ГОУ ВПО «Дагестанский государственный институт народного хозяйства»

Кафедра «Информационные технологии»
Загидова Хатима Магомедовна
Магомедова Динара Сахратулаевна

Учебное пособие
по дисциплине

«Основы Web-программирования»
(курс лекций)

Для студентов 4 курса дневного и заочного отделений факультета «Прикладная информатика (в экономике)»
Направления подготовки 230700 «Прикладная информатика»
Профиля подготовки «Прикладная информатика в образовании»

Махачкала – 2011

***Разработчики учебного пособия*:** Загидова Хатима Магомедовна, старший преподаватель кафедры «Информационные технологии» ДГИНХ; Магомедова Динара Сахратулаевна, старший преподаватель«Информационные технологии» ДГИНХ

Внутренний рецензент: Раджабов Карахан Якубович, кандидат экономических наук, доцент кафедры информационных технологий, начальник департамента факультета «Прикладная информатика (в экономике)».

Внешний рецензент: Гаджиев Хаджимурад Магомедович, кандидат экономических наук, профессор, заведующий кафедрой «Вычислительная техника» Дагестанского филиала московского института радиотехники, электроники и автоматики (технический университет) в г. Махачкале.

Учебное пособие разработано с учетом п.41 Типового положения об образовательном учреждении высшего профессионального образования (высшем учебном заведении) РФ, утвержденного постановлением Правительства РФ от 5 апреля 2001г. №204, а также в соответствии с письмами Министерства образования и науки РФ 19.05.2000г. №14-52-357ин/13 «О порядке формирования основных образовательных программ высшего учебного заведения на основе государственных образовательных стандартов» и от 17.04.2006г. №02-55-77ин/ак

Загидова Х.М., Магомедова Д.С. Учебное пособие по дисциплине «Основы Web-программирования» для направления подготовки «Прикладная информатика», профиля подготовки «Прикладная информатика в образовании» – Махачкала: ДГИНХ, 2011. – 176 с.

Рекомендовано к утверждению и к изданию Учебно-методическим советом ДГИНХ Проректор по учебной работе ДГИНХ, председатель Учебно-методического совета, доктор экономических наук, профессор Казаватова Н.Ю. 23 апреля 2011 г.	Одобрено Советом факультета «Прикладная информатика (в экономике)» Председатель Совета, к.э.н., доцент Раджабов К.Я. 16 апреля 2011 г.
Одобрено кафедрой «Информационные технологии», протокол № 7 от 21 марта 2011 г. зав. кафедрой к.ф-м.н., Галяев В.С.	

Печатается по решению Учебно-методического совета Дагестанского государственного института народного хозяйства.

Содержание.

№ п/п	Тема и план лекций	Количество часов
1.	**Тема 1: Введение в предмет.** 1. Основные понятия Интернет: World Wide Web, Веб-серверы, Веб-страницы, Протоколы, Браузеры, Адресация в сети Интернет 2. Классификация веб-сайтов	**2**
2.	**Тема 2. Правила построения простейших HTML-документов.** 1. Общие понятия языка разметки гипертекста – HTML 2. Структура HTML-документа 3. Ввод и оформление текста 4. Создание списков 5. Создание ссылок	**2**
3.	**Тема 3. Добавление изображений и мультимедиа на web-страницы.** 1. Встраивание изображений 2. Добавление мультимедийных файлов 3. Вставка фоновых изображений и фоновой музыки на веб-страницу	**2**
4.	**Тема 4. Создание таблиц средствами языка HTML.** 1. Проектирование простейших HTML-таблиц 2. Создание сложных таблиц и их форматирование	**2**
5.	**Тема 5. Создание фреймов в HTML.** 1. Разбиение окна браузера на фреймы 2. Ссылки внутри фреймов 3. Создание плавающих фреймов	**2**
6.	**Тема 6. Конструирование форм в языке HTML.** 1. Элемент FORM 2. Создание полей формы с помощью тегов <Textarea>, <Select> и <Input>	**2**
7.	**Тема 7. Карта графических ссылок.** 1. Технология создания карты графических ссылок 2. Характеристика основных тегов и атрибутов	**2**

8.	**Тема 8. Оформление веб-страниц с использованием каскадных таблиц стилей.** 1. Введение в таблицы стилей и язык CSS 2. Применение таблиц стилей CSS в форматировании текста 3. Структурное форматирование в CSS 4. Свойства для форматирования таблиц в CSS 5. Пользовательские формы в CSS 6. Позиционирование объектов	**2**
9.	**Тема 9. Основы веб-дизайна.** 1. Что такое дизайн сайта и зачем он нужен? 2. Основные аспекты формирования стиля веб - дизайна 3. Понимание роли графики в web-дизайне 4. Анимация, звук и видео в web-дизайне	**2**
10.	**Тема 10. Основы языка JavaScript: Переменные и типы данных, условные операторы и операторы циклов, функции и объекты.** 1. Введение в JavaScript 2. Встраивание JavaScript-кода в HTML-документы 3. Переменные и типы данных в JavaScript 4. Выражения и операторы в JavaScript 5. Функции и объекты в JavaScript	**6**
11.	**Тема 11. Особенности работы со встроенными объектами в Java Script.**	**2**
12.	**Тема 12. События, поддерживаемые JavaScript. Объект event.** 1. Интерактивность на веб-страницах. События в языке JavaScript *2.* Объект event	**2**
13.	**Тема 13. Особенности языка PHP. Синтаксис, грамматика и функции PHP.** 1. Переменные и константы в PHP 2. Выражения и операторы в PHP 3. Условная и циклическая обработка в PHP *4.* Полезные PHP-функции	**2**
14.	**Тема 14. PHP. Операторы INCLUDE и REQUIRE. Особенности написания функций.** 1. Инструкция require 2. Инструкция include 3. Общий синтаксис определения функции	**2**

15.	**Тема 15. Работа с классами в PHP.** 1. Классы, объекты и объявления методов *2.* Функции для работы с классами и объектами	**2**
16.	**Тема 16. PHP. Доступ к базам данных.** 1. Работа с базой данных MySQL 2. Язык запросов MySQL *3.* Работа с таблицами в MySQL	**2**
17.	**Тема 17. Активные серверные страницы (ASP). Особенности синтаксиса.** 1. ASP – активные серверные страницы *2.* Создание простейшей ASP-страницы	**2**
	ИТОГО:	**34**

Аннотация

Дисциплина «Интернет - программирование» занимает одно из наиболее важных мест в образовательном процессе данного направления, поскольку дает студентам навыки создания освоения возможностей языков HTML, JavaScript, PHP для программирования Web-сайтов и Web-интерфейсов к базам данных.Представленный материал соответствует образовательной программе по направлению, содержит в себе лекции, лабораторные задания, задания, упражнения, вопросы для повторения и самостоятельной работы, мультимедийные материалы, тестовые задания, экзаменационные вопросы. Все разделы и темы последовательны и тесно увязаны друг с другом.

Учебное пособие предназначено для студентов 3 курса дневного факультета «Прикладная информатика (в экономике)» направления подготовки «Математика и компьютерные науки» профиля подготовки «Математическое и компьютерное моделирование

Состав курса – 180 часов (5 з.е.):

Лекционная часть – 17 часа;

Лабораторные занятия – 34 часов;

Самостоятельная работа – 84 часа.

Формы контроля.

Выполнение лабораторных и промежуточных контрольных работ;

Семестровая аттестация;

Итоговый экзамен в конце семестра.

При составлении данного учебного пособия были использованы следующие источники:

1. Алексеев А.П. Введение в Web-дизайн: учебное пособие. Издательство: СОЛОН-ПРЕСС, 2008 г., 185 с.

2. Алексеев А.П. Введение в Web-дизайн: учебное пособие. Издательство: СОЛОН-ПРЕСС, 2008 г. 185 стр.
3. Гончаров А.Ю. Web-дизайн: HTML, JavaScript и CSS. Карманный справочник.
   Издательство: КУДИЦ-ПРЕСС, 2007 г. 321 стр.
4. Горнаков С.Г. Осваиваем популярные системы управления сайтом (CMS). Издательство: ДМК Пресс, 2009 г., 326 с.
5. Дронов В. В подлиннике. Adobe Dreamweaver CS4 + Видеокурс (+CD). BHV-СПб, 2009 г., 832 с.
6. Дронов В. Профессиональное программирование. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов. BHV-СПб, 2011 г., 416 с.
7. Едомский Ю. Техника Web-дизайна для студента. Издательство: БХВ-Петербург, 2005 г. 390 стр.
8. Кирьянов Д.В., Кирьянова Е.Н. Adobe Flash CS3 – это просто! Создаем Web-анимацию. Издательство: БХВ-Петербург, 2008 г., 234 с.

Рекомендован к использованию и изданию кафедрой «Информационные технологии» ДГИНХ (протокол №7, 21.03.2011), Советом факультета «Прикладная информатика (в экономике)» (16.04.2011), Методическим советом ДГИНХ (23.04.2011 г.).

Тема 1: Введение в предмет.

1. *Основные понятия Интернет: World Wide Web, Веб-серверы, Веб-страницы, Протоколы, Браузеры, Адресация в сети Интернет*
2. *Классификация веб-сайтов*

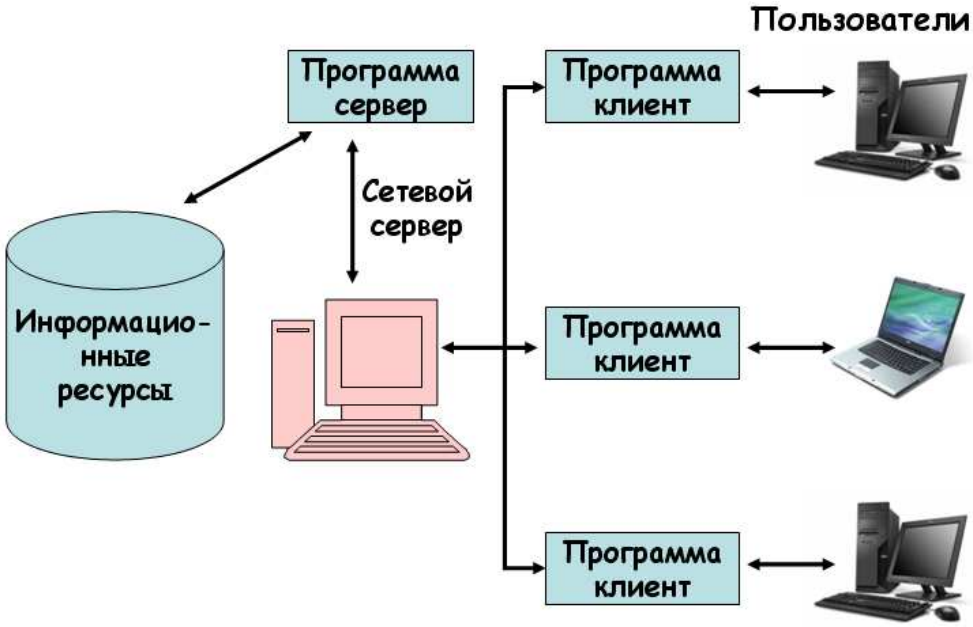
Вопрос 1. Основные понятия Интернет: World Wide Web, Веб-серверы, Протоколы, Браузеры, Адресация в сети Интернет.

Интернет — это всемирное объединение компьютеров, которые могут обмениваться между собой информацией, если быть более точными, *Интернет* - это глобальная компьютерная сеть, объединяющая локальные сети предприятий и организаций и компьютеры отдельных пользователей в общее информационное пространство. Интернет использует для связи магистральные каналы поставщиков телекоммуникационных услуг и клиентские каналы.

Провайдер – это фирма, предоставляющая конечным пользователям выход в Интернет через её локальную сеть.

Своей современной огромной популярностью Интернет обязан, прежде всего, службе *Web*. Новички часто считают эти понятия синонимами, что не совсем верно.

World Wide Web (WWW, web, веб, Всемирная паутина) — это распределенная информационная система, предоставляющая доступ к гипертекстовым документам по протоколу HTTP (HyperText Transfer Protocol). WWW - это сетевая технология, построенная на клиент-серверной архитектуре и использующая инфраструктуру Интернет для взаимодействия между сервером и клиентом.



Общая схема сетевой программной архитектуры «клиент-сервер»

Серверы WWW (web-серверы) - это хранилища гипертекстовой (в общем случае) информации. Документы, представленные в виде гипертекста, называют *web-страницами.*

Основу любой Web-страницы составляет *гипертекст*, то есть текст, в котором содержатся гиперссылки. Они связывают информацию, вызывая рисунки, фотографии, аудио- и видеовставки. *Гиперссылка* позволяет одним щелчком мыши перейти от одного места просматриваемого документа к другому так же легко, как и к документу, размещенному на сервере в другом конце света. Каждый документ обычно Связан со следующим и т.д. Ссылки, как нити паутины, соединяют между собой огромное количество документов на всем земном шаре. Ссылки могут быть перекрестными, то есть на один документ можно ссылаться из разных первоначальных источников. Именно перекрестные ссылки между документами превращают гигантские массивы информации, накапливаемой в Сети, в единую информационную систему.

Для пользователя *гиперссылка* внешне выглядит как выделенный фрагмент текста (обычно подчеркнутый) или рисунок, с которым связан адрес другого Web-документа. Некоторые рисунки-ссылки объединяются в большое изображение, образуя графические карты. Указатель мыши при попадании на гиперссылку любого вида меняет свою форму. Щелчок по гиперссылке левой кнопкой мыши приводит к загрузке новой затребованной страницы.

Ссылок на страницах так много, и переходы по ним бывают так непредсказуемы, что необходим навык, чтобы не заблудиться в многообразии Web-документов. Для этого браузер имеет кнопки Forward (Вперед) и Back (Назад), которые позволяют вернуться на недавно

просмотренную нужную страницу.

Основная функция Интернета состоит в установлении и обслуживании соединений подключенных к сети пользователей. За установление соединения и управление установленным сеансом «отвечает» протокол TCP. Чтобы такое соединение было возможным, у каждого пользователя имеется свой уникальный *IP – адрес*, который присваивается каждой точке подключения к сети и состоит из 4 компонент — чисел в диапазоне от 0 до 255 (каждое из чисел записывается в отдельный байт, а общая длина адреса составляет четыре байта).

IP-адреса однозначно определяют компьютеры, обменивающиеся данными по сети, но человеку неудобно пользоваться ими. Конечно, легче воспринимать не код, а название. Для распространения такого удобства в сети Интернет была введена *служба адресации*, преобразующая имена доменов в IP-адреса.

Доменным адресом называется последовательность нескольких слов, разделенных точками. Такая форма адреса легче запоминается, поскольку позволяет зашифровывать в себе ассоциативные связи с адресатом.

Доменные адреса преобразуются в IP-адреса специальными программами — *серверами службы DNS* (Domain Name Service, служба доменных имен). Как правило, такие серверы функционируют на компьютерах провайдеров услуг Интернета или компьютерах, отвечающих за организацию работы интрасети. Доменные адреса хранятся на серверах DNS в форме распределенной базы данных. При попытке связаться с информационным ресурсом по доменному имени последнего, это имя пересылается серверу DNS вместе с IP-адресом компьютера пользователя. Получивший доменный адрес DNS-сервер просматривает свою базу данных и возвращает компьютеру пользователя искомый IP-адрес ресурса. Весь процесс происходит без участия пользователя и незаметно для него.

Протоколы Интернета

Протокол – это набор соглашений и правил, определяющих порядок обмена информацией в компьютерной сети.

Протокол TCP/IP (1974)

- **TCP (*Transmission Control Protocol*)**
  - файл делится на пакеты размером не более 1,5 Кб
  - пакеты передаются независимо друг от друга
  - в месте назначения пакеты собираются в один файл
- **IP (*Internet Protocol*)**
  - определяет наилучший маршрут движения пакетов

Протокол FTP - Услуги по передаче клиенту файлов произвольного формата.

Протокол SMTP P0P3 - Услуги по передаче почтовых сообщений в форматах электронной почты.

Протокол HTTP - Услуги по передаче данных в форматах гипертекста и офисных программ MS Office.

Протокол SQL SRV - Услуги по передаче запросов к базе данных на языке запросов SQL и результатов их обработки в СУБД.

Организация передачи данных в Интернете

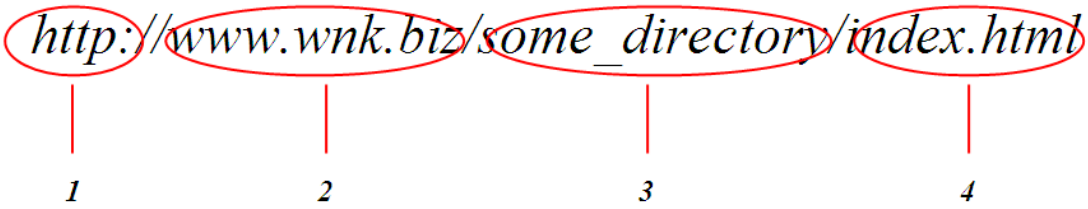
Службой или услугой Интернета называется совокупность протокола передачи данных типа, характерного для указанной службы, программного обеспечения клиента службы и сервера службы, а также информационных ресурсов этой службы. Службы соответствуют различным видам информационного сервиса, который может получать пользователь сети. Службы и соответствующие протоколы обмена, появились в Интернете в разное время и развивались по-разному. Однако у них есть и общие черты. Главной из них является использование сетевой программной архитектуры клиент-сервер.

В основе программной *архитектуры клиент-сервер* лежит взаимодействие двух программ, работающих в общем случае на различных компьютерах сети: клиентской программы (в просторечии клиент) и серверной программы (сервер).

Клиентская программа, взаимодействуя с пользователем через удобный интерфейс, формирует запрос к серверу в соответствии с требованиями сетевого протокола используемой службы. Затем этот запрос по установленному с помощью протоколов TCP/IP сетевому соединению направляется серверной программе. Серверная программа ставит запросы пользователей запросы в очередь и обслуживает их. В результат обслуживания поступившего запроса формируется ответ (в большинстве случаев содержащий запрошенные данные), который и отправляется обратно клиенту. Получив и собрав данные ответа, клиентская программа отображает их на экране компьютера пользователя или сохраняет их на его накопителе.

Возможность организовывать гиперссылки появляется благодаря тому, что каждый Web-документ, равно как и другие файлы-ресурсы, имеет свой адрес, который указывает его координаты в сетевом пространстве. В последних спецификациях он называется URI (Universal Resource Identificator) — Универсальный Идентификатор Ресурсов.

Структуру URI проще разобрать на примере:



1. Адрес читается слева направо. Первая часть адреса — http: / / — указывает тип сервиса, которым вы хотите воспользоваться. В частности http: // указывает на протокол HTTP, который использует Web-сервис. Могут встречаться и другие протоколы, например, mail to:// — обращение к электронной почте. Заметим, что в Интернете применяются соглашения операционной системы Unix, в которой для разделения каталогов используется косая черта «/» (слеш), в отличие от DOS, где при указании пути употребляется «обратный» слеш «\».

2. Вторая часть адреса, www.wnk.biz/, является доменным именем сервера. Особенности образования этих имен приведены ниже.

3. Третья часть, some_directory/, указывает путь (каталог some_directory/) к файлу на сервере, который вы собираетесь открывать.

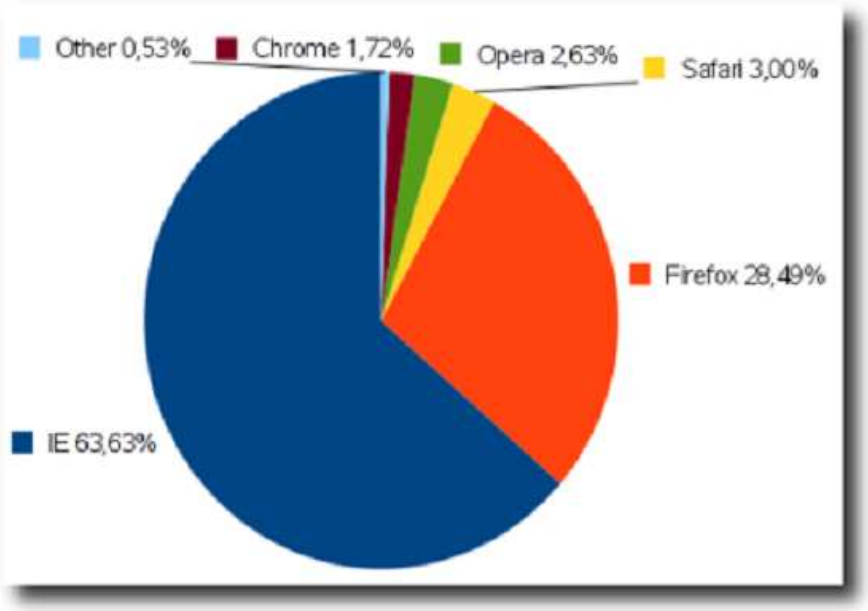
4. Последним (index.html) следует имя самого файла - index, и его расширение -html. Если имя файла отсутствует, то сервер открывает страницу, заданную по умолчанию. Часто такая страница называется index.html, или index.htm, или default.htm. Она служит начальной страницей, на которой есть гиперссылки, ведущие ко всем другим значимым страницам этого сайта.

Несколько web-страниц, объединенных общей тематикой, оформлением, связанных гипертекстовыми ссылками и обычно находящихся на одном и том же web-сервере, называются *web-сайтом*. По данным за октябрь 2009 г. (http://netcraft.com) во Всемирной паутине насчитывалось более 230 млн. сайтов, размещенных на миллионах web-серверов. Для загрузки и просмотра информации с web-сайтов используются специальные программы — браузеры.

Браузер - это программа, способная понимать гипертекст и отображать содержимое веб-страниц. Термин «браузер» прочно укоренился в компьютерной среде. Само слово происходит от английского browser, что в переводе означает «просмотрщик», то есть программа для просмотра. В русскоязычных версиях операционной системы Windows фигурирует официальный (не особо удачный) термин «обозреватель», который плохо приживается.

Браузер, веб-обозреватель (web-browser) — это клиентское программное обеспечение для доступа к веб-серверам по протоколу HTTP и просмотра веб-страниц. Как правило, браузеры дополнительно

поддерживают и ряд других протоколов (например ftp, file, mms, pop3).



Статистика браузеров за период 07.2008 г - 10.2010 г.

Наибольшую распространенность в мире имеют четыре браузера (с появлением Google Chrome их стало пять): Internet Explorer (Microsoft), Firefox (Mozilla Foundation), Safari (Apple) и Opera (Opera Software ASA) *Internet Explorer* (IE) - браузер, разработанный компанией Майкрософт, является наиболее распространенным в мире. Это связано с тем, что IE тесно интегрирован с ОС Windows и поставляется вместе с ней. Платформозависим (поддержка сторонних ОС прекращена, начиная с версии 5). Единственный браузер, напрямую поддерживающий технологию ActiveX. Не полностью совместим со стандартами W3C, в связи с чем требует дополнительных затрат от веб-разработчиков.

Firefox - свободный кроссплатформенный браузер, разрабатываемый Mozilla Foundation и распространяемый под тройной лицензией GPL/LGPL/MPL. В основе браузера - движок Gekko, который изначально создавался для Netscape Communicator. Однако, вместо того, чтобы предоставить все возможности движка в стандартной поставке, Firefox реализует лишь основную его функциональность, предоставляя пользователям возможность модифицировать браузер в соответствии с их требованиями через поддержку расширений (add-ons), тем оформления и плагинов.

Такая лёгкая расширяемость достигается, в основном, за счёт использования в интерфейсе специально разработанного исключительно для Gecko языка разметки XUL и стандартных JavaScript и CSS. Это иногда приводит к более медленной работе интерфейса и повышенным требованиям к оперативной памяти. Для тех, кому нужны функции движка Gecko, а не расширяемый интерфейс, существуют сторонние браузеры, в которых интерфейс реализован по-другому; например,

Camino, SeaMonkey, Epiphany и Kazehakase.

Safari - проприетарный браузер, разработаный корпорацией Apple и входящий в состав операционной системы Mac OS X. Бесплатно распространяется для операционных систем семейства Microsoft Windows. В браузере используется уникальный по производительности интерпретатор JavaScript, опережающий аналоги более чем в 2 раза. Среди других особенностей: функция «Snapback», которая позволяет мгновенно вернуться к исходным результатам поиска или к верхнему уровню любого веб-сайта; те же графические технологии, что и в Mac OS X; предварительная поддержка стандартов CSS3 и HTML 5; автоматическое распознавание и подгрузка отсутствующих в системе шрифтов; интеграция мультимедийных технологий QuickTime.

Google Chrome — браузер с открытым исходным кодом, разрабатываемый компанией Google. Первая стабильная версия вышла 11 декабря 2008 года. В отличие от многих других браузеров, в Chrome каждая вкладка является отдельным процессом. В случае если процесс обработки содержимого вкладки зависнет, его можно будет завершить без риска потери данных других вкладок. Еще одна особенность - интеллектуальная адресная строка (Omnibox). К возможности автозаполнения она добавляет поисковые функции с учетом популярности сайта, релевантности и пользовательских предпочтений (истории переходов). Планируется кроссплатформенность.

Opera — кроссплатформенный многофункциональный веб-браузер, разработанный в 1994 году группой исследователей из норвежской компании Telenor. Дальнейшая разработка ведется Opera Software ASA.

Браузер Opera обладает высокой скоростью работы и совместим с основными стандартами WWW. Отличительными особенностями Opera долгое время являлись многостраничный интерфейс (система вкладок) и возможность масштабирования веб-страниц целиком, вместе с графикой. Функция Small Screen Rendering позволяет переформатировать документ для отображения на экранах мобильных устройств. В Opera, кроме стандартных способов навигации предусмотрены так называемые «жесты мышью». На разных этапах развития в Opera были интегрированы возможности почтового/новостного клиента Opera Mail, адресной книги, клиента сети BitTorrent, агрегатора RSS, клиента IRC, менеджера закачек, WAP-браузера, а также поддержка виджетов — графические модулей, работающих вне окна браузера.

Специальные редакции, Opera Mobile и Opera Mini, существенно опережают прочие браузеры по числу установок на мобильные телефоны. Opera также единственный браузер для игровых систем Nintendo DS и Wii. Движок Opera (Presto) лицензирован Adobe и интегрирован в пакет для веб-разработчиков Adobe Creative Suite.

До 2005 года Opera распространялся в виде ad-ware, а с 2005 года версии Opera для ПК и устройств мобильной связи (кроме Opera Mobile) доступны бесплатно.

Вопрос 2. Классификация веб-сайтов.

Веб-сайт (от англ. website: web — «паутина», «сеть» и site — «место», букв. «место в сети») или просто сайт — в компьютерной сети объединённая под одним адресом (доменным именем или IP-адресом) совокупность электронных документов (файлов) частного лица или организации. По умолчанию подразумевается, что сайт располагается в сети Интернет. Все веб-сайты в совокупности составляют Всемирную паутину. Для прямого доступа клиентов к веб-сайтам на серверах был специально разработан протокол HTTP. Веб-сайты иначе называют Интернет-представительством человека или организации. Когда говорят «своя страничка в Интернет», то подразумевается целый веб-сайт или личная страница в составе чужого сайта. Кроме веб-сайтов в сети Интернет также доступны WAP-сайты для мобильных телефонов.

Изначально веб-сайты представляли собой совокупность статических документов. В настоящее время большинству из них свойственна динамичность и интерактивность. Для таких случаев специалисты используют термин *веб-приложение* — готовый программный комплекс для решения задач веб-сайта.

В большинстве случаев в Интернете одному веб-сайту соответствует одно доменное имя. Именно по доменным именам сайты идентифицируются в глобальной сети. Возможны иные варианты: один сайт на нескольких доменах или несколько сайтов под одним доменом. Обычно несколько доменов используют крупные сайты (веб-порталы) чтобы логически отделить разные виды предоставляемых услуг (mail.google.com, news.google.com, maps.google.com). Нередки и случаи выделения отдельных доменов для разных стран или языков. Например, google.ru и google.fr логически являются сайтом Google на разных языках, но технически это разные сайты. Объединение нескольких сайтов под одним доменом характерно для бесплатных хостингов.

Аппаратные сервера для хранения веб-сайтов называются веб-серверами. Сама услуга хранения называется веб-хостингом. Раньше каждый сайт хранился на своём собственном сервере, но с ростом Интернета технологическим улучшением серверов на одном компьютере стало возможно размещение множества сайтов (виртуальный хостинг). Сейчас сервера для хранения только одного сайта называются выделенными.

Один и тот же сайт может быть доступен по разным адресам и хранится на разных серверах. Копия оригинального сайта в таком случае называется зеркалом. Существует также понятие оффлайновая версия

сайта — это копия сайта, которая может быть просмотрена на любом компьютере без подключения к компьютерной сети и использования серверного программного обеспечения (ПО).

В настоящее время веб-сайты классифицируются по следующим критериям:

1. По доступности сервисов:

Открытые — все сервисы полностью доступны для любых посетителей и пользователей.

Полуоткрытые — для доступа необходимо зарегистрироваться (обычно бесплатно).

Закрытые — полностью закрытые служебные сайты организаций (в том числе корпоративные сайты), личные сайты частных лиц. Такие сайты доступны для узкого круга людей. Доступ новым людям обычно даётся через т. н. инвайты (приглашения).

2. По природе содержимого:

Статические — всё содержимое заранее подготавливается. Пользователю выдаются файлы в том виде, в котором они хранятся на сервере.

Динамические — содержимое генерируется специальными скриптами (программами) на основе других данных из любого источника.

3. По физическому расположению:

Внешние сайты сети Интернет.

Локальные сайты — доступны только в пределах локальной сети. Это могут быть как корпоративные сайты организаций, так и сайты частных лиц в локальной сети провайдера.

4. По схеме представления информации, её объёму и категории решаемых задач можно выделить следующие типы веб-ресурсов:

Интернет-представительства владельцев бизнеса (торговля и услуги, не всегда связанные напрямую с Интернетом):

Сайт-визитка — содержит самые общие данные о владельце сайта (организация или индивидуальный предприниматель). Вид деятельности, история, прайс-лист, контактные данные, реквизиты, схема проезда. Специалисты размещают своё резюме. То есть подробная визитная карточка.

Представительский сайт — так иногда называют сайт-визитку с расширенной функциональностью: подробное описание услуг, портфолио, отзывы, форма обратной связи и т. д.

Корпоративный сайт — содержит полную информацию о компании-владельце, услугах/продукции, событиях в жизни компании. Отличается от сайта-визитки и представительского сайта полнотой представленной информации, зачастую содержит различные

функциональные инструменты для работы с контентом (поиск и фильтры, календари событий, фотогалереи, корпоративные блоги, форумы). Может быть интегрирован с внутренними информационными системами компании-владельца (КИС, CRM, бухгалтерскими системами). Может содержать закрытые разделы для тех или иных групп пользователей — сотрудников, дилеров, контрагентов и пр.

Каталог продукции — в каталоге присутствует подробное описание товаров/услуг, сертификаты, технические и потребительские данные, отзывы экспертов и т. д. На таких сайтах размещается информация о товарах/услугах, которую невозможно поместить в прайс-лист.

Интернет-магазин — веб-сайт с каталогом продукции, с помощью которого клиент может заказать нужные ему товары. Используются различные системы расчётов: от пересылки товаров наложенным платежом или автоматической пересылки счета по факсу до расчётов с помощью пластиковых карт.

Промо-сайт — сайт о конкретной торговой марке или продукте, на таких сайтах размещается исчерпывающая информация о бренде, различных рекламных акциях (конкурсы, викторины, игры и т. п.).

Сайт-квест — Интернет-ресурс, на котором организовано соревнование по разгадыванию последовательности взаимосвязанных логических загадок.

Информационные ресурсы:

Тематический сайт — веб-сайт, предоставляющий специфическую узкотематическую информацию о какой-либо теме.

Тематический портал — это очень большой веб-ресурс, который предоставляет исчерпывающую информацию по определённой тематике. Порталы похожи на тематические сайты, но дополнительно содержат средства взаимодействия с пользователями и позволяют пользователям общаться в рамках портала (форумы, чаты) — это среда существования пользователя.

Веб-сервис — услуга созданная для выполнения, каких либо задач в рамках сети WORLD WIDE WEB.

Тема 2. Правила построения простейших HTML-документов.

1. *Общие понятия языка разметки гипертекста – HTML*
2. *Структура HTML-документа*
3. *Ввод и оформление текста*
4. *Создание списков*
5. *Создание ссылок*

Вопрос 1. Общие понятия языка разметки гипертекста – HTML.

Для создания простейших web-страниц часто используется язык гипертекстовой разметки HTML. Существуют общие правила записи *HTML-документов* и принципы, используемые при создании сайтов.

Конечно, каждый сайт индивидуален, но существуют общие правила языка *HTML*, которым нужно обязательно следовать – только в этом случае HTML-код будет верно распознаваться и отображаться браузером.

Основой языка HTML является *элемент*. Он несет в себе определенную информацию, может описывать документ в целом или способ форматирования текста. Элементы можно назвать основой построения сайта.

Название элемента помещается в угловые скобки, например *<Имя тега>*, - полученное выражение называют тегом. В языке HTML существует два типа тегов – одиночные и парные теги, парные теги в свою очередь подразделяют на открывающий и закрывающий. В основном парные теги используются при форматировании текста, они задают начало и конец блока форматирования (открывающий тег включает форматирование, а закрывающий выключает). При этом основными отличиями в записи закрывающих тегов является указание символа « / » после угловой скобки перед именем тега и отсутствие атрибутов:

<Имя тега> …… </ Имя тега>

Иногда закрывающий тег вообще не требуется, а иногда его можно пропустить, однако для корректной обработки документа рекомендуется всегда использовать закрывающий тег. По назначению элемента зачастую можно догадаться, требуется ли ему закрывающий тег.

Элементы применяются для того, чтобы сказать браузеру, какой блок вы хотите видеть в определенном месте страницы, а также какую информацию этот блок должен содержать. Кроме того, браузеру нужно сообщить, как отображать эту информацию. Для этого используют *атрибуты* элементов.

С помощью атрибутов можно указывать различные способы отображения информации, добавляемой с помощью одинаковых элементов, а в некоторых случаях применение элемента без атрибутов не дает результатов.

Значения атрибутов задаются после знака равенства в кавычках. Некоторым атрибутам присущ набор фиксированных значений, для других количество значений не ограничено.

