ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Б1.В.07 Программирование веб-приложений

Направление подготовки (специальность) 09030303 **"**Информационная безопасность автоматизированных систем **"**

Профиль образовательной программы "Информационная безопасность автоматизированных систем критически важных объектов"

Форма обучения *очная*

СОДЕРЖАНИЕ

1. Конспект лекций ……..4

1.1 Лекция № 1-2 *Архитектура WWW. Обзор Web-технологий……4*

1.2 Лекция № 3-4 *Введение в HTML. Структура HTML документа……5*

1.3 Лекция № 5-6 *Форматирование текста* Интерактивная форма ……*8*

1.4 Лекция № 7-8 Ссылки. Графика.……*9*

1.5 Лекция № 9 Таблицы в HTML. Табличная верстка. Интерактивная форма……..*13*

1.6 Лекция № 10 Интерактивные формы HTML. Фреймы…… *17*

1.7 Лекция №11 Каскадные таблицы стилей CSS. Форматирование блоков. Форматирование текста. Интерактивная форма.……21

1.8 Лекция № 12 Слои. CSS верстка**……..**……*25*

1.9 Лекция № 13-14 Введение в JavaScript. Синтаксис языка. Объектная модель JavaScript.……..……*32*

1.10 Лекция № 15 Типы данных. Операторы JavaScript.……..……*34*

1.11 Лекция № 16 Обработка событий. Примеры эффективного 2программирования на JS.……..……*36*

1.12 Лекция № 17 Основы DHTML. Интерактивная форма.……..……*37*

2. Методические указания по проведению практических занятий ……41

2.1 Практическое занятие № ПЗ-1 *Обзор Web-технологий……41*

2.2 Практическое занятие № ПЗ-2 *Структура HTML документа……..……45*

2.3 Практическое занятие № ПЗ-3 *Форматирование текста……..47*

2.4 Практическое занятие № ПЗ-4 *Ссылки. Графика……..49*

2.5 Практическое занятие № ПЗ-5 *Таблицы в HTML……..53*

2.6 Практическое занятие № ПЗ-6 *Табличная верстка……..57*

2.7 Практическое занятие № ПЗ-7 *Интерактивные формы HTML……..60*

2.8 Практическое занятие № ПЗ-8 *Фреймы……..63*

2.9 Практическое занятие № ПЗ-9 *Каскадные таблицы стилей CSS……64*

2.10 Практическое занятие № ПЗ-10 *Форматирование блоков. Форматирование текста……..65*

2.11 Практическое занятие № ПЗ-11 *Слои*……....68

2.12 Практическое занятие № ПЗ-12 *CSS верстка……..76*

2.13 Практическое занятие № ПЗ-13 *Структура HTML документа……82*

2.14 Практическое занятие № ПЗ-14 *Синтаксис языка. Объектная модель JavaScript.* ……93

2.15 Практическое занятие № ПЗ-15 *Типы данных. Операторы JavaScript……..99*

2.16 Практическое занятие № ПЗ-16 *Типы данных. Операторы JavaScript……..102*

2.17 Практическое занятие № ПЗ-17 *Обработка событий……..107*

1. КОНСПЕКТ ЛЕКЦИЙ

1. 1 Лекция №1-2 (4 часа).

Тема: «Архитектура WWW. Обзор Web-технологий.»

1.1.1 Вопросы лекции:

1) Архитектура WWW.

2) Программы для работы с сервером.

3) История развития гипертекста.

1.1.2 Краткое содержание вопросов:

1) Архитектура WWW.

WWW построена по хорошо известной схеме "клиент-сервер". Программа-клиент выполняет функции интерфейса пользователя и обеспечивает доступ практически ко всем информационным ресурсам Internet. В этом смысле она выходит за обычные рамки работы клиента только с сервером определенного протокола. База данных HTML-документов -- это часть файловой системы, которая содержит текстовые файлы в формате HTML и связанные с ними графику и другие ресурсы. Особое внимание хотелось бы обратить на документы, содержащие элементы экранных форм. Эти документы реально обеспечивают доступ к внешнему программному обеспечению.

2) Программы для работы с сервером

Прикладное программное обеспечение, работающее с сервером, можно разделить на программы-шлюзы и прочие. Шлюзы -- это программы, обеспечивающие взаимодействие сервера с серверами других протоколов, например ftp, или с распределенными на сети серверами Oracle. Прочие программы -- это программы, принимающие данные от сервера и выполняющие какие-либо действия: получение текущей даты, реализацию графических ссылок, доступ к локальным базам данных или просто расчеты.

3) История развития гипертекста.

К 1989 году гипертекст представлял новую, многообещающую технологию, которая имела относительно большое число реализаций с одной стороны, а с другой стороны делались попытки построить формальные модели гипертекстовых систем, которые носили скорее описательный характер и были навеяны успехом реляционного подхода описания данных. Идея Т. Бернерс-Ли заключалась в том, чтобы применить гипертекстовую модель к информационным ресурсам, распределенным в сети, и сделать это максимально простым способом. Он заложил три краеугольных камня системы из четырех существующих ныне, разработав:

1. язык гипертекстовой разметки документов HTML (HyperText Markup Language);
2. универсальный способ адресации ресурсов в сети URL (Universal Resource Locator);
3. протокол обмена гипертекстовой информацией HTTP (HyperText Transfer Protocol).
4. Позже команда NCSA добавила к этим трем компонентам четвертый:
5. универсальный интерфейс шлюзов CGI (Common Gateway Interface).

Идея HTML - пример чрезвычайно удачного решения проблемы построения гипертекстовой системы при помощи специального средства управления отображением. На разработку языка гипертекстовой разметки существенное влияние оказали два фактора: исследования в области интерфейсов гипертекстовых систем и желание обеспечить простой и быстрый способ создания гипертекстовой базы данных, распределенной на сети.

В 1989 году активно обсуждалась проблема интерфейса гипертекстовых систем, т.е. способов отображения гипертекстовой информации и навигации в гипертекстовой сети. Значение гипертекстовой технологии сравнивали со значением книгопечатания. Утверждалось, что лист бумаги и компьютерные средства отображения/воспроизведения серьезно отличаются друг от друга, и поэтому форма представления информации тоже должна отличаться. Наиболее эффективной формой организации гипертекста были

признаны контекстные гипертекстовые ссылки, а кроме того было признано деление на ссылки, ассоциированные со всем документом в целом и отдельными его частями.

С момента разработки первой версии языка (HTML 1.0) прошло уже пять лет. За это время произошло довольно серьезное развитие языка. Почти вдвое увеличилось число элементов разметки, оформление документов все больше приближается к оформлению качественных печатных изданий, развиваются средства описания не текстовых информационных ресурсов и способы взаимодействия с прикладным программным обеспечением. Совершенствуется механизм разработки типовых стилей. Фактически, в настоящее время HTML развивается в сторону создания стандартного языка разработки интерфейсов как локальных, так и распределенных систем.

1. 2 Лекция №3-4 (4 часа).

Тема: «Введение в HTML. Структура HTML документа.»

1.2.1 Вопросы лекции:

1) Основы HTML документа.

2) Закрывающие теги.

3) Незакрывающиеся теги.

4) Обязательные теги.

1.2.2 Краткое содержание вопросов:

1) Основы HTML, теги.

HTML - это обычный, текстового вида файл, в котором то, что мы обычно видим на страничках, перемежается невидимым для просмотра из браузера кодом. Вот этот-то невидимый код и есть язык разметки HTML.

HTML - это не язык программирования, - он служит лишь для разметки странички, придания определенного вида тому или иному элементу, будь то таблица, текст или картинки.

Осуществляется это путем присвоения каждому элементу своих параметров, которые распознает браузер. Параметры эти могут быть заданы как для одного, так и для группы или типа элементов. Тип элементов может быть таким: таблицы, ячейки, ссылки, текст и т.п. То есть что-то, что можно назвать одним термином. Отдельные свойства можно присваивать и выбранным элементам персонально.

2) Закрывающиеся теги.

Параметры отображения элементов задаются при помощи тегов, в которых и задается желаемый вид того или иного элемента нашей странички.

Теги-контейнеры.

Прежде всего стоит определить то, как эти самые теги располагаются. Дело в том, что теги в большинстве своем состоят из двух частей - это открывающий (он же - содержащий параметры) и замыкающий, то есть, "конец" тега. Заданные в теге параметры действуют только между его началом и концом, то есть, только внутри тега:

<начало 1-го тега>
текст текст текст текст текст текст текст текст
<начало 2-го тега>

текст текст текст текст текст текст текст текст
<начало 3-го тега>
текст текст текст текст текст текст текст текст
<Конец 2-го тега>
<Конец 2-го тега>
<Конец 1-го тега>

Также следует заметить, что некоторые параметры вложенных тегов могут воздействовать и на вышестоящие, "старшие" теги. Например, это типично для таблиц, вложенных одна в другую.

Если у внутренней таблицы размер задан 100 пикселов*, то у внешней не может быть размера меньше 100 пикселов + толщина рамки. Таким образом вложенная таблица как бы "распирает" ту, в которую она вложена. При этом, даже если у нее не задан размер, а количество текста в ней значительно, она будет распирать "старшую" таблицу на такой размер, который необходим для отображения соответствующего количества строк текста. Это называется "обратная связь".

3) Незакрывающиеся теги.

Это теги переноса -
, принудительно перекидывающие все, что располагается после них, в новую строку. Благодаря этому тегу можно писать стихи столбиками.

Вот такая строчка:

текст текст текст текст
 текст текст текст
 текст текст текст

При просмотре ее из браузера выдаст вам столбик -

текст текст текст
текст текст текст
текст текст текст

Тег этот может располагаться в коде как в середине текста (пример выше), так и заканчивать строку:

текст текст текст
текст текст текст

Или же разделять строки:

текст текст текст

текст текст текст

Браузеру не важно положение тега - все отступы и "сдвиги" нужны только для упрощения понимания и ориентирования в коде. Конечно, количество пробелов внутри кода увеличивает размер HTML-странички, поэтому злоупотреблять ими не стоит. Но от того, что вы выделите какие-либо теги определенным образом, хуже не станет. Все пробелы больше одного браузеры просто "не замечают". То есть, сколько ни ставьте вы пробелов в тексте, браузер отобразит все равно с одним пробелом:

в коде написано:
текст текст - браузер покажет:

текст текст

Впрочем, если "ну очень хочется", то пробелов можно наставить принудительно. Делается это размещением в коде "; ".

; - обозначение пробела. Неразрывного пробела. То есть в этом месте будет пустое пространство. Но если простой текст с пробелами браузер может перенести на другую строку на месте любого пробела, то слова, разделенные ; , он воспримет как единое целое. Вот потому не стоит злоупотреблять "гибкостью" и слишком часто использовать обозначение ; . Не стоит использовать его и для "выравнивания" текста внутри таблицы. Для этого есть более простые и правильные способы. Не рекомендуется уптреблять ; больше двух-трех раз подряд. Можете считать это правилом, поскольку при большем количестве символов наверняка проще прописать требуемое выравнивание в тег нужного элемента.

Еще один тег, не нуждающийся в закрытии, это тег <hr>

Это элемент "отчеркивание", - просто черта от края до края тега, внутри которого она применяется. Но ее можно и ограничить, например, указав в ее теге размер:

<hr align="center" width="50%">

Код содержит команды: выравнять по центру, размер сделать в 50% доступной ширины.

4) Обязательные теги.

<!DOCTYPE >

- обозначает тип документа и формат. Облегчает распознавание другими программами этого файла. Указывает в начале версию стандарта HTML и язык документа.

<html>

- обозначает начало ХТМЛ. Благодаря этому браузер убеждается, что открываемое ею именно html, а не картинка или MP3.

<head>

-обозначает начало "головы" - контейнер, для прикрепления стилей, скриптов и многих других важных вещей.

<title>

Заголовок

</title>

- тут вписывается название листа в окна браузера - там, где "свернуть, распахнуть и закрыть"

<body>

- контейнер, в котором хранится вся видимая на сайте информация

1. 3 Лекция №5-6 (4 часа).

Тема: «Форматирование текста»

1.3.1 Вопросы лекции:

1. Теги заголовков.

2. Теги для форматирования текстов.

3. Абзацы, средства переноса текста

1.3.2 Краткое содержание вопросов:

1. Теги заголовков

Заголовки являются важным элементом веб-страницы, они помогают упорядочить текст, сформировать его структуру. В спецификации HTML существует шесть уровней заголовков, благодаря чему можно легко выделять темы и подтемы.

Заголовки являются блочными элементами, всегда начинаются с новой строки. Браузер автоматически добавляет перед заголовком и после него пустую строку.

Текст в заголовках влияет на индексацию сайта поисковыми системами, так как многие роботы обращают внимание именно на содержимое заголовков сайта, поэтому лучше всегда использовать эти теги, меняя внешний вид и размер с помощью стилей.

При использовании заголовков необходимо учитывать их иерархию, т.е. за <h1> должен следовать <h2> и т.д. Также не допускается вложение других тегов в теги <h1>...<h6>.

Тег <h1>

Заголовок самого верхнего уровня, выделяется самым крупным шрифтом. На странице тег <h1> рекомендуется использовать только один раз, по возможности частично дублируя заглавие страницы. Тег должен быть уникальным для каждой страницы сайта. Рекомендуется прописывать тег в начале статьи, используя ключевое слово в тексте заголовка. Размер шрифта в браузере равен 2em, верхний и нижний отступ по умолчанию 0.67em.

Тег <h2>

Им обозначаются подзаголовки тега <h1>. Размер шрифта в браузере равен 1.5em, верхний и нижний отступ по умолчанию 0.83em.

Тег <h3>

Показывает подзаголовки тега <h2>. Размер шрифта в браузере равен 1.17em, верхний и нижний отступ по умолчанию 1em.

Теги <h4>, <h5>, <h6>

Обозначают подзаголовки четвёртого, пятого и шестого уровня. Размер шрифта в браузере равен 1em / 0.83em / 0.67em, верхний и нижний отступ по умолчанию 1.33em / 1.67em / 2.33em соответственно.

2. Теги для форматирования текста

Тег <b>

Также используется для задания жирного начертания шрифта. Относится к тегам физической разметки.

Тег

Отображает шрифт курсивом, придавая тексту значимость.

Тег <i>

Используется для отображения шрифта курсивом.

Тег <small>

Уменьшает размер шрифта на единицу по отношению к обычному тексту.

Тег

Задаёт полужирное начертание шрифта, относится к тегам логической разметки, указывая браузеру на важность текста.

Тег <sub>

Используется для создания нижних индексов. Сдвигает текст ниже уровня строки, уменьшая его размер.

Тег <sup>

Используется для создания степеней. Сдвигает текст выше уровня строки, уменьшая его размер.

Тег <ins>

Выделяет текст в новой версии документа, подчёркивая его.

Тег <del>

Перечёркивает текст. Используется для выделения текста, удаленного из документа.

3. Абзацы, средства переноса текста

Тег <p>

Разбивает текст на отдельные абзацы, отделяя друг от друга пустой строкой. Браузер автоматически добавляет верхний и нижний отступ, равный 1em, при этом отступы соседних абзацев «схлопываются».

Тег

Переносит текст на следующую строку, создавая разрыв строки.

Тег <hr>

Используется для разделения контента на веб-странице. Отображается в виде горизонтальной линии.

Лекция №7-8 (4 часа)

1.4 Лекция №9 (2 часа)

Тема: «Таблицы в HTML. Табличная верстка»

1.4.1 Вопросы лекции:

1) Описание таблиц.

2) Столбы и строки

3) Дополнительные возможности.

4) Пример таблицы.

5) Параметры WIDTH и HEIGHT

6) Параметр ALIGN

1.4.3 Краткое содержание вопросов.

1) Описание таблицы.

Описание таблиц должно располагаться внутри раздела документа <BODY>. Документ может содержать произвольное число таблиц, причем допускается вложенность таблиц друг в друга. Каждая таблица должна начинаться тэгом <TABLE> и завершаться тэгом </TABLE>. Внутри этой пары тегов располагается описание содержимого таблицы. Любая таблица состоит из одной или нескольких строк, в каждой из которых задаются данные для отдельных ячеек.

Каждая строка начинается тэгом <TR> (Table Row) и завершается тэгом </TR>. Отдельная ячейка в строке обрамляется парой тегов <TD> и </TD> (Table Data) или <TH> и </TH> (Table Header). Тег <TH> используется обычно для ячеек-заголовков таблицы, а <TD> — для ячеек-данных. Различие в использовании заключается лишь в типе шрифта, используемого по умолчанию для отображения содержимого ячеек, а также расположению данных внутри ячейки. Содержимое ячеек типа <TH> отображается полужирным (Bold) шрифтом и располагается по центру (ALIGN=CENTER, VALIGN=MIDDLE). Ячейки, определенные тэгом <TD> по-умолчанию отображают данные, выровненные влево (ALIGN=LEFT) и посередине (VALIGN=MIDDLE) в вертикальном направлении.

Тэги <TD> и <TH> не могут появляться вне описания строки таблицы <TR>. Завершающие коды </TR>, </TD> и </TH> могут быть опущены. В этом случае концом описания строки или ячейки является начало следующей строки или ячейки, или конец таблицы. Завершающий тег таблицы </TABLE> не может быть опущен.

2) Столбы и строки.

Количество строк в таблице определяется числом открывающих тегов <TR>, а количество столбцов — максимальным количеством <TD> или <TH> среди всех строк. Часть ячеек могут не содержать никаких данных, такие ячейки описываются парой следующих подряд тегов — <TD>, </TD>. Если одна или несколько ячеек, располагающихся в конце какой-либо строки, не содержат данных, то их описание может быть опущено, а браузер автоматически добавит требуемое количество пустых ячеек. Отсюда следует, что построение таблиц, в которых в разных строчках располагается различное количество столбцов одного и того же размера, не разрешается.

3) Дополнительные возможности.

Таблица может иметь заголовок, который заключается в пару тегов <CAPTION> и </CAPTION>. Описание заголовка таблицы должно располагаться внутри тегов <TABLE> и </TABLE> в любом месте, однако вне области описания любого из тегов <TD>, <TH> или <TR>. Согласно спецификациям языка HTML расположение описания заголовка регламентировано более строго: оно должно располагаться сразу же после тега <TABLE> и до первого <TR>. Мы рекомендуем придерживаться этого правила.

По умолчанию текст заголовка таблицы располагается над ней (ALIGN=TOP) и центрируется в горизонтальном направлении.

Перечисленные теги могут иметь параметры, число и значения которых различны. Однако в простейшем случае теги используются без параметров, которые принимают значения по умолчанию.

4) Пример таблицы.

Этих сведений вполне достаточно для построения элементарных таблиц. Приведем пример простейшей таблицы, состоящей из двух строк и двух столбцов.

<HTML>

```
<HEAD>
<TITLE>Пример простейшей таблицы</TITLE>
</HEAD>
<BODY>
<TABLE BORDER>
<TR>
<TD>Ячейка 1 строки 1</TD>
<TD>Ячейка 2 строки 1</TD>
</TR>
<TR>
<TD>Ячейка 1 строки 2</TD>
<TD>Ячейка 2 строки 2</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

5) Параметры WIDTH и HEIGHT

При отображении таблиц их ширина и высота автоматически вычисляются браузером и зависят от многих факторов: значений параметров, заданных в описании всего документа, данной таблицы, отдельных ее строк и ячеек, содержимого ячеек, а также параметров, задаваемых при просмотре документа в том или ином браузере, например, типа и размеров шрифта, размеров окна просмотра и др. При отображении расчет размеров таблиц выполняется автоматически с учетом этих факторов, при этом делается попытка представить таблицу в наиболее удобном виде — расположить таблицу так, чтобы она помещалась и окне просмотра. Общая схема просмотра больших документов, как правило, сводится к линейной прокрутке содержимого документа по вертикали и чтении текста, перемежаемого различными таблицами, изображениями и т. п. Это относится как к HTML-документам, так и к обычным документам, создаваемым в любых текстовых редакторах. Большинство как текстовых редакторов (например, Microsoft Word), так и HTML-браузеров автоматически форматируют текст так (если возможно), чтобы длина строк не превосходила ширину окна просмотра. Это позволяет избежать необходимости горизонтальной прокрутки документа. Аналогичные действия предпринимаются браузерами с таблицами — по возможности форматировать их таким образом, чтобы ширина таблицы не превосходила ширины окна просмотра. Можно сделать вывод, что ширина таблиц является более важным, первостепенным параметром, расчет которого выполняется в первую очередь по сравнению с высотой.

В большинстве случаев динамическое определение размеров таблицы дает в результате эстетически соразмерное изображение с эффективным использованием реальной площади окна просмотра. Однако бывает необходимо принудительно указывать ширину или высоту таблицы. Для этой цели используются параметры WIDTH (ширина таблицы) и HEIGHT (высота таблицы) тега <TABLE>. Форма записи: WIDTH=num или WIDTH=num%, где num — численное значение ширины всей таблицы в пикселах или в

процентах от всего размера окна. Заметим, что допустимо задавать значения, большие 100%, хотя трудно представить себе случай, где это необходимо. Пример:

<TABLE WIDTH=200>.

Аналогичные параметры могут задаваться и для отдельных ячеек. Заметим, что задание конкретного значения параметра, например WIDTH=200, не означает, что таблица в любом случае будет иметь указанную ширину, а лишь определяет рекомендуемую ширину, которая будет выдержана по возможности. Поясним это на примерах. Пусть имеется таблица, которая в данных условиях по умолчанию имела бы ширину, меньшую заданной. В этом случае браузер увеличит ширину таблицы до требуемой путем пропорционального расширения всех колонок таблицы. При сужении окна просмотра ширина таблицы изменяться не будет, и, возможно, для ее просмотра потребуется горизонтальная прокрутка. Если же таблица по умолчанию имеет ширину, большую заданной, то браузер сделает попытку уменьшить ее ширину за счет, во-первых, сокращения ширины отдельных колонок, для которых заданная ширина больше необходимой, во-вторых, разбиением текста в отдельных ячейках на несколько строк с увеличением высоты таблицы. Эти действия могут не обеспечить требуемого размера таблицы, и тогда она будет иметь минимально возможную ширину. Такие же действия предпринимаются для таблиц, у которых не указаны размеры, при сужении окна просмотра.

Конкретные алгоритмы настройки таблиц для различных браузеров могут несколько отличаться.

6) Параметр ALIGN

Данный параметр тега <TABLE> определяет горизонтальное расположение таблицы в области просмотра. Допустимые значения — LEFT (выравнивание влево) и RIGHT (выравнивание вправо). По умолчанию таблицы выровнены по левому краю. Заметим, что среди допустимых значений нет типичного значения для параметра выравнивания — CENTER. В некоторых источниках по языку HTML значение CENTER (по центру) приводится в качестве допустимого в данном случае. Это соответствует спецификации HTML, но на практике и Netscape Navigator, и Microsoft Internet Explorer реализуют только два значения. Дело в том, что присутствие параметра ALIGN в тэге <TABLE> не только определяет месторасположение таблицы, но и разрешает выполнить обтекание таблицы текстом с противоположной стороны аналогично обтеканию картинок. Обтекание таблицы текстом с двух сторон не предусматривается ни в каких случаях. Для более точного управления обтеканием следует использовать тег
 с параметром CLEAR так же, как это выполняется для <IMG>. Если параметр ALIGN опущен, то место справа и/или слева от таблицы всегда будет пустым независимо от ее ширины. Если таблица не требует обтекания текстом, то можно добиться ее расположения по центру окна просмотра. Для этого, например, можно все описание таблицы поместить внутри пары тегов <CENTER> и </CENTER>.

1.5 Лекция № 10 (2 часа)

Тема: «Интерактивные формы HTML. Фреймы»

1.5.1 Вопросы лекции:

1) Простейшие формы.

2) Флажок (checkbox)

3) Переключатель(radio)

4) Кнопка сброса формы(Reset)

5) Выпадающий список (select)

6) Текстовое поле (text)

7) Поле для ввода пароля (password)

8) Многострочное поле ввода текста (textarea)

9) Скрытое текстовое поле

10) Кнопка отправки формы (submit)

1.5.2 Краткое содержание вопросов:

1) Простейшие формы.

Зачастую на Web – сайтах можно встретить страницы с размещенными на них HTML - формами. Веб-формы – удобный способ получения информации от посетителей вашего сайта. Пример тому – гостевая книга, – которая обеспечивает обратную связь с посетителями и разработчиками сайта. Формы так же удобны и для разработчиков сайта при разработке CMS, которая позволяет поддерживать главное свойство сайта - актуальность. Данная статья посвящена основам создания HTML-форм, их обработке и способам передачи данных из экранных форм в PHP-сценарии.

Создание простой формы

Теги <form> и </form> задают начало и конец формы. Начинающий форму тег <form> содержит два атрибута: action и method. Атрибут action содержит адрес URL сценария, который должен быть вызван для обработки сценария. Атрибут method указывает браузеру, какой вид HTTP запроса необходимо использовать для отправки формы; возможны значения POST и GET.

Главное отличие методов POST и GET заключается в способе передачи информации. В методе GET параметры передаются через адресную строку, т.е. по сути в HTTP-заголовке запроса, в то время как в методе POST параметры передаются через тело HTTP-запроса и никак не отражаются на виде адресной строки.

```
<form method="post" action="../admin/add_story.php">
</form>
```

2) Флажок (checkbox)

Флажки checkbox предлагаю пользователю ряд вариантов, и разрешает выбор нескольких из них.

```
<input name="Имя переключателя" type="Тип" value="Значение">
```

Группа флажков состоит из элементов <input>, имеющих одинаковые атрибуты name и type(checkbox). Если вы хотите, чтобы элемент был отмечен по умолчанию необходимо пометить его как checked. Если элемент выбран, то сценарию поступит строка имя=значение, в противном случае в обработчик формы не придет ничего, т.е. не выбранные флажки вообще никак не проявляют себя в переданном наборе данных.

Пример:

```
<input name="mycolor" type="checkbox" value="red" checked>Красный(выбран по умолчанию)
```

<input name="mycolor" type="checkbox" value="blue">Синий
<input name="mycolor" type="checkbox" value="black">Черный
<input name="mycolor" type="checkbox" value="white">Белый

3) Переключатель(radio)

Переключатели radio предлагают пользователю ряд вариантов, но разрешает выбрать только один из них.

<input name="Имя переключателя" type="Тип" value="Значение">

Переключатель (radio) имеет атрибуты name, type и value. Атрибут name задает имя переключателя, type задает тип radio, а атрибут value задает значение. Если пользователь выберет переключатель, то сценарию будет передана строка имя=значение. При необходимости можно указать параметр checked, который указывает на то, что переключатель будет иметь фокус (т.е. будет отмечен по умолчанию) при загрузке страницы. Переключатели также можно объединять в группы, для этого они должны иметь одно и тоже имя.

Пример:

<input name="mycolor" type="radio" value="white"> Белый
<input name="mycolor " type="radio" value="green" checked> Зеленый (выбран по умолчанию)
<input name="mycolor " type="radio" value="blue"> Синий
<input name="mycolor " type="radio" value="red"> Красный
<input name="mycolor " type="radio" value="black"> Черный

4) Кнопка сброса формы(Reset)

<input type="Тип" name="Имя кнопки" value="Надпись на кнопке">

При нажатии на кнопку сброса(reset), все элементы формы будут установлены в то состояние, которое было задано в атрибутах по умолчанию, причем отправка формы не производиться.

Пример:

<input type="reset" name="Reset" value="Очистить форму">

5) Выпадающий список (select)

Тэг <select> представляет собой выпадающий или раскрытый список, при этом одновременно могут быть выбраны одна или несколько строк.

Список начинается с парных тегов <select></select>. Теги <option></option> позволяют определить содержимое списка, а параметр value определяет значение строки. Если в теге <option> указан параметр selected, то строка будет изначально выбранной. Параметр size задает, сколько строк будет занимать список. Если size равен 1, то список будет выпадающим. Если указан атрибут multiple, то разрешено выбирать несколько элементов из списка(при size = 1 не имеет смысла).

<select name="Имя списка" size = “Размер” multiple>
<option value=”Значение”>Отображаемый текст в списке</option>
</select>

При передаче данных выпадающего списка сценарию передается строка имя=значение, а при раскрытом списке передается строка имя=значение1&имя=значение2&имя=значениеN.

6) Текстовое поле (text)

Позволяет пользователям вводить различную информацию.

<input type="Тип" name="Имя поля" size="Размер" maxlength="Макс. количество символов">

При создании обычного текстового поля размером size и максимальной допустимой длины maxlength символов, атрибут type принимает значение text. Если указан параметр value, то поле будет содержать отображать value-текст. При создании поля не забывайте указывать имя поля, т.к. этот атрибут является обязательным.

Пример:

<input type="text" name="txtName" size="10" maxlength="5" value="Текст по умолчанию">

7) Поле для ввода пароля (password)

Полностью аналогичен текстовому полю, за исключением того что символы, набираемые пользователем, не будут отображаться на экране.

Пример:

<input type="password" name="txtName" size="10" maxlength="5">

8) Многострочное поле ввода текста (textarea)

Многострочное поле ввода текста позволяет отправлять не одну строку, а сразу несколько. По умолчанию тег создает пустое поле шириной в 20 символов и состоящее из двух строк.

<textarea name="Имя поля" cols="Ширина поля " rows="Число строк">Текст</textarea>

Многострочное поле ввода текста начинается с парных тегов <textarea></textarea>. Тэг name задает имя многострочного поля. Также можно указать ширину поля(cols) и число строк(rows). При необходимости можно указать атрибут readonly, который запрещает редактировать, удалять и изменять текст, т.е. текст будет предназначен только для чтения. Если необходимо чтобы текст был изначально отображен в многострочном поле ввода, то его необходимо поместить между тэгами <textarea></textarea>.

Пример:

<textarea name="txtArea" cols="15" rows="10" readonly> Текст, который изначально будет отображен в многострочном поле ввода и который нельзя изменять, т.к. указан атрибут readonly </textarea>

9) Скрытое текстовое поле

Позволяет передавать сценарию какую то служебную информацию, не отображая её на странице.

<input name="Имя" type="Тип" value="Значение">

Скрытое поле начинается с тега <input>, атрибуты которого являются name, type и value. Атрибут name задает имя поля, type определяет тип поля, а атрибут value задает значение поля.

Пример:

<input name="email" type="hidden" value="spam@nospam.ru">

10) Кнопка отправки формы (submit)

Служит для отправки формы сценарию.

<input type="Тип" name="Имя кнопки" value="Текст кнопки">

При создании кнопки для отправки формы необходимо указать 2 атрибута: type=“submit” и value=”Текст кнопки”. Атрибут name необходим если кнопка не одна, а несколько и все они созданы для разных операций, например кнопки "Сохранить", "Удалить", "Редактировать" и т.д. После нажатия на кнопку сценарию передается строка имя=текст кнопки.

Кнопка для загрузки файлов (browse)

Служит для реализации загрузки файлов на сервер. Объект browse начитается с парных тегов <form></form>. Начинающий тэг <form> содержит необходимый атрибут encrypt. Атрибут encrypt принимает значение multipart/form-data, который извещает сервер о том, что вместе с обычной информацией посылается и файл. При создании текстового поля также необходимо указать тип файла – “file”.

