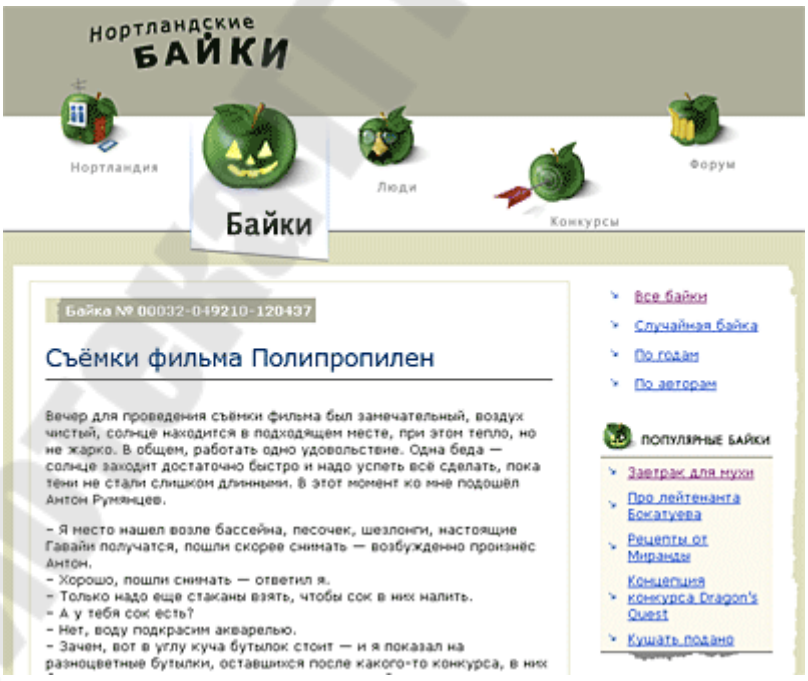


Рисунок 6.3 – Изображение макета для раздела «Байки»

В «шапку» сайта необходимо внести изменения, а в CSS включить параметры новых появившихся элементов. И так для каждого

раздела сайта. Во время работы над шаблонами и по ее окончанию происходит проверка, которая должна ответить на вопросы:

1. корректно ли отображаются страницы в популярных браузерах?
2. происходит сохранение целостности данных при изменении размера шрифта в браузере как в большую, так и меньшую сторону?
3. можно продолжать работу с сайтом, если отключить показ изображений?
4. как существенно влияет на вид страниц разрешения монитора?

Следует также учесть, что статьи могут иметь разный объем и веб-страница должна сохранять свой вид независимо от этого. Если ошибки найдены, то в шаблоны с их учетом вносятся исправления, и так до тех пор, пока число ошибок не будет сведено к минимуму.

Итогом работы верстальщика является набор шаблонов повторяющих рисунки дизайнера, но сделанных в виде HTML-документов. А также стилевой файл, в котором прописаны не только атрибуты, необходимые для верстки, но и параметры основного текста, заголовков, подзаголовков и других текстовых элементов. Это позволяет по единым шаблонам формировать любое число веб-страниц, оформление и вид которых будет одинаков.

6.2 Особенности верстки веб-страниц

Основная задача верстки веб-страниц это сформирование документа, корректно отображающегося (возможно с небольшими различиями) на основных платформах и в браузерах. Рассмотрим основные моменты возникающие при верстке веб-страниц.

Изначально разработчику сайта ширина окна браузера пользователя неизвестна, поскольку она может меняться в самых широких пределах. Ширина зависит от разрешения монитора, длины его диагонали, размера окна и еще некоторых варьируемых данных. Иными словами предугадать ее заранее простыми средствами не представляется возможным. С учетом этой особенности утвердилось два способа верстки: **фиксированный** и «**резиновый**».

При фиксированном макете, задают общую ширину макета жестко заданной и равной определенной величине. Если взять некоторую обобщенную статистику посетителей сайтов и посмотреть, какое разрешение монитора они преимущественно используют, то узнаем, что это 1024 x 768, 1280 x 1024 пикселов. Возьмем за ориентир 1024 пик-

селов, тогда общая ширина макета за вычетом вертикальной полосы прокрутки и рамки браузера окажется порядка 1000 пикселов.

Преимущество такой схемы следующее. Раз общая ширина макета точно известна, то мы можем легко подгонять под нее дизайн и делать изображения уже известной ширины. В целом подобная верстка приближается к верстке печатного буклета, и в том и другом случае ширина носителя информации строго задана, за счет чего верстка хоть частично, но упрощается.

Недостаток – недостаточно эффективное использование свободной площади. Действительно, для монитора с большой диагональю или высоким разрешением экрана документ будет смотреться по-другому, чем на предполагаемых 1024 пикселах. Чтобы хоть как-то уменьшить пустое пространство, макет обычно помещают по центру окна браузера.

«Резиновый» макет, основывается на том, что в качестве одной из единиц измерения выступают проценты. Общая рабочая ширина окна браузера – 100%, и колонки макета в сумме не должны ее превышать, поэтому для удобства, как правило, везде применяют процентную запись. При изменении размеров окна происходит переформатирование данных страницы, чтобы они вписались в новую ширину. Но существуют недостатки присущие «резиновой» верстке. Хотя веб-страница и подстраивается под ширину окна браузера, при достижении некоторой величины читать текст становится неудобно – строки слишком длинные. Эту проблему пользователь может решить сам, подобрав комфортный размер окна браузера.

Верстать «резиновый» макет сложнее, чем фиксированной ширины. Это связано с тем, что приходится учитывать множество дополнительных факторов и знать некоторые приемы верстки. К тому же, популярные браузеры неоднозначно трактуют некоторые параметры и в них «резиновый» макет может отображаться по-разному.

Любой макет имеет некоторую минимальную ширину, при достижении которой веб-страница «рассыпается» или появляется горизонтальная полоса прокрутки. «Резиновый» дизайн характеризуется активным использованием фоновых изображений, которые по горизонтали собираются без швов встык.

Пролистывание большого документа на компьютере происходит сверху вниз. Для удобства листания предназначены вертикальные полосы прокрутки, клавиатурные комбинации, колесо прокрутки мыши. А вот перемещение по горизонтали происходит не так удобно, поэто-

му горизонтальной полосы прокрутки быть не должно. Из чего следует, что веб-страница должна вписываться в окно браузера по ширине, а по высоте может изменяться в очень широком диапазоне.

Чем больше на странице информации, тем больше высота документа и тем сложнее находить нужные данные. Поэтому текст структурируют, разбивают на блоки и каждому из них дают свой заголовок, чтобы взгляду читателя было за что зацепиться.

Также следует учесть, что объем статей на сайте может достаточно сильно различаться между собой. При этом будет меняться и высота страницы, поэтому макет должен отображался без ошибок, несмотря на различное значение высоты.

Создание макета сайта не может обойтись без рисунков. Особенность их использования в том что все объекты на странице имеют прямоугольную форму, но не содержимое, благодаря чему требуемый нам дизайн можно конструировать с помощью набора изображений. Данная особенность породила некоторые техники связанные с версткой:

Активное использование рисунков – рисунки не только применяются для иллюстрации текста, но и для формирования макета страницы (например, для создания привлекательного дизайна, в качестве «распорки» между ячейками таблицы, градиентные заливки, фоновые изображения и т.д.).

Разрезание изображения на фрагменты – если рисунок слишком велик, его можно разделить на прямоугольники. Причем модифицировать картинку можно не целиком, а заменяя отдельные блоки, имитируя тем самым анимацию.

Фоновый рисунок – удобен тем, что он может заполнять всю отведенную ширину или высоту под блоком. Это позволяет создавать линии или другие декоративные элементы, которые привязываются к ширине или высоте текста и не зависят от размеров окна. Также поверх фона можно накладывать текст, что также расширяет возможности по дизайну веб-страниц.

Картинки вместо текста – создание анимированных **gif** или **Flash**, изображений для привлечения внимания. Однако имеется и оборотная сторона – рисунки занимают больший объем, чем рядовой текст, их сложнее править, они не индексируются поисковыми машинами, их показ пользователи могут отключить.