<Имя тега Атрибут = “значение атрибута”> … … … </ Имя тега>

Вопрос 2. Структура HTML-документа.

Элементы и их атрибуты являются основой языка HTML, но для правильного отображения страницы в браузерах еще важно верно создать структуру документа. Для этого существуют строгие правила. Есть элементы, без которых HTML-документ не может обойтись, потому что именно они определяют его структуру.

Для создания структуры документа и хранения служебной

информации в нем предусмотрено много элементов, которые охватывают все необходимые пункты построения документа. Визуально структуру HTML-документа можно разбить на три основные части (рис. 1):

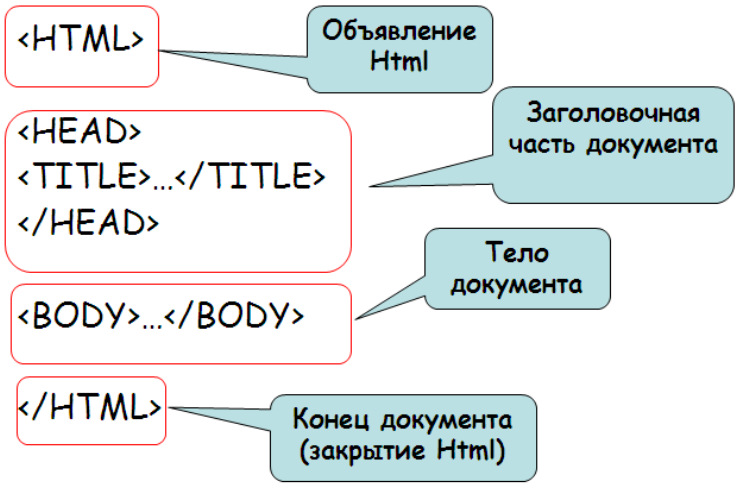


Рис. 1. Структура HTML-документа

— *Строка объявления типа документа (объявление HTML)* – эта строка дает браузеру общую информацию об HTML-документе;

— *Декларативный заголовок (заголовочная часть)*. Информация, вводимая в элемент HEAD, не отображается в окне браузера, а помогает ему в обработке страницы. В заголовке должны присутствовать как открывающий, так и закрывающий теги < HEAD > и </ HEAD >, между которыми располагаются другие элементы, несущие служебную информацию о странице. Элементы, находящиеся внутри элемента HEAD играют очень важную роль: данные, содержащиеся в них, помогают браузеру в обработке страницы, а поисковым системам – в индексации документа. Элемент TITLE, располагающийся в элементе HEAD, задает название страницы, которое будет отображаться в строке заголовка окна браузера. Элемент TITLE требует наличия закрывающего тега. Некоторые поисковые системы используют текст, содержащийся в этом элементе, для поиска и выводят его в качестве заголовка результата поиска, поэтому корректно и качественно составленное заглавие может привлечь посетителей на сайт;

— *Тело документа (содержимое HTML-страницы)*. Внутри элемента BODY располагается сам документ: весь текст, находящийся между открывающим тегом и закрывающим тегом будет отображаться браузером. Все элементы, отвечающие за форматирование документа, помещают внутрь элемента BODY. Атрибуты элемента BODY применяются для того, чтобы установить общие для всего документа свойства, и в этом отношении возможности данного

элемента достаточно широкие: задать цвет ссылок, цвет фона, цвет текста и т.д. (см. табл. 1).

Таблица 1. Атрибуты тега BODY

ALINK	Определяет цвет активной ссылки.
BACKGRQUND	Указывает на URL-адрес изображения, используемого в качестве фонового.
BGCOLOR	Определяет цвет фона документа
BGPRQPERTIES	Если установлено значение FIXED, фоновое изображение не прокручивается.
LEFTMARGIN	Устанавливает границу левого поля в пикселях.
LINK	Определяет цвет еще не просмотренной ссылки.
TEXT	Определяет цвет текста.
TOPMARGIN	Устанавливает границу верхнего поля в пикселях.
VLINK	Определяет цвет уже просмотренной ссылки.

Вопрос 3. Ввод и оформление текста.

Ввод текстовой информации на сайт осуществляется внутри элемента BODY. Однако простое расположение текста внутри элемента BODY недостаточно, его необходимо оформлять средствами языка HTML, например, разделять на абзацы или создавать заголовки. Для оформления текста HTML предоставляет много возможностей: можно менять цвет, размер и шрифт текста.

Сначала рассмотрим общие принципы структурного форматирования текста, а затем перейдем непосредственно к управлению его внешним видом.

Создание заголовков

Заголовки – важный элемент сайта, они помогают систематизировать текст. В HTML можно создавать заголовки шести уровней.

Заголовки создаются с помощью элементов H1, H2, H3, H4, H5, H6, например:

<H1>……..</H1>
<H2>……..</H2>

<H3>……..</H3>
<H4>……..</H4>
<H5>……..</H5>
<H6>……..</H6>

Самым крупным считается заголовок первого уровня, а самым мелким – шестого (см. рис. 2).

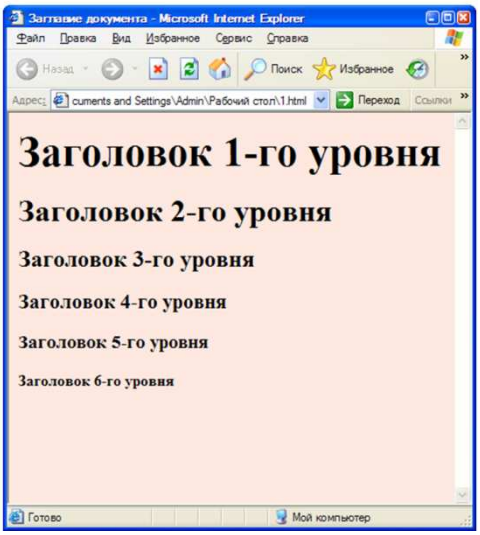


Рис. 2. Заголовки в HTML

Для заголовка любого уровня можно задать выравнивание по горизонтали. Это делается при помощи атрибута *Align*. Значения атрибута *Align* приведены в таблице 2.

Таблица 2. Значения атрибута Align

Значение	**Функция**
LEFT	Выравнивание текста по левой границе окна браузера.
CENTER	Выравнивание по центру окна браузера.
RIGHT	Выравнивание по правой границе окна браузера.
JUSTIFY	Выравнивание по ширине окна браузера (только для заголовков длиннее строки).

Создание абзацев

Абзацы делят текст на логические части и на письме выделяются отступом от края листа. В HTML абзацы отделяются друг от друга расстоянием в одну строку.

Для организации абзацев в HTML предусмотрен элемент *P*, который подразумевает наличие закрывающего тега.

Для элемента *P* также можно задать атрибут *Align*, определяющий выравнивание. Значения атрибута приведены в табл. 2.

Создание обрывов строк

В HTML есть возможность перенести текст на новую строку, не заканчивая абзац. Обычно браузер переносит слова в зависимости от размера окна, и возможность самостоятельно определить место переноса может пригодиться при записи стихов или для отделения различных элементов друг от друга.

Для переноса текста на новую строку служит элемент *BR*, он не требует закрывающего тега.

Форматирование текста

В HTML есть много элементов для форматирования текста, и все они делятся на две группы: логические и физические.

Логические элементы сообщают браузеру о том, какой тип информации в них содержится, например важный текст или цитата. Браузер сам решает, как отобразить такой текст. По сути, логические элементы разбивают документ на логические части и при этом не обязывают браузер отображать текст, расположенный внутри элемента, каким-либо конкретным образом. Некоторые из логических элементов могут вообще не изменять отображение текста, поэтому при их рассмотрении будем делать упор на то, как они определяют значение текста, а не на то, как они его форматируют. Элементы логического форматирования представлены в таблице 3, и требуют наличия закрывающего тега.

Таблица 3. Элементы логического форматирования

<CITE>	Используется для выделения цитат или названий книг и статей, при этом текст обычно выводится курсивом.
<CODE>	Применяется для вывода небольшого куска программного кода шрифтом фиксированной ширины.
<EM>	Этот элемент обычно используется для выделения важных фрагментов текста. Браузеры обычно отображают такой текст курсивом.
<KBD>	Элемент, выделяющий шрифтом фиксированной ширины текст, вводимый пользователем с клавиатуры.
<SAMP>	Используется для выделения нескольких символов шрифтом фиксированной ширины.
<STRONG>	Этот элемент обычно используется для выделения важных фрагментов текста. Браузеры обычно отображают такой текст полужирным шрифтом.
<VAR>	Используется для отметки имен переменных. Обычно такой текст отображается курсивом.

Физические элементы для форматирования текста сообщают браузеру, как должен выглядеть текст, расположенный внутри элемента. Все элементы физического форматирования представлены в таблице 4, и

требуют наличия закрывающего тега.

Таблица 4. Элементы физического форматирования

<B>	Выделяет текст полужирным шрифтом.
<I>	Выделяет текст курсивом.
<TT>	Выводит текст шрифтом фиксированной ширины.
<U>	Элемент подчёркивания.
<STRIKE>	Элемент зачеркивания. Отображается текст, перечеркнутый горизонтальной линией.
<BIG>	Выводит текст шрифтом большего размера.
<SMALL>	Выводит текст шрифтом меньшего размера.
<SUB>	Сдвигает текст ниже уровня строки и выводит его (если возможно) шрифтом меньшего размера.
<SUP>	Сдвигает текст выше уровня строки и выводит его (если возможно) шрифтом меньшего размера.

Язык HTML позволяет вкладывать элементы форматирования друг в друга. При этом их действия суммируются. Если вложить в элемент B элемент I, то получится текст, написанный полужирным курсивом. При этом нужно следить за правильным закрытием элементов: тот, что открыт раньше закрывается позже, например:

<B> <I> Полужирный курсив </I> </B>

Рассмотрим пример использования тегов физического форматирования:

```
<HTML>
<HEAD>
<TITLE>Заглавие документа</TITLE>
</HEAD>
<body bgcolor="BBE0E3" text="black">
Выделение текста <B>полужирным шрифтом</B> <BR>
Выделение текста <I>Курсив</I> <BR>
Выделение текста <U>подчеркиванием</U>  <BR>
Можно написать текст,<TT> как на пишущей машинке </TT>  <BR>
Можно создать <SUP> надстрочный текст </SUP>
или даже <SUB> подстрочный </SUB> <BR>
Текст можно <S> перечеркнуть </S>
</body>
</HTML>
```

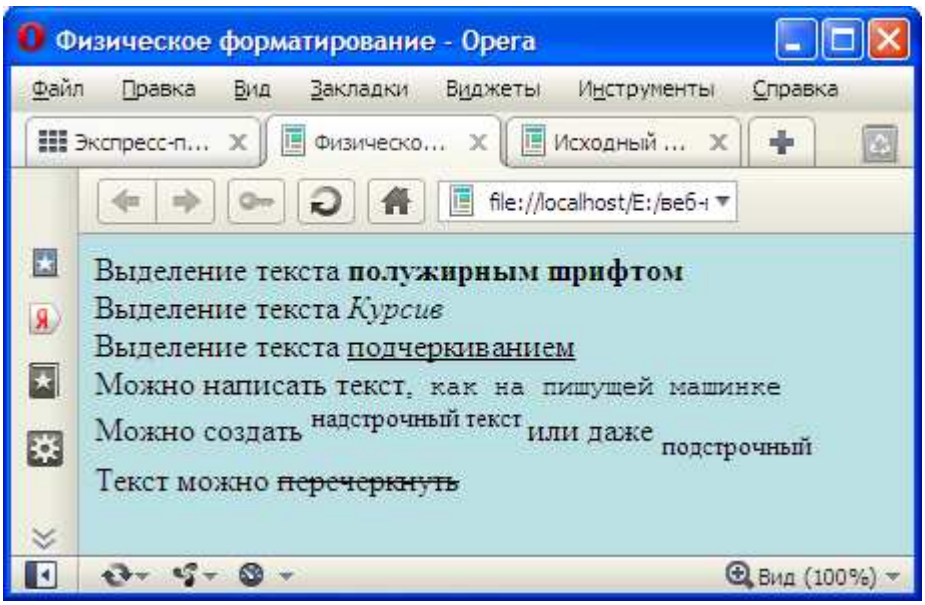


Рис. 3. Физическое форматирование

Элемент PRE

В том случае, когда у вас уже есть отформатированный нужным образом текст, т.е. с сохранением всех пробелов и переносов строк, и вы не хотите ничего менять используется тег *PRE*. Например:

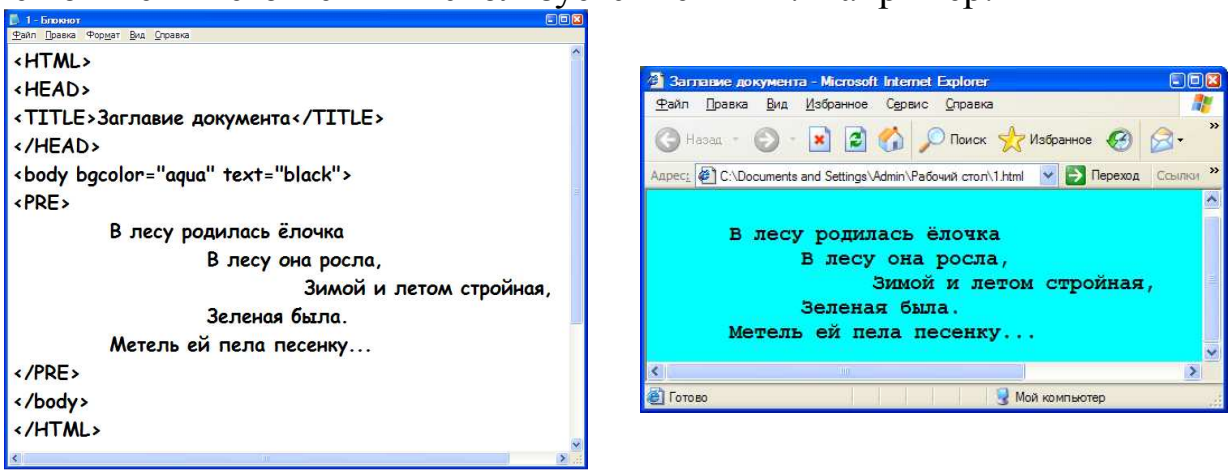


Рис. 4. Использование элемента PRE

Элемент FONT

Элемент *FONT* задает параметры шрифта для текста. Параметры текста задаются с помощью атрибутов элемента FONT. Можно определить шрифт, размер и цвет текста, расположенного внутри него. Элемент FONT должен обязательно содержать хотя бы один из трех атрибутов:

COLOR – задает цвет текста;

FACE – задает гарнитуру шрифта;

SIZE – определяет размер шрифта в относительных единицах от 1 до 7.

Тэг <BASEFONT>

Наряду с элементом FONT используется идентичный ему элемент <*BASEFONT*>, в котором предусмотрены все те же атрибуты – *COLOR*,

FACE, SIZE, разница заключается только, в том, что <BASEFONT> предназначен для всего документа, а FONT – для отдельного фрагмента документа. Например:

```
<HTML>
<HEAD>
<TITLE>Заголовок документа</TITLE>
</HEAD>
<body bgcolor="#BBE0E3" text="000000">
<BASEFONT size="5" face="MS Sans Serif" color="FF0080">
Пример форматирования текста<br>
<FONT size="6" face="Comic Sans MS" color="008040">
Всем привет!!! - это форматирование отдельного фрагмента текста
</FONT><br>
Возврат к предыдущим настройкам шрифта
</body>
</HTML>
```

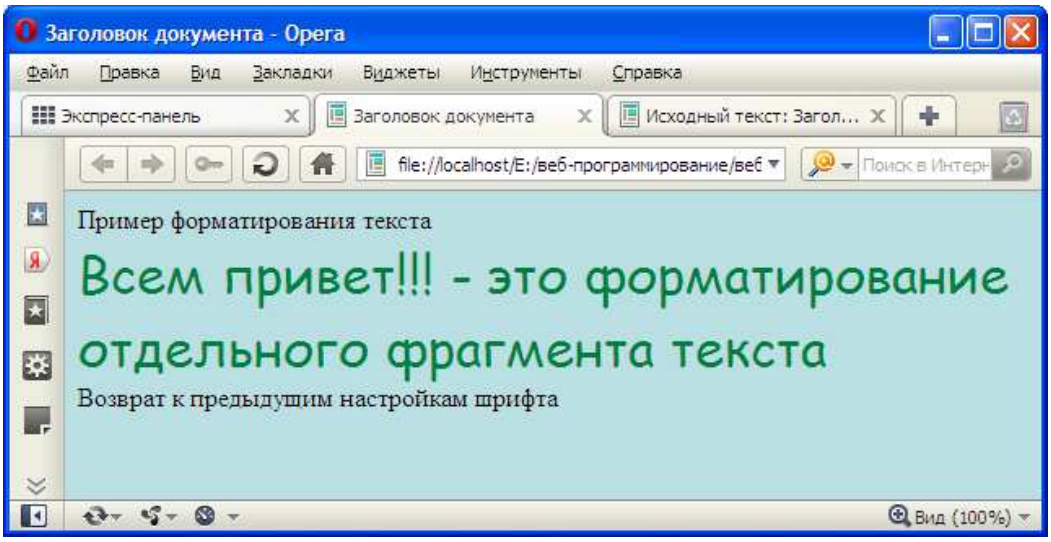


Рис. 5. Использование элементов FONT и BASEFONT

Вопрос 4. Создание списков

Списки – важный инструмент, они применяются для организации и группировки данных. Это может пригодиться при создании карты сайта (т.е. его оглавления), описания сложных структур и других подобных объектов.

В HTML можно выделить несколько типов списков, которые отличаются по типам представления информации:

- Маркированный;
- Нумерованный;
- Список определений;
- Смешанные списки.

Маркированные списки – это списки, в которых пункты отмечаются с помощью различных символов. Такие списки еще называют

ненумерованными, или неупорядоченными, потому что для элементов данного списка последовательность неважна. Эти списки можно использовать для простого перечисления объектов или их свойств.

Для создания маркированных списков в HTML предусмотрен элемент *
...
*. Каждый следующий элемент списка вкладывается в отдельный контейнер *<li>*. Закрывающий тег подразумевается, но в HTML может быть опущен, поскольку однозначно восстанавливается по наличию следующего элемента или по концу списка.

У элемента *ul* есть атрибут *TYPE*, определяющий вид маркера списка (значения атрибута приведены в табл. 5). Например:

```
<UL TYPE=“square”>
    <LI>Раз</LI>
    <LI>Два</LI>
    <LI>Три</LI>
    <LI>Четыре</LI>
    <LI>Пять</LI>
</UL>
```

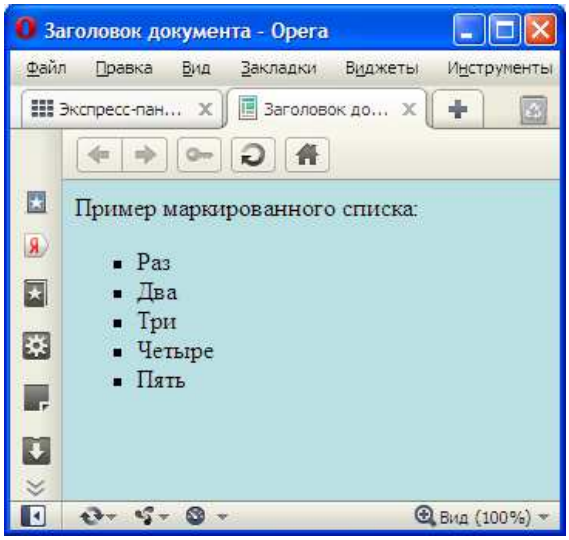


Рис. 6. Маркированный список

Таблица 5. Значения атрибута TYPE в теге UL

Значение	**Пример**
circle	○
disk	●
square	■

При создании маркированных списков с помощью элемента *LI* можно задать вид маркера отдельно для каждого пункта списка. Например:

```
<UL>
<LI  TYPE =” circle”> Раз </LI>
```

<LI TYPE =" circle"> Два </LI>
<LI TYPE =" square"> Три </LI>
<LI TYPE =" circle"> Четыре </LI>
<LI TYPE =" disk"> Пять </LI>
</UL>

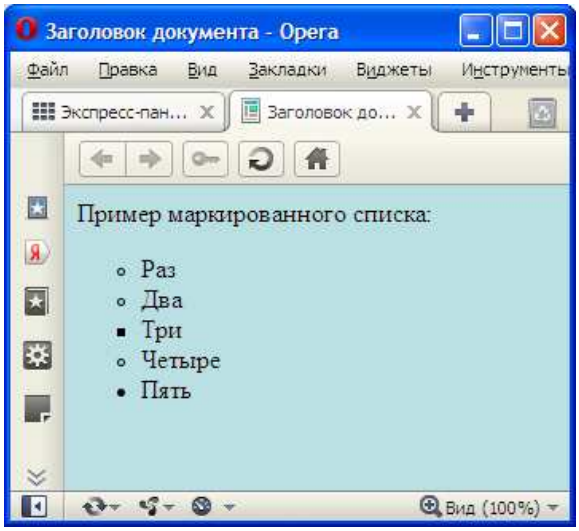


Рис. 7. Список с разными маркерами

Нумерованные списки применяются, когда порядок следования пунктов списка имеет большое значение, например, при описании алгоритмов или других пошаговых действий. Особенностью этого типа является то, что все их элементы упорядочены. Для создания упорядоченных списков применяется элемент OL, которому требуется наличие закрывающего тега, а все пункты списка находятся внутри этого элемента. Каждый следующий элемент списка вкладывается в отдельный контейнер *<li>*. Закрывающий тег подразумевается, но в HTML может быть опущен, поскольку однозначно восстанавливается по наличию следующего элемента или по концу списка. Например:

<OL>
<LI>Раз</LI>
<LI>Два</LI>
<LI>Три</LI>
<LI>Четыре</LI>
<LI>Пять</LI>
</OL>

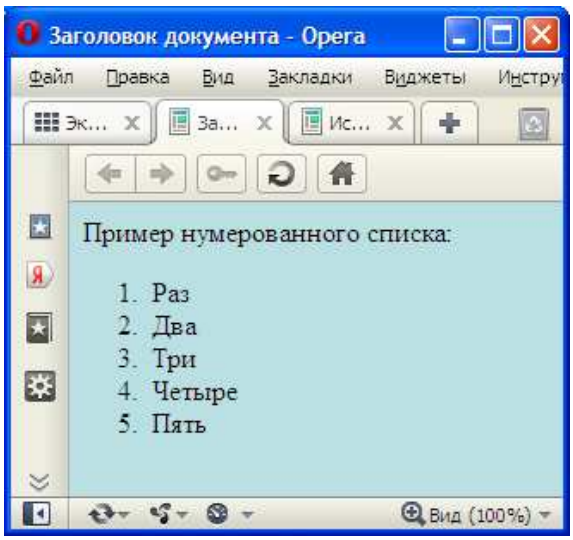


Рис. 8. Нумерованный список

Атрибут *TYPE* тега <ol> позволяет изменить стиль нумерации (значения атрибута приведены в табл. 6).

Таблица 6. Значения атрибута TYPE в теге OL

Значение	Пример
A	A, B, C
a	a, b, c
I	I, II, III
i	I, ii, iii
1	1, 2, 3

Например:

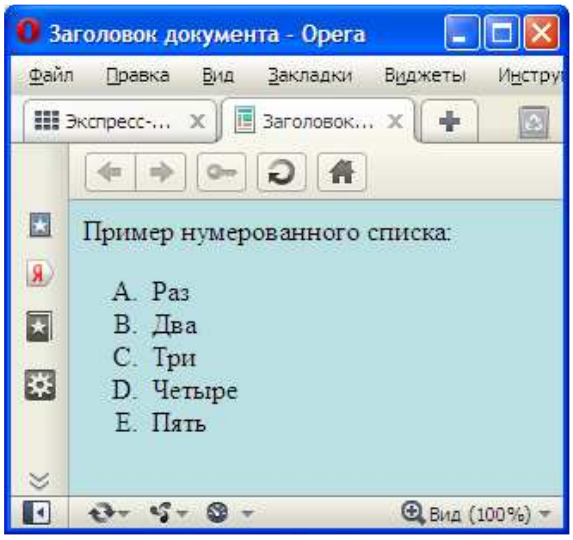


Рис. 9. Использование атрибута TYPE

Как и в примере с маркированными списками, в элементе LI можно

задать разный вид нумерации с помощью атрибута *TYPE*.

Атрибуты *START* и *VALUE* позволяют изменить порядок нумерации списка:

- START – дает возможность присвоить начальному элементу номер, отличный от 1, т.е. указывает с какого числа начинать нумерацию всего списка;
- VALUE – принудительно назначает элементу произвольный номер.

Например:

```
<OL START=4>
<LI>Иванов</LI>
<LI VALUE="10">Петров</LI>
<LI> Сидоров </LI>
</OL>
```

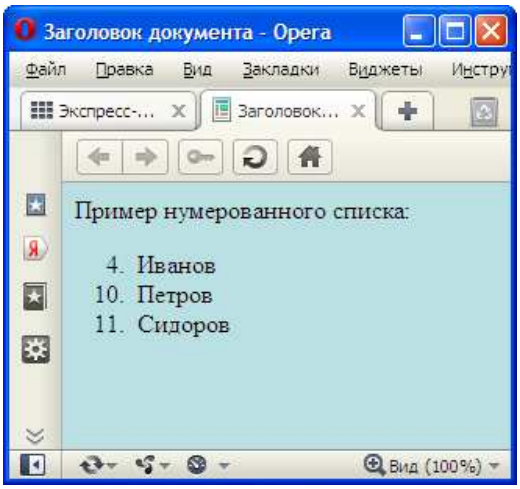


Рис. 10. Использование атрибутов START и VALUE

Бывает, что на сайте нужно создать список терминов или словарь. Это особенно актуально для сайтов узкой направленности. Для создания подобных конструкций служит список определений. Элементы *списков определений* состоят из названия элемента и его описания, поэтому такие списки в какой-то мере напоминают толковые словари.

Для организации списков определений служат элементы:

- <DL> – начало списка
- <DT> - название элемента
- <DD> – определение элемента.

Например:

```
<DL>
<DT>Сервер
<DD>Компьютер, ресурсы которого доступны с рабочих станций сети
<DT>Протокол
<DD>Формальный набор правил, позволяющий обмениваться информацией
<DT>Электронная почта
```

<DD>Технология обмена информацией в электронном виде
</DL>

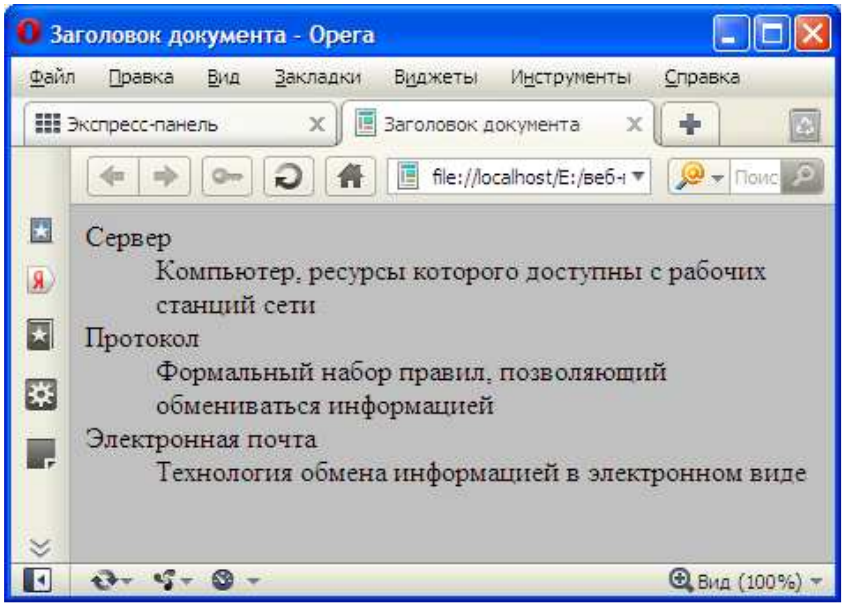


Рис. 11. Списки определений

Смешанные списки - это списки с несколькими уровнями вложенности. В HTML можно комбинировать и вкладывать друг в друга списки разных типов, но при этом нужно внимательно следить за тем, где заканчивается вложенный список, а где список верхнего уровня, иначе будут проблемы с отображением информации (см. рис. 12).

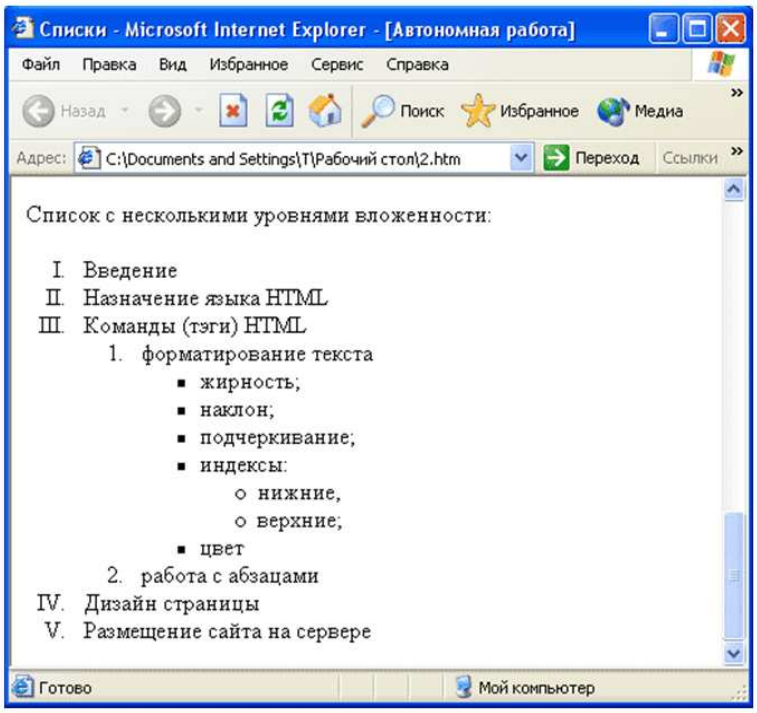


Рис. 12. Пример списка с несколькими уровнями вложенности

Из примера видно, что элементы внутренних списков отступают от элементов списка более высокого уровня.

Вопрос 5. Создание ссылок

По сути, работа в Интернет поставлена на использовании гиперссылок. Ссылки связывают документы, разбросанные по всему Интернету, в одну сеть. Ваш сайт может находиться на разных компьютерах, но для посетителя он будет казаться единым целым, и все это благодаря ссылкам.

Можно выделить два типа ссылок: внешние и внутренние. Первые связывают страницы в один сайт и помогают передвигаться по нему. Вторые помогают передвигаться в рамках одной страницы.

Внешние ссылки

Внешними называют ссылки на объекты, расположенные вне текущей страницы. Это могут быть картинки, другие страницы сайта, мультимедийные приложения. Основой внешних ссылок является URL-адрес объекта, на который вы собираетесь сослаться.

Для создания гиперссылок в HTML служит элемент *A*, который требует наличия закрывающего тега. Внутри элемента располагается текст, который будет выделен как ссылка. Адрес элемента для перехода записывается в элементе *A* в качестве значения атрибута *HREF*. Таким образом, перемещение по сайту становиться удобным, легким и прозрачным для пользователя. Например:

<A HREF=”aboutme.html”> Обо мне </A>

Текст ссылки «Обо мне» отображается подчеркнутым, а указатель мыши меняет вид при наведении на ссылку. Когда посетитель сайта щелкнет кнопкой мыши на ссылке, он перейдет на страницу, которая указана в качестве адреса *”aboutme.html”*.

Внутренние ссылки

Внутренние ссылки организуют переходы внутри одного HTML-документа. Они применяются, когда на одной странице очень много текста. Для простоты навигации можно создать ссылки, при щелчке кнопкой мыши на которых пользователь автоматически перейдет к нужной части документа.

Чтобы создать такую ссылку, сначала нужно определить место, к которому ссылка будет переходить (метка). Это делается при помощи атрибута *NAME* элемента A. Необходимый текст ссылки также заключается в элемент A. Хотя совершенно не обязательно помещать туда текст, можно просто установить теги этого элемента в месте, к которому браузер должен переходить при щелчке кнопкой мыши на ссылке.

В качестве значения атрибута NAME можно взять любое имя, желательно, чтобы оно характеризовало текущее место, т. к. так будет проще пользоваться метками, например:

<A NAME=”metka”> Это метка для перехода </A>

Затем нужно создать ссылку на эту метку. Ссылка на внутреннюю

метку создается так же, как и ссылка на внешний документ, только вместо URL-адреса желаемой страницы надо ввести адрес метки, например в виде:

<Body>
<A NAME=”metka1”> Это метка для перехода 1 </A>
<A HREF=” #metka1”> Щелкните кнопкой мыши для перехода к метке 1 </A>
<A NAME=”metka2”> Это метка для перехода 2 </A>
<A HREF=” #metka2”> Щелкните кнопкой мыши для перехода к метке 2 </A>
</Body>

Внутренние ссылки при отображении ничем не отличаются от внешних, а текст, отмеченный как метка, никак не выделяется.

Внутренние ссылки по своему синтаксису такие же, как и внешние, поэтому атрибуты элемента A применимы практически для обоих типов.

Таблица 7. Атрибуты тега A

href	Имя и путь к странице Переход на ссылку внутри документа #
target	Имя окна, в которое будет загружена ссылка: – blank – открывает ссылку в новом окне браузера – top – в окно стоящее поверх остальных – self – в текущее окно; – parent – открывает ссылку в родительском окне (по умолчанию)
name	Имя метки в документе
title	Задает всплывающую подсказку для ссылки. Значение атрибута – текст всплывающей подсказки.

Тема 3. Добавление изображений и мультимедиа на web-страницы.

1. *Встраивание изображений*
2. *Добавление мультимедийных файлов*
3. *Вставка фоновых изображений и фоновой музыки на веб-страницу*

Вопрос 1. Встраивание изображений.

При создании сайта невозможно обойтись без использования активных объектов вроде картинок, видео или Flash-анимации. Однако неграмотное использование мультимедиа на странице способно погубить даже самые лучшие и интересные сайты. Поэтому надо уделять внимание не только техническим аспектам встраивания мультимедиа, но и эстетическим аспектам и учитывать моменты, связанные с удобством для посетителя.