```
<form enctype="multipart/form-data" action="upload.php" method="post">
 Загрузить файл: <input name="my_file" type="file">
  <input type="submit" value="Отправить">
</form>
```

Рамка (fieldset)

Объект fieldset позволяет вам нарисовать рамку вокруг объектов. Имеет закрывающий тэг </fieldset>. Заголовок указывается в тэгах <legend></legend>. Основное назначение объекта – задавание различных стилей оформления.

Пример:

```
<fieldset>
<legend>Программное обеспечение(заголовок рамки)</legend>
Текст, который будет помещен внутри рамки.</fieldset>
```

1.6 Лекция №11 (2 часа)

Тема: «Каскадные таблицы стилей CSS. Форматирование блоков. Форматирование текста.»

1.6.1 Вопросы лекции:

1) Описание каскадных таблиц.

2) История возникновения.

3) Методы и синтаксис

4) Синтаксис таблицы стилей

5) Определение правил CSS

1.6.2 Краткое содержание вопросов:

1) Описание каскадных таблиц.

Каскадные (многоуровневые) таблицы стилей - cascading style sheets (CSS) - это мощный стандарт на основе текстового формата, определяющий представление данных в броузере.

Если формат HTML предоставляет информацию о составе документа, то таблицы стилей сообщают как он должен выглядеть. Таким образом каскадные таблицы стилей дают возможность хранить содержимое отдельно от его представления.

Стиль включает все типы элементов дизайна: шрифт, фон, текст, цвета ссылок, поля и расположение объектов на странице.

CSS разрабатывались так, чтобы обеспечить больший уровень контроля над размещением текста и графики.

Каскадные таблицы стилей обеспечивают должный уровень единства оформления, организации и контроля во время разработки узла, который является недостижимым с помощью одного только HTML.

CSS предполагает 3 типа таблиц стилей - встроенные, внедренные (внутренние) и связанные (внешние).

2) История возникновения.

Впервые идея форматирования HTML-документов с помощью CSS была рекомендована Консорциумом W3C в 1996 году. Эта рекомендация, которая была обновлена в 1998 году, используется Web-разработчиками и по сей день.

Что значит слово "каскадный"?

Термин "каскадный" означает, что в одной странице HTML могут использоваться разные стили. Броузер, поддерживающий таблицы стилей, будет следовать их порядку (как по каскаду), интерпретируя информацию стилей.

Это означает, что вы можете использовать все три типа стилей, и броузер будет интерпретировать сначала связанные, затем внедренные и, наконец, встроенные стили. Даже если ко всему узлу будут применены образцы стилей, можно будет управлять отдельными аспектами страниц с помощью внедренных стилей, а отдельными областями внутри этих страниц - с помощью встроенных стилей.

Другой аспект каскадирования - наследование (inheritance). Наследование означает, что если не указано иное, то конкретный стиль будет унаследован другими элементами страницы HTML. Например, если вы примените определенный цвет текста в теге <p>, то все теги внутри этого абзаца наследуют этот цвет, если не оговорено иное.

3) Методы и синтаксис

Существует ряд методов, с помощью которых таблицы стилей могут применяться к документу HTML. Синтаксис соответствует реальной структуре информации, содержащейся внутри таблицы стиля.

Существует три метода для применения таблицы стилей к документу HTML:

Встроенный (Inline). Этот метод позволяет взять любой тег HTML и добавить к нему стиль. Использование встроенного метода предоставляет максимальный контроль над всеми свойствами Web-страницы. Предположим, вы хотите задать внешний вид отдельного абзаца. Вы можете просто добавить атрибут style к тегу абзаца, и броузер отобразит этот абзац с помощью параметров стиля, добавленного в код.

Внедренный (Embedded). Внедрение позволяет контролировать всю страницу HTML. При использовании тега <style>, помещенного внутри раздела <head> страницы HTML, в код вставляются детализированные атрибуты стиля, которые будут применяться ко всей странице.

Связанный (Linked или External). Связанная таблица стилей - мощный инструмент, который позволяет создавать образцы стилей (master styles), которые можно затем применять ко всему узлу. Основной документ таблицы стилей (расширение .css) создается Web-дизайнером. Этот документ содержит стили, которые будут едиными для всего Web-узла (неважно, содержит одну страницу или тысячи страниц). Любая страница, связанная с этим документом, будет использовать указанные стили.

4) Синтаксис таблицы стилей

Таблицы стилей строятся в соответствии с определенным порядком (синтаксисом), в противном случае они не могут нормально работать.

Синтаксис всех методов, используемых для применения стилей к документа HTML, практически одинаков. Таблицы стилей составляются из определенных частей. Эти части включают следующие элементы:

Указатель (Selector). Указатель является элементом, к которому будут применяться назначаемые вами атрибуты. Это может быть просто тег типа заголовка (H1) или абзаца (P). Таблицы стилей позволяют использовать различные объекты, включая классы, которые будут кратко обсуждаться далее.

Свойство (Property). Свойство определяет указатель. Например, если в качестве указателя выбран абзац, вы можете включить свойства, определяющие этот указатель. В свойства входят такие элементы, как поля, шрифты и фоновые изображения. В таблицах стилей существует много свойств, которые можно использовать для того, чтобы определить указатель.

Значение (Value). Значения определяют свойства. Предположим, что у вас есть заголовок первого уровня H1(указатель) и вы включаете свойство type family (семейство шрифта). Шрифт, который на самом деле будет применен к указанному фрагменту, задается значением этого свойства.

5) Определение правил CSS

Итак, каскадная таблица стилей - это набор правил форматирования тегов HTML. Например, для того, чтобы цвет фона Web-страницы сделать черным, необходимо объявить следующее правило форматирования:

```
body{background:black}
```

Будьте внимательны! По умолчанию цвет шрифта - черный!

В данном случае объявлено правило форматирования тега body, а именно - свойству стиля background присвоено значение black (черный). В результате применения этого правилацвет фона всего документа изменится на черный.

Обратите внимание: в таблице стилей теги HTML не заключаются в круглые скобки.

Свойства CSS должны находиться в фигурных скобках.

Для каждого тега HTML можно указать не одно, а несколько свойств стиля.

Изменим с помощью CSS не только цвет фона Web-страницы, но и цвет шрифта (на белый).

```
body{background:black;color:white}
```

Формат самого правила не имеет значения, главное - правило должно читаться удобно и легко. Например, вышеприведенное правило можно записать и так:

```
body{
background:black;
color:white}
```

Одно и то же правило сстиля можно применить сразу к нескольким различным тегам HTML-страницы. Например:

```
body,td,h1{
background:black;
color:white}
```

Встроенный стиль

Встроенный стиль применяется к любому тегу HTML с помощью атрибута style следующим образом:

```
<P style="font: 12pt Courier New"> The text in this line will
display as 12 point text using the Courier New font.
</P>
```

Или:

```
<p style="font: 12pt Arial">
The text in this line will display as 18 point text using the
 Arial font.
</p>
```

Можно добавлять встроенный стиль в любой тег HTML, в котором эта операция будет иметь смысл. Среди таких тегов можно назвать абзацы, заголовки, горизонтальные линии, якоря и ячейки таблиц. Ко всем этим элементам логично применять встроенные стили.

Существуют два тега, которые помогают применять встроенные стили к разделам страницы. Это теги (division - раздел) и (промежуток).

Эти теги определяют диапазон текста, так что все, находящееся между ними, будет оформлено с помощью нужного стиля. Единственное различие между и состоит в том, что создает принудительный разрыв строки, а - нет.

Следовательно, нужно использовать для изменения стиля любой части текста, меньшей абзаца.

Вот пример работы тега :

<div style="font-family: Garamond; font-size: 18 pt;>"AII of the

text within this section is 18 point Garamond.

AII of the text within this section is 18 point Garamond.

и тега :

 This text appears in the color red,

with no line break after the closing span tag and the rest of

the text.

This text appears in the color red, with no line break after the closing span tag and the rest of the text.

Хотя встроенные стили довольно полезны - гораздо лучше разрабатывать стандартные стили для всего Web-узла и затем применять их, используя внедренные или связанные таблицы стилей.

Внедренные стили используют тег <style>, расположенный между тегами </head> и <body> в стандартном документе HTML.

Рассмотрим пример внедренного стиля:

```
<html>
<head>
<title>Embedded Style Sheet Example </title>
</head>
<style>
BODY{
background: #FFFFFF;
color: #000000;
}
H1{
font: 14pt verdana; color: #CCCCCC;}
P{font: 12pt times;}
A{color: #FFOOOO; text-decoration: none}
```

</style>

Как видно из примера, приведенного выше, таблица стилей теперь отличается от кода стандартной страницы HTML, но все же логику проследить нетрудно. В нашем случае для основной части страницы (body) указаны цвет фона, цвет текста и верхнее, левое и правое поля в дюймах.

Для заголовка первого уровня (HI) указывается шрифт (название шрифта и размер в пунктах). В этом и состоит удобство каскадных таблиц стилей - вы можете задавать размеры не только в пунктах, но и пикселях (px), процентах (75%) и сантиметрах (cm).

Существует и несколько новых единиц измерения, самой примечательной из которых является m.

В отличие от пунктов или пикселов, представляющих абсолютные размеры различных объектов, 1m - ширина строчной буквы m в том шрифте о котором идет речь. Правда, большинство браузеров об этом не догадываются и поэтому определяют 1m просто как размер, заданный по умолчанию. Например, в Internet Explorer 4.0 и выше для гарнитуры Verdana размер 1m соответствует 12 пунктам. Точнее сказать "соответствует значению, которое пользователь установил в качестве размера шрифта по умолчанию. Таким образом, если установить в браузере размер шрифта по умолчанию 16 пунктам, то и 1m тоже станет равна 16 пунктам, и все остальные размеры будут соответственно увеличены.

Но! Не все браузеры способны понимать эти новые единицы измерения. Я попробовала применить новую единицу измерения m во внедренной таблице стиля страницы "Дизайн на основе правил" (webrules.htm). Браузер Internet Explorer 5.0 показал эту страницу таким образом, как она задумывалась. Когда же просмотр страницы осуществлялся браузером Internet Explorer 3.0, результат был плачевным - несколько черных штрихов и никакого текста.

Лекция № 12 (2 часа)

1.7 Лекция № 13-14 (4 часа)

Тема: «Введение в JavaSript. Синтаксис языка. Объектная модель JavaScript.»

1.7.1 Вопросы лекции:

1) Что такое JavaScript?

2) Компиляция и интерпретация

3) Что умеет JavaScript

4) Ограничения JavaScript

5) Уникальность объектной модели.

1.7.2 Краткое содержание вопросов:

1) Что такое JavaScript?

JavaScript изначально создавался для того, чтобы сделать web-странички «живыми». Программы на этом языке называются скриптами. В браузере они подключаются напрямую к HTML и, как только загружается страничка – тут же выполняются.

Программы на JavaScript – обычный текст. Они не требуют какой-то специальной подготовки.

В этом плане JavaScript сильно отличается от другого языка, который называется Java.

Когда создавался язык JavaScript, у него изначально было другое название: «LiveScript». Но тогда был очень популярен язык Java, и маркетологи решили, что схожее название сделает новый язык более популярным.

Планировалось, что JavaScript будет эдаким «младшим братом» Java. Однако, история распорядилась по-своему, JavaScript сильно вырос, и сейчас это совершенно независимый язык, со своей спецификацией, которая называется ECMAScript, и к Java не имеет никакого отношения.

У него много особенностей, которые усложняют освоение, но по ходу учебника мы с ними разберёмся.

JavaScript может выполняться не только в браузере, а где угодно, нужна лишь специальная программа – интерпретатор. Процесс выполнения скрипта называют «интерпретацией».

2) Компиляция и интерпретация

Для выполнения программ, не важно на каком языке, существуют два способа: «компиляция» и «интерпретация».

Компиляция – это когда исходный код программы, при помощи специального инструмента, другой программы, которая называется «компилятор», преобразуется в другой язык, как правило – в машинный код. Этот машинный код затем распространяется и запускается. При этом исходный код программы остаётся у разработчика.

Интерпретация – это когда исходный код программы получает другой инструмент, который называют «интерпретатор», и выполняет его «как есть». При этом распространяется именно сам исходный код (скрипт). Этот подход применяется в браузерах для JavaScript.

Современные интерпретаторы перед выполнением преобразуют JavaScript в машинный код или близко к нему, оптимизируют, а уже затем выполняют. И даже во время выполнения стараются оптимизировать. Поэтому JavaScript работает очень быстро.

Во все основные браузеры встроен интерпретатор JavaScript, именно поэтому они могут выполнять скрипты на странице. Но, разумеется, JavaScript можно использовать не только в браузере. Это полноценный язык, программы на котором можно запускать и на сервере, и даже в стиральной машинке, если в ней установлен соответствующий интерпретатор.

3) Что умеет JavaScript?

Современный JavaScript – это «безопасный» язык программирования общего назначения. Он не предоставляет низкоуровневых средств работы с памятью, процессором, так как изначально был ориентирован на браузеры, в которых это не требуется.

Что ж касается остальных возможностей – они зависят от окружения, в котором запущен JavaScript. В браузере JavaScript умеет делать всё, что относится к манипуляции со страницей, взаимодействию с посетителем и, в какой-то мере, с сервером:

Создавать новые HTML-теги, удалять существующие, менять стили элементов, прятать, показывать элементы и т.п.

Реагировать на действия посетителя, обрабатывать клики мыши, перемещения курсора, нажатия на клавиатуру и т.п.

Посылать запросы на сервер и загружать данные без перезагрузки страницы (эта технология называется "AJAX").

Получать и устанавливать cookie, запрашивать данные, выводить сообщения…

4) Ограничения JavaScript

JavaScript – быстрый и мощный язык, но браузер накладывает на его исполнение некоторые ограничения…

Это сделано для безопасности пользователей, чтобы злоумышленник не мог с помощью JavaScript получить личные данные или как-то навредить компьютеру пользователя.

Этих ограничений нет там, где JavaScript используется вне браузера, например на сервере. Кроме того, современные браузеры предоставляют свои механизмы по установке плагинов и расширений, которые обладают расширенными возможностями, но требуют специальных действий по установке от пользователя

Большинство возможностей JavaScript в браузере ограничено текущим окном и страницей.

JavaScript не может читать/записывать произвольные файлы на жесткий диск, копировать их или вызывать программы. Он не имеет прямого доступа к операционной системе.

Современные браузеры могут работать с файлами, но эта возможность ограничена специально выделенной директорией – «песочницей». Возможности по доступу к устройствам также прорабатываются в современных стандартах и частично доступны в некоторых браузерах.

JavaScript, работающий в одной вкладке, не может общаться с другими вкладками и окнами, за исключением случая, когда он сам открыл это окно или несколько вкладок из одного источника (одинаковый домен, порт, протокол).

Есть способы это обойти, и они раскрыты в учебнике, но они требуют специального кода на оба документа, которые находятся в разных вкладках или окнах. Без него, из соображений безопасности, залезть из одной вкладки в другую при помощи JavaScript нельзя.

Из JavaScript можно легко посылать запросы на сервер, с которого пришла страница. Запрос на другой домен тоже возможен, но менее удобен, т. к. и здесь есть ограничения безопасности.

5) Уникальность объектной модели.

Есть как минимум три замечательных особенности JavaScript:

Полная интеграция с HTML/CSS.

Простые вещи делаются просто.

Поддерживается всеми распространёнными браузерами и включён по умолчанию.

Этих трёх вещей одновременно нет больше ни в одной браузерной технологии.

Поэтому JavaScript и является самым распространённым средством создания браузерных интерфейсов.

Перед тем, как вы планируете изучить новую технологию, полезно ознакомиться с её развитием и перспективами. Здесь в JavaScript всё более чем хорошо.

HTML 5 – эволюция стандарта HTML, добавляющая новые теги и, что более важно, ряд новых возможностей браузерам.

Вот несколько примеров:

Чтение/запись файлов на диск (в специальной «песочнице», то есть не любые).

Встроенная в браузер база данных, которая позволяет хранить данные на компьютере пользователя.

Многозадачность с одновременным использованием нескольких ядер процессора.

Проигрывание видео/аудио, без Flash.

2D и 3D-рисование с аппаратной поддержкой, как в современных играх.

Многие возможности HTML5 всё ещё в разработке, но браузеры постепенно начинают их поддерживать.

Тенденция: JavaScript становится всё более и более мощным и возможности браузера растут в сторону десктопных приложений.

Сам язык JavaScript улучшается. Современный стандарт ECMAScript 5 включает в себя новые возможности для разработки, ECMAScript 6 будет шагом вперёд в улучшении синтаксиса языка.

Современные браузеры улучшают свои движки, чтобы увеличить скорость исполнения JavaScript, исправляют баги и стараются следовать стандартам.

Тенденция: JavaScript становится всё быстрее и стабильнее, в язык добавляются новые возможности.

Очень важно то, что новые стандарты HTML5 и ECMAScript сохраняют максимальную совместимость с предыдущими версиями. Это позволяет избежать неприятностей с уже существующими приложениями.

Впрочем, небольшая проблема с «современными штуками» всё же есть. Иногда браузеры стараются включить новые возможности, которые ещё не полностью описаны в стандарте, но настолько интересны, что разработчики просто не могут ждать.

…Однако, со временем стандарт меняется и браузерам приходится подстраиваться к нему, что может привести к ошибкам в уже написанном, основанном на старой реализации, JavaScript-коде. Поэтому следует дважды подумать перед тем, как применять на практике такие «супер-новые» решения.

При этом все браузеры сходятся к стандарту, и различий между ними уже гораздо меньше, чем всего лишь несколько лет назад.

1.8 Лекция №15 (2 часа)

Тема: «Типы данных. Операторы JavaScript.

1.8.1 Вопросы лекции:

1. Основные типы данных
2) Операторы JavaScript

1.8.2 Краткое содержание вопросов:

1) Основные типы данных

В JavaScript существует несколько основных типов данных.

В этой главе мы получим о них общее представление, а позже, в соответствующих главах подробно познакомимся с использованием каждого типа в отдельности.

Число «number»

```
var n = 123;
n = 12.345;
```

Единый тип число используется как для целых, так и для дробных чисел.

Существуют специальные числовые значения Infinity (бесконечность) и NaN (ошибка вычислений).

Например, бесконечность Infinity получается при делении на ноль:

```
alert( 1 / 0 ); // Infinity
```

Ошибка вычислений NaN будет результатом некорректной математической операции, например:

```
alert( "нечисло" * 2 ); // NaN, ошибка
```

Эти значения формально принадлежат типу «число», хотя, конечно, числами в их обычном понимании не являются.

Строка «string»

```
var str = "Мама мыла раму";
str = 'Одинарные кавычки тоже подойдут';
```

В JavaScript одинарные и двойные кавычки равноправны. Можно использовать или те или другие.

Тип символ не существует, есть только строка.

В некоторых языках программирования есть специальный тип данных для одного символа. Например, в языке C это char. В JavaScript есть только тип «строка» string. Что, надо сказать, вполне удобно.

Булевый (логический) тип «boolean»

У него всего два значения: true (истина) и false (ложь).

Как правило, такой тип используется для хранения значения типа да/нет, например:

```
var checked = true; // поле формы помечено галочкой
checked = false;    // поле формы не содержит галочки
```

О нём мы поговорим более подробно, когда будем обсуждать логические вычисления и условные операторы.

Специальное значение «null»

Значение null не относится ни к одному из типов выше, а образует свой отдельный тип, состоящий из единственного значения null:

```
var age = null;
```

В JavaScript null не является «ссылкой на несуществующий объект» или «нулевым указателем», как в некоторых других языках. Это просто специальное значение, которое имеет смысл «ничего» или «значение неизвестно».

В частности, код выше говорит о том, что возраст age неизвестен.

Специальное значение «undefined»

Значение undefined, как и null, образует свой собственный тип, состоящий из одного этого значения. Оно имеет смысл «значение не присвоено».

Если переменная объявлена, но в неё ничего не записано, то её значение как раз и есть undefined:

```
var x;
alert( x ); // выведет "undefined"
```

Можно присвоить undefined и в явном виде, хотя это делается редко:

```
var x = 123;
x = undefined;
alert( x ); // "undefined"
```

В явном виде undefined обычно не присваивают, так как это противоречит его смыслу. Для записи в переменную «пустого» или «неизвестного» значения используется null.

Объекты «object»

Первые 5 типов называют «примитивными».

Особняком стоит шестой тип: «объекты».

Он используется для коллекций данных и для объявления более сложных сущностей.

Объявляются объекты при помощи фигурных скобок {...}, например:

```
var user = { name: "Вася" };
```

Мы подробно разберём способы объявления объектов и, вообще, работу с объектами, позже, в главе Объекты как ассоциативные массивы.

Оператор typeof

Оператор typeof возвращает тип аргумента.

У него есть два синтаксиса: со скобками и без:

Синтаксис оператора: typeof x.

Синтаксис функции: typeof(x).

Работают они одинаково, но первый синтаксис короче.

Результатом typeof является строка, содержащая тип:

```
typeof undefined // "undefined"
typeof 0 // "number"
typeof true // "boolean"
typeof "foo" // "string"
typeof {} // "object"
typeof null // "object"  (1)
typeof function(){} // "function"  (2)
```

Последние две строки помечены, потому что typeof ведет себя в них по-особому.

Результат typeof null == "object" – это официально признанная ошибка в языке, которая сохраняется для совместимости. На самом деле null – это не объект, а отдельный тип данных.

Функции мы пройдём чуть позже. Пока лишь заметим, что функции не являются отдельным базовым типом в JavaScript, а подвидом объектов. Но typeof выделяет функции отдельно, возвращая для них "function". На практике это весьма удобно, так как позволяет легко определить функцию.

К работе с типами мы также вернёмся более подробно в будущем, после изучения основных структур данных.

Итого есть 5 «примитивных» типов: number, string, boolean, null, undefined и 6-й тип – объекты object.

2) Операторы JavaScript

Операторы – это основа для построения любой программы. На самом деле, любая программа на JavaScript состоит из множества операторов: каждое из рассмотренных нами выражений, заканчивающееся символом точки с запятой (;), также является оператором. Фактически, точка с запятой – это тоже оператор (в других языках он называется «пустым оператором»).

Вообще же операторы управляют процессом выполнения программы. Набор операторов JavaScript типичен для языков структурного программирования – все они подразделяются на условные операторы, операторы циклов и операторы управления объектами. Есть и еще один оператор, который нам уже знаком – это оператор комментариев.

Программы на JavaScript могут состоять как из одиночных операторов, так и из программных блоков. Программный блок – это группа операторов, заключенных в фигурные скобки ({ и }).

Что касается пустого оператора, состоящего только из точки с запятой, то он может быть применен в любом месте программы, где, по правилам синтаксиса JavaScript, может (или должен) находиться любой оператор. Прежде всего, точку с запятой ставят в конце каждого выражения для того, чтобы это выражение стало оператором.

Основное отличие пустого оператора от остальных состоит в том, что сам по себе он никак не влияет на выполнение программы.

Условный оператор if

К условным операторам в JavaScript относятся if…else и switch. Они служат для определения набора команд, которые должны быть выполнены в случае, если условие, заданное в таком операторе, истинно.

Оператор if выполняет проверку своего условия, после чего, если результатом будет истина (true), то выполняется блок операторов, следующих за условием. Если же результат – ложь (false), то в том случае, если предусмотрена часть else, будет выполнен блок операторов, определенных для else, в противном случае программа просто будет выполняться дальше. Например, если нам требуется вывести приветствие только в том случае, если значение переменной x больше 100, то при помощи условного оператора мы можем реализовать следующий код:

```
if (x>100) alert("Привет!");
```

В данном примере, когда условие истинно (т.е. z больше 100), выводится окно с текстом «Привет!». В противном случае ничего не выводится. Если же требуется что-то сообщить посетителю в любом случае, то можно предусмотреть часть else:

```
if (x>100) alert("Привет!");
  else alert(x);
```

Теперь, если значение переменной x будет меньше 100, то вместо приветствия будет выведено окно со значением переменной x.

Если в качестве выполняемого кода надо написать больше одного оператора, то следует использовать программный блок. Например, если после вывода приветствия надо было бы еще и изменить значение переменной x, то мы получим такой код:

```
if (x>100) {
  alert("Привет!");
  x = 0;
}
```

Блок в данном случае объединяет 2 выражения-оператора. Если бы мы не использовали блок, то выражение «alert("Привет!");» относился бы к уловному оператору if, а второе выражение было бы выполнено в любом случае. Вообще же предпочтительно всегда использовать блоки для любых операторов, в том числе и условных – это позволяет более наглядно видеть то, что и когда должно быть выполнено. Например, одно и то же выражение можно записать как без блоков:

```
if (a!=b) alert("Не равно!") else alert("Равно");
```

А можно и с их использованием:

```
if (a!=b)
 {
  alert("Не равно!");
 }
else
 {
  alert("Равно");
 };
```

Сравните эти записи: хотя для машины (интерпретатора JavaScript) обе они идентичны, человеку гораздо проще увидеть суть программы во втором случае, особенно если представить себе, что данный код является всего лишь фрагментом программы.

Условный оператор if имеет также сокращенную форму – в виде условной операции “? :”. Так, рассмотренной выше пример можно записать в таком вот виде:

```
(a!=b) ? alert("Не равно!") : alert("Равно");
```

Кроме того, оператор if может быть вложенным (как, впрочем, и любой другой оператор языка JavaScript). Например, если нам надо проверить определенное условие (скажем, является ли значение переменной x больше 200) только в том случае, если другое условие верно. В таком случае один оператор if вкладывается в другой:

```
if (x>100) {
  if (x>200) {
    alert("Большой привет!");
  } else {
  alert("Привет!");
  };
};
```

Здесь сначала проверяется, действительно ли x больше 100, и если такое условие верно, то значение x проверяется вновь, но уже на предмет того, больше ли оно, чем 200 или нет. Если второе условие верно, то при помощи alert выводится фраза «Большой привет!», а если нет (блок else для вложенного if), то выводится «Привет!». Ну а в том случае, если не верно первое условие, что ничего не выводится, поскольку весь блок, вложенный во «внешний» оператор if, будет проигнорирован.

Семафор switch

Только что мы рассмотрели пример использования условного оператора if для проверки на множество условий. Однако для этих целей в JavaScript имеется специализированный оператор – switch. Оператор switch (семафор, переключатель) выполняет тот блок операторов, метка которого соответствует значению выражения переключателя. Он имеет следующий синтаксис:

```
switch (выражение) {
```

```
 case метка: выражение; break;
 ...
 default: выражение;
}
```

Выполнение оператора switch начинается с вычисления значения переключателя (т.е. выражения, находящегося в скобках сразу после ключевого слова switch). После этого управление передается блоку операторов, следующих за меткой, совпадающей со значением. Если таковой не находится, то управление будет передано специальной метке – default (если она имеется). Так, последний пример с if можно представить с использованием семафора следующим образом:

```
switch (x) {
 case 1: y=100; break;
 case 2: y=150; break;
 case 3: y=50; break;
 case 4: y=200; break;
 default: y=0;
}
```

В этом примере, когда переменная x имеет значения от 1 до 4, для нее предусмотрены определенные действия, если же она не впишется ни в одно из предусмотренных значений, то будет выполнено выражение, указанное в части default, а если бы таковой не оказалось, то ничего не было бы выполнено.

Что касается оператора break, то он используется для того, чтобы указать интерпретатору, что после того, как будет найдено подходящее условие, а написанное для него выражение – выполнено, следует завершить выполнение оператора. В противном случае (если после выражения не указать break), то будет выполнены все последующие условия – до тех пор, пока не попадется break или не будет достигнут конец данного оператора switch.

Наконец, подобно случаю с if, для семафора так же полезно использовать блоки, поскольку это улучшает простоту восприятия исходного кода программы:

```
switch (x) {
 case 1: {
   y=100;
   break;
 }
 case 2: {
   y=150;
   break;
 }
...
};
```

Следует отметить, что в качестве значений, оцениваемых при помощи оператора switch, могут быть не только числа, но и строки. Этим JavaScript в лучшую сторону отличается от некоторых других языков программирования, имеющих ограничения на оцениваемые при помощи семафора типы данных.

Оператор цикла for

Для написания практически любой программы, помимо операторов условия, требуются операторы цикла, и в JavaScript, разумеется, они имеются. Прежде всего, это оператор цикла for. Он имеет следующий синтаксис:

for ([начальное значение]; [условие]; [выражение приращения]) ТелоЦикла;

Квадратные скобки говорят о том, что заключенный в них элемент является необязательным. И действительно, для цикла for допустимо не указывать никаких выражений, так что в простейшем случае мы получаем бесконечный цикл:

for (;;);

Но, как правило, все эти значения все-таки указывают. В качестве выражения инициализации указывают выражение, которое должно быть выполнено перед началом цикла, в качестве условия – выражение, которое будет оцениваться при каждой итерации цикла и разрешать его продолжение, пока истинно, а приращение определяет изменение для выражения в условии. Рассмотрим такой цикл на следующем примере:

for (var i=0; i<10; i++) ;

Здесь сначала инициализируется переменная цикла (i), которой назначается значение 0, после чего определяется условие, которое разрешает дальнейшие итерации цикла до тех пор, пока является истинным, и, наконец, задается приращение переменной цикла. Данный цикл будет выполнен 10 раз – до тех пор, пока значение переменной i (счетчика цикла) не достигнет 10 и не приведет к тому, что условие станет ложью.

Следует отметить, что выражение приращения выполняется уже после того, как будет выполнено тело цикла (в данном случае представленное пустым оператором). Таким образом, схема работы цикла for выглядит следующим образом:

Инициализируется значение переменной-счетчика

Проверяется условие. Если оно оказывается ложью, цикл прекращает работу, если нет – то он выполняется

Происходит собственно выполнение тела цикла

Выполняется приращение счетчика и происходит возврат на этап 2.

Соответственно, если условие изначально ложно, то цикл не будет выполнен ни разу. А если условие задано неверно (например, для рассматриваемого примера было бы задано условие i=>0), то цикл будет выполняться бесконечно.

Практическое применение циклов крайне разнообразно. Если привести наиболее общий пример из программирования, то цикл – идеальный способ заполнения массива. Например, если требуется организовать цикл для заполнения массива из 10 числовых значений последовательно возрастающими числами, то можно записать:

for (var i=0; i<10; i++) MyArray[i+1]=i+1;

В данном случае элементам массива MyArray последовательно назначаются значения от 1 до 10. Собственно говоря, учитывая гибкость языка JavaScript, можно указать действие по заполнению непосредственно в выражении приращения:

for (var i=0; i<10; MyArray[i++]=i);

В данном случае значение счетчика (переменной i) все равно будет увеличиваться на 1 при каждой итерации цикла, при этом попутно будет происходить и заполнение массива.

Поскольку для циклов требуется наличие хотя бы одного оператора в теле цикла, то в случаях, подобных рассмотренному выше, используется пустой оператор. А если используется более одного оператора, то блок операторов должен быть заключен в фигурные скобки.

Отметим, что выражение приращения вовсе не обязательно должно именно увеличивать счетчик – допустимо изменение и в обратную сторону. Важно лишь, чтобы условие также учитывало эту особенность, иначе можно получить либо бесконечный, либо никогда не выполняющийся цикл.

Рассмотрим цикл for с отрицательным приращением на примере вычисления математического факториала (последовательное произведение всех целых чисел от 1 до самого числа). Для этого нам понадобится следующий цикл:

var num;

for (var rez=1; num; num--) {

