Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра экономики

ИНТЕРАКТИВНЫЙ МАРКЕТИНГ И ЭЛЕКТРОННАЯ КОММЕРЦИЯ

Лабораторный практикум

для студентов специальности «Маркетинг» дневной формы обучения

Минск 2006

УДК 339.138(075.8)
ББК 65.290-2 я73
И 73

Автор-составитель:
Е.В. Бесчастная

Интерактивный маркетинг и электронная коммерция: Лаб. практикум для студ. спец. «Маркетинг» дневной формы обуч. / Сост. Е.В. Бесчастная. – Мн.: БГУИР, 2006. – 78 с.
И 73
ISBN 985-444-944-0

Данное пособие содержит систематизированные основы создания HTML-страниц с использованием Java-приложений. Эти материалы будут способствовать развитию системного подхода к вопросам электронной коммерции в профессиональной подготовке будущих специалистов экономического профиля.

УДК 339.138(075.8)
ББК 65.290-2 я73

ISBN 985-444-944-0

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сложный характер и динамизм современных мирохозяйственных связей вызвали быстрый рост спроса на услуги связи и информатики, что в свою очередь привело к созданию новых телекоммуникационных технологий, порождающих новые услуги и соответственно увеличивающуюся потребность в них. Многие деловые люди и организации столкнулись с необходимостью использования в своей повседневной деятельности электронной почты, информационных ресурсов Internet и других интерактивных коммерческих информационных служб, подключения ПК к локальной сети, пересылки файлов в другой город, участия в телеконференциях и т.д.

Объем и способы информирования специалистов-маркетологов с помощью средств компьютерных коммуникаций коренным образом изменились за последние годы. И если ранее подобные средства предназначались лишь для узкого круга специалистов и опытных пользователей ПК, то теперь они рассчитаны на самую широкую аудиторию.

В настоящее время передача данных с помощью компьютеров, использование локальных и глобальных компьютерных сетей становится столь же распространенным, как и сами ПК.

Целью данного курса лабораторных работ является подготовка экономистов – пользователей ПК, не являющихся специалистами в области вычислительной техники, к умелому использованию глобальных компьютерных сетей, коммуникационного оборудования и программных средств.

После окончания изучения представленного раздела студенты должны уметь применять новейшие версии броузера Microsoft Internet Explorer для доступа к различным службам Internet, создавать WWW-сайты и Интернет-магазины с помощью языка гипертекстовой разметки, а также JavaScript для дальнейшего использования их на Web-серверах.

ЛАБОРАТОРНАЯ РАБОТА № 1
Создание простейших Web-страниц

Цель работы:

- приобрести начальные навыки создания простейших Internet-документов;
- научиться выполнять форматирование созданных Web-страниц;
- приобрести опыт создания списочных структур различных типов;
- изучить возможности языка HTML для представления данных в табличном виде;
- научиться создавать многоколоночные документы.

1. Общие положения языка HTML

Язык HTML (Hypertext Markup Language, язык разметки гипертекста) – это язык, на котором создаются Web-страницы. HTML-документы могут просматриваться различными типами Web-броузеров. Когда документ создан с использованием HTML, Web-броузер может интерпретировать HTML для выделения различных элементов документа и первичной их обработки. Использование HTML позволяет форматировать документы для их представления с использованием шрифтов, линий и других графических элементов на любой системе, их просматривающей.

Web-страницы могут быть созданы с помощью:

1) обычного текстового редактора;
2) редактора, способного сохранять в формате HTML;
3) специализированного редактора;
4) специализированной системы.

HTML-документы сохраняются на диске как обычные текстовые документы в формате ASCII. Для распознавания Web-страниц по их именам общепринято обозначать такие файлы использованием расширений .HTM (для Windows 3.1) или .HTML (для Windows 95/98/NT/XP, Macintosh и Unix).

Кроме полезного текста в HTML-документах используются специальные управляющие последовательности символов – тэги.

Структура тэга имеет следующий вид:

<ТЭГ параметр1=значение1...>
текст
другие конструкции
</ТЭГ>
или
<ТЭГ параметр1=значение1...>
текст
другие конструкции

Чаще всего тэги используются попарно, окружая размеченные фрагменты текста. Такие тэги называются контейнерами. Закрывающийся тэг отличается от начального только присутствием символа «/», добавляемого перед именем тэга. При интерпретации тэгов броузер не делает различия между строчными и прописными буквами. Поэтому сами тэги можно набирать на любом регистре. Зачастую параметр (атрибут) является необязательной величиной и его можно пропускать.

2. Структура HTML-документа

Когда Web-броузер получает документ, он определяет, как документ должен быть интерпретирован. Самый первый тэг, который встречается в документе, должен быть тэгом <HTML>. Данный тэг сообщает Web-броузеру, что документ написан с использованием HTML. Минимальный HTML-документ будет выглядеть так:

<HTML> ...тело документа... </HTML>

В общем виде Web-документ имеет следующую структуру:

```
<HTML>
 <HEAD>
  <TITLE>
   … заголовок
  </TITLE>
 </HEAD>
 <BODY>
  … тело документа
 </BODY>
</HTML>
```

Тэги <HTML> и </HTML> заключают внутри себя все элементы HTML-кода, указывая, что используется язык HTML.

Тэги <HEAD> и </HEAD> обозначают заголовок Web-документа. Как правило, заголовок содержит название документа, информацию для индексирования и некоторые общие установки для данного документа. Тэг заголовочной части документа должен быть использован сразу после тэга <HTML> и более нигде в теле документа.

Тэги <BODY> и </BODY> обрамляют оставшуюся часть документа (тело). Здесь размещается основная смысловая текстовая и графическая информация.

Разделение документа на заголовочную часть и тело имеет лишь смысловую нагрузку. Текст, приведенный в любой из этих частей, на экране выглядит совершенно одинаково.

Внутри контейнера <HEAD> может использоваться тэг <TITLE>, как показано выше. Большинство Web-броузеров отображают содержимое этого тэга в заголовке окна, содержащего документ и в файле закладок, если он поддерживается Web-броузером. Заголовок документа не появляется при отображении самого документа в окне.

Как любой язык, HTML позволяет вставлять в тело документа пояснительный текст (комментарии), который сохраняется при передаче документа по сети, но не отображается броузером. Синтаксис комментария следующий:

<!-- Это комментарий -->

Комментарии могут встречаться в документе где угодно и в любом количестве.

3. Форматирование текста

При выводе на экран текста броузер игнорирует дополнительные пробелы, символы табуляции и возврата каретки, пустые строки. Их можно использовать для того, чтобы сделать текст HTML-документа более легко читаемым, но для правильного его отображения на Web-странице следует использовать специальные тэги.

3.1. Цвета в HTML-документе

Язык HTML определяет следующие типы цветов: BGCOLOR (цвет фона для тела документа), TEXT (цвет, используемый при выводе на экран текста из данного документа), LINK (цвет, который будет использоваться при выводе на экран текста из еще не выбранных вами гипертекстовых связей), VLINK (цвет, который будет использоваться при выводе на экран текста из уже проверенных вами гипертекстовых связей), ALINK (цвет, которым будут выделяться в тексте гипертекстовые связки в тот момент, когда пользователь щелкает по ним клавишей мыши).

Существует две формы задания цвета: символьная (указывается название одного из предопределенных цветов) и цифровая (комбинация RGB:#RRGGBB).

Символьные идентификаторы основных цветов и их RGB-комбинации приведены ниже: BLACK (#000000) – черный, SILVER (#C0C0C0) – серебряный, GRAY (#808080) – серый, WHITE (#FFFFFF) – белый, MAROON (#800000) – темно-красный, RED (#FF0000) – красный, PURPLE (#800080) – темно-сиреневый, FUCHSIA (#FF00FF) – сиреневый, GREEN (#008000) – зеленый, LIME (00FF00) – ярко-зеленый, OLIVE (#808000) – оливковый, YELLOW (#FFFF00) – желтый, NAVY (#000080) – темно-синий, BLUE (#0000FF) – голубой, TEAL (#008080) – сине-зеленый, AQUA (#00FFFF) – ярко-голубой.

Атрибут, указывающий на цвет, может использоваться в тэгах <BODY>, <FONT>, <HR>, <MARQUEE>, <TABLE>.

3.2. Элементы форматирования на уровне блоков

Тэг абзаца (параграфа)

<P ALIGN=LEFT|CENTER|RIGHT|JUSTIFY NOWRAP> текст </P>

разделяет два абзаца пустой строкой. Может не иметь пары </P>. Параметр ALIGN задает выравнивание информации. Применение параметра NOWRAP дает возможность писать текст без переноса слов. Для центрирования текста или графики можно использовать также контейнер <CENTER>.

Тэги заголовков

<H1|H2|H3|H4|H5|H6 align=...> текст </H1|H2|H3|H4|H5|H6>

используются для выделения структурных частей текста. Каждый стиль заголовка имеет свой размер. Тэг <H1> имеет наибольший размер.

Тэг горизонтальной линейки

<hr align=... size=... width=... noshade>

предназначен для вычерчивания горизонтальной линии. Атрибут SIZE задает толщину линии в пикселях, WIDTH – ее ширину в пикселах или процентах от ширины окна броузера. Атрибут NOSHADE позволяет представить линию без тени в виде простой темной полоски.

Тэг конца строки

вызывает переход на новую строчку без разрыва абзаца.

Тэг <WBR> – определяет место возможного (рекомендуемого) переноса (разрыва) строки.

Контейнер <NOBR> заключает в себе текст, который не должен разбиваться на строки, даже если она выходит за границы экрана. Вместо этого броузер позволит горизонтально прокручивать текст.

Для отображения на экране символов табуляции, возврата каретки, дополнительных пробелов используется контейнер <PRE>. При отображении такого текста используется моноширинный шрифт. Внутри контейнера могут использоваться другие тэги форматирования.

Контейнер <BLOCKQUOTE> предназначен для обозначения в документе цитаты из другого источника. Текст, обозначенный этим тэгом, отступает от левого края документа на 8 пробелов.

```
<HTML>
<head><title>Приёмы форматирования</title></head>
<h1>Заголовок уровня1</h1></font>
деление на строки<br>
для отображения в броузере<br>
<h2>Заголовок уровня2</h2>
<p>можно также использовать тег абзаца, но он парный
```

```
</p>
<p>это новый абзац</p>
<h3 >Заголовок уровня3</h3>
<hr Noshade width=50% size=4 font color=gold>
<pre>
Существуют специальные теги для текста,
предварительно разбитого на строки и с лишними пробелами</pre>
разрыв строки</p>
</body></html>
```

3.3. Элементы, задающие шрифт

<TT> – телетайпный текст (моноширинный).
<I> – стиль с наклонным шрифтом (курсив).
<B> – стиль с жирным шрифтом.
<U> – стиль с подчеркиванием текста.
<BIG> – печать текста шрифтом увеличенного размера (большего, чем окружающий текст).
<SMALL> – печать текста шрифтом уменьшенного размера (меньшего, чем окружающий текст).
<SUB> – печать текста со сдвигом вниз(нижний индекс).
<SUP> – печать текста со сдвигом вверх (верхний индекс).
<STRIKE> или <S> – стиль с перечеркиванием текста.
<FONT size=… color=... face=...> текст </FONT>

size – устанавливает размер шрифта, который будет использоваться текстом, содержащимся в пределах элемента font. Можно задать абсолютный размер шрифта, указав какое-либо целое число от 1 до 7. Для шрифта можно также указывать относительный размер, сообщая атрибуту целое число со знаком (например, это может быть size="+1" или size="-2").

color – указывает цвет, которым будет выделен данный кусок текста. Цвета задаются в виде RGB-значения с шестнадцатеричной нотацией либо выбирается один из стандартных цветов.

face – задает гарнитуру шрифта.

```
<HTML>
<head><body bgcolor=fuschia text=teal><title>Приёмы форматирования</title></head>
<h1><font color=maroon>Заголовок уровня1</h1></font>
<center>можно центрировать текст<br>
деление на строки<br>
для отображения в броузере<br></center>
<h2 Align=right>Заголовок уровня2</h2>
```

<right>можно также использовать тег абзаца, но он парный
</right>
<JUSTIFY>это новый абзац
<h3 Align=left>Заголовок уровня3</h3>
<hr font color=gold>
<pre>
Существуют специальные теги для текста,
предварительно разбитого на строки и с лишними пробелами</pre>
</body></html>

4. Специальные тэги HTML

Тэг <ADDRESS> используется для выделения автора документа и его адреса (например e-mail). Синтаксис:

<ADDRESS> Адрес-автора </ADDRESS>

Некоторые символы являются управляющими символами в HTML и не могут напрямую использоваться в документе:

- левая угловая скобка «<»
- правая угловая скобка «>»
- амперсанд «&»
- двойные кавычки «"»

Чтобы использовать данные символы в документе, необходимо заменить их escape-последовательностями: < – < > – > & – & " – ".

Существует большое количество escape-последовательностей для обозначения специальных символов, например "©" для обозначения знака © и "®" для значка ®. Одной из особенностей является замена символов во 2-й части символьной таблицы (после 127-го символа) на escape-последовательности для передачи текстовых файлов с национальными языками по 7-битным каналам.

Escape-последовательности чувствительны к регистру: НЕЛЬЗЯ использовать < вместо <.

5. Списки

Существует три основных вида списков в HTML-документе: нумерованный, ненумерованный, список определений.

Можно создавать вложенные списки, используя различные тэги списков или повторяя одни внутри других. Для этого просто необходимо разместить одну пару тэгов (стартовый и завершающий) внутри другой. Будут ли элементы вложенного списка иметь те же маркеры, обозначающие элемент списка, зависит от броузера.

Для создания заголовка списка используется контейнер <LH>.

В нумерованном списке броузер автоматически вставляет номера элементов по порядку. Если вы удалите произвольное количество элементов нумерованного списка, то остальные номера автоматически будут пересчитаны.

Нумерованный список заключается в контейнер <OL>. Каждый элемент списка начинается с тэга <LI>. Например:

<OL>
<LI> Первый пункт списка
<LI> Второй пункт списка
<LI> ...
</OL>

Тэг <OL> может иметь параметры:

<OL TYPE=A|a|I|i|1 START=n>

где TYPE – вид счетчика: A – большие латинские буквы (A,B,C...); a – маленькие латинские буквы (a,b,c...); I – большие римские цифры (I,II,III...); i – маленькие римские цифры (i,ii,iii...); 1 – обычные цифры (1,2,3...). Используется по умолчанию.

START=n – число, с которого начинается отсчет.

Для ненумерованных списков броузер обычно использует маркеры для пометки элемента списка. Вид маркера, как правило, настраивает пользователь броузера.

Ненумерованный список заключается в контейнер <UL>. Как и в случае нумерованного списка каждый элемент начинается с тэга <LI>.