6.3 Макет сайта. Табличная верстка

Помимо своего прямого назначения, таблицы также используются для позиционирования элементов веб-страницы и создания дизайна сайта. Благодаря наличию множества параметров, как у самой таблицы, так и у ее ячеек, с помощью сочетания таблиц и рисунков можно формировать практически любые декоративные элементы. Причем их легко можно приспосабливать под любой макет, поскольку ширину таблицы можно устанавливать и в процентах и в пикселях. Преимущества табличной верстки:

1. **Удобство и широкие возможности верстки**. Таблицы давно используются для размещения разных элементов на веб-странице. Наличие большого числа настраиваемых параметров.
2. **Создание колонок** – колонная модульная сетка применяется на сайтах очень часто. Основной текст и навигацию по сайту удобнее располагать в разных колонках. При изменении размера окна браузера, колонки сохраняют свою ширину (фиксированный маке) или она меняется пропорционально («резиновый» макет). Высота разных колонок при использовании таблиц всегда остается одинаковой, независимо от объема их содержимого.
3. **«Резиновый» макет** – 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.
4. **«Склейка» изображений** – рисунки часто разрезают на отдельные фрагменты, а затем собирают их вновь в одно целое, динамически заменяя одни фрагменты другими. Это требуется для различных дизайнерских изысков например, создания эффекта перекатывания, анимации или уменьшения объема файлов. Таблицы позволяют 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.
5. **Фоновые рисунки** – в ячейки таблицы позволяют задавать фоновый рисунок, который в зависимости от размеров ячейки может повторяться по горизонтали, вертикали или сразу в двух направлениях. За счет этого приема на странице создаются декоративные линии, рамки, добавляется тень под элементом.

6. **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.
7. **Поддержка браузерами** – браузеры достаточно вольно толкуют некоторые параметры CSS. В этом смысле таблицы отображаются в разных браузерах практически одинаково, поэтому создание веб-страниц упрощается.

Несмотря на описанные достоинства таблиц, у них есть и определенные недостатки, которые порой заставляют искать другие способы верстки. Недостатки табличной верстки:

1. **Долгая загрузка** – Особенность табличной верстки, в том что пока таблица не загрузится, на экране содержимое ячеек отображаться не будет. Браузеры используют такой подход, чтобы получить всю информацию о таблице для правильного форматирования ее содержимого. Но 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 стилевых свойств.
2. **Громоздкий код** – таблицы содержат сложную иерархическую структуру вложенных тегов, которая увеличивает объем кода, и повышает сложность изменения отдельных параметров. 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 веб-страницы.
3. **Плохая индексация поисковыми системами** – 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. **Плохое разделение содержимого и оформления** – идеальный 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 веб-страницы и править его.

5. **Несоответствие стандартам** – в настоящее время стандартом является верстка слоями с использованием CSS.

6.4 Применение таблиц для верстки сайта. Макет из двух колонок

Двухколонная модульная сетка часто применяется на сайтах, при этом, как правило, в одной колонке располагается основной материал (текст статьи, например), а во второй – ссылки на разделы сайта и другая информация. Для создания такого макета хорошо подходит таблица – каждая ячейка выступает в качестве отдельной колонки, что позволяет легко регулировать различные параметры отображения документа.

Рассмотрим простой вариант, с фиксированной шириной левой колонки, а ширина правой колонки варьируется в зависимости от размеров окна браузера. Для этого требуется задать общую ширину всей таблицы в процентах через параметр **width** тега <**TABLE**> и для первой ячейки установить ее ширину в пикселах или процентах также с помощью параметра **width**, но уже для тега <**TD**>. Вертикальное выравнивание содержимого ячеек по верхнему краю определяется параметром **valign** со значением **top**. Это требуется для того, чтобы при разном объеме содержимого ячеек, они не сдвигались бы относительно друг друга, а начинались одинаково от верхнего края.

Пример.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<body>
<table width="100%" cellspacing="0" cellpadding="5" border="2"
bordercolor="black">
<tr>
  <td colspan="2" bgcolor="#FBF0DB">Заголовок</td>
</tr>
```

```
<tr>
  <td width="200" valign="top">Левая колонка</td>
  <td valign="top">Правая колонка</td>
</tr>
<tr>
   <td colspan="2">Дно</td>
</tr>
</table>
</body>
</html>
```

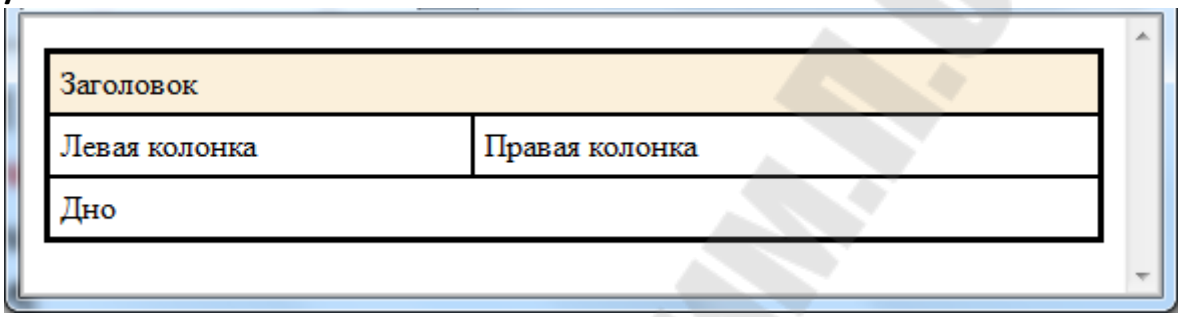


Рисунок 6.4 – Табличный макет сайта

Рассмотрим тот же вариант, но с фиксированной шириной макета 800px и выравниванием по центру относительно окна браузера. Для этого необходимо задать общую ширину всей таблицы атрибутом **width="800"** и атрибутом **align="center"** выровнять макет по центру.

Пример.

```
<table width="800" align="center" cellspacing="0" cellpadding="5"
border="2" bordercolor="black">
```

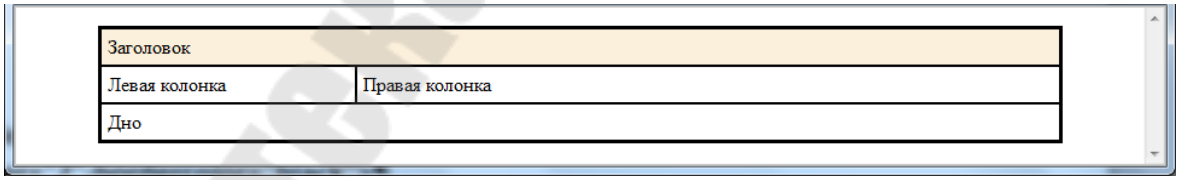


Рисунок 6.5 – Табличный макет сайта

Параметры настройки отображения таблицы можно заменить стилевыми атрибутами с теми же значениями.

Пример.

```
<html>
<head>
<style type="text/css">
#maket {
```

```
 width: 100%; /* Ширина всей таблицы в процентах */
 border-collapse: collapse; /* Отображать только одинарные линии */
}
#maket #header{
 background: #ccc; /* Цвет фона ячеек */
}
#maket TD {
 vertical-align: top; /* Вертикальное выравнивание в ячейках */
 border: 2px solid black; /* Граница вокруг ячеек */
}
TD#leftcol {
 width: 200px; /* Ширина левой колонки в пикселах */
}
#maket #footer{
 background: #ccc; /* Цвет фона ячеек */
}
</style>
</head>
<body>
<table id="maket" cellspacing="0" cellpadding="0" >
<tr>
 <td id="header" colspan="2">Заголовок</td> </tr>
 <td id="leftcol">Левая колонка</td>
 <td>Правая колонка</td>
</tr>
<tr>  <td id="footer" colspan="2">Дно</td> </tr>
</table>
</body>
</html>
```

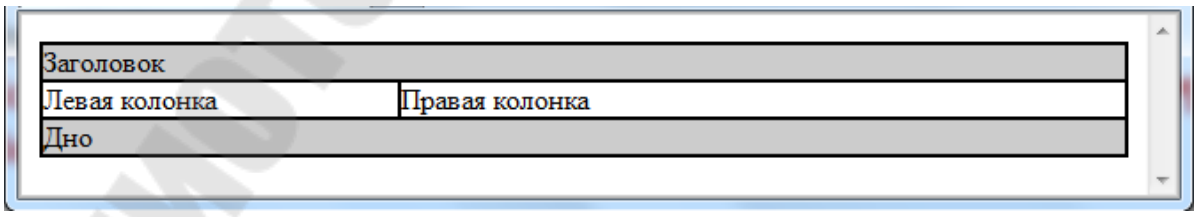


Рисунок 6.6 – Табличный макет сайта

Расстояние между колонками регулируется параметром **cellpadding** тега <**TABLE**>. Поскольку **cellpadding** определяет расстояние от границы ячейки до края ее содержимого, то пространство

между содержимым разных колонок будет равно удвоенному значению этого параметра. Используя стили, а в частности, атрибут **padding**, можно регулировать значение отступа для каждой колонки

Пример. В данном примере значения параметров **cellspacing** и **cellpadding** равны нулю, а расстояние между содержимым колонок определяется аргументом **padding-right**, который добавляется к левой ячейке через идентификатор с именем **leftcol**.