Для встраивания изображений в HTML-документ применяется элемент IMG. Он имеет обязательный атрибут src, значением которого должен быть адрес встраиваемого изображения.

Например, простейший вариант записи для включения картинки на страницу:

< IMG src = “ image.jpg ” >

При такой записи размер картинки на экране будет соответствовать ее реальному размеру. Если необходимо значительно изменить размер изображения, то лучше использовать специальные графические пакеты, однако в небольших пределах допустимо использовать и атрибуты элемента IMG.

Размер изображения

Чтобы редактировать размер картинки, используют атрибуты *WIDTH* и *HEIGHT*. Их значения указываются в пикселях или процентах от размера окна (в этом случае после размера ставится знак %). Можно указать только один из атрибутов, и тогда второй будет вычисляться автоматически для сохранения пропорций рисунка, например:

< IMG src = “ image.jpg ” WIDTH = “150”> - задана только ширина в пикселях

< IMG src = “ image.jpg ” HEIGHT = “45%”> - задана только высота в процентах

< IMG src = “ image.jpg ” WIDTH = “150” HEIGHT = “150”> - задана и ширина и высота в пикселах.

Выравнивание изображения

Расположение картинки влияет на общий вид страницы, на восприятие текста вокруг нее. Удобство чтения текста, находящегося около картинки, сильно зависит от их взаимного расположения.

Есть множество вариантов выравнивания картинок относительно текста, и за все отвечает атрибут *align* элемента *IMG*. Он позволяет выравнивать изображения с правой, с левой стороны окна или относительно элементов строки.

Горизонтальное выравнивание:

- Left – по левому краю;
- Right – по правому краю;

Вертикальное выравнивание:

- Top – выравнивание верхней границы картинки по самому высокому элементу строки;
- Texttop – выравнивание верхней границы картинки по самому высокому элементу текста;
- Middle – середина изображения выравнивается по базовой линии строки;
- Absmiddle – середина изображения выравнивается по середине строки;
- Baseline – выравнивание нижней границы изображения по базовой линии строки;
- Bottom – аналогично Baseline;
- Absbottom – нижняя граница изображения выравнивается по нижней границе текущей строки.

При горизонтальном выравнивании текст плотно обтекает картинку и для внешнего вида страницы важным параметром становится расстояние между текстом и изображением.

Расстояние между текстом и изображением

Расстояние между текстом и картинкой влияет на читаемость текста. Очень маленькое или очень большое расстояние может оказаться неудобным.

Чтобы отодвинуть текст от картинки, используют атрибуты *Hspace* и *Vspace* элемента IMG. Они задают расстояние в пикселях между изображением и текстом по горизонтали и вертикали, например:

< IMG src = " image.jpg " WIDTH = "150" HEIGHT = "150" Hspace = "10" Vspace = "10">

Рамка вокруг изображения

Кроме регулировки расстояния до текста, есть еще один способ отделить картинку от остального содержимого страницы – поставить изображение в рамку. Это делается при помощи атрибута Border, значение которого указывает ширину рамки в пикселях, например:

< IMG src = " image.jpg " WIDTH = "150" HEIGHT = "150" Hspace = "10" Vspace = "10" *Border = "15" >*

Альтернативный текст

Иногда случается так, что картинка не загружается, например, когда пользователь отключил загрузку картинок. В таких ситуациях может помочь атрибут *Alt.* Его значение – текст, который будет показан при наведении указателя мыши на картинку, например:

< IMG src = " image.jpg " WIDTH = "150" HEIGHT = "150" Hspace = "10" Vspace = "10" *Border = "15" Alt = "Текст, поясняющий фотографию">*

Ссылки изображения

Используя картинки, можно повысить функциональность сайта, создавая графические ссылки. Для создания графической ссылки достаточно поместить элемент IMG внутри тегов *<A > … </A>*, при этом вокруг картинки появится рамка, которой можно управлять с помощью атрибута Border, например:

<A HREF=”aboutme.html”> < IMG src = “ myfoto.jpg ” WIDTH = “150” HEIGHT = “150” Hspace = “10” Vspace = “10” *Border = “15” Alt = “Это я!!!”></A>*

Вопрос 2. Добавление мультимедийных файлов.

Если помимо картинок, необходимо вставить на веб-страницу аудио или видео можно воспользоваться несколькими способами: поместить на страницу ссылку на тот мультимедиа-файл, который необходим или просто встроить его на страницу вместе с проигрывателем.

Ссылки на мультимедийные файлы

Ссылки на файлы мультимедиа ничем не отличаются от ссылок на другие страницы. При щелчке кнопкой мыши на такой ссылке браузер откроет музыкальный файл в новом окне. Например:

<A HREF=”music.mp3” title = “Прослушайте пожалуйста”> Музыка для души </A>

Указание ссылки на объект – самый простой вариант для реализации доступа к нему. Плюсом этого способа являются маленький объем страницы и простота использования. С помощью ссылки посетитель может сохранить файл у себя на компьютере, а затем проиграть его, когда захочет.

Однако этот способ не помогает, если нужно, чтобы пользователь посмотрел Flash-файл. С их помощью обычно «раскрашивают» сам сайт; яркие и динамичные Flash-объекты могут нести смысловую нагрузку и быть неотъемлемой частью страницы, поэтому их удобнее встраивать сразу в страницу, а не передавать по ссылке.

Встраивание объектов

При загрузке мультимедиа-файла на страницу сразу надо учитывать с помощью какой программы будет воспроизводиться файл, потому что загружать динамический объект нужно вместе с проигрывателем. Для обращения к этой программе необходимо использовать элемент *OBJECT* (требует закрывающего тега). Он позволяет управлять многими параметрами проигрывателя.

Элемент OBJECT использует программы, находящиеся на компьютере посетителя, и с их помощью воспроизводит активное содержимое.

Атрибуты элемента OBJECT

У элемента OBJECT существует много атрибутов, все они являются

обязательными, точнее, у этого элемента нет строго определенных атрибутов. Однако среди них можно выделить наиболее важные, которые упростят и ускорят обработку вашего объекта браузером.

Таблица 1. Атрибуты тега OBJECT

Classid	Указывает адрес программы, которая будет работать с объектом. В качестве значения атрибута указывается полный или относительный путь к файлу программы.
Data	В качестве значения принимает адрес файла, который необходимо запустить с помощью проигрывателя.
Type	Задает тип объекта, который указан в атрибуте data.
Standby	Задает текст, который будет отображаться на экране до тех пор, пока объект не загрузится целиком.
Width **Height**	Устанавливают ширину и высоту объекта.
Align	Служит для выравнивания объекта на странице, его значения аналогичны значениям этого атрибута для элемента IMG: – Left – по левому краю; – Right – по правому краю; – Top – выравнивание верхней границы картинки по самому высокому элементу строки; – Texttop – выравнивание верхней границы картинки по самому высокому элементу текста; – Middle – середина изображения выравнивается по базовой линии строки; – Absmiddle – середина изображения выравнивается по середине строки; – Baseline – выравнивание нижней границы изображения по базовой линии строки; – Bottom – аналогично Baseline; – Absbottom – нижняя граница изображения

	выравнивается по нижней границе текущей строки.
Hspace **Vspace**	Задает расстояние между объектом и текстом.

Благодаря атрибутам элемента OBJECT можно менять общие параметры вашего объекта, но есть еще один важный элемент, позволяющий задавать параметры самого приложения, в котором происходит воспроизведение.

Элемент PARAM

Это главный элемент, предназначенный для управления проигрывателем. С его помощью можно управлять панелями проигрывателя, регулировать начальную громкость воспроизведения и другие параметры приложения; в общем, этот элемент позволяет гибко управлять программой, которая воспроизводит мультимедийное содержимое. Команды для каждой из программ обработчиков свои, и их легко можно найти в Интернете.

Элементов PARAM внутри элемента OBJECT может быть много, так как для каждого управляющего параметра нужен свой элемент.

Используют два основных атрибута элемента PARAM: name и value. Первый задает имя параметра приложения, а второй – значение этого параметра.

Элемент PARAM дает неограниченные возможности для управления воспроизведением мультимедиа на вашей странице.

Вопрос 3. Вставка фоновых изображений и фоновой музыки на веб-страницу.

В языке HTML есть возможность создавать фоновые изображения и фоновую музыку на веб-страницах. Рассмотрим подробнее:

Фоновое изображение – это графический файл, содержащий картинку (желательно небольшого размера), который многократно выводится на экран, заполнив все окно. Фоновая графика задается в теге <BODY> в начале документа HTML, тому, как задается цвет фона. При этом используется атрибут BACKGROUND, значением которого является имя графического файла. Например:

<HTML>
<HEAD>
<TITLE>Заголовок документа</TITLE>
</HEAD>
<body bgcolor="#BBE0E3" text="000000"

BACKGROUND="ARROW6.GIF">
</body>
</HTML>

Для воспроизведения фоновой музыки проще всего применить тег <BGSOUND> с его основными атрибутами:

- Src – указывает музыкальный файл, используемый в качестве фона;
- Loop – задает количество проигрываний файла;
- Volume – задает громкость звучания файла;
- Balance – задает смещение звука по панораме.

Пример:

<HTML>
<HEAD>
<TITLE>Заголовок документа</TITLE>
</HEAD>
<body bgcolor="#BBE0E3" text="000000" BACKGROUND="ARROW6.GIF">
<BGSOUND SRC=”MUSIKFON.MP3” LOOP=”-1” BALANCE=”-5000” VOLUME=”0” >
</body>
</HTML>

Тема 4.Создание таблиц средствами языка HTML.

1. *Проектирование простейших HTML-таблиц*
2. *Создание сложных таблиц и их форматирование*

Вопрос 1. Проектирование простейших HTML-таблиц.

В технической литературе и в различных документах таблицы используют, чтобы расположить информацию в простом и понятном виде. В языке HTML таблицы используются в двух случаях: для представления числовых данных, разбитых по строкам и столбцам, или как средство форматирования веб-страниц. Ячейки таблицы могут содержать любые HTML-элементы, например заголовки, списки, абзацы, фигуры, графику, а также элементы форм.

Таблица состоит из ячеек, образующихся при пересечении строк и столбцов. В стандартной таблице каждая строка и каждый столбец содержат одинаковое количество ячеек. Однако таблицы могут очень сильно отличаться от стандартной. Например, в таблице ячейки могут объединяться по строкам и столбцам.

При создании таблицы лучше начать с ее планирования, потом будет легче верстать, так как вы наглядно будете видеть вашу таблицу и будет меньше вероятности ошибиться. Для этого вам следует выбрать оптимальный способ создания таблицы.

Создание тела таблицы

В построении HTML-таблиц нет ничего сложного. Описание таблиц должно располагаться внутри элемента BODY. Документ может содержать произвольное количество таблиц, допускается вложение таблиц друг в друга. Каждая таблица должна начинаться тегом <TABLE> и завершается тегом </TABLE>. Например:

<BODY>
<TABLE>
……
……....
</TABLE>
</BODY>

Все прочие элементы таблицы должны быть вложенными в элемент TABLE. Название таблицы определяется тегами <CAPTION> … </CAPTION>. Выравнивание названия задается с помощью атрибута align, который может принимать значения top (над таблицей) и bottom (под таблицей). По умолчанию наименование располагается над таблицей.

Ячейки таблицы

При создании ячеек таблицы необходимо создать нужное количество строк, затем поделить строки на ячейки. Количество пар тегов <TR> и </TR> определяет количество горизонтальных строк в таблице. Количество пар тегов <TD> и </TD>, расположенных между тегами соответствующей строки, определяет количество ячеек в пределах строки. Например:

<BODY>
<TABLE>
<TR> <TD> … </TD> <TD> … </TD> <TD> … </TD> </TR>
<TR> <TD> … </TD> <TD> … </TD> <TD> … </TD> </TR>
</TABLE>
</BODY>

Для того чтобы созданная ячейка являлась заголовком в таблице (текст в ячейке выравнивается по центру и выделяется полужирным шрифтом), нужно воспользоваться парой тегов <TH> и </TH>.

Граница таблицы

В рассмотренных выше примерах, в таблице будут отсутствовать границы. Граница задается с помощью атрибута BORDER элемента TABLE. Значением атрибута задается ширина границы таблицы, которая указывается в пикселях. Благодаря атрибуту BORDERCOLOR можно задать цвет границы таблицы, указав код цвета.

Рассмотрим пример создания простой HTML-таблицы:

<HTML>
<HEAD><TITLE>Пример простой таблицы</TITLE></HEAD>
<BODY color="#FFCCCC">

```
<TABLE  border="2">
 <TR>
  <TD >1-я ячейка первой строки</TD>
  <TD >2-я ячейка первой строки</TD>
  <TD >3-я ячейка первой строки</TD>
 </TR>
 <TR>
  <TD>1-я ячейка второй строки</TD>
  <TD>2-я ячейка второй строки</TD>
  <TD>3-я ячейка второй строки</TD>
 </TR>
 <TR>
  <TD>1-я ячейка третьей строки</TD>
  <TD>2-я ячейка третьей строки</TD>
  <TD>3-я ячейка третьей строки</TD>
 </TR>
</TABLE>
 </BODY>
</HTML>
```

Рис. 1. Пример простой HTML-таблицы

Вопрос 2. Создание сложных таблиц и их форматирование.

Для того чтобы разнообразить вид таблиц можно использовать дополнительные атрибуты, такие как:

CELLSPACING – определяет расстояние между ячейками таблицы в пикселях;

CELLPADDING – определяет расстояние между содержимым ячейки и ее границей в пикселях;
FRAME – позволяет управлять отображением внешней границы; может принимать следующие значения:

Above – отображается только верхняя линия границы;

Below – видна только нижняя линия границы;

Box – отображается внешняя граница таблицы;

Border – видна внешняя граница таблицы (аналогично box);

Hsides – отображаются только горизонтальные линии границы;

Lhs – видна только левая линия границы;

Rhs – видна только правая линия границы;

Void - внешняя граница таблицы не отображается;

Vsides – видны только вертикальные линии границы.

RULES – позволяет управлять отображением внутренними границами таблицы; может принимать следующие значения:

All – граница отображается вокруг каждой ячейки;

Cols – видны только вертикальные границы между столбцами;

Groups – отображаются только вертикальные границы между группами столбцов или горизонтальные границы между группами строк;

None – границы между ячейками не отображаются;

Rows – видны только горизонтальные границы между группами строк.

Атрибуты FRAMES и RULES – это новые атрибуты в языке HTML, поэтому они поддерживаются только последними версиями браузеров.
WIDTH – задает ширину таблицы в элементе TABLE (значение можно задавать как в абсолютных, так и в относительных еденицах);
HEIGTH – задает высоту таблицы в элементе TABLE (значение можно задавать как в абсолютных, так и в относительных еденицах);

Ширину и высоту можно задавать также и для ячеек таблицы. Просто необходимо использовать элементы с соответствующими атрибутами. При этом вовсе не обязательно задавать размеры каждой отдельной ячейки. При изменении ширины и высоты ячейки все соседние ячейки в пределах столбца будут отображаться с учетом нового значения.

Объединение ячеек таблицы

На практике встречается достаточно большое количество таблиц, в которых одна ячейка объединяет в себе несколько ячеек по высоте и ширине. В HTML ячейки объединяют с помощью атрибутов COLSPAN и ROWSPAN.
COLSPAN – определяет количество ячеек, на которые простирается данная ячейка по горизонтали;
ROWSPAN – определяет количество ячеек, на которые простирается

данная ячейка по вертикали.

Рассмотрим пример объединения ячеек:

```
<HTML>
<HEAD><TITLE>Пример сложной таблицы</TITLE></HEAD>
<BODY color="#FFCCCC">
<TABLE cellspacing="3" cellPadding="3" width="70%"  border="2">
 <TR>
  <TD Colspan="3">1-я ячейка первой строки</TD>
 </TR>
 <TR>
  <TD Rowspan="2">1-я ячейка второй строки</TD>
  <TD>2-я ячейка второй строки</TD>
  <TD>3-я ячейка второй строки</TD>
 </TR>
 <TR>
  <TD>1-я ячейка третьей строки</TD>
  <TD>2-я ячейка третьей строки</TD>
 </TR>
</TABLE>
 </BODY>
</HTML>
```

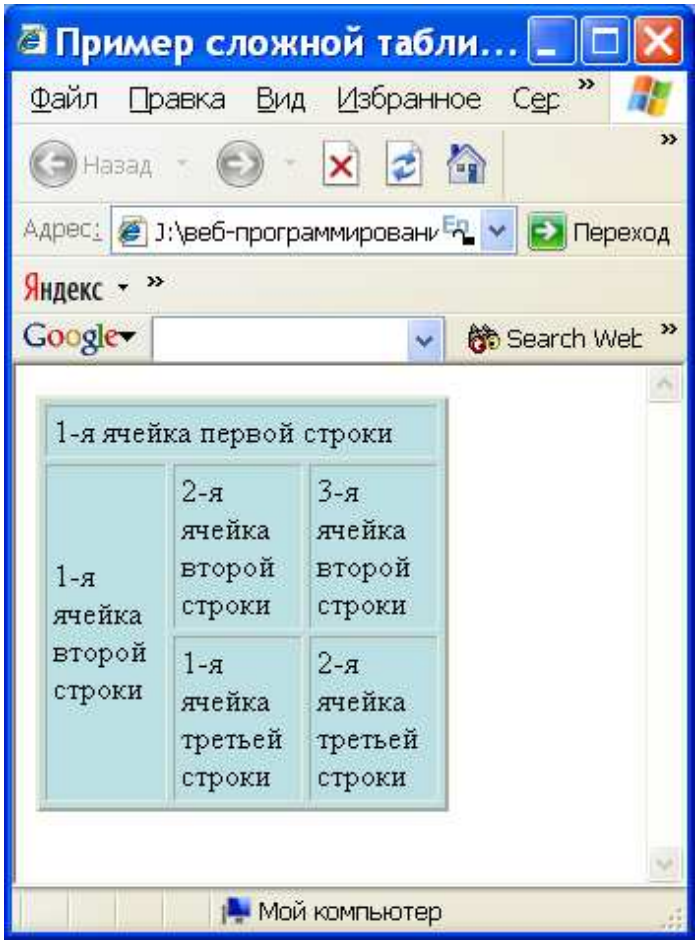


Рис. 2. Пример объединения ячеек

Установка фонового цвета или рисунка ячейки

В HTML возможны различные варианты установки фонового цвета или рисунка. Благодаря атрибуту BGCOLOR можно изменять цвет содержимого ячейки, строки, группы столбцов, группы строк, таблицы целиком.

С помощью атрибута BACKGROUND можно задать графический фон ячейки или таблицы целиком, указав путь к изображению. Например:

```
<HTML>
<HEAD><TITLE>Пример сложной таблицы</TITLE></HEAD>
<BODY>
<TABLE cellspacing="3" cellPadding="3" width="70%" bgcolor="#BBE0E3" border="2">
  <TR bgcolor="#BBE0E3">
    <TD Colspan="3">1-я ячейка первой строки</TD>
  </TR>
  <TR>
    <TD Rowspan="2">1-я ячейка второй строки</TD>
    <TD bgcolor="blue">2-я ячейка второй строки</TD>
    <TD BACKGROUND = “fontable.jpeg”>3-я ячейка второй строки</TD>
  </TR>
</TABLE>
 </BODY>
</HTML>
```

Создание вложенных таблиц

Существуют такие моменты, когда необходимо создать ячейки, границы которых не должны совпадать. Именно для таких моментов, команды, управляющие созданием и отображением таблиц, допускают вложение одного элемента TABLE внутрь другого, поэтому в ячейке можно создать новую таблицу с независимой структурой.

```
<HTML>
<HEAD><TITLE>Пример сложной таблицы</TITLE></HEAD>
<BODY>
<TABLE cellspacing="8" cellPadding="8" width="70%" bgcolor="#BBE0E3" border="2">
  <TR bgcolor="#BBE0E3">
    <TD Colspan="3" align="center">1-я ячейка первой строки</TD>
  </TR>
  <TR>
    <TD Rowspan="2">1-я ячейка второй строки</TD>
    <TD bgcolor="blue">2-я ячейка второй строки</TD>
    <TD>
```

```
<TABLE  border="2">
 <TR>
  <TD >1-я ячейка первой строки вложенной таблицы</TD>
  <TD >2-я ячейка первой строки вложенной таблицы</TD>
  <TD >3-я ячейка первой строки вложенной таблицы</TD>
 </TR>
 <TR>
  <TD>1-я ячейка второй строки вложенной таблицы</TD>
  <TD>2-я ячейка второй строки вложенной таблицы</TD>
  <TD>3-я ячейка второй строки вложенной таблицы</TD>
 </TR>
</TABLE>
</TD>
 </TR>
</TABLE>
</BODY>
</HTML>
```

Рис. 3. Пример вложенных HTML-таблиц

Тема 5. Создание фреймов в HTML.

1. *Разбиение окна браузера на фреймы*
2. *Ссылки внутри фреймов*
3. *Создание плавающих фреймов*

Вопрос 1. Разбиение окна браузера на фреймы.

Фреймы (frames) – это HTML-элементы, позволяющие разделить страницу на несколько независимых окон. В каждое окно можно поместить собственную независимую HTML-страницу, в которой может находиться ссылка из одного окна в другое. Они применяются в основном для организации постоянно находящихся на экране меню, в то время как в другом окне располагается непосредственно сама информация.

Фреймы чаще всего используют при создании оглавлений,

неподвижных элементов интерфейса и т.д.

При использовании фреймов потребуется несколько HTML-документов. Один из них называется установочным (базовым). В нем описывается расположение фреймов, но нет текста и графики. Исходные файлы выводятся во фреймы и могут содержать текст и графику.

Чтобы HTML-страница содержала фреймы, а экран был разделен на области, необходимо заменить пару тегов *<BODY> … </BODY>* в коде HTML-страницы парой тегов *<FRAMESET> … </FRAMESET>* . Необходимо также указать хотя бы один из атрибутов *COLS* или *ROWS*. С помощью этих атрибутов задают размеры фреймов и количество строк и столбцов в разбиении страницы.

Атрибут *ROWS* определяет количество строк, а атрибут *COLS* – количество столбцов. В значении этих атрибутов необходимо указывать не количество строк или столбцов, а значение ширины или высоты каждого фрейма через запятую. Размеры фреймов могут задаваться как в абсолютных, так и в относительных величинах. Области, полученные в результате разбиения, и будут являться фреймами.

Чтобы в каждой из областей отображалась информация, необходимо использовать элемент *FRAME*, обязательный атрибут *SRC* которого укажет адрес документа, отображаемого внутри фрейма. Если адрес отсутствует, то будет отображен пустой фрейм.

Рассмотрим создание фреймов на примерах:

```
<HTML>
<HAED> <TITLE> Эта страница содержит 3 фрейма </TITLE> </HEAD>
<frameset cols="20%, 30%, *">
<frame src="blue.html" >
<frame src="red.html" >
<frame src="green.html" >
</HTML>
```

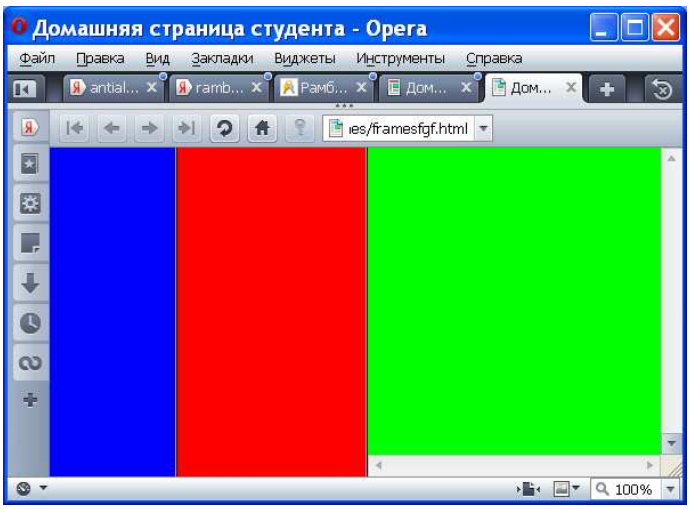


Рис. 1. Пример страницы, содержащей три вертикальных фрейма

```
<HTML>
<HAED> <TITLE> Эта страница содержит 3 фрейма </TITLE>
</HEAD>
<frameset rows="20%, 30%, *">
<frame src="blue.html" >
<frame src="red.html" >
<frame src="green.html" >
</HTML>
```

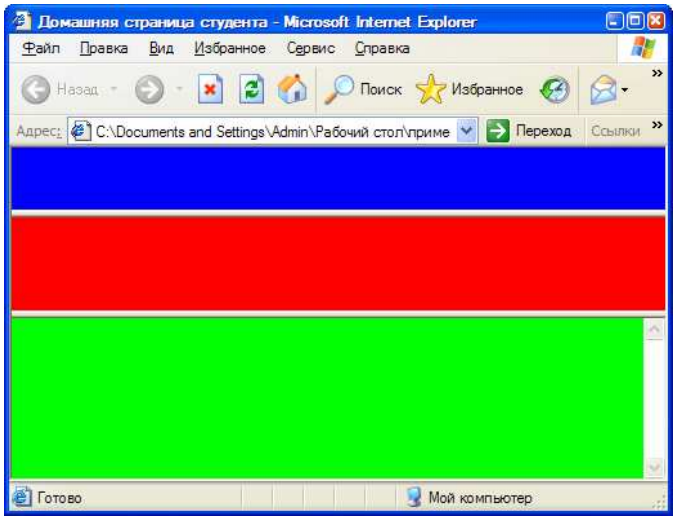


Рис. 2. Пример страницы, содержащей три горизонтальных фрейма

Границы фрейма

Границей фрейма можно управлять, как и любыми другими элементами фрейма. Для этого существует несколько атрибутов элемента *FRAMESET*, позволяющих настраивать границы фреймов. Ширина границы определяется атрибутом *BORDER*. По умолчанию значение ширины границы равно пяти. Чтобы граница фрейма отсутствовала, необходимо либо указать значение ширины границы равное нулю, либо

присвоить значение *NO* атрибуту *FRAMEBORDER*.

С помощью атрибута *BORDERCOLOR* определяется цвет границы фрейма, необходимо только указать код или соответствующее зарезервированное имя цвета.

Полосы прокрутки

В некоторых случаях полосы прокрутки нарушают дизайн HTML-страницы. Если содержимое фрейма не помещается в отведенную область, то автоматически появляются полосы прокрутки для просмотра информации. Для управления отображением полос прокрутки используется атрибут SCROLLING, принимающий три основных значения:

Yes – всегда вызывает появление полос прокрутки, независимо от объема информации;

No – запрещает появление полос прокрутки;

Auto – полосы прокрутки отображаются при необходимости (по умолчанию).

Рассмотрим пример страницы:

```
<html>
<head><title>фреймы</title></head>
<frameset cols="50%, *" rows="50%, *">
<frame src="frame2.html" scrolling="yes">
<frame src="frame1.html">
<frame src="frame1.html" scrolling="no">
<frame src="frame4.html">
</frameset>
</html>
```

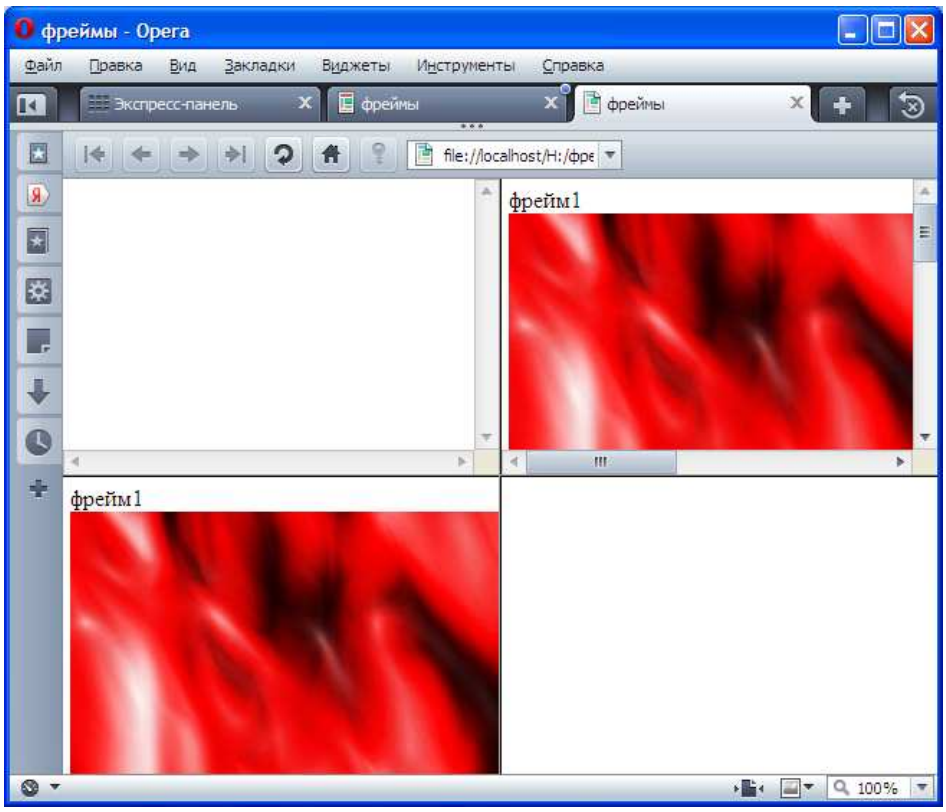


Рис. 2. Пример управления полосами прокрутки

Изменение размеров фреймов

По умолчанию размеры фреймов можно изменять с помощью указателя мыши. Необходимо просто навести указатель мыши на границу между фреймами и, зажав левую кнопку мыши, переместить границу влево или вправо, вверх или вниз.

Для того чтобы заблокировать возможность изменения пользователем размеров фреймов, следует воспользоваться атрибутом *noresize* элемента FRAME.

```
<frameset cols = 20%, *>
    <frame src=frame1.html noresize>
    <frame src=frame2.html>
</frameset>
```

Атрибут noresize не требует никаких значений. Для страницы с двумя фреймами этот параметр можно указать лишь в одном месте. Естественно, если у одного фрейма нельзя изменять размеры, то у близлежащего к нему размеры также не будут меняться.

Управление пространством внутри фрейма

По умолчанию браузер отображает фреймы с полосой чистого пространства (полями) между содержимым и границей фрейма, равной 10 пикселей. Управлять этим пространством фрейма можно с помощью атрибутов MARGINHEIGHT и MARGINWIDTH.

Атрибут MARGINWIDTH определяет высоту свободного

пространства слева и справа от содержимого фрейма, а MARGINHEIGHT – сверху и снизу соответственно. Задавая величины полей, надо использовать оба атрибута, так как браузер может автоматически сократить размеры тех полей, которые вы не указали, до наименьшего значения. Значения этих атрибутов всегда должны указываться в абсолютных значениях (пикселях), а минимальное значение равняется одному пикселю.

Рассмотрим пример HTML-документа, который содержит всего два фрейма (рис.). Изменим поля первого фрейма, а поля второго оставим по умолчанию. Чтобы разница была заметна невооруженным взглядом, в каждый фрейм загрузим изображения.

```
<html>
<head>
<title>
Создание фреймов
</title>
</head>
<frameset cols=”45%, *”>
<frame src=”1.jpg” marginheight=45 marginwidth=15>
<frame src=”2.jpg”>
</frameset>
</html>
```

В соответствии с кодом страницы первого фрейма значительно отличаются от полей второго.

Стоит также отметить, что значения полей необходимо указывать для каждого фрейма в отдельности. Если использовать запись *<frame src=”1.jpg” marginheight=45 marginwidth=1>*, то браузер просто проигнорируют значения атрибутов, а не определит их для всех фреймов.

Вопрос 2. Ссылки внутри фреймов.

В большинстве случаев при переходе по ссылке в окне браузера текущий документ заменяется новым. При использовании фреймов схема загрузки документов отличается от стандартной. Главное отличие – это возможность загружать документ в выбранный фрейм из другого. Для этого применяется атрибут *TARGET* элемента *A*. В качестве значения используется имя фрейма, в который будет загружаться документ, указанный атрибутом *name*. Например:

```
<html>
<head><title>фреймы</title></head>
<frameset cols="50%, *" rows="50%, *">
<frame src="frame2.html" scrolling="yes">
<frame src="frame1.html" name="frame1">
<frame src="frame1.html" scrolling="no">
```

<frame src="frame4.html" name="frame4">
</frameset>
</html>

При этом в странице, содержащей ссылку на другой документ, необходимо обязательно указать атрибут *TARGET* в элементе *A:*
<A href=page.html TARGET ="frame1"> Ссылка на другую страницу</A>

Если пользователь щелкнет кнопкой мыши на данной ссылке, то новый документ загрузится во фрейм1, на что указывает атрибут *TARGET ="frame1"*.

Стоит также отметить, что имя фрейма должно начинаться с цифры или латинской буквы. Следующие имена используются в качестве зарезервированных:

Target = “_blank” – документ загрузится в новом окне;

Target = “_self” – документ загрузится в текущий фрейм;

Target = “_parent” – если документ с гиперссылкой находиться во вложенном наборе фреймов, то документ загружается в родительское окно или набор фреймов;

Target = “_top” – отменяет все фреймы и загружает документ в полное окно браузера.

Используя фреймы, не стоит забывать и об особенностях гиперссылок. Если атрибут target не указан, то целевой документ загружается по умолчанию в тот же фрейм. Если указано несуществующее имя фрейма, то открывается новое окно браузера, получая при этом заданное имя. Для внешних ссылок следует задавать значения атрибута target либо _top, либо _blank, чтобы чужие сайты не отображались в ваших фреймах, а занимали полное окно.

Вопрос 3. Создание плавающих фреймов.

Браузеры последних версий позволяют использовать плавающие фреймы, добавляемые с помощью элемента IFRAME. Плавающий фрейм находиться внутри обычного документа, не содержащего тегов <FRAMESET> … </ FRAMESET>, и позволяет добавлять на страницу любые другие независимые документы. В элементе IFRAME можно указывать те же атрибуты, что и в элементах, описывающих обычные фреймы. Кроме того, можно использовать следующие атрибуты: *width, height, hspace, vspace, align*, значения которых совпадают со значениями соответствующих атрибутов элемента IMG. На рис. 3 изображено окно браузера, содержащее плавающий фрейм.

