

Министерство образования Республики Беларусь

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

Кафедра «Информатика»

Т. В. Тихоненко, В. О. Лукьяненко

ОСНОВЫ WEB-ПРОГРАММИРОВАНИЯ

ПОСОБИЕ

по одноименному курсу

для студентов специальности 1-40 04 01

«Информатика и технологии программирования»

дневной формы обучения

Электронный аналог печатного издания

Гомель 2016

УДК 004.738.52(075.8)
ББК 32.973.202я73
Т46

*Рекомендовано к изданию научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого
(протокол № 10 от 25.05.2015 г.)*

Рецензент: доц. каф. «Информационные технологии» ГГТУ им. П. О. Сухого
канд. физ.-мат. наук, доц. *Е. Г. Стародубцев*

Тихоненко, Т. В.
Т46 Основы web-программирования : пособие по одному курсу для студентов специальности 1-40 04 01 «Информатика и технологии программирования» днев. формы обучения / Т. В. Тихоненко, В. О. Лукьяненко. – Гомель : ГГТУ им. П. О. Сухого, 2016. – 121 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц; 32 Mb RAM; свободное место на HDD 16 Mb; Windows 98 и выше; Adobe Acrobat Reader. – Режим доступа: <http://elib.gstu.by>. – Загл. с титул. экрана.

ISBN 978-985-535-316-5.

Представлен курс лекций по дисциплине «Основы web-программирования», предназначенный для освоения базовых технологий создания HTML-страниц. Изложен доступным языком и содержит множество примеров, показывающих возможности использования HTML5 и CSS3. Уделено внимание правилам использования графики, видео и звука при создании web-страниц, даны принципы формирования карт-изображений и пользовательских форм. Приведены примеры верстки сайтов.

Для студентов специальности 1-40 04 01 «Информатика и технологии программирования» дневной формы обучения.

УДК 004.738.52(075.8)
ББК 32.973.202я73

ISBN 978-985-535-316-5

© Тихоненко Т. В., Лукьяненко В. О., 2016
© Учреждение образования «Гомельский
государственный технический университет
имени П. О. Сухого», 2016

ОГЛАВЛЕНИЕ

ГЛАВА 1 ВВЕДЕНИЕ И ОСНОВНЫЕ ПОНЯТИЯ	5
1.1 Понятие Всемирной паутины.....	5
1.2 Понятия web-сервера, web-сайта, web-страницы и их отличия.....	5
ГЛАВА 2 ВВЕДЕНИЕ В ЯЗЫК ГИПЕРТЕКСТОВОЙ РАЗМЕТКИ СТРАНИЦ HTML	7
2.1 Понятие тега. Типы тегов	7
2.2 Структура и правила оформления HTML-документа.....	8
2.3 Основные элементы форматирования текста. Элементы блочные и строчные	9
2.4 Понятие элементов и атрибутов.....	12
2.5 Специальные символы языка HTML.....	16
2.6 Типы HTML-документов, валидация HTML-кода	17
ГЛАВА 3 ГИПЕРССЫЛКИ И ГРАФИЧЕСКИЕ ВОЗМОЖНОСТИ HTML	18
3.1 Механизмы адресации на ресурсы в Интернете. Понятие гиперссылки.....	18
3.2 Элемент <a>... и его атрибуты	18
3.3 Размещение изображений на web-странице. Типы файлов изображений.....	20
3.4 Элемент IMG и его атрибуты	21
3.5 Навигационные карты	23
ГЛАВА 4 ВВЕДЕНИЕ В КАСКАДНЫЕ ТАБЛИЦЫ СТИЛЕЙ CSS. ОСНОВЫ РАБОТЫ С CSS.....	27
4.1 Основные цели и задачи CSS. Способы добавления стилей на web-страницу	27
4.2 Основные понятия и определения. Грамматика языка стилей	30
4.3 Создание классов и их применение на web-странице	32
4.4 Единицы измерения размеров.....	34
4.5 Применение селекторов к элементам на web-странице	36
4.6 Декоративные возможности CSS: форматирование текста.....	50
ГЛАВА 5 СПИСКИ И ТАБЛИЦЫ	55
5.1 Структурирование информации на web-странице при помощи списков. Типы списков	55
5.2 CSS свойства для списков.....	57

5.3 Таблица и ее элементы	58
5.4 Правила задания атрибутов для таблицы и ее ячеек. Объединение ячеек	60
5.5 Верстка при помощи таблиц. Вложенные таблицы.....	63
5.6 Таблицы и стили	65
ГЛАВА 6 ФРЕЙМЫ И ФОРМЫ	69
6.1 Фреймы и их описание на языке HTML. Задание логики взаимодействия фреймов	69
6.2 Форма и ее элементы. Методы отправки информации из полей формы	72
6.3 Формы и стили	82
ГЛАВА 7 ИСПОЛЬЗОВАНИЕ СТИЛЕЙ ДЛЯ МАКЕТИРОВАНИЯ ...	83
7.1 Декоративные возможности CSS: формирование рамок и отступов	83
7.2 Наследование и каскадирование	86
7.3 Позиционирование элементов на странице и управление моделью элемента	88
ГЛАВА 8 ОСНОВЫ WEB-ПРОГРАММИРОВАНИЯ ПРИ ВЕРСТКЕ WEB-СТРАНИЦ	93
8.1 Этапы и особенности верстки web-страниц	93
8.2 Режимы браузеров	95
8.3 Приемы использования таблиц на web-странице	96
8.4 Верстка на CSS. Создание простого блока	98
8.6 Современная верстка сайта при помощи CSS	100
8.6 Современная верстка на HTML5 и CSS3	104
ГЛАВА 9 ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ HTML И CSS ...	115
9.1 Размещение мультимедийной информации на web-странице	115
9.2 Импорт CSS	117
9.3 Использование внешних объектов.....	118
9.4 Описание метаинформации сайта.....	120
ЛИТЕРАТУРА	121

ГЛАВА 1 ВВЕДЕНИЕ И ОСНОВНЫЕ ПОНЯТИЯ

1.1 Понятие Всемирной паутины

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

Начиная с момента своего возникновения, разработкой спецификации языка HTML стала заниматься специально созданная организация – Консорциум W3C (<http://www.w3c.org>) [1].

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

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

Системой «клиент–сервер» называют механизм передачи информации между удаленным компьютером, предоставляющим свои ресурсы в распоряжение пользователей, и пользовательским компьютером, эксплуатирующим эти ресурсы. Сервер должен представлять собой «информационный портал», т. е. он является достаточно боль-

шим виртуальным пространством, состоящим из множества различных тематических разделов меньшего размера, либо некоторого количества самостоятельных проектов.

Разберемся в том, как это происходит. Цифра 1 на рисунке 1.1 – это запрос «клиента» на сервер о необходимом документе, адрес которого набирается либо вручную, либо с помощью нажатия по гиперссылке. Цифра 2 рисунка 1.1 означает, что сервер сформировал ответ в виде текста и отдал его «клиенту». Далее пользовательский браузер разбирает этот текст, анализирует его, определяет, какие есть еще файлы, которые используются на этой web-странице. Цифра 3 на рисунке 1.1 показывает запрос, который идет для получения этих файлов (картинки, таблицы стилей, флэш и другие отдельные фрагменты). Причем, цифра 3 уходит на тот же самый сервер, но по факту все эти файлы могут находиться на других серверах. Получив все части web-страницы (цифра 4 на рисунке 1.1), браузер отдает пользователю необходимый документ (цифра 5 на рисунке 1.1). Вот основные этапы получения информации из Всемирной паутины.



Рисунок 1.1 – Организация системы «клиент–сервер»

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

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

ГЛАВА 2 ВВЕДЕНИЕ В ЯЗЫК ГИПЕРТЕКСТОВОЙ РАЗМЕТКИ СТРАНИЦ HTML

2.1 Понятие тега. Типы тегов

Любой язык разметки состоит из двух основных частей: информация, которую нужно передать (т. е. текст, который размечается), и данные разметки, которые можно узнать по угловым скобкам `< >`. Такого образа конструкции называются *тегами*. Другими словами, тег – это инструкция браузеру, указывающая способ отображения передаваемой информации.

Теги могут быть двух видов:

– одноэлементные или одиночные `<..>`. Элементы, образованные при помощи одиночных тегов, называют *пустыми*;

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

Есть еще специальный вид тега, который в HTML называется *декларацией*. Он используется исключительно для служебной информации. Одним из видов декларации является *комментарий*. Любой HTML-комментарий должен начинаться с конструкции `<!--` и заканчиваться конструкцией `-->`. Между ними может находиться любой текст, цифры, символы и прочее, за исключением тегов.

HTML-комментарии позволяют размечать структуру HTML-кода, давая заголовки различным логическим блокам элементов. Впоследствии такая разметка поможет быстро сориентироваться и найти необходимый фрагмент кода. Кроме того, в комментариях можно указать информацию об авторском праве и прочие персональные данные, которые будут проиндексированы поисковыми системами. Следует помнить, что вложенность комментариев недопустима.

2.2 Структура и правила оформления HTML-документа

В силу своей специфики HTML-документы должны создаваться на основе некоторых правил. Чтобы обеспечить согласованность HTML-документов с этими правилами, а также сделать возможной проверку их корректности, в HTML-документах должны присутствовать определенные элементы. В языке HTML есть заранее определенный набор таких элементов. Вся необходимую информацию об элементах языка HTML можно найти на сайте консорциума [1].

Абсолютно в любом языке разметки есть единственный корневой элемент. *Корневым* называют элемент, внутри которого находятся все остальные элементы. В HTML корневым элементом является `<html>...</html>`. Любой HTML-документ должен начинаться тегом `<html>` и заканчиваться тегом `</html>`. Этот тег указывает на то, что данный документ содержит в себе HTML-текст. Большинство браузеров способно распознать HTML-документ и без указания данного тега, тем не менее, пропускать раздел HTML разработчикам не рекомендуется.

Языком HTML предусмотрены еще два обязательных раздела:

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

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

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

Язык HTML не чувствителен к регистру, например, BODY эквивалентно body или BoDu. Однако лучше писать в нижнем регистре, поскольку в HTML5 используется только нижний регистр.

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

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

Таким образом, любой HTML-документ имеет следующую структуру:

```
<html>
  <head>
    <title>Заголовок документа</title>
  </head >
  <body>
    ...текст документа...
  </body >
</html>
```

После создания HTML-файла его необходимо просмотреть в браузере, результат отображен на рисунке 2.1.

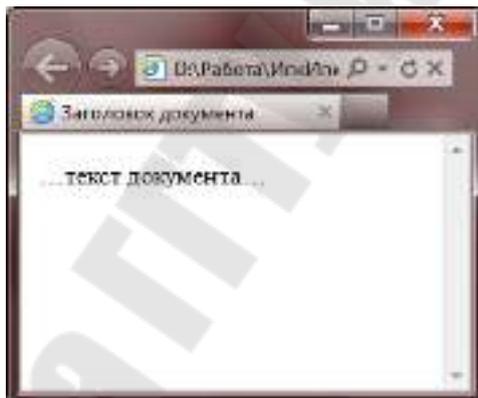


Рисунок 2.1 – Вид HTML-документа в окне браузера

2.3 Основные элементы форматирования текста. Элементы блочные и строчные

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

В случае необходимости указания специальных свойств отдельному фрагменту текста используются теги *текстовых блоков* `` и `<div>`. Теги `` и `<div>` являются яркими представителями строчных и блочных элементов, соответственно. Поэтому между `` и `<div>` имеются существенные отличия. Во-первых, `` берет начало в области физического форматирования текста, а `<div>` является исключительно структурным тегом. Во-вторых, `<div>` создает принудительный перенос строки на одну позицию после своего закрывающего тега. Тег `` позволяет назначать новые правила отображения текстовых фрагментов без изменения структуры документа. Обращаем Ваше внимание на то, что для тегов `<div>` и `` обязательно наличие закрывающих тегов `</div>` и ``.

Особенности работы тегов `</div>` и `` можно увидеть на рисунке 2.2.

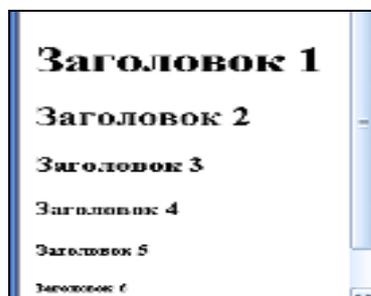


Рисунок 2.2 – Блочные и строчные элементы

Метки вида `<hi>` ($i = 1 \div 6$) описывают *заголовки* шести различных уровней. На рисунке 2.3 показано, что заголовок `<h6>` будет минимальным, а `<h1>` – самым большим. На рисунке 2.3, б изображен вид заголовков различных уровней в окне браузера, а на рисунке 2.3, а – их HTML-код.



а)



б)

Рисунок 2.3 – Заголовки различных уровней

Использование заголовков помогает отделить фрагменты текста по смыслу, а также помечать важность этих фрагментов. Заголовки являются блочными элементами, поэтому внутри заголовков могут находиться только строчные элементы.

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

Элемент `<hr>` (*горизонтальная линия*) является одиночным, пустым и блочным. Этот элемент образует полосу и выполняет функцию визуального разделителя фрагментов содержимого HTML-страницы. Элемент `<hr>` является блочным.

При создании HTML-документа для обозначения *абзаца* (*параграфа*) используется специальный парный тег `<p>...</p>`, который разделяет фрагменты текста вертикальным отступом.

Если необходимо перейти на новую строку, не прерывая абзаца, лучше использовать строчный пустой элемент `
`. Использование тега `
` очень удобно при публикации стихов.

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

Для форматирования коротких цитат используется элемент `<q>...</q>`. Содержимое элемента отображается без разрыва строки. Цитаты, оформленные с помощью элемента `<cite>...</cite>`, выделяются курсивом.

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

К элементам *визуального оформления* относятся теги:

`<i>...</i>` – выделение текста курсивом;

`...` – выделение текста полужирным шрифтом.

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

`<i>Полужирный курсив</i>`.

К элементам *логического форматирования* относятся следующие элементы:

 – выделение важных фрагментов текста *курсивом*.

 – выделение наиболее важных фрагментов полужирным.

Рекомендуется заменять <i> на , а на .

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

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

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

<cite> – выделение цитат *курсивом*.

<code> – отображение фрагментов программного кода моноширинным шрифтом.

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

<samp> – выделение нескольких символов моноширинным шрифтом.

<dfn> – определение вложенного термина *курсивом*.

2.4 Понятие элементов и атрибутов

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

Синтаксис записи тега с атрибутами следующий:

<тег имя_атр_1="значение_1"... имя_атр_n="значение_n">.

Например: <hr align = "left" width = "50%" size = 4>.

Все атрибуты можно разделить на три основные группы:

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

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

– *уникальные*. К этой группе относятся индивидуальные атрибуты некоторых элементов.

Отметим, что корневой элемент `<html>` атрибутов не имеет.

Рассмотрим некоторые атрибуты для рассмотренных выше элементов.

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

Для заголовков `<hi>...</hi>` (*i* – цифра от 1 до 6) атрибут `align` означает, что его содержимое будет выравниваться по левому краю при `align = "left"` (по умолчанию); по правому краю – при `align = "right"`, все строки будут по центру при `align = "center"`, выравнивание произойдет как по левому, так и по правому краю при `align = "justify"`.

У пустого элемента `<hr>` есть 4 атрибута: `align`, `noshade`, `size`, `width`, `color`. Например, тег `<hr width = 15>` определяет горизонтальную линию в 15 пикселей.

Значение цвета в атрибутах языка HTML может задаваться несколькими способами.

– *по шестнадцатеричному значению*.

Для задания цветов используются шестнадцатеричная система, которая базируется на числе 16. Цифры будут следующие: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Числа больше 15 в шестнадцатеричной системе образуются объединением двух чисел в одно. Например, числу 255 в десятичной системе соответствует число FF в шестнадцатеричной системе. Чтобы не возникало путаницы в определении системы счисления, перед шестнадцатеричным числом ставят символ решетки #, например, #666999. Каждый из трех цветов – красный, зеленый и синий – может принимать значения от 00 до FF. Таким образом, обозначение цвета разбивается на три составляющие: #RRGGBB, где первые два символа отмечают красную компоненту цвета, два средних – зеленую, а два последних – синюю. Допускается использо-

вать сокращенную форму вида #RGB, где каждый символ следует удваивать (#RRGGBB). К примеру, запись #FE0 расценивается как #FFEE00.

– по названию.

Браузеры поддерживают некоторые цвета по их названию. В таблице 1.1 приведены шестнадцатеричный код, название и описание. Более полную таблицу цветов можно найти в [1], [3].

Таблица 1.1 – Некоторые стандартные цвета

Код	Название	Описание
#000000	ЧЕРНЫЙ	black
#FFFFFF	БЕЛЫЙ	white
#FF0000	КРАСНЫЙ	red
#008000	ЗЕЛЕНый	green
#0000FF	СИНИЙ	blue
#808000	ОЛИВКОВый	olive
#FFFF00	ЖЕЛТый	Yellow
#C0C0C0	СЕРЕБРИСТый	Silver
#808080	СЕРый	Gray
#FF00FF	ФУКСИНОВый	Fuchsia
#000080	УЛЬТРАМАРИН	Navy
#008080	СИЗый	Teal
#00FF00	СВЕТЛО-ЗЕЛЕНый	Lime
#800080	ПУРПУРНый	Purple
#800000	КАШТАНОВый	maroon

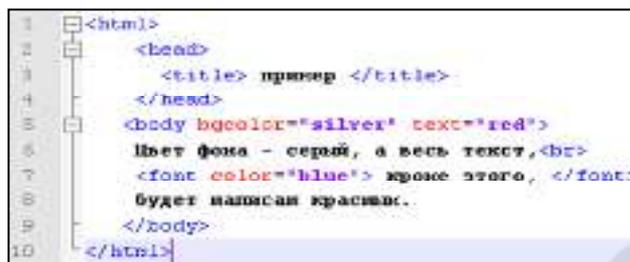
– с помощью RGB.

Можно определить цвет, используя значения красной, зеленой и синей составляющей в десятичном исчислении. Каждый из трех цветов может принимать значения от 0 до 255. Также можно задавать цвет в процентном отношении. Вначале указывается ключевое слово rgb, а затем в скобках, через запятую указываются компоненты цвета, например, rgb(255, 0, 0) или rgb(100%, 20%, 20%).