```
  rez=rez*num;
};
```

Здесь в начальном условии определена переменная, в которой будет храниться вычисляемое значение (rez) и ей присвоено значение 1. В качестве условия указана другая переменная (num), представляющая собой число, для которого и будет вычисляться факториал. Поскольку нам надо будет прекратить выполнение цикла, как только num достигнет 0, то мы указали просто переменную num: истиной она будет считаться только до тех пор, пока ее значение отличается от нуля. Наконец, в конце каждого цикла будет происходить декрементация переменной num.

В итоге, если переменной num присвоить значение 5, то после прохождения цикла переменная rez получит значение 120. Хотя в результате работы такого цикла получится выполнение как бы наоборот (т.е. не вместо 1*2*3*4*5, на самом деле выполняется 5*4*3*2*1), это никак не помешает получить верный результат.

Еще одним вариантом цикла for является его использование совместоно с оператором in, применяемого для обхода свойств объекта (или ассоциативного массива). Очень часто его используют в JavaScript в качестве аналога оператора foreach в PHP (так же имеется в VB, C# и в последних версиях Delphi). Такое его использование будет рассмотрено в параграфе "Операторы манипулирования объектами".

Операторы цикла while и do-while

С помощью цикла while можно выполнять один или несколько операторов до тех пор, пока верно заданное условие. Его синтаксис имеет следующий вид:

```
while (условие) {
  Тело цикла
}
```

В простейшем случае использование цикла while может сводиться к следующему:

```
while (a>5) {
  a--;
}
```

Здесь переменная a будет уменьшаться при каждой итерации до тех пор, пока условие не станет ложью.

От цикла for цикл while отличается, прежде всего, тем, что за изменение условия несет ответственность не определение цикла, а его исполняемый блок (тело). Хотя, на самом деле, цикл for можно использовать точно так же:

```
for ( ; a>5 ; ) {
  a--;
}
```

Подобно тому, как у цикла for, сначала происходит оценка выражения, указанного в качестве условия. И если заданное условие неверно изначально, то цикл while не будет выполнен ни разу – точно так же, как и for. Поэтому в тех случаях, когда требуется выполнить операторы цикла хотя бы один раз, можно использовать оператор do…while. Этот оператор выглядит как перевернутый цикл while:

```
do {
  Тело цикла
} while (условие)
```

Когда интерпретатор JavaScript доходит до оператора do…while, то сначала выполняются операторы тела цикла, и уже потом – проверяется условие. Если условие истинно, то цикл повторяется, и так до тех пор, пока условие не станет ложным. Это может быть полезным во многих случаях – например, когда надо в любом случае произвести какую-либо операцию, вне зависимости от того, что задано в условии. Конечно, можно было бы обойти это условие, например, несколько изменив условие

цикла или продублировав «обязательный» код перед циклом, но, в конечном итоге, оба цикла while служат, прежде всего, для удобства разработчиков.

1.12 Лекция №16 (2 часа)

Тема: «Обработка событий. Примеры эффективного программирования на JS»

1.12.1 Вопросы лекции

1. Примеры эффективного программирования на PHP

1.12.2 Краткое содержание вопросов

Время на сайте:

С помощью php легко узнать текущее время на сервере, например, во время загрузки на сервере было 19:12:22 20.05.2016, но в отличии от javascript это время не меняется онлайн, т.е. отображается то время, которое было на сервере в момент загрузки данной страницы.

Чтобы вывести дату на сайте нужно воспользоваться функцией date().

Маленькая хитрость: обычно в конце страницы ставят копирайт так Copyright © Computerlessons.ru, 2007 - 2016. Обратите внимание на вторую дату, она выводится с помощью php и в этом случае не нужно каждый год вручную подправлять число.

Подсветка php-кода:

Вы наверное уже обратили внимание, что на этой странице я привожу примеры php-кодов в цветном варианте. Цветной код удобно читать, нежели чёрно-белый. Например, я мог бы показывать вам коды так:

Но всё же согласитесь, цветной код лучше! Выше я привожу вам пример функцииhighlight_file(), с её помощью можно выводить на страницу php-код из другого файла. Ниже посмотрите функцию highlight_string(), где можно раскрашивать код непосредственно на странице.

Маленькая поправочка: следите за ковычками внутри этих функций, если используете двойную ковычку ", то внутри используйте только одинарные ', если нужно использовать двойную ковычку, то её необходимо экранировать обратным слешем \". То же касается и одинарных ковычек.

1.9 Лекция №17 (2 часа)

Тема: «Основы DHTML. Интерактивная форма.

1.9.1 Вопросы лекции

1) Происхождение DHTML.

2) Содержание DHTML.

3) Создание сценариев.

4) Основные события DHTML.

1.9.2 Краткое содержание вопросов

1) Происхождение DHTML.

DHTML (динамический HTML) – это набор средств, которые позволяют создавать более интерактивные Web-страницы без увеличения загрузки сервера. Другими словами, определенные действия посетителя ведут к изменениям внешнего вида и содержания страницы без обращения к серверу.

DHTML построен на объектной модели документа (Document Object Model, DOM), которая расширяет традиционный статический HTML-документ. DOM обеспечивает динамический доступ к содержимому документа, его структуре и стилям. В DOM каждый элемент Web-страницы является объектом, который можно изменять. DOM не определяет новых тэгов и атрибутов, а просто обеспечивает возможность программного управления всеми тэгами, атрибутами и каскадными листами стилей (CSS).

По мере развития индустрии создания Web-сайтов возникла потребность расширить возможности использовать мультимедиа как инструмента Web-дизайнера при создании Web-страниц.

Эту проблему решает динамический язык HTML (DHTML).

DHTML дает возможность создавать элементы Web-страниц (типа текста и графики) интерактивными и динамическими. При этом Web-страницы DHTML загружаются просто мгновенно.

Все последнии версии программного обеспечения Web-браузеров Microsoft и Netscape поддерживают язык DHTML.

Теперь в распоряжении Web-дизайнеров есть целая куча новых интерактивных эффектов, включая возможности организации текста и графики в виде презентаций типа телевизионных заставок.

2) Содержание DHTML.

Эффекты DHTML создаются с помощью трех технологий: HTML, каскадных таблиц стилей и сценариев.

Хотя все эти технологии существовали уже долгое время, сейчас они используются вместе и предоставляют дизайнерам возможности создания Web-страниц, которые выглядят и работают лучше, чем когда-либо прежде.

Самое главное - это научиться мыслить динамически!

HTML служит основой для эффектов DHTML.

Каскадные таблицы стилей (CSS) предоставляют возможности точной установки графических элементов на Web-страницах (см. статью "Каскадные таблицы стилей").

Кроме того, существуют эффекты изменения внешнего вида текста и графики на странице - называемые "фильтры". Фактические возможности фильтров определяются компонентами Web-браузера.

Существуют статические и динамические фильтры.

Статические фильтры просто изменяют внешний вид элемента. Динамические фильтры позволяют изменить графический элемент со скоростью, задаваемой пользователем. Работа динамических фильтров основана на сценариях.

3) Создание сценариев.

С помощью CSS каждый элемент Web-страницы можно не только точно установить в определенное место, но также сделать доступным для применения специальных операций и задания нужных свойств. Эти свойства управляются с помощи сценариев (scripts). Сценарий делает элементы Web-страницы динамическими - кнопки "нажимаются", текст появляется и исчезает, а изображения просто летают по экрану.

DHTML можно реализовывать с помощью двух языков сценариев: VBScript (Visual Basic Scripting) и JavaScript. Visual Basic Scripting и является упрощенной версией языка программирования Microsoft Visual Basic. JavaScript - это версия языка программирования Java (от фирмы Sun Microsystems) для создания сценариев.

Хотя для создания DHTML можно использовать любой язык сценариев, JavaScript все же является наиболее многосторонним, так как сейчас он поддерживается броузерами и от Microsoft, и от Netscape. Кроме того, синтаксис JavaScript аналогичен синтаксису языков Java и C++, которые знакомы многим Web-разработчикам. Microsoft и Netscape поддерживают JavaScript, но они находятся на различных стадиях реализации.

Будьте осторожны при работе с DHTML на разных платформах. Если вы создаете страницы для определенного браузера, как при разработках для корпоративной сети, выбор становится вопросом личных вкусов.

Некоторые Web-разработчики используют комбинацию JavaScript и VBScript, чтобы сделать свои узлы динамическими.

Написание сценариев требует больших усилий, чем создание обычного кода HTML. С другой стороны, работа с сценариями совсем не так сложна, как реальный язык программирования, так как в этом случае не выполняется компиляция.

4) Основные события DHTML.

OnMouseOver. Перемещение указателя мыши на элемент.

OnMouseOut. Перемещение указателя мыши за пределы элемента.

OnMouseDown. Нажатие любой кнопки мыши. Внутри обработчика event.button указывает, какая кнопка нажата: 1 = левая, 2 = правая, 4 = средняя.

OnMouseUp. Отпускание любой кнопки мыши. Внутри обработчика event.button указывает, какая кнопка отпущена: 1 = левая, 2 = правая, 4 = средняя.

OnMouseMove. Перемещение указателя мыши. Внутри обработчика event.x и event.y - текущие координаты "горячей" точки курсора на экране.

OnClick. Щелчок левой кнопкой мыши на элементе или нажатие <Enter> при каком-то элементе в фокусе.

OnDblClick. Двойной щелчок левой кнопкой мыши на элементе.

OnDragStart. Запуск операции перетаскивания. Цель – запретить операцию перетаскивания путем возвращения значения false.

OnSelectStart. Запуск операции выделения элемента. Цель – запретить выделение области документа. Важно учесть, что отменяется лишь инициализация выделения, т.е. если выделение начато за пределами данной области, а на ней лишь продолжается, то помешать выделению нельзя. Событие OnSelectStart всплывающее, поэтому можно перехватить его и вернуть значение false. Это лишит посетителя возможности выделять текст на странице (см. Пример 10).

OnSelect. Выделение элемента. Следует за OnSelectStart и возникает много раз по мере того, как посетитель расширяет или сужает выделение. Событие OnSelect не всплывает. Оно возникает лишь в том разделе документа, где происходит выделение.

События клавиатуры

OnKeyPress. Нажатие и отпускание клавиши. Событие возникает много раз, если клавиша удерживается.

OnKeyDown. Нажатие клавиши. Событие возникает один раз, даже если клавиша продолжает удерживаться.

OnKeyUp. Отпускание клавиши.

Событие прокручивания

OnScroll. Использование полосы прокрутки или прокручивание элемента неявно посредством другого действия Не может отменить прокручивания, так как возникает после завершения прокручивания. Не всплывает.

События фокуса

OnFocus. Возникает, когда элемент активизируется после щелчка по нему мышью или с помощью клавиатуры. Фокус могут получить только элементы пользовательского ввода и тело документа, а не элементы содержания документа.

OnBlur. Возникает, когда элемент теряет фокус. Используется для контроля корректности ввода.

2.1 Практическое занятие № 1(2 часа).

Тема: «Обзор Web-технологий»

2.1.1 Задание для работы:

1. Создать файловую структуру Web-сайта.

2. Создать Web-страницу с использованием HTML-тегов форматирования текста.

3. Освоить форматирование текста Web-страниц

2.1.2 Краткое описание проводимого занятия:

Файловую структуру сайта образует совокупность файлов и папок, используемых при формировании Web-страниц.

Имена файлов и папок следует формировать без пробелов и только при помощи латинских букв нижнего регистра (строчных) и цифр!

Дело в том, что операционная система Windows в именах файлов и папок не различает регистра – ей все равно, какие буквы использованы – прописные или строчные. Однако Web-серверы в Интернете, как правило, устанавливаются на операционные системы семейства Unix, которые различают регистр в именах файлов и папок, а также не понимают кириллицу. Поэтому на компьютере под операционной системой Windows ссылки внутри сайта могут работать нормально, а после выгрузки на Web-сервер в Интернете могут перестать работать.

Файловая структура сайта сначала создается на локальном компьютере в отдельной папке, а затем содержание папки копируется на Web-сервер. В этой папке создаются:

стартовый HTML-файл с именем index.html;

как минимум две пустые папки для хранения HTML-файлов и графических файлов соответственно (рис. 1.1).

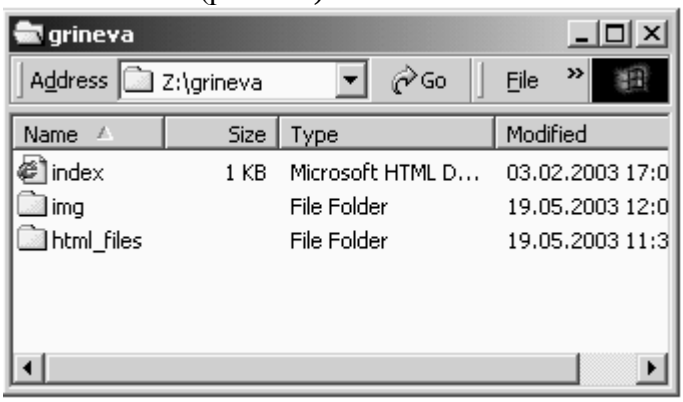


Рис. 1.1. Пример создания файловой структуры Web-сайта

Создать на диске Z (персональном сетевом ресурсе) папку по имени студента (латинскими буквами), а в ней еще две папки – **html_files** (для файлов типа *.html) и **img**(для графических файлов, как правило, типа *.jpg, *.gif и *.png).

1.2. Создание Web-страницы

При формировании любой Web-страницы может быть использовано от одного до достаточно большого количества файлов различного типа. Назначение основных типов файлов:

*.html или *.htm– содержат тексты HTML-кодов;

*.shtml или *.shtm– содержат кроме текстов HTML-кодов еще и инструкции для Web-сервера;

*.php или *.php3 – содержат кроме текстов HTML-кодов еще и серверные сценарии на языке PHP;

*.jpg, *.gif и *.png – содержат изображения;

*.css – содержат глобальные (то есть используемые на всех Web-страницах сайта) стили оформления Web-страниц сайта;

*.js – содержат глобальные сценарии на языке javascript.

Чтобы увидеть типы файлов в проводнике Windows необходимо разрешить отображать расширения для зарегистрированных типов файлов (**Мой компьютер – Сервис – Свойства папки** – вкладка **Вид** – убрать соответствующий флажок) (рис. 1.2).

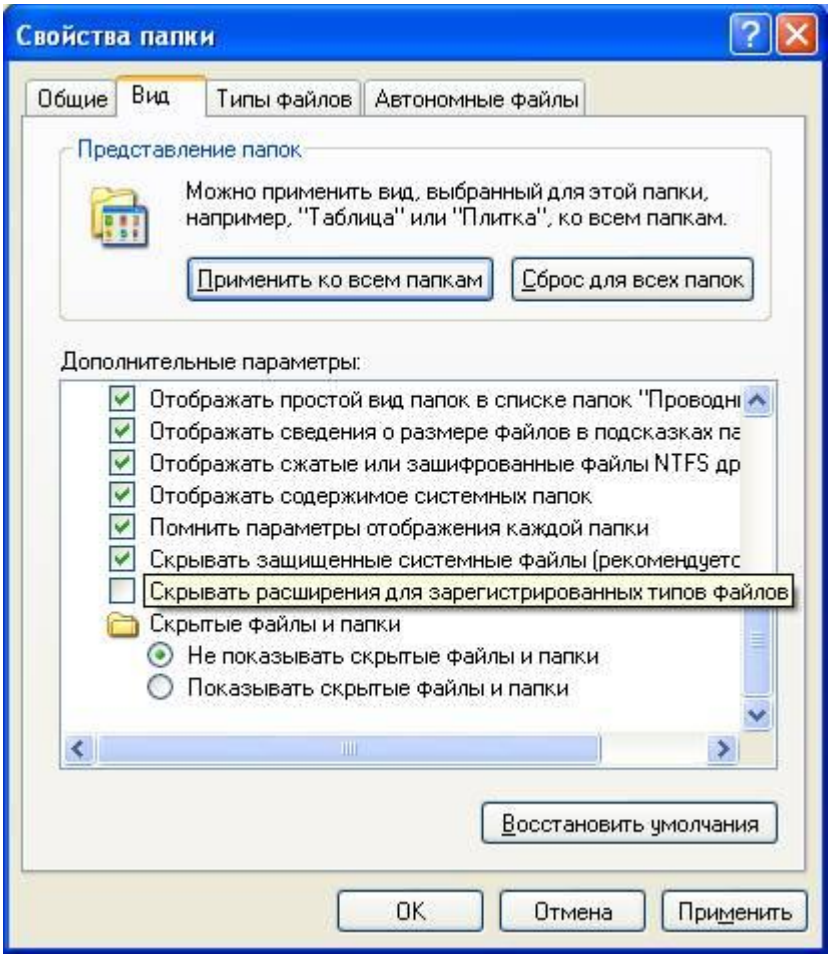


Рис. 1.2. Вкладка **Вид** в свойствах папки

Следует напомнить, что язык гипертекстовой разметки HTML позволяет отображать символы текста, представленного в кодировке ASCII, при которой один символ передается одним байтом, но как текст, выполненный в графическом текстовом редакторе, то есть с изменением размера, гарнитуры, цвета шрифта и т.п. Это резко уменьшает объем файла текста без снижения наглядности его отображения. Требуемое форматирование текста осуществляется при помощи, так называемых, тегов.

Тексты в кодировке ASCII формирует стандартное приложение Windows **Блокнот**. При помощи программы **Блокнот** набрать текст, представленный на рис. 1.3.

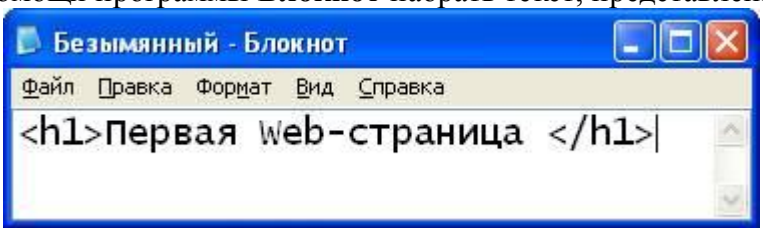


Рис. 1.3. HTML-код Web-страницы

В предложенном тексте использован только один HTML-тег – **h1**, который является парным тегом заголовка первого уровня.

Сохранить текст как HTML-файл в папке **html_files** с именем **site.html** (рис. 1.4)

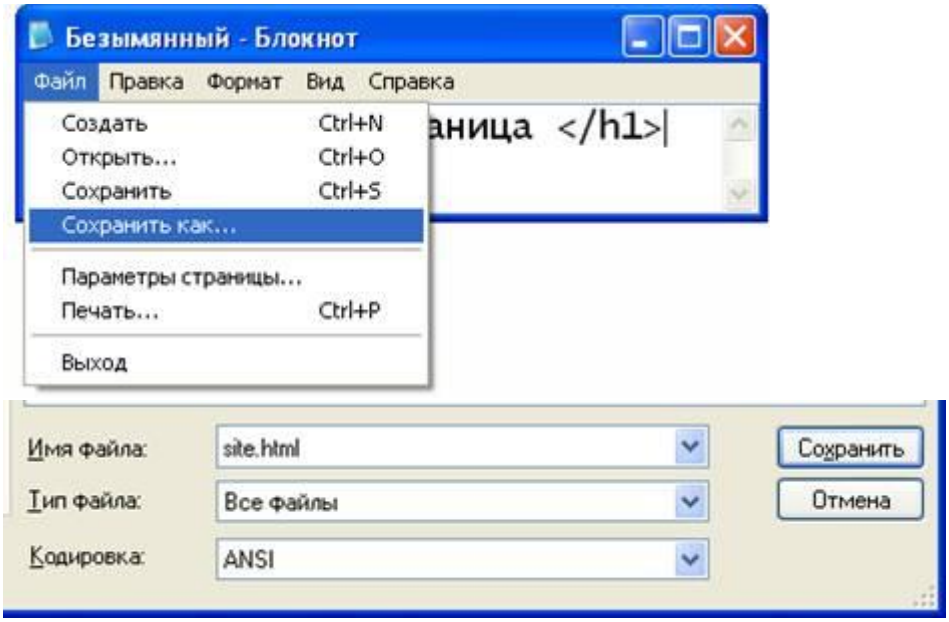


Рис. 1.4. Сохранение html-файла из Блокнота

Распознавать и выполнять HTML-теги могут только специальные приложения – браузеры. Вместе с операционной системой Windows поставляется только один браузер – **Internet Explorer**, который в дальнейшем и будем использовать для просмотра создаваемых Web-страниц. Поскольку HTML-тип файлов входит в число зарегистрированных типов файлов, то Windows по умолчанию присваивает им пиктограмму Internet Explorer (рис. 1.5) и просмотр файла автоматически осуществляется при помощи этого приложения (рис. 1.6).

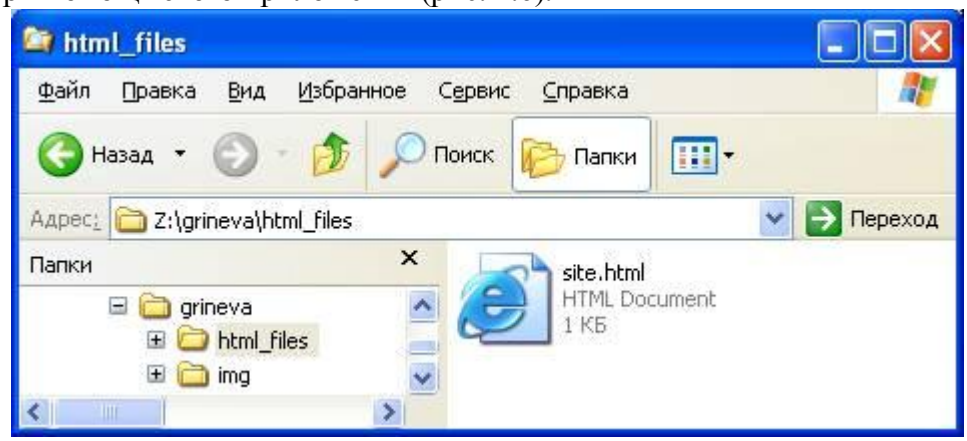


Рис. 1.5. Отображение файла HTML-типа в Windows

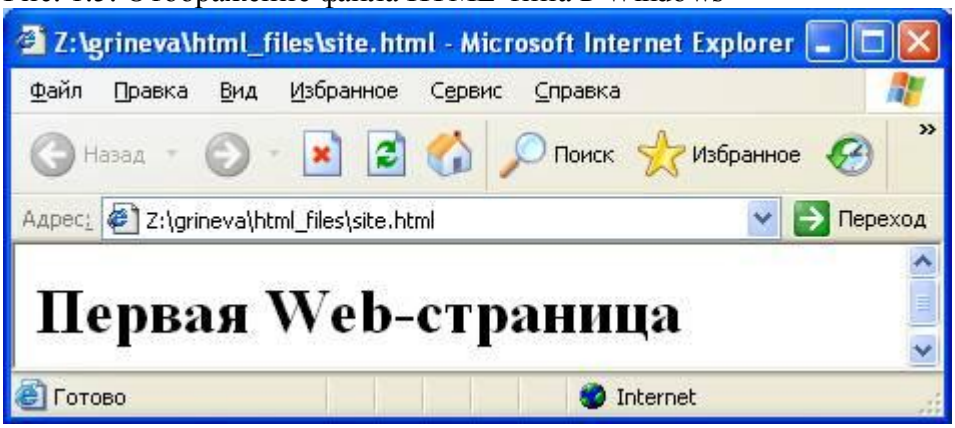


Рис. 1.6. Отображение файла **site.html** как Web-страницы

Если в HTML-файле нет заголовка Web-страницы (тега **head**) с тегом названия Web-страницы – **title**, то в строке заголовка браузера вместо названия Web-страницы отображается имя html-файла, так же как в строке адреса (см. рис. 1.6).

2.1.3 Результаты и выводы:

В результате выполнения практического задания мы освоили создание Web-страниц и форматирование их текста при помощи языка гипертекстовой разметки HTML.

2.2 Практическое занятие № 2 (2 часа).

Тема: «Структура HTML документа»

2.2.1 Задание для работы:

1. Создать файловую структуру Web-сайта.

2. Изменить стиль текста.

2.2.2 Краткое описание проводимого занятия:

В файле **site.html** при помощи **Блокнота** сформировать стандартную структуру Web-страницы с ее названием в заголовке:

```
<html>
<head> <!--- заголовок Web-страницы ---> <title>Первая страница</title>
</head>
<body> <h1>Первая Web-страница </h1>
</body>
</html>
```

Сохранить измененный файл **site.html** и убедиться, что в окне браузера в строке заголовка появилось название Web-страницы (после обновления содержания), но не отображается текст комментария, размещенного внутри тега <!-- --> (рис. 1.7).

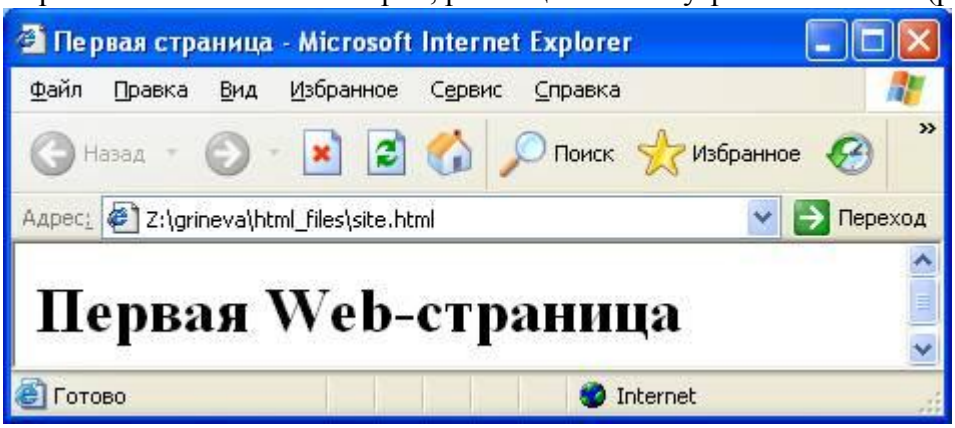


Рис. 1.7. Отображение название Web-страницы в строке заголовка браузера

С целью закрепления правил применения тегов и атрибутов HTML для форматирования текста Web-страницы выполнить следующее **упражнение:**

· осуществить выравнивание по центру заголовка при помощи атрибута align в открывающем теге заголовка:

```
<h1 align=center>Первая Web-страница </h1>
```

· добавить текст красного цвета, максимального размера 6 и шрифтом arial при помощи соответствующих атрибутов в открывающем теге font (рис. 1.8):

красный текст

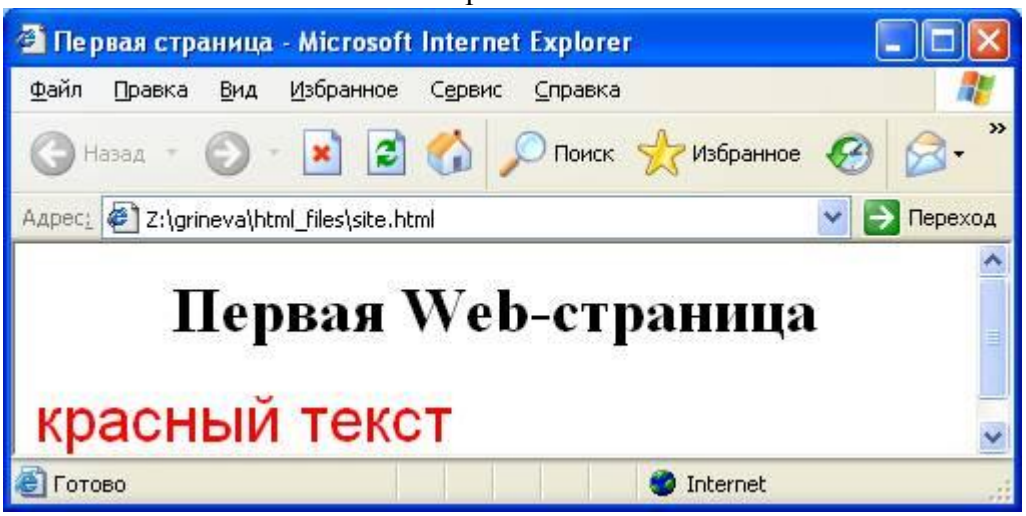


Рис. 1.8. Пример форматирования текста Web-страницы

где size={число} – размер шрифта (по умолчанию 3):

1 – самый маленький; 6 – самый большой; размер может быть относительным, например, +2 – увеличить размер шрифта на два пункта;

face="{имя шрифта}" – шрифт текста.

Цвет в атрибутах HTML может быть задан английским названием из числа стандартных цветов, например, red, green, olive или интенсивностью трех основных цветов (rrggbb):

#??.... – интенсивность красного цвета (red);

#..??.. – интенсивность зеленого цвета (green);

#....?? – интенсивность синего цвета (blue);

интенсивность каждого цвета задается числом от 0 до 255 в шестнадцатеричной системе представления от "00" до "ff", признаком шестнадцатеричного представления является символ # вначале комбинации, например, #ffffff – белый цвет;

Следует отметить, что атрибуты размещаются только в открывающем теге и отделяются друг от друга пробелами. При этом порядок следования атрибутов не имеет значения. Значение атрибута обязательно берется в двойные кавычки только в том случае, если в нем содержатся пробелы. Если в значении атрибута пробелов нет, то кавычки не обязательны.

Внутри открывающего тега **<body>** можно применить целый набор атрибутов, в частности определяющий цветовое оформление фона (при помощи атрибута **bgcolor**) и текста (**text**) сразу на всей странице, например:

<body bgcolor="#00ffff" text="#808080" >.

Самостоятельно.

Применить для форматирования текста созданной Web-страницы следующие теги:

** ** – абзац. Атрибуты:

align="{тип выравнивания}" – тип выравнивания текста в абзаце:

left – влево (по умолчанию);

right – вправо;

center – по центру;

justify – по ширине;

title="{текст}" – всплывающий текст (при наведении мыши).

Например,

 Текст абзаца

<h1> </h1>,.. <h6> </h6> – заголовок 1–го (…6-го) уровня. Атрибуты такие же как у абзаца. По умолчанию текст полужирный.

**
** – следующая строка, непарный тег перевода строки.
** ** – центрирование фрагмента текста.
** ** или ** ** – задается жирность текста.
** ** или ** ** – задается курсив.
<u> </u> – задается подчеркивание.
**** – нижний индекс.
**** – верхний индекс.

2.2.3 Результаты и выводы:

В результате выполнения данного практического задания мы научились создавать простую структуру страницы и форматировать текст с помощью тегов.

2.3 Практическое занятие № 3 (2 часа).

Тема: «Форматирование текста»

2.3.1 Задание для работы:

1. Закрепить знания по форматированию тегов.

2. Создать Web-страницу bank.html

3. Отформатировать страницу, как в примере.

2.3.2 Краткое описание проводимого занятия:

1. Закрепить учебный материал по правилам форматирования текста при помощи тегов HTML (по тексту лекций). Ответить на вопрос – как применяются теги и атрибуты при вложении тегов друг в друга?
2. Создать файл **bank.html**, в котором сформировать текстовую Web-страницу «Банк выгодных кредитов» (заголовок должен быть как на Web-странице, так и в теге **title**) примерного содержания, представленного на рис. 1.24. На рис. 1.25 приведен HTML-код примера. Рекомендуется применить свое оформление и форматирование Web-страницы.
3. Разместить созданную страницу на диске Z в папке **html_files** и продемонстрировать преподавателю.

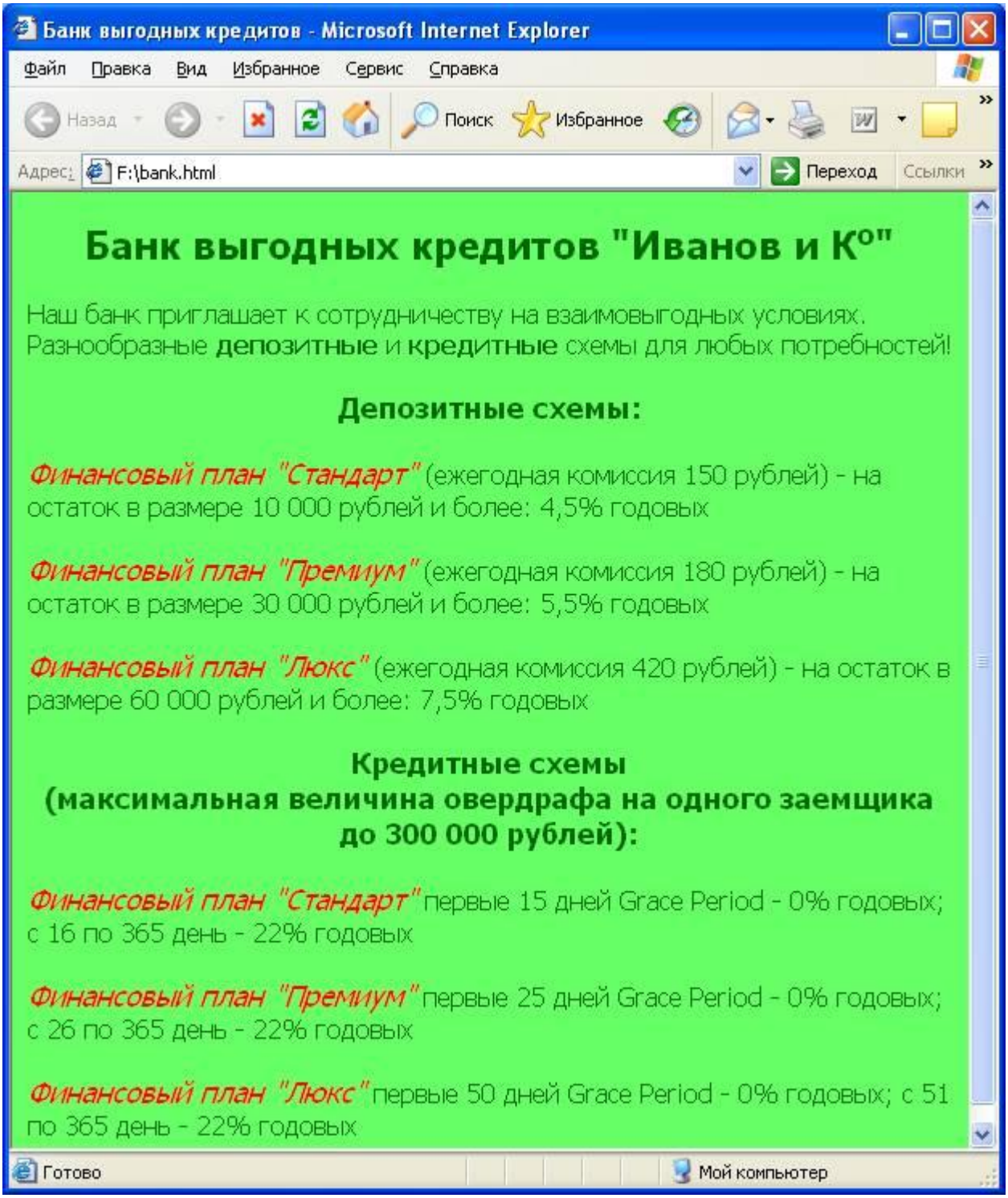


Рис. 1.24. Пример Web-страницы «Банк выгодных кредитов»