Если браузер не поддерживает плавающие фреймы, то элемент IFRAME будет проигнорирован. Для таких браузеров можно указать альтернативный текст, расположенный между парой тегов <IFRAME> … </IFRAME>. Например:
<iframe src="page.html" width="40%" height="250" align="right"

hspace="5" vspace="5" >
Ваш браузер не поддерживает плавающие фреймы!
</iframe>

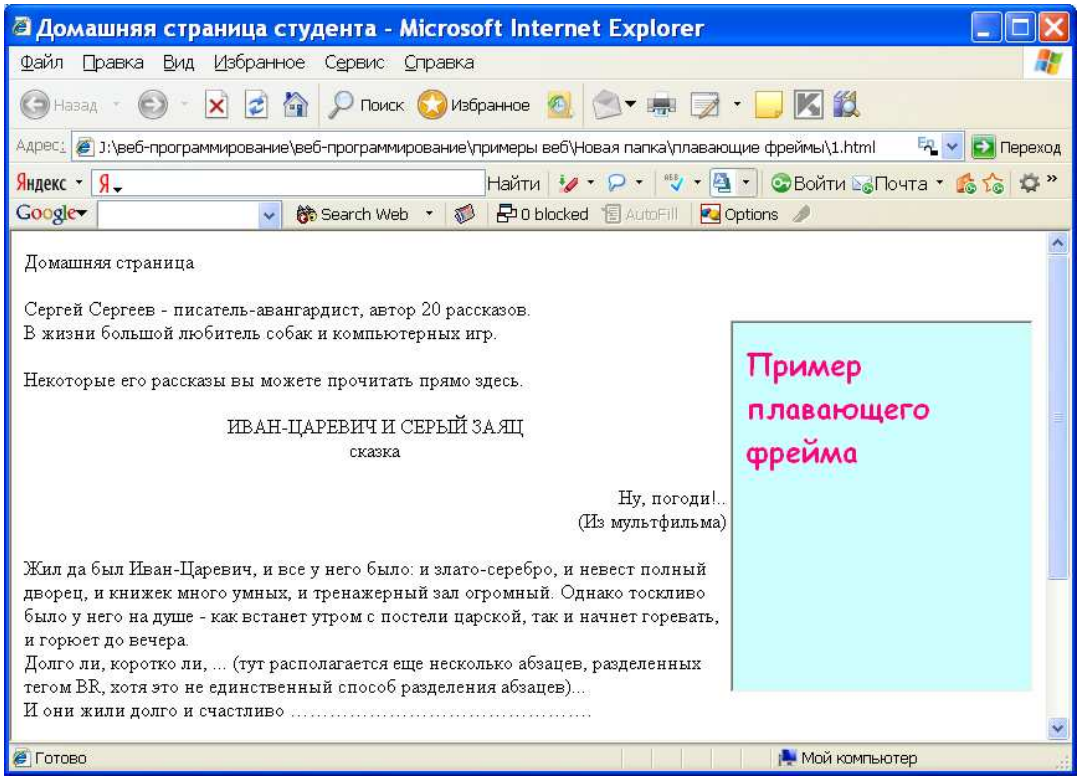


Рис. 3. Пример плавающего фрейма

Тема 6. Конструирование форм в языке HTML.

1. *Элемент FORM*
2. *Создание полей формы*

Вопрос 1. Элемент FORM.

В данной лекции мы рассмотрим такие объекты, которые помогают организовать взаимодействие пользователя с авторами сайта, - формы. Формы позволяют получать информацию о посетителях – любая информация, введенная в форму, может быть отправлена создателю сайта.

Формы напоминают анкеты, в них есть возможность создавать поля, в которые пользователь сам может ввести любую информацию, и поля-списки, где ему нужно выбрать ответ из вариантов, предложенных автором. Возможности форм по группировке и представлению информации позволяют разместить ее наилучшим образом. В умелых руках это мощный и полезный инструмент.

Главное при создании форм – следить за тем, чтобы предложенные вами поля были удобны для посетителей и приемлемы для ответов на ваши вопросы.

Технически формы передают данные от пользователя удаленному серверу. Это позволяет создавать системы регистрации пользователей, форумы, гостевые книги. Для получения и обработки данных форм используются языки веб-программирования (PHP, PERL).

Основой любой формы служит элемент *FORM*, требующий наличия закрывающего тега. С него начинается каждая форма. Атрибуты этого элемента содержат информацию, общую для всех полей формы. На самом деле в одну форму надо включать поля, объединенные логически, потому что в один момент на сервер можно отправить только одну форму.

Большинство атрибутов этого элемента влияют на обработку формы, а не на ее внешний вид. Однако необходимо знать, для чего они предназначены, чтобы понимать, как связать форму с кодом.

Главным для рассматриваемого элемента является атрибут *ACTION*, указывающий обработчик данных для формы. Обработчик – это файл, в котором описано, что нужно делать с данными формы. Создать этот файл – дело программиста, однако переменные, которые будут туда передаваться, зависят от тех, кто описывает форму. Сам обработчик может быть написан на любом языке, например это может быть файл *test.php*. Если значение атрибута *ACTION* не будет указано, то страница перезагрузиться, а все элементы формы примут значения, заданные по умолчанию, то есть в этом случае форма окажется абсолютно бесполезной. В качестве значения данного атрибута надо указать URL-путь к файлу-обработчику на сервере.

Как послать информацию соответствующей программе обработки (скрипту) указывает форме атрибут *METHOD* тега *FORM*. Обычно он получает значение POST, тогда информация формы посылается отдельно от URL. Если указано значение GET, информация формы посылается вместе с URL.

Например:

<FORM METHOD="POST" ACTION="test.php">

……

……

……

</FORM>

Вопрос 2. Создание полей формы.

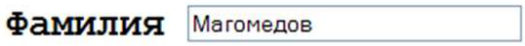
Полей, из которых можно составить форму, очень много, поэтому надо внимательно следить за выбором поля для представления конкретной информации. Говоря о каждом из видов, мы будем рассматривать, в каких случаях его можно использовать. Ведь есть поля с похожими функциями, разница будет только в удобстве применения конкретного поля в конкретной ситуации.

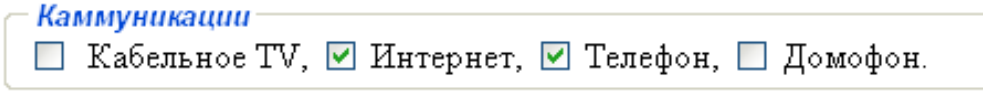
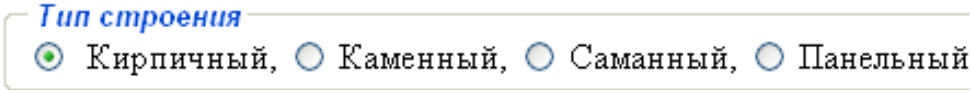
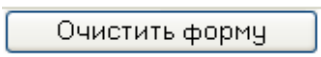
Большинство полей для формы создается с помощью элемента *INPUT*. Атрибуты элемента отличаются в зависимости от поля, для создания которого используется этот элемент. Тем не менее, есть атрибуты, которые можно, а некоторые даже нужно применять вне зависимости от типа поля.

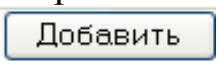
Атрибуты тега INPUT

Начнем с главного атрибута, который собственно, и задает поле, - с атрибута *TYPE*. Он принимает значение, указывающее на тип поля. В табл.1 описано, какое значение может принимать атрибут и какому полю, оно соответствует. Рассмотрим подробнее каждый тип поля и атрибуты, которые могут применяться вместе с INPUT.

Табл. 1. Значения атрибута TYPE

TEXT	Поле ввода текста. Наряду с этим атрибутом можно использовать атрибут MAXLENGHT, значение которого определяет максимальное количество символов, которые можно ввести в данную строку. Также можно задать длину поля ввода, используя атрибут SIZE. Например: 
PASSWORD	Поле для ввода пароля. Особенность этого поля в том, что все введенные буквы закрываются звездочками (или точками, в зависимости от системы) и увидеть вводимый пароль невозможно. Вместе с этим полем можно использовать те же атрибуты, что и для строки ввода текста, то есть MAXLENGHT и SIZE. Например:
BUTTON	Обычная кнопка. Надпись на кнопке задает атрибут VALUE. Например:
FILE	Поле добавления файла. Возможность отправлять файлы на сервер очень помогает в работе. Посетители могут загрузить свои фотографии, рисунки или видео. Особенность этого поля в том, что справа появляется кнопка, при нажатии которой пользователь сможет выбрать загружаемый файл через Проводник, что гораздо удобнее, чем вводить путь до файла вручную. Например:
IMAGE	Поле-картинка. С помощью картинок можно украсить форму. Их можно использовать в качестве кнопок. Для задания кнопки-картинки указываем type = «image», при

	этом необходимо задать атрибут src. Его значение – адрес картинки на сервере. Необходимо следить за размером картинки, чтобы графические кнопки не оказались больше самой формы. Для полей картинок можно использовать атрибут alt, который задает альтернативный текст. Можно задать положение картинки на странице относительно текста с помощью атрибута align.
CHECKBOX	Кнопка флажок. Флажки напоминают переключатели тем, что дают пользователю возможность выбирать из предложенных вами вариантов. Главным отличием является то, что посетитель может выбрать сразу несколько пунктов, а сами пункты обозначаются квадратиками. В группы эти элементы объединяются при помощи атрибута NAME, где должно быть установлено одинаковое значение. Для флажков можно использовать атрибут CHECKED, который можно установить для нескольких пунктов. Например: 
RADIO	Кнопка переключатель. Особенность переключателя в том, что можно выбрать только одно положение из всех возможных значений. Текст пункта вводиться сразу после тега INPUT. Для всех переключателей, которые принадлежат к одной группе, должно быть установлено одинаковое значение атрибута NAME. Вместе с переключателями можно использовать атрибут CHECKED, который задает положение, выбранное по умолчанию. Это удобно, если вы знаете самый частый вариант ответа на заданный вопрос. Например: 
RESET	Кнопка очистки формы. Надпись на кнопке задает атрибут VALUE. Например: 

SUBMIT	Кнопка отправки формы. Надпись на кнопке задает атрибут VALUE. Например: 
HIDDEN	Скрытое поле. Вместе с формой можно передавать дополнительную информацию, которая не должна быть видна пользователю. Они не видны пользователю, однако информация из них передается обработчику. Такая информация часто носит служебный характер и не связана с данными, вводимыми пользователем. Используются также атрибуты NAME и VALUE – имя и значение переменной, передаваемые обработчику.

Иногда бывает нужно запретить пользователю изменение данных поля, например, чтобы он не смог поменять уже введенную информацию. Чтобы запретить изменение данных в любое поле можно добавить атрибут *DISABLED*. При этом поле не просто нельзя будет поменять, его невозможно даже взять в фокус.

Атрибут *VALUE* задает значение, которое будет передано обработчику. По сути, он определяет данные, введенные в поле. Для текстовых полей с помощью этого атрибута можно установить значение по умолчанию. Введя в поле значение по умолчанию, можно пояснить пользователю, какие данные и в каком формате вы хотите здесь видеть. Это как образец: гораздо удобнее заполнять форму, видя перед собой пример.

Для полей, предоставляющих возможность выбора, атрибут *VALUE* определяет, какое значение будет передано обработчику при выборе конкретного пункта. В таком случае важно, чтобы значения этого параметра логически соответствовали пунктам, из которых можно выбирать.

Рассмотренные выше атрибуты можно применять в любых полях, они всегда будут действовать.

Мы разобрались со всеми возможностями элемента INPUT. Дальше рассмотрим поля, которые входят в форму, но создаются с помощью других элементов.

Поля-списки

Списки дают возможность расположить большое количество пунктов компактно. При создании списков вы можете сами определить количество видимых элементов. Можно настроить возможность выбора одного или нескольких пунктов. По функциям списки напоминают переключатели и флажки, однако, их плюс в компактности. С другой стороны, поиск по спискам занимает больше времени, хотя можно задать

элементы, выбранные по умолчанию (см. рис. 1).

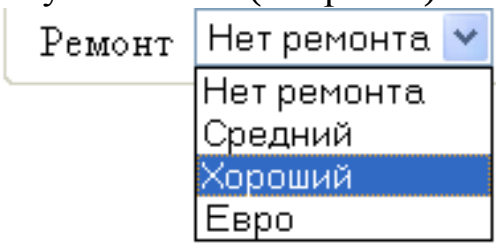


Рис. 1. Поля списки

Для создания списков используется элемент *SELECT*, у которого два атрибута, отвечающих за общее отображение списка. Атрибут *SIZE* устанавливает количество видимых элементов из списка. Например, если *SIZE= “1”* , то количество элементов списка больше одного, то появится раскрывающийся список выбора. Если значение *SIZE* больше единицы, то при количестве пунктов списка, превышающем его размер, появится полоса прокрутки, а если пунктов меньше, то в поле останется пустое место.

Атрибут *MULTIPLE* дает возможность выбирать несколько элементов из списка, для этого при выборе надо нажать и удерживать клавишу *CTRL*.

Если добавить на страницу элемент *SELECT* просто так, то вы увидите пустой список. Для добавления в список пунктов используются элементы *OPTION*, которые располагаются внутри элемента *SELECT*.

Атрибут *VALUE* задает значение, которое будет передано обработчику в случае выбора соответствующего пункта.

Атрибут *SELECTED* задает пункт, выбранный по умолчанию. Название элемента списка находится внутри элемента *OPTION*.

Рассмотрим пример создания списка:

```
<select name="gorod" size="13" align="left">
    <option value="mahachkala" selected="selected">Махачкала
    <option value="kaspisk">Каспийск
    <option value="byinaksk">Буйнакск
    <option value="dag_ogni">Даг.Огни
    <option value="izberbash">Избербаш
    <option value="derbent">Дербент
    <option value="kizlar">Кизляр
    <option value="kizilyrt">Кизилюрт
    <option value="xasavyrt">Хасавюрт
    <option value="yznosyxokymsk">Южносухокумск
 </select>
```

Результат обработки рассмотренного примера, представлен на рис.1.

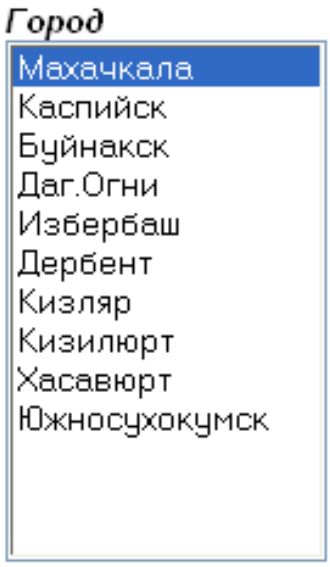


Рис. 2. Список опций

Многострочные текстовые поля

В многострочных текстовых полях обычно вводятся комментарии, сообщения и многое другое. Для создания таких полей применяется элемент *TEXTAREA*, который обязательно должен иметь закрывающий тег. Текст, вводимый по умолчанию, находится между открывающим и закрывающим тегами этого элемента. Размеры поля устанавливаются с помощью атрибутов *COLS* и *ROWS*. Первый задает размеры по горизонтали, второй - вертикали.

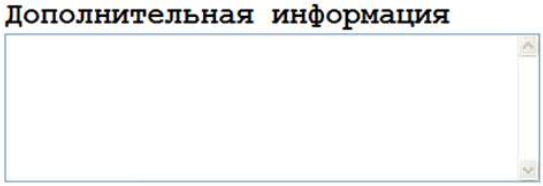
Можно запретить пользователю изменять текст в текстовом поле, что пригодится, если поле используется, например, для представления договора. Запрет вводится путем установки атрибута *READONLY*.

Еще один атрибут элемента *TEXTAREA* – *WRAP* указывает браузеру, как осуществляется перенос текста и в каком виде текст отправляется на сервер. Этот атрибут может принимать три значения:

- *HARD* – переносит строку автоматически, когда она доходит до границы окна ввода, и передает на сервер с такой расстановкой переносов;
- *SOFT* – автоматически переносит строку;
- *OFF* – переносит слова только при нажатии клавиши ENTER, и, если строка превышает горизонтальный размер окна, появляется полоса прокрутки.

Рассмотрим пример создания больших текстовых полей:

```
<B>Дополнительная
информация</B>
<textarea rows=7 cols=50>
</textarea>
```



Рассмотрим пример создания формы:

```
<HTML>
<HEAD><title>Создание формы </title></HEAD>
<BODY>
<PRE><basefont size=5><H2>Заявка на покупку товара</h2>
<form>
<B>Фамилия</B> <input type=text size=30>
<B>Пароль </B> <input type=password size=30>
<B>Адрес получателя </B> <input type=text size=50>
<B>Тип товара</B> <select>
<option>Монитор
<option>Клавиатура
<option>Мышь
<option>Процессор
<option>Видеокарта
<option>Принтер
<option>Ксерокс
</select>
<B>Код товара </B><select>
<option>02354
<option>03666a
<option>04578
<option>05688
<option>06741
<option>07852
<option>08951
</select>
<B>Количество</B> <input type=text size=10>
<B>Доставка транспортом </B>
<input type=radio name="firm"><B>фирмы </B>  <input type=radio
name="firm"><B>своим </B>
<B>Оплата наличными</B> <input type=checkbox><br>
<B>Дополнительная информация</B>
<textarea rows=7 cols=50>
</textarea>
<B><input type=button value=Отправить></B>
<B><input type=reset value=Сброс></B>
</PRE>
</BODY>
</HTML>
```

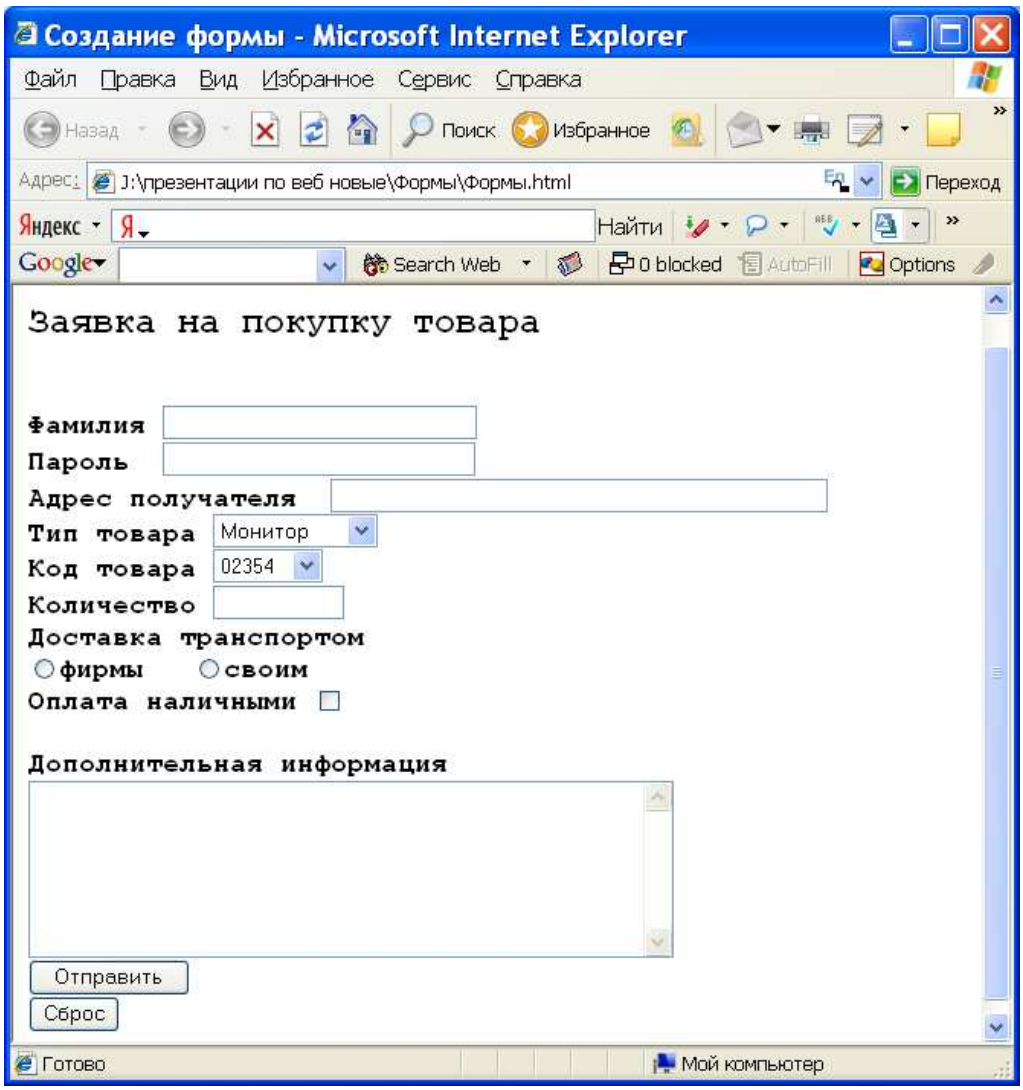


Рис. 3. Пример создания формы

Тема 7. Карта графических ссылок.

1. *Технология создания карты графических ссылок*
2. *Характеристика основных тегов и атрибутов*

Вопрос 1. Технология создания карты графических ссылок.

При создании WEB-страниц имеется возможность установить несколько гиперссылок с одного рисунка, не «разрезая» его. Такое изображение называется картой графических ссылок – *IMAGEMAP*.

Рассмотрим на примере технологическую последовательность работ создания карты графических ссылок.

1. Сначала рисунок внедряется на веб-страницу обычным образом, с помощью тега <*IMG*>:

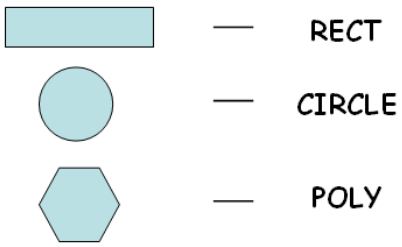
<IMG SRC="computer.gif" WIDTH="451" HEIGHT="310" BORDER="0" >

2. Затем необходимо установить в этом тэге атрибут *USEMAP*. Его значение должно содержать имя карты графических ссылок, с предварительным знаком #.

<IMG SRC="computer.gif" WIDTH="451" HEIGHT="310" BORDER="0"

USEMAP="#comptmap">

3. Карта графических ссылок начинается с тэга *<MAP>* и завершается закрывающим тэгом *</MAP>*. В тэге *<MAP>* следует установить (обязательно!) единственный возможный атрибут *NAME* - имя карты, которое предварительно задавалось в атрибуте *Usemap*.
4. Между тегами *<MAP>* и *</MAP>* должна находиться основная часть карты. Она состоит из нескольких тегов *<AREA>*, каждый из которых определяет активную область рисунка.
5. **«Активные области»** могут быть:



И определяются установкой значения атрибута *SHAPE* в теге *<AREA>*.

6. Дальше начинается самое неприятное в составлении карты графических ссылок: нужно определить координаты каждой из активных областей и записать их как значения атрибута *COORDS*.

<AREA SHAPE="rect" COORDS="0,0, 200,150">
<AREA SHAPE="circle" COORDS="200,150,40">
<AREA SHAPE="poly" COORDS="0,150,400,150, 200, 0">

7. В теге *<AREA>* необходимо установить атрибут *HREF* - определяющий гиперссылку, то есть куда должен попасть пользователь при щелчке мышью на этой «активной области». Здесь никаких сложностей нет — атрибут *HREF* тега *<AREA>* ведет себя так же, как если бы он был в обычном теге *<A>*.
8. Кроме того, бывает полезно определить также атрибут *ALT*. Его значение может содержать поясняющий текст, который будет «всплывать» при наведении мыши на «активную область».

Приведем полный HTML-код создания карты графических ссылок:

<HTML>
<HEAD>
<TITLE>Пример создания карты графических ссылок</TITLE>
</HEAD>
<body bgcolor="#BBE0E3" text="red">
<h2>
<IMG border=3 src="Рисунок1.jpg" Usemap="#comptmap">
<map name="comptmap">
<area shape="circle" coords="80, 288, 20" alt="мышка"
href="m.html">

<area shape="rect" coords="152, 253, 355, 291"
alt="клавиатура" href="klava.html">
<area shape="circle" coords="126, 146, 100"
alt="монитор" href="monitor.html">
<area shape="rect" coords="222, 25, 297, 237"
alt="системный блок" href="systemblock.html">
</map>
</body>
</HTML>

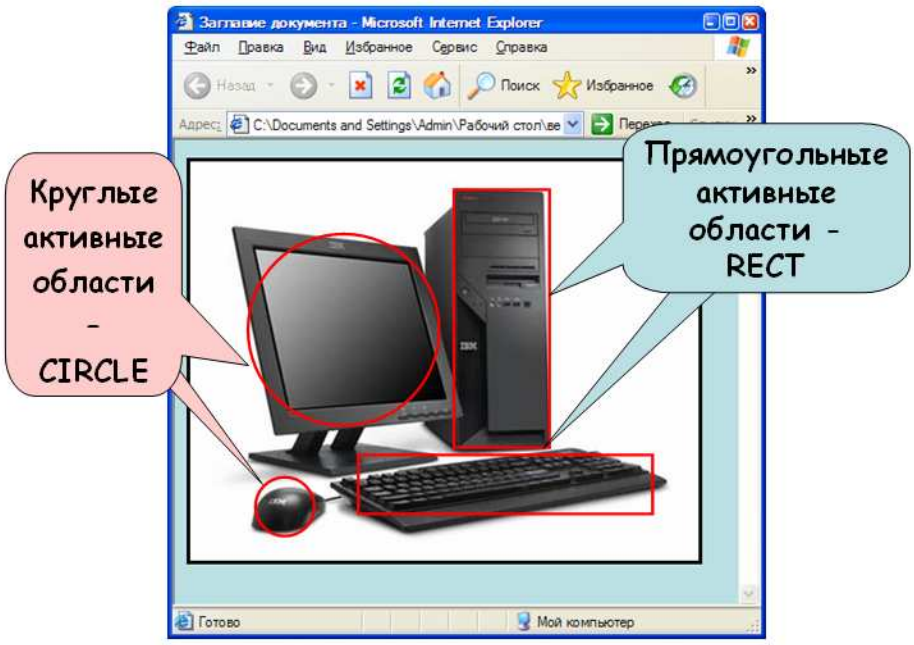


Рис. 1. Готовая карта графических ссылок

Вопрос 2. Характеристика основных тегов и атрибутов.

<IMG> – вставка изображения;
USEMAP – задает имя карты в тэге IMG;
<MAP> </MAP> – начало карты графических ссылок;
NAME – имя карты;
<AREA> – определяет активные области карты;
SHAPE – определяет форму активной области;
COORDS – определяет координаты активных областей;
ALT – поясняющий текст;
HREF – определяет гиперссылку.

Тема 8. Оформление веб-страниц с использованием каскадных таблиц стилей.

1. *Введение в таблицы стилей и язык CSS*
2. *Применение таблиц стилей CSS в форматировании текста*
3. *Структурное форматирование в CSS*
4. *Свойства для форматирования таблиц в CSS*
5. *Пользовательские формы в CSS*

6. Позиционирование объектов

Вопрос 1. Введение в таблицы стилей и язык CSS.

Язык разметки гипертекста HTML, предназначенный для определения структуры документа, - не самая лучшая система управления внешним видом веб-страниц. Однако, спецификация языка разметки HTML позволяет разработчику электронных документов изменять внешний вид некоторых элементов страниц. Для этого составляются специальные правила отображения конкретного элемента в HTML-документе, называемые каскадными таблицами стилей (Cascading Style Sheets, CSS) или стилевыми шаблонами.

Каскадный список стилей (Cascading Style Sheet, CSS), на который иногда ссылаются как на вложенную таблицу стилей, представляет собой набор спецификаций (называемых правилами), которые позволяют получить полный контроль над расположением элементов на Web-странице и внешним видом ее содержимого.

Правила каскадных списков стилей могут использоваться для выбора цвета фона, графических изображений или "водяных знаков", которые будут выводиться под содержимым страницы. Кроме того, свойство position каскадных списков стилей исключает необходимость использования скрытых таблиц для расположения элементов страницы и позволяет указывать Web-браузеру, где располагать каждый конкретный элемент.

И, наконец, правила каскадных списков стилей позволяют указывать тип и выбирать размер, цвет и вид текста. Помимо использования обычного, полужирного и курсивного начертаний, каскадные списки стилей дают возможность создавать затененный текст или текст с другими специальными эффектами без преобразования символьных данных в графические изображения, загрузка которых производится гораздо медленнее.

Помимо позиционирования элементов страницы и форматирования текста, правила каскадных списков стилей позволяют управлять расстоянием между символами, словами, строками и абзацами. Более того, правила списков стилей дают возможность установить поля Web-страницы, ширину границы вокруг части текста или других элементов и управлять выравниванием "плавающего" текста и объектов (например, графических изображений), внедренных в текст.

Несмотря на то, что каскадный список стилей можно добавить в Web-документ несколькими различными способами, наверное, самый простой способ – поместить правила списка стилей непосредственно в заголовок Web-страницы, как показано на примере следующего кода:

<html>

<head>

<style type="text/css">
. . . правила списка стилей ...
</style>
</head>
<body>
... Содержимое Web-страницы ...
</body>
</html>

Для создания списка стилей необходимо написать одно или несколько правил между начальным и конечным дескрипторами <style> </style> в разделе заголовка Web-страницы (<head>).

Также определение таблицы стилей возможно еще двумя способами:

– *Ссылка на внешний файл.* Если все стилевые шаблоны для конкретного HTML-документа разместить в одном текстовом файле (с расширением css), то с помощью специального тега *<LINK>* из текущего документа можно сделать ссылку на внешний CSS-файл стилевых шаблонов, например:
<LINK REL="stylesheet" TYPE="text/css" HREF="style.ess">.
Браузер, анализируя HTML-код, обратится по указанному пути и, обнаружив указанный файл стилевого оформления, отобразит элементы страницы в соответствии с определенными правилами CSS. Следует помнить, что конструкция указания пути к внешнему CSS-файлу должна находиться в пределах раздела HEAD HTML-документа;
– *Включение в теговые конструкции.* Любой отдельный HTML-элемент может быть подвергнут форматированию средствами CSS. Для этого необходимо задать определенное правило реализации того или иного тега, например:
<P ALIGN="justify" STYLE="color: #000000; font-family: Verdana ;">
Текст параграфа...
</P>
В этом случае задано отдельное правило для конкретного параграфа.

Любое правило таблицы стилей CSS состоит из *селектора* и *определения шаблона.*

Селектор выступает в роли указателя стилевого правила для определенного HTML-элемента или внутреннего класса (идентификатора). В качестве селектора CSS выступают элементы HTML. Переопределение стиля для конкретного элемента страницы:

BODY { color: orange; }

В этом случае весь текст в пределах раздела BODY будет оранжевым. При добавлении, например, таблицы назначение стилевого шаблона пропадет для текста внутри ячеек;

Определение шаблона — это описание стилевых правил для обозначенных элементов HTML. Правила чередуются через точку с запятой и заключаются в фигурные скобки. Например:

H3 { color: blue; font-family: Tahoma, Verdana, Arial; }

В данном примере селектором является элемент заголовка H3, для шаблона которого следует такое определение: цвет — синий, шрифт — Tahoma, либо Verdana, либо Arial. Как видно из примера, для одного селектора приведено описание, содержащее два правила — по цвету заголовка и наименованию гарнитуры. Это говорит о том, что CSS позволяет группировать несколько стилевых правил для одного селектора в рамках единого описания шаблона.

Другой особенностью таблиц стилей CSS является свойство наследования стилевых правил для нескольких селекторов одновременно, например:

TD, TH, P, { text-align: justify; color: gray: font-size: 10 px; }

Такая запись назначает единый стиль отображения текстовой информации для элементов ячейки таблицы (<td>, <th>) и параграфов (<P>) а именно: тип выравнивания — по ширине, цвет — серый, размер шрифта – 10 пикселей.

Вопрос 2. Применение таблиц стилей CSS в форматировании текста.

CSS предоставляет разработчику электронных документов гораздо более широкий набор возможностей работы с текстовой информацией, нежели стандартный HTML. Помимо способов выделения текста (подчеркивание, курсив, жирное начертание, выбор гарнитуры и размер шрифта), с помощью средств CSS возможно изменять такие параметры, как межбуквенный и межстрочный интервал, тип регистра (строчные и прописные буквы) и многое другое.

В табл. 1 приведены наиболее распространенные свойства форматирования текста в CSS.

Табл. 1. Свойства форматирования текста в CSS

Свойство	**Функция и формат записи**
font-family	Выбор гарнитуры для отображения (допускается перечисление нескольких названий через запятую) *font-family: Tahoma, Arial;*
font-size	Размер шрифта *font-size: 11px;*
font-style	Выбор наклона текста (курсив). Задает стиль написания шрифта — прямой (обычный), курсив и косой (сильно наклоненный). Соответственно, может принимать одно из трех значений: normal - обычный, italic - курсив,

	oblique - косой (сильный наклон). Например: *font-style: italic;*
font-weight	Наличие/отсутствие жирного начертания. Принимает значения: normal - нормальный, bold - жирный, bolder - жирнее, lighter - светлее. Например: *font-weight: bold;*
font-variant	Преобразование строчных букв в прописные с изменением пропорций. Принимает два значения: normal - нормальный, small-caps - малые заглавные. Например: *font-variant: small-caps;*
text-decoration	Подчеркивание текста. Принимает значения: none - оформление отсутствует (используется по умолчанию); underline - текст подчеркивается; overllne - оформляется чертой сверху; line-through - текст зачеркивается; blink - текст мигающий. Например: *text-decoration: underline;*
text-align	Определение типа выравнивания текста. Может принимать следующие значения: left - выравнивает текст по левому краю, right - по правому краю, center - по центру, justify - равномерно распределяет текст по ширине всего блока (от правого до левого края). Например: *text-align: right;*
text-intend	Определяет отступ в красной строке (по умолчанию - ноль). Задается в числовых значениях или в процентах относительно ширины блока.
text-transform	Выбор регистра букв. Может принимать значения: uppercase - буквы преобразуются в прописные (верхний регистр); lowercase - в строчные (нижний регистр);

	capitalize - начальные буквы слов преобразуются в заглавные. Например: *text-transform: uppercase ;*
letter-spacing	Межбуквенный интервал. Для сжатия текста следует использовать отрицательные значения. Например: *letter-spacing: 1em;*
word-spacing	Задает интервал между словами. Работает аналогично предыдущему элементу, но изменяет интервал не между буквами, а между словами.
line-height	Междустрочный интервал. Например: *line-height: 5mm;*
Color	Цвет текста Например: *color: #FFFFFF;*
Background-color	Цвет фона текста Например: *background-color: white;*

Единицы измерения в таблицах стилей

Все единицы измерения свойств элементов в CSS можно разделить на абсолютные и относительные (табл. 2).