Для вставки цвета фона в строку с тегом <body> нужно добавить атрибут **bgcolor** и указать его значение – название цвета или его шестнадцатеричный вид. Фон страницы станет красным, если прописать одну из следующих строк:

```
<body bgcolor="red"><!-- использовано название цвета-->
<body bgcolor="#FF0000"><!--использован шестнадцатеричный вид
цвета-->
```

Для задания цвета текста в HTML есть специальный атрибут **text**, который может применяться только в теге `<body>`.



```
1 <html>
2   <head>
3     <title> пример </title>
4   </head>
5   <body bgcolor="silver" text="red">
6     Цвет фона - серый, а весь текст, <br>
7     <font color="blue"> кроме этого, </font>
8     будет написан красным.
9   </body>
10 </html>
```

Рисунок 2.4 – Атрибуты тегов `<body>` и ``

Тег `<body bgcolor = "silver" text = "red">` задаст цвет фона HTML-страницы серым, а цвет текста – красным (рисунок 2.4).

Для определения свойства шрифта в HTML есть специальный элемент `...`. Этот элемент имеет 3 атрибута: `color`, `size`, `face`.

Синтаксис элемента следующий:

```
<font color="знач_1" size="знач_2" face="знач_3">
любой текст
</font>.
```

Чтобы написать текст красным цветом, необходимо вставить перед ним парный тег `...` и присвоить его атрибуту **color** значение `red` или `#FF0000` (см. рисунок 2.4):

```
<font color = "red">Красный текст</font>
ИЛИ
<font color = "#FF0000">Красный текст</font>.
```

Вторым атрибутом элемента `...` является размер шрифта – **size**. Параметр этого атрибута может быть описан либо абсолютной, либо относительной величиной. Абсолютная величина подразумевает использование в качестве параметра целого числа, указывающего на высоту шрифта в пунктах. Относительная же величина, обозначаемая целым числом со знаком плюс или минус (например, `+2` или `-1`), – это количество пунктов, которые следует прибавить или отнять от размера шрифта, используемого браузером (по умолчанию 12pt). Так, запись `` говорит о том, что размер помеченного таким образом текста будет на один пункт больше, чем обычный текст документа. Кроме того, размер шрифта можно записы-

вать целыми числами от 1 до 7 (самый мелкий шрифт имеет размер 1, а самый большой – 7; `size="3"` – значение шрифта по умолчанию).

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

```
<font face = "Times New Roman">  
Текст с начертанием Times New Roman  
</font>.
```

Следует помнить, что если на компьютере посетителя Вашей web-страницы не установлен шрифт с указанным названием, то браузер использует свой стандартный шрифт. Вы можете задать несколько возможных шрифтов, тогда в случае отсутствия первого шрифта браузер будет использовать второй, если нет второго, использует третий и т. д. Например:

```
<font face = "Comic Sans MS, Courier New, Courier">.
```

2.5 Специальные символы языка HTML

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

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

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

1. Символы, отображающие элементы HTML-форматирования.
2. Символы оформления документа.
3. Буквы иностранных алфавитов.

Полный перечень всех специальных символов, а также их кодовые и числовые конструкции можно найти в [1], [3].

2.6 Типы HTML-документов, валидация HTML-кода

Существует множество причин, по которым веб-страница может отображаться некорректно: отсутствие необходимых закрывающих тэгов, неверно отформатированные таблицы, ошибки JavaScript, ошибки в CSS, некорректные запросы к серверу и т. д. Зачастую такие ошибки – следствие того, что разработчик просто не проверяет свою работу, не проводит валидацию кода. В других случаях ошибки – следствие недостатка знаний и опыта при разработке веб-страниц.

Нарушить правильность отображения HTML-страниц может также игнорирование использования тэга DOCTYPE или неправильное его использование.

DOCTYPE сообщает, согласно каким правилам синтаксиса проводить проверку текущего документа. К примеру, для HTML 4.01 и XHTML 1.0 существует по три разных типа DOCTYPE, для HTML5 – лишь один, который переводит браузер в стандартный режим:

```
<!DOCTYPE html>.
```

Если на HTML-странице используется этот тип DOCTYPE, значит на HTML-странице используется HTML5 [1]–[2].

ГЛАВА 3 ГИПЕРССЫЛКИ И ГРАФИЧЕСКИЕ ВОЗМОЖНОСТИ HTML

3.1 Механизмы адресации на ресурсы в Интернете. Понятие гиперссылки

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

- ресурс, на который ссылается документ, существует;
- структура гиперссылки верна с точки зрения HTML.

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

3.2 Элемент `<a>...` и его атрибуты

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

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

Элемент `<a>...` имеет следующие атрибуты:

href – адрес любого файла в Интернете. Он может быть абсолютными, т. е. указывается полный адрес странички (например, `http://tut.by/index.html`) и относительным, когда указывается файл относительно текущего (например, `index.html`).

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

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

_blank – загружает гиперссылку в новом окне браузера.

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

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

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

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

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

Если щелкнуть на ссылку, откроется другой документ, в данном случае – index.html.

Создать ссылку на адрес электронной почты можно следующим образом:

```
<a href = "mailto:Email@tut.by"> Email </a>.
```

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

Ссылка на закладку в том же документе имеет следующий вид:

```
<a href="#Имя_закладки">Название раздела</a>.
```

Так выглядит ссылка на закладку в другом документе:

```
<a href="Имя_документа#Имя_закладки">  
Название раздела  
</a>.
```

Сама закладка будет такой:

```
<a name="Имя_закладки"> ...текст... </a> или  
<тег id="Уникальное_имя_закладки">...текст...</тег>.
```

Щелкнув по фразе «Название раздела», пользователь будет попадать на определенную Вами закладку.

Для того чтобы при наведении на ссылку курсором и при клике она меняла свой цвет, в тег `<body>` нужно добавить еще несколько параметров: `text` – цвет текста; `link` – цвет ссылки; `vlink` – цвет пройденной ссылки; `alink` – цвет активной ссылки, когда подводится к ней курсор. Данные атрибуты определяют свойства для всего документа. Поместив такой код в HTML-документ, уже не надо будет назначать каждый раз цвет текста и цвет ссылок, везде он будет таким, каким определен в теге `<body>`.

3.3 Размещение изображений на web-странице. Типы файлов изображений

На данный момент практически все браузеры широко поддерживают три формата: GIF, JPEG, PNG [1]–[2]. Рассмотрим кратко каждый из них.

Формат GIF (Graphic Interchange Format – формат обмена графикой), созданный в 1987 г., является первым форматом, который появился для обмена медийной информацией. Изображение, закодированное в формат GIF, всегда содержит от 2 до 256 цветов. Такая палитра цветов называется *индексированной* или *фиксированной*.

Формат JPEG (Join Photographic Experts Group – объединенная группа экспертов в области фотографии) – это графический стандарт, созданный на основе одноименного алгоритма сжатия, изображений с потерей качества, кодирующего не идентичные элементы, а межпиксельные интервалы. Особенностью формата JPEG является то, что он всегда искажает картинку. Невозможно создать изображение формата JPEG с качеством как у оригинала. Искажение происходит при каждом сжатии. Максимальное число цветов, которое может содержать изображение в формате JPEG, достигает 16 млн.

Формат PNG (Portable Network Graphics – портативная сетевая графика) – это формат, который поддерживает «черезстрочность» не только по горизонтали, но и по вертикали. Формат PNG имеет следующие разновидности: PNG-8, PNG-24, PNG-32.

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

3.4 Элемент IMG и его атрибуты

Картинку можно вставить как объект на web-странице, что осуществляется с помощью элемента ``. Данный элемент относят к числу особенных, которые еще называют строчными элементами с замещающим контентом (содержимым). К таким элементам относятся: `img`, `iframe`, `object`, `embed`. Эти элементы ведут себя как строчные, но внутри них отображается посторонний внешний контент. Чтобы показать (загрузить) картинку, необходимо прописать следующую строку: ``.

Например,

```
 или  
, или  
.
```

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

```
.
```

Изображение можно вставлять внутрь гиперссылок, тем самым делая его кликабельным:

```
<a href="URL" target="окно">  
  
</a>
```

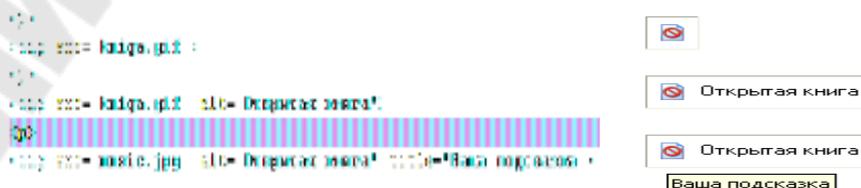


Рисунок 3.1 – Использование атрибута alt

На рисунке 3.1 представлен случай, когда изображение не загрузилось, но область под него была сформирована. Чтобы пользователь представлял, что на этом месте должна быть картинка, разработчики ставят атрибут `alt`, где пишут описание своего изображения.

Рассмотрим вопрос расположения изображений относительно текста. На рисунке 3.2 представлен самый простой случай, при котором рядом с изображением располагается только одна строка текста.

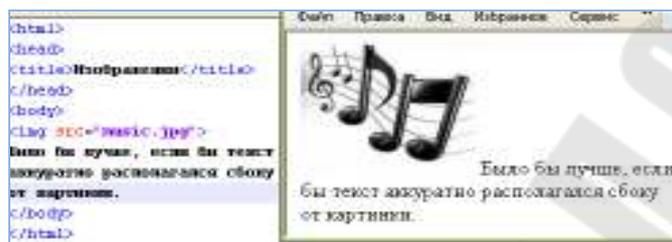


Рисунок 3.2 – Расположение изображений относительно текста

Как же сделать так, чтобы текст располагался весь рядом с изображением, а не только одна его строка? В этом случае поможет атрибут `align`: ``, это означает, что картинка будет прижата к левому краю экрана, а текст будет обтекать ее справа. Чтобы сделать наоборот (картинка справа, текст слева), надо прописать: ``.

Атрибут `align` у картинок может принимать и другие значения. Текст может располагаться внизу картинки (по умолчанию) `align="bottom"`, посередине – `align="middle"`, и вверху – `align="top"`.

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

```
<body background = "путь_к_файлу">
```

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

3.5 Навигационные карты

Уже говорилось об изображениях и о том, как изображение сделать ссылкой. Пользователи интернета знают, что при нажатии на разные области (части) одного и того же изображения, можно попадать на разные страницы. Это называется *навигационной картой*.

Навигационные карты задаются элементом `<map>...</map>`. Тэг `<map>` включает в себя тэг(и) `<area>`, которые определяют геометрические области внутри карты и ссылки, связанные с каждой областью. Конструкция элементов при создании кликабельных областей может быть следующей:

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

Кликабельные области бывают трех различных форм [2]:

Rect – прямоугольная форма; координаты задаются в пикселях и записываются через запятую – x_1, y_1, x_2, y_2 . (x_1, y_1) – координаты левого верхнего угла изображения и (x_2, y_2) – правый нижний угол.

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

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

Рассмотрим эти области на примерах.

Прямоугольники. Возьмем изображение, представленное на рисунке 3.3.



Рисунок 3.3 – Прямоугольная кликабельная область

Картинка простая – всего лишь серый квадрат. Представьте, что черный прямоугольник, нарисованный на нем, – это изображение ка-

кой-либо кнопки, а серое – какой-нибудь сложный фон. Нужно создать навигационную карту, где область прямоугольника будет ссылкой.

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

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

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

Параметр `coords` определяет координаты (положение нашей геометрической формы). Число координат и порядок их значений зависят от выбранной нами формы (рисунок 3.4).

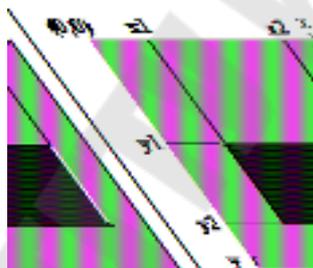


Рисунок 3.4 – Определение координат прямоугольной области

Отсчет ведется от левого верхнего угла картинке, считайте его началом координат (0;0). Если работаем с прямоугольной областью, то нужны координаты верхнего-левого и нижнего-правого углов области. Порядок записи координат для параметра `coords` следующий:

```
<area shape="rect" coords="x1,y1,x2,y2">
```

В нашем примере у прямоугольника координаты такие:

$$x1 = 83, y1 = 122, x2 = 177, y2 = 185.$$

Значит, код будет выглядеть следующим образом:

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

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

```
<map>
  <area href = "document1.html" shape = "rect"
    cords = "83, 122, 177, 185">
</map>
```

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

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

```
<map name="karta1">
  <area href = "document1.html" shape = "rect"
    cords = "83, 122, 177, 185">
</map>
```

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

```

...текст...
<map name="karta1">
  <area href = "document1.html" shape = "rect"
    cords = "83, 122, 177, 185">
</map>
```

Круги. Для создания круглой области нужны будут координаты ее центра (x, y) и длина радиуса R в пикселях (рисунок 3.5). Порядок записи следующий:

```
<area shape="circle" coords="x,y,R">.
```

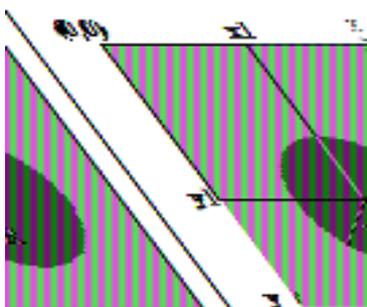


Рисунок 3.5 – Округлая кликабельная область

Многоугольники. Указывая точки (координаты углов), они как бы соединяются, и можно получить очень разнообразные фигуры (рисунок 3.6). Используя значение атрибута **shape poly**, можно делать самые разнообразные области – от скромного треугольника до шестиконечной звезды.

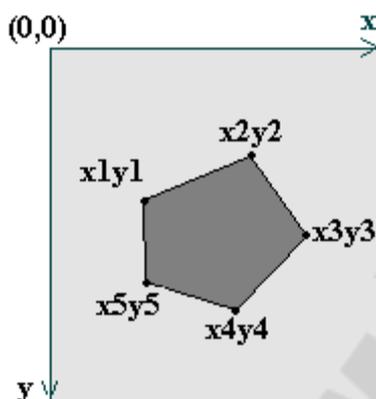


Рисунок 3.6 – Определение координат многоугольной области

Зададим тип области:

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

Координаты пишутся по следующему принципу:

```
<area shape = "poly" cords = "x1,y1,x2,y2,...,xN,yN">.
```

Расшифровывается это так: «координаты первого угла (x1,y1); координаты второго угла (x2,y2), еще множество углов и их координаты (...); координаты последнего угла (xN,yN)».

ГЛАВА 4 ВВЕДЕНИЕ В КАСКАДНЫЕ ТАБЛИЦЫ СТИЛЕЙ CSS. ОСНОВЫ РАБОТЫ С CSS

4.1 Основные цели и задачи CSS.

Способы добавления стилей на web-страницу

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

Стандарты листов стилей для web-страницы были разработаны консорциумом W3C в 1995–1996 гг. Слово «каскадные» означает, что листы стилей позволяют создавать иерархию стилевых свойств, согласно которой локальный стиль отменяет глобальный стиль.

CSS-код – это список написанных особым образом инструкций для браузера, который «говорит», как и где отображать элементы web-страницы. Другими словами, стили представляют собой набор параметров, управляющих видом и положением элементов web-страницы. Для наглядности рассмотрим примеры (рисунок 4.1, а и б).

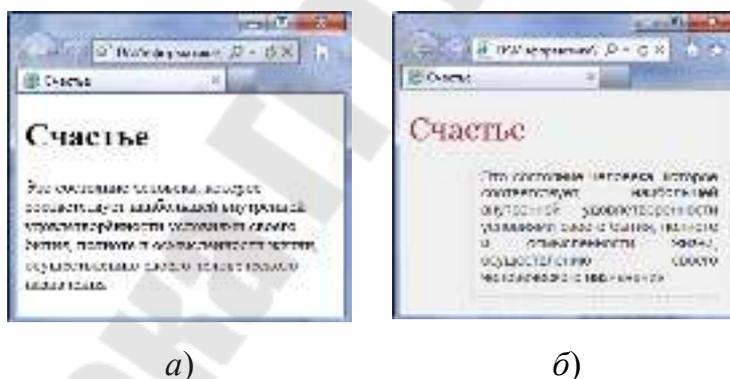


Рисунок 4.1 – Web-страницы на HTML и на CSS

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

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

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

В файле style.css как раз и описаны все параметры оформления таких тегов, как <body>, <h1> и <p>. Сами теги в коде HTML пишутся

как обычно. На файл со стилем можно ссылаться из любого места web-документа, что приводит в итоге к сокращению объема повторяющихся данных. Благодаря разделению кода и оформления, повышается гибкость управления видом документа и скорость работы над сайтом.

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

Связанные стили

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

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

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

Содержимое файла `style.css`, подключаемого посредством тега `<link>`, приведено ниже:

```
H1 {
  color: #001180;
  font-size: 200%;
  font-family: Arial, Verdana, sans-serif;
  text-align: center;
}
P {
  padding-left: 20px;
}
```

Как видно из кода файла `style.css`, он не хранит никаких данных, кроме синтаксиса CSS. В HTML-документе содержится только ссылка на файл со стилем, т. е. таким способом в полной мере реализуется принцип разделения кода и оформления сайта. Поэтому использование связанных стилей является наиболее универсальным и удобным методом добавления стиля на сайт.

Глобальные стили

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

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

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

Внутренние стили

Внутренний или встроенный стиль является расширением для одиночного тега, используемого на web-странице. Для определения стиля используется атрибут `style`, а его значением выступает набор стилевых правил:

```
<html>
  <head>
    <title>Внутренние стили</title>
  </head>
```

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

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

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

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

Отметим, что заголовок `h1` задается красным цветом размером 36 пикселей с помощью внутреннего стиля, а заголовок `h2` задается зеленым цветом через таблицу глобальных стилей.

4.2 Основные понятия и определения. Грамматика языка стилей

Как уже было отмечено ранее, стилевые правила записываются в своем формате, отличном от HTML. Основным понятием выступает *селектор* – это некоторое имя стиля, для которого добавляются пара-

метры форматирования. В качестве селектора выступают теги, классы и идентификаторы. Общий способ записи имеет следующий вид:

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

Стилевые свойства разделяются между собой точкой с запятой. Обратите внимание, что в конце последнего правила символ ; тоже ставится. Форма записи правил зависит от желания разработчика, поскольку CSS не чувствителен к регистру, переносу строк, пробелам и символам табуляции.

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

Правило 1. Для селектора допускается добавлять каждое стилевое свойство и его значение по отдельности:

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

Очевидно, что такая запись не очень удобна. Приходится повторять несколько раз один и тот же селектор, да и легко запутаться в их количестве. Поэтому пишите все свойства для каждого селектора вместе. Указанный набор записей в таком случае получит следующий вид:

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

Такая форма записи более наглядная и удобная в использовании.