```
<html>
<head><title>Банк выгодных кредитов</title></head>

<body bgcolor="#66ff66" text="#006600" >
<font  face="Tahoma"><h2 align="center">Банк выгодных кредитов "Иванов и
К<sup>о</sup>"</h2>
<p>Наш банк приглашает к сотрудничеству на взаимовыгодных условиях.<br>
Разнообразные <strong>депозитные</strong> и <strong>кредитные</strong> схемы для
любых потребностей!</p>
<h3 align="center">Депозитные схемы:</h3>
<p><em><font size="+1" color="#ff0000">Финансовый план "Стандарт" </font></em>
(ежегодная комиссия 150 рублей) - на остаток в размере 10 000 рублей и более:
4,5% годовых </p>
<p><em><font size="+1" color="#ff0000">Финансовый план "Премиум"
</font></em>(ежегодная комиссия 180 рублей) - на остаток в размере 30 000 рублей и
более: 5,5% годовых </p>
<p><em><font size="+1" color="#ff0000">Финансовый план "Люкс"
</font></em>(ежегодная комиссия 420 рублей) - на остаток в размере 60 000 рублей и
более: 7,5% годовых </p>

<h3 align="center">Кредитные схемы <br>(максимальная величина овердрафа на одного
заемщика  до 300 000 рублей):</h3>
<p><em><font size="+1" color="#ff0000">Финансовый план "Стандарт" </font></em>
первые 15 дней Grace Period - 0% годовых; с 16 по 365 день - 22% годовых
</p>
<p><em><font size="+1" color="#ff0000">Финансовый план "Премиум" </font></em>
первые 25 дней Grace Period - 0% годовых; с 26 по 365 день - 22% годовых
</p>
<p><em><font size="+1" color="#ff0000">Финансовый план "Люкс" </font></em>первые 50
дней Grace Period - 0% годовых; с 51 по 365 день - 22% годовых  </p>
</font>
</body>
</html>
```

Рис. 1.25. HTML-код примера Web-страницы «Банк выгодных кредитов»

В приведенном примере код *** *** является кодом символа неразрывного пробела, который применяется в тех случаях, когда в этом месте перенос нежелателен, например, между числом и единицей измерения. В HomeSite кнопка вставки такого символа – **nb** находиться на панели **Common**.

Следует отметить, что переносы в HTML-тексте (по нажатию клавиши Enter) не отображаются на Web-странице и применяются для удобного представления HTML-кода.

2.3.3 Результаты и выводы:

В результате выполнения данного практического занятия мы научились стилизовать веб-страницу в зависимости от требований, к выполняемой работе с помощью html тегов.

2.4 Практическое занятие № 4(2 часа).

Тема: «Ссылки и графика»

2.4.1 Задание для работы:

1. Создать файловую структуру Web-сайта.
2. Создать графику на веб странице.
3. Прикрепить ссылки

2.4.2 Краткое описание проводимого занятия:

В общем, изображения на *web*-страницах могут использоваться двумя способами: в качестве фонового изображения, на котором располагаются элементы основного документа, и изображения, встраиваемые в документ.

На *web*-страницах в подавляющем большинстве случаев используется растровая графика двух форматов: *GIF* и *JPG*. Эти два формата непосредственно поддерживаются всеми браузерами, а для использования большинства других графических форматов потребуются специальные средства.

Встраивание изображений в HTML-документы.Для встраивания изображений в *HTML*-документ следует использовать тэг <*img*>, имеющий единственный обязательный параметр *SRC*, определяющий *URL*-адрес файла с изображением.

<IMG SRC="http://www.server.ru/pic/risunok.gif">

или

<IMG SRC=risunok.gif >

Данный тэг может иметь ряд других параметров.

При включении графического изображения в документ можно указывать его расположение относительно текста или других элементов страницы. Способ выравнивания изображения задается значением параметра *ALIGN* тэга <*IMG*>, который может принимать следующие значения:

· *TOP* – верхняя граница изображения выравнивается по самому высокому элементу текущей строки;

· *TEXTTOP* – верхняя граница изображения выравнивается по самому высокому текстовому элементу текущей строки;

· *MIDDLE* – выравнивание середины изображения по базовой линии текущей строки;

· *ABSMIDDLE* – выравнивание середины изображения посередине текущей строки;

· *BASELINE* или *BOTTOM* – выравнивание нижней границы изображения по базовой линии текущей строки;

· *ABSBOTTOM* – выравнивание нижней границы изображения по нижней границе текущей строки;

· *LEFT* – изображение прижимается к левому полю окна. Текст обтекает изображение с правой стороны;

· *RIGHT* – изображение прижимается к правому полю окна. Текст обтекает изображение с левой стороны.

По умолчанию изображения выравниваются по базовой линии.

В чем разница между базовой линией и нижней границей строки? Или различие между самым высоким элементом строки и самым высоким текстовым элементом строки? Результат действия этих параметров может отличаться в зависимости от содержимого рассматриваемой строки.

Базовая линия (*BASELINE* или *BOTTOM*) – это нижняя часть линии текста, которая проводится без учета нижней части некоторых символов, например, букв типа *j, q, y*. В отличие от выравнивания по базовой линии, при задании значения *absbottom* выравнивание выполняется по нижней части самого низкого элемента в строке, т. е. по одному из символов строки, имеющему элементы, лежащие ниже базовой линии.

Тэг <*IMG*> имеет два необязательных параметра, указывающих размеры изображения при отображении – *WIDTH* и *HEIGHT*. Значения параметров могут указываться как в пикселах, так и в процентах от размеров окна просмотра.

Любой из этих параметров может быть опущен. Если задан только один из параметров, то при загрузке рисунка второй параметр будет вычисляться автоматически из условий сохранения пропорций.

Для тэга <*IMG*> можно задавать параметры *HSPACE* и *VSPACE*, значения которых определяют отступы от изображения, оставляемые пустыми, соответственно, по горизонтали и вертикали.

Изображение на *web*-странице можно поместить в рамку различной ширины. Для этого служит параметр *BORDER* тэга <*img*>. В качестве значения параметра используется число, означающее толщину рамки в пикселах.

Одним из параметров тэга <*img*> является параметр *ALT*, определяющий альтернативный текст. Его указание даёт возможность пользователям неграфических браузеров или пользователям, работающим в режиме отключения загрузки изображений, получить некоторую текстовую информацию о встроенных изображениях.

<IMG SRC=risunok.gif ALT="Это рисунок">

Использование изображения в качестве ссылки. Графические изображения могут использоваться не только в качестве иллюстраций, но и выполнять роль указателей

гипертекстовых связей. Для обеспечения работы изображения в качестве ссылки на другие ресурсы достаточно включить изображение внутрь тэга-контейнера *<A>*.

Любая часть такого изображения будет работать как указатель ссылки на указанный документ. Существуют возможности задания изображений, отдельные фрагменты которого будут указывать на различные ресурсы.

<A HREF=risunok.HTML><IMG SRC=risunok.gif></A>

1. Откройте **Блокнот**.

2. Создайте *HTML*-документ с заголовком **Графика**.

3. Сохраните файл в папку *D:\Users\...\Web\Lab5* с именем *index5.html* .

4. Откройте **Paint**.

5. В **Paint** создайте рисунок размером 50х50 точек и сохраните его в папку *D:\Users\...\Web\ Lab5* в формате *JPG* под именем *bgpic*.

6. В **Paint** создайте рисунок размером 100х200 точек и сохраните его в папку *D:\Users\...\Web\ Lab5* в формате *JPG* под именем *pic1*.

7. В **Paint** создайте рисунок размером 200х200 точек и сохраните его в папку *D:\Users\...\Web\ Lab5* в формате *JPG* под именем *pic2*.

8. Сделайте рисунок *bgpic* фоновым для созданной *web*-страницы.

9. Вставьте рисунок *pic1* в *HTML*-документ, сделайте его ссылкой на документ *about.html*. Для картинки напишите альтернативный текст "Информация о разработчике".

10. Создайте в *HTML*-документе нумерованный список, состоящий из 10 произвольных строк.

11. Сделайте, чтобы текст обтекал картинку с левой стороны.

12. Создайте *HTML*-документ, в нём:

· измените цвет фона;

· создайте заголовок " Информация о разработчике";

· вставьте рисунок *pic2*. Для картинки сделайте рамку;

· создайте абзац, в котором напишите своё Ф.И.О., *e-mail*, факультет, направления подготовки, группу, название дисциплины, № лабораторной работы, год;

· сделайте, чтобы абзац выравнивался по ширине;

· сделайте, чтобы текст обтекал картинку с правой стороны;

· измените цвет, размер и тип шрифта текста абзаца;

· *E-mail* оформите в виде гиперссылки;

· создайте горизонтальную линию. Для линии задайте цвет, длину и толщину. Сделайте, чтобы она выравнивалась по центру.

13. Сохраните файл в папку *D:\Users\...\Web\Lab5* с именем *about.html.*

2.4.3 Результаты и выводы:

В результате выполнения данного практического задания мы научились работать с графикой и ссылками в Html, а также связывать их.

2.5 Практическое занятие № 5(2 часа).

Тема: «Таблицы в HTML»

2.5.1 Задание для работы:

1. Создать HTML страницы.

2. Написание сложной таблицы.

3. Вставка в таблицу текста и изображений.

2.5.2 Краткое описание проводимого занятия:

В *HTML* таблицы используются не только традиционно, как метод представления данных, но и как средство форматирования *web*-страниц.

Описание таблиц должно располагаться внутри раздела документа *BODY*. Документ может содержать произвольное число таблиц, причём допускается вложенность таблиц друг в друга. Каждая таблица должна начинаться тэгом <*table*> и завершаться тэгом <*/table*>. Внутри этой пары тэгов располагается описание содержимого таблицы. Любая таблица состоит из одной или нескольких строк, в каждой из которых задаются данные для отдельных ячеек.

Каждая строка начинается тэгом <*TR*> и завершается тэгом <*/TR*>. Отдельная ячейка в строке обрамляется парой тэгов <*TD*> и <*/TD*> или <*TH*> и <*/TH*>.
Тэг <*TH*> используется обычно для ячеек-заголовков таблицы, а <*TD*> – для ячеек-данных. Тэги <*TD*> и <*TH*> не могут появляться вне описания строки таблицы <*TR*>. Завершающие коды <*/TR*>, <*/TD*>и <*/TH*> могут быть опущены.

Каждую отдельную ячейку внутри таблицы можно рассматривать как область для независимого форматирования. Все правила, которые действуют для управления отображением текста, могут быть использованы для форматирования текста внутри ячейки.

Таблица может иметь заголовок, который заключается в пару тэгов <*CAPTION*> и <*/CAPTION*>. Описание заголовка таблицы должно располагаться сразу же после тэга <*table*> и до первого <*TR*>. Тэг заголовка таблицы имеет необязательный параметр *ALIGN*, принимающий значения *TOP* (заголовок над таблицей) или *BOTTOM* (заголовок под таблицей).

В тэге <*table*> могут использоваться параметры:

· *BORDER* – определяет толщину рамки;

· *CELLSPACING* – определяет расстояние между ячейками;

· *CELLPADDING* – определяет расстояние между содержимым ячейки и её границами;

· *WIDTH* – определяет ширину таблицы;

· *ALIGN* – определяет горизонтальное расположение таблицы в области просмотра. Допустимые значения *LEFT* и *RIGHT*.

Браузер *MS Internet Explorer* разрешает кроме перечисленных пяти параметров использовать параметры:

HEIGHT – определяет высоту таблицы;

BGCOLOR – определяет фоновый цвет таблицы;

BORDERCOLOR – определяет цвет всех элементов рамок таблицы;

BORDERCOLORLIGHT – окрашивает в заданный цвет левый и верхний края всей таблицы и соответственно правый и нижний края каждой ячейки;

bordercolordark – окрашивает в заданный цвет правый и нижний края всей таблицы и соответственно левый и верхний края каждой ячейки;

BACKGROUND – определяет фоновый рисунок для таблицы.

Эти параметры могут применяться как для таблицы в целом, так и для отдельных ячеек.

Для сложных таблиц характерна потребность в объединении нескольких смежных ячеек по горизонтали или по вертикали в одну. Данная возможность реализуется с помощью параметров *colspan* и *rowspan*, задаваемых в кодах <*TD*> или <*TH*>. Форма записи: *coLSPAN=num*, где *num* – числовое значение, определяющее, на сколько столбцов следует расширить текущую ячейку по горизонтали. Применение параметра *ROWSPAN* аналогично, только здесь указывается количество строк, которые должна захватить текущая ячейка по вертикали.

<TABLE WIDTH=50% BORDER=2 CELLSPACING=2 CELLPADDING=2>

<CAPTION>Название таблицы</CAPTION>

<TR> <TH BGCOLOR=green COLSPAN=2>ячейка-заголовок</TH> </TR>

<TR> <TD>ячейка 1</TD> <TD>ячейка 2</TD> </TR>

<TR> <TD>ячейка 3</TD> <TD>ячейка 4</TD> </TR>

</TABLE>

Для форматирования данных внутри ячеек таблицы предусмотрены параметры горизонтального выравнивания *ALIGN* и вертикального выравнивания *valign*.

Параметр *ALIGN* может принимать значения *LEFT, RIGHT* и *CENTER.*

Параметр *valign* может принимать значения *TOP* (по верхнему краю), *BOTTOM* (по нижнему краю), *MIDDLE* (посередине), *BASELINE* (по базовой линии). По умолчанию – *MIDDLE*.

1. Откройте **Блокнот**.

2. Создайте *HTML*-документ с заголовком **Таблицы**.

3. Сохраните файл в папку *D:\Users\...\Web\Lab6* с именем *index6.html* .

4. Создадим невидимую таблицу:

<TABLE>

<TR>

<TD WIDTH=200>

</TD>

<TD WIDTH=800>

</TD>

<TD WIDTH=200>

</TD>

</TR>

</TABLE>

5. Откройте **Paint**.

6. В **Paint** создайте рисунок размером 50х50 точек и сохраните его в папку *D:\Users\...\Web\ Lab6* в формате *JPG* под именем *bgpic*.

7. В **Paint** создайте рисунок размером 100х200 точек и сохраните его в папку *D:\Users\...\Web\ Lab6* в формате *JPG* под именем *pic*.

8. В первой ячейке таблицы напишите: Оренбургский Государственный Аграрный Университет.

Измените цвет текста. Сделайте, чтобы он выравнивался по центру.

9. Во второй ячейке создайте видимую таблицу с заголовком "**Вложенная таблица"** шириной – 600, вида:

Введите произвольный текст в ячейки вложенной таблицы.

10. Для вложенной таблицы:

· измените цвет заливки первой строки;

· в первую ячейку второй строки вставьте фоновый рисунок *bgpic*.

11. В третьей ячейке невидимой таблицы создайте надпись "Разработчик" и оформите её как гиперссылку на документ *about.html*.

12. Создайте *HTML*-документ, в нём:

· измените фоновый цвет;

· создайте заголовок " Разработчик";

· вставьте рисунок *pic*. Для картинки сделайте рамку;

· создайте список, в котором напишите своё Ф.И.О., *e-mail*, факультет, направления подготовки, группу, название дисциплины;

· сделайте, чтобы текст обтекал картинку с правой стороны;

· измените цвет, размер и тип шрифта списка;

· *E-mail* оформите в виде гиперссылки;

· создайте горизонтальную линию. Для линии задайте цвет, длину и толщину. Сделайте, чтобы она выравнивалась по центру;

· создайте надпись:

® *Оренбургский Государственный Аграрный Университет*.

13. Сохраните файл в папку *D:\Users\...\Web\Lab6* с именем *about.html.*

2.5.3 Результаты и выводы:

В процессе выполнения данного практического задания были приобретены навыки работы со сложными таблицами и интеграция изображений в них.

2.6 Практическое занятие № 1 (2 часа).

Тема: «Табличная верстка»

2.6.1 Задание для работы:

1. Создать файловую структуру Web-сайта.

2. Создать файл стилей

3. Создать файл каркас

2.6.2 Краткое описание проводимого занятия:

Создать на "Рабочем столе" каталог "lab4".
Создать в каталоге "lab4" текстовый файл "styles.css", для этого

Запустить текстовый редактор, например, программу "Блокнот" (меню "Пуск"->"Программы"->"Стандартные"->"Блокнот")
В меню "Файл" выбрать пункт "Сохранить как..."
Указать каталог "lab4"
Указать имя файла "styles.css" (имя файла взять в двойные кавычки, для того, чтобы автоматически к названию файла не добавлялось расширение .txt)
Нажать на кнопку "Сохранить"

Начиная с первой строки файла "styles.css" между символами "/*" и "*/" расположить комментарий, указывающий на авторские права в произвольной форме на английском языке, например:

/* Author: Alexey I. Petrov */

Далее в файле "styles.css" вручную задать все необходимые свойства тегов, используемых в Ваших html-страницах.

Например, для тега <body> можно установить свойства отступов от границ окна, равными нулю и цвет фона:

```
body
 {
  margin: 0;
  background-color: lightyellow;
 }
```

для задания цвета и размера текста, размещаемого внутри тегов в файл стилей "styles.css" можно добавить:

```
{
  color: navy;
  font-size: 14pt;
 }
```

для задания свойств заголовков первого уровня <h1></h1>:

```
h1
 {
  color: blue;
  font-size: 36pt;
  font-weight: bold;
  font-family: Sans-Serif, Arial, Tahoma, Helvetica;
  text-align: center;
 }
```

для задания свойств таблиц <table></table>:

```
table
 {
  width: 100%;
  height: 100%;
  border-width: 0;
  border-spacing: 0;
  padding: 0;
  background-color: lightyellow;
 }
```

для задания свойств всех ячеек таблиц <td></td>:

```
td
 {
  border-width: 1;
  border-spacing: 0;
  border-color: green;
  border-style: solid;
  text-align: center;
 }
```

и так далее для всех тегов, которые будут встречаться в html-коде Ваших трех страниц.

Сохранить файл "styles.css" на диск.

Создать в каталоге "lab4" текстовый файл "index.html".
В файле "index.html" вручную разместить html-код страницы.

Например, можно написать основную структуру html-страницы:

```
<html>

<head>

<title></title>

<link rel='stylesheet' type='text/css' href='styles.css'>

</head>



<body>

</body>

</html>
```

Тег <link> подключает внешнюю таблицу стилей "styles.css".
Между тегами <title></title> расположить название страницы, например, "Томск - Главная".
Затем, между тегами <body></body> расположить таблицу, состоящую, например, из четырех строк, а в каждой строке по три ячейки, причем в первой, второй и четвертой строке три ячейки объединены в одну:

```
<table>

<tr>

<td colspan='3'>Содержимое первой ячейки первой строки</td>

</tr>

<tr>

<td colspan='3'>Содержимое первой ячейки второй строки</td>

</tr>

<tr>

<td>Содержимое первой ячейки третьей строки</td>
```

<td>Содержимое второй ячейки третьей строки</td>

<td>Содержимое третьей ячейки третьей строки</td>

</tr>

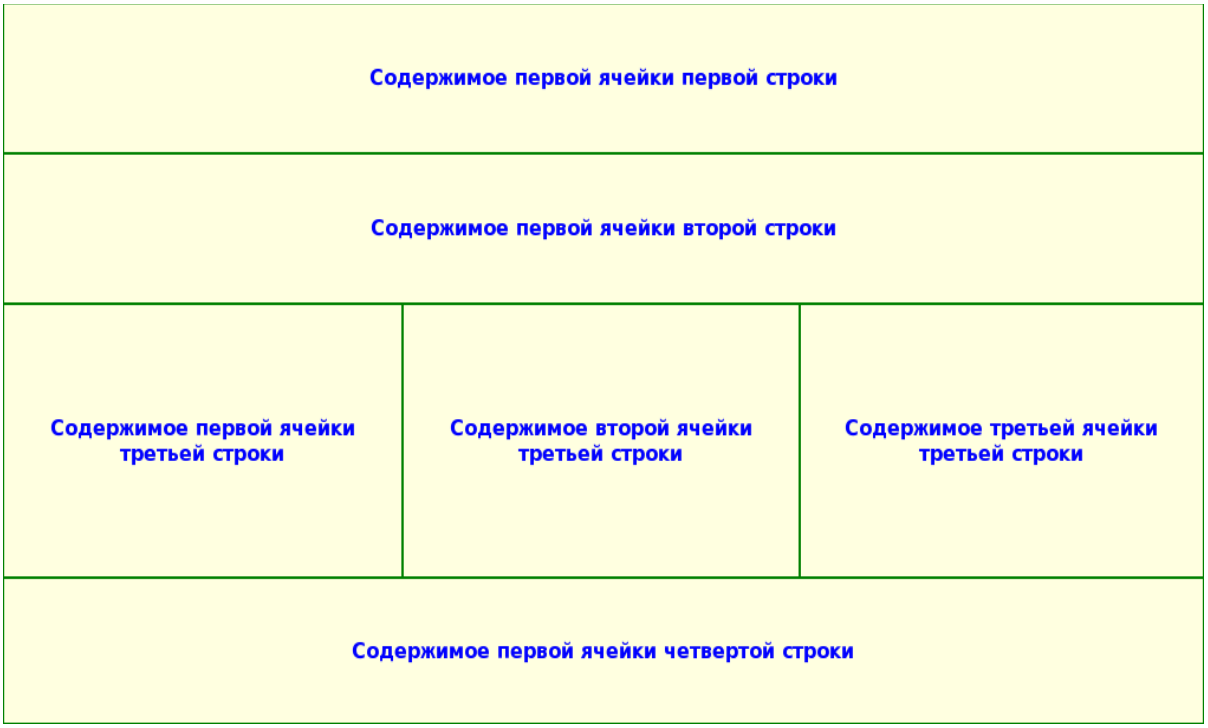
<tr>

<td colspan='3'>Содержимое первой ячейки четвертой строки</td>

</tr>

</table>

Такая таблица в этом случае должна выглядеть примерно так:



2.6.3 Результаты и выводы:

В результате выполнения данного практического задания мы научились создавать табличный каркас и подключать к нему таблицы стилей.

2.7 Практическое занятие № 7 (2 часа).

Тема: «Интерактивные формы HTML»

2.7.1 Задание для работы:

1. Создать файловую структуру Web-сайта.

2. Написать функционал форм в HTML.

3. Задать стили для данных форм.

2.7.2 Краткое описание проводимого занятия:

Одной из наиболее важных функций мобильных приложений является их способность взаимодействовать с человеком. В *HTML* для этого существуют теги отображения на странице интерактивных элементов управления - списков, кнопок, текстовых полей и других активизируемых пользователем элементов управления. С их помощью *пользователь* может предоставлять мобильному приложению тот или иной вид информации. Эта *информация* может обрабатываться двумя способами - непосредственно на мобильном устройстве посредством сценариев JavaScript, код которых включается в разметку страницы, или путем отправки данных на веб-*сервер* для их дальнейшей обработки серверным сценарием. Во всех случаях для этих целей используются интерактивные *элементы управления HTML*. Краткое их обозрение приведено в Табл. 9.1:

Таблица 9.1.

Элемент управления	HTML-элемент	Описание
Однострочное текстовое поле	<input type="text"> <input type="password">	Выводит однострочное текстовое поле для ввода текста. Если для атрибута type указано значение password, вводимые пользователем символы отображаются в виде звездочек (*) или маркеров-точек (●)
Многострочное текстовое поле	<textarea>...</textarea>	Текстовое поле, в которое можно ввести несколько строчек текста
Флажок	<input type="checkbox">	Выводит поле флажка, который можно установить или очистить
Переключатель	<input type="radio">	Выводит переключатель - элемент управления в виде небольшого кружка, который можно включить или выключить. Обычно создается группа переключателей с одинаковым значением атрибута name, вследствие чего можно выбрать только один из них
Кнопка	<input type="submit"> <input type="image"> <input type="reset"> <input type="button">	Выводит стандартную кнопку, активизируемую щелчком мыши. Кнопка типа submit всегда собирает информацию с формы и отправляет ее для обработки. Кнопка типа image делает то же самое, но позволяет вместо текста на кнопке вставить изображение. Кнопка типа reset очищает поля формы от введенных пользователем данных. А кнопка типа button сама по себе не делает ничего. Чтобы ее нажатие выполняло какое-либо действие, для нее

		нужно добавить сценарий JavaScript
Список	<select>... </select>	Выводит список, из которого пользователь может выбирать значения. Для каждого значения списка добавляется элемент<option>

Рассмотрим особенности форматирования этих элементов посредством таблиц стилей.

Формы в мобильных приложениях

HTML - формы существовали с самых ранних времен языка *HTML*, и с тех пор они практически не изменились. Реализуются они тегами <form> ...</form>, внутрь которых помещаются интерактивные *элементы управления*. Элемент <form> удерживает вместе эти *элементы управления*, которые также называются полями. В случае отправки данных на *сервер*, он сообщает браузеру, куда отправлять данные после нажатия пользователем кнопки отправки, предоставляя *URL* в атрибуте action тега <form>. Но если работа с данными будет выполняться на мобильном устройстве клиента посредством сценария JavaScript, то для атрибута action можно просто указать *значение* #. При организации общения с пользователем, когда ввод данных отсутствует, например, в случае навигации (см. далее пример со сменой изображений), *тег* <form> использовать не нужно.

Пример тега <form> с передачей данных на *сервер*:

```
<form id = "simpForm" action="formAction"  method="POST">
```

Пример кода формы для мобильного виджета:

```
<form id = "simplmobileForm" action=#>
  <label for = "accept"> HTML5 </label>
  <input id = "accept" type = "checkbox" value="yes" checked/>
 </form>
```

При определении элементов управления на мобильных устройствах очень важно определить надпись с помощью тега <label>, особенно на устройствах с сенсорным интерфейсом. Если, например, вставить в форму флажок и не снабдить его надписью, то для установки этого флажка пользователю потребуется выполнить касание (щелчок) по области очень малых размеров. При использовании тега <label> *пользователь* может выполнить касание по любой части назначенной флажку надписи. *Тег* <label>связывают с управляющим элементом атрибутом "for", и размещают его рядом с управляющим элементом (до или после). При этом на странице надпись размещается слева или справа от элемента. Пример надписей, связанных с управляющим элементом формы "checkbox" (тестирование в настольном браузере):

Рис. 9.1.

Форматирование интерактивных элементов выполняется в таблице стилей правилами, в которых указываются параметры элементов. Для представленных примеров содержание правил для элементов "checkbox " соответственно:

```
/*Для первого маленького флажка*/
input[type="checkbox"]
{
  width: 120px;
  height: 50px;
}

/*Для второго флажка, который больше*/
input[type="checkbox"]
{
  width: 100px;
  height: 70px;
}
```

2.7.3 Результаты и выводы:

В результате выполнения данного практического задания мы научились работать с формами в HTML и подключать к ним стили.

2.8 Практическое занятие № 8 (2 часа).

Тема: «Фреймы»

2.8.1 Задание для работы:

1. Создать файловую структуру Web-сайта.
2. Создать страницы фреймов и объединить их.

2.8.2 Краткое описание проводимого занятия:

Frame - рамка, кадр. Фреймы делят пространство окна браузера на независимые разделы, в которых отображается различная информация.

Очень удобно использовать фреймы когда необходимо отображать на экране данные из разных источников. Чтобы сделать фрейм, надо создать новую Вэб-страницу, с тэгами <FRAMESET><FRAME>.

Дескриптор <FRAMESET> формирует набор фреймов, которые делят пространство окна на строки и столбцы. Далее необходимо задать значения высоты/ширины всех строк/столбцов, выраженные в процентах относительно текущих габаритов окна браузера, пикселях или в виде символа звездочки. Символ звездочки говорит о том, что размеры фреймов зависят от габаритов остальных фреймов страницы.

Дескриптор <FRAME> служит для определения структуры и содержимого конкретного фрейма.

Ниже, в качестве примера, приведен код страницы, которую можно посмотреть здесь.