Абсолютные единицы измерения используются в случае, когда известны характеристики того устройства, которое отображает информацию.

Относительные единицы измерения определяют масштаб форматируемого элемента относительно других элементов, что позволяет сохранить первозданность документа при выводе на передающее устройство, характеристики которого заранее не известны.

Табл. 2. Единицы измерения в CSS

Абсолютные	**Относительные**
in (дюйм ~ 2,5 см)	*em (высота шрифта элемента)*
mm (миллиметр)	*ex (высота буквы x)*
cm (сантиметр)	px (пиксел)
pt (пункт ~ 1/7 дюйма)	*% (процентное соотношение)*
pc (пика = 12 пунктов)	

Рассмотрим на примере некоторые свойства форматирования текста в CSS (рис. 1):

<HTML>

```
<HEAD>
<TITLE>Форматирование текста в CSS</TITLE>
<STYLE TYPE="text/css">
<! —
text { font-family: Tahoma; color: 1003366;
line-height: 7mm; font-size: 12pt; }
kursiv { font-style: italic; }
SPAN. font { font-size: llpx; }
color { background-color: #CECECE; }
-->
</STYLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="black" LINK="#OOFFOO"
ALINK="#OOFFOO" VLINK="blue" STYLE="line-height: 8mm;">
<FONT CLASS="text">
CSS предоставляет разработчику электронных документов гораздо более широкий набор возможностей работы с текстовой информацией, нежели стандартный HTML. Помимо способов выделения текста (
<FONT STYLE="text-decoration: underline;"> подчеркивание </FONT>,
<FONT ID="kursiv">курсив</FONT>,
<FONT STYLE="font-weight: bold;">жирное начертание</FONT>,
<FONT STYLE="font-family: Times New Roman;">выбор гарнитуры</FONT> и <SPAN CLASS="font">размер шрифта </SPAN>),
с помощью средств CSS возможно
<FONT CLASS="color">изменять такие параметры</FONT>, как
<FONT STYLE="letter-spacing: 5px;"> межбуквенный </FONT>
и междустрочный интервал,
<FONT STYLE="text-transform: uppercase;">тип регистра</FONT>
(строчные и прописные буквы) и многое другое.
</FONT>
</BODY>
</HTML>
```

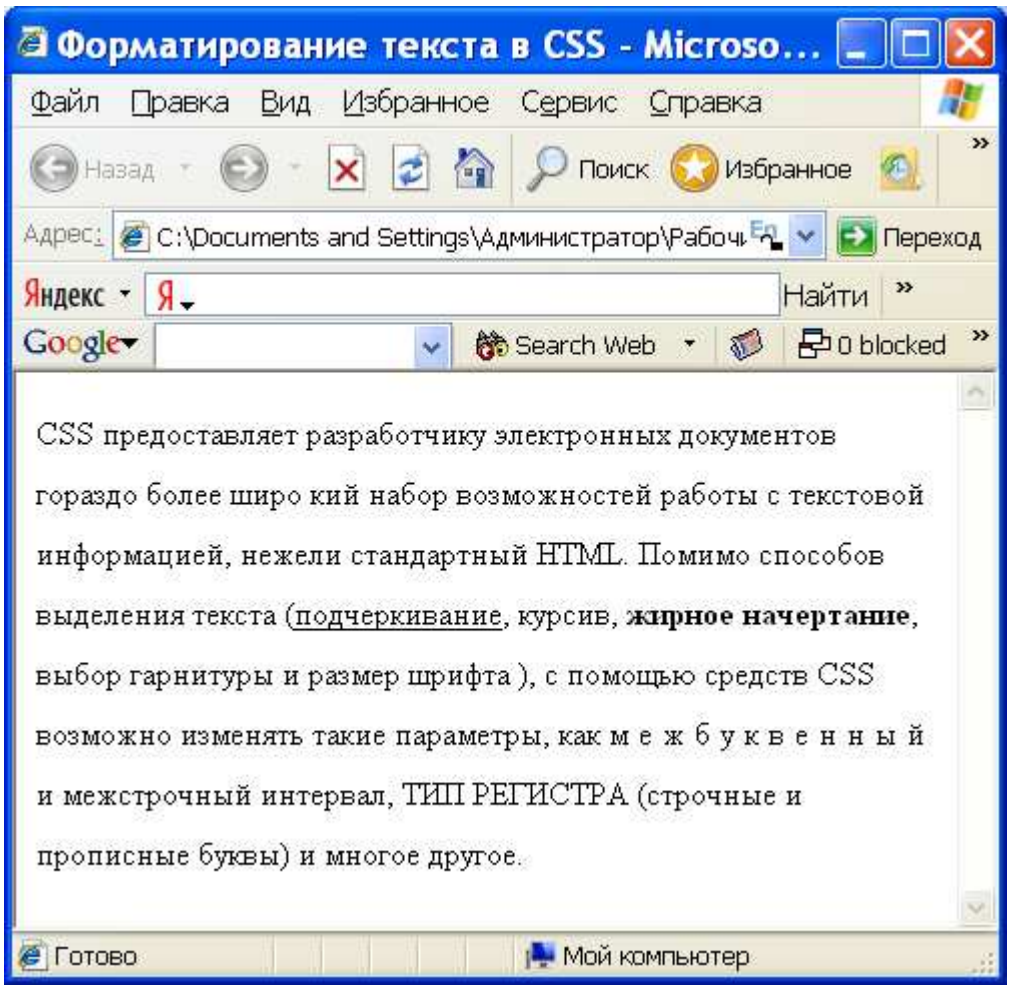


Рис. 1. Пример форматирования текста в CSS

Вопрос 3. Структурное форматирование в CSS.

Также каскадные таблицы стилей CSS привнесли много нового и заметно расширили функциональность форматирования структурных элементов электронного документа: P, DIV и пр.

В табл. 3 приведены наиболее распространенные свойства структурного форматирования в CSS.

Табл. 3. Свойства структурного форматирования в CSS

Свойство	**Функция и формат записи**
border-width	Ширина границы структурного элемента. Например: *border-width: 20px;*
border-style	Тип декоративного отображения границы элемента. Например: *border-style: solid;*
border-color	Цвет границы структурного элемента. Например: *border-color: gray;*
list-style-type	Тип нумерованного или маркированного списка. Например: *list-style-type: square;*
list-style-image	Указание пути к графическому маркеру. Например:

	list-style-image: u r l (« b u l l e t . g i f ") ;
margin	Определение размера поля относительно верхнего, правого, нижнего и левого края структурного элемента. Например: *margin: 1px 2px 3px 4px;*
padding	Определение отступа от верхнего, правого, нижнего и левого края структурного элемента. Например: *padding-top: 10em;* *padding-right: 25px;*
width	Ширина структурного элемента. Например: *width: 300px;*
height	Высота структурного элемента. Например: *height: 120px;*
background-color	Цвет фона структурного элемента. Например: *background-color: #CECECE;*
float	Плавающее расположение структурного элемента относительно других элементов. Принимает три значения: none - без обтекания, left - обтекание слева, right - обтекание справа. Например: *float: left;*

Рассмотрим на примере некоторые свойства структурного форматирования в CSS (рис. 2):

```
<HTML>
<HEAD>
<TITLE>Структурное форматирование в CSS</TITLE>
<STYLE TYPE="text/css">
    <!--
    .text { font-family: Tahoma; color: #003366;
    line-height: 7mm; font-size: 12pt;
    margin: 10px 10px 10px 10px; padding-left: 15px;
    padding-right: 15px; border-color: #000080;
    border-style: dotted; background-color: #D6D6D6;
    width: 500px; text-align: justify; }
    #kursiv { font-style: italic; }
    SPAN. font { font-size: 11px; >
    .color { background-color: #CECECE; }
    IMG { float: left; width: 100px; height: 100px; }
    LI {list-style-type: circle; }
```

```
        -- >
</STYLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="black" LINK="#00FF00"
ALINK="#00FF00"
VLINK="blue">
<p class="text">
<IMG SRC="picture.jpg ">
CSS предоставляет разработчику электронных документов гораздо
более широкий набор возможностей работы с текстовой информацией,
нежели стандартный HTML. Помимо способов выделения текста:
<p>

<UL>
<LI><FONT STYLE="text-decoration: underline;
">подчеркивание</FONT>
<LI><FONT ID="kursiv">Курсив</FONT>
<LI><FONT STYLE=" font -weight: bold; ">жирное начертание</FONT>
<LI><FONT STYLE="font-family: Comic Sans MS; ">выбор
гарнитуры</FONT>
<LI><SPAN CLASS="font">размер шрифта</SPAN>
</UL>
<P CLASS="text">
с помощью средств CSS возможно <FONT CLASS="color">изменять
такие параметры</FONT>, как <FONT STYLE=" letter-spacing: 5px;">
межбуквенный </FONT>
и междустрочный интервал, <FONT STYLE="text-transform: uppercase;">
тип регистра</FONT> (строчные и прописные буквы) и многое другое.
</p>
</BODY>
</HTML>
```

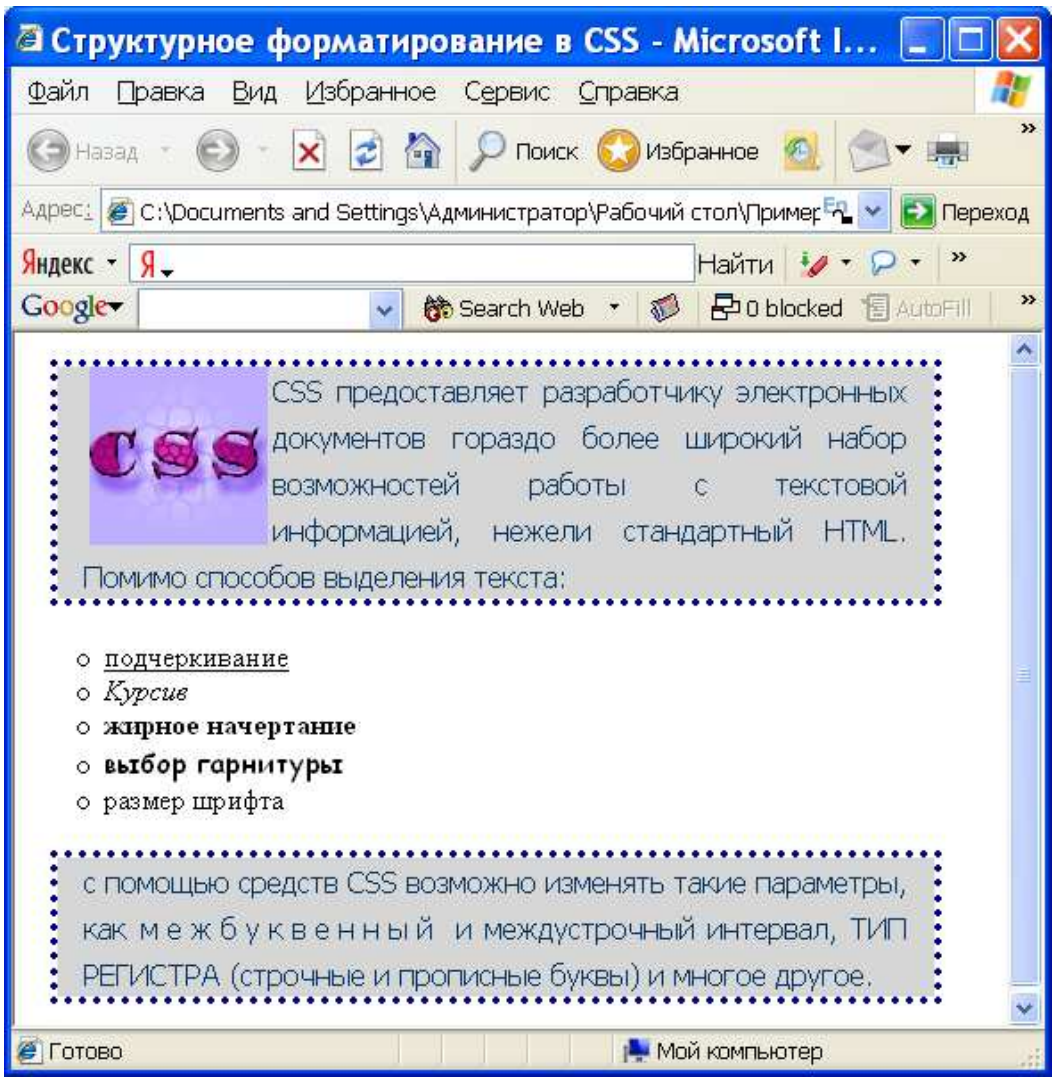


Рис. 2. Структурное форматирование в CSS

Как видно из примера, некоторые структурные элементы позволяют применять свойства CSS, относящиеся к форматированию текста: text-align, background-color и др. Таким образом, при формировании таблиц стилей допускается сочетание свойств CSS как для текстового, так и для структурного форматирования.

Вопрос 4. Свойства для форматирования таблиц в CSS.

Таблицы – одно из главных средств расположения элементов на странице. Они позволяют расположить меню рядом с текстом и могут подстраиваться под монитор компьютера. В CSS таблицы могут принадлежать одному из двух типов групп: строкам или столбцам.

Основные свойства CSS для работы с таблицами представлены в табл. 4**.**

Табл. 4. Свойства форматирования таблиц в CSS

Свойство	**Функция и формат записи**
caption-side	Указывает расположение заголовка таблицы. Это свойство имеет только элемент HTML <caption>. Может

	принимать четыре значения: top - заголовок располагается над таблицей, bottom - под таблицей, left - слева от таблицы, right - справа от таблицы. Например: *caption {* *caption-side: bottom;* *}*
empty-cells	Указывает, отображать или нет границы пустых ячеек. Может принимать значения: show - показывать, hide - спрятать.
table-layout	Используется элементами таблицы. Принимает значения: auto - ширина таблицы и ее ячеек вычисляется браузером автоматически, fixed - ширина таблицы фиксирована. Непоместившиеся данные обрезаются или выходят за рамки таблицы, если не задано свойство overflow.
vertical-align	Указывает вертикальное выравнивание ячейки таблицы. Принимает значения: baseline - выравнивание по основной линии блока, top - по верхнему краю (в таблицах), middle - по середине, bottom - по нижнему краю (в таблицах).
text-align	Горизонтальное выравнивание содержимого ячеек.
border	Задает стиль границы и размер.
border-collapse	Позволяет выбрать модель границ таблицы. Значение *separate* задает модель с отдельными границами. Значение *collapse* задает модель с пересекающимися границами.
border-spacing	Указывает горизонтальное и вертикальное расстояние между границами соседних ячеек.
border-style	Задает стиль границ таблицы. Принимает следующие значения: none – граница отсутствует; hidden – аналогично значению none; dotted – граница представляется рядом точек; dashed – граница представляется рядом коротких линейных сегментов;

	solid – граница представляется единым сегментом линии; double – граница представляется двумя сплошными линиями; groove – граница выглядит вдавленной; ridge – граница выглядит выпуклой; inset – в модели с отдельными границами весь блок выглядит вдавленным; outset – в модели с отдельными границами весь блок выглядит выпуклым.

Рассмотрим на примерах свойство border-style с его возможными значениями:

border-style:none;

border-style:hidden;

border-style: dotted;

border-style:dashed;

border-style:solid;

border-style:double;

Вопрос 5. Пользовательские формы в CSS.

Язык разметки HTML предлагает разработчику электронных документов широкий набор возможностей по организации пользовательских форм. Однако при развитой функциональности внешний вид всевозможных текстовых полей, переключателей, ниспадающих меню зачастую оставляет желать лучшего. Это досадное упущение компенсируется посредством таблиц стилей CSS.

В табл. 5. приведены наиболее распространенные свойства пользовательских форм в CSS.

Табл. 5. Свойства для форматирования форм в CSS

Свойство	**Функция и формат записи**
Background-color	Цвет фона элемента формы. Например: *background-color: white;*
border-width	Толщина рамки вокруг элемента формы. Например: *border-width: 1px;*
border-style	Тип декоративного отображения рамки вокруг элемента формы. Например: *border-style: solid;*
border-color	Цвет рамки вокруг элемента формы. Например:
border-top border-right border-bottom border-left	Толщина рамки для верхней, правой, нижней и левой части элемента формы. Например: *border-top: 1px;* *border-right: 2px;* *border-bottom: 3px;* *border-left: 4px;*
font-family	Свойства текста внутри элемента формы (аналогичны свойствам форматирования обычного текста) . Например: *font-family: Tahoma;*

font-size	Размер поля вокруг элемента формы (вверху, справа, внизу, слева). Например: *font-size: 12em;*
font-weight	Размер внутреннего отступа от края элемента формы. Например: *font-weight: bold;*
margin	Ширина элемента формы. Например: *margin: 2px 2px 2px 2px*
padding	Высота элемента формы. Например: *padding-top: 1px;* *padding-right: 2px;*
width	Цвет фона элемента формы. Например: *width: 200px;*
height	Толщина рамки вокруг элемента формы. Например: *height: 50px;*

Рассмотрим на примере некоторые свойства пользовательских форм в CSS:

```
<HTML>
<HEAD>
<TITLE>Пользовательские формы в CSS</TITLE>
<STYLE TYPE="text/css">
<!--
   INPUT.text { font-family: Tahoma; color: #003366;
            font-size: 12px; padding-left: 5px;
            padding-right: 5px; border-color: black;
            border-style: solid; border-width: 1px;
            background-color: #F9F9F9; width: 250px; }
   TEXTAREA. field { font-family: Verdana, Arial;
                color: #003366; font-size: 12px;
                padding-left: 5px; padding-right: 5px;
                border-color: black; border-style: solid;
                border-width: 1px; background-color: #F9F9F9; }
   INPUT.button { font-family: Tahoma; coLor: #FFCC00;
              font-size: llpx; font-weight: bold;
              border-color: white; border-style: solid;
              border-width: 1px; background-color: gray;
              width: 200px; }
   BODY, TD { font-family: Tahoma; color: #003366;
           font-size: 11px; }
   TH { color: #FFFFFF; }
-->
</STYLE>
```

```
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="black" LINK="#OOFFOO"
ALINK="#OOFFOO"
VLINK="blue">
<H2 ALIGN="center">ПОЛЬЗОВАТЕЛЬСКАЯ АНКЕТА</H2><FORM
NAME="user_anketa" ACTION="/cgi-bin/anketa.cgi" METHOD="post"
ENCTYPE="multipart/form-data">
<TABLE BORDER="0" CELLPADDING="5" CELLSPACING="3"
ALIGN="center" WIDTH="98%">
<TR>
<TH BGCOLOR="#999999" WIDTH="50%">Личные данные</TH>
<TH BGCOLOR="#999999" WIDTH="50%">Увлечения</TH>
</TR>
<TR>
<TD VALIGN="top" BGCOLOR="#EEEEEE" WIDTH="50%">
Ваше имя : <BR>
<INPUT TYPE="text" SIZE="30" MAXLENGTH="35" NAME="name"
CLASS="text">
<BR>
Ваш пароль : <BR>
<INPUT TYPE="password" SIZE="30" MAXLENGTH="35"
NAME="password"
CLASS="text">
<BR>
Ваша фотография : <BR>
<INPUT TYPE="file" SIZE="18" NAME="photo" CLASS="text">
</TD>
<TD VALIGN="top" BGCOLOR= "#EEEEEE" WIDTH="50%">
<INPUT TYPE="checkbox" NAME="hobby" VALUE="computers"
CHECKED>Компютеры
и Интернет
<BR>
<INPUT TYPE="checkbox" NAME="hobby" VALUE="art">Литература и
искусство
<BR>
<INPUT TYPE="checkbox" NAME="hobby" VALUE="music">Музыка и
танцы
<BR>
<INPUT TYPE="checkbox" NAME="hobby" VALUE="auto">Автомобили
<BR>
<INPUT TYPE="checkbox" NAME="hobby" VALUE="sport">Спорт и
активный отдых
```

```
</TD>
</TR>
<TR>
<TH BGCOLOR="#999999" WIDTH="50%">Место жительства</TH>
<TH BGCOLOR="#999999" WIDTH="50%">Комментарии</TH>
</TR>
<TR>
<TD VALIGN="top" BGCOLOR="#EEEEEE" WIDTH="50%">
<SELECT NAME="city" SIZE="5" STYLE="font-family: Verdana; font-size:
11px;">
<OPTION VALUE="" STYLE="background-color: #CECECE; "> ---
Выберите город ---
<OPTION VALUE="Moscow">Москва
<OPTION VALUE="Spb" STYLE="background-color: #CECECE;">Санкт-
Петербург
<OPTION VALUE="Samara">Самара
<OPTION VALUE="Ryazan" STYLE="background-color:
#CECECE;">Рязань
<OPTION VALUE="Krasnoyarsk">Красноярск
<OPTION VALUE="Komsomolsk" STYLE="background-color:
#CECECE; ">Комсомольск-на-Амуре
<OPTION VALUE="Vladivostok">Владивосток
</SELECT>
<BR><BR>
<SELECT STYLE="font-family: Verdana; font-size: 11px;">
<OPTION VALUE=""> ---Выберите страну ---
<OPTION VALUE="Russia">Россия
<OPTION VALUE="England">Англия
<OPTION VALUE="USA">США
<OPTION VALUE="Cermany">Германия
<OPTION VALUE="Fгance">Франция
</SELECT>
</TD>
,<TD VALIGN="top" BGCOLOR="#EEEEEE" WIDTH="50%">
<TEXTAREA COLS="25" ROWS="5" NAME="comments"
CLASS="field"></TEXTAREA>
<BR>
<INPUT TYPE="radio" NAME="site" VALUE="first" CHECKED>Я впервые
на этом
сайте
<BR>
<INPUT TYPE="radio" NAME="site" VALUE="not_first">Я тут не в
```

первый раз
</TD>
</TR>
<TR>
<TD COLSPAN="2" ALIGN="center" BGCOLOR="#999900">
<INPUT TYPE="submit" VALUE="Отправить"
CLASS="button"> <INPUT TYPE="reset" VALUE="Очистить"
CLASS="button">
</TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>

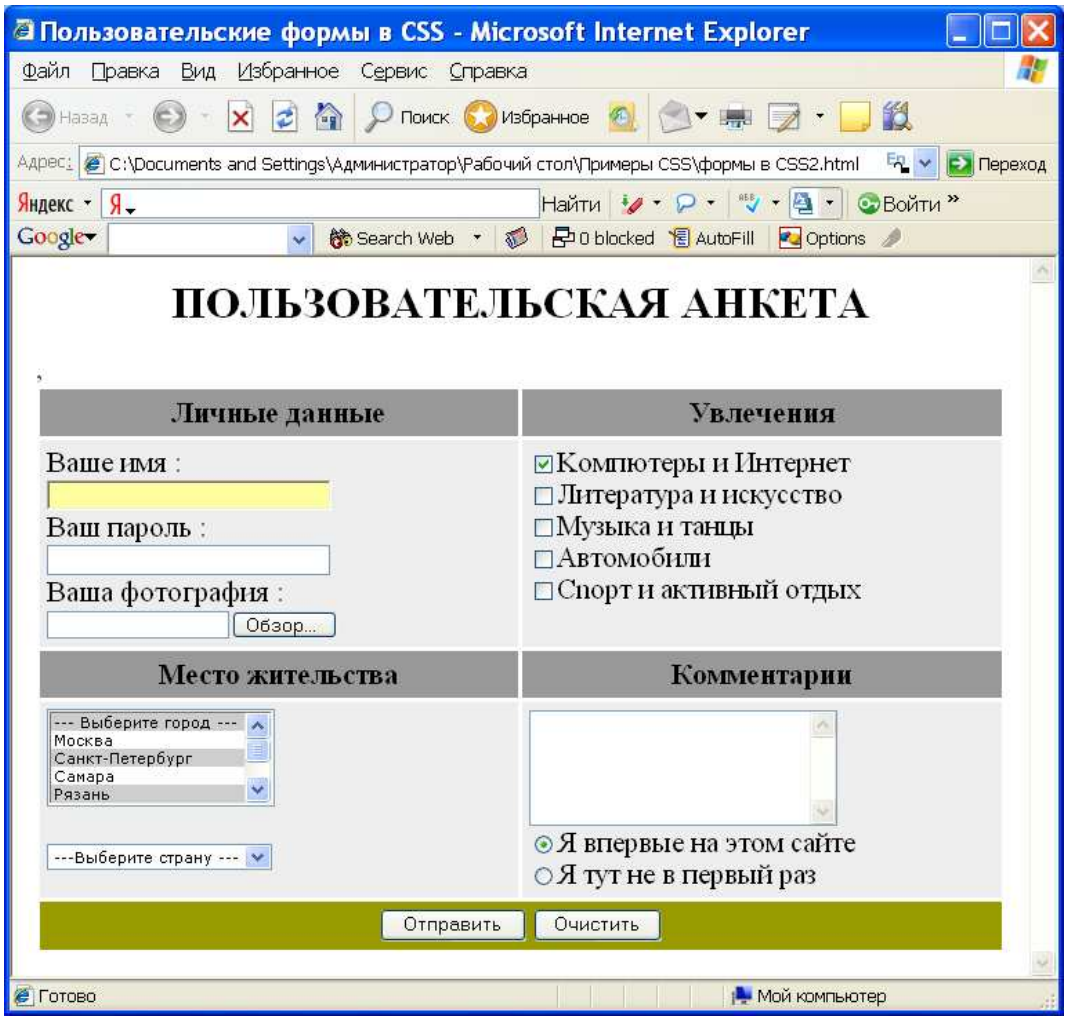


Рис. 4. Пользовательские формы в CSS

Вопрос 6. Позиционирование объектов.

Исходя из концепции языка разметки HTML, все элементы документа выводятся браузером в той последовательности, в какой размещены теговые конструкции в коде. CSS позволяет задавать порядок и последовательность отображения тех или иных HTML-элементов в

зависимости от определенных событий на странице или манипуляций, осуществляемых со стороны пользователя.

Другими словами, с помощью средств CSS можно прибегнуть к позиционированию (пространственному расположению) необходимых объектов в пределах электронного документа.

Существуют два типа визуального позиционирования элементов: абсолютное и относительное. Для описания каждого типа используются специальные свойства position, top и left.

Абсолютное позиционирование

Абсолютное позиционирование подразумевает четкое фиксирование выбранного элемента на странице, независимо от остальных элементов документа, например:

<IMG SRC="picture.gif" WIDTH="100" HEIGHT="100" ALT="Рисунок" STYLE="position: absolute; top: 10px; left: 25px;">

В данном случае графическое изображение абсолютно спозиционировано и размещается в 10 пикселях от верхнего и в 25 пикселях от левого края своего родительского элемента (в качестве родительского элемента выступает верхняя левая точка структуры документа).

Относительное позиционирование

Относительное позиционирование позволяет расположить выбранный объект в зависимости от размещения остальных элементов документа (т. е. относительно других объектов страницы), например:

<DIV ID="text" STYLE="position: relative; top: 50px; left: 50px;">

Тема 9. Основы веб-дизайна.

1. *Что такое дизайн сайта и зачем он нужен?*
2. *Основные аспекты формирования стиля веб - дизайна*
3. *Понимание роли графики в web-дизайне*
4. *Анимация, звук и видео в web-дизайне*

Вопрос 1. Что такое дизайн сайта и зачем он нужен?

В основе создания Web-страниц лежат простейшие языки программирования, базирующиеся на HTML-кодах. HTML – это язык разметки документа, позволяющий форматировать и верстать страницы, используя самую различную информацию (текстовую, графическую, звуковую). Заметим, что совсем до недавнего времени созданием Web-страниц (что совершенно естественно) занимались только программисты. Однако вопросы эстетического характера при оформлении Web-страниц и сайтов отступали на второй план. Сейчас же требования к этим документам настолько возросли, что к процессу их создания подключились дизайнеры-художники.

Вот и появилась новая профессия Web-дизайнер. Однако Web-дизайнеров так мало, и Internet по-прежнему кишит не удобно сделанными, не привлекательными, мало интересными страницами. Появление специальных книг, освещающих вопросы художественной композиции и цветоведения, дизайна интерфейсов, шрифтов и текстов в применении к Web-страницам, немного облегчило работу программистов. Вместе с тем, не имея навыков к дизайнерской деятельности, применить полученные знания на практике им бывает весьма затруднительно. Что же касается художников-дизайнеров, то при определенном желании освоить элементарные азы программирования совсем не сложно.

Тем более что в настоящее время появляется все больше вспомогательных программных продуктов, облегчающих процесс написания кода. Кроме того, подобную проблему иногда решают очень просто. Над созданием Web-документов одновременно работают программисты и художники, так же, как и многие специалисты. Исключает ли это знание методов работы напарника? Конечно же, нет. Особенно для разработки макета сайта, где решаются композиционные задачи.

Обратим внимание на то, что важно для Web-дизайнера. А именно, далеко не только знание языка программирования HTML. Хотя, разумеется, профессиональный Web-дизайнер должен знать все тонкости и нюансы внутренней структуры своей Web-страницы. По этому поводу можно заметить, что внимание к дизайну страницы и программированию должно быть разделено пополам. Поэтому, заранее говорим вам, что стоит обратить особое внимание на создание дизайна страницы.

Web-дизайнеры часто спорят о выборе программных средств

компьютерной графики. Некоторые утверждают, что важны только профессиональные инструменты графического дизайна. Другие же выражают распространенную точку зрения, что важны не инструменты, а проблемы дизайна в Internet. Инструменты, в конечном счете, ничего не значат. И все же, если вы хотите стать профессионалом в области Web-дизайна, то необходимо в совершенстве владеть не только компьютерными инструментами высшего уровня, но и знаниями классического дизайна. Обойтись без него нам не представляется возможным, поскольку дизайн несет в себе различные виды информации. И мы видим в нем аналогию между языком Web и грамотным и эффективным подходом к любой визуальной коммуникации, чтобы говорить языком изобразительных форм.

Именно художественно-выразительные методы дизайна помогут реализовать творческие замыслы, проявить оригинальность мышления в процессе поиска и решения практических задач по созданию Web-страницы или сайта

Знание таких продуктов как Adobe и Macromedia дают возможность работать с фотографиями, создавать изображения с помощью цвета и текста, сканировать их, а также анимировать, совершенствовать, оптимизировать графику. А вот дальше можно определить три уровня обучения Web-дизайну. На первом уровне Вы познакомитесь с общими сведениями о Web-программировании. Сюда входит изучение по созданию простых Web-страниц с помощью HTML-кода, разметкой текста, ссылки, приемы форматирования текста, как сохранить страницу и просмотреть ее в браузере. Не менее важны сведения подготовки изображений для Web-страницы с помощью графического редактора Adobe Photoshop.

На втором уровене изучения Web-дизайна, необходимо изучить основные технические особенности Web – это теги, атрибуты, ссылки, фреймы и т.д.

И, наконец, на третьем уровне можно стать почти Web–профессионалом, если внимательно изучить вопросы разработки контента (содержания) сайта и его структуры.

Вопрос 2. Основные аспекты формирования стиля веб - дизайна

Перед тем, как составить свою таблицу стилей веб - дизайна, надо выделить основные аспекты формирования этих стилей. Основоположные составляющие каждого из них. Проще говоря, выделить те признаки, по которым различаются эти самые стили.

Композиционное решение. Взаиморасположение объектов, расстановка, соотношение размеров, форм и т. д. Композиционное решение сайта сразу определяет положение основного контента, положение меню навигации, способы прокрутки на странице

(горизонтальное, вертикальное, внутри блоков).

Цветовая схема. Цвет, составляющий дизайн сайта, является особенно важным компонентом. Цветовая схема сайта формирует впечатление от сайта, создает настроение. Естественно, что все цвета сайта должны соответствовать общей идее сайта, целевой аудитории. Важно, чтобы цвета сайта гармонировали друг с другом и ни в коем случае не раздражали посетителей. Создание цветовой гармонии — одна из первых задач дизайнера.

Шрифтовые схемы. Выбранному стилю дизайна всегда должен соответствовать шрифт. Шрифт, его цветовая схема, верстка просто обязаны идти в общем контексте сайта. Выбор категории и гарнитуры, размера влияет на восприятие дизайна не меньше "красивой картинки".

Наличие/отсутствие определенных графических символов и их модификаций. Каждому стилю веб дизайна присущи определенные элементы. Создавая макет сайта в каком-нибудь стиле, стоит задуматься о стилевом соответствии. К графическим символам на сайте относятся логотип, пиктограммы, иконки и т. д. Глупо выглядел бы веб сайт, выполненный в органическом стиле, с плавными природными линиями, с легким дизайном и с логотипом в стиле хай-тек.

Способы обработки изображений. Технические приемы. Для каждого стиля веб-дизайна присущи свои методы и приемы. Например, векторная или растровая обработка изображений. Разумное применение анимации, различных технологий (Flash, JavaScript, DHTML...). Различные методы художественной отрисовки элементов. На одних сайтах допустима компьютерная графика, на других идеально подходит качественно обработанная фотография, на третьих необходима 3D визуализация.

Вопрос 3. Понимание роли графики в web-дизайне

Графику на Web-страницы помещают по тем же причинам, что и картинки в буклеты или журналы — чтобы помочь читателям лучше понять то, что вы пытаетесь сказать. Назначение графики — дополнять текст. Например, Web-сайт, разработанный для "продвижения" крупного университета, например, Университета штата Калифорния в Беркли, будет содержать изображения студенческого городка, аудиторий, и, возможно, фотографии преподавателей и других сотрудников университета. Посетители сайта не только прочтут о преимуществах учебы в этом университете, но смогут также увидеть его корпуса, общежития, спортивные площадки. Просмотрев страницы, посетители получат полное представление о студенческой жизни в Беркли. Один лишь текст не может передать столько же информации, как текст в сочетании с несколькими хорошо подобранными изображениями. При использовании описательного текста, пусть даже и хорошо написанного, недостаточно полагаться на воображение читателя, в котором создается образ

описываемого вами на странице. И, наоборот, когда для усиления текста вы используете рисунки, ваши читатели видят именно то, что вы имели в виду.