Пример:

<UL>
<LI> Первый пункт списка
<LI> Второй пункт списка
<LI> ...
</UL>

Тэг <UL> может иметь параметр:

<UL TYPE=disc|circle|square>

Тип тэга <UL> определяет внешний вид маркера как вид по умолчанию (disc), круглый (circle) или квадратный (square). *Пример:*

<html><title>
Использование списков</title>
<body>
<H3>Домашняя страница</H3>
<h4>Ненумерованный список моих интересов</h4>
<font color=red>
<lh>Занятия в свободное время</lh>
 компьютер
 бассейн

```
</ul></font></hr>
<h4>Нумерованный список моих интересов</h4>
<ol type=1>
<lh><b>Моё путешествие</b></lh>
<li>прибытие в Варшаву
<li>далее в Будапешт
<li>потом в Рим
</ol><hr>
<ol type=A>
<lh><b>Продолжение путешествия</b></lh>
<li>Автобусом в Берлин
<li>Поездом в Варшаву
<li>Поездом в Минск
</ol><hr>
</body></html>
```

Список определений заключается в контейнер <DL>. Список состоит из двух частей: термина и его описания. Каждый термин начинается тэгом <DT> , а описание – тэгом <DD>.

Пример:

```
<DL>
<DT> Отдел маркетинга
<DD> Данный отдел занимается продвижением продуктов и услуг
<DT> Финансовый отдел
<DD> Данный отдел занимается всеми финансовыми операциями
<DT> Отдел кадров
<DD> Данный отдел занимается учетом и набором новых сотрудников фирмы, распределением отпусков, отслеживанием больничных листов и т.д.
</DL>
```

6. Таблицы

Таблицы в HTML организуются как набор столбцов и строк. Ячейки таблицы могут содержать любые HTML-элементы, такие, как заголовки, списки, абзацы, фигуры, графику, а также элементы форм.

Синтаксис определения таблицы в общем виде:

```
<TABLE BORDER=...  WIDTH=... >
 <TR>
        <TD параметры=… > 1-я клетка 1-й строки </TD>
        <TD параметры=... > 2-я клетка 1-й строки </TD>
 </TR>
 <TR>
  <TD> 1-я клетка 2-й строки </TD>
  ...
 </TR>
 ...
</TABLE>
```

Основные тэги таблицы:

- Таблица <TABLE>...</TABLE>. Это основные тэги, описывающие таблицу. Все элементы таблицы должны находиться внутри этих двух тэгов. По умолчанию таблица не имеет обрамления и разделителей. Обрамление добавляется атрибутом BORDER.
- BGCOLOR – задает цвет фона таблицы.
- BORDERCOLOR – задает цвет рамки.
- BORDERCOLORLIGHT – задает цвет светлой части трехмерной рамки.
- BORDERCOLORDARK – задает цвет темной части трехмерной рамки.
- Подпись: <CAPTION>...</CAPTION>. Данный тэг описывает название таблицы (подпись). Тэг <CAPTION> должен присутствовать внутри <TABLE>...</TABLE>, но снаружи описания какой-либо строки или ячейки. По умолчанию <CAPTION> имеет атрибут ALIGN=top, но может быть явно установлен в ALIGN=bottom. ALIGN определяет, где будет поставлена подпись (сверху или снизу таблицы). Подпись всегда центрирована в рамках ширины таблицы.
- Строка таблицы: <TR>...</TR>. Количество строк таблицы определяется количеством встречающихся пар тэгов <TR>..</TR>. Строки могут иметь атрибуты ALIGN и VALIGN, которые описывают визуальное положение содержимого строк в таблице.
- Ячейка таблицы: <TD>...</TD>. Описывает стандартную ячейку таблицы. Ячейка таблицы может быть описана только внутри строки таблицы. Каждая ячейка должна быть пронумерована номером колонки, для которой она

описывается. Если в строке отсутствует одна или несколько ячеек для некоторых колонок, то броузер отображает пустую ячейку. Расположение данных в ячейке по умолчанию определяется атрибутами ALIGN=left и VALIGN=middle. Данное расположение может быть исправлено как на уровне описания строки, так и на уровне описания ячейки.

• Заголовок таблицы: <TH>...</TH>. Ячейка заголовка таблицы имеет ширину всей таблицы; текст в данной ячейке имеет атрибут BOLD и ALIGN=center.

Основные атрибуты таблицы:

• BORDER – используется в тэге TABLE. Если данный атрибут присутствует, граница таблицы прорисовывается для всех ячеек и для таблицы в целом. BORDER может принимать числовое значение, определяющее ширину границы, например BORDER=3.

• ALIGN. Если атрибут ALIGN присутствует внутри тэгов <CAPTION> и </CAPTION>, то он определяет положение подписи для таблицы (сверху или снизу). По умолчанию ALIGN=top. Если атрибут ALIGN встречается внутри <TR>, <TH> или <TD>, он управляет положением данных в ячейках по горизонтали. Может принимать значения left (слева), right (справа) или center (по центру).

• VALIGN – встречается внутри тэгов <TR>, <TH> и <TD>. Он определяет вертикальное размещение данных в ячейках. Может принимать значения top (вверху), bottom (внизу), middle (по середине) и baseline (все ячейки строки прижаты кверху).

• NOWRAP – говорит о том, что данные в ячейке не могут логически разбиваться на несколько строк и должны быть представлены одной строкой.

• COLSPAN – указывает, какое количество ячеек будет объединено по горизонтали для указанной ячейки. По умолчанию – 1.

• ROWSPAN – указывает, какое количество ячеек будет объединено по вертикали для указанной ячейки. По умолчанию – 1.

• COLSPEC – позволяет задавать фиксированную ширину колонок либо в символах, либо в процентах, например COLSPEC="20%".

• CELLSPACING – задает расстояние между ячейками таблицы (по умолчанию 2 пикс.).

• CELLPADDING – определяет растояние между рамкой ячейки и ее содержимым (по умолчанию 1 пикс.).

• WIDTH – задает ширину таблицы либо в абсолютных единицах, либо в процентах относительно размера экрана. Используя внутри тэга <TD>, можно указывать ширину ячейки.

• HEIGHT – то же, что и WIDTH, но определяет высоту таблицы.

• FRAME – позволяет описывать внешние рамки таблицы. Может принимать следующие значения:

– VOID – нет рамки;
– ABOVE – отображает внешнюю часть рамки;

– BELOW – отображает нижнюю часть рамки;
– HSIDES – отображает верхнюю и нижнюю части рамки;
– LHS – отображает левую часть рамки;
– RHS – отображает правую часть рамки;
– VSIDES – отображает левую и правую части рамки;
– BOX или BORDER – отображет все части рамки.
– RULES – описывает рамки внутри таблицы. Может принимать следующие значения:

NONE – нет рамок;

GROUPS – отображает горизонтальные части рамки между группами таблицы;

ROWS – отображает горизонтальные части рамки внутри таблицы;

COLS – отображает вертикальные части рамки внутри таблицы;

ALL – отображает все части рамки внутри таблицы.

• Верхние и нижние колонтитулы таблицы задаются контейнерами <THEAD>, <TFOOT>.

Иногда требуется поместить в таблицу пустые ячейки. Это можно сделать, поместив в ячейку пробел (&#32), неразрывный пробел ( ) или тэг
.

Пример:

```
<TABLE BORDER=5>
 <CAPTION ALIGN=bottom> Таблица №1 /CAPTION>
            <TR>
             <TD ROWSPAN=2></TD>
             <TH COLSPAN=2>Среднее значение</TH>
            </TR>
 <TR>
  <TH>Рост</TH>
  <TH>Вес</TH>
 </TR>
            <TR>
             <TD>Мужчины</TD>
             <TD ALIGN=center>174</TD>
             <TD ALIGN=center>78</TD>
            </TR>
            <TR>
             <TD>Женщины</TD>
             <TD ALIGN=center>165</TD>
             <TD ALIGN=center>56</TD>
            </TR>
</TABLE>
```

Результаты работы программы приведены в табл. 1.

Таблица 1

	Среднее значение	
	Рост	Вес
Мужчины	174	78
Женщины	165	56

*Пример:*Использование таблиц для создания макета страницы:

```
<html>
<head>
<title>
Таблицы
</title></head>
<body bgcolor=fuchia>
<table width=100% cellspadding=30>
<tr align=center>
<td colspan=2 bgcolor=#ff08a3><font size=5 face=arial color=fuchia>
Использование таблиц </font></td>
</tr>
<tr>
<td width=50% bgcolor=silver><font color=green >таблицы часто исполь-
зуются для создания макетов страниц
</font></td>
<td bgcolor=green> <font color=silver >это помогает расположить текст
произвольным образом. Не вводите в тег таблицы атрибут <b>boder</b>, чтобы
не отображать рамку таблицы.</font></td>
</tr></body></html>
```

Порядок выполнения работы

1. Создать с помощью программы «Блокнот» гипертекстовый файл. Имя файла – ваша фамилия на английском языке. Файл разместить на указанном диске в папке с номером вашей группы.
2. Задать заголовок, отображаемый в строке заголовка окна броузера. Текст заголовка – ваша фамилия на русском языке.
3. Установить заданные цвета для фона и текста документа. Написать произвольный абзац текста самым крупным шрифтом.
4. Осуществить выравнивание абзаца (отцентрировать, прижать вправо, влево, выровнять по ширине).
5. Написать определение трех любых понятий курса различными цветами и гарнитурами (Arial, Times, моноширинный шрифт). Сам термин должен быть выведен полужирным курсивом с подчеркиванием.

6. Написать формулу

$$(a+b)^2 = a^2 + 2ab + b^2;$$

$$c_{ij} = a_{ij} + b_{ij}.$$

7. Отделить формулу от основного текста горизонтальной линией заданной ширины, толщины и цвета.

8. Создать список (нумерованный, ненумерованный) студентов вашей группы (3 человека). Заголовок списка – "Группа 105xxx". Цвета заголовка и элементов списка должны отличаться друг от друга.

9. Изменить нумерацию элементов списка.

10. Создать список определений для двух понятий курса.

11. Создать таблицу вида (табл. 2).

Таблица 2

Параметр	Температура, °С		
	100	200	300
A_1	2	3	4
A_2	10	15	12

12. Создать таблицу вида:

Текст, написанный гарнитурой Arial синим цветом	Текст, написанный гарнитурой Times красным цветом

Контрольные вопросы

1. Общие положения языка HTML.
2. Структура HTML-документа.
3. Использование цвета в HTML-документах.
4. Элементы форматирования текста средствами языка HTML на уровне блоков.
5. Элементы языка HTML, задающие шрифт.
6. Специальные тэги языка HTML.
7. Структура таблицы в HTML-документе.
8. Основные атрибуты таблицы.
9. Вычерчивание рамок и управление размером и цветом ячеек таблицы.

ЛАБОРАТОРНАЯ РАБОТА № 2
Вставка графики и анимации в Internet-документы. Практическое использование ссылок и пользовательских списков на WWW-страницах. Фреймы

Цель работы:

- научиться использовать статические и динамические графические изображения в WWW-документах;
- научиться реализовывать связи между гипертекстовыми файлами и ссылки на точки внутри HTML-документа;
- приобрести опыт создания фреймовой структуры документа.

1. Графика в HTML

Одним из способов использования графики в HTML является внедрение графических образов в документ, что позволяет пользователю непосредственно увидеть изображения. Такая техника проектирования документов называется «inline image».

Синтаксис тэга:

<IMG SRC="URL" ALT="text" HEIGHT=n1 WIDTH=n2
ALIGN=top|middle|bottom|texttop|absmiddle|baseline|absbottom|left|right
BORDER=n VSPACE=m1 HSPACE=m2 ISMAP>

URL – адрес, указывающий броузеру, где находится рисунок. Рисунок должен храниться в графическом формате, поддерживаемом броузером. На сегодняшний день большинством броузеров поддерживаются форматы GIF и JPG (JPEG). GIF лучше воспроизводит детали изображения. Используется для хранения черно-белых рисунков, рисунков с ограниченным количеством цветов, графики с текстом. JPG занимает меньше места. Используется для хранения отсканированных фотографий, изображений со сложной цветовой палитрой и числом цветов, большим 256.

ALT="text" задает текст, который будет отображен броузером, не поддерживающим отображение графики или с отключенной подкачкой изображений. Обычно это короткое описание изображения, которое пользователь мог бы увидеть на экране. Если данный параметр отсутствует, то на месте рисунка большинство броузеров выводит пиктограмму (иконку), активизировав которую, пользователь может увидеть изображение. Тэг ALT рекомендуется, если ваши пользователи используют броузер, не поддерживающий графический режим, например Lynx.

HEIGTH=n1 используется для указания высоты рисунка в пикселах. Если данный параметр не указан, то используется оригинальная высота рисунка. Это параметр позволяет сжимать или растягивать изображения по вертикали, что в свою очередь позволяет более четко определять внешний вид документа. Одна-

ко некоторые броузеры не поддерживают данный параметр. С другой стороны, экранное разрешение у вашего клиента может отличаться от вашего, поэтому будьте внимательны при задании абсолютной величины графического объекта.

WIDTH=n2 позволяет задать абсолютную ширину рисунка в пикселах.

ALIGN используется, чтобы сообщить броузеру, куда поместить следующий блок текста. Это позволяет более строго задать расположение элементов на экране. Если данный параметр не используется, то большинство броузеров располагает изображение в левой части экрана, а текст – справа от него.

BORDER=n позволяет автору определить ширину рамки вокруг рисунка (n=0…9).

VSPACE=m1 позволяет установить размер в пикселах пустого пространства над и под рисунком, чтобы текст не наезжал на рисунок. Особенно это важно для динамически формируемых изображений, когда нельзя заранее увидеть документ.

HSPACE=m2 выполняет то же, что и VSPACE, но только по горизонтали.

ISMAP сообщает броузеру, что данное изображение позволяет пользователю выполнять какие-либо действия, щелкая мышью на определенном месте изображения. Данная возможность является расширением HTML и в круг вопросов, освещаемых в данном пособии, не входит.

Большинство броузеров позволяет включать в документ фоновый рисунок. Некоторые пользователи любят фоновую графику, некоторые нет. Ненавязчивый полупрозрачный рисунок (обои) обычно хорошо выглядит в качестве фона для большинства документов.

Описание фонового рисунка включается в тэг BODY и выглядит следующим образом:

<BODY BACKGROUND="picture.gif">

Тэг <EMBED> аналогичен <IMG>, но позволяет включать в Web-страницу файлы произвольного типа. Для них требуется подключение дополнительных модулей (plug-in). Примеры файлов: AVI-файлы и др.

Пример:

```
<HTML>
<head><title>рисунок</title></head>
<body>
<<EMBED src="clock.avi"></body></html>
```

2. Динамические изображения и анимация

Кроме обычных GIF-файлов, содержащих статические изображения, язык HTML дает возможность включать в документы динамические GIF-файлы.

Бегущая строка. Синтаксис тэга

<MARQUEE BEHAVIOR=alternate|slide GCOLOR=...
WIDTH=... HEIGHT=...> текст </MARQUEE>

3. Гипертекстовые связи

Гипертекстовые связи (ссылки) являются наиболее важным элементом Web-страниц. С их помощью вы делаете документ связанным и структурированным, что позволяет пользователю получать необходимую ему информацию максимально быстро и удобно.