```
TD#leftcol {
 padding-right: 40px; /* Поле справа от текста */
 width: 200px; /* Ширина левой колонки в пикселах */
}
```

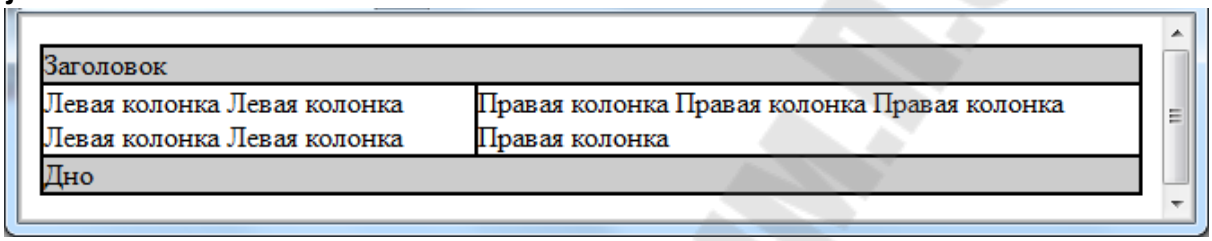


Рисунок 6.7 – Табличный макет сайта

Аналогично отступы можно регулировать не только справа, но и с других сторон каждой ячейки. Для задания поля со всех сторон:

```
TD#leftcol {
padding: 5px; /* Поля вокруг содержимого ячеек */
 width: 200px; /* Ширина левой колонки в пикселах */
}
```

Чтобы визуально отделить одну колонку от другой используют разные приемы, самым распространенным, является использование фонового цвета. Лучше указывать цвет через стили, это позволяет вынести оформление страницы в отдельный файл. Стилевое оформление задается параметром **background**.

Пример.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<style type="text/css">
#maket {
 width: 100%; /* Ширина всей таблицы в процентах */
 border-collapse: collapse; /* Отображать только одинарные линии */
}
```

```
#maket #header{
 background: #FBF0DB; /* Цвет фона ячеек */
}
#maket TD {
 vertical-align: top; /* Вертикальное выравнивание в ячейках */
 border: 2px solid black; /* Граница вокруг ячеек */
 padding: 5px; /* Поля вокруг ячеек */
}
TD#leftcol {
 width: 200px; /* Ширина левой колонки в пикселах */
 background: #ccc; /* Цвет фона левой колонки */
}
TD#rightcol {
 background: #fc3; /* Цвет фона правой колонки */
}
#maket #footer{
 background: #FBF0DB; /* Цвет фона ячеек */
}
</style>
</head>
<body>
<table cellspacing="0" cellpadding="0" id="maket">
<tr>
  <td id="header" colspan="2">Заголовок</td>
</tr>
 <td id="leftcol">Левая колонка</td>
 <td id="rightcol">Правая колонка</td>
</tr>
<tr>
  <td id="footer" colspan="2">Дно</td>
</tr>
</table>
</body>
</html>
```

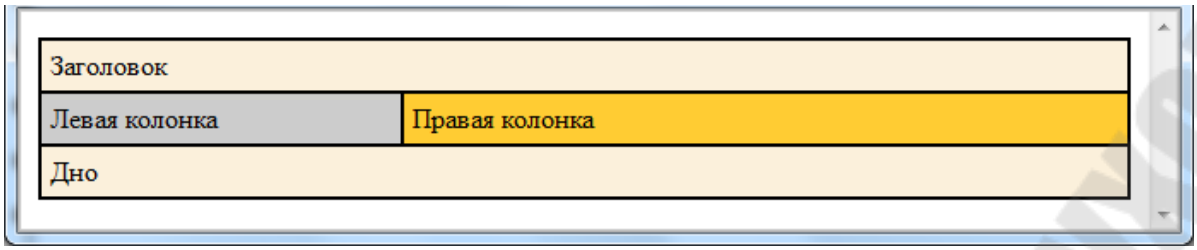


Рисунок 6.8 – Табличный макет сайта

Использование полей не всегда подходит для установки нужного расстояния между колонок. Например, в случае, когда поля вокруг текста нельзя включать в силу разных соображений. Тогда в качестве разделителя между колонками можно использовать дополнительный столбец.

Пример.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<style type="text/css">
#maket {
 width: 100%; /* Ширина всей таблицы в процентах */
 border-collapse: collapse; /* Отображать только одинарные линии */
}
#maket #header{
 background: #FBF0DB; /* Цвет фона ячеек */
}
#maket TD {
 vertical-align: top; /* Вертикальное выравнивание в ячейках */
 padding: 5px; /* Поля вокруг ячеек */
}
TD#leftcol {
 width: 200px; /* Ширина левой колонки в пикселах */
 background: #ccc; /* Цвет фона левой колонки */
}
#maket #spacer {
 width: 10px /* Расстояние между колонками */
}
TD#rightcol {
 background: #fc3; /* Цвет фона правой колонки */
}
#maket #footer{
```

```
 background: #FBF0DB; /* Цвет фона ячеек */
}
</style>
</head>
<body>
<table cellspacing="0" cellpadding="0" id="maket">
<tr> <td id="header" colspan="3">Заголовок</td> </tr>
 <td id="leftcol">Левая колонка </td>
 <td id="spacer"></td>
 <td id="rightcol">Правая колонка</td>
</tr>
<tr> <td id="footer" colspan="3">Дно</td> </tr>
</table>
</body>
</html>
```

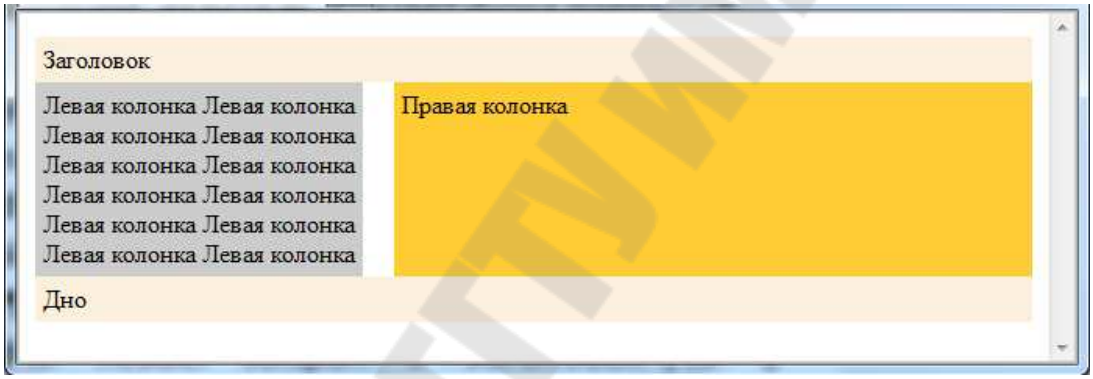


Рисунок 6.9 – Табличный макет сайта

Разделять колонки можно не только с помощью цвета фона и пустого пространства, но и добавлением линии между колонок. Для этого также можно использовать соответствующие стили: border-left, **border-right**, **border-top**, **border-bottom** или **border** – для всех четырех границ сразу. Общий вид:

border-bottom: [border-width || border-style || border-color] | inherit

Значение **border-width** определяет толщину границы. Для управления ее видом предоставляется несколько значений свойства **border-style**. Значение **border-color** –устанавливает цвет границы. Значение **inherit** – наследует значение родителя.

Рисунок 6.10 – Стили рамок

Пример.