Перед тем как добавлять на Web-страницу графику, надо задать себе вопрос: какие из изображений лучше всего проиллюстрируют то, что вы хотите сказать? Если цель Web-сайта — привлечь новых студентов, выберите те изображения, которые с наибольшей вероятностью помогут достичь этой цели. Создавая Web-сайт, никогда не забывайте о своей ответственности как дизайнера. Не выплескивайте на своих посетителей слишком много информации на одной странице, и не предоставляйте им слишком много альтернат сразу. Вам решать, что и когда увидят ваши посетители. Предоставляя посетителям чересчур большой объем данных за один раз, вы рискуете вызвать "информационную перегрузку", что может привести к формированию у ваших посетителей ошибочного мнения или совершения ими опрометчивого выбора.

Создание Web-графики

Включая в свою Web-страницу изображения, можно обратиться к нескольким источникам графики. Можно приобрести компакт-диск с готовыми изображениями, или же выгрузить Web-изображения с сайтов, торгующих графикой (например, http://photosphere.com или http://creative.gettyimages.com).

Еще один вариант - сделать фотографии с помощью цифрового или пленочного фотоаппарата, Затем можно сохранить на диске изображения, отснятые цифровой камерой, или с помощью сканера отсканировать предварительно напечатанные фотографии в файлы на своем компьютере. И, наконец, можно создать цифровые аппликации и иллюстрации с помощью дизайнерских приложений, таких как Adobe Photoshop или Image Ready. За исключением варианта покупки готовой графики, понадобятся одна или несколько программ, которые помогут в обработке и подготовке изображений к использованию их в Internet. Графические приложения не только экономят время при подготовке изображений, но также дают возможность управлять содержимым изображения.

Учет скорости доступа в Internet

Еще одним важным моментом при включении графики в Web-страницу являются временные задержки, вызванные загрузкой файлов изображений. Хотя такие варианты подключения к Internet, как цифровые, абонентские линии (Digital Subscriber Line — DSL) и модемы для физических линий позволяют пересылать большие объемы данных по Internet за короткие промежутки времени, не у всех есть доступ к высокоскоростному подключению к Internet. Всякий раз, когда добавляется на Web-страницу графика, увеличивается время, необходимое для полной загрузки страницы и ее содержимого с Web-сервера.

Графика — неотделимая составляющая Internet. Однако если из-за больших графических файлов посетители покидают сайт, не дожидаясь загрузки изображений, вряд ли можно сказать, что графика приносит пользу.

Достижение баланса между текстом и графикой на Web-странице

Иногда одних лишь слов недостаточно для передачи сообщения. Однако создание Web-страницы, содержащей и текст, и рисунки требует больших усилий, нежели просто вставки графики в страницу в произвольном порядке. Вам нужно найти правильное соотношение между информацией, представленной в виде текста, и информацией, передаваемой посредством изображений. Однако даже самые выразительные изображения требуют пояснений.

Текст передает важное историческое событие; вместе с фотографией, два этих элемента создают моментальный снимок во времени и генерируют более действенное сообщение, чем каждый из них в отдельности.

Цвет — отличное средство для привлечения внимания посетителей. Цветные изображения привлекают внимание лучше, чем черно-белые, и, кстати, публикация цветного изображения на Web-странице обойдется ничуть не дороже, чем черно-белая картинка.

Привлечь внимание посетителя к определенным картинкам на странице можно с помощью одной хитрости: комбинирования цветных изображений с черно-белыми. Поместите изображение с доминирующими цветами на страницу, где остальные (менее важные) изображения являются черно-белые. Поскольку человека тянет сначала посмотреть цветную картинку, можно сфокусировать взгляд посетителя на изображении, передающем суть вашего сообщения, когда он впервые пробегает глазами по вашей странице, а затем позволить посетителю перейти к другим изображениям, которые окончательно доведут до его сознания то, что вы хотели сказать.

Комбинировать текст и изображения на Web-странице вовсе не сложно. Все, что нужно делать, это выбирать выразительные картинки и затем следовать нескольким простому правилу: всегда подбирать изображения, имеющие отношение к тексту.

Вопрос 4. Анимация, звук и видео в web-дизайне

Сегодня практически невозможно путешествовать по Internet, не встречая Web-страниц, содержащих какую-то анимацию, звук, видео или комбинацию всего перечисленного. От простых анимированных GIF-изображений до Flash-фильмов и даже полноэкранных интерактивных изображений со звуком и движением, в анимации, встраиваемой в Web-страницы, заложен огромный потенциал "оживления" вашего дизайна. К сожалению, избыток анимации и звука способен вызывать у посетителей

раздражение и отвлекать их внимание. Развитие технологий в Internet обусловлено стремлением к достижению двух противоположных целей:

- В попытках угодить посетителям авторам Web-страниц приходится передавать через Web все более и более сложные типы содержимого (звук, анимацию и видео).
- Чтобы добраться до посетителей Web-сайта, размеры файлов содержимого должны быть достаточно небольшими для обеспечения быстроты загрузки, или же посетители, потеряв терпение, отправятся на другие сайты.

Посетители требуют все больше и больше информации. Однако, требуя развлечений и информации, они зачастую не желают провести несколько лишних секунд в ожидании поступления дополнительного содержимого. К счастью, две противоположные цели — дизайн и ожидание — стимулируют развитие новых технологий, позволяющих вам предоставлять посетителям больше информации быстрее, чем когда-либо раньше.

Анимация фактически представляет собой серию неподвижных изображений, демонстрируемых быстро друг за другом. Иллюзия движения создается при удержании вашими глазами одного изображения (инертность зрительного восприятия), в то время как программа, воспроизводящая анимацию, отображает на экране следующее изображение. Чтобы создать эффект плавного движения, программа должна отображать каждое изображение (или кадр) в течение доли секунды после удаления предыдущего, так, чтобы глаз видел новое изображение, как только предыдущее изображение исчезнет.

Основная роль Web-анимации состоит в привлечении внимания посетителей к вашему Web-сайту. Однако Web-анимация также может развлекать и информировать. Маленькие дети (в особенности те, которые только начинают читать) обожают движение, и их внимание всегда сосредоточивается на нем. Следовательно, с помощью анимации можно захватить и надолго сосредоточить внимание маленького ребенка на концепции, которую вы пытаетесь передать.

Кроме того, на Web-странице можно применить анимацию и звук для описания этапов сложной процедуры или процесса. Например, анимированная последовательность (или, может быть, видеоролик), демонстрирующая процедуру программирования видеомагнитофона на запись фильма, в сопровождении голоса диктора, поясняющего действия, намного эффективнее текстового документа, описывающего эту же процедуру. Конкретное предназначение разрабатываемой вами Web-страницы в значительной мере будет определять тип анимации и программного обеспечения, необходимого для ее создания.

Анимация — важная составляющая дизайна Web-сайта, и у вас,

Web-дизайнера, есть возможность выбрать из нескольких методов те, которые помогут вам в достижении конкретных стоящих перед вами целей.

Когда использовать и не использовать анимацию на web-страницах

Как уже упоминалось, анимация привлекает внимание, развлекает и информирует. Однако сколь бы чудесной ни была анимация, при использовании анимации на Web-страницах следует соблюдать и определенную осторожность:

- Используйте анимацию для привлечения внимания, но только не переборщите. Анимация захватывает внимание, но избыток ее может превратиться в проблему. Если вы создаете анимацию для привлечения внимания посетителя, воздержитесь от использования более чем одного, возможно, двух анимированных объектов на каждой странице.
- Используйте анимацию, чтобы подчеркнуть общение Web-страницы, проследите затем, чтобы она действительно приносила пользу. Старайтесь использовать анимацию, перекликающуюся со статическим содержимым страницы. Например, вносит ли вращающийся логотип в верхнем левом углу окна браузера должный вклад в дизайн и сообщение страницы, или же вы поместили его туда просто чтобы доказать, что вы это умеете?
- Не используйте анимацию там, где она отвлекает внимание. Избегайте использования анимации, на Web-страницах, содержащих большие объемы текста. Анимация оказывает влияние на концентрацию внимания пользователя и отвлекает его от чтения текста. Анимация привлекает внимание, однако избыток анимации мешает посетителю сосредоточиться.
- Не используйте анимацию ради анимации. Поскольку файлы анимации имеют большие размеры и, следовательно, им требуется больше времени на загрузку и отображение, используйте анимацию только там, где она действительно вносит в страницу весомый вклад.

Работа с Gif-анимацией

Анимация с использованием формата графического обмена (GIF — Graphic Interchange Format) — наиболее популярная на сегодняшний день форма анимации в Internet. Такая популярность вызвана несколькими причинами:

- GIF-анимацию легко создавать. GIF-анимацию можно создать с помощью ряда приложений для создания изображений и Web-дизайна, таких как Adobe ImageReady и Macromedia Fireworks.

Кроме того, из сайтов условно-бесплатных программ (shareware), в числе которых http:// www.shareware.com и http://www.tucows.com, можно выгрузить условно бесплатные программы, позволяющие создавать GIF-анимацию на основе существующих изображений. Среди популярных условно-бесплатных программ можно назвать GIF Builder и (Macintosh) GIF Construction Set (Windows).

- GIF-анимация не требует специального программного обеспечения для браузеров. Почти все Web-браузеры воспроизводят GIF-анимацию, что означает, что браузеру не нужно специальное программное обеспечение или подключаемый модуль.
- GIF - это стандартный формат Web-файлов. Все браузеры, поддерживающие графические изображения, будут отображать GIF-файлы. В редких случаях, когда браузер не поддерживает GIF-анимацию, браузер будет отображать статическую версию анимации (обычно первый кадр анимационной последовательности, хранящейся в GIF-файле).
- GIF-анимация использует потоковые технологии. Начав загрузку GIF-анимации, Web-браузер отображает кадры по мере их получения. Посетителю не приходится дожидаться окончания загрузки файла, чтобы увидеть анимацию.

Анимированные GIF-изображения работают так, как традиционная, рисованная покадровая анимация, которую создает компания Уолта Диснея и ей подобные. Каждый GIF-файл содержит определенное число ячеек, где каждая ячейка представляет собой часть анимации. При сохранении GIF-файла программа "складывает" изображения (то есть ячейки) одно поверх другого и сохраняет их все в одном файле.

Поскольку анимация привлекает внимание, GIF-анимация широко используется в маркетинге и рекламе. Всем нам приходится лицезреть на вездесущие рекламные баннеры, "болтающиеся" вверху практически каждой коммерческой Web- страницы, рекламирующие все что угодно, от мягких игрушек до подписки на серьезные журналы.

Еще одна причина широкого распространения GIF-анимации состоит в простоте ее вставки в HTML-документ. GIF-анимация встраивается в Web-страницы с помощью того же дескриптора <img>, который используется для вставки графики и иллюстраций, как показано ниже:

<img src="neon.gif" width="50” height="50">

Информация, поясняющая браузеру, как построить анимацию GIF-файла (скорость, количество циклов), внедрена в сам файл, а не записана в HTML- коде.

Тема 10. Основы языка JavaScript: Переменные и типы данных, условные операторы и операторы циклов, функции и объекты.

1. *Введение в JavaScript*

2. *Встраивание JavaScript-кода в HTML-документы*
3. *Переменные и типы данных в JavaScript*
4. *Выражения и операторы в JavaScript*
5. *Функции и объекты в JavaScript*

Вопрос 1. Введение в JavaScript

Файлы Web-страниц, написанные на языке HTML (Hypertext Markup Language - язык разметки гипертекста), определяют, внешний вид в окне браузера таких элементов, как текст, таблицы и изображения. При этом HTML работает только со статическими элементами.

JavaScript – это язык сценариев (scripting language), который может применяться в файлах Web-страниц наряду с HTML, например, для придания Web-страницам свойств диалоговых окон. Эти интерактивные возможности позволяют разработчику динамически управлять элементами Web-страниц. JavaScript можно также применять для изменения элементов Web-страницы
при взаимодействии с ней пользователя. Например, можно предусмотреть, чтобы текст, отображаемый в текстовом поле формы, изменялся, когда пользователь щелкает на кнопке.

Сценарий JavaScript может выполняться в браузере без каких-либо дополнительных программных средств.

Вопрос 2. Встраивание JavaScript-кода в HTML-документы

Обычно сценарии внедряются в HTML-документ тремя стандартными способами:

- в виде гиперссылки;
- в виде обработчика события;
- в рамках элемента *SCRIPT*.

Разберем способы встраивания сценариев в веб-страницы по порядку.

Встраивание сценариев в гиперссылку

Как известно, для создания на веб-страницах гиперссылок применяется элемент *A*, у которого есть атрибут *href* для указания адреса страницы или файла, куда приведет гиперссылка. Однако вместо адреса данный атрибут может содержать и JavaScript-сценарий, если перед сценарием поставить слово *JavaScript:* (со знаком :).

Рассмотрим пример такой гиперссылки:

* Нажмите сюда *

Когда пользователь щелкает кнопкой мыши на гиперссылке, интерпретатор сценариев JavaScript получает следующий код:

alert ('Привет, мир!');

В результате выполнения этого кода на экране появляется окно с сообщением «Привет, мир!».

Здесь сценарий состоит из вызова всего одной функции alert (),

которая выводит на экран окно с текстом, полученным в качестве параметра. Следует помнить, что если браузер не поддерживает JavaScript или поддержка этого языка отключена, гиперссылка будет отображена, но работать не будет, что может вызвать недоумение у пользователей. Поэтому во избежание проблем это следует учесть.

Встраивание сценариев для обработки событий

При работе с такой графической операционной системой, как Windows, вы могли заметить, что она всегда готова отреагировать на ваши действия: щелкнули кнопкой мыши на значке программы – программа запустилась, нажали клавишу в текстовом редакторе – появился соответствующий символ и т.д. Все это некие события, которые приходят от мыши, клавиатуры, таймера и многих других устройств. Аналогично и браузер может реагировать на различные события.

Поведение каждого элемента зависит от событий, которые постоянно возникают в браузере. Например, пользователь щелкнул кнопкой мыши на тексте, переместил указатель мыши, загрузил HTML-документ. Браузер сам решает, как реагировать на каждое событие. Однако можно перехватывать отдельные события и задавать новое поведение при их наступлении. В языке HTML у многих элементов есть атрибуты, соответствующие различным событиям. Эти атрибуты могут содержать код языка JavaScript, выполняемый в ответ на событие. Например:

* это простой текст *

Элемент *P* просто объединяет текст в параграфы, однако мы возложили на него еще одну функцию. Определив атрибут *onclick*, текст будет реагировать на щелчок кнопкой мыши (рис. 1).

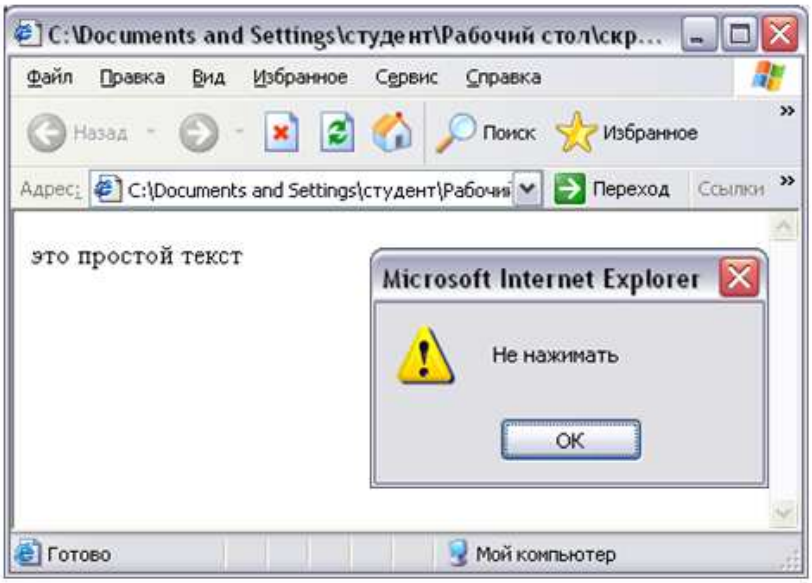


Рис. 1. После щелчка кнопкой мыши на тексте появляется сообщение

В языке HTML практически каждый элемент имеет атрибуты для обработки каких-либо событий. Для обработки событий данным

атрибутам назначается код на языке JavaScript, который выполняется, когда произойдет событие. Благодаря возможности обработки событий можно создавать интерактивные страницы, отвечающие на действия пользователей.

Встраивание сценариев в рамках элемента Script

Кроме возможности обработки событий, для внедрения сценариев в веб-страницы в языке разметки HTML есть специальный элемент Script. Браузер, встретив тег < Script > в процессе разбора кода, интерпретирует текст, последующий до закрывающего тега < /Script >, как сценарий на каком-либо языке. Для указания языка сценария используется атрибут type. Например, для указания языка JavaScript:

```
<Script type="text/javaScript">
Здесь располагается код сценария
</script>
```

При использовании сценариев JavaScript можно этот атрибут не указывать, так как значение *"text/javaScript"* является значением по умолчанию.

Кроме того, иногда для указания типа языка используют атрибут *language*. Например:

```
<html>
<body>
<br>
Это обычный HTML-документ
<br>
<script language="JavaScript">
document.write("Это и есть JavaScript!" )
</script>
<br>
Выходим обратно в HTML
</body>
</html>
```

Вопрос 3. Переменные и типы данных в JavaScript

Переменная – это имя, присваиваемое ячейке памяти компьютера, которая хранит определенные данные во время работы сценария. Переменные есть в каждом языке программирования, они облегчают программисту работу по манипулированию с данными. В отличие от других языков, переменные в JavaScript не должны иметь заранее определенный и неизменный тип. Тип переменной может изменяться в процессе выполнения программы и зависит от типа помещаемых в него данных.

Имена переменных не должны совпадать с ключевыми словами,

которых насчитывается несколько десятков. Имя переменной может состоять из латинских букв (строчных и прописных), цифр и символа подчеркивания «_». В языке JavaScript прописные и строчные буквы различаются (в отличие от HTML и CSS), поэтому при назначении имен следует быть особо внимательными.

Объявление переменной означает, что вы даете команду зарезервировать место для хранения данных, при этом указанный идентификатор будет использоваться в качестве имени переменной. Чтобы объявить переменную, надо после ключевого слова VAR указать ее идентификатор. Можно сразу объявить несколько переменных, задав их имена через запятую. Например:

Var myData;
Var x, y, z;

В процессе объявления для переменной можно установить ее начальное значение:

Var myStr = "Здравствуйте";
Var x=100, y= - 50, z=12;

Переменные могут быть глобальными и локальными. Понятие области действия переменных тесно связано с функциями, которые являются отдельными блоками кода. Переменная, объявленная внутри функции, является локальной, и только эта функция имеет доступ к ее значению. Локальные переменные создаются и уничтожаются вместе с соответствующей функцией.

Глобальные переменные должны объявляться вне функций. Такие переменные позволяют функциям обмениваться данными.

Язык JavaScript поддерживает четыре простых типа и объекты. К простым типам относятся:

- целочисленный,
- вещественный,
- строковый,
- булев (логический).

Целочисленный тип. Значение переменной может задаваться в десятичном формате. Например:

var a=125;

В восьмеричном значение начинается с нуля:

var a=011;

На шестнадцатеричный указывает префикс 0x:

var bull=0x9d;

Целочисленные константы, или литералы, представляют собой последовательность цифр со знаками «+» и «-» или без них.

Вещественный тип. Имеет несколько форматов задания вещественных

чисел (с плавающей запятой):

var a= 723.456;

Вещественные константы (литералы) имеют аналогичный синтаксис и указываются в выражениях без имени.

Строковый тип. Значение переменной представляет собой строку символов или цифр, которая берется в кавычки. Как и в HTML, кавычки могут быть одинарные и двойные. Для выделения строки следует использовать кавычки только одного вида (например, правильно: "Остап", неправильно: ‘Остап"). Длинные строковые литералы нельзя разрывать и переносить на следующую строку.

Булев (логический) тип. Переменные этого типа принимают одно из двух значений — *false* или *true* — и используются при проведении логических операций. Специальное значение *null* указывает, что переменная имеет неопределенное значение. Неопределенное значение *undefined* задается в случае, если переменная была объявлена, но к моменту использования ее в вычислениях так и не была инициализирована.

Вопрос 4. Выражения и операторы в JavaScript

Выражения – это набор переменных, констант, операторов. У любого выражения есть свое значение – результат вычисления выражения. Значение может быть числом, строкой или логическим значением.

В JavaScript есть два типа выражений: те, которые присваивают значение переменной, и те, которые просто вычисляют выражение без присваивания его значения переменной. Например, выражение 4+6 складывает два значения, в результате чего получается значение 10. А выражение d=5 присваивает переменной d значение 5. Если манипуляции производятся с одной величиной, то операция называется унарной, а если с двумя, то бинарной.

Операторы присваивания

В JavaScript можно использовать операторы присваивания, описанные в табл. 1.

Табл.1. Операторы присваивания

Оператор	**Операция**	**Пояснение**
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%= (остаток от деления)	x%=y	x=x%y

Арифметические операторы

К арифметическим операторам относят: +, -, *, /, % (модуль), ++ (инкремент (увеличение числа на единицу)), -- (декремент (уменьшение

числа на единицу)).

Логические операторы

К логическим операторам относятся следующие операции:

- ! – унарная операция отрицания (NOT);
- && – бинарная операция И (AND), истинна, когда оба операнда истинны;
- || – бинарная операция ИЛИ (OR), истинна, когда один из операндов равен true.

Операторы сравнения

Операторы сравнения могут использоваться и с числами, и со строками. Операторы и их описание приведено в таблице 2.

Табл.2. Операторы сравнения

==	Равенство – возвращает true, если операнды равны
! =	Неравенство – возвращает true, если операнды не равны
>	Больше – возвращает true, если первый операнд больше второго
>=	Больше или равно – возвращает true, если первый операнд больше или равен второму
<	Меньше – возвращает true, если первый операнд меньше второго
<=	Меньше или равно – возвращает true, если первый операнд меньше или равен второму
===	Идентичность – возвращает true, если операнды равны без преобразования типов
! ==	Неидентичность – возвращает true, если операнды не равны без преобразования типов

Конкатенация строк

Для слияния (конкатенации) строк используется оператор +, например:

str = “Hello” + “world”

Операторы

Редко какая-либо программа или сценарий имеют линейный алгоритм. Обычно в ходе работы часто проверяются различные условия и в зависимости от результата принимаются какие-то решения. В процессе создания сценариев обычно требуется изменить порядок выполнения кода. Для автоматизации работы в коде также используются циклы, которые позволяют намного снизить трудоемкость разработки.

Оператор If … else

Оператор условного перехода if существует практически во всех языках программирования и разветвляет выполнение программы в зависимости от истинности проверяемого условия.

Синтаксис оператора:

if (логическое выражение)
{
    операторы
}

Сначала вычисляется *логическое выражение*, затем, если оно равно *true*, выполняются *операторы*, если же оно равно *false*, то операторы пропускаются, и продолжается выполнение сценария.

Часто удобно использовать полную форму условного оператора if … else. С его помощью можно задать действия, которые необходимо выполнить, если логическое выражение равно false:

if (логическое выражение)
{
    операторы1
}
else
{
    операторы2
}

Оператор switch

Оператор switch позволяет сравнить значение с множеством других. Этот оператор дает возможность предусмотреть операторы по умолчанию, которые выполняются, если не найдено ни одно соответствие:

switch (выражение)
{
    Case значение1: операторы
    Case значение2: операторы
    ……
    Default: операторы
}

Для многократного выполнения кода используют операторы цикла. Кроме того, циклы предоставляют удобные средства для манипулирования массивами.

Цикл FOR

Оператор for служит для создания цикла. Он имеет следующий синтаксис:

for (выражение инициализации; выражения условия; выражения цикла)
{
    операторы
}

Выражение инициализации обычно служит для задания начального значения счетчика цикла. *Выражение условия* позволяет прекратить

выполнять цикл, когда условие перестанет выполняться, то есть примет значение false. *Выражение цикла* обычно осуществляет декремент счетчика цикла. Любое из этих выражений может быть пропущено, но соответствующая точка с запятой должна стоять.

Цикл while

Оператор while сходен с оператором for, но он не производит инициализацию и инкремент счетчика в своем объявлении. Синтаксис этого оператора следующий:

while (выражение условия)
{
операторы
}

Если выражение условия в цикле while сразу ложно, то операторы не выполняются ни разу.

Цикл do … while

Оператор do … while практически идентичен оператору while, в котором проверка условия осуществляется в конце, а do … while гарантирует выполнение операторов, по крайней мере, один раз:

do
{
    операторы
} while (выражения условия)

Вопрос 5. Функции и объекты в JavaScript

Функции

Функции представляют собой последовательность операторов, которые выполняются в определенный момент — момент вызова функции. Для задания функции необходимо описать ее по следующей форме:

function f_name ([arg1, arg2,...])
{
список_используемых_операторов;
[return возвращаемая_величина]
}

Признаком описания служит ключевое слово *function*, после которого следуют имя функции, задаваемое пользователем, и в скобках список аргументов.

Порядок следования параметров в функциях JavaScript менять нельзя, в отличие от атрибутов элементов и свойств в правилах CSS. Сами аргументы могут отсутствовать, но скобки остаются и в этом случае. После скобок точка с запятой отсутствует. Описание размещается в фигурных скобках. Если функция должна вернуть какое-либо значение в основную программу, то последним ее оператором является *return*, после

которого идет возвращаемое значение. При отсутствии *return* возвращаемое значение полагается неопределенным (*undefined*).

Для использования функции осуществляется ее вызов из программы. Признаком функции является наличие скобок.

```
…
f_name(x,y);        // вызов функции
…
```

Вызов функции может происходить только после ее описания. Возвращаемое значение может возвращаться прямо в выражение и использоваться для присвоения или продолжения вычислений.

Например:

```
…
c= f_name(x,y);
a=b+f_name(x,y);
…
```

Переменные, определенные внутри функции, могут быть использованы только внутри функции. Другими словами, область их видимости ограничена телом функции.

Объекты

Объекты — наиболее сложный, однако и наиболее мощный тип данных современного программирования. Языки, поддерживающие объекты, называют объектно-ориентированными. Большинство современных систем программирования — Delphi, C++, Java, C# - являются объектно-ориентированными. Объекты, конструируемые в JavaScript, не обладают всеми свойствами, присущими объектам, например в C++, но позволяют строить основные конструкции приложений.

Объект представляет собой совокупность тесно взаимосвязанных и взаимодействующих данных и функций. Данные в этом случае называются свойствами объекта, а функции — методами объекта. Свойства могут быть заданы не только простыми типами данных (целыми, вещественными, строками), но и другими новыми объектами.

Наличие объектов позволяет отделять основное содержание программы от огромного количества второстепенных деталей. Вся работа с деталями при этом «прячется» в объекты. Если провести аналогию, то ситуация сходна со стилями. Простые программы можно написать, не привлекая объекты, так же, как и одну страницу можно оформить без использования стилей. Но написание сложной программы в обозримые сроки возможно, только если научиться работать с более крупными блоками данных, такими как объекты.

Имя объекта подчиняется общим правилам формирования имен. А вот вызов, или доступ, к свойствам осуществляется по собственным

правилам, иногда называемым точечной нотацией. Они заключаются в том, что слева направо последовательно указываются объект и его свойство. Разделителем между ними служит точка. Например, если имя объекта team, а его свойство - driver, то вызвать это свойство можно следующим образом:

team.driver

Вызов методов-функций осуществляется аналогично, с учетом особенностей вызова функций. Например, вызов метода qo(parameter), принадлежащего объекту team, имеет вид:

team.go("Черноморе")*

где вызвана функция с параметром-строкой "Черноморск".

В том случае, когда свойство является объектом и надо обратиться к свойству или методу этого вложенного объекта, запись продолжается аналогичным образом.

Тема 11. Особенности работы со встроенными объектами в Java Script.

Объекты JavaScript объединяют в себе переменные, именуемые свойствами, и функции, управляющие этими свойствами, именуемые методами. Кроме того они могут реагировать на события. Пользователь может создавать свои объекты, но JavaScript содержит ряд встроенных (базовых объектов):

Объект String

В языке JavaScript строковый тип данных отсутствует, а работа со строками организуется при помощи объекта String, который предоставляет средства для форматирования и выделения части строк. Для этой цели он содержит обширный перечень методов (табл. 1-2).

Табл. 1. Свойства объекта String

Элемент	**Описание**
Constructor	Имя функции, создающей прототип объекта
Length	Количество символов в строке
Prototype	Позволяет создавать дополнительные свойства

Табл. 2. Методы объекта String

Элемент	**Описание**
Anchor()	Создает «якорь» для гиперссылки
Big()	Помещает строку между парными тегами <big> </big>
Bold()	Помещает строку между парными тегами <bold></bold>
charAt()	Возвращает символ по указанному положению (индексу) в строке
charCodeAt()	Возвращает число, являющееся значением указанного символа в кодировке Unicode

Concat()	Объединяет две строки в одну и возвращает новую строку
Fixed()	Помещает строку между парными тегами <tt> </tt>
fontColor()	Помещает строку между парными тегами
fontSize()	Помещает строку между парными тегами
fromCharCode()	Возвращает строку, созданную по заданной последовательности числовых значений кодировки Unicode
indexOf()	Возвращает индекс первого вхождения шаблона в строку
Italics()	Помещает строку между парными тегами
lastIndexOf()	Возвращает индекс последнего вхождения шаблона в строку
Link()	Создает гиперссылку из строки
Match()	Используется для сравнения регулярного выражения со строкой
Replace()	Применяется для поиска и замены подстроки новой последовательностью символов
Search()	Используется для поиска подстроки по заданному шаблону
Slice()	Возвращает подстроку указанной строки
Small()	Помещает строку между парными тегами
Split()	Разбивает строку на сегменты строк
Strike()	Помещает строку между парными тегами <strike> </strike>
Sub()	Помещает строку между парными тегами
Substr()	Возвращает подстроку строки заданной длины, начиная с заданного индекса
Substring()	Возвращает подстроку строки между начальным и конечными индексами
Sup()	Помещает строку между парными тегами
toLowerCase()	Возвращает строку, целиком переведенную в нижний регистр
toSource()	Возвращает строку, представляющую исходный код строкового объекта
toString()	Возвращает символы, переданные как данные для

	строки
toUpperCase()	Возвращает строку, целиком переведенную в верхний регистр
valueOf()	Аналогично toString()

Рассмотрим пример использования методов объекта String:

MyText = "Я люблю Html";

MyText = MyText.subString(0/8) + "JavaScript";

В результате строка MyText примет значение «Я люблю JavaScript».

Объект Number

Объект Number обычно применяется для доступа к постоянным значениям, например к отрицательной бесконечности. Свойства и методы этого объекта указаны в табл. 3 – 4.

Табл. 3. Свойства объекта Number

Элемент	**Описание**
Constructor	Ссылка на функцию, создающую прототип объекта
MAX_VALUE	Наибольшее число в машинном представлении
MIN_VALUE	Наименьшее число в машинном представлении
NaN	Специальное значение, не равное ни одному числу
NEGATIVE_INFINITY	Специальное значение отрицательной бесконечности
POSITIVE__INFINITY	Специальное значение положительной бесконечности
prototype	Позволяет создание дополнительных свойств

Табл. 4. Методы объекта Number

Элемент	**Описание**
toSource()	Возвращает строку, создавшую объект Number
toString()	Преобразует число в строку
valueOf()	Возвращает числовое значение объекта Number

Объект Boolean

Объект Boolean обычно используется для преобразования объекта в логическое значение. Этот объект имеет ограниченнойе количество методов и свойств (табл. 5).

Табл. 5. Методы и свойства объекта Boolean

Тип	Элемент	Описание
Свойство	prototype	Позволяет создание дополнительных свойств

Метод	toString()	Преобразует логическое значение в строку true или false

Объект Array

С массивами мы уже знакомы, однако массивы — это объекты, которые имеют свойства и методы, облегчающие работу программисту (табл. 6-7).

Табл. 6. Свойства объекта Array

Элемент	**Описание**
constructor	Ссылка на функцию, создающую прототип объекта
index	Для массива, созданного путем сравнения с регулярным выражением, это свойство возвращает индексную позицию, в которой произошло совпадение
input	Для массива, созданного путем сравнения с регулярным выражением, это свойство возвращает первоначальную строку
length	Количество элементов в массиве
prototype	Позволяет создание дополнительных свойств

Табл. 7. Методы объекта Array

Элемент	**Описание**
concat()	Объединяет два массива и возвращает новый массив
join()	Объединяет все элементы массива в строку
pop()	Удаляет последний элемент из массива и возвращает этот элемент
push()	Добавляет один или несколько элементов в конец массива и возвращает новую длину массива
reverse()	Инвертирует расположение элементов в массиве
shift()	Удаляет первый элемент из массива и возвращает этот элемент
slice()	Удаляет несколько элементов из массива и возвращает новый массив
splice()	Добавляет и/или удаляет элементы из массива
sort()	Сортирует элементы массива
toSource()	Преобразует элементы в строки, заключенные в квадратные скобки
toString()	Возвращает строку, содержащую в себе значения всех элементов
unshift()	Добавляет один или несколько элементов в начало массива и возвращает новую длину массива

valueOf()	Возвращает массив элементов, разделенных запятыми

Объект Function

С помощью объекта Function можно манипулировать функцией как объектом. Этот объект содержит специфичных методов и свойств (табл. 8-9).

Табл. 8. Свойства объекта Function

Элемент	**Описание**
Arguments	Массив фактических аргументов функции
Arguments.callee	Ссылка на тело выполняемой функции
Arguments.caller	Ссылка на функцию, вызвавшую данную функцию
Arguments.length	Количество элементов в массиве arguments
arity	Количество формальных аргументов функции
constructor	Ссылка на функцию, создающую прототип объекта
length	Аналогично arity
prototype	Позволяет создание дополнительных свойств

Табл. 9. Методы объекта Function

Элемент	**Описание**
apply()	Позволяет выполнить операцию наследования метода
call()	Позволяет выполнить операцию наследования метода
toSource()	Возвращает строку, содержащую тело функции
toString()	Аналогично toSource()
valueOf()	Возвращает указатель на функцию

Объект Date

Объект Date предоставляет значительное количество свойств и методов, связанных с датами и временем (табл. 10-11). В качестве базовой даты JavaScript принято 1 января 1970 года.