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

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

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

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

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

```
div {
    width: 200px; /* Ширина блока */
}
```

Комментарии можно добавлять в любое место CSS-документа, а также писать текст комментария в несколько строк. Вложенные комментарии недопустимы.

4.3 Создание классов и их применение на web-странице

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

```
Тег.Имя класса {
    свойство1: значение;
    свойство2: значение;
    ...
    свойствоN: значение;
}
```

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

```

<html>
<head>
  <title>Классы</title>
  <style>
    P { /*Обычный абзац*/
      text-align:justify; /*Выравнивание текста по ширине*/
    }
    P.cite { /*Абзац с классом cite*/
      color: navy; /*Цвет текста*/
      margin-left:20px; /*Отступ слева*/
      border-left:2px dotted navy; /*Граница слева от текста*/
      padding-left:15px; /*Расстояние от линии до текста*/
    }
  </style>
</head>
<body>
  <p>Слово «каскадные» означает, что листы стилей позволяют ...</p>
  <p class="cite">CSS-код – это список инструкций...</p>
</body>
</html>

```

Результат примера показан на рисунке 4.2.

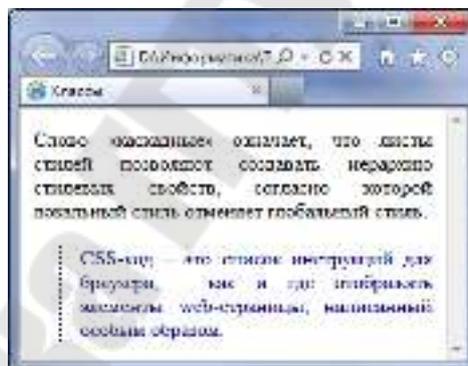


Рисунок 4.2 – Вид текста, оформленного с помощью стиливых классов

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

```

.Имя класса {
    свойство1: значение;
    ...
    свойствоN: значение;
}

```

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

4.4 Единицы измерения размеров

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

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

Числа. Значением может выступать целое число, содержащее цифры от 0 до 9, и десятичная дробь, в которой целая и десятичная часть разделяются точкой:

```
<style>
  p {
    font-weight: 700; /*Жирное начертание*/
    line-height: 1.5; /*Межстрочный интервал*/
  }
</style>
```

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

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

Проценты необязательно должны быть целым числом, допускается использовать десятичные дроби, например, значение 56,8%, но не всегда.

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

Относительные единицы обычно используют для работы с текстом, либо когда надо вычислить процентное соотношение между элементами. Перечислим основные относительные единицы:

- em* – размер шрифта текущего элемента;
- ex* – высота символа x;
- px* – пиксель;

% – процент.

В примере ниже показано применение пикселей и em для задания размера шрифта.

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

Результат данного примера показан ниже (рисунок 4.3).

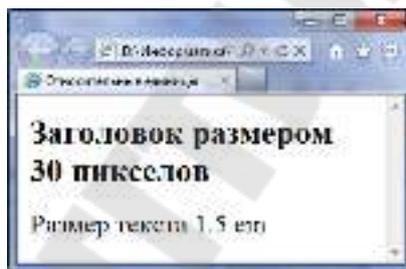


Рисунок 4.3 – Размер текста при различных единицах

Абсолютные единицы применяются реже, чем относительные, и обычно – при работе с текстом. Перечислим основные абсолютные единицы: in (дюйм); cm (сантиметр); mm (миллиметр); pt (пункт); ps (пика). Самой распространенной единицей является пункт, который используется для указания размера шрифта.

При установке размеров обязательно указывайте единицы измерения, например, `width: 30px`. В противном случае браузер не сможет показать желаемый результат, поскольку не понимает, какой размер требуется. Единицы не добавляются только при нулевом значении (`margin: 0`).

Цвет. Цвет в CSS можно задавать тремя способами: по шестнадцатеричному значению, по названию и в формате RGB (см. § 2.4).

```

body{
  background-color:#C0C0C0; /*Цвет фона web-страницы*/
}
h1{
  background-color:RGB(29,160,200); /*Цвет фона под заголовком*/
}
p{
  background-color:maroon; /*Цвет фона под текстом абзаца*/
}

```

Адреса. Адреса применяются для указания пути к файлу, например, для установки фоновой картинке на странице. Для этого применяется ключевое слово *url()*, внутри скобок пишется относительный или абсолютный адрес файла:

```

body{background:url('http://webimg.ru/images/156_1.png')
  no-repeat;}
div{background: url(images/warning.png) no-repeat;}

```

В селекторе *body* используется абсолютный адрес к графическому файлу, а в селекторе *div* – относительный.

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

4.5 Применение селекторов к элементам на web-странице

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

Синтаксис применения идентификатора следующий:

```

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

```

При описании идентификатора вначале указывается символ решетки (#), затем идет имя идентификатора. Оно должно начинаться с латинского символа и может содержать в себе символы дефиса (-) и подчеркивания (_). Использование русских букв в именах идентифи-

катора недопустимо. В отличие от классов идентификаторы должны быть уникальны, иными словами, встречаться в коде документа только один раз.

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

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

```
Тег#Имя идентификатора {  
    свойство1: значение;  
    ...  
    свойствоN: значение;  
}
```

Вначале указывается имя тега, затем без пробелов – символ решетки и название идентификатора.

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

Рассмотрим несколько типичных вариантов применения таких селекторов.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Одно из нескольких значений атрибута. Некоторые значения атрибутов могут перечисляться через пробел, например, имена клас-

сов. Чтобы задать стиль при наличии в списке требуемого значения, применяется следующий синтаксис:

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

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

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

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

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

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

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

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

Синтаксис применения псевдоклассов следующий:

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

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

Условно все псевдоклассы делятся на три группы:

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

Рассмотрим подробнее все псевдоклассы.

Псевдоклассы, определяющие состояние элементов

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

:active

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

:link

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

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

:focus

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

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

:hover

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

:visited

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

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

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

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

Псевдоклассы, имеющие отношение к дереву документа

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

:first-child

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

```
<html>
<head>
  <title>Псевдоклассы</title>
  <style type="text/css">
    b:first-child { color: red; } /*Красный цвет текста*/
  </style>
```

```

</head>
<body>
  <p><b>Динамический HTML</b> построен на объектной
  <b>модели</b>, которая расширяет традиционный <b>статический</b>
  HTML-документ.</p>
  <p><b>Таблицы стилей CSS</b> документа могут быть
  <b>изменены</b> в любое время.</p>
</body>
</html>

```

Результат примера показан ниже (рисунок 4.4).

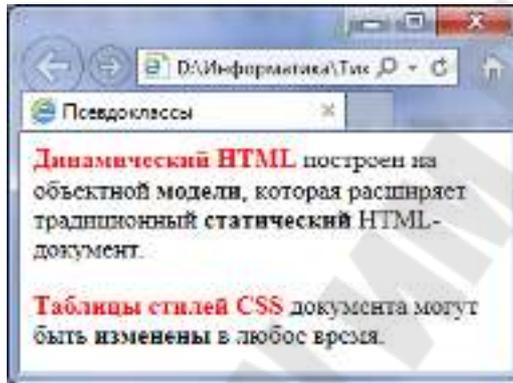


Рисунок 4.4 – Использование псевдокласса `first-child`

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

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

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

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

Синтаксис использования псевдоэлементов следующий:

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

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

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

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

Далее перечислены все псевдоэлементы, их описание и свойства.

:after

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

```
<style>
  p.new:after {
    content: "-ДА!"; /*Добавляем после текста абзаца*/ }
</style>
```

Результат примера показан на рисунке 4.5.

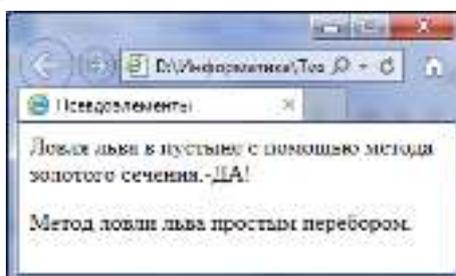


Рисунок 4.5 – Добавление текста к абзацу с помощью `:after`

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

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

:before

По своему действию `:before` аналогичен псевдоэлементу `:after`, но вставляет контент до элемента. Таким образом, например, можно добавить маркеры своего типа к элементам списка посредством скрывания стандартных маркеров и применения псевдоэлемента `:before`. В данном случае псевдоэлемент `:before` устанавливается для селектора `li`, определяющего элементы списка. Добавление желаемых символов происходит путем задания значения свойства `content`. Обратите внимание, что в качестве аргумента необязательно выступает текст, могут применяться также символы юникода.

:first-letter

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

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

:first-line

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

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

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

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

```
teg1 teg2 { ... }
```

В этом случае стиль будет применяться к *teg2*, когда он размещается внутри *teg1*, как показано ниже:

```
<teg1>  
  <teg2> ... </teg2>  
</teg1>
```

Использование контекстных селекторов продемонстрировано ниже:

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

В данном примере показано обычное применение тега `` и этого же тега, когда он вложен внутрь абзаца `<p>`. При этом меняется цвет и шрифт текста, как показано на рисунке 4.6.

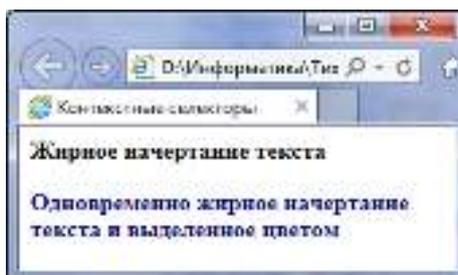


Рисунок 4.6 – Оформление текста в зависимости от вложенности тегов

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

Дочерним называется элемент, который непосредственно располагается внутри родительского элемента.

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

В данном примере применяется несколько контейнеров, которые в коде располагаются один в другом. Нагляднее это видно на дереве элементов; так называется структура отношений тегов документа между собой (рисунок 4.7).

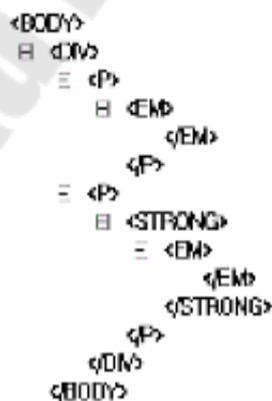


Рисунок 4.7 – Дерево элементов

На рисунке 4.7 в удобном виде представлена вложенность элементов и их иерархия. Здесь дочерним элементом по отношению к тегу <div> выступает тег <p>. Вместе с тем тег не является дочерним для тега <div>, поскольку он расположен в контейнере <p>.

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

Селектор 1 > Селектор 2 {Описание правил стиля}

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

Если снова обратиться к примеру 4.11, то стиль вида `p>em{color:red}` будет установлен для первого абзаца документа, поскольку тег `` находится внутри контейнера `<p>`, и не даст никакого результата для второго абзаца. А все из-за того, что тег `` во втором абзаце расположен в контейнере ``, поэтому нарушается условие вложенности.

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

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

Результат данного примера показан на рисунке 4.8.

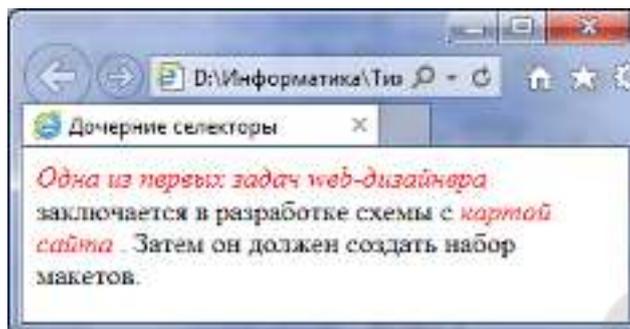


Рисунок 4.8 – Цвет текста, заданный с помощью дочернего селектора

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

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

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

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

```
<p>Наша мама <b>мыла </b> большую раму.</p>
```

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

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

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

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

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

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

Селектор 1 + Селектор 2 {Описание правил стиля}

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

В примере ниже показана структура взаимодействия тегов между собой.

```
<html>
  <head>
    <title>Соседние селекторы</title>
    <style>
      b + i {
        color: red; /*Красный цвет текста*/
      }
    </style>
  </head>
  <body>
    <p>WEB-<b>дизайнер</b>должен создать набор <i> макетов </i>
    </p>
    <p>Эти макеты должны отображать <i>содержание</i> и <i> на-
   avigационные элементы </i> каждой страницы</p>
  </body>
</html>
```

Результат примера показан на рисунке 4.9.

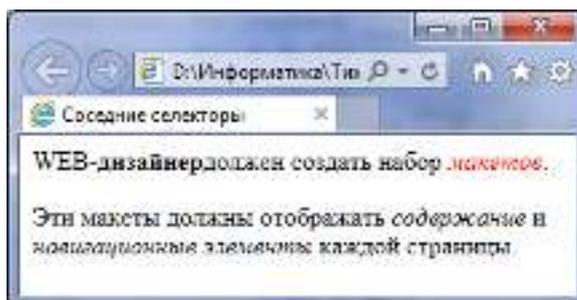


Рисунок 4.9 – Выделение текста цветом при помощи соседних селекторов

В данном примере происходит изменение цвета текста для содержимого контейнера `<i>`, когда он располагается сразу после контейнера ``. В первом абзаце такая ситуация реализована, поэтому слово «макетов» в браузере отображается красным цветом. Во втором абзаце, хотя и присутствует тег `<i>`, но по соседству никакого тега `` нет, так что стиль к этому контейнеру не применяется.

Иногда требуется установить одновременно один стиль для всех элементов web-страницы, например, задать шрифт или начертание текста. В этом случае поможет универсальный селектор, который соответствует любому элементу web-страницы.

Для обозначения *универсального селектора* применяется символ звездочки (*) и в общем случае синтаксис будет следующий:

* {Описание правил стиля}

В некоторых случаях указывать универсальный селектор необязательно. Так, например, записи `*.class` и `.class` являются идентичными по своему результату.

4.6 Декоративные возможности CSS: форматирование текста

Приступим к изучению правил CSS. Начнем с правил, которые относятся к *шрифтам*.

Font-family: семейства шрифта | тип шрифта;

Задаёт описание шрифта. Аналогом в HTML является атрибут `face` элемента `font`.

Например, `font-family: Arial, Geneva, Helvetica, sans-serif.`

Если название шрифта состоит из нескольких слов, разделенных пробелами, например, Times New Roman, то в отличие от HTML необходимо это название заключить в кавычки.

Font-size: величина | %;

Задаёт величину шрифта, которую можно задавать в em, ex и px. Другое значение – это проценты (%). Вы можете написать 2em, что будет означать увеличение шрифта в два раза. Все правила, которые начинаются с приставки font, наследуются. Это означает, что все элементы на web-странице уже получают какой-то размер шрифта, который можно изменить, прописав, например, `h1{font-size:2em;}`. Последнее правило, следует понимать так: элементу h1 «пришел» размер шрифта такой же как и всем, а Вы его увеличили в 2 раза. Отметим, что правила `font-size:2em;` и `font-size:200%;` работают одинаково. Проценты берутся от того размера, который «пришел» к элементу.

Еще есть 7 размеров, которые коррелируют с атрибутом size:

- абсолютные величины: xx-small, x-small, small, medium, large, x-large, xx-large;
- относительные величины: larger, smaller.

Следующее правило задает *начертание шрифта по жирности*.

Font-width: normal | bold | bolder | lighter | число от 100 и до 900;

Отметим, что 400 = normal, 700 = bold. На практике пока работают только первые два значения font-width.

Font-style: normal | italic | oblique;

Это правило задает *наклон текста* нормальный, курсив или наклон прямого текста. Поясним, что в некоторых шрифтах нет курсивов и поэтому для них возможен только наклонный вариант. Например, шрифт Tahoma имеет только наклон. Это означает, что если браузер не находит наклонного начертания у шрифта, то он его сам «наклоняет».

Следующий способ изменения символов называется *капителью*.

Font-variant: normal | small-caps;

Font-variant: `small-caps;` – означает, что строчные буквы свое начертание меняют на заглавные, а размер остается такой же. Вид для русского текста становится неприглядным. Поэтому такое правило встречается очень редко.

Line-height: `normal` | величина | %;

Это правило задает *высоту линии шрифта*.

Есть пространство, симметрично откладываемое вверх и вниз, которое служит для того, чтобы строки текста между собой не слиплись. Это пространство называется размер линии (`line-height`). Соотношение размера шрифта и высоты линии для каждого шрифта уникально. Для таких шрифтов, как `Helvetica`, `Arial` соотношение шрифта используется 120%. Высота линии на 20% выше, чем высота шрифта. Для `Times New Roman` – 125%. Для того чтобы вернуть к исходному значению, мы применяем правило `line-height:normal;`, а правило `line-height:200%;` задает соотношение линии и высоты шрифта 200%, тем самым делает строки редкими.

Можно использовать все 6 правил в одном:

Font: `font-style font-variant font-weight font-size font-family`

Простые правила перечисляются через запятую и порядок их важен. Среди этих пяти правил два последних (`font-size` и `font-family`) обязательны! Без них написать правило `font` нельзя. Если пропустить 3 первых правила, то браузер поставит значения по умолчанию (`normal`). Правило `line-height` задается через слеш (/).

Например, `font: bold 10/12px Arial;`

Рассмотрим правила, которые относятся к *тексту*.

Следующее правило работает для блочных элементов и служит для *выравнивания контента* внутри.

Text-align: `left` | `right` | `center` | `justify;`

Чтобы выделить текст *горизонтальной линией*, следует использовать правило:

Text-decoration: `none` | `overline` (|) `underline` (|) `line-through;`

Значения правила `text-decoration`:

`none` – не выделять текст линией;
`overline` – рисовать горизонтальную линию над текстом;
`underline` – рисовать горизонтальную линию под текстом;
`line-through` – перечеркнуть текст;

Если, например, написать `a{text-decoration:none;}`, то это правило уберет подчеркивание у ссылки.

Позволяет задавать *отступ* красной строки следующее правило, которое работает для блочных элементов:

Text-indent: величина | %;

Например, правило `text-indent:50%;` задает красную строку с середины поля. Можно задавать и отрицательные значения.

Когда Вы получаете макет сайта, в котором используется заглавное начертание всех заголовков, то не торопитесь включать `serif` и набирать текст заглавными буквами. Используйте обычный набор, потому что есть возможности трансформации этого содержания. Правило, которое позволяет выполнить *трансформацию* текста:

Text-transform: none | capitalize | uppercase | lowercase;

`uppercase;` – любой текст делает заглавными буквами.

`lowercase;` – любой текст делает строчными буквами.

`capitalize;` – каждое слово начинается с большой буквы.

Рассмотрим *промежутки* между символами и словами, соответственно:

letter-spacing: normal | величина;

word-spacing: normal | величина;

`normal` – это обычное состояние.