Ссылки могут указывать на другой документ, специальное место данного документа или выполнять другие функции, например запрашивать файл по FTP-протоколу для отображения его броузером. URL может указывать на специальное место по абсолютному пути доступа или указывать на документ в текущем пути доступа, что часто используется при организации больших структурированных Web-сайтов. В URL после имени файла через диез может указываться специальный маркер. Данный элемент является ссылкой на строку (точку) внутри HTML-документа.

Синтаксис тэга, позволяющий задать ссылку в HTML-документе, следующий:

<A HREF="URL"> текст-ссылка </A>

Тэг <A HREF="URL">открывает описание ссылки, а тэг </A> – закрывает его. Любой текст, находящийся между данными двумя тэгами, подсвечивается специальным образом Web-броузером. Обычно этот текст отображается подчеркнутым и выделенным синим (или другим заданным пользователем) цветом. Текст, обозначающий URL, не отображается броузером, а используется только для выполнения предписанных им действий при активизации ссылки (обычно при щелчке мыши на подсвеченном или подчеркнутом тексте).

Можно делать ссылки на различные участки или разделы одного и того же документа, используя специальный скрытый маркер. Это позволяет быстро переходить от одного раздела к другому внутри документа, не используя прокрутку экрана. Как только вы щелкнете на ссылке, броузер переместит вас на указанный раздел документа, а строка, в которой стоит маркер данного раздела (обычно первая строка раздела или его заголовок) будет размещена на первой строке окна броузера (если данная строка не присутствует уже на экране).

Для создания такой ссылки необходимо выполнить следующие шаги:

1. Создайте маркер раздела:
<A NAME="named_anchor"> Текст, к которому идет переход </A>
2. Создайте ссылку на данный маркер:
<A HREF="#named_anchor"> текст-ссылка </A>

4. Фреймы

Фреймы – достаточно мощный механизм представления информации на Web-страницах. С помощью фреймов экран разделяется на несколько прокручиваемых областей, в каждой из которых отображается содержимое отдельной страницы и даже отдельного Web-узла. Каждый фрейм может иметь следующие свойства:

- Каждый фрейм имеет свой URL, что позволяет загружать его независимо от других фреймов.
- Каждый фрейм имеет собственное имя (параметр NAME), позволяющее переходить к нему из другого фрейма.
- Размер фрейма может быть изменен пользователем прямо на экране при помощи мыши (если это не запрещено указанием специального параметра).

Данные свойства фреймов позволяют создавать продвинутые интерфейсные решения, такие как:

- Размещение статической информации, которую автор считает необходимым постоянно показывать пользователю, в одном статическом фрейме. Это может быть графический логотип фирмы, авторское право, набор управляющих кнопок.
- Помещение в статическом фрейме оглавления всех или части Web-документов, содержащихся на Web-сервере, что позволяет пользователю быстро находить интересующую его информацию.
- Создавать окна результатов запросов, когда в одном фрейме находится собственно запрос, а в другом – результаты запроса.
- Создавать формы типа «мастер-деталь» для Web-приложений, обслуживающих базы данных.

Формат документа, использующего фреймы, внешне очень напоминает формат обычного документа, только вместо тэга BODY используется контейнер FRAMESET, содержащий описание внутренних HTML-документов, содержащих собственно информацию, размещаемую во фреймах.

```
<HTML>
<HEAD>...</HEAD>
<FRAMESET>...</FRAMESET>
</HTML>
```

Однако фрейм-документ является специфичным видом HTML-документа, поскольку не содержит элемента BODY и какой-либо информационной нагрузки соответственно. Он описывает только фреймы, которые будут содержать информацию (кроме случая двойного документа, который мы рассмотрим позже).

Общий синтаксис фреймов:

```
<FRAMESET COLS="value" | ROWS="value">
 <FRAME SRC="URL1">
 <FRAME SRC="URL2">
 ...
</FRAMESET>
```

Общий контейнер FRAMESET описывает все фреймы, на которые делится экран. Можно разделить экран на несколько вертикальных или горизонтальных фреймов. Внутри контейнера могут находиться только тэг <FRAME>, вложенные контейнеры <FRAMESET> и <NOFRAME>, которые позволяют строить двойные документы для броузеров, поддерживающих и не поддерживающих фреймы.

Данный тэг имеет два параметра: ROWS и COLS.

ROWS="список-определений-горизонтальных-подокон". Данный тэг содержит описания некоторого количества подокон, разделенные запятыми. Каждое описание представляет собой числовое значение размера подокна в пикселах, процентах от всего размера окна или связанное масштабное значение. Количество подокон определяется количеством значений в списке. Общая сумма высот подокон должна составлять высоту всего окна (в любых измеряемых величинах). Отсутствие атрибута ROWS определяет один фрейм, величиной во все окно броузера.

COLS="список-определений-вертикальных-подокон". То же самое, что и ROWS, но делит окно по вертикали, а не по горизонтали.

В качестве списка определений могут выступать:

1. Простое числовое значение определяет фиксированный размер подокна в пикселах. Это не самый лучший способ определения размера подокна, т.к. различные броузеры имеют различный размер рабочего поля, не говоря уже о различных экранных разрешениях у пользователя. Если вы используете данный способ описания размера, то настоятельно рекомендуется сочетать его с каким-либо другим, чтобы в результате вы точно получили 100 %-ное заполнение окна броузера.

2. Значение величины подокна в процентах от 1 до 100. Если общая сумма процентов описываемых подокон меньше или превышает 100, то размеры всех фреймов пропорционально увеличиваются или уменьшаются до суммы 100 %.

3. Число со звездочкой. Вообще говоря, числовое значение в данном описании является необязательным. Символ «*» указывает на то, что все оставшееся место будет принадлежать данному фрейму. Если указывается два или более фрейма с описанием «*» (например "*,*"), то оставшееся пространство делится поровну между этими фреймами. Если перед звездочкой стоит цифра, то она указывает пропорцию для данного фрейма (во сколько раз он будет больше аналогично описанного чистой звездочкой). Например, описание

"3*,*,*", говорит, что будет создано три фрейма с размерами 3/5 свободного пространства для первого фрейма и по 1/5 для двух других.

Примеры:

<FRAMESET COLS="50,*,50"> – описывает три фрейма, два по 50 точек справа и слева, и один внутри этих полосок.

<FRAMESET ROWS="20%,3*,*"> – описывает три фрейма, первый из которых занимает 20 % площади сверху экрана, второй 3/4 оставшегося от первого фрейма места (60 % всей площади окна), а последний – 1/4 (20 % всей площади окна).

<FRAMESET ROWS="*,60%,*"> – аналогично предыдущему примеру.

Тэги <FRAMESET> могут быть вложенными, т.е. например:

```
<FRAMESET ROWS="50%,50%">
 <FRAMESET COLS="*,*"
 </FRAMESET>
</FRAMESET>
```

Тэг FRAME описывает каждый фрейм в отдельности. Рассмотрим более детально каждый компонент.

<FRAME SRC="URL" NAME="frame_name" MARGINWIDTH="n1" MARGINHEIGHT="n2" SCROLLING=yes|no|auto NORESIZE>

SRC="URL" – описывает URL документа, который будет отображен внутри данного фрейма. Если он отсутствует, то будет отображен пустой фрейм. В качестве URL допустимо использование не только HTML-, но и GIF- или JPEG-файла.

NAME="frame_name" – описывает имя фрейма. Имя фрейма может быть использовано для определения действия с данным фреймом из другого HTML-документа или фрейма (как правило, из соседнего фрейма этого же документа). Имя обязательно должно начинаться с символа. Содержимое поименованных фреймов может быть задействовано из других документов при помощи специального атрибута TARGET, описываемого ниже.

MARGINWIDTH="n1" или MARGINHEIGHT="n2" – используются, если автор документа хочет указать величину вертикальных или горизонтальных разделительных полос между фреймами сбоку. Значения n1 и n2 указываются в пикселах и не могут быть меньше единицы.

SCROLLING="yes | no | auto" – позволяет задавать наличие полос прокрутки у фрейма. Параметр yes указывает на обязательное присутствие у фрейма полос прокрутки, параметр no – на отсутствие, auto – определяет наличие полос прокрутки только при их необходимости (значение по умолчанию).

NORESIZE. По умолчанию размер фрейма можно изменить при помощи мыши, так же как и размер окна Windows. NORESIZE отменяет данную возможность. Если у одного фрейма установлен атрибут NORESIZE, то у соседних фреймов тоже не может быть изменен размер со стороны данного.

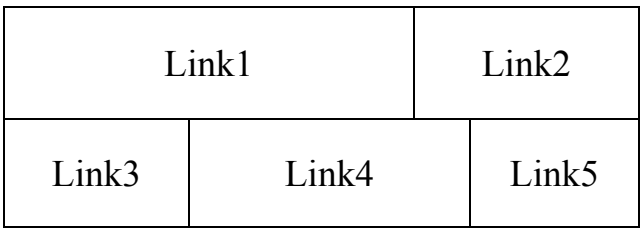
BORDER – управляет толщиной рамки.

BORDERCOLOR – задает цвет рамки.

FRAMEBORDER=no – отключает отображение рамки.

Контейнер NOFRAMES используется в случае, если вы создаете документ, который может просматриваться броузерами, как поддерживающими, так и не поддерживающими фреймы. Данный тэг помещается внутри контейнера FRAMESET, а все, что находится внутри тэгов <NOFRAMES> и </NOFRAMES>, игнорируется броузерами, поддерживающими фреймы.

Рассмотрим реализацию фреймов для подобного разбиения окна:




```
<FRAMESET ROWS="*,*">
 <NOFRAMES>
  <H1>Ваша версия Web-броузера не поддерживает фреймы!</H1>
 </NOFRAMES>
 <FRAMESET COLS="65%,35%">
  <FRAME SRC="link1.html">
  <FRAME SRC="link2.html">
 </FRAMESET>
 <FRAMESET COLS="*,40%,*">
  <FRAME SRC="link3.html">
  <FRAME SRC="link4.html">
  <FRAME SRC="link5.html">
 </FRAMESET>
</FRAMESET>
```

С появлением фреймов сразу возникает вопрос: «А как сделать так, чтобы, нажимая на ссылку в одном фрейме, инициировать появление информации в другом?»

Ответом на данный вопрос является планирование взаимодействия фреймов. Каждый фрейм может иметь собственное имя, определяемое параметром NAME при описании данного фрейма. Существует также специальный атрибут – TARGET, позволяющий определять, к какому фрейму относится та или иная операция. Формат данного атрибута следующий:

TARGET="windows_name"

Данный атрибут может встречаться внутри различных тэгов:

TARGET в тэге A. Это самое прямое использование TARGET. Обычно при активизации пользователем ссылки соответствующий документ появляется в том же окне (или фрейме), что и исходный, в котором была ссылка. Добавле-

ние атрибута TARGET позволяет произвести вывод документа в другой фрейм. Например:

<A HREF="mydoc.html" TARGET="Frame1"> Переход во фрейм №1 </A>

TARGET в тэге BASE. Размещение TARGET в тэге BASE позволит вам не указывать при описании каждой ссылки фрейм-приемник документов, вызываемых по ссылкам. Это очень удобно, если в одном фрейме у вас находится меню, а в другой выводится информация.

Пример:

Документ №1.

```
<FRAMESET ROWS="20,*">
 <FRAME SRC="doc2.htm" NAME="Frame1">
 <FRAME SRC="doc3.htm" NAME="Frame2">
</FRAMESET>
```

Документ №2 (doc2.htm).

```
<HTML>
 <HEAD>
  <BASE TARGET="Frame2">
 </HEAD>
 <BODY>
  <A HREF="url1"> Первая часть</A>
  <A HREF="url2"> Вторая часть</A>
 </BODY>
</HTML>
```

TARGET в тэге AREA. Также можно включать тэг TARGET в описание ссылки при создании карты изображения.

Пример:

```
    <AREA SHAPE="circle" COORDS="100,100,50"
HREF="http://www.softexpress.com" TARGET="Frame1">
```

TARGET в тэге FORM. То же относится и к определению формы. В данном случае после обработки переданных параметров формы результирующий документ появится в указанном фрейме.

<FORM ACTION="url" TARGET="window_name">

Зарезервированные имена фреймов служат для разрешения специальных ситуаций. Все они начинаются со знака подчеркивания. Любые другие имена фреймов, начинающиеся с подчеркивания, будут игнорироваться броузером.

TARGET="_blank". Данное значение определяет, что документ, полученный по ссылке, будет отображаться в новом окне броузера.

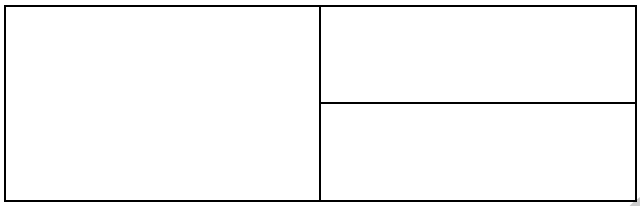
TARGET="_self". Данное значение определяет, что документ, полученный по ссылке, будет отображаться в том же фрейме, в котором находится ссылка. Это имя удобно для переопределения окна назначения, указанного ранее в тэге BASE.

TARGET="_parent". Данное значение определяет, что документ, полученный по ссылке, будет отображаться в родительском окне, вне зависимости от параметров FRAMESET. Если родительского окна нет, то данное имя аналогично "_self".

TARGET="_top". Данное значение определяет, что документ, полученный по ссылке, будет отображаться на всей поверхности окна, вне зависимости от наличия фреймов. Использование данного параметра удобно в случае вложенных фреймов.

Microsoft Internet Explorer поддерживает и так называемые плавающие фреймы. Они могут появляться в любом месте экрана. Текст, расположенный на главной странице, обтекает этот фрейм.

<IFRAME ALIGN=RIGHT WIDTH=50% HEIGHT=300
SRC="FILE.HTM">
</IFRAME>



Порядок выполнения работы

1. Вставить в документ произвольное графическое изображение.
2. Вставить с новой строки тот же графический файл, но вдвое меньшего размера.
3. Вставить с новой строки анимационный AVI-файл.
4. Включить в документ файл видеозаписи.
5. Задать указанный рисунок в виде фонового.
6. Вставить в документ бегущую строку шириной в пол-экрана. На синем фоне белыми буквами перемещается название факультета.
7. Вставить графическое изображение. Задать альтернативный текст. Рядом написать свою фамилию. Выполнить вертикальное выравнивание текста (прижать вверх, вниз, отцентрировать).
8. Создать с помощью программы «Блокнот» гипертекстовый файл. Имя файла – ваша фамилия. Файл разместить на указанном диске в папке с номером вашей группы.
9. Вставить в документ текстовую ссылку на графический файл.
10. Вставить графическую ссылку на гипертекстовый файл.
11. Создать таблицу вида (табл. 4).

Таблица 4

Текст, написанный гарнитурой Arial синим цветом		Текст, написанный гарнитурой Times красным цветом

Изображение импортировать из библиотеки рисунков MS WORD.