Табл. 10. Свойства объекта Date

Элемент	**Описание**
constructor	Ссылка на функцию, создающую прототип объекта
prototype	Позволяет создание дополнительных свойств

Табл. 11. Методы объекта Date

Элемент	**Описание**
getDate()	Возвращает день месяца по указанной дате от 1 до 31 (по местному времени)
getDay()	Возвращает день недели по указанной дате от 0 до 6 (по местному времени)
getFullYear()	Возвращает год в рамках локального времени, представленный четырьмя цифрами
getHours()	Возвращает час по указанной дате от 0 до 23 (по местному времени)
getMilliseconds()	Возвращает миллисекунды по указанной дате (по

	местному времени)
getMinutes()	Возвращает минуты по указанной дате от 0 до 59 (по местному времени)
getMonth()	Возвращает месяц по указанной дате от 0 до 11 (по местному времени)
getSeconds()	Возвращает секунды по указанной дате от 0 до 59 (по местному времени)
getTime()	Возвращает число миллисекунд, истекших с момента 00:00:00 1 января 1970 года (по местному времени)
getYear()	Возвращает год по указанной дате (по местному времени)
parse()	Преобразует переданные строковые данные в миллисекунды, прошедшие с 0 часов 0 минут 0 секунд 1 января 1970 года до указанной даты (по местному времени)
setDate()	Устанавливает день месяца для указанной даты (по местному времени)
setFullYear()	Устанавливает год для указанной даты (по местному времени)
setHours()	Устанавливает час для указанной даты (по местному времени)
setMilliseconds()	Устанавливает миллисекунды для указанной даты (по местному времени)
setMinutes()	Устанавливает минуты для указанной даты (по местному времени)
setMonth()	Устанавливает месяц для указанной даты (по местному времени)
setSeconds()	Устанавливает секунды для указанной даты (по местному времени)
setTime()	Устанавливает числовое значение объекту Date (по местному времени)

Например, чтобы узнать текущий день недели, можно воспользоваться следующим кодом:

```
MyDate=new Date();
alert (MyDate.getDate());
```

Этот код отобразит число от 0 до 6, соответствующее дню недели. Например, 0 соответствует воскресенью.

Объект Math

Для математических вычислений JavaScript поддерживает объект Math, который содержит ряд математических констант и функций (табл.

12-13).

Табл. 12. Свойства объекта Math

Элемент	**Описание**
E	Константа Эйлера (число e)
LN10	Натуральный логарифм десяти
LN2	Натуральный логарифм двух
LOG10E	Логарифм числа e по основанию 10
LOG2E	Логарифм числа e по основанию 2
PI	Отношение длины окружности к ее диаметру (число π)
SQRT1_2	Корень квадратный из 1/2
SQRT2	Корень квадратный из 2

Табл. 13. Методы объекта Math

Элемент	**Описание**
abs()	Абсолютная величина заданного значения
acos()	Арккосинус заданного значения в радианах
asin()	Арксинус заданного значения в радианах
atan()	Арктангенс заданного значения в радианах
atan2()	Арктангенс показателя заданного значения в радианах (арктангенс частного от деления аргументов)
ceil()	Возвращает наименьшее целое число, большее или равное данному числу
cos()	Косинус числа
exp()	Возвращает число — константу Эйлера, возведенную в заданную степень
floor()	Возвращает наибольшее целое число, меньшее или равное данному числу
log()	Натуральный логарифм числа
max()	Максимальное из двух чисел
min()	Минимальное из двух чисел
pow()	Возведение числа, заданного в первом аргументе, в степень, заданную вторым аргументом
random()	Возвращает число, равномерно распределенное в интервале от 0 до 1
round()	Округляет аргумент до ближайшего целого
sin()	Возвращает синус числа
sqrt()	Возвращает корень квадратный из числа
tan()	Возвращает тангенс числа
toSourse()	Копия объекта

toString()	Строковое представление объекта

Объект Object

Все объекты в JavaScript — и встроенные, и пользовательские — наследуются от объекта Object, то есть его методы и свойства (табл. 14-15) присутствуют в каждом объекте, но, как правило, их переопределяют.

Табл. 14. Свойства объекта Object

Элемент	**Описание**
constructor	Ссылка на функцию, создающую прототип объекта
prototype	Позволяет создание дополнительных свойств

Табл. 15. Методы объекта Object

Элемент	**Описание**
toLocaleString()	Возвращает строковое представление объекта с учетом интернациональных установок
toSource()	Возвращает строку, являющуюся строкой, создавшей объект
toString()	Возвращает строку, содержащую значение объекта
toValueOf()	Возвращает примитивное значение заданного объекта

Тема 12. События, поддерживаемые JavaScript. Объект event.

1. *Интерактивность на веб-страницах. События в языке JavaScript*
2. *Объект event*

Вопрос 1. Интерактивность на веб-страницах. События в языке JavaScript

Интерактивное взаимодействие со страницей, как и все современное программирование, основано, прежде всего, на обработке событий. События в программировании — это реальные физические действия, такие как перемещение мыши, нажатие клавиши, или виртуальные, которые происходят, например, в процессе выполнения программы — открытие окна, его активация и т.п. Объединяет их то, что после наступления события между программой и операционной системой передаются сообщения. Получив такое сообщение, программа, которой оно предназначалось, вызывает некоторую функцию-обработчик события.

Сам HTML не предусматривает обработку событий, если не считать переходы по ссылкам и отправку форм. Но он предусматривает механизм связи события с обработчиком, реализованном на каком-либо языке. Технология обработки событий в HTML не отличается большой гибкостью и возможностями по сравнению с другими развитыми языками, но зато организация обработки событий достаточно проста.

Практически каждый элемент имеет набор атрибутов (мы будем называть их атрибутами событий), начинающихся префиксом *on*. Перечислим те, которые встречаются наиболее часто:

- onclick,
- ondblclick,
- onmousedown,
- onmouseup,
- onmouseover,
- onmouseout,
- onmousemove,
- onkeypress,
- onkeydown,
- onkeyup,
- onfocus,
- onblur.

Название атрибута формируется следующим образом. Префикс *on* ставится перед названием события, например, *click* означает щелчок, *dblclick* — двойной щелчок и т.п. Значением атрибута, которое, как обычно, надо взять в кавычки, служит код JavaScript, если он небольшого размера, но чаще этот код является вызовом функции.

Атрибуты событий, перечисленные выше, позволяют связать

элементы HTML с обработчиками, которые выполняются при помощи скриптов. Весь сложный процесс формирования и прохождения события от щелчка мышью, прохождения сигнала через операционную систему до поступления и прохождения внутри браузера остается скрытым от разработчика страницы. Ему остается лишь описать реакцию на сигнал.

Рассмотрим пример простейшей организации обработки события.

```
<html>
<head>
<title>События и HTML</title>
</head>
<body>
<h1 onmouseover="alert('Поможем детям. Будем помнить, что дети - цветы жизни. Я приглашаю вас сейчас же сделать взносы и помочь детям.')">
Союз «Милосердие»
</body>
</html>
```

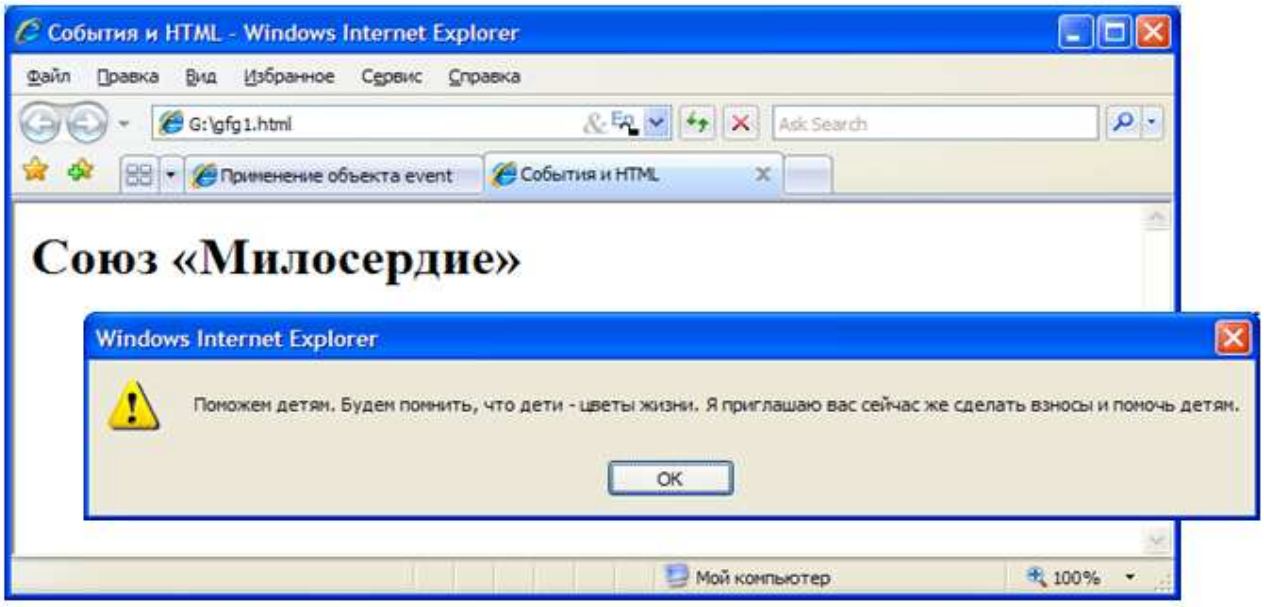


Рис. 1. Подключение событий

Сначала, по мере чтения, выполняется код HTML. Без каких-либо особенностей открывается окно браузера с заголовком. Но элемент <h1> становится активным, как ссылка. При попадании на него указателя мыши возникает событие. В данном примере процедура обработки события вызывается встроенными средствами HTML, она указана как значение атрибута *onmouseover*. Управление передается обработчику события, то есть вызывается процедура *alert ('строка')*. Обработчик, как и положено, выполнен средствами JavaScript. Эта простая встроенная функция вызывает на экран диалоговое окно предупреждения. В функцию должна

быть передана строка (берется в одинарные кавычки), которая и появится в окне (рис. 1).

Перечислим подробнее назначение атрибутов событий в табл.1.

Табл. 1. События в JavaScript

События	**Назначение**
onclick	Назначает обработчик события click, то есть функцию, которая должна вызываться после щелчка мышью по элементу, в котором расположен параметр.
ondblclick	Назначает обработчик двойного щелчка мышью.
onmousedown	Назначает обработчик нажатия левой кнопки мыши.
onmouseup	Назначает обработчик отпускания левой кнопки мыши.
onmouseover	Назначает обработчик наведения указателя мыши на элемент, например, на блок текста или изображение.
onmousemove	Назначает обработчик перемещения мыши.
onmouseout	Назначает обработчик отведения указателя мыши от элемента.
onkeypress	Назначает обработчик события нажатие и отпускание клавиши.
onkeydown	Назначает обработчик события нажатие и удержание клавиши.
onkeyup	Назначает обработчик отпускания клавиши.
onfocus	Данному элементу передан фокус ввода, то есть он готов среагировать на нажатие клавиши Enter. Часто фокус ввода передаётся клавишей-табулятором Tab.
onblur	Обратное действие — элемент теряет фокус ввода, и последний передается другому элементу.
onchange	Параметр обслуживает несколько элементов, таких как <input>, <select>, <textarea>, и назначает обработчик события изменение пользователем исходного текста формы.

Вопрос 2. Объект event

В ряде событий необходимо получить большую информацию, чем просто свидетельство его возникновения. Если речь идет о мыши, то могут понадобиться координаты курсора, если о клавиатуре, то сведения о том, какая клавиша нажата и т.д. Такие дополнительные сведения о событии можно получить, имея доступ к специальному элементу *event*. Возникновение события сопровождается созданием нового объекта event,

который является свойством объекта window. Это свойство-объект позволяет отменить всплывание события. Кроме того, объект event позволяет установить, какой элемент генерировал событие.

К основным свойствам объекта event относятся (см. табл. 2):

Табл. 2. Свойства объекта EVENT

Свойство	**Характеристика**
type	содержит тип события. Имя события связано с именем обработчика, только не содержит префикса on.
srcElement	содержит информацию об элементе, который сгенерировал событие.
cancelBubble	булева переменная, определяет всплывание события, или отменяет его. При всплывании события имеет значение false, которое задается по умолчанию.
altKey	свойство, содержащее информацию о том, была ли нажата клавиша Alt. Значение этой булевой переменной, равное true, указывает, что клавиша была нажата.
ctrlKey	свойство, содержащее информацию о том, была ли нажата клавиша Ctrl.
shiftKey	свойство, содержащее информацию о том, была ли нажата клавиша Shift.
clientX clientY	содержит, соответственно, x и y координаты курсора мыши относительно клиентской части окна браузера.
offsetX offsetY	содержит, соответственно, x и y координаты курсора мыши относительно элемента контейнера, в котором расположен элемент-источник события.
screenX screenY	содержит, соответственно, x и y координаты курсора мыши относительно экрана.

Рассмотрим пример использования объекта event (рис. 2):

```
<html>
<head><title>Применение объект even</title></head>
<body>
<h1>Элемент заголовками </h1>
<div id=my_element onclick="my_fun2()">Это текстовой параграф
</body>
<script type="text/javascript">
<!--
function my_fun2()
{alert('x='+window.event.offsetX+'\n'+'y='+window.event.offsetY);}
-->
</script>
```

<*html*>

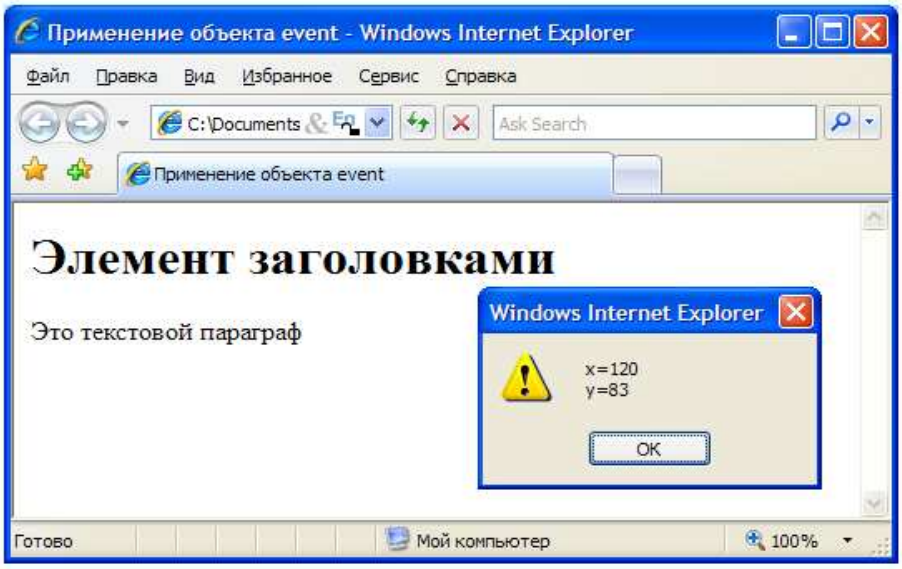


Рис. 2. Применение объекта EVENT

Тема 13. Особенности языка PHP. Синтаксис, грамматика и функции PHP.

1. *Переменные и константы в PHP*
2. *Выражения и операторы в PHP*
3. *Условная и циклическая обработка в PHP*
4. *Полезные PHP-функции*

PHP – это язык серверных скриптов (server scripting language), встраиваемый в HTML, который интерпретируется и выполняется на сервере.

PHP называется языком серверных скриптов в отличие от JavaScript, который является языком клиентских скриптов. Это значит, что PHP-скрипт выполняется на сервере, а клиенту передается результат его работы, тогда как в JavaScript-код полностью передается на клиентскую машину и только там выполняется браузером.

Программы PHP хранятся в стандартных текстовых файлах, которые можно создавать в любом редакторе. Обычно надо сохранять файлы, давая им расширение .php, чтобы веб-сервер мог их выполнить, хотя с соответствующими настройками можно обрабатывать любые расширения.

PHP-сценарии можно внедрять в HTML-код за счет определения кода сценариев между начальным и конечным дескрипторами PHP (<?...?> или <?php...?>) или начальным и конечным HTML-десКрИПТОраМИ Сценария (<script language="php">...</script>). Во-вторых, в PHP-сценарии можно внедрять HTML-дескрипторы. При этом они заключаются в кавычки и отмечаются PHP-оператором echo. Оператор echo "выводит" HTML-дескрипторы в HTML-код Web-страницы, которую PHP-процессор передает Web-серверу. Например, следующая строка вставляет в HTML-код Web-страницы дескриптор горизонтальной линейки (<hr>):

<?php echo "<hr>" ; ?>

Вопрос 1. Переменные и константы в PHP

Как и в большинстве языков программирования, в PHP для хранения программ используются переменные. Переменные можно рассматривать как контейнеры для хранения данных программы.

Переменную можно объявить в любом месте сценария, но до ее первого использования. Вообще, по большому счету, переменные в PHP не нуждаются в объявлении, но для того чтобы было проще необходимо объявлять переменную следующим образом:

$имя = значение;

В имени переменной можно использовать только латинские буквы, цифры и знак подчеркивания. Не допускается использование пробелов и пробельных символов. Имя переменной не может начинаться с цифры.

Чтобы PHP понимал, что перед ним переменная, нужно перед ее именем всегда указывать знак доллара.

При инициализации (присвоении значения) переменной не нужно указывать тип переменной. PHP автоматически изменит тип переменной в зависимости от указанного значения. С одной стороны, такое поведение PHP значительно упрощает жизнь, поскольку не нужно задумываться о типах переменных. С другой стороны, вся работа по контролю типов возложена на вас.

Переменной можно присвоить как значение, так и результат функции или выражения:

$i = 7*(5-3)*$j;

Как и в других языках программирования, можно определить константы (постоянные значения) с помощью директивы define:

define (“const” , “значение”)

Обращение к константам производится без знака $.

Вопрос 2. Выражения и операторы в PHP

При формировании выражений на языке PHP используются операнды (аргументы операций), знаки операций, вызовы функций и скобки для задания приоритета операций, например:

$i = 7*(5-3)*$j;

В арифметических выражениях используются следующие знаки операций:

- + - сложение;
- - - вычитание;
- * - умножение;
- / - деление;
- A%B — остаток от деления A на B;
- $a + + - инкремент (увеличение на 1);
- $b+ + - декремент (уменьшение на 1).

Результатом арифметической операции является число. Наибольшим приоритетом обладают операции инкремента и декремента, потом следуют операции умножения, деления и остатка. После них — операции сложения и вычитания.

К логическим относятся следующие операции:

- ! - унарная операция отрицания (NOT);
- && - бинарная операция И (AND), истинна, когда оба операнда истинны;
- || - бинарная операция ИЛИ (OR), истинна, когда один из операндов равен true.

Результатом логической операции является одно из значений — true или false. Кроме логических операций, значения true и false возвращают операторы сравнения:

- > - больше;

- < - меньше;
- = = - равно;
- = = = - равно (используются для корректного сравнения строк);
- != - не равно;
- > = - больше или равно;
- < = - меньше или равно.

Над строками можно выполнить всего две операции: конкатенация (слияние строк) и операцию [n] — обращение к символу n строки:

```
$s = “Hello”;
// конкатенация обозначается точкой (.)
$s = $s . “World!”;      // $s = “Hello World!”
echo $s [1];             // выведет 'e'
```

Нумерация символов строки начинается с 0.

Вопрос 3. Условная и циклическая обработка в PHP

PHP-сценарий представляет собой последовательность инструкций, которые должен выполнить PHP-процессор для выполнения определенной задачи. Все простые сценарии, рассмотренные до сих пор в этой главе, выполнялись процессором от первого оператора до последнего в строгом порядке. С усложнением сценариев возникает необходимость в выполнении одного набора операторов, если определенное условие соблюдается и, возможно, другого набора в противном случае. Другими словами, возникает необходимость, чтобы процессор принимал решения и действовал в соответствии с принятыми решениями. При написании сценариев для принятия процессором решений, реализуется условная обработка. Другими словами, на основании одного или нескольких условий процессор будет выполнять определенные операторы.

Для принятия решений сценарии должны выполнять проверку определенного рода. Например, один сценарий может проверять правильность введенного посетителем пароля, а второй — верное ли количество цифр ввел посетитель в номере телефона. Для выполнения таких проверок сценарий будет использовать приведенные выше операции сравнения. Операции сравнения позволяют определить, каким образом одно значение соотносится с другим. Другими словами, с помощью операций сравнения PHP-процессор проверяет равно, больше или меньше одно значение другого. При выполнении процессором операций сравнения для сравнения двух значений результат выражается значениями true и false, Т.е. значения соответствуют (true) или не соответствуют (false).

Тесно связана с процессом принятия решений в сценарии и возможность повторения выполнения одного или нескольких операторов определенное количество раз или до возникновения известного состояния.

Оператор *while* позволяет организовать цикл, выполняемый процессором до тех пор, пока определенное условие соблюдается.

Оператор for дает возможность организовать цикл, в котором PHP-процессор выполняет набор операторов определенное число раз.

Использование оператора if для проверки условия

Оператор if позволяет использовать операции сравнения PHP для указания PHP-процессору на необходимость проверки условия, а затем выполнения операторов, основываясь на результатах проверки. Если проверка дает true, процессор будет выполнять операторы, находящиеся после if. В противном случае, если проверка дает false, процессор пропускает операторы после if. Формат оператора if выглядит следующим образом:

if (условие)

оператор;

Обратите внимание, что в конце строки if (условие), с которой начинается оператор if, точка с запятой (;) не ставится, поскольку оператор продолжается на следующей строке.

При использовании оператора if для условной обработки в некоторых случаях при соблюдении условия выполнятся только один оператор; в других же ситуациях выполняемых в случае соблюдения условия операторов может быть несколько. Когда сценарий указывает процессору *ВЫПОЛНИТЕ ТОЛЬКО ОДИН* оператор, следующий за if, используется простой оператор:

```
if ($state == "ca")
echo "Был введен код штата Калифорния.";
```

Чтобы процессор при соблюдении условия выполнял несколько операторов, их необходимо сгруппировать с помощью фигурных скобок ({}). Операторы, заключенные в скобки, называются составным оператором, пример которого показан ниже:

```
if ($username != "Konrad")
{
header("Location: http://NVBizNet.com/htmlwdtt/InvalidUser.htm");
exit
```

Запоминание терминов "простой оператор" и "составной оператор" не является обязательным, тем не менее, важно знать, что связанная группа операторов должна заключаться в фигурные скобки ({ }) .

В двух предыдущих примерах процессор использует операторы if для определения необходимости выполнения одного или нескольких операторов. В первом примере, если условие соблюдается, процессор помещает на Web-страницу текст. Аналогично, во втором примере, если условие соблюдается, т.е. значение переменной $usermame не равно "Konrad", процессор перенаправляет Web-браузер на новую Web-страницу. В обоих примерах, если условие не соблюдается, процессор пропускает (т.е. не выполняет) операторы, содержащиеся в блоке if. Тем не менее, в большинстве случаев требуется, чтобы при соблюдении условия выполнялся один блок операторов, а при несоблюдении — другой блок. Операторы, которые должны выполнятся в случае, когда условие не соблюдается, указываются в сценарии в блоке else. Формат оператора else выглядит следующим образом:

```
if (условие)
оператор;
else
оператор;
```

В следующем сценарии оператор if-else используется для проверки значения переменной $username. Если значение равно "Konrad", сценарий

перенаправляет посетителя на страницу weicome.htm, в противном случае будет открыта страница InvalidUser.htm:

```
if ($username != "Konrad")
{
header ("Location: Weicome.htm") ;
exit
}
else
{
header("Location: http://NVBizNet.com/htmlwdtt/InvalidUser.htm");
exit
}
```

Помимо применения оператора if-else для выполнения одного набора операторов при соблюдении условия и другого набора при несоблюдении условия в некоторых случаях возникает необходимость проверки нескольких связанных условий или же многократной проверки одного и того же условия. Например, предположим, что сценарий должен определять междугородний телефонный код посетителя путем сравнения значения переменной $city с несколькими городами, телефонные коды которых сценарию известны. Следующий оператор использует последовательные операторы if-else для иллюстрации упрощенного примера выполняемой процессором обработки:

```
if ($city == "New York")
$areaCode = 212;
elseif ($city — "Houston")
$areaCode = 2 81;
elseif ($city == "Phoenix")
$areaCode = 602;
elseif ($city == "Seattle")
$areaCode = 206;
else
$areaCode = "**UNKNOWN**";
```

При выполнении PHP-процессором приведенного выше оператора if сначала сценарий проверяет, является ли город посетителя Нью-Йорком (New York). В случае положительного результата сравнения переменной $areaCode присваивается значение 212. Если посетитель живет не в Нью-Йорке, процессор выполняет следующий оператор if-else и определяет, является ли местом жительства посетителя Хьюстон (Houston). Подобная обработка будет проводиться до тех пор, пока PHP-процессор не определит город или дойдет до последнего оператора else. Если соответствие для города посетителя найдено не будет, переменной $areacode будет присвоено значение **"**UNKNOWN**"** (то есть,

"НЕИЗВЕСТНЫЙ").

Приведенная в предыдущем примере структура оператора if применяется для выполнения оператора (или группы операторов) при проверке на неравенство (значение переменной меньше, больше или не равно другому значению) или при необходимости проверки значений нескольких связанных между собой переменных. Для проверки значения одной переменной на равенство числу или строке PHP предоставляет более подходящий для этой цели оператор switch. Например, следующий оператор switch более очевиден и элегантен, нежели операторы if и elseif из предыдущего примера:

```
switch ($city)
{
case "New York" :
$areaCode = 212;
break;
case "Houston":
$areaCode = 2 81;
break;
case "Phoenix":
$areaCode = 602 ;
break;
case "Seattle":
$areaCode = 206;
break;
default:
$areaCode = "**UNKNOWN**";
}
```

При выполнении оператора switch процессор сравнивает значение выражения, следующего после ключевого слова switch с каждым из значений, находящихся после ключевых слов case. Если ни одно из значений, указанных после case, не совпадает со значением выражения после switch, процессор будет выполнять операторы, соответствующие оператору default.

Оператор break в предыдущем примере важен по той причине, что он указывает PHP-процессору прекратить выполнение операторов case и перейти к выполнению оператора, находящегося непосредственно после switch. Например, если бы в операторе case "Houston" был пропущен оператор break, переменная $areaCode сначала получила бы значение 281, а затем сравнение бы продолжалось, и ей было бы присвоено значение 602 . Тем не менее, в некоторых случаях оператор break может опускаться специально. Предположим, например, что одни и те же операторы должны выполняться для нескольких операторов case, как в случае с

первыми тремя операторами case в следующем примере:

```
switch ($city)
{
case "Henderson"
case "Las Vegas"
case "Pahrump"
$areaCode = 7 0 2;
break;
case "New York" :
$areaCode = 212 ;
break;
case "Houston":
$areaCode = 2 81;
break;
case "Phoenix":
$areaCode = 6 02;
break;
case "Seattle":
$areaCode = 206;
break;
default:
$areaCode = "**UNKNOWN**";
}
```

Поскольку Хендерсон (Henderson), Лас-Вегас (Las Vegas) и Пахрамп (Pahrump) имеют один и тот же телефонный код (702), необходимо, чтобы PHP-процессор выполнил оператор присваивания - $areaCode = 702 для каждого из этих трех городов.

Использование оператора for для повторения операторов заданное количество раз

Одной из наиболее часто встречаемых операций является повторение одного или нескольких операторов определенное количество раз. В этом случае применяется оператор for. Оператор for в языке PHP содержит три выражения и простой или составной оператор. Этот оператор обладает следующим синтаксисом:

for (выражение1, выражение2, выражение3) оператор;

PHP-процессор вычисляет значение первого выражения (выражение1) один раз перед началом выполнения цикла. В начале каждой итерации вычисляется значение второго выражения (выражение2). Если значение второго выражения равно true, PHP-процессор выполняет оператор или группу операторов, находящихся между фигурными скобками, как в следующем примере:

```
for (выражение!, выражение2, выражение3)
{
оператор;
оператор;
оператор;
}
```

Если же значение второго выражения будет равно false, PHP-процессор завершит выполнение цикла for. В конце каждой итерации вычисляется значение выражения выражение3.

В цикле for можно использовать любые выражения языка **PHP,** причем любое из трех выражений может быть пустым. Тем не менее, если пустым является второе выражение (выражение2 в приведенном выше синтаксисе), PHP-процессор будет выполнять цикл бесконечно. Таким образом, следующий цикл for с пустыми вторым и третьим выражениями будет выполняться до тех пор, пока условный оператор break в теле цикла (if $count = 10) не приведет к тому, что выполнение сценария продолжится с первого оператора, находящегося после цикла for:

```
for ($count= 1; ; )
{
echo "Count = $count<br>";
if ($count==10) break;
++$count;
}
```

Рассмотрим еще один пример. Следующий оператор for (с тремя непустыми выражениями) выводит на экран первые десять значений, хранящихся в массиве $stateNames:

```
for ($statelndex= 0; $statelndex <= 9; ++$statelndex)
```

*echo "\$stateNames[$statelndex] = $stateNames[$statelndex]
";*

Третье выражение увеличивает значение переменной $stateindex после выполнения очередной итерации цикла for. Пока значение второго выражения ($stateindex <= 9) равно true, PHP-процессор будет выполнять блок операторов цикла for. В приведенном примере это единственный оператор echo.

Использование цикла while для повторения выполнения операторов в случае соблюдения условия

Для ситуаций, в которых PHP-процессор должен выполнять операторы до тех пор, пока некоторое условие соблюдается, но не обязательно определенное количество раз, можно использовать цикл while. Формат цикла while выглядит следующим образом:

```
while ($условие)
{
оператор;
оператор;
}
```

При выполнении цикла while процессор сначала проверяет условие (содержащееся в скобках после ключевого слова while). Если условие соблюдается, будут выполняться операторы в блоке цикла while (т.е. операторы, находящиеся в фигурных скобках ({}) после ключевого слова while). После выполнения последнего оператора цикла процессор снова проверяет условие. Если оно соблюдается, блок операторов выполняется снова, после чего снова проверяется условие и т.д.

Когда, в конце концов, условие даст false, процессор продолжит выполнение сценария с оператора, который находится непосредственно после цикла while.

Цикл do-while аналогичен циклу while. Он также заставляет процессор выполнять операторы при соблюдении условия, но вместо проверки условия перед выполнением операторов (как в случае с циклом while), цикл do-while проверяет условие после выполнения операторов. Формат цикла do-while выглядит следующим образом:

```
do
{
оператор;
оператор;
}
while ($условие);
```

Поскольку процессор проверяет условие после выполнения операторов тела цикла, они будут выполнены по крайней мере один раз, даже если при первой проверке условие соблюдаться не будет.

Вопрос 4. Полезные PHP-функции

Так как в PHP строка — самый главный тип данных. Длина строки практически ничем не ограничена (разве что размер памяти, занимаемым сценарием и физическим остатком ресурсов компьютера), поэтому к строке можно привести любой объект. Рассмотрим основные строковые функции.

Определение длины строки

Функция *strlen ()* используется очень часто, поскольку значение длины строки требуется для многих ее преобразований. Использовать ее очень просто — передаем ей строку, а функция вернет количество символов, например:

```
$str = “привет!”;
echo strlen ($str);
```

Поиск и замена в строке

Для поиска в строке используется функция *strpos ()*. Ей нужно передать три параметра — строку, в которой нужно произвести поиск, искомую строку и позицию начальной строки, с которой нужно начинать поиск. Если поиск нужно начать с начала строки, то позицию можно не указывать. Функция возвращает позицию, с которой начинается вхождение искомой строки в начальную строку. Если искомая строка не найдена, функция возвращает false.

Рассмотрим пример использования функции strpos ():

```
$position = strpos (“Привет, мир”, “мир”);
if ($position = = = false)
    echo “Искомая подстрока не найдена”;
else
    echo “Искомая строка начинается с позиции $position”;
```

Теперь рассмотрим функцию замены текста str_replace():

```
str_replace(mixed найти, mixed  заменить, mixed где [ , int &к-во_замен] )
```

Данная функция заменяет строку «найти» строкой «заменить» в строке «где». Функция производит количество замен, заданное переменной «к-во_замен» (этот параметр необязательный). Функция работает с копией строки «где» и возвращает результат — измененную копию строки «где» (но сама строка «где» не изменяется). Если была задана переменная, содержащая количество замен, то она будет содержать количество реально произведенных замен. Например, вы могли задать три замены, а функция произвела только две замены.

Сравнение строк

Функции *strcmp ()* и *strcasecmp ()* используются для сравнения строк. Первая функция осуществляет сравнение строк с учетом регистра символов, а вторая — без учета. Функции возвращают следующие значения:

- 0, если лексикографически строки равны

- -1, если первая строка лексикографически меньше, чем вторая
- 1, если первая строка лексикографически больше, чем вторая

Что такое лексикографическое сравнение строк? Например, у вас есть два слова «Арбуз» и «Борис», и вам нужно расположить их по алфавиту в порядке возрастания. Первое слово начинается на букву «А», которая в алфавите следует раньше буквы «Б», на которую начинается второе слово. Поэтому первая строка будет лексикографически меньше второй строки.

Функции удаления пробельных символов

Некоторые пользователи указывают много пробельных символов (пробелы, символы табуляции) в начале и в конце строки. Для удаления таких символов используется функция trim (). функция не изменяет исходную строку, работает с ее копией, поэтому если вам нужно изменить саму строку, то нужно использовать вот такой оператор:

$str = trim ($str);

Функции chop() и ltrim() работают аналогично, но удаляют пробельные символы только в конце и в начале строки соответственно.

Тема 14. PHP. Операторы INCLUDE и REQUIRE. Особенности написания функций.

1. *Инструкция require*
2. *Инструкция include*
3. *Общий синтаксис определения функции*

Вопрос 1. Инструкция require

Эта инструкция позволяет разбить текст программы на несколько файлов. Ее формат такой:

require имя_файла;

При запуске (именно при запуске, а не при исполнении!) программы интерпретатор просто заменит инструкцию на содержимое файла имя_файла (этот файл может также содержать сценарий на PHP, обрамленный, как обычно, тэгами <? и ?>). Причем сделает он это только один раз (в отличие от include, а именно, непосредственно перед запуском программы. Это бывает довольно удобно для включения в вывод сценария всяких "шапок" с HTML-кодом. Рассмотрим примеры:

Пример 1.