`величина` – это значение в `em`, `ex` и `px`. Эти значения могут быть положительными (буквы или слова будут реже) и отрицательными (символы или слова будут сдавливаться).

Вертикальное выравнивание строчных элементов:

Vertical-align: baseline | sub | super | top | text-top | middle | bottom | text-bottom | величина | %;

`baseline` – это выравнивание текста по базовой линии (по умолчанию);

`sub` – надиндекс;

`super` – подиндекс;

`text-top` – выравнивание по верхней границе текстовой строки;

`text-bottom` – выравнивание по нижней границе текстовой строки;

`middle` – выравнивание по середине текстовой строки;

`top` – для ячеек таблицы – выравнивание по верхней границе;

`bottom` – для ячеек таблицы – выравнивание по нижней границе;

В ячейках таблицы `vertical-align` означает выравнивание контента по границам ячейки.

величина – это значения в `px`, `em` и `ex`. Может принимать как положительные, так и отрицательные значения.

Данное правило задает *пробельные символы*.

White-space: `normal` | `pre` | `nowrap`;

`normal` – в строчном контенте браузер пробельные символы сокращает и делает по ним перенос. В блочном контенте он их просто игнорирует;

`pre` – это имитация элемента `pre` в HTML. Напомним, что пробельные символы не сокращаются, а остаются как есть; никакого автоматического переноса не происходит;

`nowrap` – пробельные символы сокращаются, но для автоматического переноса не используются.

Рассмотрим правила, позволяющие задать *фоновые цвета*.

Задаёт фоновый *цвет*:

Background-color: `цвет` | `transparent`;

По умолчанию все элементы на странице прозрачные (`transparent`).

Позволяет задать фоновую *картинку*:

Background-image: `none` | `url`;

По умолчанию – `none` – нет фоновой картинки. Если необходимо задать картинку, то надо прописать следующее правило: `background-image:url (picture.jpg);`.

ГЛАВА 5. СПИСКИ И ТАБЛИЦЫ

5.1 Структурирование информации на web-странице при помощи списков. Типы списков

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

- неупорядоченную информацию;
- упорядоченную информацию;
- определения.

Неупорядоченные списки создаются с помощью элемента `...`, внутри которого может находиться не менее одного элемента `...`:

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

Для того чтобы управлять внешним видом маркера, необходимо применить атрибут `type` со значением `"circle"` или `"square"`.

Например, `<ul type="circle">` – маркер в виде окружности;
`<li type="square">` – маркер в виде квадрата.

Упорядоченный список, создаваемый с помощью элемента `...`, может содержать информацию, в которой важен порядок, например, рецепт:

- а. Тщательно смешать сухие ингредиенты.
- б. Влить жидкость.
- в. Смешивать 10 минут.
- г. Выпекать в течение часа при температуре 300 градусов.

Реализация в HTML упорядоченного списка выглядит так:

```
<ol type = "a">
<li> Тщательно смешать сухие ингредиенты.</li>
<li> Влить жидкость.</li>
<li> Смешивать 10 минут.</li>
<li> Выпекать в течение часа при температуре 300 градусов. </li>
</ol>
```

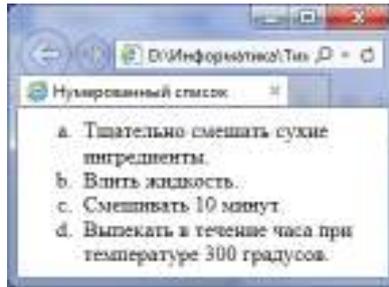


Рисунок 5.1 – Упорядоченный список

Для управления внешним видом нумерованного списка необходимо применить атрибут `type` со значениями, приведенными в таблице 5.1.

Таблица 5.1 – Типы нумераций упорядоченного списка

Начальный тег	Вид номера на экране
<code></code>	Нумерация выполняется арабскими цифрами (1, 2, 3, ...)
<code><ol type="1"></code>	Нумерация выполняется арабскими цифрами (1, 2, 3, ...)
<code><ol type="A"></code>	Нумерация выполняется прописными буквами (A, B, C, ...)
<code><ol type="a"></code>	Нумерация выполняется строчными буквами (a, b, c, ...)
<code><ol type="I"></code>	Нумерация выполняется большими римскими цифрами (I, II, III, ...)
<code><ol type="i"></code>	Нумерация выполняется малыми римскими цифрами (i, ii, iii, ...)

Если необходимо начать нумерацию не с первого номера, а, например, с 5, то используйте атрибут `start` со значением "5".

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

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

Списки могут быть *вложенными* (смешанными).

5.2 CSS свойства для списков

В этом разделе рассмотрим основные свойства CSS, которые применяются для списков.

Универсальное свойство, позволяющее одновременно задать стиль маркера, его положение, а также изображение, которое будет использоваться в качестве маркера.

```
list-style: list-style-type || list-style-position ||  
list-style-image | inherit
```

Любые комбинации трех значений, определяющих стиль маркеров, разделяются между собой пробелом. Комбинации значений должны следовать в указанном порядке: вначале идет тип маркера, затем положение и картинка. Ни одно значение не является обязательным, поэтому неиспользуемые можно опустить.

Каждое свойство правила `list-style` имеет свое значение. Рассмотрим каждое отдельно.

list-style-type – изменяет вид маркера для каждого элемента списка. Это свойство используется только в случае, когда значение `list-style-image` установлено как `none`. Маркеры различаются для маркированного списка (тег ``) и нумерованного (тег ``).

Синтаксис записи этого правила следующий:

```
list-style-type: circle | disc | square | armenian | decimal  
| decimal-leading-zero | georgian | lower-alpha | lower-  
greek | lower-latin | lower-roman | upper-alpha | upper-latin  
| upper-roman | none | inherit
```

Значения зависят от того, к какому типу списка они применяются: маркированному или нумерованному.

Маркированный список

`circle` – маркер в виде кружка.

`disc` – маркер в виде точки.

`square` – маркер в виде квадрата.

Нумерованный список

`armenian` – традиционная армянская нумерация.

`decimal` – арабские числа (1, 2, 3, 4,...).

`decimal-leading-zero` – арабские числа с нулем впереди для цифр меньше десяти (01, 02, 03,...).

`georgian` – традиционная грузинская нумерация.

`lower-alpha` – строчные латинские буквы (a, b, c, d,...).
`lower-greek` – строчные греческие буквы (α , β , γ , δ ,...).
`lower-latin` – это значение аналогично `lower-alpha`.
`lower-roman` – римские числа в нижнем регистре (i, ii, iii ...).
`upper-alpha` – заглавные латинские буквы (A, B, C, D,...).
`upper-latin` – это значение аналогично `upper-alpha`.
`upper-roman` – римские числа в верхнем регистре (I, II, III ...).
`none` – отменяет маркеры для списка.

`inherit` – наследует значение родителя. Internet Explorer до версии 7.0 включительно не поддерживает значение `inherit`.

`list-style-position` – определяет, как будет размещаться маркер относительно текста. Имеется два значения: `outside` – маркер вынесен за границу элемента списка и `inside` – маркер обтекается текстом. Синтаксис следующий:

```
list-style-position: inside | outside
```

`list-style-image` – устанавливает адрес изображения, которое служит в качестве маркера списка. Это свойство наследуется, поэтому для отдельных элементов списка для восстановления маркера используется значение `none`.

```
list-style-image: none | url('путь к файлу') | inherit
```

`none` – отменяет изображение в качестве маркера для родительского элемента.

`url` – относительный или абсолютный путь к графическому файлу. Значение можно указывать в одинарных, двойных кавычках или без них.

`inherit` – наследует значение родителя.

5.3 Таблица и ее элементы

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

В устройстве таблицы легче разобраться на примере (рисунок 5.2).



Рисунок 5.2 – Простая таблица

В данном примере на рисунке 5.2 создана таблица с фиксированной шириной (`width="200"`), но лучше использовать проценты, так как в этом случае размер таблицы будет изменяться в зависимости от размера окна браузера.

Таблица начинается открывающимся тегом `<table>` и завершается закрывающимся `</table>`. В примере на рисунке 5.2 – таблица из двух строк и двух столбцов, вместе образующих 4 ячейки. Границы таблицы заданы `border = "2"`.

Рассмотрим создание простой таблицы поэтапно.

Сначала зададим две строки таблицы.

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

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

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

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

5.4 Правила задания атрибутов для таблицы и ее ячеек. Объединение ячеек

Тег `<table>` может включать следующие атрибуты:

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

`border` – устанавливает толщину рамки. По умолчанию таблица рисуется без рамки.

`bordercolor` – устанавливает цвет окантовки.

`bgcolor` – устанавливает цвет фона для всей таблицы.

`background` – заполняет фон таблицы изображением.

`cellspacing` – определяет расстояние между рамками ячеек таблицы в пикселях.

`cellpadding` – определяет расстояние в пикселях между рамкой ячейки и текстом.

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

`frame` – управляет внешней окантовкой таблицы, может принимать следующие значения:

`void` – окантовки нет (значение по умолчанию).

`above` – только граница сверху.

`below` – только граница снизу.

`hsides` – границы сверху и снизу.

`vsides` – только границы слева и справа.

`lhs` – только левая граница.

`rhs` – только правая граница.

`box` – рисуются все четыре стороны.

`border` – также все четыре стороны.

`rules` – управляет линиями, разделяющими ячейки таблицы. Возможные значения атрибута `rules`:

`none` – нет линий (значение по умолчанию).

`groups` – линии будут только между группами рядов.

`rows` – только между рядами.

`cols` – только между колонками.

`all` – между всеми рядами и колонками.

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

`top` – значение по умолчанию, заголовок над таблицей по центру.

`left` – заголовок над таблицей слева.

`right` – заголовок над таблицей справа.

`bottom` – заголовок под таблицей по центру.

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

`align` – устанавливает горизонтальное выравнивание текста в ячейках строки. Может принимать значения:

`left` – выравнивание влево;

`center` – выравнивание по центру;

`right` – выравнивание вправо.

`valign` – устанавливает вертикальное выравнивание текста в ячейках строки. Принимает допустимые значения:

`top` – выравнивание по верхнему краю;

`center` – выравнивание по центру (по умолчанию);

`bottom` – выравнивание по нижнему краю.

`bgcolor` – устанавливает цвет фона строки или ячейки.

`background` – заполняет фон строки или ячейки изображением.

Следующие атрибуты могут применяться только для ячеек.

`width` – определяет ширину ячейки в пикселях.

`height` – определяет высоту ячейки в пикселях.

`nowrap` – присутствие этого атрибута показывает, что текст должен размещаться в одну строку.

`background` – заполняет фон ячейки изображением.

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

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

Теги, устанавливающие шрифт (``, `<i>`, ``, ``), необходимо повторять для каждой ячейки.

Подробнее остановимся на атрибутах `colspan` и `rowspan`.

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

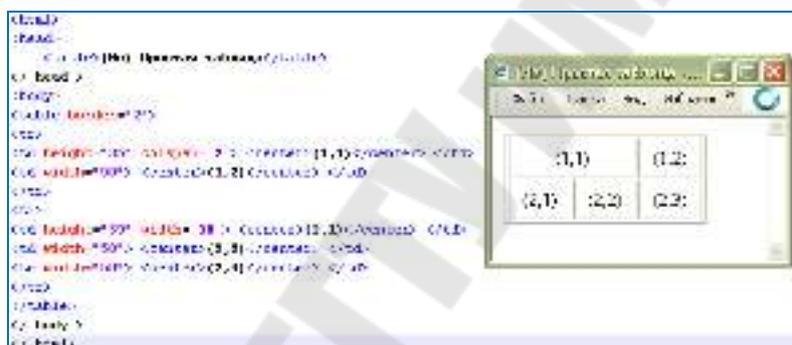


Рисунок 5.3 – Использование атрибута `colspan`

В таблице на рисунке 5.3 использован параметр `colspan = 2`, прописанный для ячейки (1,1), где первая цифра – это номер ряда, а вторая – номер столбца (т. е. (1,2) – первый ряд, второй столбец и т. д.).

Обратите внимание, на то, что параметр `width` для ячейки (1,1) в примере на рисунке 5.4 не указан. Если необходимо задать этот атрибут, то в данном примере для ячейки (1,1) его надо было бы прописать равным 100 пикселям, так как все-таки ячейка (1,1) длиннее других в два раза.

И второе – на что следует обратить внимание, в рассмотренном примере на рисунке 5.3 нет ячейки (1,3), т. е. в первом ряду всего лишь две ячейки, так как ячейка (1,1) равна сама по себе двум ячейкам по длине (что мы и указали параметром `colspan`). Если Вы прописали ячейку (1,3), тогда у Вас получилась бы такая «таблица», как на рисунке 5.4.

(1,1)	(1,2)	(1,3)
(2,1)	(2,2)	(2,3)

Рисунок 5.4 – Ошибочное построение таблицы

Теперь рассмотрим параметр `rowspan`. Принцип действия здесь тот же. Попробуйте самостоятельно прописать код для таблицы, что представлена на рисунке 5.5.

(1,1)	(1,2)	(1,3)
(2,1)	(2,2)	

Рисунок 5.5 – Использование атрибута `rowspan`

5.5 Верстка при помощи таблиц. Вложенные таблицы

Изначально, назначение тега `<table>` сводилось к размещению на странице структурированных данных. По мере усложнения сайтов этот тег превратился в инструмент для разметки каркаса web-страниц. Способ верстки, использующий тег `<table>`, называется *табличным*. При помощи этого тега можно быстро построить достаточно сложный каркас страницы.

Построение каркаса страницы (верстка) осуществляется путем вложения таблиц друг в друга. Сначала создается основная таблица, затем в ее ячейках создаются другие таблицы, в которых, в свою очередь, также можно создать таблицы. Приведем пример создания каркаса страницы (для краткости приведем только содержание раздела `<body>`).

```

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

```

```

        </tr>
      </table>
    </td>
  </tr>
  <tr>
    <td>Нижний блок</td>
  </tr>
</table>

```

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

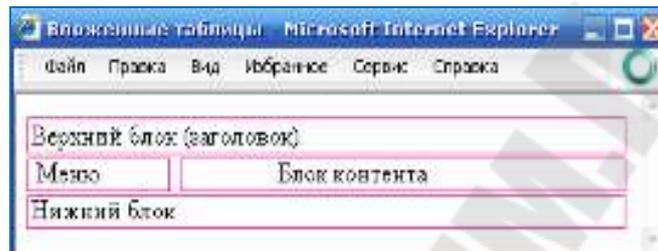


Рисунок 5.6 – Пример табличной верстки

Рассмотрим еще один вид верстки. *Комбинированной* называется верстка с одновременным использованием тегов-контейнеров `<div>` и конструкции `<table>`.

```

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

```

В итоге будет достигнут тот же результат (рисунок 5.6), но с меньшими затратами.

5.6 Таблицы и стили

Воспользовавшись мощью стилей, можно весьма расширить средства по оформлению таблиц, удачно вписать их в дизайн сайта и нагляднее представить табличные данные.

Далее речь пойдет об оформлении таблиц с помощью стилей.

Цвет фона ячеек таблицы устанавливается через свойство `background`, которое применяется к селектору `table`. При этом следует помнить о правилах использования стилей, в частности, наследовании свойств элементов. Если одновременно с `table` задать цвет у селектора `td` или `th`, то он и будет установлен в качестве фона.

```
<html>
  <head>
    <title>Цвет фона</title>
    <style type="text/css">
      table {
        background: maroon; /* Цвет фона таблицы */
        color: white; /* Цвет текста */
      }
      td {
        background: navy; /* Цвет фона ячеек */
      }
    </style>
  </head>
  <body>
    <table cellpadding="4" cellspacing="1">
      <tr><th>Заголовок 1</th><th>Заголовок 2</th></tr>
      <tr><td>Ячейка 3</td><td>Ячейка 4</td></tr>
    </table>
  </body>
</html>
```

В данном случае получим синий цвет фона у ячеек (тег `<td>`) и красный – у заголовка (тег `<th>`). Это связано с тем, что стиль для селектора `th` не определен, поэтому «просвечивается» фон родителя, в данном случае – селектора `table`. А цвет для селектора `td` указан явно, вот ячейки и «заливаются» синим цветом. То же самое происходит и с цветом текста. Для всех элементов таблицы он установлен белым. Результат данного случая показан на рисунке 5.7.

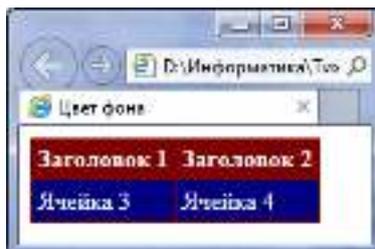


Рисунок 5.7 – Изменение цвета фона

Поля внутри ячеек – это расстояние между краем содержимого ячейки и ее границей. Обычно для этой цели применяется атрибут **cellpadding** тега **<table>**. Он определяет значение поля в пикселях со всех сторон ячейки. Допускается использовать стилевое свойство **padding**, добавляя его к селектору **td**, как показано ниже:

```
td, th {  
    padding: 5px; /* Поля вокруг текста */  
}
```

В данном примере с помощью группирования селектором поля установлены одновременно для селектора **td** и **th**.

Границы и рамки между ячейками можно установить несколькими методами, при этом рассмотрим два из них, которые непосредственно связаны со стилями.

Известно, что атрибут **cellspacing** тега **<table>** задает расстояние между ячейками таблицы. Если используется разный цвет фона таблицы и ячеек, то между ячейками возникнет сетка линий, цвет которых совпадает с цветом таблицы, а толщина равна значению атрибута **cellspacing** в пикселях.

Заметим, что это не совсем удобный способ создания границ, поскольку он имеет ограниченную область применения. Так можно получить только одноцветную сетку, а не вертикальные или горизонтальные линии в нужных местах.

Стилевое свойство **border** одновременно задает цвет границы, ее стиль и толщину вокруг элемента. Когда требуется создать отдельные линии на разных сторонах, лучше использовать его производные – **border-left**, **border-right**, **border-top** и **border-bottom**; они, соответственно, определяют границу слева, справа, сверху и снизу.

Применяя свойство **border** к селектору **table**, мы добавляем рамку вокруг таблицы в целом, а к селектору **td** или **th** – рамку вокруг ячеек, как показано ниже:

```

table {
    border: 5px double #000; /*Рамка вокруг таблицы*/
}
td, th {
    padding: 5px; /*Поля вокруг текста*/
    border: 1px solid #fff; /*Рамка вокруг ячеек*/
}

```

В данном случае используется двойная рамка черного цвета вокруг самой таблицы и сплошная рамка белого цвета вокруг каждой ячейки. Результат будет как на рисунке 5.8.