```
TD#leftcol {
 width: 200px; /* Ширина левой колонки в пикселах */
 background: #ccc; /* Цвет фона левой колонки */
 border-right: 4px solid blue; /* Параметры линии */
}
```

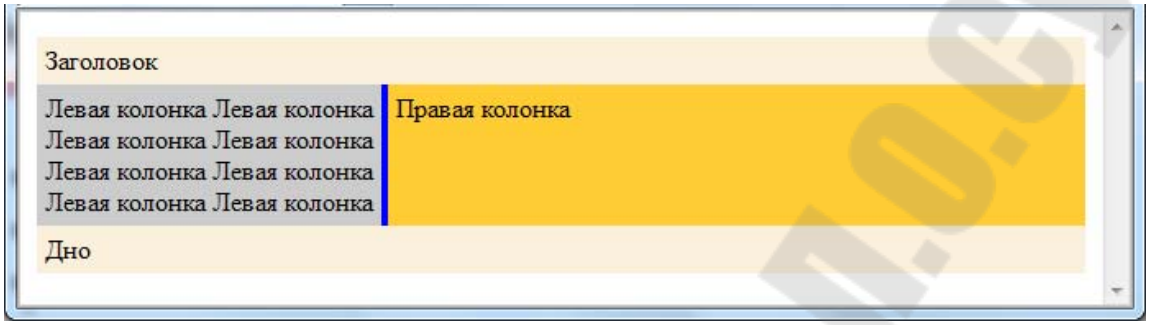


Рисунок 6.11 – Табличный макет сайта

Создание колонок с помощью таблиц процесс достаточно простой и быстрый, необходимо создать таблицу и определить ее визуальные атрибуты. К тому же большинство параметров, определяющих вид таблицы, можно вынести в стили и таким образом ускорить процесс добавления однотипных таблиц и документов на их основе.

При двухколонном макете применяются разные средства по оформлению колонок. Например, используется фоновая заливка, рамка вокруг колонок, изменяется расстояние между ними или установка вертикальной разделительной линии. Все это управляется с помощью стилей, что ведет к сокращению кода, расширяет варианты модификаций таблиц и удобство разработки сайта.

6.5 Применение таблиц для верстки сайта. Макет из трех колонок

Использование трех колонок на страницах сайта обусловлено объемом информации, которую требуется показать посетителю. Обычно одна колонка, самая широкая, отдается под текст основного материала, статьи, например, а остальные колонки применяются для навигации, рекламы, анонсов и т.д. Принцип создания трехколонной модульной сетки с помощью таблицы аналогичен созданию двухколонной сетки, поэтому остановимся лишь на некоторых моментах.

Ширина разных колонок зависит от используемого макета – фиксированного или «резинового». При макете фиксированной ширины

общая ширина таблицы задается в пикселах и остается постоянной независимо от разрешения монитора и размера окна браузера. При этом ширину отдельных колонок также имеет смысл установить в пикселах. Пусть ширина макета задана как 800 пикселов, а колонки соответственно 200, 400 и 200 пикселов. При определении ширины колонок следует принимать во внимание значение параметра **cellpadding**. На ширину ячеек этот аргумент не влияет, но зато уменьшает область, которая отводится под содержимое ячеек.

Пример.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<body>
<table width="800" align="center" cellspacing="0" cellpadding="5"
border="2" bordercolor="black">
<tr>  <td colspan="3" bgcolor="#FBF0DB">Заголовок</td> </tr>
<tr>
  <td width="200" valign="top">Навигация</td>
  <td width="400" valign="top">Контент</td>
  <td width="200" valign="top">Реклама</td>
</tr>
<tr>  <td colspan="3">Дно</td>  </tr>
</table>
</body>
</html>
```

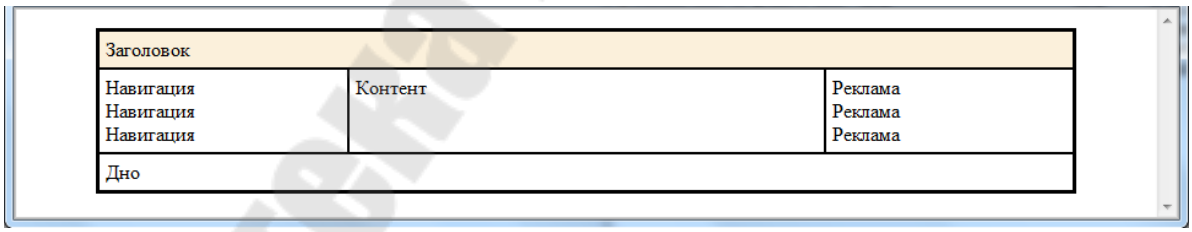


Рисунок 6.12 – Табличный макет сайта фиксированной ширины

Ширину всех ячеек в подобном случае задавать не обязательно. Так, если не указать ширину одной ячейки, то она будет вычислена автоматически исходя из общей ширины таблицы и ширины остальных ячеек. В других случаях, например, когда не установлена ширина двух ячеек, их размер определяется по содержимому. Поскольку содержимое ячеек варьируется от страницы к странице, то ширина также будет «плавать», поэтому ширину колонок лучше указывать задавать.

При «резиновом» макете ширина таблицы устанавливается в процентах от ширины окна браузера и, таким образом, напрямую зависит от нее. Здесь возможны два варианта:

1. ширина всех ячеек задана в процентах;
2. сочетание процентов и пикселов, когда ширина одних ячеек устанавливается в процентах, а других – в пикселах.

В первом случае вначале устанавливается ширина всей таблицы в процентах, а затем ширина отдельных ячеек. Причем в сумме ширина ячеек должна получиться 100%, несмотря на то, что размер таблицы может быть иным. Это обусловлено тем, что ширина таблицы вычисляется относительно доступного пространства веб-страницы, а размер ячеек устанавливается относительно всей таблицы в целом.

Пример.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<style type="text/css">
#maket {
 width: 90%; /* Ширина всей таблицы в процентах */
 border-collapse: collapse; /* Отображать только одинарные линии */
}
#maket TD {
 vertical-align: top; /* Вертикальное выравнивание в ячейках */
 border: 2px solid black; /* Граница вокруг ячеек */
 padding: 5px; /* Поля вокруг ячеек */
}
#maket #header{
 background: #FBF0DB; /* Цвет фона ячеек */
}
#col1 {
 width: 20%; /* Ширина первой колонки */
 background: #fc0; /* Цвет фона первой колонки */
}
#col2 {
 width: 40%; /* Ширина второй колонки */
 background: #f0f0f0; /* Цвет фона второй колонки */
}
#col3 {
 width: 40%; /* Ширина третьей колонки */
 background: #fc0; /* Цвет фона третьей колонки */
```