```
<html>
<head>
<title>Пример работы с фреймами</title>
</head>
<frameset rows="200,*">
<frame name="frame1" src="lsn017.html">
<frame name="frame2" src="lsn016.html">
</frameset>
</html>
```

2.8.3 Результаты и выводы:

В результате выполнения данного практического задания мы создали простые фреймы и объединил их между собой.

2.9 Практическое занятие № 9 (2 часа).

Тема: «Каскадные таблицы стилей CSS

2.9.1 Задание для работы:

1. Создать таблицу и заполняем ее текстом.

2. Задать нужные стили в соответствии с нужным макетом.

2.9.2 Краткое описание проводимого занятия:

1. Создайте связанную таблицу стилей для своего мини-сайта, рассказывающего про вас и ваши увлечения. Опишите в таблице стилей цветовую схему вашего сайта, используемые шрифты, создайте отдельный класс для ссылок основного меню, так чтобы они отличались визуально.

2. Для полученного макета html-страницы создайте внедренную таблицу стилей, так, чтобы ваш макет визуально напоминал дизайн-макет, подготовленный дизайнером сайта (рисунок 36). Обратите внимание на цвета, отступы, заголовки, оформление ссылок на листе.

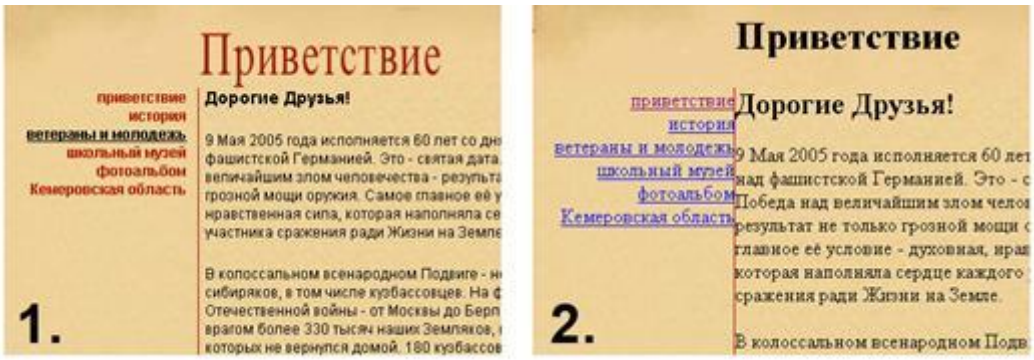


Рис. 36. Дизайн-макет страницы (1), которая должна получиться у вас после создания таблицы стилей у предложенной html-страницы (2).

2.9.3 Результаты и выводы:

В результате выполнения данного практического занятия мы научились работать с макетом и на основе его стилей создавать свою разметку страницы.

2.10 Практическое занятие № 10 (2 часа).

Тема: «Форматирование блоков. Форматирование текста»

2.10.1 Задание для работы:

1. Создать файловую структуру Web-сайта.
2. Выполнить несколько заданий, связанных с форматированием текста.
3. Создать заголовки разного уровня.

2.10.2 Краткое описание проводимого занятия:

Форматирование текста.

Создайте web страницу (с заголовком) которая бы содержала стихотворение из трех строф. Фон страницы должен отличаться от черного. В начале стихотворения должно быть написано его название. Выровняйте строфы следующим образом: мальчики - правая - центр – левая, девушки левая - центр - правая

Первая строфа отформатирована следующим образом:

Длина слова							

Тип формата	выделенный	*наклонный*	подчеркнутый	нижнии индекс	верхний индекс	uni spaţiat	~~зачеркнут~~

Во второй строфе замените фон. Покажите каждую строку текста различными цветами увеличивая размер символов.

В третьей строфе покажите каждую строку текста уменьшая размер символов.

Ниже создайте орнамент из горизонтальных линий, а под ними напишите имя автора.

Luceafărul

A fost odată ca-n ~~poveşti~~

A fost ca niciodată

Din rude mari împărăteşti

O prea frumoasă fată

Şi era una la părinţi

Şi mândră-n toate cele

Cum e fecioara între sfinţi

Şi luna între stele

Din umbra falnicilor bolţi

Ea pasul şi-l îndreaptă

Lângă fereastra unde-n colţ

Luceafărul aşteaptă

Mihai Eminescu

Упражнение 1–Создать самый простой документ с использованием тегов:

<html>, <head> , <title>, <body>, <H1>, <P>,

Заголовки, выравнивание заголовков и параграфов с использованием тегов:

<html>, <head>, <title>, <body>, <H1 ALIGN=CENTER>, <H2 ALIGN=RIGHT>, <H6 ALIGN=LEFT>, <P>, <DIV>, <DIV ALIGN=CENTER>

1. Отобразите на экране текст "Hello, World!" в виде заголовков в порядке уменьшения от h1 до h6 и увеличения от h6 до h1

2. Повторите упражнение 1, но со следующим расположением названия: слева, в центре, вправо самым экономичным способом написания HTML кода

· **Принудительное прекращение абзаца, комментарии с использованием тегов:**

<!- -->, <html>, <head>, <title>, <body>, <H4 ALIGN=LEFT>, <P>, <H5> и получение результатов .

2.10.3 Результаты и выводы:

В результате выполнения данного практического задания мы закрепили навыки форматирования текста и выполнили три практических задания.

2.11 Практическое занятие № 11 (2 часа).

Тема: «Слои»

2.11.1 Задание для работы:

1. Создать HTML макет сайта.

2. Подключить стили, в соответствии с заданием.

2.11.2 Краткое описание проводимого занятия:

Цель работы – ознакомление с основами языка HTML, приемами форматирования текста, получение практических навыков создания Web-страниц.

Теоретические сведения

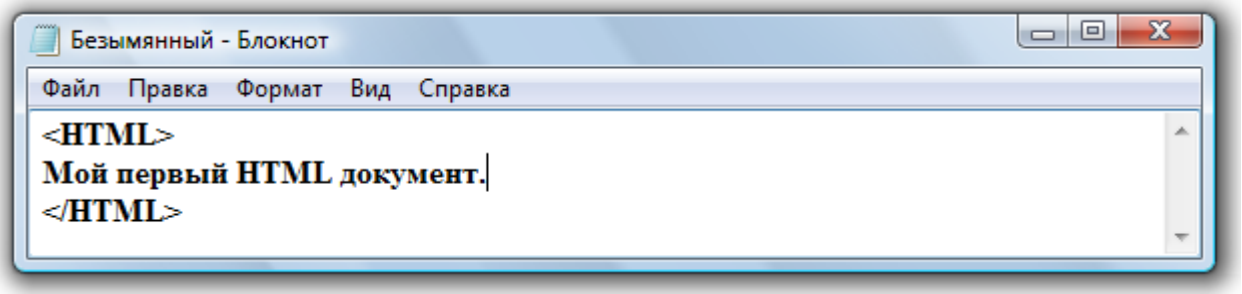
Язык HTML (Hyper Text Markup Language - язык разметки гипертекста) является языком форматирования текстовых документов для представления их в WWW - "всемирной паутине".

HTML-документ – это обычный текстовый файл в формате ASCII, который содержит специальные маркеры, управляющие отображением компонентов документа на странице.

Для просмотра HTML документа необходима специальная программа - браузер. Для написания HTML документов достаточно простейшего текстового редактора, не добавляющего в текст никаких специальных символов. При работе в Windows наиболее простым вариантом является использование редактора Notepad (он же Блокнот в русских версиях Windows). Запуск редактора осуществляется через кнопку "Пуск"--->"Стандартные"--->"Блокнот".

Создание HTML файла

1. Создайте в каталоге группы свой личный каталог, в нем для каждой лабораторной работы необходимо создавать отдельный каталог, например lab1, lab2 и т. д.
2. Запустите текстовый редактор Notepad (Блокнот): "Пуск"--->"Стандартные"--->"Блокнот";
3. Установите "Формат"--->"Перенос по словам" (или проконтролируйте, чтобы там стояла галочка) и наберите текст согласно образцу:

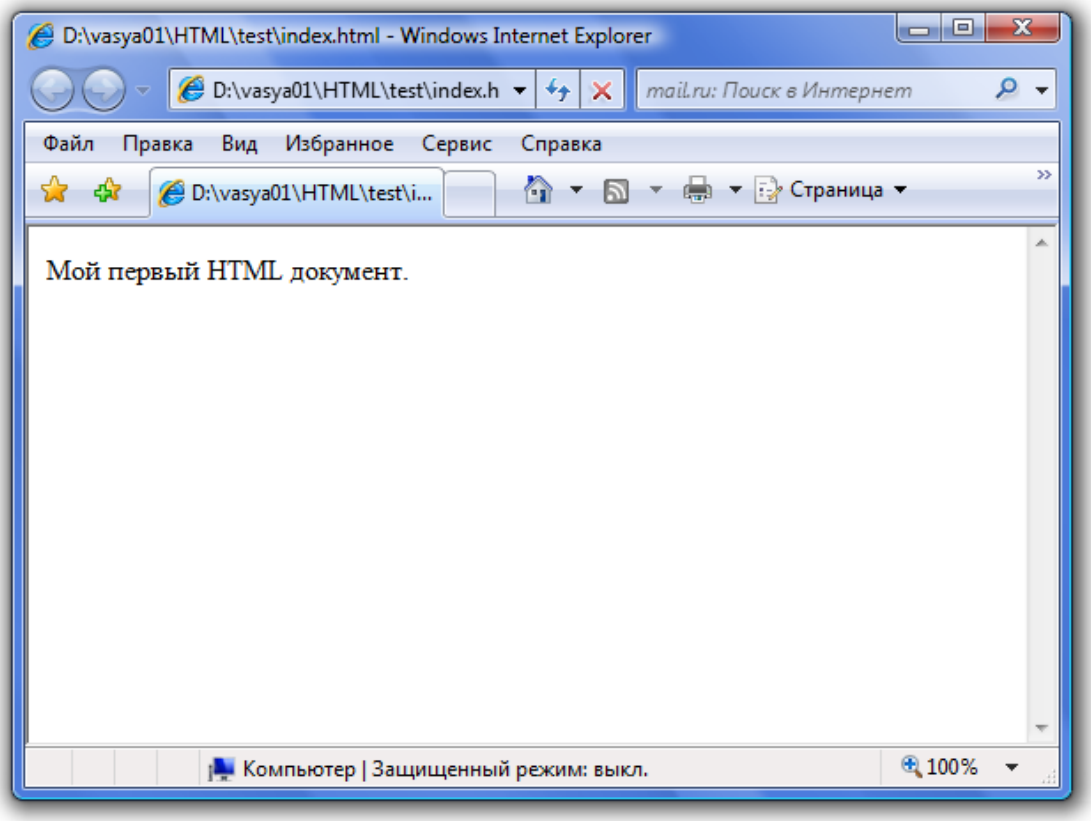


4. Сохраните документ: "Файл"--->"Сохранить как"

5. Выберите папку для сохранения lab1
6. Выберите "Тип файла": Все файлы
7. Выберите "Имя файла": index.html
8. Нажмите кнопку "Сохранить"

Просмотр HTML файла при помощи браузера

1. Зайдите в каталог lab1 своего личного каталога
2. Если указанные выше действия (по созданию HTML-файла) были выполнены правильно, в каталоге lab1 будет находиться файл index.html
3. Двойной клик левой кнопкой мыши по файлу запустит браузер для просмотра Вашего первого HTML документа:



Редактирование HTML файла

1. Для внесения изменений в HTML документ проще всего пользоваться контекстным меню. Кликните по файлу правой кнопкой мыши и оно появится;
2. Выберите "Открыть с помощью" ---> "Блокнот" и редактируйте файл;
3. Для сохранения файла используйте "Файл"---> "Сохранить" или комбинацию клавиш "Ctrl+S";
4. Для просмотра файла в браузере не обязательно закрывать Блокнот;
5. Запустите браузер не закрывая Блокнот (как обычно, двойным левым кликом);
6. Вносите изменения в Блокноте и сохраняйте их ("Файл"---> "Сохранить" или "Ctrl+S");
7. Переключитесь на браузер и выберите обновление содержания клавишей "F5" (для Internet Explorer) или кнопкой с изображением циклических стрелочек на панели инструментов;

8. При создании HTML-документов можно переключаться между редактором и браузером (сохраняя изменения в редакторе и обновляя изображение в браузере) до получения требуемого результата.

Подобным образом создаются, редактируются и просматриваются все остальные HTML-документы в данном лабораторном практикуме.

Дескрипторы/Теги

Тегами называют специальные управляющие коды, записываемые в тексте в угловых скобках. Теги определяют составные части документа, расположение объектов на странице, параметры форматирования текста, позволяют организовать гипертекстовые ссылки и т.д. Когда программа просмотра (браузер), сканируя документ, встречает маркер, она воспринимает его как команду, которую надо расшифровать и исполнить.

Теги HTML состоят из угловых скобок «<» и «>», в которые заключено название тега. Название может быть указано как прописными, так и строчными буквами. В общем случае теги также могут содержать перечень атрибутов и их значений. Атрибуты изменяют параметры элементов, задаваемых тегами. В этом случае формат тега имеет вид:

<НАЗВАНИЕ атрибут_1=значение_1 атрибут_2=значение_2 …>

Если значение атрибута содержит пробелы, то оно должно быть заключено в кавычки:

<НАЗВАНИЕ атрибут=”значение атрибута”>

Все теги делятся на 2 вида: простые и парные. Простые теги обычно используются для реализации какого-либо элемента документа, например, принудительного перевода строки (тег
), горизонтальной черты (<HR>) или подключения изображения (<IMG …>)

Особенностью парных тегов является то, что они не только реализуют какие-либо элементы, но и могут содержать в себе другие теги или просто текст. В таких случаях два тега и часть документа, отделенная ими, образуют блок, называемый HTML-элементом или контейнером. Тег, обозначающий начало контейнера называют открывающим тегом, а обозначающий конец – закрывающим. Закрывающий тег образуется из открывающего добавлением слеша «/» перед названием тега.

Например, все содержимое текстового файла, являющегося документом HTML, должно находиться между тегами <HTML> и </HTML>. Т.е. любой HTML-файл имеет следующий вид:

<HTML>

Содержание

</HTML>

Атрибуты парного тега записываются в открывающем теге, закрывающий тег атрибутов содержать не может.

Структура HTML документа

Простой HTML-документ состоит из двух частей: заголовка и тела документа. Заголовок представляет собой обобщенное описание документа. В теле документа находится его содержательная часть. Начало и конец заголовка документа задают парные теги <HEAD> и </HEAD>, а начало и конец тела документа – теги <BODY> и </BODY>. Таким образом, все простые документы имеют вид:

<HTML>

<HEAD>

Заголовок документа

</HEAD>

<BODY>

Содержательная часть документа

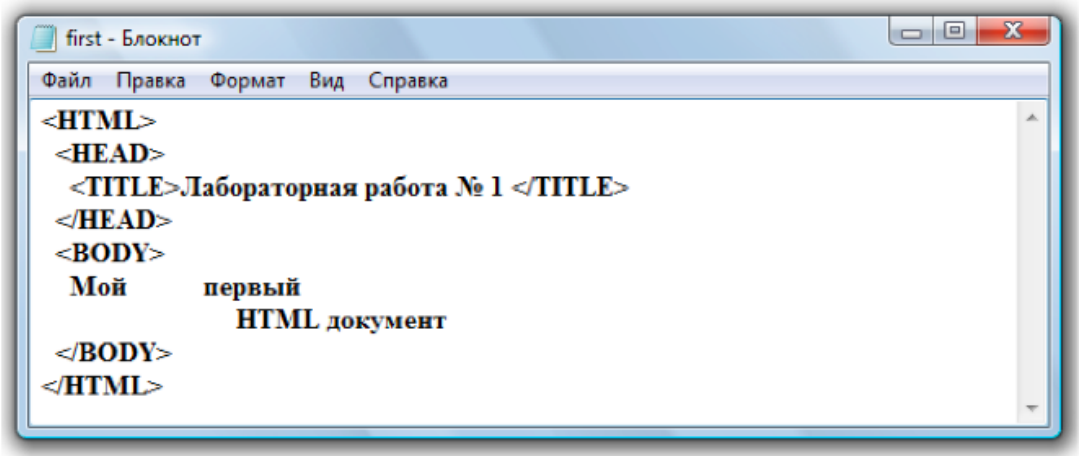
</BODY>

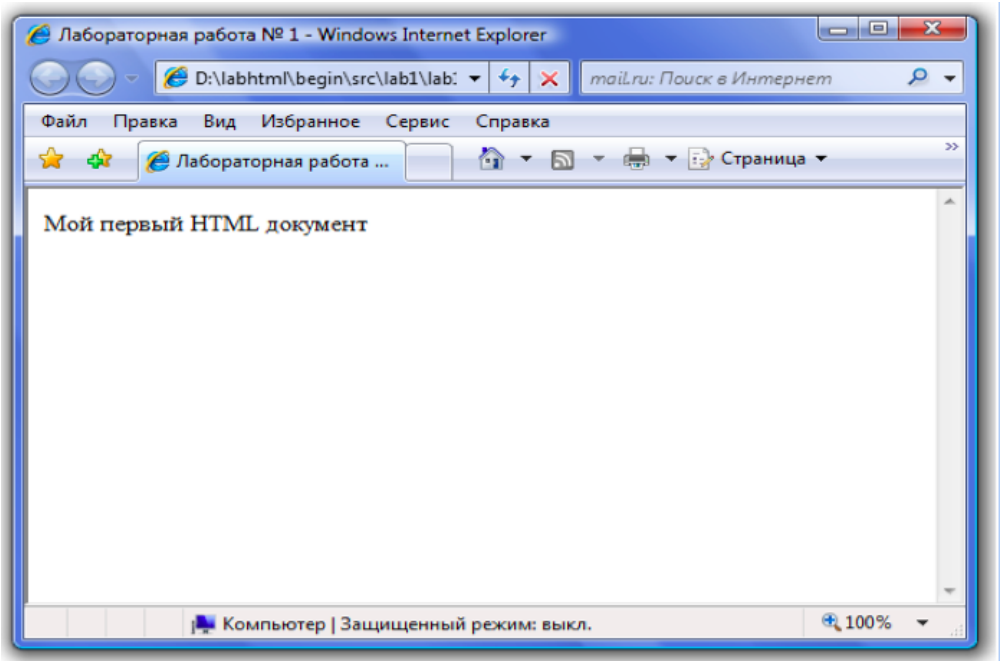
</HTML>

Внутри контейнера <HEAD> размещаются элементы <TITLE>, <META> и <BASE>.

Элемент <TITLE> содержит название документа, отображаемое обычно в строке заголовка окна браузера. Например,

<TITLE>Лабораторная работа №1</TITLE>





Обратите внимание, что вверху окна браузера появилось название документа «Лабораторная работа №1», это результат действия тега <TITLE>.

Роботы многих поисковых систем используют содержание контейнера <TITLE> для создания поискового образа документа.

Элемент <META> содержит управляющую информацию, которую браузер использует для правильного отображения и обработки содержания тела документа. Основной формат тега <META> таков:

<META [name=”имя мета-записи”] [http-equiv=”имя_HTTP-операции”] content=”текст”>

Практика показывает, что можно указывать одновременно и атрибут name, и атрибут http-equiv с одинаковыми значениями. Это связано с тем, что одни роботы индексирования анализируют содержание META-элемента по атрибуту name, а другие – по атрибуту http-equiv.

В каждом теге <META> допускается использование только одной мета-записи, поэтому, когда нужно использовать несколько мета-записей, создают целый список из тегов <META>.

Мета-запись keywords используется для перечисления слов и выражений, максимально соответствующих теме страницы. Поисковые сервера анализируют эти слова, когда определяют, соответствует ли данная страница запросу. Значением параметра keywords обычно служит перечень ключевых слов данного документа. Например

<META name=”keywords” content=”лабораторная работа, HTML, тег, гиперссылка, фрейм”>

Мета-запись description позволяет создать аннотацию страницы, которая отображается в качестве пояснения к ссылке на документ в отчете поисковой машины о выполненном запросе, например

<META name=”description” content=”Лабораторный практикум по HTML содержит четыре работы, позволяющие освоить основы языка HTML и научиться создавать привлекательные Web-страницы”>

Мета-записи author и copyright предназначены для защиты прав авторов страницы:

<META name=”author” content=”И.И.Иванов, ivanov@mail.ru”>

<META name=”copyright” content=”описание авторских прав”>

С помощью мета-записи content-type можно указать браузеру тип и кодировку Web-страницы, например

<META http-equiv=content-type content=”text/html; charset=windows-1251”>

Разметка гипертекстовых ссылок обычно выполняется в частично заданных (относительных) адресах, когда URL задается относительно текущего местоположения документа. Такой стиль разметки удобен тем, что при переносе всего дерева документов в другое место не потребуется менять систему гипертекстовых ссылок внутри документов. По умолчанию в качестве базы выбирается каталог, в котором размещен HTML-документ. Если использование текущего каталога в качестве базы неудобно, можно с помощью тега <BASE> назначить URL, относительно которого будут определяться частичные адреса. Например,

<BASE href=”http://i5.bstu.spb.ru/portal/”>

Тело документа или содержимое Web-страницы ограничивается маркерами <BODY> и </BODY>. Оно может содержать:

– заголовки (H1 – H6);

– элементы на уровне текста;

– гипертекстовые ссылки;

– мультимедийные фрагменты.

Элемент <BODY> имеет набор ключевых атрибутов, используемых для того, чтобы задать повторяющееся фоновое изображение, дополнительный цвет фона и цвет, который будет применяться при печати на экране обычного текста и гипертекстовых связей:

– background определяет адрес URL, откуда будет браться фоновое изображение документа;

– bgproperties запрещает прокрутку фонового изображения вместе с содержимым документа, если установлено значение fixed;

– bgcolor определяет цвет фона для тела документа;

– text задает цвет отображаемого на экране текста и обычно используется при изменении фонового цвета атрибутами bgcolor и background;

– link устанавливает цвет отображения не выбранных пользователем гипертекстовых связей;

– alink задает цвет, которым будут выделяться в тексте гипертекстовые связки в тот момент, когда пользователь щелкает по ним клавишей мыши;

– vlink определяет цвет отображения уже проверенных пользователем гипертекстовых связей;

– leftmargin устанавливает ширину левого поля документа в пикселях;

– topmargin устанавливает размер верхнего поля документа в пикселях.

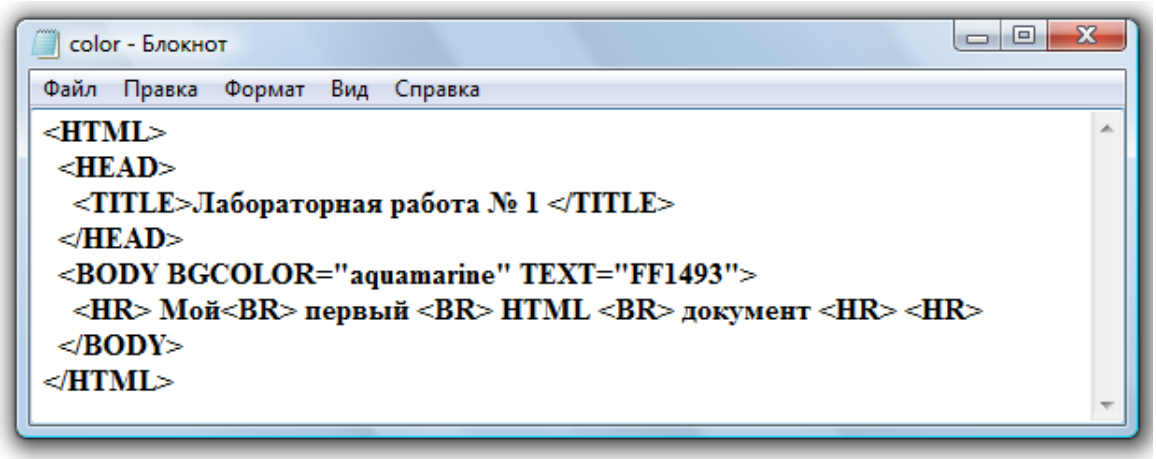
Например,

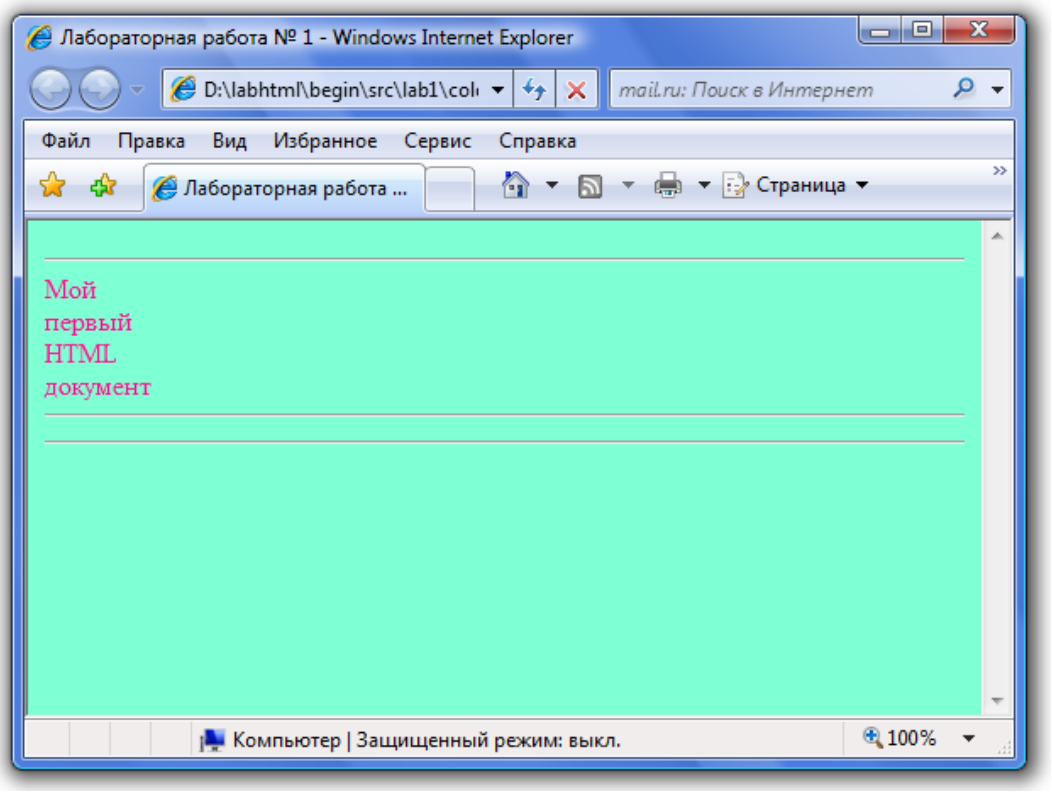
<BODY background =”picture1.jpg” bgproperties =fixed leftmargin =100>

Текст на фоне рисунка, смещенный вправо на 100 пикселей

</BODY>

Цвета в языке HTML задаются по схеме RGB шестнадцатеричными числами (например, link=“#C0FFC0”) или одним из шестнадцати общепринятых названий для цвета. Первоначально эти цвета были выбраны в соответствии со стандартными цветами, которые использовала палитра VGA. Названия цветов и соответствующие значения RGB приведены в таблице цветов.





Задание

Создайте Web-страницы, соответствующие тематике варианта индивидуального задания. Каждая страница должна включать в себя обязательные элементы: заголовок окна, заголовки

различных уровней, текстовое наполнение.

Таблица цветов

Цвет	Название	Шестнадцатеричный код
Черный	Black	#000000
Серебряный	Silver	#C0C0C0
Серый	Gray	#808080
Белый	White	#FFFFFF
Бордовый	Maroon	#800000
Красный	Red	#FF0000
Пурпурный	Purple	#800080
Фиолетовый	Fuchsia	#FF00FF
Зеленый	Green	#008000
Лимонный	Lime	#00FF00
Оливковый	Olive	#808000
Желтый	Yellow	#FFFF00
темно-синий	Navy	#000080
Синий	Blue	#0000FF
Голубой	Teal	#008080
зеленовато-голубой	Aqua	#00FFFF

2.11.3 Результаты и выводы:

В результате выполнения данного практического задания мы закрепили навыки работы со слоями, а также изучили HTML палитру и научились работать с ней.

2.12 Практическое занятие № 12 (2 часа).

Тема: «CSS верстка»

2.12.1 Задание для работы:

1. Создать блочную верстку сайта.
2. Создать табличную верстку сайта.
3. Задать верстkам нужные таблицы стилей.

2.12.2 Краткое описание проводимого занятия:

С плоской версткой все предельно просто. Если Вы в качестве базовых элементов страницы не используете ни таблицы, ни блоки, то это и есть плоская верстка. Сейчас сайты так не делают, поэтому идем дальше.

Табличная верстка

Достоинства и недостатки табличной верстки:

	Недостатки таблиц:
• Простое создание колонок • "Резиновая" верстка • "Склейка" изображений • Однозначная трактовка браузерами	• Долгая загрузка • Громоздкий код • Плохая индексация поисковиками • Нет разделения содержимого и оформления • Несоответствие стандартам • Невозможность конвертации в мобильную версию

Пример табличной верстки

Вот что мы хотим получить:

Рис. 6.1. Пример табличной верстки

Впрочем, у Вас уже достаточно знаний, чтобы
выполнить данный пример самостоятельно:
По макету с рис. 6.1. сверстать страницу табличной верстkой.

Применение табличной верстки Целесообразно в том случае, если

- Высота колонок должна быть одинаковой
- Макет должен занимать всю высоту окна браузера, независимо от объема информации
- Нет времени на сложную верстку

Блочная верстка

Слой — это элемент веб-страницы, созданный с помощью тега `<div>`, к которому применяется стилевое оформление. Благодаря этому тегу HTML-код распадается на ряд четких наглядных блоков, за счет чего верстка слоями называется также блочной верстkой. Код при этом получается более компактным, чем при табличной верстке, к тому же поисковые системы его лучше индексируют.

Сейчас я Вам продемонстриую как разметить страницу с рис. 6.2.

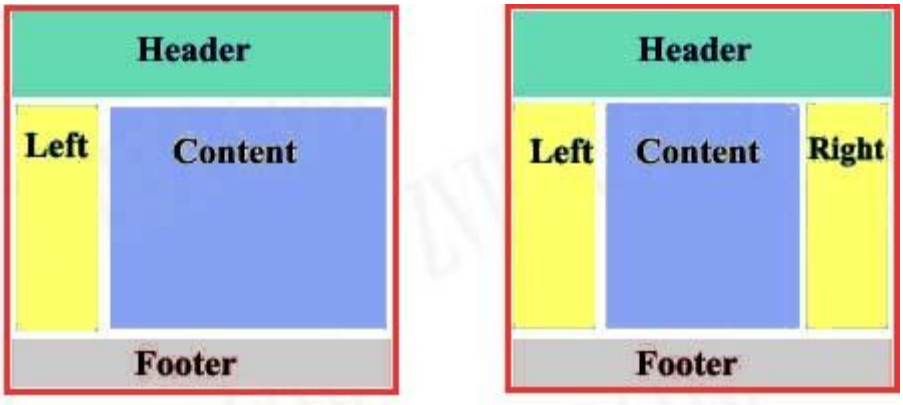


Рис. 6.2. Пример для блочной разметки

На рис 6.2. представлены варианты для резиновой верстки с двумя и тремя колонками. Давайте выполним разметку двух колонок по шагам:

Шаг 1

Сначала, как обычно, выполним разметку HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
 <html>
 <head>
 <meta http-equiv="Content-
Type"content="text/html; charset=windows-1251">
 <title>Блочная верстка</title>
 <link href="style_1.css" rel="stylesheet"
type="text/css">
 </head>
   <body>
                <div id="container">

                        <div id="header">

        <h1>Header</h1>
                        </div>

                        <div id="left">
                        <h3>left Content</h3>
                        </div>

                      <div id="content">
                        <h1>Main Content </h1>
                        <p>Lorem ipsum dolor
sit amet, consectetuer adipiscing elit.
                        Praesent aliquam,  justo
convallis luctus rutrum, erat nulla fermentum
                        Integer turpis arcu,
pellentesque eget, cursus et, fermentum ut,
sapien.
                        Fusce metus mi, eleifend
sollicitudin, molestie id, varius et, nibh.
                    Donec nec libero.</p>
                        <h1>Content</h1>
                        <p>Lorem ipsum dolor
sit amet, consectetuer adipiscing elit.</p>
                        <p>Quisque ornare risus
quis  ligula Phasellus tristique
                        purus a augue
condimentum adipiscing. Aenean  sagittis.
                        Etiam leo pede, rhoncus
venenatis, tristique in, vulputate at, odio.</p>
                        </div>
                        <div id="footer">

        <p><strong>Footer</strong></p>
                        </div>
                </div>
```

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

Обратите внимание на присутствие базового (главного) контейнера, который уже содержит все остальные. Наличие этого главного контейнера позволяет регулировать общие парамеры страницы, что значительно облегчает верстку слоями.