12. Создать таблицу вида (табл. 5).

Таблица 5

Файл 1	Файл 3
Файл 2	Файл 4
Изображение	

Надписи в таблице являются ссылками на файлы 1-4 соответственно, изображение – ссылкой на графический файл.

13. Создать список определений для двух понятий курса.
14. Произвольный элемент списка оформить как ссылку на гипертекстовый файл.
15. В конце документа разместить ссылку на начало (первую строку) гипертекстового файла.
16. Создать фреймовую структуру

Фрейм 1	Фрейм 2
Фрейм 3	

Фрейм 1	Фрейм 2
	Фрейм 3

Контрольные вопросы

1. Форматы графических файлов, используемых в Internet-документах.
2. Вставка графических объектов в HTML-файлы.
3. Использование динамических изображений и видеозаписей.
4. Вставка бегущей строки.
5. Создание списков различных видов средствами языка HTML.
6. Гипертекстовые связи между HTML-документами.
7. Ссылки на точки внутри документа.
8. Фреймы. Их назначение.
9. Синтаксис фреймов.
10. Планирование фреймов и взаимодействие между фреймами.
11. Зарезервированные имена фреймов.
12. Плавающие фреймы.

ЛАБОРАТОРНАЯ РАБОТА № 3

Общие сведения о JavaScript

***Цель работы**:* ознакомиться и получить общие сведения о языке составления скриптов JavaScript.

1. Основные понятия о JavaScript

JavaScript – язык для составления скриптов, разработанный фирмой Netscape. С помощью JavaScript Можно легко создавать интерактивные Web-страницы. Для запуска скриптов, написанных на языке JavaScript, понадобится броузер, способный работать с JavaScript – например, Netscape Navigator (начиная с версии 2.0) или Microsoft Internet Explorer (MSIE – начиная с версии 3.0).

2. Размещение JavaScript на HTML-странице

Код скрипта JavaScript размещается непосредственно на HTML-странице. Чтобы увидеть, как это делается, рассмотрим следующий пример:

```
<html>
<body>
<br>
Это обычный HTML документ.
<br>
  <script language=”JavaScript”>
    document.write(“А это JavaScript!”)
  </script>
<br>
Вновь документ HTML.
</body>
</html>
```

С первого взгляда пример напоминает обычный файл HTML. Единственное новшество здесь – конструкция:

```
  <script language=»JavaScript»>
    document.write(“А это JavaScript!”)
  </script>
```

Чтобы видеть, как этот скрипт работает, запишите данный пример как обычный файл HTML и загрузите его в броузер, имеющий поддержку языка JavaScript.

Результат выполнения этого файла (если используется броузер, имеющий поддержку JavaScript, то на экране будет 3 строки):

Это обычный HTML документ.
А это JavaScript! А это JavaScript!
Вновь документ HTML.

Данный пример демонстрирует тэг признака <script>. Все, что стоит между тэгами <script> и </script>, интерпретируется как код на языке JavaScript. Здесь также демонстрируется пример использования инструкции *document.write()* – одной из наиболее важных команд, используемых при программировании на языке JavaScript. Команда *document.write()* используется, когда необходимо что-либо написать в текущем документе (в данном случае таковым является HTML-документ). Таким образом, программа на JavaScript в HTML-документе пишет фразу *«А это JavaScript!»*.

3. Броузеры без поддержки JavaScript

Броузеры, не имеющие поддержки JavaScript, «не знают» и тэга <script>. Они игнорируют его и печатают все стоящие вслед за ним коды как обычный текст. Иными словами, код JavaScript, приведенный в программе, окажется вписан открытым текстом прямо посреди HTML-документа. На этот случай имеется специальный способ скрыть исходный код скрипта от старых версий броузеров – будем использовать для этого тэг комментария из HTML - <!-- →. В результате новый вариант исходного кода будет выглядеть следующим образом:

```
<html>
<body>
<br>
Это обычный HTML документ.
<br>
 <script language=”JavaScript”>
 <!–hide from old browsers
   document.write(“А это JavaScript!”)
 // →
 </script>
<br>
Вновь документ HTML.
</body>
</html>
```

В этом случае броузер без поддержки JavaScript будет печатать:

Это обычный HTML документ.
Вновь документ HTML.

Без использования HTML-тэга комментария броузер без поддержки JavaScript напечатал бы:

Это обычный HTML документ.
Document.write(«А это JavaScript!»)
Вновь документ HTML.

Обратите внимание, что невозможно полностью скрыть исходный код JavaScript. Приведенный пример имеет целью предотвратить распечатку кода скрипта на старых броузерах – тем не менее посетитель сможет увидеть код посредством пункта меню «View document source». Не существует также способа скрыть что-либо от просмотра в исходном коде (и увидеть, как выполнен тот или иной трюк).

4. События

События и обработчики событий являются очень важной частью для программирования на языке JavaScript. События, главным образом, инициируются теми или иными действиями посетителя. Если он щелкает по кнопке, происходит событие *«Click»*. Если указатель мыши пересекает какую-либо ссылку гипертекста – происходит событие *MouseOver*. Существует несколько различных типов событий. Можно заставить JavaScript-программу реагировать на некоторые из них. И это может быть выполнено с помощью специальных программ обработки событий. Так, в результате щелчка по кнопке может создаваться выпадающее окно. Это означает, что создание окна должно быть реакцией на событие щелчка – *Click*. Программа – обработчик событий, которая используется в данном случае, называется *onClick*. Приведенный ниже код представляет простой пример программы обработки события *onClick*:

<form>
<input type= «button» value= «Проверьте этот пример» onClick= «alert('Это текст')»>
</form>

Данный пример имеет несколько новых особенностей – рассмотрим их по порядку. Создается форма с кнопкой. Первая новая особенность – onClick= «alert('Это тест')» в тэге <input>. Этот атрибут определяет, что происходит, когда нажимается кнопка. Таким образом, если имеет место событие *Click*, выполняется вызов alert('Это тест'). Это и есть пример кода на языке JavaScript (обратите внимание, что в этом случае не используется тэг <script>).Функция *alert()* позволяет создавать выпадающие окна. При ее вызове необходимо в скобках задать некую строку. В нашем случае это «*Это тест»*. И это как раз будет тот текст, что появится в выпадающем окне. Таким образом, когда посетитель нажимает кнопку, скрипт создает окно, содержащее текст *«Это тест»*. Обратите внимание, что в команде document.write()использовались двойные кавычки («»), а в конструкции alert() – только одинарные. В большинстве случаев можно использовать оба типа кавы-

чек. Однако в последнем примере написано onClick= «alert('Это тест')»– то есть использовались и двойные, и одинарные кавычки. Если бы мы написали onClick= «alert(«Это тест»)», то становится неясно, к которой из частей конструкции имеет отношение функция обработки событий onClick, а к которой – нет. Поэтому необходимо в данном случае сочетать оба типа кавычек. Не имеет значения, в каком порядке использовались кавычки – сначала двойные, а затем одинарные или наоборот. То есть можно точно так же написать и onClick='alert(«Это тест»)'.

Можно использовать в скрипте множество различных типов функций обработки событий.

К базовым событиям, поддерживаемым в JavaScript, относятся (табл. 6):

Таблица 6

События	Описание
OnAbort	Происходит при прерывании загрузки графического изображения
OnBlur	Происходит, когда какой-либо элемент теряет фокус
OnChange	Происходит при изменении значения текстового поля
OnClick	Происходит при нажатии кнопки мыши в области элемента
OnError	Происходит при ошибке загрузки документа или графического изображения
OnFocus	Происходит при получении элементом фокуса
OnLoad	Происходит по завершении загрузки страниц или графического изображения
onMouseOver	Происходит при перемещении курсора мыши в области элемента
onMouseOut	Происходит при перемещении курсора мыши в области элемента
OnReset	Происходит при нажатии кнопки типа Reset
OnSelect	Происходит при выборе текста в текстовом поле.
OnSubmit	Происходит при нажатии кнопки типа Submit
OnUnload	Происходит при переходе на другую страницу или при завершении работы с броузером

Рассмотрим несколько примеров, демонстрирующих работу с этими событиями.

onAbort

```
<script language="JavaScript">
function doAbort() {
alert("Загрузка изображения прервана")
}
</script>
<body>
<img src="Зима.jpg onAbort="doAbort()"">
</body>
```

Такой обработчик может быть необходим в тех случаях, когда изображение, используемое в качестве изображения-карты (image-map), имеет большой размер, и пользователь не дождался его полной загрузки – нажал кнопку «стоп». В этом случае функциональность карты будет нарушена и пользователь не сможет перемещаться по узлу.

onLoad

```
<script language="JavaScript">
function doLoad() {
alert("Загрузка изображения завершена")
}
</script>
<body>
<img src=" Зима.jpg  onLoad="doLoad()"">
</body>
```

Событие onLoad происходит после завершения загрузки документа или графического изображения.

onMouseOver и onMouseOut

```
<script language="JavaScript">
function doMouseOver() {
alert("Указатель на ссылке")
}
function doMouseOut()
{
alert("Укакзатель за пределами ссылки")
}
</script>
<body>
<a href="new.html"
```

```
    onMouseOver="doMouseOver()"
onMouseOut="doMouseOut()">
Оглавление</a>
</body>
```

Итак, если используется броузер Netscape Navigator, то выпадающее окно содержит текст, что был передан функции *JavaScript alert*. Такое ограничение накладывается по соображениям безопасности. Такое же выпадающее окно можно создать и с помощью метода prompt(). Однако в этом случае окно будет воспроизводить текст, введенный пользователем. А потому скрипт, написанный злоумышленником, может принять вид системного сообщения и попросить пользователя ввести некий пароль. Если текст помещается в выпадающее окно, то тем самым пользователю дается понять, что данное окно было создано Web-броузером, а не вашей операционной системой. И поскольку данное ограничение наложено по соображениям безопасности, нельзя просто так удалить появившееся сообщение.

5. Функции

Основная часть программ, написанных на языке JavaScript, содержит функции. В большинстве случаев функции представляют собой лишь способ связать вместе нескольких команд. Рассмотрим скрипт, печатающий текст, три раза подряд. Рассмотрим простой подход:

```
<html>
<script language="JavaScript">
<!-- hide

document.write("Добро пожаловать на мою страницу!<br>")
document.write("Это JavaScript!<br>")

document.write("Добро пожаловать на мою страницу!<br>")
document.write("Это JavaScript!<br>")

document.write("Добро пожаловать на мою страницу!<br>")
document.write("Это JavaScript!<br>")
// -->
</script>
</html>
```

Скрипт напишет следующий текст:

Добро пожаловать на мою страницу!
Это JavaScript!

Если посмотреть на исходный код скрипта, то видно, что для получения необходимого результата определенная часть его кода была повторена три раза. Можно решить ту же задачу еще лучше. Пример скрипта для решения той же самой задачи:

```
<html>
<script language="JavaScript">
<!-- hide

function myFunction() {
  document.write("Добро пожаловать на мою страницу!<br>")
  document.write("Это JavaScript!<br>")
}

myFunction()
myFunction()
myFunction()

// -->
</script>
</html>
```

В этом скрипте определили некую функцию, состоящую из следующих строк:

```
function myFunction() {
  document.write("Добро пожаловать на мою страницу!<br>")
  document.write("Это JavaScript!<br>");
}
```

Все команды скрипта, что находятся внутри фигурных скобок – {} – принадлежат функции *myFunction()*. Это означает, что обе команды document.write() теперь связаны воедино и могут быть выполнены при вызове указанной функции. И действительно, в нашем примере есть три вызова этой функции – можно увидеть, что мы написали строку *myFunction()* три раза сразу после того, как дали определение самой функции. То есть как раз и сделали три вызова. Это означает, что содержимое этой функции (команды, указанные в фигурных скобках) было выполнено трижды. Функции могут также использоваться совместно с процедурами обработки событий.

Пример:

```
<html>
<head>
```

```
<script language="JavaScript">
<!-- hide
function calculation() {
  var x= 12
  var y= 5
  var result= x + y

  alert(result)
}
// -->
</script>

</head>
<body>

<form>
<input type="button" value="Вычислить" onClick="calculation()">
</form>

</body>
</html>
```

При нажатии кнопки осуществляется вызов функции *calculation()*. Как можно заметить, эта функция выполняет некие вычисления, пользуясь переменными *x*, *y* и *result*. Переменные определяются при помощи ключевого слова *var*. Переменные могут использоваться для хранения различных величин – чисел, строк текста и т.д. Так строка скрипта *var result= x + y* сообщает броузеру о том, что необходимо создать переменную *result* и поместить туда результат выполнения арифметической операции *x + y* (т.е. 5 + 12). После этого в переменную *result* будет размещено число 17. В данном случае команда *alert(result)* выполняет то же самое, что и *alert(17)*. Иными словами, получаем выпадающее окно, в котором написано число 17.

Порядок выполнения работы

1. Создать с помощью программы "Блокнот" гипертекстовый файл. Имя файла – ваша фамилия. Файл разместить на указанном диске в папке с номером вашей группы.

2. Набрать скриптовые программы, приведенные в методических указаниях к выполнению данной лабораторной работы.

3. Объявите несколько переменных и присвойте им значения методом prompt(). Подсчитайте значение факториала этих переменных и выведите эти значения в окно методом alert().

4. Создать функцию, которая выдаёт прощальное сообщение по событию onUnload.

Контрольные вопросы

1. Размещение JavaScript на HTML-странице.
2. События и обработчики событий.
3. Функции.

ЛАБОРАТОРНАЯ РАБОТА № 4

Иерархия объектов в JavaScript

Цель работы: изучить структуру объектов в JavaScript

1. Создание формы

Для задания формы на Web-странице служит тег <FORM>. Если не рассматривать обработку данных формы на стороне сервера, то можно рассматривать три параметра формы name, onreset и onsubmit. Атрибут name необходим для того, чтобы различать формы на странице. Параметры onreset и onsubmit ссылаются на функцию обработчик; onreset – при нажатии кнопки reset, onsubmit – при нажатии кнопки Submit.