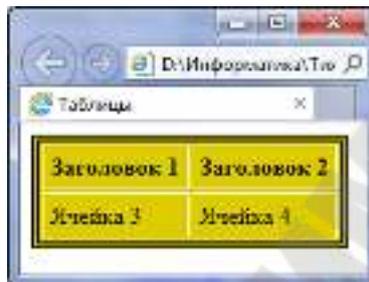


Рисунок 5.8 – Граница вокруг таблицы и ячеек

Обратите внимание, что в месте состыковки ячеек образуются двойные линии. Они получаются опять же за счет действия атрибута `cellspacing` тега `<table>`. Хотя в коде примера этот атрибут нигде не фигурирует, браузер использует его по умолчанию. Если задать `<table cellspacing="0">`, то получим не двойные, а одинарные линии, но удвоенной толщины. Для изменения указанной особенности применяется стилевое свойство `border-collapse` со значением `collapse`, которое добавляется к селектору `table`:

```

table {
    border-collapse: collapse; /*Убираем двойные границы
    между ячейками*/
}

```

При добавлении `border-collapse: collapse` значение атрибута `cellspacing` тега `<table>` игнорируется.

Выравнивание содержимого ячеек по умолчанию происходит по левому краю. Исключением из этого правила служит тег `<th>`, он определяет заголовок, в котором выравнивание происходит по центру. Чтобы изменить способ выравнивания, применяется стилевое свойство `text-align` со значениями `right`, `left`, `justify`, `center` и `inherit`.

Выравнивание по вертикали в ячейке всегда происходит по ее центру, если это не оговорено особо. Это не всегда удобно, особенно для таблиц, у которых содержимое ячеек различается по высоте. В таком случае выравнивание устанавливают по верхнему краю ячейки с помощью свойства **vertical-align**.

Правило `table-layout` определяет, как браузер должен вычислять ширину ячеек таблицы, основываясь на ее содержимом.

Применяется к тегу `<table>` или к элементу, у которого значение `display` установлено как `table` или `inline-table`.

Формат записи следующий:

```
table-layout: auto | fixed | inherit
```

`auto` - браузер загружает всю таблицу, анализирует ее для определения размеров ячеек и только после этого отображает.

`fixed` - ширина колонок в этом случае определяется либо с помощью тега `<col>`, либо вычисляется на основе первой строки. Если данные о форматировании первой строки таблицы по каким-либо причинам получить невозможно, в этом случае таблица делится на колонки равной ширины. При использовании этого значения содержимое, которое не помещается в ячейку указанной ширины, будет «обрезано», либо наложено поверх ячейки. Это зависит от используемого браузера, но в любом случае ширина ячейки меняться не будет. Для корректной работы этого значения обязательно должна быть задана ширина таблицы.

`inherit` - наследует значение родителя.

ГЛАВА 6 ФРЕЙМЫ И ФОРМЫ

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

Существуют различные приложения, необязательно web-страницы, где окно разделено на несколько областей, в каждую из которых загружен какой-то документ. Причем эти области ведут себя независимо.

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

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

Для создания фрейма используется тег `<frameset>`, который заменяет тег `<body>` в документе и применяется для разделения экрана на области. Внутри данного тега находятся теги `<frame>`, которые указывают на HTML-документ, предназначенный для загрузки в область (рисунок 6.1).

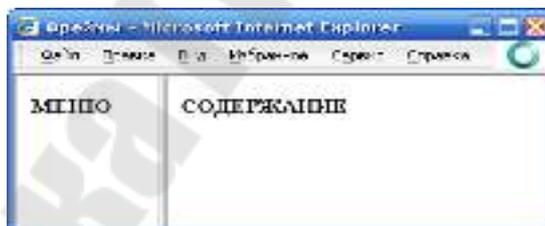


Рисунок 6.1 – Пример разделения окна браузера на два фрейма

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

Рассмотрим этапы создания фреймов на основе страницы, продемонстрированной на рисунке 6.1. Понадобится три файла: `index.html` – определяет структуру документа; `menu.html` – загружа-

ется в левый фрейм и content.html – загружается в правый фрейм. Из них только index.html отличается по структуре своего кода от других файлов (пример 6.1).

Пример 6.1

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

В примере 6.1 окно браузера разбивается на две колонки с помощью атрибута cols, левая колонка занимает 100 пикселей, а правая – оставшееся пространство, заданное символом звездочки. Ширину или высоту фреймов можно также задавать в процентном отношении наподобие таблиц.

В теге <frame> задается имя HTML-файла, загружаемого в указанную область с помощью атрибута src. В левое окно будет загружен файл, названный menu.html (пример 6.2), а в правое – content.html (пример 6.3). Каждому фрейму желательно задать его уникальное имя, чтобы документы можно было загружать в указанное окно с помощью атрибута name.

Пример 6.2

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

Пример 6.3

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

```
<p>СОДЕРЖАНИЕ</p>
</body>
</html>
```

Кроме обычных фреймов, HTML допускает использование так называемого *плавающего фрейма*, который можно создать с помощью тега `<iframe>`, он имеет обязательный атрибут `src`, указывающий на загружаемый во фрейм документ. Формат этого тега выглядит очень просто:

```
<iframe width="ширина" height="высота" src="адрес фрейма">
</iframe>
```

Пример 6.4

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

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

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

```
<html>
<head>
  <title>Плавающий фрейм</title>
</head>
<body>
  <p>
    <a href="p1.html" target="color">RGB</a>
    <a href="p2.html" target="color">CMYK</a>
    <a href="p3.html" target="color">HSB</a>
  </p>
```

```

    <p>
      <iframe src="face.html" name="color" width="100%"
height="200"></iframe>
    </p>
  </body>
</html>

```

В данном примере добавлено несколько ссылок, они открываются во фрейме с именем `color`. Результат представлен на рисунке 6.2.

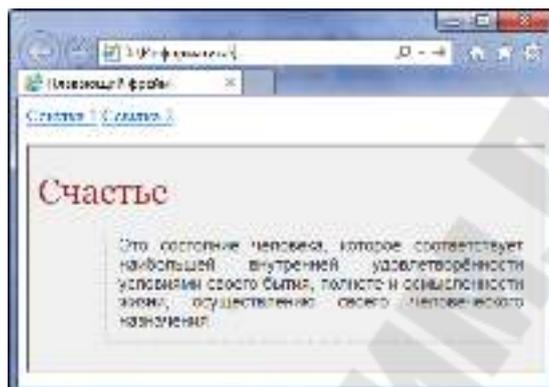


Рисунок 6.2 – Плавающий фрейм

6.2 Форма и ее элементы. Методы отправки информации из полей формы

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

Формы образуются с помощью элемента `<form>...</form>`. Этот элемент блочный, внутри него можно размещать любой контент, кроме других форм.

Формат задания формы следующий:

```

<form name = "имя_формы" action = "URL" enctype =
"тип_кодировки" method = "метод_передачи_данных" target = "значение">
  Содержание формы
</form>

```

В качестве параметра атрибута `action` в кавычках указывается строка вызова скрипта, который использует данная форма (при нажатии кнопки), например, "`http://www.gstu.by/имя_сценария.php`". Если в качестве значения атрибута `action` указать адрес электронной почты, например, `action="mailto:gstu@gstu.by"`, тогда браузер автоматически отправит результаты, введенные в форму, по указанному адресу. Для корректной интерпретации данных используется параметр `enctype="text/plain"`.

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

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

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

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

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

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

```
<input type = "тип_элемента" name = "имя" value = "стро-
ка" checked size = "целое_число" maxlength = "целое_число"
align = "значение" src = "URL" tabindex = "значение">
```

Атрибуты тега <input>:

type – задает тип элемента формы;

name – задает уникальное имя для каждого элемента формы;

value – указывает первоначальное значение текущего поля;

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

size – определяет размер текстового поля в символах;

maxlength – определяет максимально возможную длину текстового поля в символах для полей ввода текста;

align – определяет положение элементов формы на web-странице;

src – используется совместно с атрибутом type = image и задает URL нужного изображения;

tabindex – позволяет установить порядок перемещения фокуса по элементам формы при нажатии клавиши Tab.

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

Формат записи:

```
<input type="text" name="имя_поля" value="начальный текст, со-
держательный в поле" size="ширина поля" maxlength="максимальное количество
вводимых символов">
```

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

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

Формат задания:

```
<input type = password name = "имя_поля" value = "началь-
ный текст, содержащийся в поле" size = "ширина поля" maxlength
= "максимальное количество вводимых символов">
```

Скрытые поля не отображаются на странице и встраиваются в HTML-файл, когда необходимо передать серверу техническую ин-

формацию. Скрытые поля служат доступной альтернативой файлам *cookies* – специальным файлам, в которых сохраняются индивидуальные настройки пользователя, и позволяющим, например, восстановить последнее состояние формы при повторном посещении пользователем содержащей эту форму страницы.

Формат задания:

```
<input type = hidden name = "имя_поля" value = "текст, содержащийся в поле">
```

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

Многострочные текстовые поля используются для передачи текста большого размера.

Формат записи:

```
<textarea name = "имя элемента" rows="целое число" cols="целое число" wrap=значение disabled readonly>
Текст по умолчанию
</textarea>
```

Атрибуты тега `<textarea>`:

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

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

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

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

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

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

`readonly` – текстовое поле недоступно для изменения пользователем, в него не допускается вводить новый текст или модифициро-

вать существующий, оно не может получить фокус. Поле отображается обычным цветом.

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

Поле выбора файлов создает на экране кнопку, при нажатии на которую появляется Проводник Windows, позволяющий присоединить к отсылаемым на сервер данным любой файл с локального компьютера пользователя (рисунок 6.3). Рядом с кнопкой отображается небольшое текстовое поле, куда автоматически заносится имя отсылаемого файла и путь к нему на локальном диске. Поле выбора файлов работает корректно лишь при методе пересылки Post и формате кодировки multipart/form-data.

Формат задания:

```
<input type = "file" name = "имя" size = "ширина поля" max-  
length = "максимальная длина текста">
```

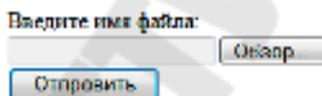


Рисунок 6.3 – Поле выбора файла

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

Формат задания:

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

По умолчанию начальное положение флажка считается неустановленным. Чтобы задать начальное положение установленного

флажка, надо дополнить его атрибутом `checked`. Значением установленного флажка является строка, заданная атрибутом `value`.

В рассмотренном примере надписи рядом с флажками созданы как простой текст. Для того чтобы флажок устанавливался не только щелчком непосредственно по квадратику флажка, но и щелчком по надписи (рисунок 6.4), необходимо связать надпись с флажком с помощью тега `label`, в котором содержится ссылка на связанный элемент управления с помощью атрибута `for`. Этому атрибуту ставится в соответствие идентификатор `id`.

```
<input type = checkbox id = WindowsXP name = system
value = "WXP" checked>
  <label for = WindowsXP>Windows XP</label><br>
<input type = checkbox id = Windows2000 name = system
value = "W2000">
  <label for = Windows2000> Windows 2000</label><br>
<input type = checkbox id = Windows98 name = system
value = "W98">
  <label for = Windows98> Windows 98</label><br>
```

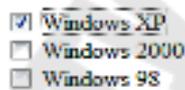


Рисунок 6.4 – Связывание надписи с флажком

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

Формат задания:

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

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

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

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

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

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

Кнопку можно создать двумя способами:

1. Использование тега `<input>`.

Формат задания:

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

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

Формат задания:

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

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

Подача запроса (`submit`) и сброс (`reset`).

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

Формат задания:

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

или

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

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

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

Формат задания:

```
<input type=reset name="имя кнопки" value="надпись на кнопке">
```

или

```
<button type=reset> Надпись на кнопке </button>
```

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

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

Формат записи:

```
<input type=image name="имя кнопки" src="URL изображения" value="надпись на кнопке">
```

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

Таблица 6.1 – Примеры задания кнопок

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

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

– *раскрывающийся список*, представляющий собой однострочное поле треугольной стрелкой, которая раскрывает список;

– *поле-список*, в котором на экран выводится заданное число строк; для просмотра всех строк список может быть снабжен полосой прокрутки;

– *список со множественным выбором*, позволяющий благодаря полосе прокрутки просматривать все позиции списка и выбирать одновременно несколько позиций.

Формат записи:

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

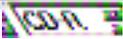
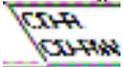
Атрибуты тега <select>:

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

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

selected – делает текущий элемент списка выделенным.

Таблица 6.2 – Примеры задания списка

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

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

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

Теги <legend> вводят надпись, которая помещается в разрыв рамки, обрамляющей группу (рисунок 6.5).



Рисунок 6.5 – Легенда

6.3 Формы и стили

В этом подразделе опишем некоторые интересные возможности, которые позволяют сделать HTML5 в связке с CSS3 с элементами форм.

Начнем с атрибута **required** для элемента формы `input`. Он устанавливает поле формы обязательным для заполнения перед отправкой формы на сервер. Если обязательное поле пустое, браузер выведет сообщение, а форма отправлена не будет. Вид и содержание сообщения зависит от браузера и меняться пользователем не может. По умолчанию атрибут `required` выключен. На рисунке 6.6 отображен результат в браузерах Chrome, Firefox и Internet Explorer.



Рисунок 6.6 – Атрибут `required` в различных браузерах

Псевдокласс **:invalid** применяется к полям формы, содержание которых не соответствует указанному типу. Например, для `type="number"` должно вводиться число, а не буквы, для `type="email"` – корректный адрес электронной почты.

В таблице 6.1 приведены некоторые новые значения, дано их описание и вид в окне браузера Chrome.

Таблица 6.1 – Новые значения атрибута `type`

Тип	Описание	Вид в браузере
color	Виджет для выбора цвета	
date	Поле для выбора календарной даты	
email	Для адресов электронной почты	
number	Ввод чисел	
range	Ползунок для выбора чисел в указанном диапазоне	

ГЛАВА 7 ИСПОЛЬЗОВАНИЕ СТИЛЕЙ ДЛЯ МАКЕТИРОВАНИЯ

7.1 Декоративные возможности CSS: формирование рамок и отступов

У каждого элемента, согласно блочной модели, есть 4 области (рисунок 7.1): *область контента* (белый прямоугольник, задается правилами `width` и `height`), в которой располагается внутреннее содержание элемента (текст, другие элементы). Между этим содержимым и рамкой располагается расстояние, которое называется *внутренний отступ* (правило `padding`). Далее следует сама *рамка* (правило `border`). После рамки идет *внешний отступ* (правило `margin`) – область, которая определяет взаимоотношения данного элемента с другими, она всегда прозрачная. Кроме этого, она может влиять и на размеры элемента в определенных случаях.

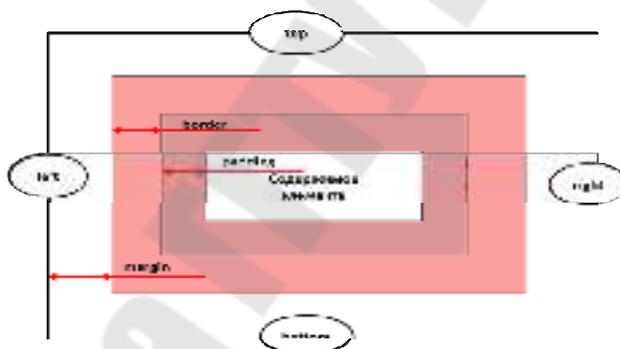


Рисунок 7.1 – Блочная модель

Внешняя область – правило `margin` – имеет четыре стороны: `top`, `bottom`, `right`, `left`, что дает возможность указать для этих четырех сторон по отдельности отступы следующими правилами:

```
Margin-top: auto | величина | %;  
Margin-bottom: auto | величина | %;  
Margin-right: auto | величина | %;  
Margin-left: auto | величина | %;
```

`auto` – расстояние будет рассчитывать сам браузер автоматически. По горизонтали `auto` почти всегда равно 0. По вертикали `auto` всегда 0.

величина — измеряется в *em*, *ex* и *px*. Величина может иметь и отрицательное значение, это позволяет элементам «заходить» друг на друга.

Есть возможность объединить все четыре правила в одно сборное правило `margin`:

```
Margin: Margin-top Margin-right Margin-bottom Margin-left;
```

В это правило через пробелы ставим четыре значения в строгой последовательности, начиная сверху по часовой стрелки, обходя все четыре стороны. Такая последовательность работает для любых правил, где есть четыре стороны.

Например, `margin: 10px 20px 20px 30px;`

Можно написать значения только для *top*, *right* и *bottom*. Это будет означать, что *left* будет такой же как и *right*. Если *top* и *bottom* равны и *right* и *left* равны, то можно написать два значения, *top* и *right*. Если написана одна цифра, то все значения одинаковые. Однако, если по горизонтали значения разные, то нужно писать 4 цифры.

Рассмотрим *внутренний отступ* — правило `padding`.

```
Padding-top: величина | %;  
Padding-right: величина | %;  
Padding-bottom: величина | %;  
Padding-left: величина | %;
```

Величина задается в *em*, *ex*, *px*. В отличие от внешнего отступа внутренний отступ бывает только положительным. Максимум, что можно сделать, это контент к рамке приблизить вплотную.

Проценты (%) работают так же как и для `margin`. Рассчитывается от ширины области, в которой находится элемент.

`Padding` по умолчанию равен 0. Все 4 значения можно записать в одно единственное правило с помощью такого же принципа, как и в правиле `margin`.

```
Padding:Padding-top Padding-right Padding-bottom Padding-left;
```

Для задания *рамки* используют правило `border`.

Рамки имеют три стилистических правила. Правила `border` можно задавать словами, а можно величиной (*em*, *ex*, *px*).

Border-width: величина | (thin | medium | thick);

medium – средний (по умолчанию);

thin – тонкий;

thick – толстый.

Border-color: цвет; – указание цвета рамки. По умолчанию цвет рамки тот же, что и у элемента. Если у элемента цвет зеленый, то и рамка будет зеленой.

Border-style: none | dotted | dashed | solid | double | groove | ridge | inset | outset ;

none – рамку не рисовать;

dotted – прерывистая линия (точка с пунктиром);

dashed – штриховая линия;

solid – сплошная линия;

double – двойная линия;

groove – имитация выпуклости рамки;

ridge – имитация выпуклости рамки;

inset – имитация того, как элемент вжат в страницу;

outset – имитация того, как элемент отжат от страницы.