```
}
#maket #footer{
 background: #FBF0DB; /* Цвет фона ячеек */
}
</style>
</head>
<body>
<table id="maket" align="center" cellpadding="5" cellspacing="0">
<tr> <td id="header" colspan="3">Заголовок</td></tr>
<tr>
   <td id="col1">Навигация</td>
   <td id="col2">Контент 1</td>
   <td id="col3">Контент 1</td>
</tr>
<tr><td id="footer" colspan="3">Дно</td></tr>
</table>
</body>
</html>
```

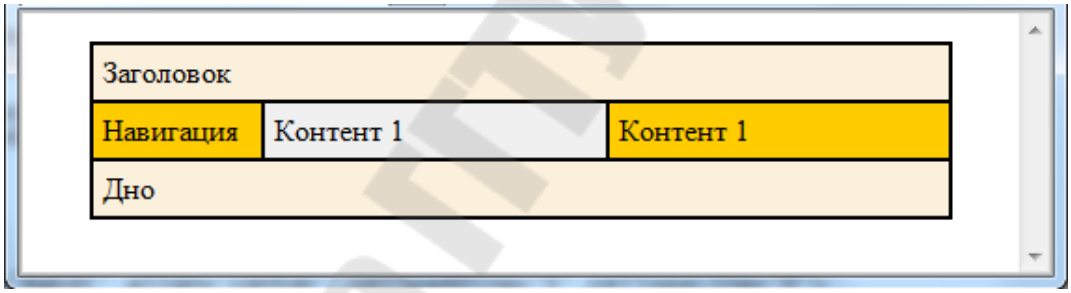


Рисунок 6.13 – «Резиновый» табличный макет сайта

Процентная запись для таблиц имеет ряд преимуществ – используется все свободное пространство веб-страницы, а сам макет подстраивается под ширину окна браузера. Вместе с тем каждая таблица имеет некоторый минимальный размер, при достижении которого таблица уже не уменьшается и начинает отображаться горизонтальная полоса прокрутки. Такой минимальный размер зависит от содержимого таблицы. Если, например, в каждую из трех ячеек поместить по рисунку шириной 200 пикселов, то общая ширина таблицы не может быть меньше 600 пикселов плюс значения полей вокруг изображений.

Рассмотрим два основных варианта, когда для задания ширины колонок одновременно применяются проценты и пикселы. Первый вариант состоит в том, что размер крайних колонок устанавливается в

пикселах, а ширина средней колонки вычисляется автоматически, исходя из заданной ширины таблицы.

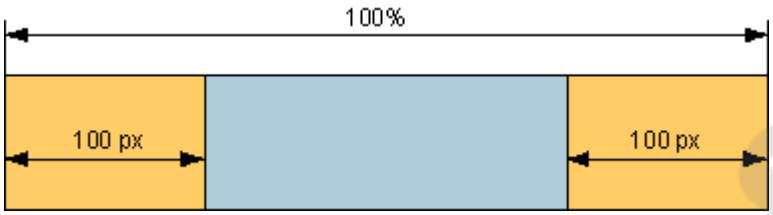


Рисунок 6.14 – Табличный макет. Ширина средней колонки определяется браузером

Для создания подобного макета понадобится таблица с тремя ячейками. Ширину первой и третьей ячейки устанавливаем в пикселах, а ширину средней ячейки намеренно не задаем. При этом обязательно следует определить общую ширину всей таблицы.

Пример.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<style type="text/css">
TABLE {
 width: 100%; /* Ширина таблицы */
}
TD {
 vertical-align: top; /* Выравнивание по верхнему краю ячейки */
}
#col1 {
 width: 100px; /* Ширина первой колонки */
 background: #fc0; /* Цвет фона первой колонки */
}
#col2 {
 background: #afccdb; /* Цвет фона второй колонки */
}
#col3 {
 width: 100px; /* Ширина третьей колонки */
 background: #fc0; /* Цвет фона третьей колонки */
}
</style>
</head>
<body>
```

```
<table cellpadding="5" cellspacing="0">
<tr>
        <td id="col1">Колонка 1</td>
        <td id="col2">Колонка 2</td>
        <td id="col3">Колонка 3</td>
</tr>
</table>
</body>
</html>
```

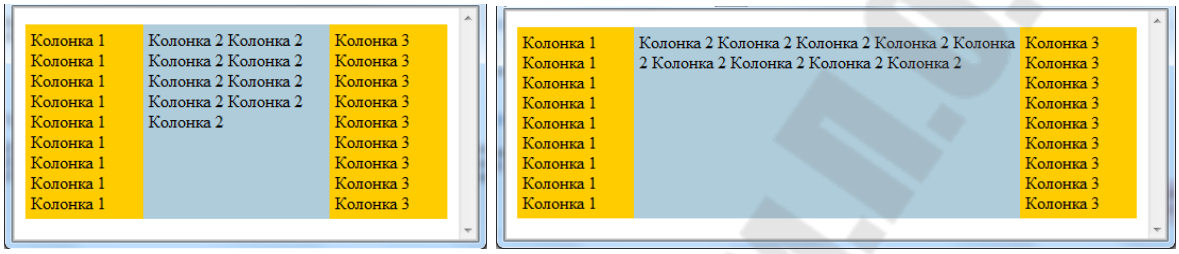


Рисунок 6.15 – Табличный макет. Ширина средней колонки определяется браузером

Во втором варианте ширина двух колонок устанавливается в процентах, а третьей – в пикселах. В этом случае обойтись одной таблицей не удастся, поскольку если ширина всей таблицы равна 100%, первой колонки – 100 пикселов, а оставшихся колонок по 20%, то простое вычисление показывает, что размер первой колонки получается равным 60%. Поэтому заданное значение в пикселах браузером будет проигнорировано, а размер установлен в процентах.

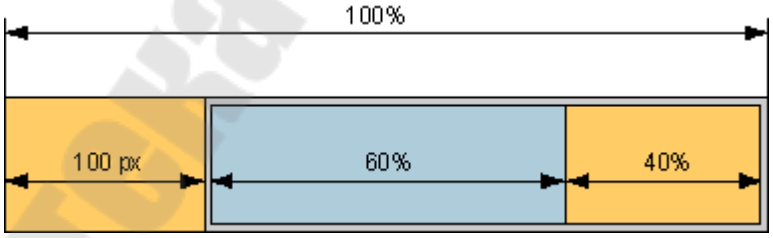


Рисунок 6.16 – Табличный макет. Применение вложенных таблиц

Вначале создаем таблицу заданного размера с двумя ячейками. Левая ячейка будет выступать в роли первой колонки, и для нее устанавливаем требуемую ширину в пикселах. Ширину для правой ячейки не определяем, поэтому она будет занимать оставшееся пространство, а также служить каркасом для других колонок. Внутрь этой ячейки добавляем вторую таблицу, тоже состоящую из двух ячеек. И уже для них определяем ширину в процентах.

Пример.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<style type="text/css">
TABLE {
width: 100%; /* Ширина таблиц */
}
TD {
vertical-align: top; /* Выравнивание по верхнему краю ячейки */
}
#col1 {
width: 100px; /* Ширина первой колонки */
background: #fc0; /* Цвет фона первой колонки */
}
#col2 {
width: 60%;
background: #afccdb; /* Цвет фона второй колонки */
}
#col3 {
width: 40%; /* Ширина третьей колонки */
background: #fc0; /* Цвет фона третьей колонки */
}
#col1, #col2, #col3 {
padding: 5px; /* Поля вокруг текста */
}
</style>
</head>
<body>
<table cellpadding="0" cellspacing="0">
<tr>
<td id="col1">Колонка 1 </td>
<td>
 <table cellpadding="0" cellspacing="0">
  <tr>
   <td id="col2">Колонка 2</td>
   <td id="col3">Колонка 3</td>
  </tr>
 </table>
</td>
```