Согласно описанию стандарта HTML 4, на форме можно разместить кнопки трёх видов: button, reset и submit. Тип submit соответствует стандартной кнопке диалогового окна OK, a reset – CANCEL. Тип button – нейтральный тип. Кнопки всех видов имеют одинаковые атрибуты: onclick – функция обработчик события onclick, value – значение кнопки (оно будет на ней написано), name – имя кнопки (по которому происходит обращение к кнопке). Итак, тег, задающий кнопку, будет выглядеть следующим образом:

```
<INPUT type=”submit” value=”эта кнопка OK” name=”кнопка1”>
<INPUT type=”reset” value=”эта кнопка Cancel” name=”кнопка2”>
<html>
<body >
<script language="javascript">
<!--
var butt="red";
function change_color(butt){
document.body.bgcolor=butt;
}
//-->
</script>
<form name="forma">
```

```
<INPUT type="reset" value="КРАСНЫЙ" name="RED" ON-
CLICK="change_color(this.name)" >
<INPUT type="reset" value="Голубой" name="Cyan" on-
Click="change_color(this.name)">
</form>
</body>
</html>
```

При помощи типа IMAGE можно создать графическую кнопку.
Например:

```
<html>
<body >
<form name="forma">
<INPUT type="image" value="Image" src="Зима.gif" width=50 higth=40>
</form>
</body>
</html>
```

В языке JavaScript все элементы на Web-странице выстраиваются в иерархическую структуру. Каждый элемент предстает в виде объекта. И каждый такой объект может иметь определенные свойства и методы. То есть существует иерархия объектов, на которые опирается разметка HTML. Рассмотрим простую HTML-страницу:

```
<html>
<head>
<title>Пример</title>
</head>
<body background="Зима.gif" TEXT="#000000" LINK="#B50000"
VLINK="#800080" ALINK="#800080">

<center>
<img src="mmm.gif" name="pic1">
</center>

<p>
<form name="myForm">
Name:
<input type="text" name="name" value="Вова"><br>
e-Mail:
<input type="text" name="email" value="test@test.com"><br><br>
<input type="button" value="Нажми меня" name="myButton" on-
Click="alert('Привет')">
</form>
```

```
</p>

<center>
<p>
<img src="lll.gif" name="pic2" width=700 height=3>
</p>

<p>
<a href="clock.html">Пример</a>
</p>
</center>

</body>
</html>
```

Итак, имеем два рисунка, одну ссылку и форму с двумя полями для ввода текста и одной кнопкой. С точки зрения языка JavaScript окно броузера – это некий объект *window*. Этот объект также содержит в свою очередь некоторые элементы оформления, такие как строка состоятия. Внутри окна можно разместить документ HTML. Такая страница является не чем иным, как объектом *document*. Это означает, что объект *document* представляет в языке JavaScript загруженный на настоящий момент документ HTML. К свойствам объекта *document* относятся, например, цвет фона для Web-страницы. Однако гораздо важнее то, что все без исключения объекты HTML являются свойствами объекта *document*. Примерами объекта HTML являются, например, ссылка или заполняемая форма. Разумеется, нужно иметь возможность получать информацию о различных объектах в этой иерархии и управлять ею. Для этого необходимо знать, как в языке JavaScript организован доступ к различным объектам. Как видно, каждый объект иерархической структуры имеет свое имя. Следовательно, если необходимо узнать, как можно обратиться к первому рисунку на нашей HTML-странице, нужно сориентироваться в иерархии объектов. Первый объект такой структуры называется *document*. Первый рисунок на странице представлен как объект *images[0]*. Это означает, можно получать доступ к этому объекту, записав в JavaScript *document.images[0]*. Если же, например, необходимо знать, какой текст ввел пользователь в первый элемент формы, то сначала необходимо выяснить, как получить доступ к этому объекту. И снова начинаем с вершины иерархии объектов. Затем прослеживаем путь к объекту с именем *elements[0]* и последовательно записываем названия всех объектов, которые минуем. В итоге выясняется, что доступ к первому полю для ввода текста можно получить, записав: *document.forms[0].elements[0]* Для того чтобы определить содержимое поля формы, необходимо воспользоваться свойством *value*, которое как раз и соответствует введенному тексту. Итак, чтобы прочитать искомое значение, нужно написать на языке JavaScript строку

name= document.forms[0].elements[0].value;

Полученная строка заносится в переменную *name*. Следовательно, теперь можно работать с этой переменной. Например, можно создать выпадающее окно, воспользовавшись командой *alert("Привет" + name)*. В результате, если пользователь введет в это поле слово «*Вова»*, то по команде *alert("Привет " + name)* будет открыто выпадающее окно с приветствием «*Привет Вова»*. Если работа ведется с большими страницами, то процедура адресации к различным объектам по номеру может стать весьма запутанной. Существует возможность присваивать различным объектам уникальные имена. Как это делается, можно увидеть в примере:

```
<form name="myForm">
Name:
<input type="text" name="name" value="Вова"><br>
...
```

Эта запись означает, что объект *forms[0]* получает теперь еще и второе имя - *myForm*. Точно так же вместо *elements[0]* можно писать *name* (последнее было указано в атрибуте name тэга <input>). Таким образом, вместо

name= document.forms[0].elements[0].value;

Можно записать

name= document.myForm.name.value;

Это значительно упрощает программирование на JavaScript, особенно в случае с большими Web-страницами, содержащими множество объектов. (Обратите внимание, что при написании имен необходимо еще следить и за положением регистра – т.е. нельзя написать *myform* вместо *myForm*) В JavaScript многие свойства объектов доступны не только для чтения. Возможно записывать в них новые значения.

Пример кода на JavaScript, иллюстрирующего такую возможность – интересующий фрагмент записан как свойство onClick второго тэга <input>:

```
<form name="myForm">
<input type="text" name="input" value="Привет!">
<input type="button" value="Нажмите меня,плизззззззз..."
  onClick="document.myForm.input.value= 'Спасибо! :-)'; ">
```

Теперь рассмотрим более сложный пример использования различных объектов. Попытайтесь разобрать его самостоятельно, для облегчения этой задачи в код скрипта введены комментарии.

Исходный код скрипта:

```
<html>
<head>
<title>Объекты</title>

<script language="JavaScript">
<!-- hide

function first() {

// создает выпадающее окно, где размещается
// текст, введенный в поле формы
  alert("Значение элемента text: " +
    document.myForm.myText.value);
}

function second() {

// данная функция проверяет состояние переключателей

  var myString= "Переключатель ";

// переключатель включен, или нет?

  if (document.myForm.myCheckbox.checked) myString+= "включен"
    else myString+= "не включен";

// вывод строки на экран
  alert(myString);
}

// -->
</script>
</head>
<body bgcolor=lightblue>

<form name="myForm">
<input type="text" name="myText" value="Привет">
<input type="button" name="button1" value="Элемент text"
```

```
  onClick="first()">
<br>
<input type="checkbox" name="myCheckbox" CHECKED>
<input type="button" name="button2" value="Переключатель"
  onClick="second()">
</form>

<br>
<br>

<script language="JavaScript">
<!-- hide

document.write("Цвет фона: ");
document.write(document.bgColor + "<br>");

document.write("На второй кнопке написано: ");
document.write(document.myForm.button2.value);

// -->
</script>

</body>
</html>
```

Кроме объектов *window* и *document* в JavaScript имеется еще один важный объект – *location*. В этом объекте представлен адрес загруженного HTML-документа. Например, если загружена страница *index.html*, то значение *location.href* как раз и будет соответствовать этому адресу. В данном примере кнопка загружает в текущее окно новую страницу:

```
<form>
<input type=button value="К оглавлению"
  onClick="location.href='index.htm'">
</form>
```

2. Проверка информации, введенной в форму

Формы широко используются в Интернет. Информация, введенная в форму, часто посылается обратно на сервер или отправляется по электронной почте на некоторый адрес. Проблема состоит в том, чтобы убедиться, что введенная пользователем в форму информация корректна. Легко проверить ее перед пересылкой в Интернет с помощью языка JavaScript.

Создадим простой скрипт. Допустим, HTML-страница содержит два элемента для ввода текста. В первый из них пользователь должен вписать свое имя, во второй элемент – адрес для электронной почты. Можно ввести туда какую-нибудь информацию и нажать клавишу. Попробуйте также нажать клавишу, не введя в форму никакой информации.

Введите Ваше имя:

Введите Ваш адрес e-mail:

Что касается информации, введенной в первый элемент, то выведется сообщение об ошибке, если туда ничего не было введено. Любая представленная в элементе информация будет рассматриваться на предмет корректности. Это не гарантирует, что пользователь введет не то имя. Броузер даже не будет «возражать» против чисел. Например, если ввести число 17, то получите приглашение «Hi 17!». Так что эта проверка не может быть идеальна. Второй элемент формы более сложный. Попробуйте ввести простую строку – например, ваше имя. Сделать это не удастся до тех пор, пока вы не укажите @ в вашем имени... Признаком того, что пользователь правильно ввел адрес электронной почты, служит наличие символа @.

```
<html>
<head>
<script language="JavaScript">
<!-- Скрыть

function test1(form) {
  if (form.text1.value == "")
    alert("Пожалуйста, введите строку!")
  else {
   alert("Hi "+form.text1.value+"! Форма заполнена корректно!");
  }
}

function test2(form) {
  if (form.text2.value == "" ||
      form.text2.value.indexOf('@', 0) == -1)
        alert("Неверно введен адрес e-mail!");
  else alert("OK!");
}
// -->
</script>
</head>
```

```
<body>
<form name="first">
Введите Ваше имя:<br>
<input type="text" name="text1">
<input type="button" name="button1" value="Проверка" onClick="test1(this.form)">
<P>
Введите Ваш адрес e-mail:<br>
<input type="text" name="text2">
<input type="button" name="button2" value="Проверка" onClick="test2(this.form)">
</body>
</html>
```

Рассмотрим сначала HTML-код в разделе body. Здесь создаются два элемента для ввода текста и две кнопки. Кнопки вызывают функции test1(...) или test2(...), в зависимости от того, которая из них была нажата. В качестве аргумента к этим функциям мы передаем комбинацию *this.form*, что позже позволит адресоваться в самой функции именно к тем элементам, которые необходимы. Функция test1(form) проверяет, является ли данная строка пустой. Это делается посредством *if (form.text1.value == "")...* . Здесь *form* – это переменная, куда заносится значение, полученное при вызове функции от *this.form*. Можно извлечь строку, введенную в рассматриваемый элемент, если к *form.text1* припишем «value». Чтобы убедиться, что строка не является пустой, мы сравниваем ее с "". Если же окажется, что введенная строка соответствует "", то это значит, что на самом деле ничего введено не было. И пользователь получит сообщение об ошибке. Если же строка была введена верно, пользователь получит подтверждение - ok. Рассмотрим функцию *test2(form)*. Здесь вновь сравнивается введенная строка с пустой – "" (чтобы удостовериться, что что-то действительно было введено читателем. Комбинация символов || является оператором OR (ИЛИ). Команда if проверяет, чем заканчивается первое или второе сравнения. Если хотя бы одно из них выполняется, то и в целом команда if имеет результатом true, а стало быть, будет выполняться следующая команда скрипта. Таким образом, сообщение об ошибке будет получено, если либо предоставленная строка пуста, либо в ней отсутствует символ @. (Второй оператор в команде if следит за тем, чтобы введенная строка содержала @.).

3. Проверка на присутствие определенных символов

В некоторых случаях понадобится ограничивать информацию, вводимую в форму, лишь некоторым набором символов или чисел. Достаточно вспомнить о телефонных номерах – представленная информация должна содержать лишь цифры (предполагается, что номер телефона как таковой не содержит никаких

символов). Необходимо проверять, являются ли введенные данные числом. Сложность ситуации состоит в том, что большинство людей вставляют в номер телефона еще и разные символы, например: 01234-56789, 01234/56789 or 01234 56789 (с символом пробела внутри). Не следует принуждать пользователя отказываться от таких символов в телефонном номере. А потому необходимо дополнить скрипт процедурой проверки цифр и некоторых символов.

Исходный код этого скрипта:

```
<html>
<head>
<script language="JavaScript">
<!-- hide


function check(input) {
  var ok = true;

  for (var i = 0; i < input.length; i++) {
    var chr = input.charAt(i);
    var found = false;
    for (var j = 1; j < check.length; j++) {
      if (chr == check[j]) found = true;
    }
    if (!found) ok = false;
  }

  return ok;
}

function test(input) {

  if (!check(input, "1", "2", "3", "4",
        "5", "6", "7", "8", "9", "0", "/", "-", " ")) {

    alert("Input not ok.");
  }
  else {
    alert("Input ok!");
  }
}

// -->
</script>
```

```
</head>

<body>
<form>
Telephone:
<input type="text" name="telephone" value="">
<input type="button" value="Check"
  onClick="test(this.form.telephone.value)">
</form>
</body>
</html>
```

Функция *test()* определяет, какие из введенных символов признаются корректными.

4. Предоставление информации, введенной в форму

Самый простой способ для передачи информации, внесенной в форму, состоит в передаче данных формы по электронной почте (этот метод рассмотрим подробнее).

```
<form method=post action="mailto:your.address@goes.here" enctype="text/plain">

Нравится ли Вам эта страница?
  <input name="choice" type="radio" value="1">Вовсе нет.<br>
  <input name="choice" type="radio" value="2" CHECKED>Напрасная трата времени.<br>
  <input name="choice" type="radio" value="3">Самый плохой сайт в Сети.<br>
  <input name="submit" type="submit" value="Send">
</form>
```

Параметр *enctype="text/plain"* используется для того, чтобы пересылать именно простой текст без каких-либо кодируемых частей. Это значительно упрощает чтение такой почты.

Если необходимо проверить форму прежде, чем она будет передана в сеть, то для этого можно воспользоваться программой обработки событий *onSubmit*. Поместите вызов этой программы в тэг <form>.

Пример:

```
function validate() {
  // check if input ok
  // ...

  if (inputOK) return true
    else return false;
}
...
<form ... onSubmit="return validate()">
...
```

Форма, составленная таким образом, не будет послана в Интернет, если в нее внесены некорректные данные.

5. Выделение определенного элемента формы

С помощью метода *focus()* можно сделать форму более дружественной. Так, можно выбрать, какой элемент будет выделен в первую очередь, т.е. броузер сам установит курсор на указанный элемент формы, так что пользователю не придется щелкать по форме, прежде чем что-либо занести туда. Сделать это можно с помощью следующего фрагмента скрипта:

```
function setfocus() {
  document.first.text1.focus();
}
```

Эта запись выделяет первый элемент для ввода текста в скрипте. Необходимо указать имя для всей формы – в данном случае она называется *first* – и имя одного элемента формы – *text1*. Если нужно чтобы при загрузке страницы данный элемент выделялся, то для этого можно дополнить тэг <body> атрибутом onLoad. Это будет выглядеть как

```
<body onLoad="setfocus()">
```

Остается еще дополнить пример следующим образом:

```
function setfocus() {
  document.first.text1.focus();
  document.first.text1.select();
}
```

Порядок выполнения работы

1. Создать с помощью программы "Блокнот" гипертекстовый файл. Имя файла – ваша фамилия. Файл разместить на указанном диске в папке с номером вашей группы.

2. Набрать скриптовые программы, приведенные в методических указаниях к выполнению данной лабораторной работы.

3. Вставьте на свою страницу проверку пароля доступа, который бы выделялся при загрузке страницы.

Контрольные вопросы

1. Объект *window* в JavaScript.
2. Объект *document* в JavaScript.
3. Объект *location* в JavaScript.

ЛАБОРАТОРНАЯ РАБОТА № 5
Создание меню средствами JavaScript.Окна и динамическое управление документами

Цель работы: ознакомиться и получить общие сведения по совместному использованию фреймов и JavaScript; изучить возможности JavaScript, связанные с окнами и динамическим управлением документами.