Можно написать для каждой стороны комбинированное правило.

```
Border-top: (width | color | style);
```

```
Border-right: (width | color | style);
```

```
Border-bottom: (width | color | style);
```

```
Border-left: (width | color | style);
```

Например, нарисуем рамку одинаковую со всех сторон, только сверху она будет другого цвета.

```
border: border-width border-style border-color;
```

```
Border: 1px solid black;
```

```
Border-top: 1px solid blue;
```

Осталось рассмотреть *область с контентом* (содержимое элемента).

```
Width: auto | величина | %;
```

```
Height : auto | величина | %;
```

Эти два правила задают границы контента. По умолчанию эти правила принимают значения `auto`: браузер самостоятельно определяет высоту и ширину любых элементов. Для элементов уровня блок ширина принимается `auto`.

Проценты ширины и высоты берутся относительно ширины и высоты области, в которой находится этот элемент.

Если одновременно задать `width`, `height`, `padding`, `border`, то эти значения складываются.

7.2 Наследование и каскадирование

Наследованием называется перенос правил форматирования для элементов, находящихся внутри других. Такие элементы являются дочерними и они наследуют некоторые стилевые свойства своих родителей, внутри которых располагаются.

Разберем наследование на примере таблицы. Особенностью таблиц можно считать строгую иерархическую структуру тегов. Вначале следует контейнер `<table>`, внутри которого добавляются теги `<tr>`, а затем идут теги `<td>`. Если в стилях для селектора `table` задать цвет текста, то он автоматически устанавливается для содержимого ячеек, как показано ниже:

```
<html>
<head>
  <title>Наследование</title>
  <style type="text/css">
    table{
      color: red; /*Цвет текста*/
      background: #333; /*Цвет фона таблицы*/
      border: 2px solid red; /*Красная рамка вокруг таблицы*/
    }
  </style>
</head>
<body>
  <table cellpadding="4" cellspacing="0">
    <tr>
      <td>Ячейка 1</td>
      <td>Ячейка 2</td>
    </tr>
    <tr>
      <td>Ячейка 3</td>
      <td>Ячейка 4</td>
    </tr>
```

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

Таблица в этом случае примет вид как на рисунке 7.2.



Рисунок 7.2 – Наследование параметров цвета

В примере для всей таблицы установлен красный цвет текста, поэтому в ячейках он также применяется, поскольку тег `<td>` наследует свойства тега `<table>`. При этом следует понимать, что не все стилевые свойства наследуются. Так, `border` задает рамку вокруг таблицы в целом, но никак не вокруг ячеек. Аналогично не наследуется значение свойства `background`. Тогда почему цвет фона у ячеек в данном примере темный, раз он не наследуется? Дело в том, что у свойства `background` в качестве значения по умолчанию выступает `transparent`, т. е. прозрачность. Таким образом, цвет фона родительского элемента «проглядывает» сквозь дочерний элемент.

Чтобы определить, наследуется значение стилевого свойства или нет, требуется заглянуть в справочник по свойствам CSS по следующему адресу: www.w3c.org в разделе CSS 2.1 (Appendix F. Full property table).

Аббревиатура CSS расшифровывается как Cascading Style Sheets (каскадные таблицы стилей), где одним из ключевых слов выступает «каскад». Под *каскадом* в данном случае понимается одновременное применение разных стилевых правил к элементам документа – с помощью подключения нескольких стилевых файлов, наследования свойств и других методов. Чтобы в подобной ситуации браузер понимал, какое в итоге правило применять к элементу, и не возникало конфликтов в поведении разных браузеров, введены определенные приоритеты.

Ниже приведены приоритеты браузеров, которыми они руководствуются при обработке стилевых правил. Чем выше в списке находится пункт, тем ниже его приоритет, и наоборот.

Самым низким приоритетом обладает *стиль браузера*.

Стиль пользователя может включить пользователь сайта через настройки браузера. Такой стиль имеет более высокий приоритет и переопределяет исходное оформление документа. В браузере Internet Explorer подключение стиля пользователя делается через меню *Сервис > Свойство обозревателя > Кнопка «Оформление»*.

Стиль автора добавляет к документу разработчик стиля.

Ключевое слово *!important* играет роль в том случае, когда пользователи подключают свою собственную таблицу стилей. Если возникает противоречие, когда стиль автора страницы и пользователя для одного и того же элемента не совпадает, то *!important* позволяет повысить приоритет стиля или, иными словами, его важность.

Синтаксис применения *!important* следующий: вначале пишется желаемое стилевое свойство, затем через двоеточие – его значение и в конце после пробела указывается ключевое слово *!important*.

Повышение важности требуется не только для регулирования приоритета между авторской и пользовательской таблицей стилей, но и для повышения специфичности определенного селектора.

7.3 Позиционирование элементов на странице и управление моделью элемента

При работе с элементами предполагается, что элементы на веб-странице будут идти в той же последовательности, в которой они стоят в коде. Есть два правила, которые этот нормальный поток могут нарушить:

```
float: none | left | right;  
position: static | absolute | relative;
```

Правило `float` определяет «плавание». Как только элементу поставит правило `float: left|right;`, то происходит следующее:

- элемент становится блочным;
- плавающий элемент свои размеры определяет в соответствии со своим контентом. Это значит, что если в плавающем элементе контента нет, а ширина и высота были заданы, то он просто «схлопнется»;
- блочные элементы плавающие элементы игнорируют (исключением являются таблицы), а строчные элементы их обтекают;
- плавающие элементы в схемах с `margin colabs` не участвуют.

Можно сделать так, чтобы блочные элементы, идущие в коде за плавающими, учитывали эти плавающие элементы. Для этого воспользуемся специальным правилом:

```
clear: none | left | right | both;
```

С помощью этого правила блочные элементы учитывают (слева, справа, с обеих сторон) плавающий элемент.

По умолчанию `float` и `clear` имеют значения `none`.

Плавающие элементы встают максимально высоко и максимально в левую сторону или в правую в зависимости от правила `float`. А поскольку плавающие элементы друг друга учитывают, то это позволяет поставить их в ряд.

Правило `position` тоже серьезно меняет взаимоотношения элементов. Элемент с позиционированием можно двигать относительно разных областей.

`Position: static;` – это правило по умолчанию стоит у всех элементов. Применение этого правила дает возможность поставить все элементы друг за другом.

`Position: relation;` – это относительное позиционирование. В этом случае нормальный поток не нарушается. Однако с помощью задания смещения (`top`, `left`, `right`, `bottom`) мы можем смещать этот элемент относительно своего положения.

```
top: auto | величина | %;  
left: auto | величина | %;  
right: auto | величина | %;  
bottom: auto | величина | %;
```

`Position: absolute;` – позволяет позиционировать элемент относительно каких-то других элементов на странице.

Следует запомнить несколько свойств для элемента с `position: absolute`:

- этот элемент становится блочным.
- его размеры определяются его контентом.
- ни один элемент на странице, кроме корневого, данный элемент не учитывает.

У правила `position` есть еще одно значение – `fixed`. По своему действию это значение близко к `absolute`, но в отличие от него при-

вызывается к указанной свойствами `left`, `top`, `right` и `bottom` точке на экране и не меняет своего положения при прокрутке веб-страницы.

Следующие правила используют для динамических эффектов.

Многоцелевое свойство, которое определяет, как элемент должен быть представлен в документе:

```
display: none | block | inline | list-item
```

Очевидно, что правило `display` по умолчанию есть у любого элемента. Например, для элементов `div`, `p`, `h1`, `display` находится в значении `block`. Для таких элементов, как `span`, `em`, `strong` правило `display` имеет значение `inline`. Для элементов `li` - `display: list-item;`. Значение `none` говорит о том, что элемент не будет показываться вообще.

Правило, отвечающее за видимость элемента:

```
visibility: hidden | visible | inherit;
```

`hidden;` - элемент не показывается, но место для него резервируется.

`visible;` - значение по умолчанию.

`inherit` - наследует значение родителя.

Свойство `overflow` управляет отображением содержания блочного элемента, если оно целиком не помещается и выходит за область заданных размеров.

```
overflow: auto | scroll | visible | hidden;
```

`visible;` - значение по умолчанию. Использование этого правила приведет к тому, что все, что выйдет за пределы контента, будет показано и не будет влиять на другие элементы.

`hidden;` - контент, который не влезает, скроется и у пользователя отсутствует возможность просмотреть его.

`scroll;` - появятся полосы прокрутки, которые позволят увидеть весь текст.

`auto;` - полосы прокрутки появляются сами в зависимости от размеров контента. Если контент помещается в область, то никаких полос прокрутки добавлено не будет.

Любые позиционированные элементы на веб-странице могут накладываться друг на друга в определенном порядке, имитируя тем самым третье измерение, перпендикулярное экрану. Каждый элемент может находиться как ниже, так и выше других объектов веб-страницы, их размещением по z-оси и управляет `z-index`. Это свойство работает только для элементов, у которых значение `position` задано как `absolute`, `fixed` или `relative`. Синтаксис этого правила следующий:

```
z-index : auto | величина | inherit;
```

В качестве величины используются целые числа (положительные, отрицательные и ноль). Чем больше значение, тем выше находится элемент по сравнению с теми элементами, у которых оно меньше. При равном значении `z-index` на переднем плане находится тот элемент, который в коде HTML описан ниже. Кроме числовых значений, применяется `auto` – порядок элементов в этом случае строится автоматически, исходя из их положения в коде HTML и принадлежности к родителю, поскольку дочерние элементы имеют тот же номер, что и родительский элемент. Значение `inherit` указывает, что оно наследуется у родителя.

```
Правило clip: auto | rect (top right bottom left);
```

Определяет ту часть элемента, которую Вы собираетесь показать. В статических страницах это правило используется только при сложной верстке. Сразу появляется возможность для элементов с `position: absolute | fixed`; показать часть элемента.

```
Правило cursor: [url('путь к курсору'),] | [ auto | crosshair | default | e-resize | help | move | n-resize | ne-resize | nw-resize | pointer | progress | s-resize | se-resize | sw-resize | text | w-resize | wait | inherit ]
```

Устанавливает форму курсора, когда он находится в пределах элемента. Вид курсора зависит от операционной системы и установленных параметров.

По умолчанию у всех видимых элементах используется правило `cursor: auto`;

Бывают и другие значения.

`Cursor: crosshair;` – перекрестная линия, оптический прицел. Данное правило используется там, где нужно четко попадать в какие-то координаты.

`Cursor: pointer;` –указатель-рука, которая является своеобразной подсказкой, что можно кликнуть. При наведении на элемент, у которого прописано правило `cursor: pointer;`, курсор будет меняться на изображение руки.

`Cursor: help;` – вместо курсора появится курсор со знаком вопроса. Показывает, что пользователь может кликнуть на элемент и появится всплывающая подсказка.

`Cursor: wait;` – курсор в виде песочных часов.

`Cursor: move;` – подсказка, что пользователь может передвигать элемент.

`Cursor: text;` – вид вертикальной черты при наведении на текст. Означает, что текст можно выделить и что-то с ним сделать.

ГЛАВА 8 ОСНОВЫ WEB-ПРОГРАММИРОВАНИЯ ПРИ ВЕРСТКЕ WEB-СТРАНИЦ

8.1 Этапы и особенности верстки web-страниц

Верстка 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.

Теперь необходимо проанализировать макет и решить, как его превратить в веб-страницу. Для удобства происходит логическое разбиение картинки на отдельные блоки, с которыми идет дальнейшая работа. Обычно выделяют блоки – «шапку» страницы и основной контент.

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

- цвет фона веб-страницы;
- гарнитура основного шрифта, его размер и цвет;
- размер текста отдельных модулей (новостей, например);
- цвет, размер и гарнитура шрифта заголовков;
- 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.

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

- Корректно ли отображаются страницы в популярных браузерах?

- Происходит сохранение целостности данных при изменении размера шрифта в браузере как в большую, так и меньшую сторону?

- Можно продолжать работу с сайтом, если отключить показ изображений?

- Как существенно влияет на вид страниц разрешение 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 рисунков 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.

Основная задача верстки веб-страниц – это формирование документа, корректно отображающегося (возможно, с небольшими различиями) на основных платформах и в браузерах. Далее 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 1024×768 , 1280×1024 пикселей. Возьмем за ориентир 1024 пикселя, тогда общая ширина макета за вычетом вертикальной полосы прокрутки и рамки браузера окажется около 1000 пикселей.

«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 пре-

вышать, 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.

8.2 Режимы браузеров

Во время противостояния браузеров Internet Explorer и Netscape [4] каждый из разработчиков старался улучшить свое детище, чтобы усилить позиции программы на рынке и привлечь больше пользователей. Netscape 4 и IE4 очень плохо поддерживали веб-стандарты, поэтому следующая версия IE5 должна была не только исправить ошибки IE4, но и показать улучшенную поддержку спецификации CSS.

В процессе работы над браузером IE5 его разработчики столкнулись с неожиданной трудностью. Разница при отображении страницы в разных версиях браузера была настолько велика, что множество сайтов оказались бы неработоспособными при просмотре в IE5. Идея сделать кнопку для переключения в режим совместимости пришла только в версии 8.0, поэтому разработчики IE5 пошли другим путем. Все старые страницы отображались по старым правилам, а для включения режима поддержки стандартов в код страницы необходимо добавить элемент `<!DOCTYPE>`.

Браузер IE5 под операционную систему Mac стал первым браузером, у которого появилось два режима отображения страниц – режим совместимости и стандартный режим. Идея понравилась и быст-

ро распространилась среди разработчиков других браузеров, так что подобные режимы вскоре появились в Mozilla, Safari и Opera. IE5 под Windows, а также старые браузеры вроде Netscape 4 используют только режим совместимости.

Режим браузера для просмотра конкретной веб-страницы устанавливается через элемент `<!DOCTYPE>`, который является обязательным согласно спецификации HTML и XHTML.

8.3 Приемы использования таблиц на web-странице

Благодаря вложенным таблицам, гораздо проще управлять содержимым страницы сайта.

Создайте с помощью одной большой таблицы разметку своей страницы. Теперь возникла необходимость поместить в уже размеченную страницу, например, подмену сайта. В этом поможет еще одна таблица, вложенная в ячейку исходной таблицы (выделено прямоугольником в левой части (рисунок 8.1)).

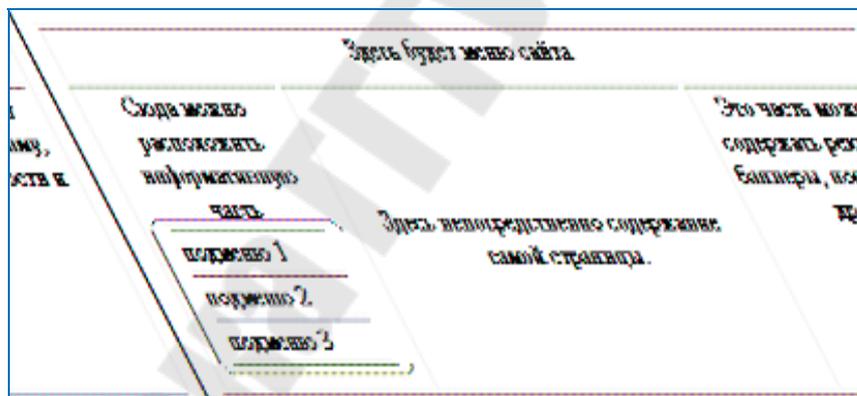


Рисунок 8.1 – Макетирование сайта с помощью вложенных таблиц

В примере ниже приведен код таблицы, представленной на рисунке 8.1.

```
<table border="1" width="600" height="200" align="center">
  <tr>
    <td colspan = "3" valign = "top" height = "30">
      <Center>Здесь будет меню сайта</Center></td>
    </tr>
    <tr align="center">
      <td valign="top" width="150" height="170">Здесь можно
        расположить информативную часть <br>
        <table border="1" width="150" height="170">
          <tr>
            <td>подменю 1</td>
            <td>подменю 2</td>
            <td>подменю 3</td>
          </tr>
        </table>
      </td>
      <td colspan="2" height="170">Здесь непосредственно содержание
        самой страницы.</td>
      <td colspan="2" height="170">Эта часть может содержать рекламные
        баннеры, новости...</td>
    </tr>
  </table>
```

```

<table border="1" width="120">
  <tr>
    <td>подменю 1</td>
  </tr>
  <tr>
    <td>подменю 2</td>
  </tr>
  <tr>
    <td>подменю 3</td>
  </tr>
</table>
</td>
<td width="300">Здесь непосредственно содержание
самой страницы.</td>
<td valign="top" width="150">Эта часть может содер-
жать рекламу, баннеры, новости и др.</td>
</tr>
</table>

```

В данном примере использовались атрибуты `width` и `height` как для самой таблицы, так и для ее ячеек; размеры заданы в пикселях. Ширина самой таблицы – 600 пикселей, высота – 200. Для ячеек задана ширина 150, 300, 150, т. е. ширина всех ячеек в сумме получается 600, а высота ячеек была задана 30 и 170, в итоге их сумма равна высоте всей таблицы. А для вложенной таблицы ширина самой таблицы задана как 120 пикселей. Это будет выглядеть как на рисунке 8.2.

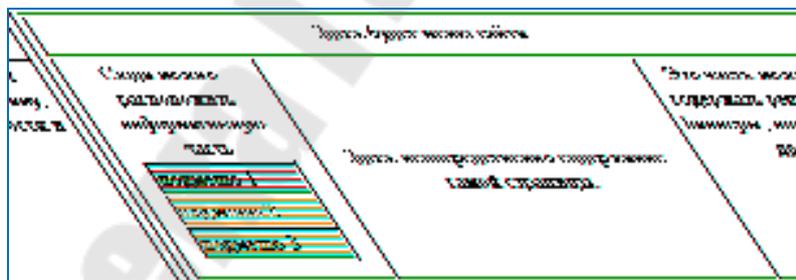


Рисунок 8.2 – Использование атрибута `background` для вложенной таблицы

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

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

Следующим шагом будет наполнение страницы содержимым, вставка вложенных таблиц, удаление рамок вокруг таблиц. В итоге может получиться web-страничка, представленная на рисунке 8.3.



Рисунок 8.3 – Пример web-страницы на «чистом» HTML (табличная верстка)

8.4 Верстка на CSS. Создание простого блока

В этом разделе речь пойдет об одном из способов создания стандартного блока на основе CSS, состоящего из «шапки» и основной части.

Для этого потребуется выполнить ряд простых действий.

Во-первых, для удобства вынесем CSS в отдельный файл.