Результат получится вот такой:

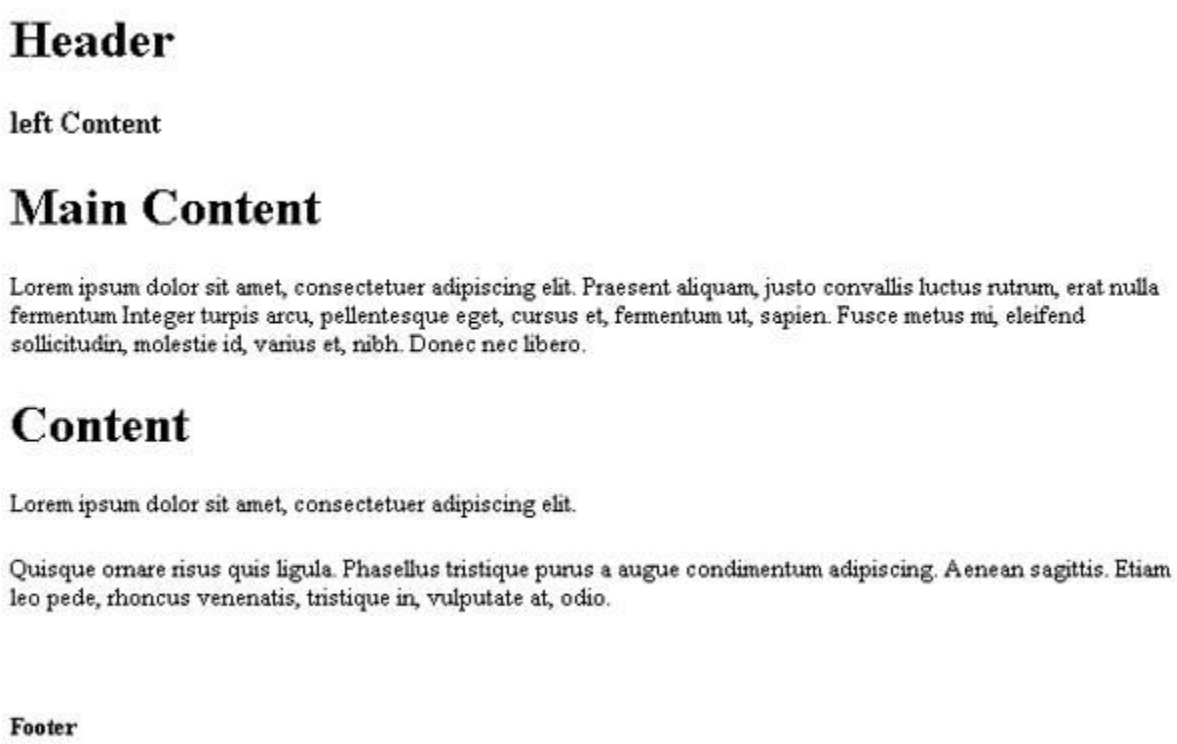


Рис. 6.3. Пример для блочной разметки. Шаг 1.

Оно и понятно, ведь стиль-то CSS у нас не написан. Итак, займемся стилями.

Шаг 2

```
/* CSS Document */
body, html {
margin:0px; /*Это хорошая практика обнулять
поля и отсупы, т.к.
 различные браузеры их по разному
воспринимают. */
padding:0px;
text-align:center; /*Выравниваем макет
(содержимое элемента body
будет посередине) по центру в старых версиях
браузеров */
}
#container{
margin:0 auto;  /*выравниваем макет по центру в
современных браузерах */
```

```
width:650px;
}
/*Здесь пишем стили для шапки сайта */
#header{
background-color:#64dab2;
}

/*Здесь пишем стили для левой колонки сайта */
#left{
background-color:#ffff66;
}

/*Здесь пишем стили для блока контента */
#content{
background-color:#83a0f3;
}
#content h1 {
margin:0px; /* Обнуляем отсупы для заголовка
первого уровня,
находящегося в блоке контента.*/
}
/*Здесь пишем стили для подвала сайта */
#footer{
background-color:#ccc;
}
```

После применения данного стиля style_1.css результат будет следующим:

Рис. 6.4. Пример для блочной разметки. Шаг 2. Первоначальное стилевое оформление

Теперь необходимо блоки Left и Content поставить рядом, как на рис. 6.2.

Меняем ширину блоку Left на 150px и устанавливаем для него обтекание:

```
  #left{
...
width:150px;  /*ширина колонки */
```

```
float:left;  /*обязательное выравнивание по
левому краю,
с включением обтекания*/
}
```

Смотрим результат, получаем не совсем то, что хотели (*рис. 6.4. а)*:

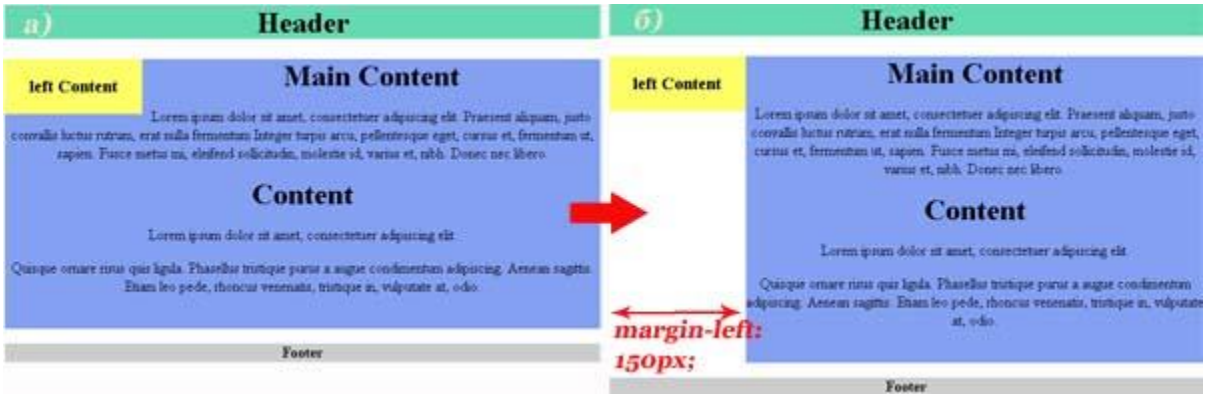


Рис. 6.4. Шаг 2. Обтекание блоков

Для того, чтобы из рис. 6.4.а) получить рис 6.4. б) необходимо сдвинуть блок Content вправо на ширину блока Left:

```
#content{
        background-color:#83a0f3;
        margin-left:150px;
}
```

Тоже самое можно сделать иначе, установив блоку Left `width:20%` и `float:left;`, а блоку Content `width:80%` и `float:right;` (см. пример в лекциях)

При этом не забудьте поставить для блока footer значение `cltar:both;`, иначе при увеличении количества текста в предыдущих блоках может получиься седующий эффект:

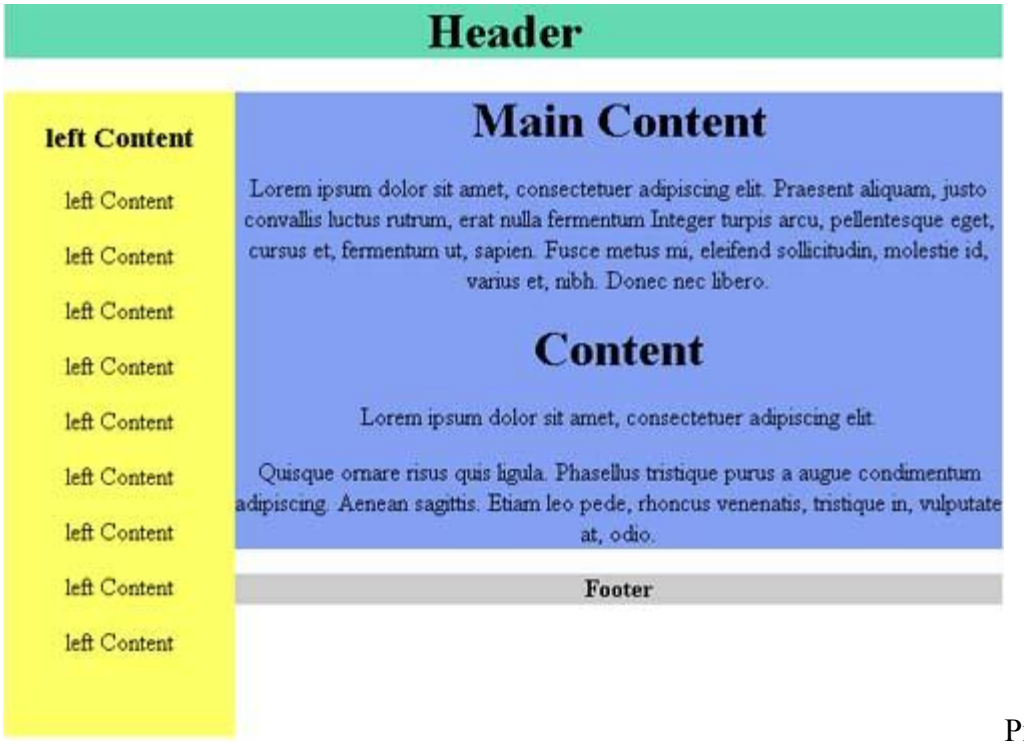


Рис. 6.5.

Необходима отмена обтекания блоков

Итак, Ваша задача По макету с рис. 6.2 (б) разработать трехколочный блочный макет, используя в качестве базы двухколочный и растянуть содержимое на всю ширину страницы.

2.12.3 Результаты и выводы:

В результате выполнения данного практического задания мы научились вестать блочно и таблично, создавать удобную структуру HTML сайта.

2.13 Практическое занятие № 13 (2 часа).

Тема: «Структура HTML документа»

2.13.1 Задание для работы:

1. Создать файловую структуру страницы.
2. Указать нужные стили.
3. Прикрепить ссылки и изображения

2.13.2 Краткое описание проводимого занятия:

1. Создайте папку, назовите ее **myhouse**. Будет правильно и профессионально, если имена всех папок и файлов Вы будете создавать латинскими строчными буквами. Задавайте имена короткими и смысловыми.
2. Наберите в Блокноте текст из листинга на рисунке 1.2. Лучше пользоваться не тем Блокнотом, который идет в поставке Windows, а более «продвинутым», например

Notepad++. Код в листинге на рисунке 1.2 в дальнейшем будет служить нам шаблоном, чтобы каждый раз не набирать структуру html-документа. Конструкцию DOCTYPE скопируйте отсюда

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

3. Сохраните файл под именем **shablon.html** в папке **myhouse**, при этом в поле **Тип файла** установите **All types** (рисунок 1.6), иначе Ваша web-страничка потом не откроется в браузере.

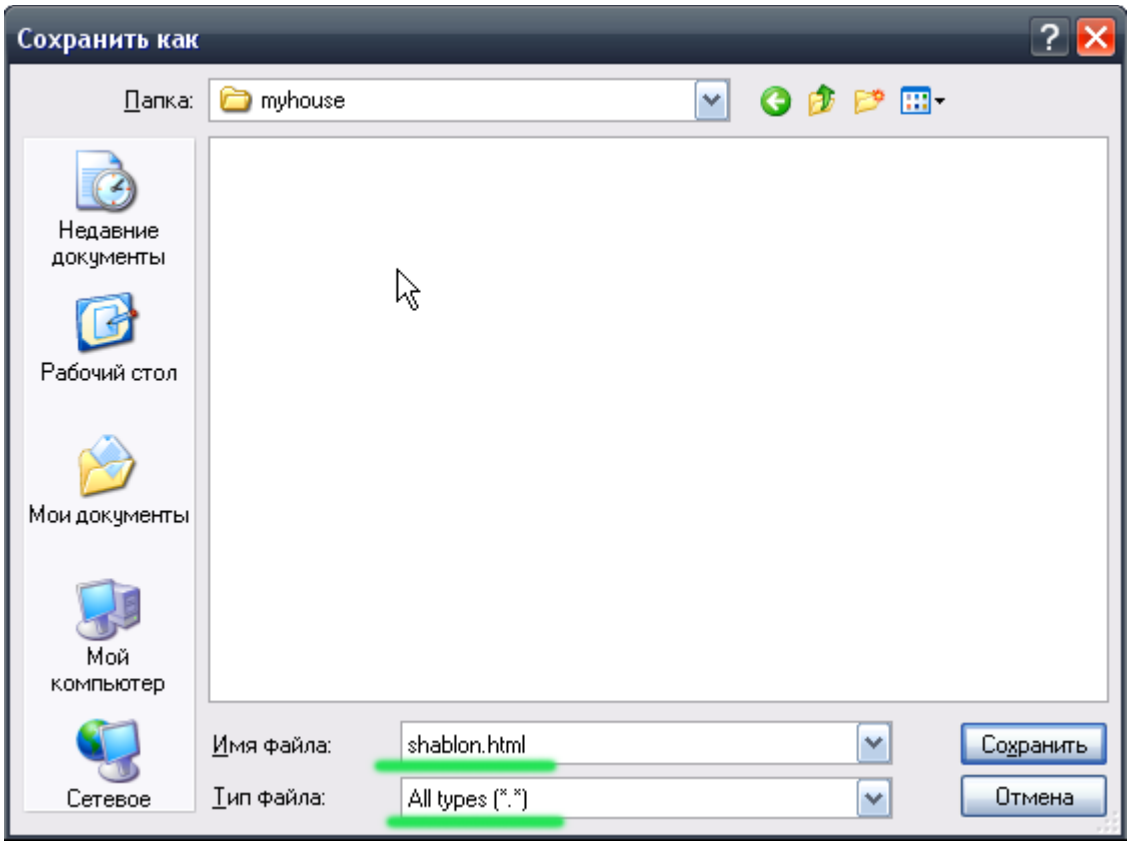


Рисунок 1.6

4. После сохранения запустите **shablon.html** двойным щелчком. В результате Ваш файл будет выглядеть следующим образом (рисунок 1.7).

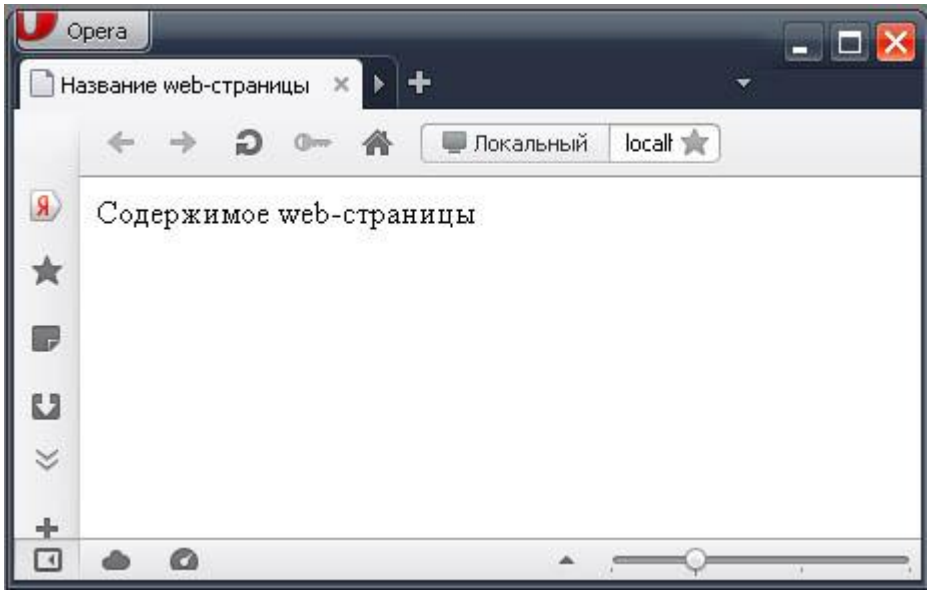


Рисунок 1.7

Если у Вас не получился результат, как на рисунке выше, значит, в коде скорей всего есть ошибка. Достаточно одного неверного символа и web-страница не будет отображаться корректно. Еще раз сверьте код с листингом на рисунке 1.2.

5. Внутри папки **myhouse** создайте папку **public_html**. Таким именем обычно называется папка, в которой хранится сайт при размещении на настоящем хостинге (также эта папка может называться www).
6. Сохраните файл **shablon.html** в папке **public_html** под новым именем**main.html**.
7. Из папки **CD/html_css_1** откройте файл**text_main.txt** в Notepad++, а также откройте файл, сохраненный под именем **main.html**
8. Скопируйте весь текст из файла **text_main.txt** и вставьте в файл**main.html** вместо фразы «Содержимое web-страницы». В теге <title> напишите слово «Главная». Вот так**<title>Главная</title>.**
9. Сохраните изменения и просмотрите файл **main.html** в браузере. Вы увидите не отформатированный текст. Даже переносы строк, которые есть в исходном тексте, браузер не делает (рисунок 1.8).

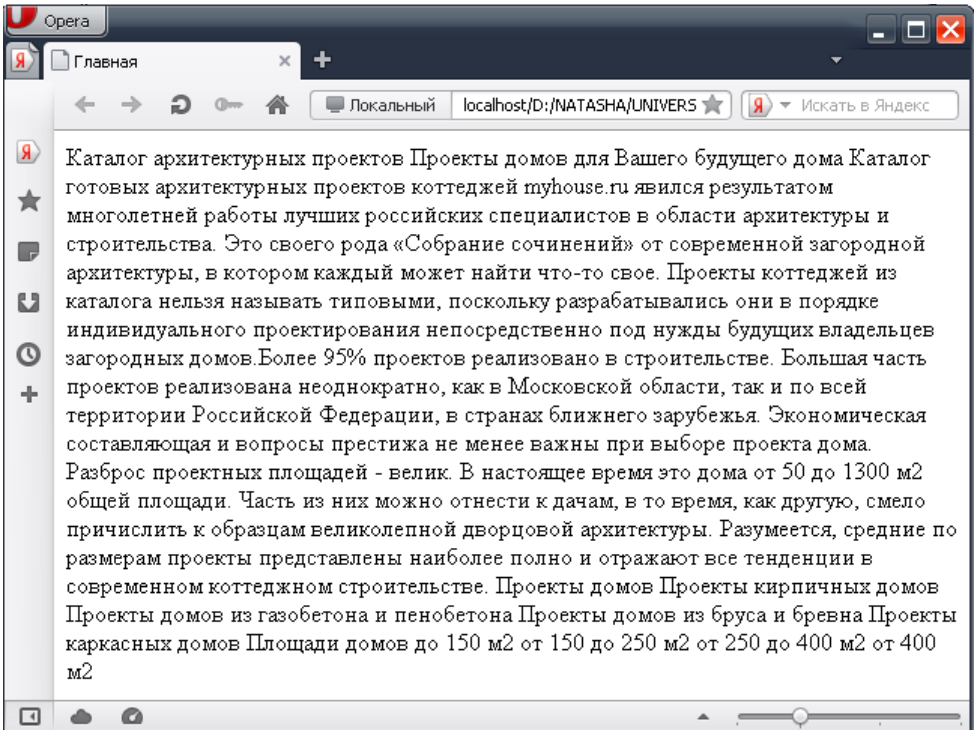


Рисунок 1.8

2. **Форматирование web-страницы тегамиHTML**

Теги предназначены для форматирования текста web-страницы. Список тегов более подробно можно посмотреть в папке **CD/Справочник HTML** в справочнике**html401_ru.chm**(в верхнем меню пункт **элементы**).

Рассмотрим некоторые из тегов.

Элементы h1,h2,h3,h4,h5,h6

Структурирование тела документа выполняется внутри элемента **<body>**с помощью заголовков, задаваемых элементамиh1,h2,h3,h4,h5,h6.

Элементы заголовков являются парными, поэтому должны имеет открывающий **<h1>** и закрывающий**</h1>**теги.

HTML располагает шестью уровнями заголовков:h1 (самый верхний),h2,h3,h4,h5 иh6 (самый нижний). Функции элементов заголовков подобны обычным стилям заголовков в текстовых редакторах.

Действие этих шести тегов представлено на рисунках ниже. На одном рисунке исходный код (рисунок 1.9), на другом – вид в браузере (рисунок 1.10).

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

<html>
    <head>
        <meta http-equiv=Content-Type content="text/html;
        charset=windows-1251">
        <title>Заголовки HTML</title>
    </head>
    <body>
        <H1>Заголовок первого уровня</H1>
        <H2>Заголовок второго уровня </H2>
        <H3>Заголовок третьего уровня </H3>
        <H4>Заголовок четвертого уровня </H4>
        <H5>Заголовок пятого уровня </H5>
        <H6>Заголовок шестого уровня </H6>
    </body>
</html>
```

Рисунок 1.9

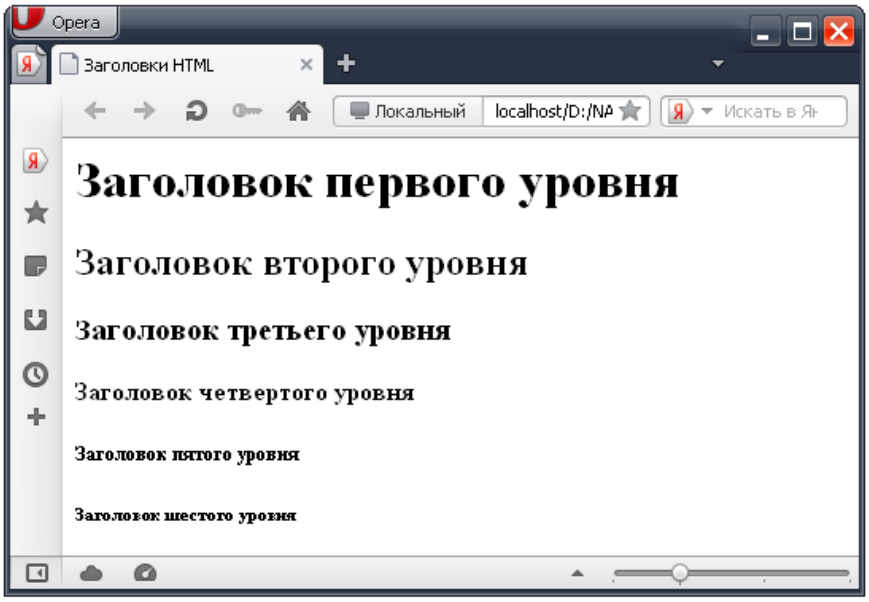


Рисунок 1.10

Разделение текста на абзацы

Тег **<p>**задает начало абзаца и вставляет сверху абзаца расстояние – отступ для отделения этого абзаца от предыдущего. Парный тег можно опустить.

Принудительный разрыв строки

Тег **
**позволяет выполнить перенос оставшейся части текста абзаца на следующую строку. Это непарный тег и в отличие от тега абзаца не увеличивает интервал между строками.

Практическое задание 2

1. Отформатируйте заголовок «Каталог архитектурных проектов» с помощью тегов <h1> и </h1>.

2. Отформатируйте заголовок «Проекты для Вашего будущего дома» с помощью тегов <h2> и </h2>.
3. Отформатируйте заголовки «Проекты домов» и «Площади домов» с помощью тегов <h3> и </h3>.
4. Основной текст разделите на абзацы с помощью тега <p>. Внутри первых двух абзацев используйте тег
 для переноса строки. В результате Ваш код должен выглядеть следующим образом (рисунок 1.11).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/htm

<html>
 <head>
  <meta http-equiv=Content-Type content="text/html; charset=windows-1251">
  <title>Главная</title>
 </head>
 <body>
<h1> Каталог архитектурных проектов</h1>

<h2>Проекты для Вашего будущего дома</h2>

<p>Каталог готовых архитектурных проектов коттеджей myhouse.ru явился резул
многолетней работы лучших российских специалистов в области архитектуры и с
<br>Это своего рода «Собрание сочинений» от современной загородной архитект
каждый может найти что-то свое.

<p>Проекты коттеджей из каталога нельзя называть типовыми, поскольку разраб
порядке индивидуального проектирования непосредственно под нужды будущих вл
загородных домов.Более 95% проектов реализовано в строительстве. Большая ча
реализована неоднократно, как в Московской области, так и по всей территори
Федерации, в странах ближнего зарубежья.

<p>Экономическая составляющая и вопросы престижа не менее важны при выборе
Разброс проектных площадей - велик. В настоящее время это дома от 50 до 130
площади. Часть из них можно отнести к дачам, в то время, как другую, смело
образцам великолепной дворцовой архитектуры. Разумеется, средние по размера
представлены наиболее полно и отражают все тенденции в современном коттеджн

<h3>Проекты домов</h3>
Проекты кирпичных домов
Проекты домов из газобетона и пенобетона
Проекты домов из бруса и бревна
Проекты каркасных домов

<h3>Площади домов</h3>
 до 150 м2
 от 150 до 250 м2
 от 250 до 400 м2
 от 400 м2
</body>
</html>
```

Рисунок 1.11

5. Просмотрите результат в браузере.

Маркированные и нумерованные списки

Средствами HTML можно создавать любые списки: нумерованные (цифровые и буквенные) и маркированные с разными типами маркеров.

Тег **
…
**формирует маркированный список.

Тег **
…
**формирует нумерованный список.

Отдельный элемент списка как в <ul>, так и в <ol> формируется с помощью тега**<li>**(непарный тег).

Практическое задание 3

1. Создайте нумерованный список под заголовком «Проекты домов».
2. Создайте маркированный список под заголовком «Площади домов». Ваш код будет выглядеть следующим образом (рисунок 1.12).

```
19  <h3>Проекты домов</h3>
20  <ol>
21      <li>Проекты кирпичных домов
22      <li>Проекты домов из газобетона и пенобетона
23      <li>Проекты домов из бруса и бревна
24      <li>Проекты каркасных домов
25  </ol>
26
27  <h3>Площади домов</h3>
28  <ul>
29      <li>до 150 м2
30      <li>от 150 до 250 м2
31      <li>от 250 до 400 м2
32      <li>от 400 м2
33  </ul>
```

Рисунок 1.12

3. Просмотрите страницу в браузере.

Списки можно вкладывать друг в друга, используя при этом различные маркеры.

Пример вложенного списка приведен на рис. 1.13

Практическое задание 4

1. Реализуйте в новом файле код, приведенный на рисунке 1.13.
2. Сохраните файл в папке **myhouse** под именем **spisok_vlozh.html.** Результат на рис. 1.13

```
<h3>Содержание</h3>

<ol>
<b><li>Графический редактор Photopshop</b>
        <ol>
        <li>Инструменты рисования
                <ul>
                <li>Кисть
                <li>Градиент
                </ul>
        <li>Создание и редактирование слоев
                <ul>
                <li>Создание и удаление слоя
                <li>Эффекты слоев
                </ul>
        </ol>


<b><li>Графический редактор CorelDRAW</b>
        <ol>
        <li>Операции с объектами
                <ul>
                <li>Копирование
                <li>Масштабирование
                </ul>
        <li>Специальные эффекты
                <ul>
                <li>Огибающие деформации
                <li>Перетекание, ореол, фигурная обрезка
                </ul>
        </ol>
</ol>
```

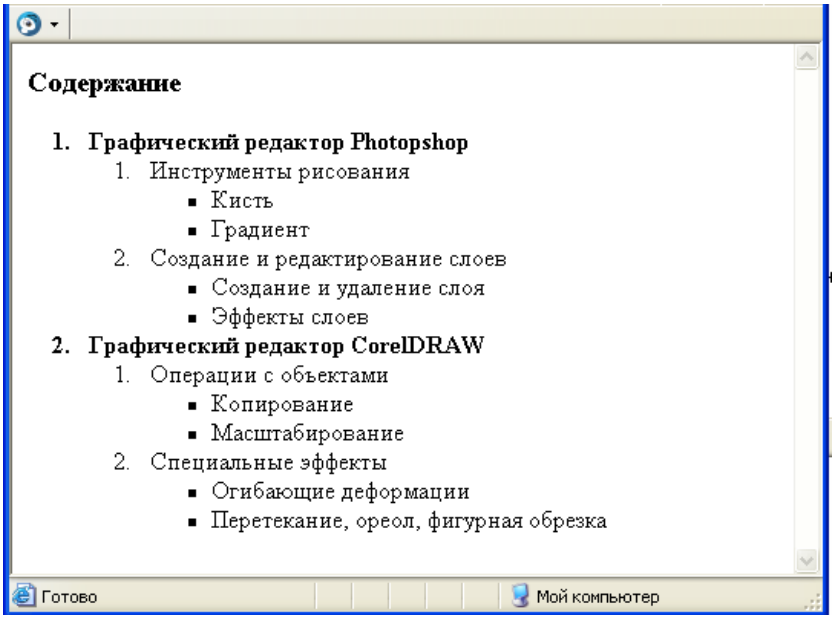


Рисунок 1.13. Пример вложенного списка

Начертания шрифтов

Тег **…**– позволяет отобразить текст полужирным шрифтом.

Тег **…**– позволяет отобразить текст в курсивном начертании.

Тег **<u>…</u>**– отображает подчеркнутый текст.

Например:

В этом случае текст будет отображен ***жирным курсивом***, но не подчеркнутым.

А в этом случае текст будет написан <u>***жирным подчеркнутым курсивом***</u>.

Нижние и верхние индексы

Тег **_{**и**}** позволяет опустить текст на полстроки ниже обычного текста.

Тег **^{**и**}** позволяет поднять текст на полстроки выше обычного текста. Текст, расположенный между этими тегами, будет изображаться меньшим шрифтом, по сравнению с обычным текстом.