1. Создание фреймов

В общем случае окно броузера может быть разбито на несколько отдельных фреймов. Это означает, что фрейм определяется как некое выделенное в окне броузера поле в форме прямоугольника. Каждый из фреймов выдает на экран содержимое собственного документа (в большинстве случаев это документы HTML). Таким образом, можно, к примеру, создать два фрейма. В первый такой фрейм загружается "домашняя страница" фирмы Netscape, а во второй – фирмы Microsoft. Хотя создание фреймов является задачей языка HTML, все же вспомните основные моменты этого процесса. Для создания фреймов необходимо два тэга: <frameset> и <frame>. HTML-страница, создающая два фрейма, в общем случае может выглядеть следующим образом:

```
<html>
<frameset rows="50%,50%">
  <frame src="page1.htm" name="frame1">
  <frame src="page2.htm" name="frame2">
</frameset>
</html>
```

При создании Web-страниц можно использовать несколько вложенных тэгов <frameset>. Следующий пример демонстрирует это:

```
<frameset cols="50%,50%">
  <frameset rows="50%,50%">
    <frame src="page3.htm">
    <frame src="page3.htm">
  </frameset>
  <frameset rows="33%,33%,33%">
    <frame src="page3.htm">
    <frame src="page3.htm">
    <frame src="page3.htm">
  </frameset>
</frameset>
```

2. Фреймы в JavaScript

JavaScript организует все элементы, представленные на Web-странице, в виде некой иерархической структуры. То же самое относится и к фреймам.

В вершине иерархии находится окно броузера (browser window). В данном случае оно разбито на два фрейма. Таким образом, окно, как объект, является родоначальником, родителем данной иерархии (parent), а два фрейма – соответственно его потомки (children). Присвоим этим двум фреймам уникальные имена - *frame1* и *frame2*. С помощью этих имен будем обмениваться информацией с двумя указанными фреймами. С помощью скрипта можно решить следующую задачу: допустим, посетитель активирует некую ссылку в первом фрейме, однако соответствующая страница должна загружаться не в этот же фрейм, а в другой. Примером такой задачи может служить составление меню (или навигационных панелей), где один фрейм всегда остается неизменным, но предлагает посетителю несколько различных ссылок для дальнейшего изучения данного сайта. Чтобы решить эту задачу, рассмотрим три случая:

- главное окно/фрейм получает доступ к фрейму-потомку
- фрейм-потомок получает доступ к родительскому окну/фрейму
- фрейм-потомок получает доступ к другому фрейму-потомку

С точки зрения объекта «окно» (window) два указанных фрейма называются *frame1* и *frame2*. Существует прямая взаимосвязь между родительским окном и каждым фреймом. Таким образом, если пишется скрипт для родительского окна – т.е. для страницы, создающей эти фреймы, то можно обращаться к этим фреймам, просто называя их по имени.

Пример:

frame2.document.write("Это сообщение передано от родительского окна".);

В некоторых случаях существует необходимость, находясь во фрейме, получать доступу к родительскому окну. Например, это бывает необходимо, если нужно при следующем переходе избавиться от фреймов. В таком случае удаление фреймов означает лишь загрузку новой страницы вместо содержавшей фреймы. В нашем случае это загрузка страницы в родительское окно. Сделать это поможет доступ к родительскому- *parent* - окну (или родительскому фрейму) из фреймов, являющихся его потомками. Чтобы загрузить новый документ, необходимо внести в *location.href* новый адрес URL. Если избавляться от фреймов не нужно, то следует использовать объект location из родительского окна. (Напомним, что в каждый фрейм можно загрузить собственную страницу, таким образом, для каждого фрейма существует собственный объект location). Итак, можно загрузить новую страницу в родительское окно с помощью команды

parent.location.href= "http://...";

И, наконец, очень часто приходится решать задачу обеспечения доступа с одного фрейма-потомка к другому такому же фрейму-потомку. Итак, как можно, находясь в первом фрейме, записать что-либо во второй – т.е., какой командой следует воспользоваться на HTML-странице *page1.htm*? Невозможно просто так вызвать *frame2*, находясь в фрейме *frame1*, который просто ничего не знает о существовании второго фрейма. С точки же зрения родительского окна второй фрейм действительно существует и называется *frame2*, а к самому родительскому окну можно обратиться из первого фрейма по имени *parent*. Таким образом, чтобы получить доступ к объекту *document*, разместившемуся во втором фрейме, нужно написать следующее:

parent.frame2.document.write ("Привет, это вызов из первого фрейма".);

3. Навигационные панели

Рассмотрим, как создаются навигационные панели. В одном фрейме создается несколько ссылок. Однако, если посетитель активирует какую-либо из них, соответствующая страница будет помещена не в тот же самый фрейм, а в соседний.

Пример: Сначала нам необходимо написать скрипт, создающий указанные фреймы. Такой документ выглядит точно так же, как и тот, что был рассмотрен ранее в этой лабораторной работе:

frames3.htm

```
<html>
<frameset rows="80%,20%">
  <frame src="start.htm" name="main">
  <frame src="menu.htm" name="menu">
</frameset>
</html>
```

Здесь *start.htm* - это та страница, которая первоначально будет показана в главном фрейме (*main*). Следующая Web-страница будет загружена во фрейм "*menu*":

menu.htm

```
<html>
<head>
<script language="JavaScript">
<!-- hide

function load(url) {
  parent.main.location.href= url;
}

// -->
</script>
</head>
<body>

<a href="javascript:load('first.html')">Первая</a>
<a href="second.html" target="main">Вторая</a>
<a href="third.html" target="_top">Третья</a>

</body>
</html>
```

Здесь можно увидеть несколько способов загрузки новой страницы во фрейм *main*. В первой ссылке для этой цели используется функция *load()*. Разберем, как это делается:

```
<a href="javascript:load('first.html')">first</a>
```

Вместо явной загрузки новой страницы броузер выполняет команду на языке JavaScript – для этого воспользуемся параметром *javascript:* вместо

обычного *href*. Далее, внутри скобок можно увидеть *'first.html'*. Эта строка передается в качестве аргумента функции *load()*. Сама же функция *load()* определяется следующим образом:

```
function load(url) {
  parent.main.location.href= url;
}
```

Параметр *ur* означает, что в примере строка 'first1.htm' при вызове функции заносится в переменную *url*. И эту новую переменную теперь можно использовать при работе внутри функций *load()*. Во второй ссылке присутствует параметр *target*. Это уже не относится к JavaScript. Это одна из конструкций языка HTML. Заметим, что в этом случае необязательно ставить перед именем указанного фрейма слово *parent*. Причина такого отступления от правил кроется в том, что параметр *target* – это функция языка HTML, а не JavaScript.

Параметр *target* использовать очень просто. Можно воспользоваться им, если необходимо всего лишь загрузить новую страницу в другой фрейм. Решение на основе языка JavaScript (примером этого служит первая ссылка) обычно используется, если нужно, чтобы при активизации ссылки произошло несколько вещей. Одна из наиболее часто возникающих проблем такого рода состоит в том, чтобы разом загрузить две страницы в два различных фрейма. И хотя можно решить эту задачи с помощью параметра *target*, использование функции JavaScript в этом случае более предпочтительно. Предположим, что имеется три фрейма с именами *frame1*, *frame2* и *frame3*. Допустим, посетитель активирует ссылку в *frame1*. И необходимо, чтобы при этом в два других фрейма загружались две различные Web-страницы. В качестве решения этой задачи можно, например, воспользоваться функцией:

```
function loadtwo() {
  parent.frame1.location.href= "first.htm";
  parent.frame2.location.href= "second.htm";
}
```

Если же нужно сделать функцию более гибкой, то можно воспользоваться возможностью передачи переменной в качестве аргумента. Результат будет выглядеть как

```
function loadtwo(url1, url2) {
  parent.frame1.location.href= url1;
  parent.frame2.location.href= url2;
}
```

После этого можно организовать вызов функции: loadtwo("first.htm", "second.htm") или loadtwo("third.htm", "forth.htm"). Очевидно, передача аргу-

ментов делает функцию более гибкой. В результате можно использовать ее многократно и в различных контекстах.

4. Создание окон

Открытие новых окон в броузере – грандиозная возможность языка JavaScript. Можно либо загружать в новое окно новые документы (например, те же документы HTML), либо (динамически) *создавать* новые материалы. Приведеный далее скрипт открывает новое окно броузера и загружает в него Web-страничку:

```
<html>
<head>

<script language="JavaScript">
<!-- hide

function openWin() {
  myWin= open("page4.htm");
}

// -->
</script>

</head>
<body>

<form>
<input type="button" value="Открыть новое окно" onClick="openWin()">
</form>

</body>
</html>
```

(Открыть новое окно)

В представленном примере в новое окно с помощью метода *open()* записывается страница *page4.htm*.

Заметим, что можно управлять самим процессом создания окна. Например, можно указать, будет иметь ли новое окно строку статуса, панель инструментов или меню. Кроме того, можно задать размер окна. Например, в следующем скрипте открывается новое окно размером 400x300 пикселей. Оно не имеет ни строки статуса, ни панели инструментов, ни меню.

```
<html>
<head>

<script language="JavaScript">
<!-- hide

function openWin2() {
  myWin= open("page4.htm", "displayWindow",
    "width=400,height=300,status=no,toolbar=no,menubar=no");
}

// -->
</script>

</head>
<body>

<form>
<input type="button" value="Открыть новое окно" onClick="openWin2()">
</form>

</body>
</html>
```

(Открыть новое окно)

Как видите, свойства окна мы формулируем в строке *"width=400,height=300,status=no,toolbar=no,menubar=no"*. **Обратите внимание также на то, что не следует помещать в этой строке символы пробела**.

Список свойств окна, которыми можно управлять (табл. 7).

В версии 1.2 языка JavaScript были добавлены некоторые новые свойства (т.е. в Netscape Navigator 4.0). Вам не следует пользоваться этими свойствами, готовя материалы для Netscape 3.x или Microsoft Internet Explorer 3.x, поскольку эти броузеры не понимают языка 1.2 JavaScript. Новые свойства окон (табл. 8).

Например, теперь с помощью этих свойств можно определить, в каком месте экрана должно находиться вновь открываемое окно. Работая со старой версией языка JavaScript, это сделать было невозможно.

Таблица 7

1	2	3
directories	Yes\ no	Позволяет указывать, отображается ли полоса кнопок для выбора каталогов
height	*количество пикселей*	Задаёт высоту окна в пикселах. Минимальное значение –100
location	Yes\ no	Позволяет указывать, отображается ли полоса для ввода адреса
menubar	Yes\ no	Позволяет указывать, отображается ли полоса меню
resizable	Yes\ no	Позволяет указывать, может ли окно менять свой размер
scrollbars	Yes\ no	Задаёт отображение горизонтальной и вертикальной полос прокрутки
status	Yes\ no	Позволяет указывать, отображается ли полоса статуса
toolbar	Yes\ no	Позволяет указывать, отображается ли полоса статуса
width	*количество пикселов*	Задаёт ширину окна в пикселах. Минимальное значение – 100
fullscreen	Yes\ no	Указывает, показывается ли новое окно на полный экран или как обычное окно. По умолчанию показывается обычное окно
channelmode	Yes\ no	Позволяет указать, отображается ли полоса каналов
top	Yes\ no	Задаёт вертикальную координату левого верхнего угла
left	число	Задаёт горизонтальную координату левого верхнего угла

Таблица 8

alwaysLowered	Yes\ no
alwaysRaised	Yes\ no
dependent	Yes\ no
hotkeys	Yes\ no
innerWidth	*количество пикселей* (заменяет width)
innerHeight	*количество пикселей* (заменяет height)
outerWidth	*количество пикселей*
outerHeight	*количество пикселей*
screenX	*количество пикселей*
screenY	*количество пикселей*
titlebar	Yes\ no
z-lock	Yes\ no

5. Имя окна

Отметим, что открывая окна, необходимо использовать три аргумента:

```
myWin= open("page4.htm", "displayWindow",
  "width=400,height=300,status=no,toolbar=no,menubar=no")
```

Второй аргумент – это имя окна. Если известно имя окна, то можно загрузить туда новую страницу с помощью записи:

```
<a href="page4.html" target="displayWindow">
```

При этом необходимо указать имя соответствующего окна (если же такого окна не существует, то с этим именем будет создано новое). Обратите внимание, что *myWin* – это вовсе не имя окна. Но только с помощью этой переменной можно получить доступ к окну. И поскольку это обычная переменная, то область ее действия – лишь тот скрипт, в котором она определена. А между тем имя окна (в данном случае это *displayWindow*) – уникальный идентификатор, которым можно пользоваться с *любого* из окон броузера.

6. Закрытие окон

Чтобы закрыть окно, вам понадобится метод close(). Сначала откроем окно и загрузим туда HTML-страницу:

```
<html>
<script language="JavaScript">
<!-- hide

function closeIt() {
  close();
}

// -->
</script>

<center>
<form>
<input type=button value="Закрыть окно" onClick="closeIt()">
</form>
</center>

</html>
```

Если теперь в новом окне нажать кнопку, то оно будет закрыто.

Open() и close() – это методы объекта window. Необходимо помнить, что следует писать не просто *open()* и *close()*, а *window.open()* и *window.close()*. Однако в рассматриваемом случае объект *window* можно опустить – нет необходимости писать префикс *window*, если нужно всего лишь вызвать один из методов этого объекта (и такое возможно только для этого объекта).