```
</tr>
</table>
</body>
</html>
```

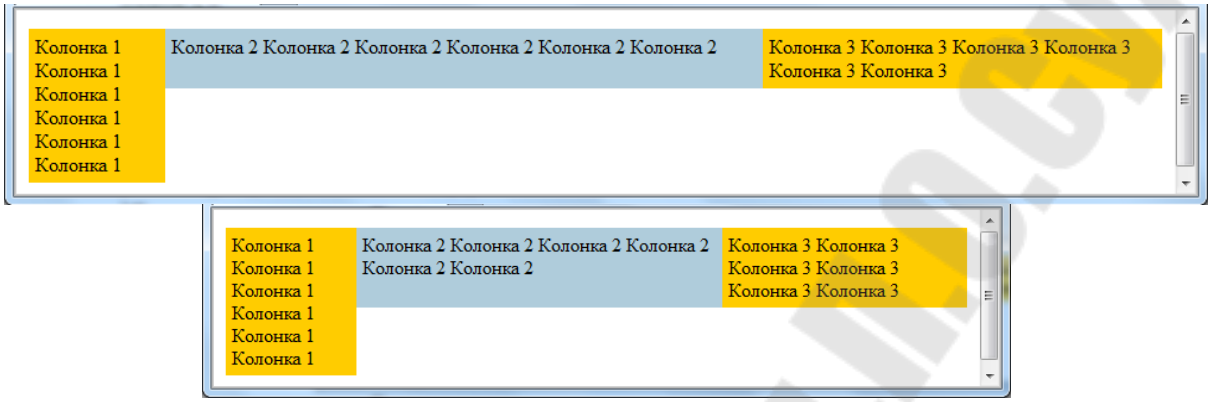


Рисунок 6.17 – Табличный макет. Применение вложенных таблиц

При создании подобного макета следует принимать во внимание следующие моменты:

1. Ширина внутренней таблицы должна быть задана как 100%, чтобы эта таблица занимала все свободное пространство.
2. Для того чтобы ячейки плотно прилегали друг к другу, для внешней таблицы необходимо обнулить значение параметров **cellpadding** и **cellspacing**. Поля можно устанавливать через атрибут **padding**.
3. Ширина второй и третьей колонки вычисляется относительно ширины ячейки, а не внешней таблицы в целом. Поэтому значение 60% в примере следует расценивать не как ширину колонки относительно всего макета, а лишь как ширину относительно внутренней таблицы.

6.6 Применение таблиц для создания рамок

Таблицы могут использоваться не только для добавления колонок или выравнивания элементов веб-страницы, но и для создания различных дизайнерских изысков вроде декоративной рамки или тени. Причем их легко можно приспосабливать под любой макет, поскольку ширину таблицы можно устанавливать и в процентах и в пикселах.

Для создания желаемой рамки ее вначале следует нарисовать в каком-нибудь графическом редакторе. Хотя конечная ширина рамки

на веб-странице может варьироваться в больших пределах, например, в случае использования резинового макета, размер уголков остается постоянным. Поэтому при создании изображения следует в первую очередь ориентироваться именно на уголки.

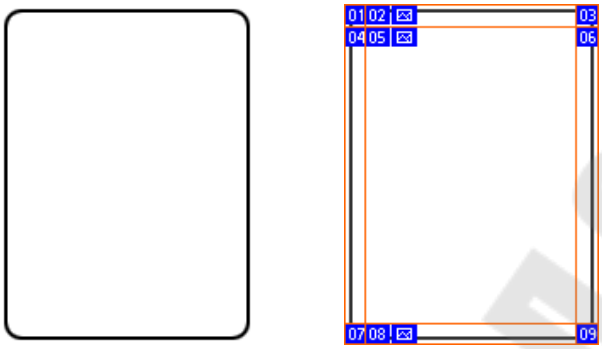


Рисунок 6.18 – Макет декоративной рамки с номерами рисунков.

Теперь изображение рамки необходимо разрезать на девять фрагментов (смотри рисунок 7.18).

Таблица 7.1 – Рисунки, необходимые для создания рамки

Рисунок	**Положение**	**Имя файла**
	Левый верхний угол	*01.gif*
	Верхняя горизонтальная линия	*02.gif*
	Правый верхний угол	*03.gif*
	Левая вертикальная линия	*04.gif*
	Правая вертикальная линия	*06.gif*
	Левый нижний угол	*07.gif*
	Нижняя горизонтальная линия	*08.gif*
	Правый нижний угол	*09.gif*

После создания нужных фрагментов формируем таблицу размером 3x3 ячейки.

Рисунок 6.19 – Макет таблица для создания рамки

Ширина самой таблицы может задаваться как в пикселах, так и процентах, от этого в итоге зависит размер рамки. А вот ширина и высота крайних ячеек должна совпадать с размерами соответствующих рисунков – уголков, вертикальных и горизонтальных линий и устанавливаться в пикселах. Также в этой таблице параметры **border**, **cellspacing** и **cellpadding** должны быть равны нулю, иначе линии не будут состыковываться между собой.

Пример.