Практическое задание 5

1. Отформатируйте в первом абзаце название myhouse.ru полужирным шрифтом (рисунок 1.14).

```
<b>myhouse.ru</b>
```

Рисунок 1.14

2. Отформатируйте во втором абзаце фразу «Более 95% проектов» подчеркнутым курсивом (рисунок 1.15).

```
<br><i><u>Более 95% проектов</u></i>
```

Рисунок 1.15

3. Отформатируйте верхние индексы в тех местах, где используются квадратные метры (рисунок 1.16).

```
<li>до 150 м<sup>2</sup>
<li>от 150 до 250 м<sup>2</sup>
<li>от 250 до 400 м<sup>2</sup>
<li>от 400 м<sup>2</sup>
```

Рисунок 1.16

4. Сохраните файл. Просмотрите через браузер. Web-страница должна выглядеть следующим образом (рисунок 1.17).

Каталог архитектурных проектов

Проекты для Вашего будущего дома

Каталог готовых архитектурных проектов коттеджей **myhouse.ru** явился результатом многолетней
работы лучших российских специалистов в области архитектуры и строительства.
Это своего рода «Собрание сочинений» от современной загородной архитектуры, в котором кажд
может найти что-то свое.

Проекты коттеджей из каталога нельзя называть типовыми, поскольку разрабатывались они в пор
индивидуального проектирования непосредственно под нужды будущих владельцев загородных д
Более 95% проектов реализовано в строительстве. Большая часть проектов реализована неоднокр
как в Московской области, так и по всей территории Российской Федерации, в странах ближнего
зарубежья.

Экономическая составляющая и вопросы престижа не менее важны при выборе проекта дома. Ра
проектных площадей - велик. В настоящее время это дома от 50 до 1300 м^2 общей площади. Часть
них можно отнести к дачам, в то время, как другую, смело причислить к образцам великолепной
дворцовой архитектуры. Разумеется, средние по размерам проекты представлены наиболее полно
отражают все тенденции в современном коттеджном строительстве.

Проекты домов

1. Проекты кирпичных домов
2. Проекты домов из газобетона и пенобетона
3. Проекты домов из бруса и бревна
4. Проекты каркасных домов

Площади домов

- до 150 м^2
- от 150 до 250 м^2
- от 250 до 400 м^2
- от 400 м^2

Рисунок 1.17

3. **Вставка изображений**

Для вставки изображений используется тег **<img>.**Обязательным для этого тега является атрибут **src**(от английского SouRCe – источник). Он определяет путь до графического файла, изображение которого должно быть выведено на web-странице.

Для вставки изображения используется команда **<img src=”Имя файла”>**

Например: **<img src=image1.jpg alt=”Картинка”>**

Атрибут alt необходим для того, чтобы при просмотре web-страницы в режиме отключенных изображений, вместо отсутствующей картинки была надпись, которая прописана в атрибуте alt. Также alt-тексты рекомендуется использовать всегда, т.к. поисковые машины анализируют их как ключевые слова при поиске изображений.

Графические файлы могут быть в формате **jpg, gif, png.**

Практическое задание 6

1. После списка площадей домов вставьте в web-страницу изображения проектов коттеджей**project_1.jpg**и**project_2.jpg** из папки **CD/ html_css_1**. Код вставки изображений будет выглядеть следующим образом (рисунок 1.18).

```
<p><img src="project_1.jpg" alt="Коттедж">
<img src="project_2.jpg" alt="Коттедж">
```

Рисунок 1.18

2. Просмотрите результат в браузере (рисунок 1.19).

Рисунок 1.19

В основном принцип использования тегов HTMLдолжен быть понятен. Они используются по одному и тому же принципу: если тег контейнерный, то есть открывающий и закрывающий тег. Если тег одиночный, то закрывающего тега нет. Является ли тег контейнерным или одиночным Вы всегда можете посмотреть в спецификации html401_ru.chm в разделе элементы.

4. **Адресация внутри сайта**

Рассмотрим варианты адресации, когда в html-файле надо разместить изображения, которые могут находиться в разных папках сайта.

Существует два вида адресации:

- абсолютная;
- относительная.

Абсолютная адресация(с использованием названий дисков компьютера)**не используется** (рисунок 1.20)

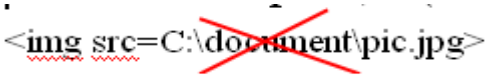


Рисунок 1.20

Используется ***относительная адресация***– адресация в пределах документа или совокупности документов на одном сервере. Чтобы сослаться на файл внутри сайта, нужно указать браузеру, какой путь он должен проделать, что бы прийти к нужному файлу.

2.13.3 Результаты и выводы:

2.14 Практическое занятие № 14 (2 часа).

Тема: «Синтаксис языка. Объектная модель JavaScript.»

2.14.1 Задание для работы:

1. Создать файловую структуру страницы.
2. Написать скрипт.
3. Подключить его к странице и настроить вывод сообщения

2.14.2 Краткое описание проводимого занятия:

Цель работы

Ознакомиться с базовым синтаксисом и основными возможностями управления содержимым веб-страницы на стороне клиента. Получить практические навыки написания клиентских скриптов с использованием языка JavaScript.

Задания к работе

Написать скрипт «Tip of the Day» (совет дня). Скрипт должен выводить случайную строку («совет») из заданного массива строк. Скрипт разместить во внешнем файле, подключить его на все страницы вашего сайта.

Методические указания

1. Динамический HTML
2. Синтаксис JavaScript

Требования к хостингу

Веб-сервер отдает веб-страницу со скриптами javascript «как есть» и все сценарии выполняются в браузере клиента. Это позволяет минимизировать затраты на размещение сайта: можно использовать дешевый хостинг, поскольку клиентские скрипты не создают нагрузки на сервер (кроме, разве что, небольшого увеличения трафика). А современные решения на основе javascript (AJAX, JSON и др.) позволяют организовать взаимодействие между клиентским скриптом и серверными приложениями.

Динамический HTML

HTML является языком разметки и не имеет каких-либо средств, которые могли бы использоваться для изменения содержимого страницы. Эту проблему решает использование языка DHTML (Dynamic HTML), поддерживающего средства программирования на клиентской стороне. Для этого в DHTML встроена поддержка скриптового языка JavaScript (должен поддерживаться браузером).

Возможности динамического управления содержимым становятся доступны при внедрении в веб-страницу кода JavaScript . Это делается с помощью тега script,

размещаемого в нужном месте веб-страницы и которым выделяют начало и конец исходного кода или указывают на подгружаемый из сети файл с исходным кодом:

Для внедрения скриптов в веб-документ используется контейнерный тег <script>...</script>, внутри которого записываются команды JavaScript (в общем случае и ряда других яыков: VBScript, php, tcl/tk ...).

Если этот тег ипользуется в теле документа (внутри тега body), то исполнение скрипта осуществляется по мере отображения веб-страницы в браузере. Если же контейнер `script` описан внутри тега `head`, то обращение к скрипту возможно только явным образом, например, через вызов функции.

```
<!-- внедрение скрипта в разметку -->
<script type="text/javascript">
      код скрипта
</script>
```

Имеется возможность вынести код JavaScript в отдельный файл (как правило с расширением .js), который затем подключить к документу следующим образом:

```
<html>
<head>
<!-- загрузка скрипта из внешнего файла -->
<script type="text/javascript">
src="http://example.com/scripts.js"></script>
</head>
...
```

Такой способ внедрения скриптов позволяет создавать своего рода библиотеки скриптов и использовать их на всех страницах сайта.

Синтаксис JavaScript

Язык JavaScript синаксически близок к языкам C/C++, Java, PHP и другим C-подобным языкам. Поэтому для тех, кто знаком с такими языками не составит труда разобраться с основными языковыми конструкциями.

Переменные
Для объявления переменных используется ключевое слова `var`. Переменные можно сразу инициализировать. Можно объявить несколько переменых сразу, разделив их запятыми.

```
var color = "#FFF", fsize = 1024 , total_count = 0, i;
var average = null;
var c = 3;
d = 0; //Ошибка!
```

Непроинициализированые переменные будут иметь неопределенное значение (undefined).

Объявлять переменные можно в любом месте скрипта, но до первого обращения

Типы данных переменным в javascript назначаются автоматически. Так же автоматически выполняется приведение типов.

Объявления массивов данных могут выполняться статически и динамически. Индексирование элементов начинается с нуля. Элементы массива могут быть проинициализированы при создании.

```
var weekdays = ["Пн", "Вт", "Ср", "Чт", "Пт"]; // статический
массив из пяти элементов


// динамическое объявление массива путем создание экземпляра
встроенного класса Array
var myarr;
myarr = new Array(256);


myarr[0] = 255;
myarr[1] = 254;


var x = myarr[7];
```

Операторы

Комментарии - предназначены для пояснения фрагментов кода или исключения фрагментов кода из обработки. Игнорируются при выполнении программы.

```
// Это однострочный комментарий.


/*
Это многострочный комментарий. Он может объединять несколько
строк и
его можно использовать в любом месте программы
*/
```

Условный оператор **if** предназначен для ветвления программы в зависимости от значения (true | false) логического выражения:

```
if (условие) {блок операторов1}
      [else {блок операторов2}]
```

Оператор выбора **switch** также как и условный оператор предназначен для выполнения ветвтления алгоритма, но позволяет анализировать множество возможных результатов проверки условия. Оператор break позволяет прервать выполнение оператора, если его не указать, то будут выполнены все последующие операторы.

```
switch (условие) {
        case значение1 : {блок операторов1; break;}
        case значение2 : {блок операторов2; break;}
        case значение3 : {блок операторов3; break;}
        ...
        [default : {блок операторов по умолчанию};]
}
```

Цикл со счетчиком **for**. Используется для циклов с заданным числом итераций (примечание: на самом деле конструкция for может использоваться и для построения любых циклов, все зависит от того, как и какие значения указаны в качестве параметров цикла).

```
for ([начальное значение]; [условие]; [приращение]) {блок операторов;}
```

Цикл с постусловием **do...while**. Выполняется, пока условие является истинным. Всегда выполняется хотя бы один раз.

```
do {блок операторов;} while (условие)
```

Цикл с предусловием **while**. Выполняется, если условие является истинным. Может не выполниться ни разу.

```
while (условие) {блок операторов;}
```

Операторы **break** и **continue** -используются для прерывания выполнения цикла или завершения текущей итерации.

Поэлементный цикл **for (... in ...)** применяется для выполнения команд над каждым элементом массива или коллекции.

```
for (переменная in массив|объект|коллекция) {блок операторов;}
```

Оператор объединения **with** представляет обращение к свойствам и методам объекта через общее имя.

```
with (объект) {блок операторов;}
```

Функции. Представляют возможность создания повторно используемого кода. Функция может принимать параметры и возвращать значение в вызывающую программу. Если в функции не предусмотрен возврат значения, то она работает как процедура.

```
function имяФункции ([список параметров]) {
       блок операторов;
       [return возвращаемоеЗначение;]
       }
```

Встроенные объекты JavaScript

JavaScript предлагает разработчику некоторый набор библиотечных функций, оформленных в виде свойств и методов различных объектов. Обращение к этим свойствами методам - через точечную нотацию.

Кратко рассмотрим некоторые встроенные объекты JavaScript.

Объект Math

Объект Math представляет математические константы и функции. Константы представлены свойствами объекта, а функции - его методами. Их назначение понятно из названий:

Свойства: E, LN2, LN10, LOG2E, LOG10E, PI, SQRT1_2, SQRT2.

Методы: abs, acos, asin, atan, ceil, cos, exp, floor, log, max, min, pow, random, round, sin, sqrt, tan.

Примеры использования:

```
var r = 1.8, theta = 30, a, x, y, D;
var rnd = Math.round(Math.random()*99)+1;


D = Math.PI*r*r;
x = Math.max(1,7,5,9);
y = Math.pow(2,10);


with (Math) {
       y = r*sin(theta);
       x = r*cos(theta);
}
```

Объект string

Встроенный объект string представляет литерал (строку символов), заключенный в одинарные или двойные кавычки или вычислимое выражение, которое может быть интерпретировано как строка.

Для объекта string определены следующие свойства и методы:

- Свойства: length (длина строки).
- Методы (не все): anchor (якорь), bold (полужирное начертание), charAt (символ в позиции), fixed (преформат), fontcolor (цвет шрифта), fontsize (размер шрифта), indexOf (индекс первого вхождения символа), italics (курсив), link (гиперссылка), substring (подстрока), toLowerCase (строчные), toUpperCase (прописные).

Несколько примеров использования объекта string:

```
var hello = "Hello, ", w = "World!";
var str = hello + w; // конкатенация строк


document.write(str.bold());
document.write(str.toUpperCase());
document.write(hello.fontsize(6));
document.write(hello.substring(0,3));


document.write(hello.link("http://localhost"));


document.write(w.indexOf("l"));


alert("string lehgth = " + str.length);
```

Вывод данных в JavaScript

Результаты работы скрипта JavaScript могут быть отображены по меньшей мере двумя способами: в окно текущего веб-документа и в диалоговое окно.

Для вывода данных в веб-документ можно использовать метод `write` объекта `document` (подробней в ЛР №5). Примеры использования этого метода приведены при описании объекта `string`. Еще несколько примеров:

```
document.write("Hello, World!");
document.write("<h1>>Hello, World!</h1>"); // внедрение HTML в
JavaScript


// обратите внимание на использование вложенных кавычек
document.write("<p><a href='http://localhost'>Link to
localhost</a></p>");
```

Для вывода различных информационных сообщений, не относящихся напрямую к содержимому веб-страницы следует использовать метод `alert`, представляемый объектом `window`. Этот метод выводит модальное диалоговое окно (рис.1),

блокирующее выполнение скрипта до нажатия пользователем кнопки в этом диалоге.

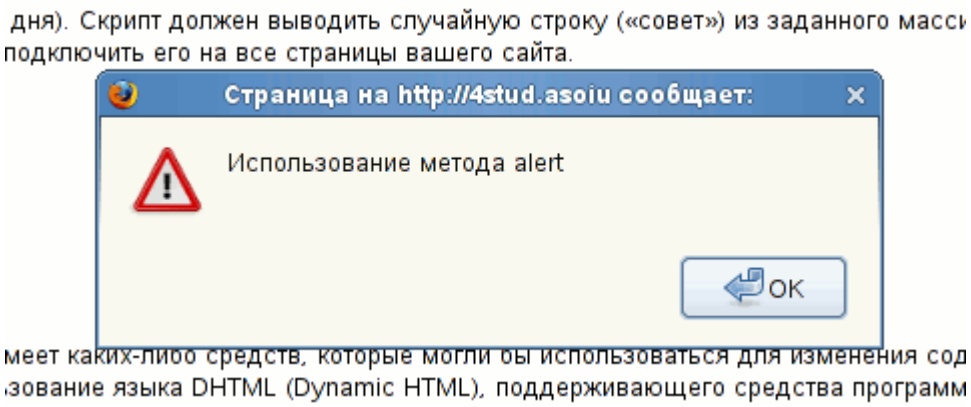


Рис. 1. Вызов окна сообщения из скрипта JavaScript

Пример использования метода `alert`:

```
var a, b, s = "=";

with (Math) {
       a = ceil(random()*100);
       b = ceil(random()*100);
}
a > b ? s = ">" : s = "<"; // тернарное сравнение

alert("A = "+a+";\nB = "+b+";\nСледовательно, A "+s+" B"); //
"\n" - перевод строки
```

2.14.3 Результаты и выводы:

В результате выполнения данного практического занятия мы научились работать со сложными Javascript шрифтами и написали скрипт с выводом предупредительного сообщения.

2.15 Практическое занятие № 15 (2 часа).

Тема: «Типы данных. Операторы JavaScript.»

2.15.1 Задание для работы:

1. Создать файловую структуру страницы.
2. Создать макет будущего приложения.
3. Подключить скрипты автозаполнения.

2.15.2 Краткое описание проводимого занятия:

1.1. ПРОЕКТИРОВАНИЕ СТРАНИЦЫ АНКЕТЫ. Создайте пустую веб-страницу. Настройте цвет фона, шрифты и другие свойства страницы, аналогичные ранее размеченным страницам сайта. Добавьте на страницу объект «форма». Поместите в форму группу кнопок выбора (radiobutton), комбинированный список с выпадающими ответами, несколько строк ввода (обязательны поля ввода адреса e-mail и номера телефона), список с возможностью выбора нескольких вариантов ответа и область ввода текста. Дайте форме и ее объектам понятные («говорящие») имена. Не следует называть форму “f”, радиокнопки “r”, “rr” или “rrr”, а область ввода текста “a”. Оформите страницу в

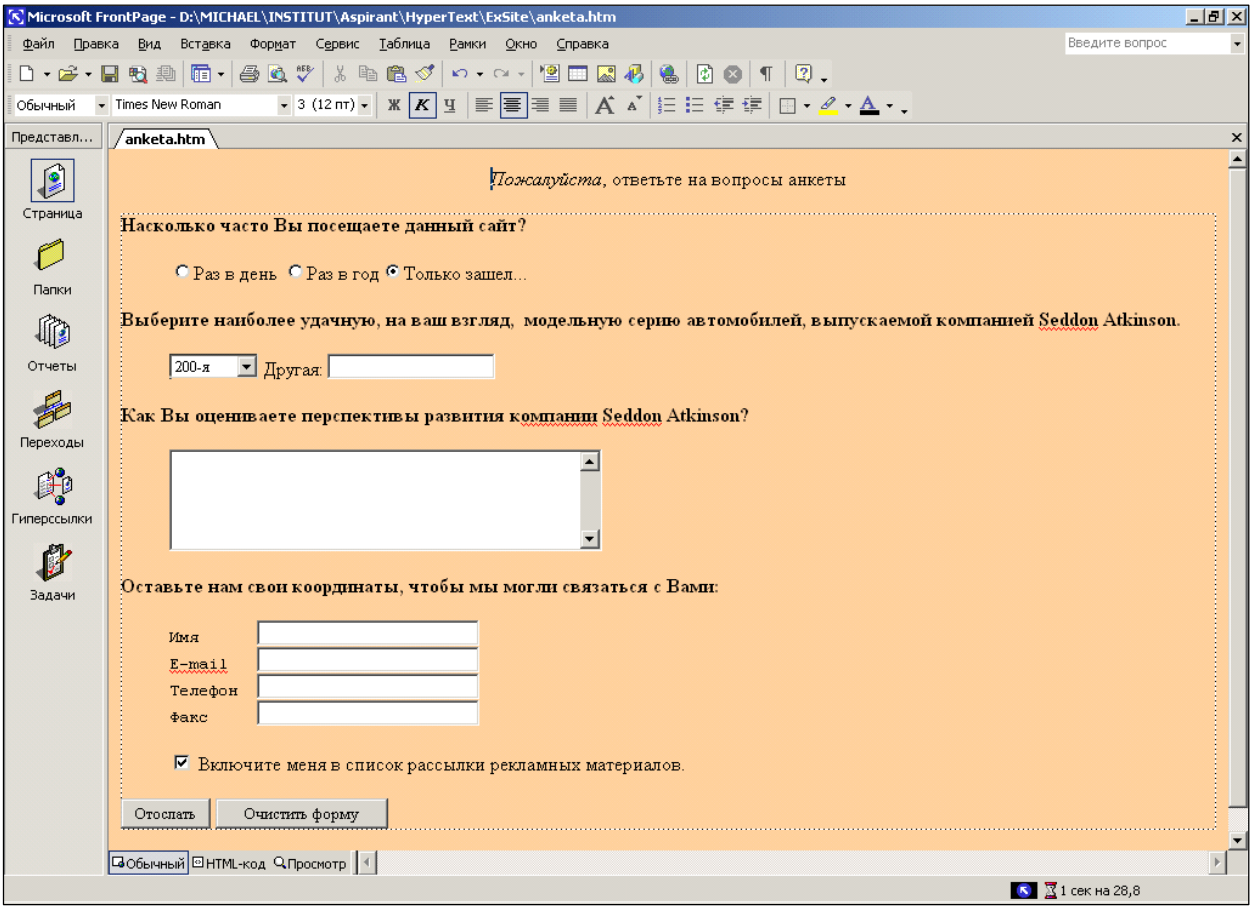


Рис. 3.2. Форма анкеты посетителя сайта

соответствии с правилами дизайна. Пример анкеты показан на рис. 3.2.

Добавьте ссылку на анкету в меню сайта.

1.2. СОЗДАНИЕ СКРИПТОВ ДЛЯ ПРОВЕРКИ ПРАВИЛЬНОСТИ ЗАПОЛНЕНИЯ ПОЛЕЙ ФОРМЫ.

1.2.1. Самый надежный способ ненавязчивого контроля правильного ввода пользователем необходимой вам информации – создать обработчик события onBlur (элемент формы из активного превращается в неактивный). Тогда проверка производится после заполнения поля, в момент перехода к следующему. Можно использовать обработчик onChange, реагирующий на каждый вводимый символ, но это лишит вашего потенциального клиента права на опечатку или на создание экзотического псевдонима. Постоянно всплывающие в процессе ввода информации сообщения могут заставить даже лояльного пользователя отказаться от сотрудничества с вами. Более того, часто такой

способ просто неприемлем (например, для проверки правильности ввода адреса e-mail). Дополнительно организуйте функцию общей повторной проверки всех полей формы по событию submit (отправка формы). Опыт показывает, что «изобретательные» корреспонденты легко находят способ обойти событие onBlur.

1.2.2. Для поля ввода имени (см. рис. 3.2) разрешенными символами являются буквы и дефис. Если желательно допустить ввод фамилии, инициалов или имени и отчества, то следует разрешить пробелы и точки. Существуют два варианта скрипта проверки правильности ввода имени: один проверяет наличие разрешенных символов, другой – наличие запрещенных. В обоих случаях в функции проверки задается строка запрещенных или разрешенных для использования в имени символов, а затем в цикле проверяются значения символов из поля ввода имени. С помощью функции indexOf или search проверяется присутствие или отсутствие текущего символа в запрещающей или разрешающей строке. По факту обнаружения несоответствия символа в поле ввода имени указанным ограничениям на экран с помощью функции alert выдается соответствующее сообщение («Обнаружен неверный символ»), курсор помещается в поле ввода имени (метод focus), а вся строка выделяется (метод select), чтобы дать возможность пользователю удалить ее одним нажатием клавиши.

1.2.3. Функции проверки ввода номера телефона или факса принципиально не отличаются от проверки ввода имени. Единственным отличием является то, что при таком виде проверки проще задать строку разрешающих символов, так как их будет заведомо меньше, чем запрещенных. В номерах телефона и факса разрешено указывать цифры, а также скобки, дефис и плюс. Действия системы по факту обнаружения нарушений аналогичны перечисленным в пункте 4.2.2.

1.2.4. Содержимое комбинированного списка для выбора альтернативного варианта ответа в контроле не нуждается, оно введено автором на этапе дизайна. Нужно проверить, введено ли что-нибудь в строку альтернативного варианта ответа (рис. 3.3). Правила дружественного интерфейса советуют при выборе варианта «Другая модель» автоматически – с помощью обработки события onChange элемента ComboBox – переводить фокус ввода на строку альтернативного варианта. В этом поле ввода допустимые комбинации символов должны принадлежать множеству всех известных моделей фирмы.

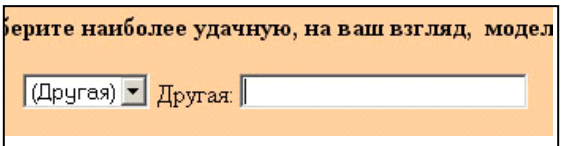


Рис. 3.3. Пример контроля заполнения поля (строки) ввода

1.2.5. Функция проверки ввода адреса e-mail может быть реализована несколькими способами. Самый простой из них – использовать функцию search. Более надежный и трудоемкий – последовательно перемещаться по строке, аналогично процедуре проверки имени, выставляя флажки-признаки наличия встреченных символов, проверяя их повторение и т.д. В обозначении e-mail адреса необходимо удостовериться в использовании только разрешенных символов. Это буквы, цифры, символы «_» и «-». Символ «@» должен присутствовать только один раз. Устанавливается наличие символа «.» (точка), отсутствие точки непосредственно после или перед символом «@», отсутствие лидирующей и замыкающей точки в обозначении адреса, отсутствие следующих друг за другом точек (многоточия), а также возможность указания доменов третьего и более высоких уровней (комбинации вида ghf56@hj.dhh.aaa.des). Действия по факту обнаружения нарушений аналогичны указанным выше (см. рис. 3.4). Отличие состоит в том, что сообщения об ошибках должны четко указывать, что именно некорректно в строке, например, «В адресе e-mail встречено два символа @».

1.2.6. Поле ввода текста может быть заполнено произвольным набором символов. Следует проверить, не осталось ли оно пустым.

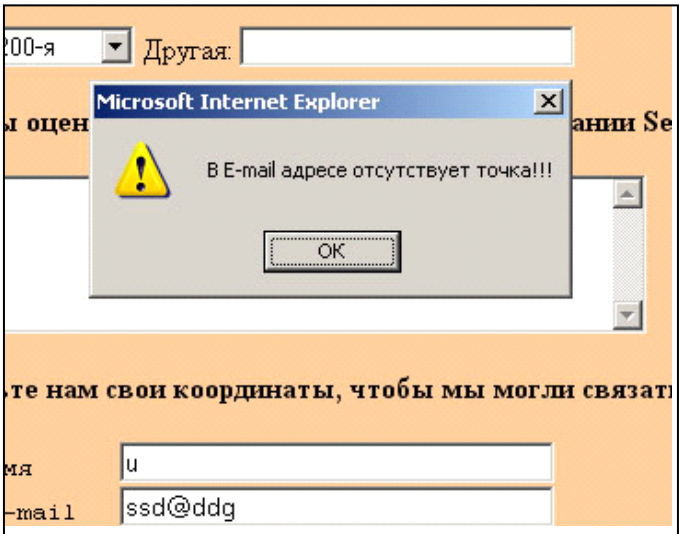


Рис. 3.4. Контроль заполнения

1.3. СТРАНИЦА ГОСТЕВОЙ КНИГИ. Создайте пустую страницу. Настройте ее свойства по аналогии с имеющимся страницами сайта и разместите на ней объект «форма» с полями ввода (рис. 3.5).

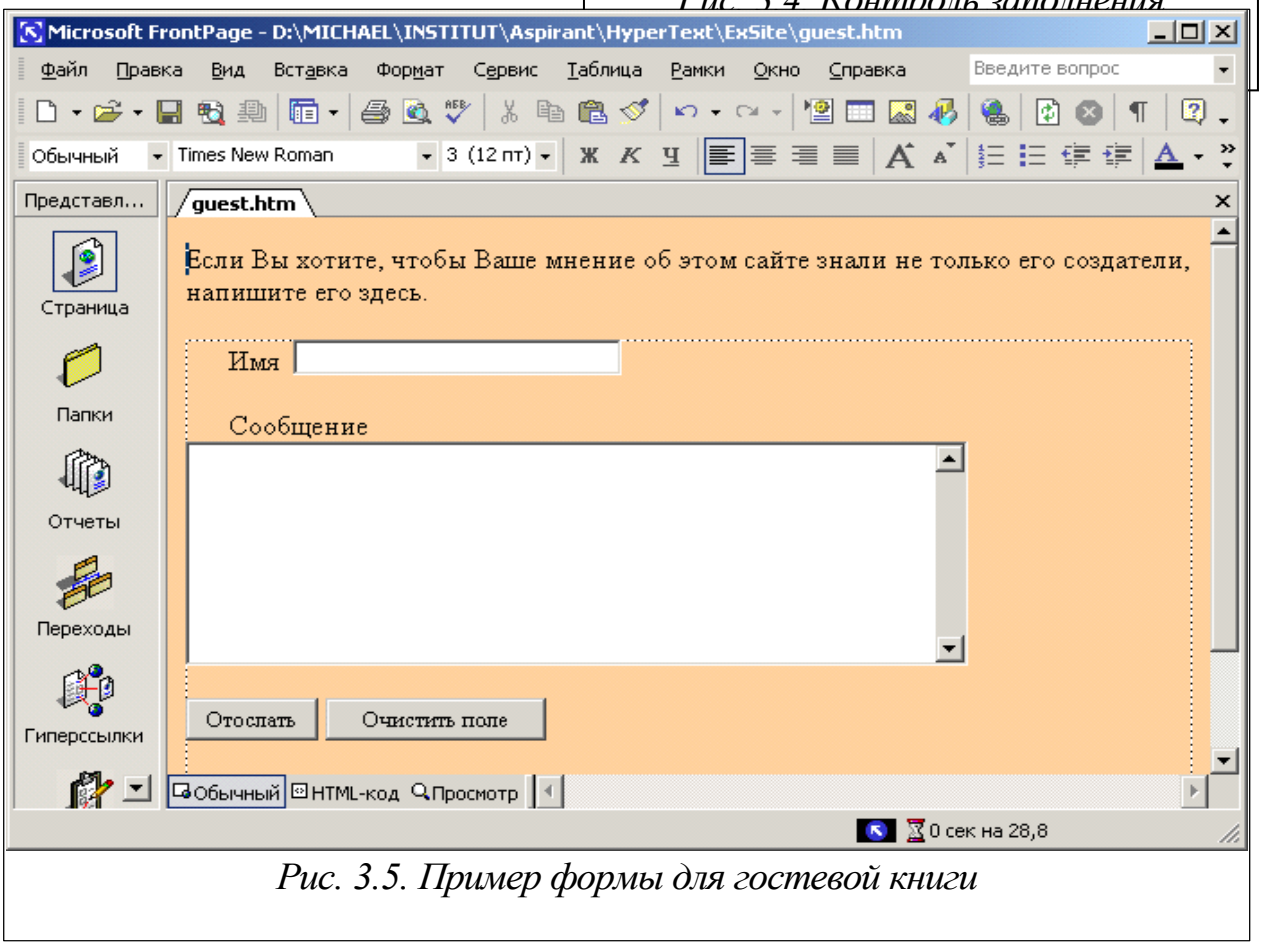


Рис. 3.5. Пример формы для гостевой книги

2.15.3 Результаты и выводы:

В результате выполнения данного практического занятия мы научились создавать обработчики событий и сложные скрипты для Javascript.

2.16 Практическое занятие № 16 (2 часа).

Тема: «Типы данных. Операторы JavaScript»

2.16.1 Задание для работы:

1. Создать файловую структуру страницы.

2. Написать скрипт для изменения цвета при нажатии.

3. Написать скрипт для изменения цвета при наведени.

2.16.2 Краткое описание проводимого занятия:

Язык программирования JavaScript был разработан Бренданом Эйком (Brendan Eich) в Netscape Communications как язык сценариев для обозревателей Netscape Navigator, начиная с версии 2.0. В дальнейшем к развитию этого языка подключилась корпорация Microsoft, чьи обозреватели Internet Explorer поддерживают JavaScript, начиная с версии 3.0. Версия Microsoft получила название JScipt, поскольку JavaScript является зарегистрированной маркой фирмы Netscape.