7. Динамическое создание документов

Рассмотрим динамическое создание документов. Можно создавать новые HTML-страницы, можно также создавать и другие документы Web, такие как VRML-сцены и т.д. Для удобства можно размещать эти документы в отдельном окне или фрейме. Для начала мы создадим простой HTML-документ, который покажем в новом окне. Рассмотрим следующий скрипт.

```
<html>
<head>
<script language="JavaScript">
<!-- hide
```

```
function openWin4() {
  myWin= open("", "displayWindow",
    "width=500,height=400,status=yes,toolbar=yes,menubar=yes")
  // открыть объект document для последующей печати
  myWin.document.open()
    // генерировать новый документ
  myWin.document.write("<html><head><title>Динамическое создание
HTML-документа")
  myWin.document.write("</title><META HTTP-EQUIV='Content-Type'
CONTENT='text/html;")
  myWin.document.write("charset=windows-1251'></head><body back-
ground='bg.gif'>")
  myWin.document.write("<center><font size=+3>")
  myWin.document.write("Этот HTML-документ создан при помощи ");
  myWin.document.write("JavaScript!");
  myWin.document.write("</font></center>")
  myWin.document.write("</body></html>")
  // закрыть документ - (но не окно!)
  myWin.document.close();
}
// -->
</script>
</head>
<body>
<form>
<input type=button value="Динамическое создание HTML-документа" on-
Click="openWin4()">
</form>
</body>
</html>
```

Рассмотрим функцию openWin4(). Сначала открывается новое окно браузера. Поскольку первый аргумент функции open() – пустая строка (″″), то это значит, что не нужно в данном случае указывать конкретный адрес URL. Браузер должен только не обработать имеющийся документ - JavaScript обязан создать дополнительно новый документ. В скрипте определяем переменную *myWin*. И с ее помощью можно получать доступ к новому окну. Обратите внимание, что в данном случае нет возможности воспользоваться для этой цели именем окна (*displayWindow*). После того как окно открыто, наступает очередь открыть для записи объект *document*. Делается это с помощью команды:

```
// открыть объект document для последующей печати
myWin.document.open()
```

Здесь обращаемся к open() – методу объекта *document*. Это совсем не то же самое, что метод open() объекта *window*. Эта команда не открывает нового окна – она лишь готовит document к предстоящей печати. Кроме того, нужно поставить перед *document.open()* приставку *myWin*, чтобы получить возможность писать в новом окне. В последующих строках скрипта с помощью вызова *document.write()* формируется текст нового документа:

```
// генерировать новый документ
myWin.document.write("<html><head><title>Dinamic HTML")
myWin.document.write("</title></head><body background='bg.gif'>")
myWin.document.write("<center><font size=+3>")
myWin.document.write("This HTML-document has been created ");
myWin.document.write("with the help of JavaScript!");
myWin.document.write("</font></center>")
myWin.document.write("</body></html>")
```

Здесь записываем в документ обычные тэги языка HTML. То есть фактически генерируем разметку HTML. При этом можно использовать абсолютно любые тэги HTML. По завершении этого необходимо закрыть документ. Это делается следующей командой:

```
// закрыть документ - (но не окно!)
myWin.document.close()
```

Как уже упоминалось ранее, можно не только динамически создавать документы, но и по своему выбору размещать их в том или ином фрейме. Например, если имеем два фрейма с именами *frame1* и *frame2*, а теперь во *frame2* нужно сгенерировать новый документ, то для этого в *frame1* достаточно будет написать следующее:

```
parent.frame2.document.open();

parent.frame2.document.write ("Здесь будет располагаться ваш HTML-код");
parent.frame2.document.close();
```

Порядок выполнения работы

1. Создать с помощью программы «Блокнот» гипертекстовый файл. Имя файла – ваша фамилия. Файл разместить на указанном диске в папке с номером вашей группы.

2. Создать навигационную панель, состоящую из трёх фреймов. В первом фрейме расположить меню из трёх компонент. Активизация первой компоненты приводит к открытию страницы, которая будет помещена не в тот же самый фрейм, а во второй. Активизация второй компоненты – открытию страницы, которая будет помещена в третий фрейм. Активизация третей – к выводу рисунка, помещённого во второй фрейм.

3. Создать форму с двумя кнопками. Нажатие первой приводит к закрытию окна, второй – к выводу сообщения.

Контрольные вопросы

1. Обращение к фреймам из родительского окна.
2. Навигационные панели.
3. Создание окон.
4. Закрытие окон.
5. Динамическое создание документов.

ЛАБОРАТОРНАЯ РАБОТА № 6
Строка состояния и таймеры

Цель работы: изучить возможности JavaScript, связанные со строкой состояния и таймерами.

1. Строка состояния

Программы на JavaScript могут выполнять запись в строку состояния – прямоугольник в нижней части окна броузера. Все, что необходимо для этого сделать – записать нужную строку в *window.status*. В следующем примере создаются две кнопки, которые можно использовать, чтобы записывать текст в строку состояния и соответственно затем его стирать.

Данный скрипт выглядит следующим образом:

```
<html>
<head>
<script language="JavaScript">
<!-- hide

function statbar(txt) {
  window.status = txt;
// -->
```

```
</script>
</head>
<body>

<form>
  <input type="button" name="look" value="Писать!"
    onClick="statbar('Привет! Это окно состояния!');">
  <input type="button" name="erase" value="Стереть!"
    onClick="statbar('');">
</form>

</body>
</html>
```

В данном примере создается форма с двумя кнопками. Обе эти кнопки вызывают функцию *statbar()*. Вызов от клавиши *Писать!* выглядит следующим образом:

```
statbar('Привет! Это окно состояния!');
```

В скобках написана строка: *'Привет! Это окно состояния!'*. Это текст, передаваемый функции *statbar()*. Функция *statbar()* определена следующим образом:

```
function statbar(txt) {
  window.status = txt;
}
```

В заголовке функции в скобках помещено слово *txt*. Это означает, что строка, которую передали этой функции, помещается в переменную *txt*. Передача функциям переменных – прием, часто применяемый для придания функциям большей гибкости. Можно передать функции несколько таких аргументов – необходимо отделить их друг от друга запятыми. Строка *txt* заносится в строку состояния посредством команды window.status=txt. Соответственно удаление текста из строки состояния выполняется как запись в *window.status* пустой строки.

Механизм вывода текста в строку состояния удобно использовать при работе со ссылками. Здесь удобно пользоваться процедурами *onMouseOver* и *onMouseOut*, чтобы отслеживать моменты, когда указатель мыши проходит над данной ссылкой. В onMouseOver необходимо возвращать результат *true* для того, чтобы броузер не выполнял далее свой собственный код обработки события MouseOver. Как правило, в строке состояния броузер показывает URL соответствующей ссылки. Если же значение *true* не возвращается, то сразу же после того, как код программы был выполнен, броузер перепишет строку состояния – т.е. текст будет тут же затерт и пользователь не сможет его увидеть.

В общем случае всегда можно отменить дальнейшую обработку события броузером, возвращая *true* в своей собственной процедуре обработки события.

2. Таймеры

С помощью функции Timeout (или таймера) можно запрограммировать компьютер на выполнение некоторых команд по истечении некоторого времени. В следующем скрипте демонстрируется кнопка, которая открывает выпадающее окно не сразу, а по истечении 3 секунд.

Скрипт выглядит следующим образом:

```
<script language="JavaScript">
<!-- hide

function timer() {
  setTimeout("alert('Время истекло!')", 3000);
}

// -->
</script>
...
<form>
  <input type="button" value="Таймер" onClick="timer()">
</form>
```

Здесь *setTimeout()* – это метод объекта window. Он устанавливает интервал времени. Первый аргумент при вызове – это код JavaScript, который следует выполнить по истечении указанного времени. В нашем случае это вызов – *"alert('Время истекло!')"*. Обратите внимание, что код на JavaScript должен быть заключен в кавычки. Во втором аргументе сообщается, когда этот код следует выполнять. При этом время нужно указывать в миллисекундах (3000 миллисекунд = 3 секунды).

```
<html>
<head>
<title> Clock in statu</title>
<script language="JavaScript">
function clock_status()
{
setTimeout("clock_status()",100);
today=new Date();
window.status=today;
```

```
}
</script>
</head>
<body onLoad="clock_status()">
</body>
</html>
```

При помощи строки *today=new Date()* создается экземпляр объекта *Date* используя конструктор *new*. Если не указано никаких параметров, то экземпляр будет содержать текущую дату и время. Рассмотрим ещё один пример с таймером:

```
    <html>
<head>
<title>Clock full</title>
</head>
<script language="JavaScript">
function fulltime()
{
var time=new Date();
document.clock.full.value=time.toLocaleString();
setTimeout('fulltime()',500)
}
</script>
<body bgcolor=fffff1 text=ff0000>
<center>
<form name=clock>
<input type=text size=17 name=full>
</form>
<script language="JavaScript">
fulltime();
</script>
</center>
</body>
</html>
```

3. Прокрутка

Выше рассматривалось, как текст помещается в строке состояния. В Интернет этим приемом пользуются повсеместно. Теперь рассмотрим, как можно запрограммировать прокрутку в основной линейке. Создать бегущую строку довольно просто. Сначала нужно записать в строку состояния текст. Затем по истечении короткого интервала времени нужно записать туда тот же самый текст, но при этом немного переместив его влево. Если это сделать несколько

раз, то у пользователя создастся впечатление, что он имеет дело с бегущей строкой. Однако при этом нужно помнить еще и о том, что необходимо каждый раз вычислять, какую часть текста следует показывать в строке состояния (как правило, объем текстового материала превышает размер строки состояния).

Рассмотрим пример:

```
<html>

<head>
<script language="JavaScript">
<!-- hide

// определение текста для прокрутки
var scrtxt = "Это JavaScript! " +
   "Это JavaScript! " +
   "Это JavaScript!";
var len = scrtxt.length;
var width = 100;
var pos = -(width + 2);
function scroll() {
 // напечатать заданный текст справа и установить таймер
 // перейти на исходную позицию для следующего шага
 pos++;
 // вычленить видимую часть текста
 var scroller = "";
 if (pos == len) {
  pos = -(width + 2);
 }
 // если текст еще не дошел до левой границы, то нужно
 // добавить перед ним несколько пробелов. В противном случае нужно
 // вырезать начало текста (ту часть, что уже ушла за левую границу
 if (pos < 0) {
  for (var i = 1; i <= Math.abs(pos); i++) {
   scroller = scroller + " ";}
  scroller = scroller + scrtxt.substring(0, width - i + 1);
 }
 else {
  scroller = scroller + scrtxt.substring(pos, width + pos);
 }
 // разместить текст в строке состояния
 window.status = scroller;

 // вызвать эту функцию вновь через 100 миллисекунд
```

```
  setTimeout("scroll()", 100);
}
// -->
</script>
</head>
<body onLoad="scroll()">
<H2>Это пример прокрутки в строке состояния средствами JavaScript.</H2>
</body>
</html>
```

Большая часть функции *scroll()* нужна для вычисления той части текста, которая будет показана пользователю. Разберем этот код. Чтобы запустить данный процесс, используется процедура обработки события onLoad, описанная в тэге <body>. То есть функция *scroll()* будет вызвана сразу же после загрузки HTML-страницы. Посредством процедуры onLoad вызывается функция *scroll()*. Сначала в функции *scroll()* устанавливается таймер. Этим гарантируется, что функция *scroll()* будет повторно вызвана через 100 миллисекунд. При этом текст будет перемещен еще на один шаг и запущен другой таймер. Так будет продолжаться без конца. Скроллинг используется в Интернет довольно широко. Из-за непрерывного скроллинга становится невозможным прочесть в строке состояния адрес URL. Эту проблему можно было решить, позаботившись о приостановке скроллига, если происходит событие onMouseOver – и соответственно продолжении, когда фиксируется onMouseOut.

Порядок выполнения работы

1. Создать с помощью программы "Блокнот" гипертекстовый файл. Имя файла – ваша фамилия. Файл разместить на указанном диске в папке с номером вашей группы.
2. Набрать скриптовые программы, приведенные в методических указаниях к выполнению данной лабораторной работы.
3. Написать процедуру, которая бы при наведении на ссылку писала сообщение пользователю в строке состояния, а при отсутствии фокуса стирала бы это сообщение.
4. Написать процедуру, реализующую прокрутку строки в окне поля формы.

Контрольные вопросы

1. Объект **Date** и его методы.
2. Функция Timeout.
3. Строка состояния.

ЛАБОРАТОРНАЯ РАБОТА № 7
Предопределенные объекты

Цель работы: изучить предопределенные объекты JavaScript.

1. Объект Date

Примерами предопределенных объектов могут служить Date, Array или Math. Есть еще несколько таких же объектов (полное описание см. в документации, предоставляемой фирмой Netscape). Для начала рассмотрим объект Date. Он позволяет работать как со временем, так и с датой. Например, можно легко определить, сколько дней еще остается до определенного события или внести в HTML-документ запись текущего времени. Рассмотрим пример, который высвечивает на экран текущее время. Сначала необходимо создать новый объект Date. Для этого используем оператор *new*:

```
today= new Date()
```

Здесь создается новый объект Date с именем *today*. Если при создании этого нового объекта Date не указана какая-либо определенная дата и время, то будут предоставлены текущие дата и время. То есть после выполнения команды today= new Date() вновь созданный объект *today* будет указывать именно те дату и время, когда данная команда была выполнена. Объект Date предоставляет методы, которые теперь могут применяться к объекту *today*. Например, это методы – getHours(), setHours(), getMinutes(), setMinutes(), getMonth(), setMonth() и так далее. Обратите внимание, что объект Date лишь содержит определенную запись о дате и времени. Он не уподобляется часам, автоматически отслеживающим время каждую секунду либо миллисекунду. Чтобы зафиксировать какие-либо другие дату и время, мы можем воспользоваться видоизмененным конструктором (это будет метод Date(), который при создании нового объекта Date вызывается через оператор new):

```
today= new Date(1999, 0, 1, 17, 35, 23)
```

При этом будет создан объект Date, в котором будет зафиксировано первое января 1999 года 17:35 и 23 секунды. Таким образом, можно выбрать дату и время по следующему шаблону:

```
Date(year, month, day, hours, minutes, seconds)
```

Заметьте, что для обозначения января нужно использовать число 0, а не 1. Число 1 будет обозначать февраль, и так далее.

Теперь напишем скрипт, печатающий текущие дату и время.

Код выглядит следующим образом:

```
<script language="JavaScript">
<!-- hide
now= new Date();
document.write("Время: " + now.getHours() + ":" + now.getMinutes() +
"<br>");
document.write("Дата: " + (now.getMonth() + 1) + "/" + now.getDate() + "/" +
        now.getYear());
// -->
</script>
```

Здесь пользуемся такими методами, как getHours(), чтобы вывести на экран время и дату, указанные в объекте Date с именем *now*. Обязательно надо увеличивать на единицу значение, получаемое от метода getMonth(). В данном скрипте не выполняется проверки на тот случай, если количество минут окажется меньше чем 10. Это значит, что можно получить запись времени примерно в следующем виде: *14:3*, что на самом деле должно было бы означать *14:03*. Решение этой проблемы рассмотрим в следующем примере.