```
<p>Пример задания декоративной рамки вокруг ячеек.</p>
<table width="400" border="0" cellspacing="0" cellpadding="0">
 <tr>
  <td height="13" width="12"><img src="01.gif" width="12"
height="13"></td>
  <td background="02.gif" height="13"></td>
  <td height="13" width="14"><img src="03.gif" width="14"
height="13"></td>
 </tr>
 <tr>
  <td background="04.gif"></td>
  <td>Содержимое</td>
  <td background="06.gif"></td>
 </tr>
 <tr>
  <td height="13"><img src="07.gif" height="13"></td>
  <td background="08.gif"></td>
  <td height="13"><img src="09.gif" height="13"></td>
 </tr>
</table>
```

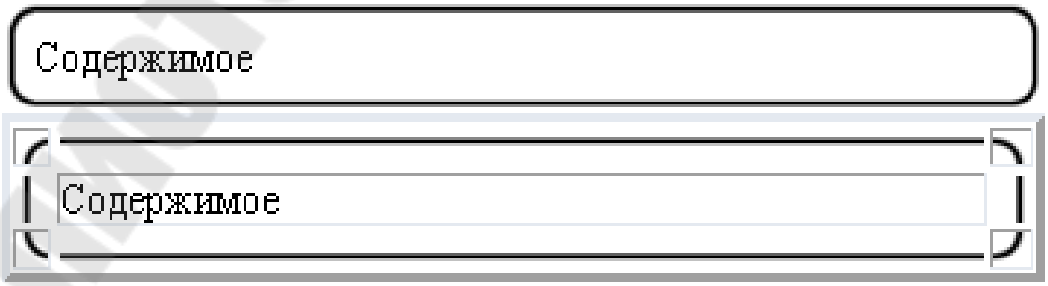


Рисунок 6.20 – Декоративная рамка

6.7 Применение таблиц для склейки изображений

Разрезание изображения на фрагменты с последующим их объединением в одну целую картинку, прием который часто используется при верстке веб-страниц. Предварительно подготовленный рисунок разрезают на части в графическом редакторе, сохраняют их как отдельные графические изображения, а затем соединяют их вместе с помощью таблицы. Рассмотрим преимущества такой техники:

1. **Создание ссылок** – отдельные рисунки при необходимости можно превращать в ссылки, причем для них можно назначать свое описание (параметр title) и альтернативный текст (параметр alt), который виден при отключении показа картинок в браузере или при наведении курсора мыши на изображение.
2. **Эффект перекатывания** – набор отдельных фрагментов позволяет создавать **rollover** или эффект перекатывания – динамическое изменение одного рисунка на другой при наведении на него курсора мыши, и обратно на прежний, когда курсор уводится прочь.
3. **Уменьшение объема файлов** – отдельными частями изображения удобней манипулировать, подбирая для них графический формат и его параметры таким образом, чтобы объем файла был минимален при сохранении приемлемого качества изображения. В итоге набор графических файлов будет занимать меньше места, и загружаться быстрее, чем один файл, содержащий целый рисунок.
4. **Анимированный GIF** – использование анимированного GIFa для изображений большого размера чревато существенным увеличением объема файла. Но если заменить лишь часть изображения анимацией, а остальные фрагменты оставить статичными. При этом общий объем нескольких файлов будет гораздо меньше, чем анимирование одного изображения.
5. **Особенности верстки** – изображения на веб-странице по своей природе прямоугольны, но, разрезав один рисунок на составляющие элементы, получим конструктор, из которого можно сложить другую фигуру. Складывать подобные фигуры на веб-странице требуется в силу разных причин, например, вместо фрагмента изображения требуется добавить текст. Кроме того, некоторые рисунки можно заменить их фоновым аналогом, и тогда конечное изображения, сохраняя свою целостность, будет за-

нимать всю доступную область документа. Однотонные рисунки можно вовсе исключить, заменив их на заливку соответствующим цветом.

6. **Психологический аспект** – когда один рисунок состоит из множества фрагментов, то браузер скачивает их в несколько потоков и показывает те, которые загрузились в первую очередь. Поэтому изображение появляется как элементы мозаики. А это сразу привлекает внимание и кажется, что загрузка происходит быстрее. Так что с технической стороны один рисунок грузится быстрее, а с позиции человеческого восприятия кажется, что набор маленьких рисунков быстрее появляется. Так же можно применить технологию когда все куски сохраняются в одном файле, а нужный фрагмент выбирается с помощью позиционирования и задания размеров, но такая техника не подходит для фоновых изображений.

Рассмотрим на примере технологию склейки изображений. В качестве примера изображения, где требуется разрезание, возьмем рисунок 6.21(а). Каждая из пяти иконок является ссылкой на соответствующий раздел, кроме того, ссылкой на главную страницу служит рисунок с названием сайта.

Рисунок 6.21 – Исходное изображение (а), вид изображения при открытии раздела «Байки» (б)

Теоретически, в данном случае можно обойтись и без разрезания, если использовать изображение–карту (теги <**MAP**> и <**AREA**>). Однако этот вариант неприемлем в силу следующих соображений. По задумке разработчиков при открытии любого раздела, иконка ему соответствующая, трансформируется, что в целом меняет изображение целиком (рисунок 6.21(б)). Если применять изображение-карту, то придется заготовить шесть различных изображений (одну для главной страницы и еще пять для каждого раздела), а это скажется в итоге на объеме пересылаемых данных, скорости отображения сайта и качестве рисунков.

Следующим этапом требуется решить, как разрезать изображение. Вариантов может быть несколько необходимо определиться с наилучшим.

Разрезание и «сборку» рисунка лучше доверить специализированной программе, в частности, это умеет делать Adobe Photoshop, Adobe ImageReady, Macromedia FireWorks и др. В дальнейшем для этой будем цели использовать **Photoshop**, так что все упоминания об инструментах и меню относится именно к этой программе.

Для удобства разрезания изображения вначале следует добавить направляющие линии, по которым затем и будет происходить разделение на фрагменты (рисунок 6.22).

Рисунок 6.22 – Исходное изображение с направляющими

Теперь используем инструмент **Slice** (активация клавишей <K>) и по направляющим обводим требуемую прямоугольную область. Обозначенная область отмечается синей рамкой с номером фрагмента в левом верхнем углу. Размер областей можно изменять через специальный инструмент **Slice Select** – . Щелкаем мышью с этим инструментом по желаемому фрагменту – цвет рамки вокруг области становится желтым, а также изменяется тональность рисунка.

Для быстрого переключения между инструментами **Slice** и **Slice Select** нажмите и удерживайте клавишу <**Ctrl**>.После чего курсором мыши можно перемещать границы фрагмента за специальные маркеры по бокам и в углах области (рисунок 6.23).

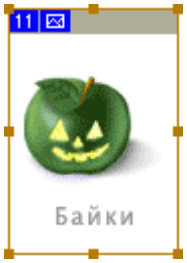


Рисунок 6.23 – Исходное изображение с направляющими

Во время изменения размеров фрагментов, следите за тем, чтобы области не пересекались друг с другом, и между ними не возникало промежутков. Хотя **Photoshop** сам отмечает подобные недочеты и принимает меры к их устранению, лучше держать все под своим контролем. После предварительного анализа и применения инструмента **Slice**, получим 18 фрагментов (рисунок 6.24).

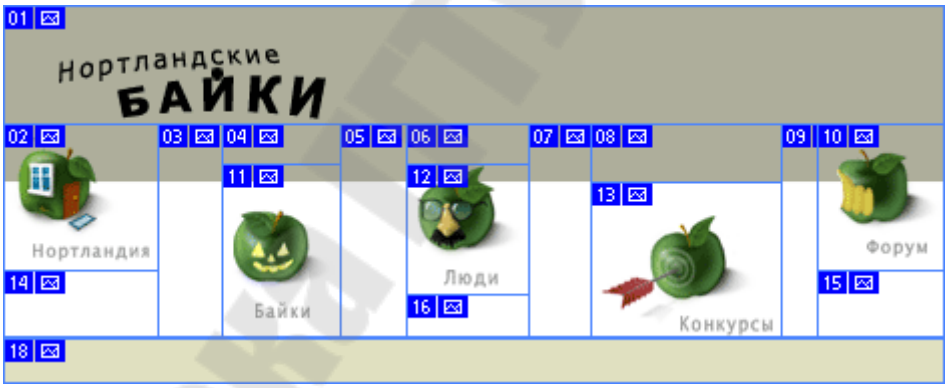


Рисунок 6.24 – Исходное изображение с направляющими

Опасаться того, что получилось много рисунков, не стоит из-за того, что часть фрагментов содержит пустое изображение (14, 16 и 15), а часть фрагментов, за исключением ширины, идентичны (3, 5, 7 и 9). Таким образом, число картинок сокращается, поскольку часть из них можно заменить одним прозрачным однопиксельным рисунком, устанавливая у него такие же размеры, как у исходного фрагмента.

Использование однопиксельного прозрачного рисунка в формате GIF достаточно распространенный прием при верстке веб-страниц. Действительно, объем файла минимален (всего 43 байта), картинку

при этом можно масштабировать до любого размера, и сквозь нее виден фон.

После того, как фрагменты обозначены, требуется сохранить все изображения на диск. Для этого выбираем пункт меню **File** > **Save for Web**... (<**Alt**>+<**Shift**>+<**Ctrl**>+<**S**>) чтобы открыть панель оптимизации графики (рисунок 6.25).

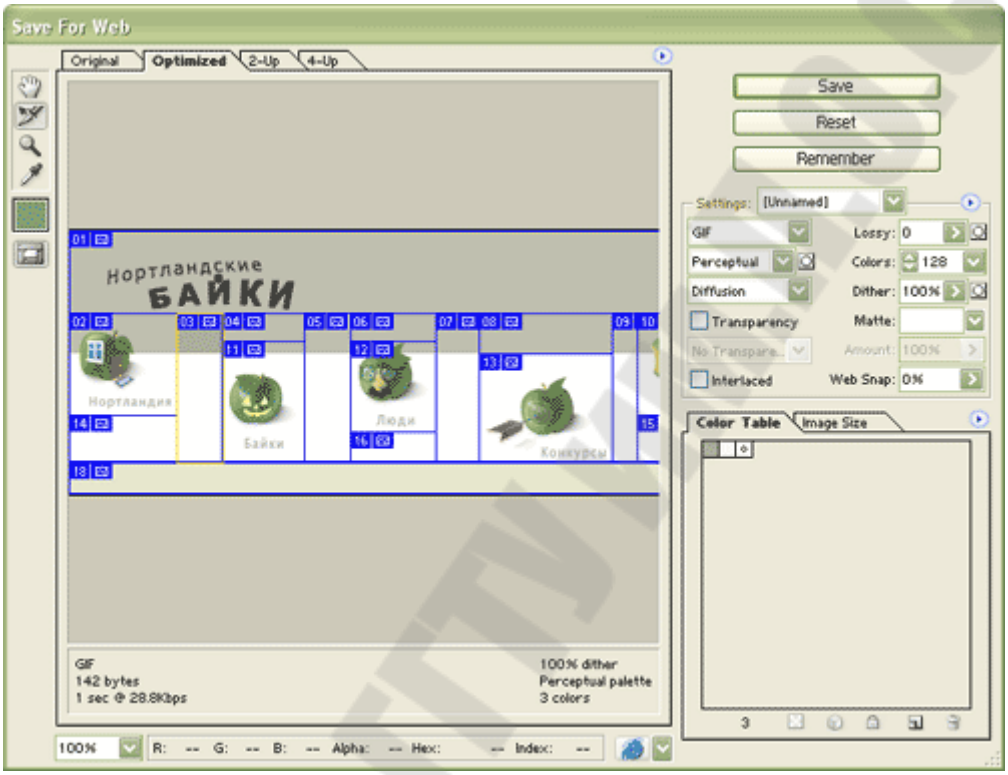


Рисунок 6.25 – Исходное изображение с направляющими

С помощью инструмента **Slice Select** можно выбирать требуемый фрагмент и устанавливать для него персональные параметры вроде количества цветов, значение потерь качества, прозрачность и т.д. Допускается выделять сразу несколько фрагментов, удерживая клавишу <**Shift**>, что позволяет устанавливать для них одинаковые параметры.

По окончанию работы с фрагментами нажимаем кнопку **Save** и указываем место на диске, куда будет сохранен HTML-документ. Рисунки сохраняются автоматически в папку **images**, а их имя образуется от имени HTML-файла с добавлением номера фрагмента. Например, сохраняемое имя будет **splash**.**php**, тогда первый фрагмент называется **splash_01.gif**, а последний – **splash_18.gif**. Кроме того, создается файл, который представляет собой уже упоминаемый прозрачный рисунок размером 1x1 пиксел. Он используется для правильного формирования изображений в таблице.

Настройки, по которым строится HTML-код и формируются имена изображений можно изменить, если при сохранении файла в разделе **Settings** выбрать пункт **Other**... .

Полученный в результате сохранения файлов HTML-код, после небольшого редактирования имеет следующий вид:

```
<html>
<body>
<table width="670" border="0" cellpadding="0" cellspacing="0" align="center">
 <tr>
  <td colspan="9"><img src="images/splash_01.gif" width="670" height="85" alt=""></td>
  <td><img src="images/spacer.gif" width="1" height="85" alt=""></td>
 </tr>
 <tr>
  <td rowspan="3"><img src="images/splash_02.gif" width="111" height="104" alt=""></td>
  <td rowspan="5"><img src="images/splash_03.gif" width="45" height="150" alt=""></td>
  <td><img src="images/splash_04.gif" width="84" height="28" alt=""></td>
  <td rowspan="5"><img src="images/splash_05.gif" width="47" height="150" alt=""></td>
  <td><img src="images/splash_06.gif" width="87" height="28" alt=""></td>
  <td rowspan="5"><img src="images/splash_07.gif" width="45" height="150" alt=""></td>
  <td rowspan="2"><img src="images/splash_08.gif" width="135" height="41" alt=""></td>
  <td rowspan="5"><img src="images/splash_09.gif" width="25" height="150" alt=""></td>
  <td rowspan="3"><img src="images/splash_10.gif" width="91" height="104" alt=""></td>
  <td><img src="images/spacer.gif" width="1" height="28" alt=""></td>
 </tr>
 <tr>
  <td rowspan="4"><img src="images/splash_11.gif" width="84" height="122" alt=""></td>
  <td rowspan="3"><img src="images/splash_12.gif" width="87" height="93" alt=""></td>
  <td><img src="images/spacer.gif" width="1" height="13" alt=""></td>
 </tr>
 <tr>
  <td rowspan="3"><img src="images/splash_13.gif" width="135" height="109" alt=""></td>
  <td><img src="images/spacer.gif" width="1" height="63" alt=""></td>
 </tr>
 <tr>
  <td rowspan="2"><img src="images/splash_14.gif" width="111" height="46" alt=""></td>
  <td rowspan="2"><img src="images/splash_15.gif" width="91" height="46" alt=""></td>
  <td><img src="images/spacer.gif" width="1" height="17" alt=""></td>
 </tr>
 <tr>
  <td><img src="images/splash_16.gif" width="87" height="29" alt=""></td>
  <td><img src="images/spacer.gif" width="1" height="29" alt=""></td>
 </tr>
 <tr>
  <td colspan="9"><img src="images/splash_17.gif" width="670" height="2" alt=""></td>
  <td><img src="images/spacer.gif" width="1" height="2" alt=""></td>
 </tr>
 <tr>
  <td colspan="9"><img src="images/splash_18.gif" width="670" height="32" alt=""></td>
  <td><img src="images/spacer.gif" width="1" height="32" alt=""></td>
 </tr>
</table>
</body>
</html>
```

Данный код еще требует доработки, поскольку требуется, чтобы горизонтальная серая и белая полоса занимали всю доступную ширину веб-страницы. Кроме того, часть фрагментов повторяется и от них можно избавиться. Чтобы получить требуемый результат, введем слой с фоновым рисунком и нашу таблицу наложим поверх него. Такое изображение представлено на рисунок 6.26.

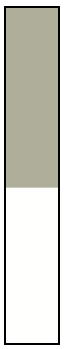


Рисунок 6.26 – Фоновый рисунок (рамка приведена для наглядности)

Теперь создаем нужный слой, назовем его **toplayer** и зададим его стилевое оформление:

```
<html>
<head>
<style type="text/css">
#toplayer {
background: #aeae9b /* Цвет фона */
 url("images/bgtop.gif") /* Фоновый рисунок */
 repeat-x; /* Повторять фон только по горизонтали */
height: 235px; /* Высота слоя */
border-bottom: 2px solid #8f8f8f; /* Линия внизу */
}
</style>
</head>
<body>
<div id="toplayer">
...
</div>
</body>
</html>
```

Параметры фона устанавливаются через свойство **background**, которое определяет путь к графическому файлу, цвет заливки и по-

вторяемость рисунка. Хотя цвет фона в таких случаях можно не указывать раз есть фоновый рисунок, но на случай того, что пользователь отключил загрузку изображений, лучше это сделать. Высота слоя также не является обязательным параметром из-за того, что таблица внутри слоя имеет заданную высоту.

Остается заменить фрагменты с номерами 3, 4, 5, 6, 7, 8, 9 14, 15 и 16 на рисунок spacer.gif, сохранив исходные размеры изображений и удалить нижнюю строку таблицы, которая для главной страницы является лишней. Использование файла spacer.gif позволяет сократить конечный код и уменьшить требуемое число графических файлов.

В ряде случаев для задания однотонных изображений можно использовать задание фона ячейки таблицы и ее размеров.

Окончательный код:

```
<html>
<head>
<style type="text/css">
#toplayer {
 background: #aeae9b url("images/bgtop.gif") repeat-x;
 height: 235px;
 border-bottom: 2px solid #8f8f8f;
}
</style>
</head>
<body>
<div id="toplayer">
<table width="670" border="0" cellpadding="0" cellspacing="0" align="center">
 <tr>
  <td colspan="9"><img src="images/splash_01.gif" width="670" height="85"></td>
  <td><img src="images/spacer.gif" width="1" height="85"></td>
 </tr>
 <tr>
  <td rowspan="3"><img src="images/splash_02.gif" width="111" height="104"></td>
  <td rowspan="5"><img src="images/spacer.gif" width="45" height="150"></td>
  <td><img src="images/spacer.gif" width="84" height="28"></td>
  <td rowspan="5"><img src="images/spacer.gif" width="47" height="150"></td>
  <td><img src="images/spacer.gif" width="87" height="28"></td>
  <td rowspan="5"><img src="images/spacer.gif" width="45" height="150"></td>
  <td rowspan="2"><img src="images/spacer.gif" width="135" height="41"></td>
  <td rowspan="5"><img src="images/spacer.gif" width="25" height="150"></td>
  <td rowspan="3"><img src="images/splash_10.gif" width="91" height="104"></td>
  <td><img src="images/spacer.gif" width="1" height="28"></td>
 </tr>
 <tr>
  <td rowspan="4"><img src="images/splash_11.gif" width="84" height="122"></td>
  <td rowspan="3"><img src="images/splash_12.gif" width="87" height="93"></td>
  <td><img src="images/spacer.gif" width="1" height="13"></td>
 </tr>
 <tr>
  <td rowspan="3"><img src="images/splash_13.gif" width="135" height="109"></td>
  <td><img src="images/spacer.gif" width="1" height="63"></td>
```

```
</tr>
<tr>
 <td rowspan="2"><img src="images/spacer.gif" width="111" height="46"></td>
 <td rowspan="2"><img src="images/spacer.gif" width="91" height="46"></td>
 <td><img src="images/spacer.gif" width="1" height="17"></td>
</tr>
<tr>
 <td><img src="images/spacer.gif" width="87" height="29"></td>
 <td><img src="images/spacer.gif" width="1" height="29"></td>
</tr>
</table>
</div>
</body>
</html>
```

Литература

1. А. Ломов. HTML, CSS, скрипты: практика создания сайтов. БХВ-Петербург, 2007 – 416с.
2. А.А. Дуванов. Web-конструирование. HTML. — СПб.: БХВ-Петербург, 2003. — 325 с.
3. Бен Хеник. HTML и CSS. Путь к совершенству. Питер, 2011 г– 336с.
4. В. Дронов. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов. БХВ-Петербург, 2011 – 416с.
5. В. Мержевич. HTML и CSS на примерах. БХВ-Петербург, 2005 – 448с.
6. И. Квинт. HTML, XHTML и CSS на 100%. Питер, 2010 – 384с.
7. Кристофер Шмитт. CSS. Рецепты программирования. БХВ-Петербург, 2011. – 672 с.
8. М. Дубаков. Создание Web-страниц. Искусство верстки. Новое знание, 2004 – 228с.
9. Тиге Дж.К. DHTML и CSS для Internet. М.: НТ Пресс, 2005
10. Э. Кастро. HTML и CSS для создания Web-страниц. НТ Пресс, 2006 – 126с.
11. http://htmlbook.ru — учебники HTML, CSS
12. http://www.w3.org – сайт World Wide Web Consortium
13. http://www.xiper.net – учебники HTML, CSS, разработка сайтов

Литвинов Дмитрий Александрович

ОСНОВЫ WEB-ПРОГРАММИРОВАНИЯ

Курс лекций
по одноименному курсу
для слушателей специальности
1-40 01 73 «Программное обеспечение
информационных систем»
заочной формы обучения

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического комплекса 18.03.15.
Рег. № 11Е.
http://www.gstu.by