JavaScript - это язык программирования, язык сценариев (скриптов), предназначенный прежде всего для создания интерактивных HTML-страниц. Программу на языке JavaScript либо встраивают прямо в HTML-файл (как в секции <**head**>, так и в секции <**body**>) с помощью тега **<script> ... </script>** и располагают код JavaScript внутри этих тегов,

<script type="text/javascript"> </script>

или помещают весь код JavaScript в отдельный файл с расширением js и связываются с ним с помощью тега Script:

<script type="text/javascript" src="scripts/JavaScriptFile.js"> </script>

Чтобы программа была выполнена, HTML-файл должен быть открыт в браузере пользователя. Так как JavaScript является в настоящее время единственным языком сценариев, который поддерживают все основные браузеры Web (Internet Explorer, Firefox, Netscape, Safari, Opera, Camino и т.д.), то он используется очень широко. Однако следует помнить, что некоторые скрипты действуют по-разному в разных браузерах (то что работает в Internet Explorer может не работать в Firefox)

JavaScript - *интерпретируемый* язык. Это означает, что для исполнения программы не требуется предварительная *компиляция* (преобразование исходного текста программы в машинный код). Текст программы *интерпретируется*, то есть анализируется и сразу же исполняется.

JavaScript - это объектно-ориентированный язык программирования (ООП), основан не на обработке команд кода, а на присвоении отдельным элементам программы конкретных событий и выполнении их, если данное событие имело место. Например, событие нажатие на кнопку приводит к изменению содержимого текстового поля.

Основными понятиями любого объектно-ориентированного языка являются объекты, классы, методы и свойства. Разберем основные понятия на конкретных примерах:

```
<script type="text/javascript">
document.write("Введите свое имя и нажмите кнопку")
</script>
```

В результате при просмотре данной страницы в браузере появится текст: "Введите свое имя и нажмите кнопку".

Основные понятия:

OBJECT *(объект)* - это то, с чем производится действие, событие. Это может быть документ, открываемый в окне браузера или само окно браузера, или какая-то часть документа, теги. Объект должен иметь уникальное имя (ID), чтобы к нему можно было обратиться.

В нашем случае объектом является документ HTML и к нему можно просто обратиться по имени: **document.**

Каждый объект обладает своими методами.

METHOD *(метод объекта)* - это действия, которые можно выполнять над объектом такого типа, или которые сам объект может выполнять.

Синтаксис кода: между именем объект и методом обязательно ставят **разделительный оператор точка**, после метода в скобках параметры метода.

```
Объект . Метод ("параметры метода")
```

Параметры метода относятся к типу данных - **строки символов**. Строки символов нужно обязательно взять в кавычки либо в одинарные, либо в двойные.

Каждый объект обладает своими свойствами.

PROPERTY (свойство) - каждый объект имеет свои свойства. Один и тот же объект может обладать многими свойствами. Часто эти свойства необходимо изменить, при возникновении некоторого события.

Для изменения свойства объекта необходимо соблюдать следующий синтаксис:

```
Объект . свойство объекта = "новое значение свойства "
```

Например, для изменения фонового цвета документа HTML (имя данного свойства **bgColor**) следует написать следующее:

```
<script type="text/javascript">
document.bgColor='red'
</script>
```

И при просмотре в окне браузера фоновый цвет HTML документа будет красным.

Обратите внимание на то, что значение свойства **red** пишется в кавычках (одинарных или двойных), т.к. значение свойства относится к типу данных строки символов.

Задание №1

1) Создайте файл, соответствующий стандарту XHTML 1.1 или XHTML 1.0 Strict
2) Напишите скрипт, задающий свойство документа фон (цвета можно выбрать здесь)

Разумеется, нас будет интересовать возможность изменения свойства при возникновении какого-либо события.

EVENT (событие) - это все, что случилось: открытие окна, загрузка в него документа, клик клавишей мышки или просто перемещение курсора по экрану, нажатие клавиши на клавиатуре - это все события, и они могут инициировать запуск больших и маленьких программ.

Стандартные события в HTML

имя события	происходит
onclick	при щелчке кнопки мыши на элементе
ondblclick	при двойном щелчке кнопки мыши на элементе
onmousedown	при нажатии кнопки мыши на элементе
onmouseup	при отпускании кнопки мыши на элементе
onmouseover	при попадании курсора мыши на элемент
onmousemove	при движении курсора мыши по элементу
onmouseout	при попадании курсора мыши за пределы элемента
onkeypress	при нажатии и отпускании клавиши на элементе
onkeydown	при нажатии клавиши на элементе
onkeyup	при отпускании клавиши на элементе

Здесь следует пояснить, что **события** (event) и **обработчики событий** (event handler) относятся к JavaScript, но они скорее «встроены» в HTML-код. Они входят в структуру документа HTML и не требуют тегов <script> и </script>. Среди разнообразных обработчиков событий для начала мы выберем один, самый популярный, — **onmouseover** (навести мышь).

Код выглядит следующим образом:

** Наведи мышь на этот текст .... **

Как уже говорилось для написания этого кода не требуются теги <script> </script>. Событие встраивается в HTML код, т.е является описанием, атрибутом тега (в данном случае тега** **) при выполнении данного события - наведении мышкой на текст данного абзаца - изменяется свойство объекта - фон документа HTML.

И здесь есть еще одна важная особенность: **document.bgColor ='red'** нужно также записать в кавычках - одинарных или двойных. Вы можете использовать любой тип кавычек. Однако если Вы вынуждены как в данном случае ставить кавычки дважды, то можно использовать только вложенные кавычки. Не имеет значения, в каком порядке Вы использовали кавычки - сначала двойные, а затем одинарные или наоборот.

МОЖНО:

onmouseover="document.bgColor ='red' " или
onmouseover='document.bgColor ="red" '

Но НЕЛЬЗЯ:

onmouseover="document.bgColor ="red" "
onmouseover='document.bgColor ='red' '
onmouseover="document.bgColor ='red" '

Задание №2

1) Создайте файл, соответствующий стандарту XHTML 1.1 или XHTML 1.0 Strict
2) Напишите скрипт, изменяющий свойство документа - фон при наведении курсором на какой-то текст (цвета можно выбрать здесь)

А если мы хотим изменить не свойство всего документа, а только свойство какого-то абзаца? Как в данном случае мы можем изменить свойство данного объекта? Есть несколько способов.

Код данного примера (ВАЖНО! Код должен быть записан в одну строчку)

** Этот абзац меняет цвет при наведении на него мышкой с синего на красный! **

Разберем код.

1. **style="color:blue"** определяется стиль текста в данном абзаце
2. **onmouseover=** событие которое может произойти с этим абзацем, в кавычках надо указать что при этом делать
3. **this.style.color='red'** изменить стиль абзаца: цвет текста на красный:
    - слово **this** используется для доступа к элементу (объекту), вызвавшему событие (к данному абзацу),
    - через точку указывается его свойство **style**, дающее доступ к стилям,
    - еще через точку указывается **конкретное свойство**, значение которого мы хотим изменить (**color** - цвет текста)
    - затем идет знак присвоить **=**,
    - и затем значение свойства "цвет текста" - красный (**'red'**).

Рассмотрим еще один пример. Изменение цвета фона текста:

** Цвет фона текста меняется на красный при наведении мышкой на него! **

Разберем код.

1. **style="background-color:blue"** определяется стиль текста в данном абзаце (цвет фона)

2. **onmouseover=** (навести мышь) и **onmouseout=** (увести мышь) события которые могут произойти с этим абзацем, в кавычках надо указать что при этом делать
3. **this.style.backgroundColor='red''** изменить стиль абзаца: цвет фона на красный.

Задание №3

1) Создайте файл, соответствующий стандарту XHTML 1.1 или XHTML 1.0 Strict
2) Создайте три абзаца
3) Напишите скрипт, изменяющий цвет текста первого абзаца при наведении мышкой
4) Напишите скрипт, изменяющий цвет фона текста второго абзаца при наведении мышкой
5) Напишите скрипт, изменяющий цвет фона текста третьего абзаца **только** при наведении мышкой

2.16.3 Результаты и выводы:

В результате выполнения данного практического занятия мы научились работать с Javascript сценариями и обработкой событий, а также смогли применить их на реальных скриптах.

2.17 Практическое занятие № 17 (2 часа).

Тема: «Обработка событий»

2.17.1 Задание для работы:

1. Создать файловую структуру Web-сайта.
2. Создать различные скрипты для проверки вводимого текста.
3. Соединить скрипты между собой.

2.17.2 Краткое описание проводимого занятия:

Цель работы

Закрепить и расширить практические знания по программированию на языке javascript. Получить представление об практическом использовании объектной модели веб-документа (DOM) и использовании веб-форм.

CAPTCHA (от англ. «Completely Automated Public Turing test to tell Computers and Humans Apart», рус., сленг. - КАПЧА) — полностью автоматизированный публичный тест Тьюринга для различия компьютеров и людей) компьютерный тест, используемый для того, чтобы определить, кем является пользователь системы: человеком или компьютером. Основная идея теста: предложить пользователю такую задачу, которую может решить человек, но которую несоизмеримо сложно предоставить для решения компьютеру. В основном это задачи на распознавание символов. CAPTCHA чаще всего используется при необходимости предотвратить использование интернет-сервисов

ботами, в частности, для предотвращения автоматических отправок сообщений, регистрации, скачивания файлов, массовых рассылок и т. п.

Задание

Написать скрипт, проверяющий код защиты от автоматического постинга и вырезающий ссылки из формы ввода комментария (на странице отзывов и комментариев)

Пояснение: В ходе выполнения задания требуется написать клиентсткую программу на javascript, которая генерирует арифметический пример, ответ на который должен дать пользователь. Другой вариант — генерация произвольной строки, которую должен воспроизвести пользователь. После того, как пользователь ввел ответ программа должна проверить его правильность. Смысл этого задания не столько в разработке эффективного теста Тьюринга, сколько в освоении javascript.

Указания к работе

- Элементы управления
- Объект document
- События
- Объект RegExp
- Примеры скриптов

Элементы управления

Элементы управления — это интерактивные объекты, позволяющие получить данные от пользователя. Их назначение и внешний вид идентичны элементам пользовательского интерфейса современных операционных систем с графическим интерфейсом (кнопки, поля ввода, чекбоксы и т.п.).

Элемент input

Тэг <input> представляет различные элементы, в зависимости от значения атрибута type (табл.1).

Таблица 1. Типы элементов управления (атрибут type)

Значение type	**Описание**
text	Однострочное поле ввода. Используйте атрибуты maxlength и size для определения максимальной длинны вводимого значения в символах и размера отображаемого поля ввода на экране (по умолчанию принимается 20 символов).
password	То же самое, что и атрибут text, но вводимое пользователем значение скрыто замещающими символами (звездочки, точки и т.п.).
checkbox	Флажок (маркер множественного выбора). Используется для отметки выбранных вариантов.
hidden	Скрытое поле. Не отображается браузером и не дает пользователю изменять присвоенные данному полю значение. Это можно сделать только программным путем (или изменением значения поля при передаче данных

Значение type	Описание
	через адресную строку или в теле запроса).
image	Кнопка-картинка. Позволяет использовать графический рисунок в качестве кнопки. Все значения атрибута value игнорируются. Само описание картинки осуществляется через атрибут src и по синтаксису совпадает с тегом <img>.
radio	Радиокнопка. Позволяет вводить одно значение из нескольких альтернатив. Для создания набора альтернатив вам необходимо создать несколько полей ввода с атрибутом type="radio" с разными значениями атрибута value, но с одинаковыми значениями атрибута name. При выборе одного из полей ввода типа radio все остальные поля данного типа с тем же именем (атрибут name) автоматически станут невыбранными на экране.
button	Пользовательская кнопка. Должна быть запрограммирована на обработку нажатий. Атрибут value содержит текст надписи на кнопке.
submit	Кнопка отправки данных. При ее нажатии будет будет вызван обработчик, описанный в заголовке формы (form action="scriptname", подробнее о теге form - в лабораторной работе №8) и ему будут переданы значения всех элементов, описанных в теге form. Атрибут value содержит текст надписи на кнопке.
reset	Кнопка сброса. При нажатии ее все поля формы примут значения, заданные по умолчанию.

Атрибуты элемента input

- type — определяет тип поля ввода. По умолчанию равно text.
- name — имя поля ввода. Используется как идентификатор переменной при передаче данных на сервер и для программного обращения к элементу из скрипта javascript.
- id — идентификатор элемента. Должен быть уникальным в пределах веб-документа.
- checked — означает, что checkbox или radio будет выбран.
- maxlength — определяет количество символов, которое пользователи могут ввести в поле ввода. При превышении количества допустимых символов броузер реагирует на попытку ввода нового символа звуковым сигналом и не дает его ввести.
- size — определяет визуальный размер поля ввода на экране в символах.
- src — URL, указывающий на картинку (используется совместно со значением type="image").
- value — значение по умолчанию или установленное значение.

Элемент textarea

Тэг <textarea> используется для того, чтобы позволить пользователю вводить более одной строки информации (многострочный текст). При передаче значения из textarea сохраняются все символы форматирования (табуляция, перевод строки, возврат каретки).

Атрибуты, используемые с тегом <textarea> задают его размеры (в символах и строках):

- rows — высота поля ввода в символах
- cols — ширина поля ввода в строках

Пример использования тега <textarea>:

<textarea rows=10 cols=50>Москва, Дмитровское шоссе, д.9Б, офис 448</textarea>

Элемент select

Элемент select отображает на странице список выбора, который может быть представлен следующими способами:

- select — выпадающий список.
- select single — развернутый список.
- select multiple — список с множественным выбором.

Примеры описания элемента select:

<select name="group">

<option>понедельник, среда, пятница</option>

<option> вторник, четверг, суббота</option>

<option> воскресенье</option>

</select>

<select single name="group" size="3">

<option>зима</option>

<option>весна</option>

<option>лето</option>

<option>осень</option>

</select>

Объект document

Объект document это абстрактная структура данных, представляющая полное описание веб-страницы. Набор свойств и методов этого объекта позволяет управлять как поведением веб-страницы целиком, так и отдельных ее объектов (элементов управления, ссылок, текстовых блоков, изображений и т.д.). Доступ к свойствам и методам реализован через стандартные программные интерфейсы (рис. 1).

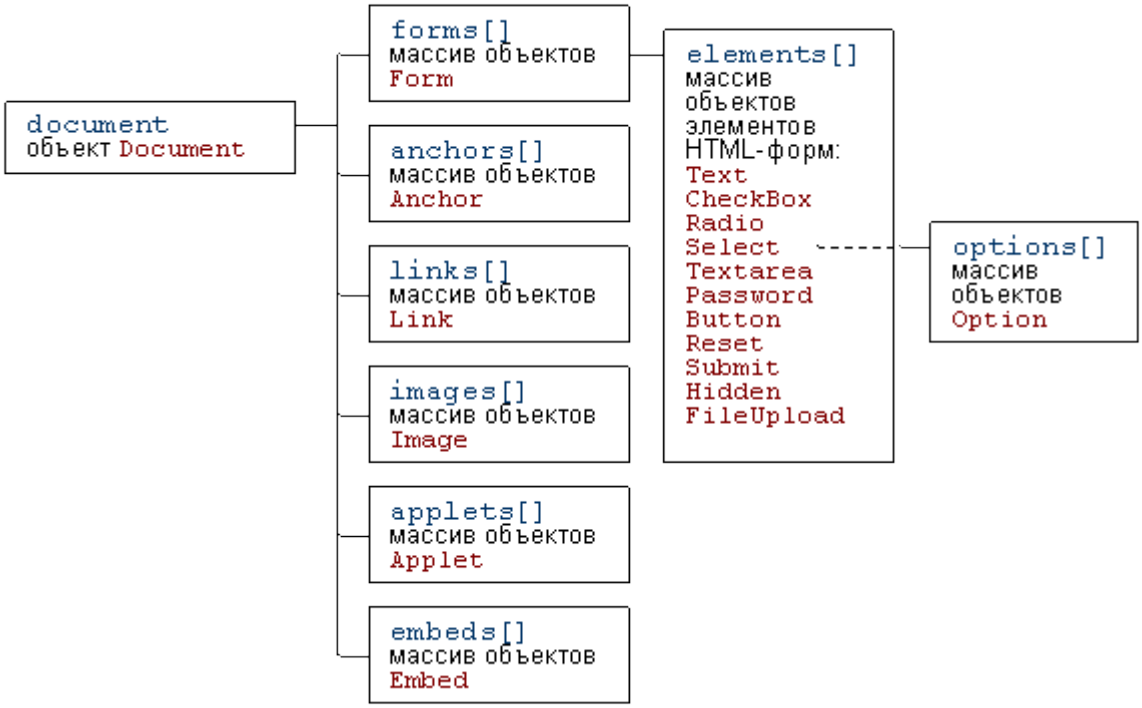


Рис. 1. Интерфейсы объекта document

Свойства объекта Document

Начнем со свойств, общих для всех браузеров. Большинство их доступны как для чтения, так и для изменения. Все значения свойств — строковые.

- title — текст заголовка документа (содержимое элемента title);
- fgColor и bgColor — цвет текста и цвет фона документа;
- linkColor, vLinkColor, aLinkColor — цвета непосещенных, посещенных и активных гиперссылок;
- lastModified (только для чтения) — дата изменения документа;
- referrer (только для чтения) — адрес источника перехода;
- URL, location — собственный адрес документа.

Более интересны и полезны для разработчика свойства-массивы объекта Document. Все они, естественно, имеют свойство length (количество элементов в массиве). Большинство свойств, специфичных для объектов, хранящихся в этих массивах, ассоциируются с атрибутами соответствующих элементов HTML (список неполный):

- объект Anchor (якорь) имеет единственное свойство name;
- объект Link (ссылка) имеет свойства href, target;
- объект Image (изображение) имеет свойства src, width, height.

К объектам документа, хранящимся в массивах images, controls и прочим, а также к элементам форм можно обращаться по имени (свойство name) или идентификатору (свойство id). Пусть, например, в документе имеется описание <**img src**="images/cat.jpg" **id**="cat_id" **name**="cat_name"> и оно является n-ым

изображением, встречающимся в документе. К этому элементу img можно обратиться по крайней мере следующими способами:

1. Как к элементу массива images по индексу (индексация начинается с 0):window.document.images[n-1]
2. Как к элементу хэш-массива images по ключу (значение name как ключ массива):window.document.images["cat_name"]
3. Используя значение атрибута name как свойство объекта: window.document.cat_name
4. Используя значение атрибута id и свойство getElementById:window.document.getElementById("cat_id")

Методы объекта Document

- open() — открывает новый документ; при этом все его содержимое удаляется.
- close() — закрывает ранее открытый документ.
- write() — записывает в документ заданную в качестве аргумента строку.
- writeln() — аналогичен предыдущему, но выведенная в документ строка заканчивается символом перевода строки.

Методы write() и writeln() весьма полезны и часто используются для динамического формирования содержимого документа. Вот как, например, можно включить в документ дату его последнего изменения:

<script>document.write(document.lastModified);</script>

События

Для всех элементов документа имеется возможность отслеживать различные события (загрузка, перемещение мыши, мышеклики и проч.) и вызывать функции обработки таких событий. В таблице 2 приведено краткое описание событий, доступных для использования в программах на javascript:

Таблица 2. События веб-документа

Событие	Описание
OnLoad	Броузер заканчивает открытие документа HTML
OnUnload	Броузер выгружает документ HTML
OnClick	Пользователь щелкнул мышью по элементу
OnDblClick	Пользователь дважды щелкнул мышью по элементу
OnMouseDown	Пользователь нажимает кнопку мыши
OnMouseOver	Пользователь перемещает мышь поверх элемента
OnMouseMove	Пользователь перемещает мышь поверх элемента
OnMouseOut	Пользователь перемещает мышь, выходя из элемента
OnFocus	Элемент получает фокус ввода
OnBlur	Элемент теряет фокус ввода
OnKeyPress	Пользователь нажимает и отпускает клавишу

Событие	Описание
OnKeyDown	Пользователь нажимает клавишу над элементом
OnKeyUp	Пользователь отпускает клавишу над элементом
OnSubmit	Данные из формы переданы Web-серверу
OnReset	Форма очищена
OnSelect	Пользователь выбирает текст в текстовом поле
OnChange	Потеря фокуса ввода элементом после изменения его значения

Назначение обработчика события выполняется путем указания имени события в виде атрибута тега, например так:

```
<a name="test" onClick="alert('Hello, world!');">say "Hello"</a>
```

При использовании событий, следует понимать, что не каждый элемент может породить определенное событие. Например в следующем примере вызов функции resetAll не произойдет, поскольку элемент <a>, никогда не породит событие onReset;

```
...
<script>
function resetAll() {
        // do something
        }
</script>
...
<a href="clear.htm" onReset="resetAll();">Сброс</a>
...
```

Объект RegExp

При работе с веб-страницами часто возникает необходимость выполнить сложную обработку текста. В javascript для этого имеется встроенный объект RegExp, который позволяет работать с регулярными выражениями.

Работа с объектном RegExp в javascript мало отличается от работы с любыми другими объектами, но сам синтаксис регулярных выражений требует понимания и практики. Хорошая статья по этой теме написана М.С.Выскорко, она приводится здесь в качестве руководства по регулярным выражениям в javascript.

Примеры скриптов

В листингах 1-6 приведены примеры простых скриптов, иллюстрирующими базовые возможности javascript при работе с объектами веб-документа. При выполнении заданий используйте предлагаемые примеры в качестве образцов.

WARNING: Имейте ввиду, что различные браузеры могут по разному выполнять код javascript (или даже не выполнять его совсем).

Листинг 1. Ограничение количества символов

```
<html>
<head>
<title>Ограничение количества вводимых символов</title>
<script type="text/javascript">
var maxLen = 25;
function checkMaxinput(form) {
        if (form.message.value.length > maxLen)
                form.message.value = form.message.value.substring(0, maxLen);
        else
                form.remLen.value = maxLen —
                form.message.value.length;
                }
</script>
</head>
<body>
<form name=myform action="somehandler.cgi">
<h1>Ограничение количества вводимых символов<h1>
<textarea name=message cols=28 rows=4 onKeyDown="checkMaxinput(this.form)"
        onKeyUp="checkMaxinput(this.form)"></textarea>
<p>Осталось <input readonly type=text name=remLen size=3 value="25"> символов</p>
</form>
</body>
</html>
```

Листинг 2. Проверка ввода

```
<html>
<head>
<title>Проверка ввода
</title>
<SCRIPT type="text/javascript">
function checkIt(){
```

```
        var t0=document.getElementById('first').value;
        var t1=document.getElementById('second').value;
        if (t0 == "" || t0 == "Имя") {
                alert("Вы не указали свое имя!"); return false;
                }
        if (t1 == "") {
                alert("Вы не ввели необходимую информацию!");
                return false;
                }
return true;
</SCRIPT>
</head>
<body>
<form method='get' action='somescript.php'>
<input id="first" type="text" size=60px value='Имя'>
<br>
<textarea id="second" rows=4 cols=60></textarea>
<br>
<input type='submit' onClick="if (!checkIt()){return false;}" value="OK">
</form>
</body>
</html>
```

Листинг 3. Управление окнами (используется объект window)

```
<html>
<head>
<title>Открытие/закрытие нового окна</title>
</head>
<body>
<p><a name="demoOpen" onClick="mywindow =
window.open('window.htm','mywin','height=120, width=300, left=100, top=30');">Открыть</a>
<a name="demoClose" onClick="mywindow.window.close();">Закрыть</a>
</body>
</html>
```

Листинг 4. Изменение оформления

```
<html>
<HEAD>
<TITLE>Изменение цвета объекта по щелчку мыши</TITLE>
</head>
<BODY>
<p onClick="fgColor='#3CB094';bgColor='#FFFF00';">CLICK 4 REDRAW</p>
</BODY>
</HTML>
```

Листинг 5. Текущее время (использован встроенный объект Date)

```
<html>
<HEAD>
<TITLE>Часы, отображающие текущее время</TITLE>
<script type="text/javascript">
function fulltime() {
        var time=new Date();
        document.clock.full.value=time.toLocaleString(); // 1-ый вариант
        document.getElementById("jsclock").innerHTML=time.toLocaleString(); // 2-ой
вариант
        setTimeout('fulltime()',500) }
</script>
</head>
<body>
<form name=clock>
<input type=text size=20 name=full><!-- 1-ый вариант -->
<span id="jsclock"></span><!-- 2-ой вариант -->
</form>
<script type="text/javascript"> fulltime(); </script>
</BODY>
</HTML>
```

Листинг 6. Определение браузера (использован объект navigator)

```
<HTML>
<HEAD>
```

```
<TITLE>Сведения о браузере</TITLE>
</HEAD>
<BODY>
<h1>Для навигации в Web вы используете:</h1>
<ul>
<SCRIPT type="text/javascript">
        document.write("<li>Имя программы:<b>"+navigator.appName+"</b>");
        document.write("<li>Версия:<b>"+navigator.appVersion+"</b>");
        document.write("<li>Пользовательский агент:<b>"+navigator.userAgent+"</b>");
        document.write("<li>Платформа: <b>"+navigator.platform+"</b>");
</SCRIPT>
</ul>
</BODY>
</HTML>
```

2.17.3 Результаты и выводы:

В результате выполнения данного практического задания мы научились работать со сложными JavaScript скриптами и создавать реакции на различные события.

2.18 Практическое занятие № 18 (2 часа).

Тема: «Принципы технологии DHTML»

2.18.1 Задание для работы:

1. Создать файловую структуру Web-сайта.
2. Задать каркас онлайн-калькулятора
3. Подключение элементов динамического HTML

2.18.2 Краткое описание проводимого занятия:

Простой калькулятор на javascript

Этот простой скрипт онлайн-калькулятора иллюстрирует некоторые возможности DHTML. HTML и CSS использованы для формирования пользовательского интерфейса калькулятора. HTML-формы и встроенный javascript — для выполнения ввода данных, вычисления четырех математических операций и вывода результа.

Исходный код примера приведен в листинге 1, внешний вид калькулятора — на рис. 1, а посмотреть его работу можно здесь.

Листинг 1. Калькулятор на javascript.

```
<html>
<head>
<meta charset="utf-8" />
<title>Калькулятор</title>
<style type="text/css">
	#calculator * {font-size: 16px;}
	#calculator table {border: solid 3px silver; border-spacing: 3px; background-color: #EEE; }
	#calculator table td {border-spacing: 3px;}
	input.display {width: 166px; text-align: right;}
	td.buttons {border-top: solid 1px silver;}
	input[type= button] {width: 40px; height: 30px;}
</style>
</head>

<body>
<form name="calc" id="calculator">
	<table>
	<tr>
	<td>
		<input type="text" name="input" size="16" class="display">
	</td>
	</tr>
	<tr>
	<td class="buttons">
		<input type="button" name="one" value="1" OnClick="calc.input.value += '1'">
		<input type="button" name="two" value="2" OnClick="calc.input.value += '2'">
		<input type="button" name="three" value="3" OnClick="calc.input.value += '3'">
```

```
            <input type="button" name="add" value="+"
OnClick="calc.input.value += '+'">

            <br>

            <input type="button" name="four" value="4"
OnClick="calc.input.value += '4'">

            <input type="button" name="five" value="5"
OnClick="calc.input.value += '5'">

            <input type="button" name="six" value="6"
OnClick="calc.input.value += '6'">

            <input type="button" name="sub" value="-"
OnClick="calc.input.value += '-'">

            <br>

            <input type="button" name="seven" value="7"
OnClick="calc.input.value += '7'">

            <input type="button" name="eight" value="8"
OnClick="calc.input.value += '8'">

            <input type="button" name="nine" value="9"
OnClick="calc.input.value += '9'">

            <input type="button" name="mul" value="x"
OnClick="calc.input.value += '*'">

            <br>

            <input type="button" name="clear" value="c"
OnClick="calc.input.value = ''">

            <input type="button" name="zero" value="0"
OnClick="calc.input.value += '0'">

            <input type="button" name="doit" value="="
OnClick="calc.input.value = eval(calc.input.value)">

            <input type="button" name="div"  value="/"
OnClick="calc.input.value += '/'">

      </td>

      </tr>

      </table>

</form>

</body>

</html>
```

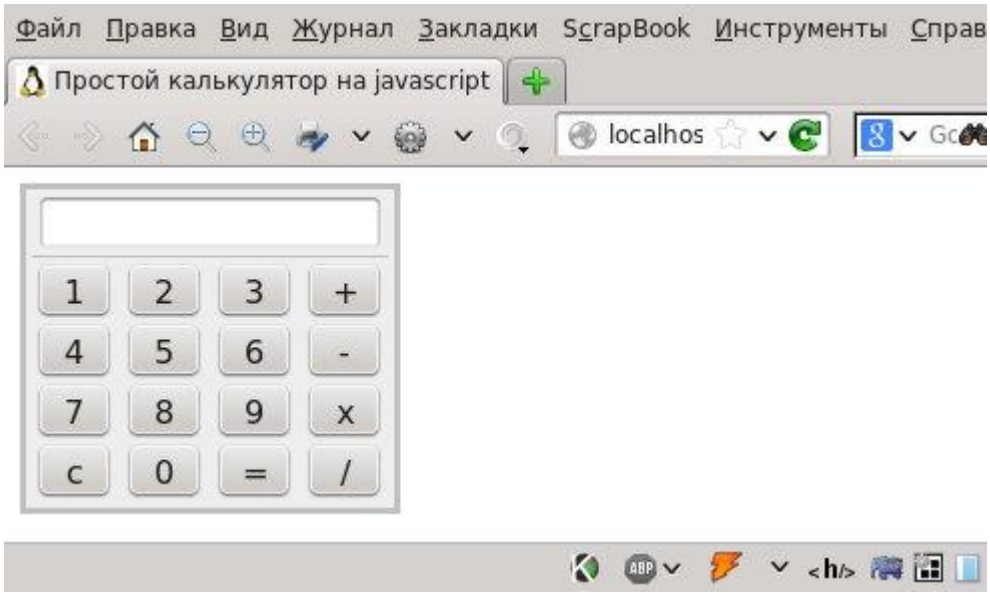


Рис. 1. Внешний вид калькулятора

Рис. 2. Научный онлайн-калькулятор

Алгоритм работы этого скрипта очень прост: по клику на кнопках в текстовое поле добавляется соответствующее значение (число, символ операции или пустая строка для сброса). По нажатию кнопки "=" вызывается библиотечная функция eval, которая принимает в качестве аргумента строку кода, который может быть выполнен. В рассматриваемом случае эта строка содержит выражение вида `"число1 операция число2"`. Результат выполнения помещается в строку вывода. По нажатию кнопки "c" выполняется очистка вывода.

Приведенный скрипт в исходном виде слабо применим на практике. Он никоим образом не претендует на возможности профессиональных онлайн-калькуляторов (один из примеров — на рис. 2) и, с точки зрения функциональности, требует доработки как минимум в следующих аспектах:

- вводимые данные не проверяются (с клавиатуры можно ввести любые данные непосредственно в поле ввода);
- с помощью мыши нет возможности вводить вещественные числа (отсутствует ввод десятичного разделителя);
- не предусмотрена обработка ошибок (в частности, деления на ноль).

С точки зрения программирования на javascript, этот пример может быть изрядно улучшен путем вынесения обработчиков во внешние функции. Также вместо имен объектов удобней использовать их идентификаторы и обращаться по ним через GetElementByID. Так или иначе, приведенный код javascript-калькулятора является

учебным примером и указанные выше недостатки вы можете устранить самостоятельно.

2.18.3 Результаты и выводы:

В результате выполнения данной практической работы мы научились работать с DHTML и применили полученные навыки для написания программы – онлайна-калькулятора.