Рассмотрим теперь скрипт, создающий на экране изображение работающих часов:

Время: 10:20:07
Дата: 7/23/2005

Исходный код скрипта:

```
<html>
<head>

<script Language="JavaScript">
<!-- hide

var timeStr, dateStr;

function clock() {
    now= new Date();

    // время
    hours= now.getHours();
    minutes= now.getMinutes();
    seconds= now.getSeconds();
    timeStr= "" + hours;
    timeStr+= ((minutes < 10) ? ":0" : ":") + minutes;
```

```
        timeStr+= ((seconds < 10) ? ":0" : ":") + seconds;
        document.clock.time.value = timeStr;
        // дата
        date= now.getDate();
        month= now.getMonth()+1;
        year= now.getYear();
        dateStr= "" + month;
        dateStr+= ((date < 10) ? "/0" : "/") + date;
        dateStr+= "/" + year;
        document.clock.date.value = dateStr;
        Timer= setTimeout("clock()",1000);
}
// -->
</script>
</head>
<body onLoad="clock()">
<form name="clock">
 Время:
  <input type="text" name="time" size="8"  value=""><br>
 Дата:
  <input type="text" name="date" size="8" value="">
</form>
</body>
</html>
```

Здесь для ежесекундной коррекции времени и даты пользуемся методом setTimeout(). Фактически это сводится к кому, что каждую секунду создается новый объект Date и заносится туда текущее время. Можно видеть, что функции clock() вызываются программой обработки события onLoad, помещенной в тэг <body>. В разделе body HTML-страницы имеется два элемента формы для ввода текста. Функция clock() записывает в оба эти элемента в корректном формате текущие время и дату. Для этой цели используются две строки *timeStr* и *dateStr*. Как было упомянуто ранее, существует проблема с индикацией, когда количество минут меньше 10 – в данном скрипте эта проблема решается с помощью следующей строки:

```
timeStr+= ((minutes < 10) ? ":0" : ":") + minutes;
```

Количество минут заносится в строку *timeStr*. Если у нас менее 10 минут, то мы еще должны приписать спереди 0. Эта строка в скрипте может показаться немного странной, и ее можно было бы переписать в более знакомом в виде:

```
if (minutes < 10) timeStr+= ":0" + minutes
  else timeStr+= ":" + minutes;
```

2. Объект Array

Начиная с версии 1.1 языка JavaScript (Netscape Navigator 3.0), можно использовать объект Array. Создать новый массив можно, записав myArray= new Array(). После этого начать заносить в массив значения:

```
myArray[0]= 17;
myArray[1]= "Вова";
myArray[2]= "Саша";
```

Массивы JavaScript обладают большой гибкостью. Размер массива устанавливается динамически. Если написать myArray[99]= "xyz", размер массива будет установлен 100 элементов. (В языке JavaScript размер массива может только увеличиваться – массив не может «сжиматься». Поэтому надо делать массивы как можно компактнее.) Не имеет значения, заносятся ли в массив числа, строки, либо другие объекты. Следующий скрипт печатает текст:

```
first element
second element
third element
```

Исходный код:

```
<script language="JavaScript">
<!-- hide

myArray= new Array();
myArray[0]= "first element";
myArray[1]= "second element";
myArray[2]= "third element";

for (var i= 0; i< 3; i++) {
  document.write(myArray[i] + "<br>");
}

// -->
</script>
```

Сначала создаем новый массив с именем myArray. Затем заносим в него три различных значения. После этого запускаем цикл, который трижды выполняет команду document.write(myArray[i] + "
");. В переменной *i* ведется отсчет циклов от 0 до 2. Заметим, что в цикле используется конструкция

myArray[i]. И поскольку *i* меняет значения от 0 до 2, то в итоге получаем три различных вызова document.write(). Иными словами, можно расписать этот цикл как

```
document.write(myArray[0] + "<br>");
document.write(myArray[1] + "<br>");
document.write(myArray[2] + "<br>");
```

При помощи свойства length можно определить размерность массива.

3. Объект Math

Если в скрипте выполняются математические рассчеты, то необходимо пользоваться объектом Math. Например, имеется метод синуса sin(). Полную информацию об этом объекте вы найдете в документации фирмы Netscape. Продемонстрируем работу метода random(). При вызове функции Math.random() получается случайное число, лежащее в диапазоне между 0 и 1. Один из возможных результатов вызова document.write(Math.random()) (при каждой новое загрузке данной страницы здесь будет появляться другое число): 0.7413992716208698 *0.3448107284244919*

Порядок выполнения работы

1. Создать с помощью программы «Блокнот» гипертекстовый файл. Имя файла – ваша фамилия. Файл разместить на указанном диске в папке с номером вашей группы.
2. Набрать скриптовые программы, приведенные в методических указаниях к выполнению данной лабораторной работы.
3. Объявите массив месяцев года. Выведите этот массив и количество дней месяцев с помощью свойства length.

Контрольные вопросы

1. Объект Math.
2. Объект Array.
3. Объект Date.

ЛАБОРАТОРНАЯ РАБОТА № 8

Объект Image

Цель работы: получить сведения об объектах Image Java Script.

1. Изображения на Web-странице

Рассмотрим объект Image, который стал доступен, начиная с версии с 1.1 языка JavaScript (т.е. с Netscape Navigator 3.0). С помощью объекта Image можно вносить изменения в графические образы, присутствующие на Web-странице. В частности, это позволяет создавать мультипликацию. Заметим, что пользователи броузеров более старых версий (таких как Netscape Navigator 2.0 или Microsoft Internet Explorer 3.0 – т.е. использующих версию 1.0 языка JavaScript) не смогут запускать скрипты, приведенные в этой части описания, или на них нельзя будет получить полный эффект. Сначала рассмотрим, как из JavaScript можно адресоваться к изображениям, представленным на Web-странице. В рассматриваемом языке все изображения предстают в виде массива. Массив этот называется *images* и является свойством объекта document. Каждое изображение на Web-странице получает порядковый номер: первое изображение получает номер 0, второе – номер 1 и т.д. Таким образом, к первому изображению мы можем адресоваться, записав document.images[0]. Каждое изображение в HTML-документе рассматривается в качестве объекта Image. Объект Image имеет определенные свойства, к которым можно обращаться из языка JavaScript. Например, можно определить, какой размер имеет изображение, обратившись к его свойствам *width* и *height*. То есть по записи document.images[0].width можно определить ширину первого изображения на Web-странице (в пикселах). Отслеживать индекс всех изображений может оказаться затруднительным, особенно если на одной странице их довольно много. Эта проблема решается назначением изображениям своих собственных имен. Если задавать изображение с помощью тэга

<img src="img.gif" name="myImage" width=100 height=100>,

то можно обращаться к нему, написав document.myImage или document.images["myImage"].

Загрузка новых изображений

Для смены изображений на Web-странице понадобится атрибут *src*. Как и в случае тэга <img>, атрибут *src* содержит адрес представленного изображения. Теперь - в языке JavaScript 1.1 – имеется возможность назначать новый адрес изображению, уже загруженному в Web-страницу. В результате изображение

будет загружено с этого нового адреса, заменив на Web-странице старое. Рассмотрим пример:

<img src="img1.gif" name="myImage" width=100 height=100>.

Здесь загружается изображение *img1.gif* и получает имя *myImage*. В следующей строке прежнее изображение *img1.gif* заменяется уже на новое – *img2.gif*:

document.myImage.src= "img2.src";

При этом новое изображение всегда получает тот же размер, что был у старого. И невозможно изменить размер поля, в котором это изображение размещается.

2. Упреждающая загрузка изображения

Один из недостатков такого подхода может заключаться в том, что после записи в *src* нового адреса начинает процесс загрузки соответствующего изображения. И поскольку этого не было сделано заранее, то еще пройдет некоторое время, прежде чем новое изображение будет передано через Интернет и встанет на свое место. В некоторых ситуациях это допустимо, однако часто подобные задержки неприемлемы. Решением проблемы была бы упреждающая загрузка изображения. Для этого создадим новый объект Image. Рассмотрим следующие строки:

```
hiddenImg= new Image();
hiddenImg.src= "img3.gif";
```

В первой строке создается новый объект Image. Во второй строке указывается адрес изображения, которое в дальнейшем будет представлено с помощью объекта *hiddenImg*. Запись нового адреса в атрибуте *src* заставляет броузер загружать изображение с указанного адреса. Поэтому, когда выполняется вторая строка нашего примера, начинает загружаться изображение *img2.gif*. Но как подразумевается самим названием *hiddenImg* ("скрытая картинка"), после того как броузер закончит загрузку, изображение на экране не появится. Оно будет лишь сохранено в памяти компьютера (или точнее в кэше) для последующего использования. Чтобы вызвать изображение на экран, можно воспользоваться строкой

document.myImage.src= hiddenImg.src;

Но теперь изображение уже немедленно извлекается из кэша и показывается на экране. Конечно, броузер должен был к моменту запроса закончить упреждающую загрузку, чтобы необходимое изображение было показано без за-

держки. Поэтому если существует необходимость предварительно загрузить большое количество изображений, то может иметь место задержка, поскольку броузер будет занят загрузкой всех картинок. Можно создать красивые эффекты, используя смену изображений в качестве реакции на определенные события. Например, можно изменять изображения в тот момент, когда курсор мыши попадает на определенную часть страницы. Проверьте, как работает следующий пример, просто поместив курсор мыши на картинку.

Исходный код этого примера выглядит следующим образом:

```
<a href="#"
 onMouseOver="document.myImage2.src='img2.gif'"
 onMouseOut="document.myImage2.src='img1.gif'">
<img src="img1.gif" name="myImage2" width=160 height=50 border=0></a>
```

При этом могут возникнуть следующие проблемы:

- пользователь пользуется броузером, не имеющим поддержки JavaScript 1.1;
- второе изображение не было загружено.

Теперь рассмотрим полный вариант скрипта, который мог бы решить эти проблемы. Чтобы этот скрипт сохранял свою гибкость, следует соблюдать два условия:

- не задано количество изображений – не должно иметь значения, сколько их используется, 10 или 100;
- не задан порядок следования изображений – должна существовать возможность изменять этот порядок без изменения самого кода.

Рассмотрим данный пример:

```
 <html>

<head>
<script language="JavaScript">
<!-- hide
 // ok, у нас броузер с поддержкой JavaScript
var browserOK = false;
 var pics;
// -->
</script>
<script language="JavaScript1.1">
<!-- hide
```

```
 // броузер с поддержкой JavaScript 1.1!
 browserOK = true;
 pics = new Array();
// -->
</script>

<script language="JavaScript">
<!-- hide

var objCount = 0; // количество изображений на Web-странице

function preload(name, first, second) {

 // предварительная загрузка изображений и размещение их в массиве

 if (browserOK) {
pics[objCount] = new Array(3);
 pics[objCount][0] = new Image();
 pics[objCount][0].src = first;
 pics[objCount][1] = new Image();
pics[objCount][1].src = second;
 pics[objCount][2] = name;
 objCount++;
}
}
function on(name){
 if (browserOK) {
 for (i = 0; i < objCount; i++) {
 if (document.images[pics[i][2]] != null)
if (name != pics[i][2]) {
// вернуть в исходное состояние все другие изображения
document.images[pics[i][2]].src = pics[i][0].src;
 } else {
 // показывать вторую картинку, поскольку курсор пересекает данное изо-
бражение
 document.images[pics[i][2]].src = pics[i][1].src;
 }
}
 }
}
function off(){
if (browserOK) {
 for (i = 0; i < objCount; i++) {
```

```
 // вернуть в исходное состояние все изображения
 if (document.images[pics[i][2]] != null)
 document.images[pics[i][2]].src = pics[i][0].src;
}
 }
}
// заранее загружаемые изображения – здесь надо указать
// изображения, которые нужно загрузить заранее, а также объект Image,
// к которому они относятся (первый аргумент). Именно эту часть
// нужно корректировать, если надо использовать скрипт
// применительно к другим изображениям
preload("link1", "img1f.gif", "img1t.gif");
preload("link2", "img2f.gif", "img2t.gif");
preload("link3", "img3f.gif", "img3t.gif");
// -->
</script>
<head>
<body>
<a href="link1.htm" onMouseOver="on('link1')"
onMouseOut="off()">
<img name="link1" src="link1f.gif"
 width="140" height="50" border="0"></a>
<a href="link2.htm" onMouseOver="on('link2')"
 onMouseOut="off()">
<img name="link2" src="link2f.gif"
 width="140" height="50" border="0"></a>
<a href="link3.htm" onMouseOver="on('link3')"
 onMouseOut="off()">
<img name="link3" src="link3f.gif"
 width="140" height="50" border="0"></a>
</body>
</html>
```

Данный скрипт помещает все изображения в массив *pics*. Создает этот массив функция preload(), которая вызвается вначале. Вызов функции preload() выглядит просто как

```
preload("link1", "img1f.gif", "img1t.gif");
```

Это означает, что скрипт должен загрузить с сервера два изображения: *img1f.gif* и *img1t.gif*. Первое из них – это та картинка, которая будет представлена, пока курсор мыши не попадает в область изображения. Когда же пользователь помещает курсор мыши на изображение, то появляется вторая картинка. При вызове функции preload() в качестве первого аргумента мы указываем слово *"link1"* и тем самым задаем на Web-странице объект Image, которому и будут

принадлежать оба предварительно загруженных изображения. Если вы посмотрите пример в разделе <body>, то обнаружите изображение с тем же именем *link1*. Мы пользуем не порядковый номер, а именно имя изображения для того, чтобы иметь возможность переставлять изображения на Web-странице, не переписывая при этом сам скрипт. Обе функции on() и off() вызываются посредством программ обработки событий onMouseOver и onMouseOut. Поскольку сам элемент image не может отслеживать события MouseOver и MouseOut, то мы обязаны сделать на этих изображениях еще и ссылки. Можно видеть, что функция on() возвращает все изображения, кроме указанного, в исходное состояние. Делать это необходимо потому, что в противном случае выделеными могут оказаться сразу несколько изображений (дело в том, что событие MouseOut не будет зарегистрировано, если пользователь переместит курсор с изображения сразу за пределы окна).

Порядок выполнения работы

1. Создать с помощью программы «Блокнот» гипертекстовый файл. Имя файла – ваша фамилия. Файл разместить на указанном диске в папке с номером вашей группы.

2. Разобрать скриптовые программы, приведенные в методических указаниях к выполнению данной лабораторной работы.

Контрольные вопросы

1. Объект Image.
2. Упреждающая загрузка изображения.

ЛИТЕРАТУРА

1. Бойс Дж. и др. Сетевые возможности Windows 95. Настольная книга пользователя. – М.: Восточная Книжная Компания, 1997.

2. Гиббсон Д. и др. Работа в E-Mail. – М.: БИНОМ, 1996.

3. Пайк М. Internet в подлиннике. – СПб.:ВНУ – Санкт-Петербург, 1997.

4. Ахметов К.С., Федоров А.Г. Microsoft Internet Explorer 4.0 для всех. – М.: КомпьютерПресс, 1997.

5. Майкл А. Ларсон. Создание Web-страниц с помощью MS Office 97. – М.: БИНОМ, 1998.

6. Кингсли-Хью Э., Кингсли-Хью К. JavaScript1.5: учебный курс. – СПб.: Питер, 2000.

7. Лещев Д. Создание интерактивного Web-сайта. Учебный курс. СПб.: Питер, 2003.

ПРИЛОЖЕНИЕ

Структура отчета

Отчет о выполненной лабораторной работе должен включать:

1. Титульный лист.
2. Постановка задачи.
3. Описание хода выполнения работы (последовательность выполняемых действий, команд, тексты HTML-файлов и т.д.).
4. Краткие ответы на контрольные вопросы.
5. Выводы.

Список использованной литературы.

Св. план 2005, поз. 138

Учебное издание

ИНТЕРАКТИВНЫЙ МАРКЕТИНГ И ЭЛЕКТРОННАЯ КОММЕРЦИЯ

Лабораторный практикум

для студентов специальности «Маркетинг» дневной формы обучения

Автор-составитель:
Бесчастная Елена Владимировна

Редактор Т.П. Андрейченко
Корректор Е.Н. Батурчик

Подписано в печать 31.05.2006. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 4,77.
Уч.-изд. л. 3,0. Тираж 150 экз. Заказ 603.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛП №02330/0131518 от 30.04.2004.
220013, Минск, П. Бровки, 6