Во-вторых, вставим в этот файл код:

```
/* Задание стилей всего блока */
#block{
    width: 250px; /* Задание ширины блока */
}
/* Задание стилей заголовка */
.head {
    text-align:center; /*Выравнивание заголовка по центру блока*/
    color: #fff; /* Задание цвета заголовка (здесь – белый) */
    background-color:#0274b0; /*Задание цвета фона (здесь – синий)*/
    border: 2px solid #ffba00; /*Задание сплошной границы блока шириной в 2 пикселя и её цвета*/
    font-size: 15px; /*Задание размера шрифта заголовка*/
    font-weight:bold; /*Задание полужирного начертания шрифта*/
    padding: 7px 0 7px 0; /* Задание верхнего и нижнего отступов текста заголовка от границ блока */
}
/* Задание стилей основного блока */
```

```

.body {
    color:#333; /* Задание цвета текста */
    background-color: #d2efff; /* Задание цвета фона */
    border: 2px solid #ffba00; /* Задание сплошной границы
    блока шириной в 2 пикселя и её цвета */
    border-top-style: none; /*Удаление верхней границы блока */
    font-size: 12px; /*Задание размера шрифта */
    padding: 5px; /*Задание отступа в 5 пикселей со всех сторон */
}

```

В-третьих, чтобы увидеть работу стилей, создадим в той же директории html-файл с кодом:

```

<html>
<head>
  <link rel="stylesheet"href="style.css"type="text/css"/>
</head>
<body>
<div id="block">
  <div class="head">
    ЗАГОЛОВОК БЛОКА
  </div>
  <div class="body">
    ОСНОВНОЙ БЛОК.<br>
    Текст блока....
  </div>
</div>
</body>
</html>

```

И наконец, сохраняем и смотрим, что получилось.

Результат изображен на рисунке 8.4.

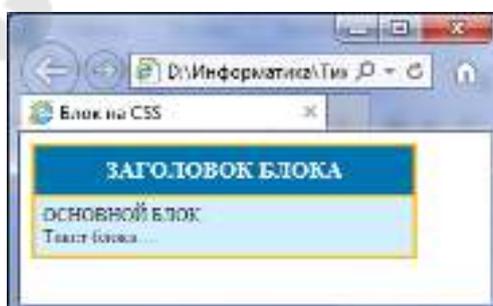


Рисунок 8.4 – Стандартный блок на CSS

8.5 Современная верстка сайта при помощи CSS

Начнем с самого простого, стандартного типа шаблонов (рисунок 8.5).

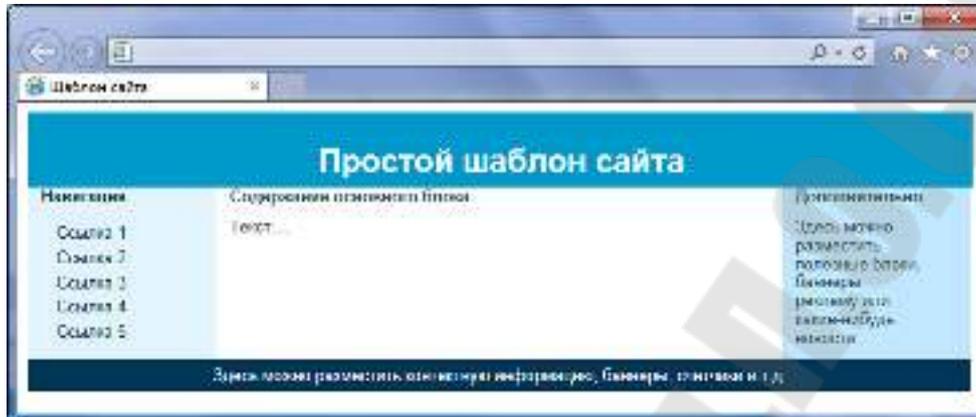


Рисунок 8.5 – Пример шаблона сайта

К особенностям шаблона, изображенного на рисунке 8.5, относятся:

- наличие 3-х колонок;
- стандартные поля заголовков;
- простое меню;
- отсутствие таблиц;
- отсутствие графики.

Этот шаблон очень удобен при изучении блочной верстки.

Как и раньше, приведем содержание *.css (пример 8.1) и *.html (пример 8.2) файлов, соответственно, с комментариями команд кода.

Пример 8.1

```
/*Задание стилей всего шаблона*/  
body {  
    font: 80% Arial;  
    text-align:center;  
    color : #2F4F4F;  
}  
  
/* Задание стилей новой строки */  
p {margin: 0 10px 10px;}  
  
/* Задание стилей ссылок */  
a {  
    padding:5px;  
    text-decoration:none;
```

```
        color:#003333;
        margin:17px;
    }

    /* Задание стилей ссылок при наведении */
    a:hover {
        text-decoration:none;
        color:#66CCFF;
        background:#0099CC;
    }

    /* Задание стилей блока заголовка */
    div#header {
        background-color:#0099CC;
        color:#fff;
        height:60px;
        line-height:80px;
        padding-left:1px;
        width:750px;
    }

    /* Задание стилей всего шаблона */
    div#all {
        text-align:left;
        width:750px;
        margin:0 auto;
    }

    /* Задание стилей навигации */
    div#navigation {
        background:#e3f4ff;
        color:#003333;
        float:left;
        width:150px;
        margin-left:-750px;
    }

    /* Задание стилей списка */
    div#navigation ul {
        margin:15px 0;
        padding:0;
        list-style-type:none;
    }

    /* Задание стилей элемента списка */
    div#navigation li{margin-bottom:5px;}

    /* Задание стилей правого столбца */
    div#extra {
```

```

        background:#c1e7ff;
        float:left;
        width:150px;
        margin-left:-150px;
    }

/* Задание стилей "подвала" */
div#footer {
    background-color:#013657;
    color:#fff;
    clear:left;
    height:25px;
}

div#footer p {
    margin:0;
    padding:5px 10px;
}

/* Задание стилей всего шаблона */
div#templ {
    float:left;
    width:100%;
}

/* Задание стилей центрального столбца */
div#content {margin: 0 150px;}

```

Пример 8.2

```

<html>
<head>
  <title>Шаблон сайта</title>
  <link      rel="stylesheet"          type="text/css"
  href="style.css">
</head>
<body>
  <div id="all">
  <div id="header">
    <h1 align="center">Простой шаблон сайта</a></h1>
  </div>
  <div id="templ">
    <div id="content">
      <p>
        <strong>Содержание основного блока</strong>
      </p>
      <p>Текст....</p>
    </div>
  </div>
  <div id="navigation">

```

```

<p><strong>Навигация</strong></p>
<ul>
  <li><a href="#">Ссылка 1</a></li>
  <li><a href="#">Ссылка 2</a></li>
  <li><a href="#">Ссылка 3</a></li>
  <li><a href="#">Ссылка 4</a></li>
  <li><a href="#">Ссылка 5</a></li>
</ul>
</div>
<div id="extra">
  <p><strong>Дополнительно</strong></p>
  <p>Здесь можно разместить полезные блоки, баннеры
  рекламу или какие-нибудь новости</p>
</div>
<div id="footer">
  <p align="center">Здесь можно разместить контактную
  информацию, баннеры, счетчики и т.д.</p>
</div>
</div>
</body>
</html>

```

Если немного доработать стилевой css-код и html-код, то без особого труда можно получить вот такой шаблон сайта (рисунок 8.6). Заметим, что такой же шаблон мы создавали с помощью табличной верстки (рисунок 8.3).

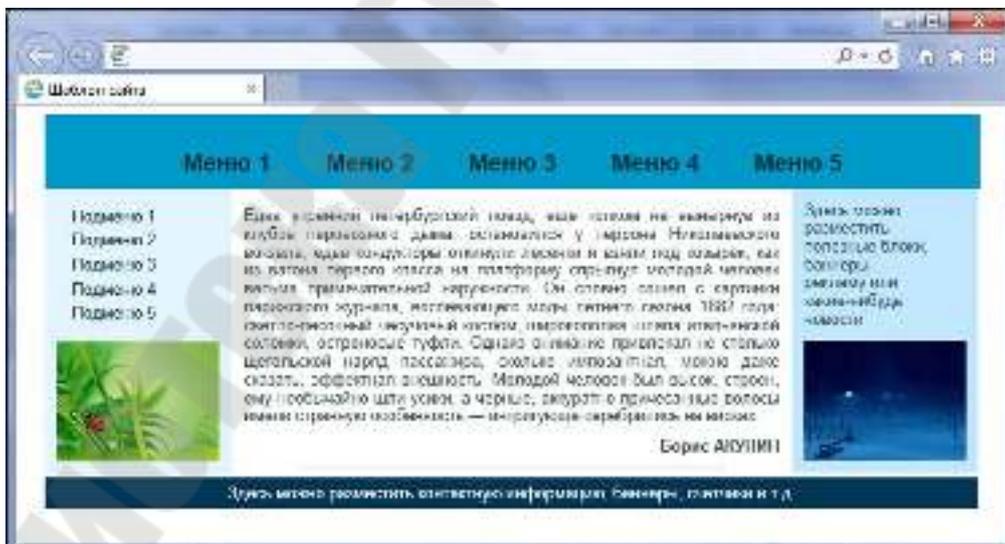


Рисунок 8.6 – Пример web-странички на HTML и CSS (блочная верстка)

8.6 Современная верстка на HTML5 и CSS3

HTML5 – это не продолжатель языка разметки гипертекста, а новая открытая платформа, предназначенная для создания веб-приложений, использующих аудио, видео, графику, анимацию и многое др.

Семантика дает представление о структуре документа и позволяет людям и программам более полно управлять данными. В HTML5 добавлено множество семантических тегов, а также поддержка RDFa, микроформатов и микроданных.

Благодаря кэшу HTML5, веб-приложения могут быстрее работать и запускаться даже без подключения к Интернету.

Можно рисовать прямо на веб-странице, используя Canvas API. Технологии WebGL и CSS3 3D позволяют отображать трехмерную графику непосредственно в браузере.

Приложения и сайты могут проигрывать аудио и видео без установки дополнительных плагинов вроде Flash-а.

CSS3 в интеграции с HTML5 позволяет управлять видом любых элементов на странице, создавать потрясающие эффекты без ущерба семантической структуры, производительности и без дополнительных скриптов.

Далее приведем некоторые его возможности:

- Поддержка геолокации – определение местоположения пользователя на карте и использование этой информации для вычисления маршрута его движения, вывода близлежащих магазинов, кинотеатров, кафе и других данных.
- Воспроизведение видеороликов.
- Воспроизведение аудиофайлов.
- Локальное хранилище – позволяет сайтам сохранять информацию на локальном компьютере и обращаться к ней позже.
- Фоновые вычисления – стандартный способ запуска JavaScript в браузере в фоновом режиме.
- Оффлайновые приложения – страницы, которые могут работать при отключении Интернета.
- Рисование – внутри тега `<canvas>` с помощью JavaScript можно рисовать фигуры, линии, создавать градиенты и трансформировать объекты на лету.
- Новые элементы форм: для даты, времени, поиска, чисел, выбора цвета и др.

Кроме этих возможностей, в HTML5 включены новые теги для разметки документа, выброшены устаревшие теги и модифицированы некоторые другие. Для верстки веб-страниц, в первую очередь, необходимо понять, что поменялось и как перевести страницу на HTML5.

Рассмотрим новые возможности CSS3:

- Появились новые селекторы, которые упрощают выборку. Например, простая задача – сделать особый стиль у последнего пункта меню (если у всех пунктов внизу есть рамка, а у последнего пункта ее нет). Раньше нужно было задавать отдельный класс для последнего пункта, а в CSS3 есть новый селектор, который поможет сразу вытащить этот последний элемент. Это сократит время работы и уменьшит размер кода. И таких полезных мелочей в CSS3 очень и очень много, благодаря новым селекторам.

- Богатые возможности по работе с фоном. Наконец-то, в CSS3 появилась возможность задать для одного элемента сразу несколько фонов. Так же их теперь можно растягивать по ширине и высоте.

- Появилась возможность задать прозрачный цвет, чего сильно не хватало раньше. Также появилась возможность задать цвет через HSL (оттенок–насыщенность–яркость).

- Очень легко стало создавать закругленные рамки. Практически в любом более-менее сложном дизайне присутствуют закругления у различных блоков и элементов формы. Раньше это была целая проблема, а в CSS3 это делается одной строчкой.

- Теперь можно задать свой шрифт на сайте, и не нужно бояться, что у кого-то он не отобразится. Вы просто скачиваете шрифт, копируете на свой сайт, а через CSS3 его подключаете к странице. В CSS 2.1 не было такой возможности, что сильно ограничивало возможности по дизайну страницы.

- Необычайно легко стало добавлять тень к элементам. На многих сайтах есть тень у некоторых элементов, и если раньше ее делали долго и напряженно, то сейчас это опять же одна небольшая строчка в файле стилей.

- Можно создавать линейные и сферические градиенты. Очень часто на страницах встречаются градиенты, и раньше без нарезки картинок, подключения ее в качестве фона и повторения по X или по Y не обходилось. В CSS3 же задать градиент очень и очень легко, а возможности там таковы, что можно создать даже очень сложный градиент, где есть 10 переходов между цветами.

- Появились трансформации. Теперь очень легко можно, например, повернуть целый блок на определенный угол, или растя-

нуть/сжать его на определенный процент, или подвинуть его (особенно полезно в совокупности с JavaScript).

- Появилась анимация. Теперь не нужно при наведении курсора мыши на элемент плавно менять его состояние через JavaScript. Огромное количество возможностей, которое встречается на сайтах, теперь можно сделать через анимацию в CSS3. В некоторых случаях это поможет Вам отказаться от того же jQuery, который весит весьма прилично, тем самым, Вы увеличите скорость загрузки страницы, что так же хорошо скажется на поисковой оптимизации.

Опять же это – не все возможности, появившиеся в CSS3, но даже эти уже позволят упростить верстку страницу, порой, многократно.

Любой код разметки начинается с доктайпа, этот элемент говорит браузеру, на каком языке разметки и его версии написан документ. Также доктайп переводит браузер в один из возможных режимов: стандартный, почти стандартный и режим совместимости. В HTML5 нет подобных делений, доктайп – один единственный и при его наличии браузер работает в стандартном режиме.

```
<!DOCTYPE html>
```

Из всех видов это – самый короткий доктайп, его легко запомнить и набирать по памяти.

Изменения претерпели и другие теги. Так, у тега `<html>` нет атрибута `xmlns`, а кодировка документа сократилась до такой записи.

```
<meta charset="utf-8">
```

Впрочем, старый способ указать кодировку также остался.

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

Атрибут `type` у тега `<script>` и `<style>` можно опустить, браузер автоматически понимает содержимое этих тегов и ему уже не требуется явно об этом напоминать. Простейший код:

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

```
<title>Пример страницы</title>
<style>
  p { color: navy; }
</style>
</head>
<body>
  <p>Страница на HTML5</p>
</body>
</html>
```

Новые теги HTML5

В HTML5 для структуры кода введено несколько новых тегов: `<article>`, `<aside>`, `<footer>`, `<header>`, `<nav>`, которые заменяют в некоторых случаях привычный `<div>`. Хотя кажется, что особой разницы между тегами `<div class="header">` и `<header>` нет, между ними «лежит огромная пропасть». Теги ориентированы не на людей, которым нет смысла заглядывать в исходный код страницы, а на машины, интерпретирующие код. Машины или роботы не понимают `<div class="header">`, для них это типовой тег разметки – замени его на `<div class="abracadabra">`, и смысл не поменяется. Другое дело – `<header>`; робот, обнаружив этот тег, воспринимает его именно как шапку сайта или раздела.

Поисковые системы начинают лучше индексировать сайт, потому что четко отделяют контент страницы от вспомогательных элементов. Речевые браузеры, предназначенные для слепых людей, пропускают заголовок и переходят непосредственно к содержимому. Сайты могут автоматически обмениваться контентом и другой информацией между собой. Все эти возможности называются семантикой и позволяют представить данные в удобном для роботов виде.

Давайте для начала сделаем шапку сайта с помощью тега `<header>`:

```
<header>
  <div class="header-bg">
    
  </div>
</header>
```

Попытка добавить в стилях фон к тегу `<header>`, ни к чему не привела; фон в некоторых браузерах отображаться не желает. Все новые теги следует сделать вначале блочными через свойство `display`, тогда они начнут корректно выводиться:

```

<style>
  header {
    display: block;
    background: #00B0D8 url(images/header-gradient.png) re-
    peat-x;
  }
</style>

```

Данный пример будет работать во всех браузерах, кроме IE7 и IE8. Internet Explorer не добавляет стиль к элементам, которые не понимает. Это недоразумение можно исправить, если создать фиктивный элемент с помощью JavaScript. Для этого включим в <head> такой код.

```

<script>
  document.createElement("header");
</script>

```

Если на странице встречается один тег, этот скрипт вполне подойдет для работы. Но не хочется повторять строку десять раз для десяти разных тегов, поэтому автоматизируем этот процесс через цикл. Сами теги указываются списком, разделяясь запятой:

```

<!--[if lt IE 9]>
  <script>
    var e = ("article, aside, figcaption, figure, footer,
    header, hgroup, nav, section, time").split(',');
    for (var i = 0; i < e.length; i++) {
      document.createElement(e[i]);
    }
  </script>
<![endif]-->

```

Сам скрипт заключается в условные комментарии, чтобы выполнялся только для IE версии 8.0 и ниже. В IE9 поддержка новых тегов HTML5 уже включена.

Пример выше не обязательно вставлять к себе на сайт, можно воспользоваться общедоступным скриптом, написанным Реми Шарпом и распространяемым по лицензии MIT. Для этого достаточно указать на него ссылку, как показано ниже:

```

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

```

```
</script>  
<![endif]-->
```

Все указанные скрипты должны располагаться в коде перед CSS.

Таким образом, для полноценного использования тегов HTML5 во всех браузерах достаточно выполнить три условия:

- установить доктайп `<!DOCTYPE html>`;
- включить скрипт из примера 8.9 или 8.10;
- в стилях для новых тегов установить `display: block`.

Теперь рассмотрим некоторые теги HTML5 более подробно, чтобы понять область их применения.

Тег `<article>...</article>` задает содержание сайта наподобие новости, статьи, записи блога, форума или др.

Internet Explorer до версии 8.0 включительно игнорирует тег `<article>`, но отображает его содержимое. Также в этом браузере любые стили не применяются к селектору `article`.

Браузер Firefox полностью поддерживает этот тег, начиная с версии 4.0, но версии 3.0 и старше также корректно отображают содержимое тега.

Тег `<aside>...</aside>` определяет блок сбоку от контента для размещения рубрик, ссылок на архив, меток и другой информации. Такой блок, как правило, называется «сайдбар» или «боковая панель».

Для этого тега доступны универсальные атрибуты.

Internet Explorer до версии 8.0 включительно игнорирует тег `<aside>`, но отображает его содержимое. Также в этом браузере любые стили не применяются к элементу, пока он не создан через скрипт, как показано в примере.

Браузер Firefox полностью поддерживает этот тег, начиная с версии 4.0, но версии 3.0 и старше также корректно отображают содержимое тега.

Тег `<canvas>` создает область, в которой при помощи JavaScript можно рисовать разные объекты, выводить изображения, трансформировать их и менять свойства. При помощи тега `<canvas>` можно создавать рисунки, анимацию, игры и др.

Формат записи следующий:

```
<canvas id="идентификатор">  
</canvas>
```

Элемент `<canvas>` имеет следующие атрибуты:

`height` – задает высоту холста. По умолчанию 300 пикселей.

`width` – задает ширину холста. По умолчанию 150 пикселей.

Ниже приведем пример создания простого рисунка с помощью элемента `<canvas>`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>canvas</title>
    <meta charset="utf-8">
    <script>
      window.onload = function() {
        var drawingCanvas = document.getElementById('smile');
        if(drawingCanvas && drawingCanvas.getContext) {
          var context = drawingCanvas.getContext('2d');
          // Рисуем окружность
          context.strokeStyle = "#000";
          context.fillStyle = "#fc0";
          context.beginPath();
          context.arc(100,100,50,0,Math.PI*2,true);
          context.closePath();
          context.stroke();
          context.fill();
          // Рисуем левый глаз
          context.fillStyle = "#fff";
          context.beginPath();
          context.arc(84,90,8,0,Math.PI*2,true);
          context.closePath();
          context.stroke();
          context.fill();
          // Рисуем правый глаз
          context.beginPath();
          context.arc(116,90,8,0,Math.PI*2,true);
          context.closePath();
          context.stroke();
          context.fill();
          // Рисуем рот
          context.beginPath();
          context.moveTo(70,115);
          context.quadraticCurveTo(100,130,130,115);
          context.quadraticCurveTo(100,150,70,115);
          context.closePath();
          context.stroke();
          context.fill();
        }
      }
    </script>
  </head>
  <body>
    <div style="text-align: center;">
      <img alt="A simple smiley face drawn on a canvas." data-bbox="173 225 869 883"/>
    </div>
  </body>
</html>
```

```

</script>
</head>
<body>
  <canvas id="smile" width="200" height="200">
    <p>Ваш браузер не поддерживает рисование.</p>
  </canvas>
</body>
</html>

```

Результат примера в браузере IE 11 показан на рисунке 8.7.



Рисунок 8.7 – Вывод рисунка с помощью тега `<canvas>`

Элемент `<figure>...</figure>` используется для группирования любых элементов, например, изображений и подписей к ним.

Internet Explorer до версии 8.0 включительно игнорирует тег `<figure>`, но отображает его содержимое. Также в этом браузере любые стили не применяются к элементу, пока он не создан через скрипт, как показано в примере.

Firefox полностью поддерживает этот тег, начиная с версии 4.0, но версии 3.0 и старше также корректно отображают содержимое тега.

Браузеры Firefox и Chrome автоматически добавляют к `<figure>` свойство `margin` со значением `1em 40px`.

Тег `<figcaption>` должен быть первым или последним элементом в группе. Синтаксис:

```

<figure>
  <figcaption>Описание</figcaption>
</figure>

```

Пример применения `<figure>`:

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>FIGURE и FIGCAPTION</title>
  <script>
    document.createElement('figure');
    document.createElement('figcaption');
  </script>
  <style>
    figure {
      background: #d9dabb; /* Цвет фона */
      display: block; /* Блочный элемент */
      width: 150px; /* Ширина */
      height: 190px; /* Высота */
      float: left; /* Блоки выстраиваются по горизонтали */
      margin: 0 10px 10px 0; /* Отступы */
      text-align: center; /* Выравнивание по центру */
    }
    figure img {
      border: 2px solid #8b8e4b; /* Параметры рамки */
    }
    figure p {
      margin-bottom: 0; /* Отступ снизу */
    }
  </style>
</head>
<body>
  <article>
    <figure>
      <p></p>
      <figcaption>Софийский собор</figcaption>
    </figure>
    <figure>
      <p></p>
      <figcaption>Польский костёл</figcaption>
    </figure>
  </article>
</body>
</html>

```

Результат данного примера показан на рисунке 8.8.

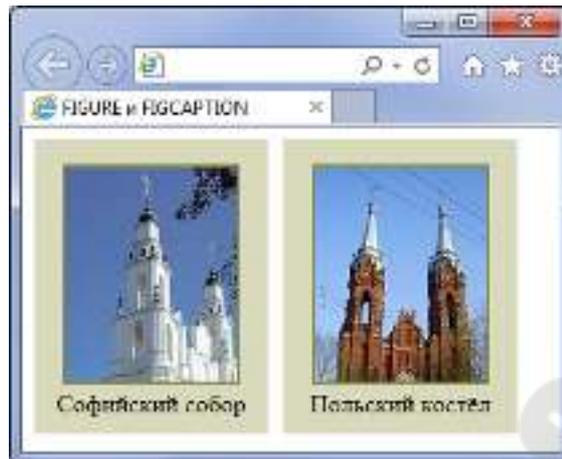


Рисунок 8.8 – Использование тегов `<figure>` и `<figcaption>`

Internet Explorer до версии 8.0 включительно игнорирует тег `<figcaption>`, но отображает его содержимое. Также в этом браузере любые стили не применяются к элементу, пока он не создан через скрипт, как показано в примере.

Firefox полностью поддерживает этот тег, начиная с версии 4.0, но версии 3.0 и старше также корректно отображают содержимое тега.

Тег `<footer>...</footer>` задает «подвал» сайта или раздела; в нем может располагаться имя автора, дата документа, контактная и правовая информация.

Internet Explorer до версии 8.0 включительно игнорирует тег `<footer>`, но отображает его содержимое. Также в этом браузере любые стили не применяются к селектору `FOOTER`.

Firefox полностью поддерживает этот тег, начиная с версии 4.0, но версии 3.0 и старше также корректно отображают содержимое тега.

Тег `<nav>` задает навигацию по сайту. Если на странице несколько блоков ссылок, то в `<nav>` обычно помещают приоритетные ссылки. Также допустимо использовать несколько тегов `<nav>` в документе. Запрещается вкладывать `<nav>` внутрь `<address>`.

Синтаксис элемента следующий:

```
<nav>ссылки</nav>
```

Internet Explorer до версии 8.0 включительно игнорирует тег `<nav>`, но отображает его содержимое. Также в этом браузере любые стили не применяются к селектору `NAV`.

Firefox полностью поддерживает этот тег, начиная с версии 4.0, но версии 3.0 и старше также корректно отображают содержимое тега.

Тег `<section>...</section>` задает раздел документа, может применяться для блока новостей, контактной информации, глав текста, вкладок в диалоговом окне и др. Раздел обычно содержит заголовок. Допускается вкладывать один тег `<section>` внутрь другого.

Internet Explorer до версии 8.0 включительно игнорирует тег `<section>`, но отображает его содержимое. Также в этом браузере любые стили не применяются к селектору `SECTION`.

Firefox полностью поддерживает этот тег, начиная с версии 4.0, но версии 3.0 и старше также корректно отображают содержимое тега [5]–[7].

ГЛАВА 9 ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ HTML И CSS

9.1 Размещение мультимедийной информации на web-странице

Под мультимедиа, в первую очередь, понимают аудио и видео. Мультимедиа в приложении к web-дизайну – это аудио- и видеоролики, размещенные на web-страницах.

Мы будем рассматривать работу с мультимедиа исключительно средствами HTML5.

Форматов мультимедийных файлов существует не меньше, чем форматов файлов графических. Это такие форматы, как wav, ogg, mp4, quicktime (один из первых). Как и в случае с интернет-графикой, браузеры поддерживают далеко не все мультимедийные форматы, а только немногие. Всю необходимую информацию о форматах можно найти в [3].

Вставка аудио-ролика

Для вставки на web-страницу аудиоролика язык HTML5 предусматривает парный тег `<audio>`, формат записи которого следующий:

```
<audio src="путь_к_файлу" autoplay controls></audio>
```

Перед тем как начнем рассматривать атрибуты тега `<audio>`, напомним, что этот тег блочный. Это означает, что нельзя вставить аудио-ролик на web-страницу в качестве части абзаца.

Рассмотрим атрибуты.

src – в качестве значения этого атрибута указывается путь к файлу вместе с расширением.

autoplay – позволит автоматически воспроизводить аудиоролик. По умолчанию браузер не будет воспроизводить аудио-ролик.

controls – добавит элементы управления воспроизведением аудиоролика, которые включают кнопку запуска и приостановки воспроизведения, шкалу воспроизведения и регулятор громкости. По умолчанию аудиоролик никак не отображается на web-странице.

autobuffer – позволяет браузеру сразу после загрузки web-страницы загружать файл аудиоролика, что позволит исключить задержку файла перед началом его воспроизведения. Данный атрибут

имеет смысл указывать в теге `<audio>` только в том случае, если там отсутствует атрибут `autoplay`.

Вставка видео-ролика

Для вставки на web-страницу видеоролика предназначен парный тег `<video>`, формат записи которого следующий:

```
<video src="путь_к_файлу" autoplay controls poster="путь_к_файлу">
</video>
```

Как и тег `<audio>`, тег `<video>` создает блочный элемент web-страницы и поддерживает атрибуты `autoplay`, `controls` и `autobuffer`.

Если воспроизведение видеоролика еще не запущено, в панели просмотра будет выведен первый его кадр или вообще ничего (конкретное поведение зависит от браузера). Но мы можем указать графическое изображение, которое будет туда выведено в качестве заставки. Для этого служит атрибут `poster` тега `<video>`; его значение указывает интернет-адрес нужного графического файла.

Одинарный тег `<source>` указывает как интернет-адрес мультимедийного файла, так и его тип MIME (подробно о типах MIME смотрите в [3]). Для этого предназначены атрибуты `src` и `type` данного тега, соответственно, например:

```
<video>
  <source src="film.ogg" type="video/ogg">
  <source src="film.mov" type="video/quicktime">
</video>
```

Данный тег `<video>` определяет сразу два видеофайла, хранящих один и тот же фильм: один – формата `OGG`, другой – формата `QuickTime`. Браузер, поддерживающий формат `OGG`, загрузит и воспроизведет первый файл, а браузер, поддерживающий `QuickTime`, – второй файл.

Отметим, что тип MIME видеофайла (и, соответственно, атрибут `type` тега `source`) можно опустить. Но тогда браузеру придется загрузить начало файла, чтобы выяснить, поддерживает ли он формат этого файла. А это лишняя и совершенно ненужная нагрузка на сеть, так что тип MIME лучше указывать всегда.

Если браузер вообще не поддерживает теги `<audio>` и `<video>`, то в таком случае он их проигнорирует и ничего на web-страницу не выведет. Однако можно указать текст замены, в котором описать воз-

никшую проблему и предложить какой-либо путь ее решения (например, сменить браузер). Такой текст замены ставят внутри тега <audio> или <video> после всех тегов <source> (если они есть).

9.2 Импорт CSS

Правило @import позволяет импортировать содержимое CSS-файла в текущую стилевую таблицу; @import не разрешается вставлять после любых объявлений, кроме @charset или другого @import.

Синтаксис:

```
@import url("имя файла") [типы носителей];  
@import "имя файла" [типы носителей];
```

В качестве типа носителя выступают различные устройства, например, принтер, КПК, монитор и др. В таблице 9.1 перечислены некоторые из них.

Таблица 9.1 – Типы носителей и их описание

Тип	Описание
all	Все типы. Это значение используется по умолчанию
aural	Речевые синтезаторы, а также программы для воспроизведения текста вслух. Сюда, например, можно отнести речевые браузеры
braille	Устройства, основанные на системе Брайля, которые предназначены для слепых людей
handheld	Наладонные компьютеры и аналогичные им аппараты
print	Печатающие устройства вроде принтера
projection	Проектор
screen	Экран монитора
tv	Телевизор

Использование типов носителей совместно с импортом файла дает возможность указывать стиль только для определенных устройств. В качестве значения используется путь к стилевому файлу, который указывается внутри необязательной конструкции url(). Путь к файлу при этом можно писать как в кавычках (двойных или одинарных), так и без них.

Браузер Internet Explorer до версии 7.0 включительно не поддерживает типы носителей при импорте стилевых файлов. Более того, при добавлении типа носителя стиливой файл вообще не загружается.

9.3 Использование внешних объектов

Элемент `<embed>` используется для загрузки и отображения объектов (например, видеофайлов, флэш-роликов, некоторых звуковых файлов и т. д.), которые исходно браузер не понимает. Как правило, такие объекты требуют подключения к браузеру специального модуля, который называется плагином, или запуском вспомогательной программы.

Спецификация HTML 4.0 рекомендует использовать тег `<object>` для загрузки внешних данных вместо тега `<embed>`. Однако некоторые браузеры не отображают таким образом нужную информацию, поэтому наилучшим вариантом будет поместить `<embed>` внутрь контейнера `<object>`.

Вид внедренного объекта зависит от установленных в браузере плагинов, типа загружаемого файла, а также от атрибутов тега `<embed>`.

Формат записи следующий:

```
<embed width="..." height="..."></embed>
```

Атрибуты:

`align` – определяет, как объект будет выравниваться на странице и способ его обтекания текстом.

`height` и `width` – высота и ширина объекта.

`hidden` – указывает, скрыть объект на странице или нет.

`hspace` и `vspace` – горизонтальный и вертикальный отступ от объекта до окружающего контента.

`pluginspage` – адрес страницы в Интернете, откуда можно скачать и установить плагин к браузеру.

`src` – путь к файлу.

`type` – MIME-тип объекта.

Тег `<applet>` предназначен для вставки на страницу апплетов – небольших программ, написанных на языке Java. Этот тег является устаревшим, взамен необходимо использовать более гибкий тег `<object>`. Между открывающим и закрывающим тегом можно добавить текст, который будет отображаться в браузере, если он не поддерживает апплеты. В противном случае текст не выводится.

Синтаксис:

```
<applet code="URL">Текст</applet>
```

Атрибуты:

`align` – задает выравнивание апплета относительно близлежащих элементов и текста.

`alt` – альтернативный текст.

`archive` – указывает путь и имя файла с архивом.

`code` – имя файла.

`codebase` – путь к папке с апплетом, который задан атрибутом `code`.

`height` и `width` – высота и ширина апплета.

`hspace` и `vspace` – горизонтальный и вертикальный отступ от апплета до окружающего контента.

Использование этого тега осуждается спецификацией HTML и его наличие приведет к невалидному коду.

Тег `<noembed>` предназначен для отображения информации на веб-странице, если браузер не поддерживает работу с плагинами. Во всех остальных случаях содержимое контейнера `<noembed>` будет проигнорировано. Плагином называется подключаемый к браузеру программный модуль, расширяющий возможности браузера. Например, к плагинам можно отнести поддержку Flash, QuickTime VR, VRML и т. д.

Синтаксис:

```
<noembed>Текст</noembed>
```

Этот тег не входит в спецификацию HTML и его наличие приведет к невалидному коду.

Тег `<param>` предназначен для передачи значений параметров Java-апплетам или объектам веб-страницы, созданным с помощью тегов `<applet>` или `<object>`. Такой подход позволяет прямо в коде HTML-документа изменять характеристики апплета без его дополнительной компиляции. Количество одновременно используемых тегов `<param>` может быть больше одного, и для каждого из них задается пара «имя/значение» через атрибуты `name` и `value`.

Апплетом называется программа, которая выполняется в составе браузера или под управлением специальной программы для ее просмотра. Апплет, как правило, пишется на языке Java, поэтому часто можно встретить сочетание «Java-апплет».

Синтаксис:

```
<param name="имя" value="значение">  
<param name="имя" value="значение" />
```

Атрибуты:

name – имя параметра.

type – МІМЕ-тип объекта.

value – значение параметра.

valuetype – тип значения параметра.

9.4 Описание метаинформации сайта

Тег `<meta>` определяет метатеги, которые используются для хранения информации предназначенной для браузеров и поисковых систем. Например, механизмы поисковых систем обращаются к метатегам для получения описания сайта, ключевых слов и других данных. Разрешается использовать более чем один метатег; все они размещаются в контейнере `<head>`. Как правило, атрибуты любого метатега сводятся к парам «имя=значение», которые определяются ключевыми словами `content`, `name` или `http-equiv`.

Синтаксис:

```
<head>  
    <meta content="...">  
</head>
```

Атрибуты:

`charset` – задает кодировку документа.

`content` – устанавливает значение атрибута, заданного с помощью `name` или `http-equiv`.

`http-equiv` – предназначен для конвертирования метатега в заголовки HTTP.

`name` – имя метатега, также косвенно устанавливает его предназначение.

ЛИТЕРАТУРА

1 The World Wide Web Consortium / Официальный сайт консорциума. – 1991. – Режим доступа: <http://www.w3.org/>. – Дата доступа: 25.04.2015.

2 Верстка веб-страниц / Сайт для верстальников web-страниц на HTML и CSS. – 2002. – Режим доступа: <http://htmlbook.ru/>. – Дата доступа: 14.12.2014.

3 Петюшкин, А. В. HTML в Web-дизайне / А. В. Петюшкин. – СПб. : БХВ-Петербург. – 2004. – 400 с. : ил.

4 Нэш, К. Война Браузеров / К. Нэш // Сети/network world. – № 01. – 1997.

5 Дронов, В. HTML 5, CSS 3 и Web 2. Разработка современных web-сайтов / В. Дронов. – СПб. : БХВ-Петербург, 2011. – 416 с.: ил. – (Профессиональное программирование).

6 Макфарланд, Д. С. Большая книга CSS3 / Д. С. Макфарланд ; пер. с англ. Н. Вильчинского. – 3-е изд. – СПб. : Питер, 2014. – 608 с.

7 Фрэйн, Б. HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Б. Фрэйн ; пер. с англ. В. Черника. – СПб. : Питер, 2014. – 298 с.

Учебное электронное издание комбинированного распространения

Учебное издание

Тихоненко Татьяна Владимировна
Лукьяненко Владимир Олегович

ОСНОВЫ WEB-ПРОГРАММИРОВАНИЯ

Пособие

по одноименному курсу

для студентов специальности 1-40 04 01

«Информатика и технологии программирования»

дневной формы обучения

Электронный аналог печатного издания

Редактор

Т. Н. Мисюрова

Компьютерная верстка

Е. Б. Яцук

Подписано в печать 13.09.16.

Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».

Ризография. Усл. печ. л. 7,21. Уч.-изд. л. 7,46.

Изд. № 32.

<http://www.gstu.by>

Издатель и полиграфическое исполнение

Гомельский государственный

технический университет имени П. О. Сухого.

Свидетельство о гос. регистрации в качестве издателя

печатных изданий за №1/273 от 04.04.2014 г.

246746, г. Гомель, пр. Октября, 48