<html>
<head><title>Title!</title></head>
<body bgcolor=yellow>

Пример 2. Файл footer.htm

©My company, 1999.
</body></html>

Пример 3. Файл script.php

<?

```
require "header.htm";
. . . работает сценарий и выводит само тело документа
require "footer.htm";
?>
```

Безусловно, это лучше, чем включать весь HTML-код в сам сценарий вместе с инструкциями программы. Вам скажет спасибо тот, кто будет пользоваться вашей программой и захочет изменить ее внешний вид. Однако, несмотря на кажущееся удобство, это все же плохая практика. Действительно, наш сценарий разрастается аж до трех файлов! А как было сказано выше, чем меньше файлов использует программа, тем легче с ней будет работать вашему дизайнеру и верстальщику (которые о PHP имеют слабое представление). О том, как же быть в этой ситуации, я расскажу позже
в пятой части книги, в главе, посвященной технике разделения кода и шаблонов.

Вопрос 2. Инструкция include

Эта инструкция практически идентична require, за исключением того, что включаемый файл вставляется "в сердце" нашего сценария не перед его выполнением, а прямо во время. Например, пусть у нас есть 10 текстовых файлов с именами file0.php, file1.php и так далее до file9.php, содержимое которых просто десятичные цифры 0, 1 ..... 9 (по одной цифре в каждом файле). Запустим такую программу:

```
for($i=0; $i<10; $i++)
{
  include "file$i.php";
}
```

В результате мы получим вывод, состоящий из 10 цифр: "0123456789". Из этого мы можем заключить, что каждый из наших файлов был включен по одному разу прямо во время выполнения цикла!

Обратите внимание на, казалось бы, лишние фигурные скобки вокруг include. Попробуем их убрать. Тут же получаем совершенно бестолковое сообщение об ошибке (или, еще хуже, программа начнет неправильно работать, а причину разыскать будет нелегко). Происходит так потому, что include не является на самом деле оператором в привычном нам смысле этого слова. Чтобы это понять, представьте, что каждый раз, когда интерпретатор встречает инструкцию include, он просто "в лоб" заменяет ее на содержимое файла, указанного в параметре. А если в этом файле несколько команд? Тогда в цикле выполнится только первая из них, а остальные будут запущены уже после окончания цикла. Так что общее правило гласит: всегда обрамляйте инструкцию include фигурными скобками, если размещаете ее внутри какой-либо конструкции.

Трансляция и проблемы с include

Как мы знаем, перед исполнением PHP транслирует программу во внутреннее представление. Это означает, что в памяти создается как бы "полуфабрикат", из которого исключены все комментарии, лишние пробелы, некоторые имена переменных и т. д.

Впоследствии это внутреннее представление интерпретируется (выполняется). Однако мы знаем также, что в программе могут встретиться такие места, "подводные камни" для интерпретатора, которые PHP не сможет оттранслировать заранее. В этом случае он их пропускает, "откладывает на потом", чтобы в момент, когда управление дойдет до определенной точки, опять запустить транслятор. Одним из таких "камней" как раз и является инструкция include. Как только управление программы доходит до нее, PHP вынужден приостановиться и ждать, пока транслятор не оттранслирует код включаемого файла. А это достаточно отрицательно сказывается на быстродействии программы, особенно большой. Поэтому, если вы пишете большой и сложный сценарий, применяйте инструкцию require вместо include, где только можно.

В пользу последнего говорит также и перспектива появления в будущем компилятора для PHP, который будет уметь сохранять оттранслированный код в исполняемые файлы (нечто подобное уже существует для программ на Perl). Если вы будете использовать include, то PHP никак не сможет определить во время компиляции, какие файлы вы собираетесь подключить в программе, поэтому в исполняемый файл их код не войдет.

Что же оптимальнее — require или include? Если вы точно уверены, что определенный файл нужно присоединить ровно один раз и в точно определенное место, то воспользуйтесь require. В противном случае более удачным выбором будет include.

Вопрос 3. Общий синтаксис определения функции

В общем виде синтаксис определения функции таков:

function имя_функции(арг1[=зн1], арг2[=зн2], ... аргN[=знN])
{
операторы_тела_функции;
}

Имя функции должно быть уникальным с точностью до регистра букв. Это означает, что, во-первых, имена MyFunction, myfunction и даже MyFuNcTiOn будут считаться одинаковыми, и, во-вторых, мы не можем переопределить уже определенную функцию (стандартную или нет — не важно), но зато можем давать функциям такие же имена, как и переменным в программе (конечно, без знака $ в начале).

Список аргументов, как легко увидеть, состоит из нескольких

перечисленных через запятую переменных, каждую из которых мы должны будем задать при вызове функции (впрочем, когда для этой переменной присвоено через знак равенства значение по умолчанию.

Конечно, если у функции не должно быть аргументов вовсе (как это сделано у функции time()), то следует оставить пустые скобки после ее имени, например:

function SimpleFunction()
{ ... }

В фигурные скобки заключается тело функции. В нем могут быть любые операторы, включая даже операторы определения других функций.

Тема 15. Работа с классами в PHP

1. *Классы, объекты и объявления методов*
2. *Функции для работы с классами и объектами*

Вопрос 1. Классы, объекты и объявления методов

Классы образуют синтаксическую базу объектно-ориентированного программирования. Их можно рассматривать как своего рода "контейнеры" для логически связанных данных и функций (обычно называемых методами). Класс представляет собой шаблон, по которому создаются конкретные экземпляры, используемые в программе. Экземпляры классов называются объектами.

Чтобы лучше понять связь между классами и объектами, можно представить класс как "чертеж" для создания объектов. По чертежу "изготавливаются" разные объекты, обладающие одними и теми же базовыми характеристиками (например, при строительстве дома — одна дверь, два окна и определенная толщина стены). Тем не менее, каждый объект существует независимо от других — изменение его характеристик никак не влияет на характеристики других объектов; например, в уже построенном доме можно прорубить дополнительное окно. Важно помнить, что у объектов все равно остается общая характеристика — количество окон.

Класс также можно рассматривать как тип данных, а объект — как переменную (по аналогии с тем, как переменная $counter относится к целому, а переменная $last_name — к строковому типу). Программа может одновременно работать с несколькими объектами одного класса как с несколькими переменными целого типа. Общий формат классов PHP приведен в примере (Объявление классов в PHP):

```
class Class_name {
var $attribute_1;
//...
var $attribute_N;
function function1() {
//...
}
//...
function functionN() {
//...
}
} // end Class_name
```

Объявление класса должно начинаться с ключевого слова class (подобно тому, как объявление функции начинается с ключевого слова function). Каждому объявлению атрибута, содержащегося в классе, должно предшествовать ключевое слово var. Атрибуты могут относиться

к любому типу данных, поддерживаемых в PHP; их можно рассматривать как переменные с небольшими различиями, о которых вы узнаете в этой главе. После объявлений атрибутов следуют объявления методов, очень похожие на типичные объявления функций.

По общепринятым правилам имена классов ООП начинаются с прописной буквы, а все слова в именах методов, кроме первого, начинаются с прописных букв (первое слово начинается со строчной буквы). Разумеется, можно использовать любые обозначения, главное — выбрать стандарт и придерживаться его.

Методы часто используются для работы с атрибутами классов. При ссылках на атрибуты внутри методов используется специальная переменная *$this*. Синтаксис методов продемонстрирован в следующем примере:

```
<?
class Webpage {
var $bgcolor;
function setBgColor($color) {
$this->bgcolor = $color;
}
function getBgColor() {
return $this->bgcolor;
}
}
?>
```

Переменная $this ссылается на экземпляр объекта, для которого вызывается метод. Поскольку в любом классе может существовать несколько экземпляров объектов, уточнение $this необходимо для ссылок на атрибуты, принадлежащие текущему объекту. При использовании этого синтаксиса обратите внимание на два обстоятельства:

- атрибут, на который вы ссылаетесь в методе, не нужно передавать в виде параметра функции;
- знак доллара ($) ставится перед переменной $this, но не перед именем атрибута (как у обычной переменной).

Создание объектов и работа с ними

Объекты создаются оператором new. Например, объект класса Webpage создается следующей командой:

```
$some_page = new Webpage;
```

Новый объект с именем $some_page обладает собственным набором атрибутов и методов, перечисленных в классе Webpage. Для изменения значения атрибута $bgcolor, принадлежащего этому конкретному объекту, можно воспользоваться определенным в классе методом setBgColor():

```
$some_page->setBgColor("black");
```

Следует помнить, что PHP также позволяет явно получить значение атрибута указанием имен объекта и атрибута:

$some_page->bgcolor="black";

Конструкторы

Довольно часто при создании объекта требуется задать значения некоторых атрибутов. Разработчики технологии ООП учли это обстоятельство и реализовали его в концепции конструкторов. Конструктор представляет собой метод, который задает значения некоторых атрибутов (а также может вызывать другие методы). Конструкторы вызываются автоматически при создании новых объектов. Чтобы это стало возможным, имя метода-конструктора должно совпадать с именем класса, в котором он содержится. Например (Использование конструктора):

```
<?
class Webpage {
var $bgcolor;
function Webpage($color) {
$this->bgcolor = $color;
}
}
// Вызвать конструктор класса Webpage
$page = new Webpage("brown");
?>
```

Раньше создание объекта и инициализация атрибутов выполнялись раздельно. Конструкторы позволяют выполнить эти действия за один этап.

В зависимости от количества передаваемых параметров могут вызываться разные конструкторы. Например, в рассмотренном примере объекты класса *Webpage* могут создаваться двумя способами. Во-первых, можно вызвать конструктор, который просто создает объект, но не инициализирует его атрибуты:

$page = new Webpage;

Во-вторых, объект можно создать при помощи конструктора, определенного в классе, — в этом случае создается объект класса Webpage и присваивается значение его атрибуту bgcolor:

$page = new Webpage("brown");

Деструкторы

Как упоминалось ранее, в PHP отсутствует непосредственная поддержка деструкторов. Тем не менее, можно легко имитировать работу деструктора, вызывая функцию PHP unset(). Эта функция уничтожает содержимое переменной и возвращает занимаемые ею ресурсы системе. С объектами unset() работает так же, как и с переменными. Допустим, вы

работаете с объектом $Webpage. После завершения работы с этим конкретным объектом вызывается функция:

unset($Webpage);

Эта команда удаляет из памяти все содержимое $Webpage. Действуя в духе инкапсуляции, можно поместить вызов unset() в метод с именем destroy() и затем вызвать его:

$Webpage->destroy();

Необходимость в вызове деструкторов возникает лишь при работе с объектами, использующими большой объем ресурсов, поскольку все переменные и объекты автоматически уничтожаются по завершении сценария.

Простое и иерархическое наследование

Класс является шаблоном, по которому создаются реальные объекты с определенными характеристиками и функциями. Нетрудно представить себе ситуацию, при которой такой объект является частью другого объекта. Например, автомобиль можно считать частным случаем категории "транспортное средство", к которой относятся и самолеты. Хотя разные типы транспортных средств сильно отличаются друг от друга, все они характеризуются атрибутами из общего набора (количество колес, мощность, максимальная скорость, модель и т. д.). Пусть конкретные значения этих атрибутов сильно различаются — атрибуты все равно присущи всем транспортным средствам. Таким образом, субклассы "автомобиль" и "самолет" наследуют общий набор базовых характеристик от суперкласса "транспортное средство". Концепция получения классом характеристик от другого, более общего класса называется наследованием.

Наследование является исключительно полезным средством программирования, поскольку его применение предотвращает копирование кода, совместно используемого структурами данных, — например, общих характеристик различных типов транспортных средств, упоминавшихся в предыдущем абзаце. В общем случае синтаксис наследования характеристик другого класса в PHP выглядит так:

```
class Class_name2 extends Class_name1
{
   объявления атрибутов;
   объявления методов;
}
```

Ключевое слово *extends* говорит о том, что класс *Class_name2* наследует все характеристики класса *Class_name1*.

Помимо возможности многократного использования кода, наследование обладает еще одним важным преимуществом — снижается

вероятность ошибок при модификации программы.

Вызов конструктора производного класса не приводит к автоматическому вызову конструктора базового класса.

```
class Class_name2 extends Class_name1
{
//объявления атрибутов;
//объявления методов;
}
```

Представление различных типов транспортных средств при помощи наследования:

```
<?
// Транспортное средство
class Vehicle {
var $model;
var $current_speed;
function setSpeed($mph) {
$this->current_speed = $mph;
}
function getSpeed() {
return $this->current_speed;
}
}
// Автомобиль
class Auto extends Vehicle {
var $fuel_type;
function setFuelType($fuel) {
$this->fuel_type = $fuel;
}
function getFuelType() {
return $this->fuel_type;
}
}
// Самолет
class Airplane extends Vehicle {
var $wingspan;
function setWingSpan($wingspan) {
$this->wingspan = $wingspan;
}
function getWingSpan() {
return $this->wingspan;
}
```

```
}
?>
```

Объекты этих классов создаются следующим образом:

```
$tractor = new Vehicle;
$gulfstream = new Airplane;
```

Приведенные команды создают два объекта. Первый объект, $tractor, относится к классу Vehicle. Второй объект, $gulfstream, относится к классу Airplane и потому обладает как общими характеристиками класса Vehicle, так и уточненными характеристиками класса Airplane.

Ситуация, при которой класс наследует свойства нескольких родительских классов, называется множественным наследованием. К сожалению, в PHP множественное наследование не поддерживается. Например, следующая конструкция невозможна в PHP:

```
class Airplane extends Vehicle extends Building...
```

Многоуровневое наследование

С увеличением размеров и сложности программ может возникнуть необходимость в многоуровневом наследовании. Иначе говоря, класс будет наследовать свои свойства от других классов, которые, в свою очередь, будут наследовать от третьих классов и т.д. Многоуровневое наследование развивает модульную структуру программы, обеспечивая простоту сопровождения и более четкую логическую структуру. Скажем, при использовании примера с транспортными средствами в большой программе может появиться необходимость в дополнительном разбиении на субклассы суперкласса Vehicle, продолжающем логическое развитие иерархии. Например, транспортные средства можно дополнительно разделить на наземные, морские и воздушные, чтобы суперкласс специализированных субклассов выбирался в зависимости от среды, в которой перемещается данное транспортное средство.

Приведенный пример, подчеркивает некоторые важные аспекты многоуровневого наследования в PHP.

Многоуровневое наследование

```
class Vehicle {
//Объявления атрибутов...
//Объявления методов...
}
class Land extends Vehicle {
//Объявления атрибутов...
//Объявления методов...
}
class Car extends Land {
//Объявления атрибутов...
```

```
//Объявления методов...
}
$nissan = new Car;
?>
```

Объект *$nissan* содержит все атрибуты и методы классов Car, Land и Vehicle. Как видите, программа получается исключительно модульной.

Допустим, когда-то в будущем вы захотите добавить в класс Land новый атрибут. Нет проблем: внесите соответствующие изменения в класс Land, и этот атрибут немедленно становится доступным для классов Land и Car, не влияя на функциональность других классов. Таким образом, модульность кода и гибкость относятся к числу основных преимуществ ООП.

Хотя масс наследует свои характеристики от цепочки родителей, конструкторы родительских классов не вызываются автоматически при создании объектов класса-наследника. Эти конструкторы могут вызываться классом-наследником в виде методов.

Абстрактные классы

В некоторых ситуациях бывает удобно создать класс, объекты которого никогда не создаются (данный класс нужен всего лишь как базовый для создания производных классов). Такие классы называются абстрактными. Абстрактные классы обычно применяются в тех случаях, когда разработчик программы хочет обеспечить обязательную поддержку некоторых функциональных возможностей всеми классами, производными от абстрактного базового класса.

В PHP отсутствует прямая поддержка абстрактных классов, однако существует простое обходное решение — достаточно определить в "абстрактном" классе конструктор и включить в него вызов die(). Вернемся к классам из листинга 6.4. Скорее всего, вам никогда не придется создавать экземпляры классов Land и Vehicle, поскольку они не могут представлять физические объекты. Для представления реальных объектов (например, автомобилей) следует создать класс, производный от этих классов. Следовательно, чтобы предотвратить возможное создание объектов классов Land и Vehicle, необходимо включить в их конструкторы вызовы die(), как показано в примере:

Создание абстрактных классов.

```
<?
class Vehicle {
//Объявления атрибутов...
function Vehicle() {
die ("Cannot create Abstract Vehicle class!");
}
//Объявления других методов...
```

```
}
class Land extends Vehicle {
//Объявления атрибутов...
function Land() {
die ("Cannot create Abstract Land class!");
}
//Объявления других методов.
}
class Car extends Land {
//Объявления атрибутов...
//Объявления методов...
}
?>
```

Попытка создания экземпляра этих абстрактных классов приведет к выдаче сообщения об ошибке и завершению программы.

Перегрузка методов

Перегрузкой методов называется определение нескольких методов с одинаковыми именами, но разным количеством или типом параметров. Как и в случае с абстрактными классами, в PHP эта возможность не поддерживается, но существует простое обходное решение, приведенное в примере:

Перегрузка методов.

```
<?
class Page {
var $bgcolor;
var $textcolor;
function Page() {
// Определить количество переданных аргументов
// и вызвать метод с нужным именем
$name = "Page".func_num_args();
// Call $name with correct number of arguments passed in
if ( func_num_args() == 0 ) :
$this->$name();
else :
$this->$name(func_get_arg(0));
endif;
}
function Page0() {
$this->bgcolor = "white";
$this->textcolor = "black";
print "Created default page";
}
```

```
function Page1($bgcolor) {
$this->bgcolor = $bgcolor;
$this->textcolor = "black";
print "Created custom page";
}
}
$html_page = new Page("red");
?>
```

В этом примере при создании нового объекта с именем $html_page передается один аргумент. Поскольку в классе был определен конструктор по умолчанию (Page()), вызывается именно он. Однако конструктор по умолчанию всего лишь выбирает, какому из конструкторов (Page0() или Page1()) следует передать управление. При выборе конструктора используются функции func_num_args() и func_get_arg(), которые, соответственно, определяют количество аргументов и читают эти аргументы.

Конечно, такое решение вряд ли можно назвать полноценной перегрузкой, но оно подойдет для тех, кто не может жить без этого важного аспекта ООП.

Вопрос 2. Функции для работы с классами и объектами

В PHP существует несколько стандартных функций для работы с классами и объектами; эти функции рассматриваются в следующих разделах. Все они часто используются на практике, особенно в процессе разработки интерфейса, администрирования кода и диагностики ошибок.

get_class_methods()

Функция get_class_methods() возвращает массив имен методов класса с заданным именем. Синтаксис функции get_class_methods():

array get_class_methods (string имя_класса)

Приведем простой пример использования get_class_methods():

Получение списка методов класса.

```
<?
//...
class Airplane extends Vehicle {
var $wingspan;
function setWingSpan($wingspan) {
$this->wingspan = $wingspan;
}
function getWingSpan() {
return $this->wingspan;
}
}
```

```
$cls_methods = get_class_methods("Airplane");
// Массив $cls_methods содержит имена всех методов,
// объявленных в классах "Airplane" и "Vehicle"
?>
```

Как видно из примера, функция get_class_methods() позволяет легко получить информацию обо всех методах, поддерживаемых классом.

get_class_vars()

Функция get_class_vars() возвращает массив имен атрибутов класса с заданным именем. Синтаксис функции get_class_vars():

array get_class_vars (string имя_класса)

Приведен пример использования get_class_vars():

Получение списка атрибутов класса функцией get_class_vars().

```
<?
class Vehicle {
var $model;
var $current_speed;
}
class Airplane extends Vehicle {
var $wingspan = 200;
}
$a_class = "Airplane";
$attribs = get_class_vars($a_class);
// $attribs = array ("model" => "",
// "current_speed" => "",
// "wingspan" => "200");
?>
```

get_object_vars()

Функция get_object_vars() возвращает ассоциативный массив с информацией обо всех атрибутах объекта с заданным именем. Синтаксис функции get_object_vars():

array get_object_vars (object имя_обьекта)

Приведем пример использования функции get_object_vars():.

Получение информации о переменных объекта.

```
<?
class Vehicle {
var $wheels;
}
class Land extends Vehicle {
var $engine;
}
class car extends Land {
var $doors;
```

```
function car($doors, $eng, $wheels) {
$this->doors = $doors;
$this->engine = $eng;
$this->wheels = $wheels;
}
function get_wheels() {
return $this->wheels;
}
}
$toyota = new car(2,400,4);
$vars = get_object_vars($toyota);
while (list($key, $value) = each($vars)) :
print "$key ==> $value <br>";
endwhile;
// Выходные данные:
// wheels ==> 2
// engine ==> 400
// doors ==> 2
?>
```

Функция get_object_vars() позволяет быстро получить всю информацию об атрибутах конкретного объекта и их значениях в виде ассоциативного массива.

method_exists()

Функция method_exists() проверяет, поддерживается ли объектом метод с заданным именем. Если метод поддерживается, функция возвращает TRUE, в противном случае возвращается FALSE. Синтаксис функции method_exists():

bool method_exists (object имя_обьекта. string имя_метода)

Приведем пример использования метода method_exists():

Проверка поддержки метода объектом при помощи функции method_exists(). <?

```
class Vehicle {
//...
}
class Land extends Vehicle {
var $fourWheel;
function setFourWheelDrive() {
$this->fourWeel = 1;
}
}
```

```
// Создать объект с именем $car
$car = new Land;
// Если метод "fourWheelDrive" поддерживается классом "Land"
// или "Vehicle", вызов method_exists возвращает TRUE;
// в противном случае возвращается FALSE.
// В данном примере method_exists() возвращает TRUE.
if (method_exists($car, "setfourWheelDrive")) :
print "This car is equipped with 4-wheel drive";
else :
print "This car is not equipped with 4-wheel drive";
endif;
?>
```

В примере функция method_exists() проверяет, поддерживается ли объектом $car метод с именем setFourWheelDrive(). Если метод поддерживается, функция возвращает логическую истину и фрагмент выводит соответствующее сообщение. В противном случае возвращается FALSE и выводится другое сообщение.

get_class()

Функция get_class() возвращает имя класса, к которому относится объект с заданным именем. Синтаксис функции get_class():

string get_class(object имя_объекта);

Приведем пример использования get_class():

Получение имени класса функцией get_class().

```
<?
class Vehicle {
//...
}
class Land extends Vehicle {
//...
}
// Создать объект с именем $car
$car = new Land;
// Переменной $class_a присваивается строка "Land"
$class_a = get_class($car);
?>
```

В результате переменной $class_a присваивается имя класса, на основе которого был создан объект $car.

get_parent_class()

Функция get_parent_class() возвращает имя родительского класса (если он есть) для объекта с заданным именем. Синтаксис функции get_parent_dass():

string get_parent_class (object имя_обьекта);

Приведем пример, который демонстрирует использование get_parent_class().

Получение имени родительского класса функцией get_parent_class().

```
<?
class Vehicle {
//...
}
class Land extends Vehicle {
//...
}
// Создать объект с именем $car
$car = new Land;
// Переменной $parent присваивается строка "Vehicle"
$parent = get_parent_class($car);
?>
```

Как и следовало ожидать, при вызове get_parent_class() переменной $parent будет присвоена строка "Vehicle".

is_subclass_of()

Функция is_subclass_of() проверяет, был ли объект создан на базе класса, имеющего родительский класс с заданным именем. Функция возвращает TRUE, если проверка дает положительный результат, и FALSE в противном случае. Синтаксис функции is_subclass_of():

bool is_subclass_of (object объект, string имя_класса)

Использование is_subclass_of() продемонстрировано в примере:

Использование функции is_subdass_of().

```
<?
class Vehicle {
//...
}
class Land extends Vehicle {
//...
}
$auto = new Land;
// Переменной $is_subclass присваивается TRUE
$is_subclass = is_subclass_of($auto, "Vehicle");
?>
```

В приведенном примере переменной $is_subclass() присваивается признак того, принадлежит ли объект $auto к субклассу родительского класса Vehicle. В приведенном фрагменте $auto относится к классу Vehicle; следовательно, переменной $is_subclass() будет присвоено значение TRUE.

get_declared_classes()

Функция get_declared_classes() возвращает массив с именами всех определенных классов. Синтаксис функции get_declared_classes():

array get_declared_classes()

Получение списка классов функцией get_declared_classes():

```
 <?
class Vehicle {
//...
}
class Land extends Vehicle {
//...
}
$declared_classes = get_declared_classes();
// $declared_classes = array("Vehicle", "Land")
}
?>
```

Тема 16. PHP. Доступ к базам данных MySQL.

1. *Работа с базой данных MySQL*
2. *Язык запросов MySQL*
3. *Работа с таблицами в MySQL*

Вопрос 1. Работа с базой данных MySQL

База данных — совокупность связанных данных, сохраняемая в двумерных таблицах информационной системы. Программное обеспечение информационной системы, обеспечивающей создание, ведение и совместное использование баз данных, называется системой управления базами данных (СУБД). Рассмотрим функции PHP, предназначенные для работы с одной из самых популярных СУБД — MySQL.

В PHP есть функции для "общения" и с другими системами управления базами данных (например, Sybase, Oracle и т. д.), но остановимся именно на MySQL в силу ее простоты и универсальности для большинства приложений. Конечно, прежде чем работать с MySQL, нужно установить соответствующее программное обеспечение — программу-сервер MySQL.

Одна из самых популярных СУБД, которые используются в Web-программировании, — MySQL. Она предназначена для создания небольших (сравнительно, конечно — скажем, не более 100 Мбайт) баз данных, и поддерживает некоторое подмножество языка запросов SQL.

SQL — специально разработанный стандарт языка запросов к базам данных. В нем присутствуют такие команды, как:

- создание/удаление таблицы;
- создание записей в заданной таблице;
- поиск/удаление записей;

- обновление некоторых полей указанной записи.

Итак, с точки зрения программы база данных MySQL представляет собой удачно организованный набор поименованных таблиц. Каждая таблица — массив (возможно, очень большой) из однородных элементов, которые будем называть записями. В принципе, запись — неделимая единица информации в базе данных, хотя по запросу можно получать и не всю ее целиком, а только какую-то часть. Запись может содержать в себе одно или несколько именованных полей. Число и имена полей задаются при создании таблицы. Каждое поле имеет определенный тип (например, целое число, строка текста, массив символов и т. д).

В таблицу всегда можно добавить новую запись. Другая операция, которую часто производят с записью (точнее, с таблицей) — это поиск. Например, запрос поиска может быть таким: "Выдать все записи, в первом поле которых содержится число, меньшее 10, во втором — строка, включающая слово word, а в третьем — не должен быть ноль". Из найденных записей в программу можно извлекать какие-то части данных (или не извлекать), также записи таблицы можно удалить.

Отметим, что обычно все упомянутые операции осуществляются очень быстро. Например, Microsoft SQL Server может за 0,01 секунды из 10 миллионов записей выделить ту, у которой значение определенного поля совпадает с нужным числом или строкой. Высокое быстродействие в большей мере обусловлено тем, что данные не просто "свалены в кучу", а определенным образом упорядочены и все время поддерживаются в таком состоянии.

Соединение с базой данных

Прежде чем работать с базой данных, необходимо установить с ней сетевое соединение, а также провести авторизацию пользователя. Для этого служит функция *mysql_connect().*

int mysql_connect ([string $hostname] [,string $username] [,string $password])

Функция *mysql_connect()* устанавливает сетевое соединение с базой данных MySQL, расположенной на хосте Shostname (по умолчанию это localhost, т. е. текущий компьютер), и возвращает идентификатор открытого соединения. Вся дальнейшая работа ведется именно с этим идентификатором. При регистрации указывается имя пользователя $username и пароль $password (по умолчанию имя пользователя, от которого запущен текущий процесс, и пустой пароль). Строка $hostname также может включать в себя номер порта в формате: имя_хоста:порт (если сервер MySQL настроен не на стандартный, а на какой-то другой порт, что делать, вообще говоря, не рекомендуется).

При следующем запуске функции с теми же самыми аргументами второе соединение не будет открыто, а функция возвратит идентификатор уже существующего. Соединение с MySQL-сервером будет автоматически закрыто по завершении работы сценария, либо же при вызове функции mysql_close(). Если вы планируете открывать только одно соединение с базой данных за все время работы сценария, то можете не сохранять возвращенное значение, а также не указывать идентификатор соединения
при вызове всех остальных функций.

int mysql_select_db(string $dbname [,int $link_identifier])

До того как послать первый запрос серверу MySQL, необходимо указать, с какой базой данных мы собираемся работать. Для этого и предназначена описываемая функция. Она уведомляет PHP, что в дальнейших операциях с соединением $link_identifier (или с последним открытым соединением, если указанный параметр не задан) будет использоваться база данных $dbname.

Если в процессе работы с MySQL возникают ошибки (например, в запросе не сбалансированы скобки или же не хватает параметров), то сообщение об ошибке и ее номер можно получить с помощью следующих двух функций.

int mysql_errno([int $link_identifier])

Функция возвращает номер последней зарегистрированной ошибки. Идентификатор соединения $link_identifier можно не указывать, если за время работы сценария было установлено только одно соединение.

string mysql_error([int $link_identifier])

Эта функция возвращает не номер, а строку, содержащую текст сообщения об ошибке. Ее удобно применять в отладочных целях.

Выполнение запросов к базе данных

Теперь подойдем непосредственно к тому, как формировать и посылать запросы к базе данных. Для этого существует одна-единственная функция — *mysql_query()* — и возвращает она не что иное, как идентификатор результирующего набора данных.

Результат сразу не пересылается клиенту и чтобы до него добраться, и служит этот идентификатор. Существует очень много функций, которые принимают его в качестве параметра и возвращают те или иные данные. Их рассмотрим чуть позже.

int mysql_query(string $query [,int $link_identifier])

Эта функция в своем роде универсальна: она посылает MySQL-серверу запрос $query и возвращает идентификатор ответа, или результата. Параметр $query представляет собой строку, составленную по правилам языка SQL. Используется установленное ранее соединение $link_identifier, а в случае его отсутствия — последнее открытое

соединение.

Есть несколько команд SQL, которые возвращают только признак, успешно они выполнились или нет (например, это команды UPDATE, INSERT и т. д.). В таком случае этот признак и будет возвращен функцией. Наоборот, для запроса SELECT возвращается как раз идентификатор вывода, нулевое значение которого свидетельствует о том, что произошла ошибка. На самом деле существует еще одна функция для выполнения запроса, но использовать ее менее удобно, поскольку всякий раз приходится указывать имя базы данных, к которой осуществляется доступ.

int mysql(string $dbname, string $query [,int $link_identifier])

Служит для тех же целей, что и функция mysql_query(), только обращение осуществляется не к текущей выбранной базе данных, а к указанной в параметре $dbname. Если вы владеете сразу несколькими базами данных и обращаетесь к ним одновременно, то, возможно, применение этой функции окажется оправданным. Как обычно, параметр $link_identifier можно опустить, тогда используется последнее открытое соединение.

Вопрос 2. Язык запросов MySQL

Все без исключения запросы к базе данных посылаются при помощи одной единственной функции — mysql_query() (или mysql()). Они должны передаваться ей в виде строкового параметра. Этот параметр, впрочем, может быть и многострочным — т. е., содержать символы перевода строки. MySQL допускает включение любого количества пробелов, символов табуляции или перевода строки везде, где разрешено использование одного пробела (в этом смысле он похож на PHP и большинство других языков программирования).

Язык SQL позволяет нам создавать довольно сложные запросы. Ниже перечислены наиболее употребительные команды MySQL.

Создание таблицы

create table ИмяТаблицы(ИмяПоля тип, ИмяПоля тип, ...)

Этой командой в базе данных создается новая таблица с колонками (полями), определяемыми своими именами (ИмяПоля) и указанными типами.

Типы полей

Перечислим все типы полей, которые могут применяться в MySQL.

Целые числа

Существует несколько разных типов целых чисел, различающихся количеством байтов данных, которые отводятся в базе данных для их хранения. Все эти типы рознятся только названиями и (с некоторыми сокращениями) записываются так:

префиксINT [UNSIGNED]

Необязательный флаг UNSIGNED задает, что будет создано поле для хранения беззнаковых чисел (больших или равных 0). Имена типов, в вобщем виде обозначенные здесь как префиксINT, приводятся в табл. 1.

Табл. 1. Типы целочисленных данных MySQL

Тип	**Описание**
TINYINT	Может хранить числа от –128 до +127
SMALLINT	Диапазон от –32 768 до 32 767
MEDIUMINT	Диапазон от –8 388 608 до 8 388 607
INT	Диапазон от –2 147 483 648 до 2 147 483 647
BIGINT	Диапазон от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807

Дробные числа

Точно так же, как целые числа подразделяются в MySQL на несколько разновидностей, MySQL поддерживает и несколько типов дробных чисел. В общем виде они записываются так:

ИмяТипа[(length,decimals)] [UNSIGNED]

Здесь length — количество знакомест (ширина поля), в которых будет размещено дробное число при его передаче в PHP, а decimals — количество знаков после десятичной точки, которые будут учитываться. Как обычно, UNSIGNED задает беззнаковые числа. Строка ИмяТипа замещается на предопределенные значения, соответствующие возможным вариантам представления вещественных чисел (табл. 2).

Табл. 2. Типы рациональных чисел в MySQL

Тип	**Описание**
FLOAT	Число с плавающей точкой небольшой точности
DOUBLE	Число с плавающей точкой двойной точности
REAL	Аналогичен DOUBLE
DECIMAL	Дробное число, хранящееся в виде строки
NUMERIC	Аналогичен DECIMAL

Строки

Строки представляют собой массивы символов. Обычно при поиске по текстовым полям по запросу SELECT не берется в рассмотрение регистр символов, т. е. строки "Вася" и "ВАСЯ" считаются одинаковыми. Кроме того, если база данных настроена на автоматическую перекодировку текста при его помещении и извлечении, эти поля будут храниться в указанной вами кодировке.

Тип строка может хранить не более length символов, где length принадлежит диапазону от 1 до 255.

VARCHAR(length) [BINARY]

При занесении некоторого значения в поле такого типа из него автоматически вырезаются концевые пробелы. Если указан флаг BINARY, то при запросе SELECT строка будет сравниваться с учетом регистра. Тип VARCHAR неудобен тем, что может хранить не более 255 символов. Вместо него рекомендуется использовать другие текстовые типы, перечисленные в табл. 3.

Табл. 3. Строковые типы данных таблиц MySQL

Тип	**Описание**
TINYTEXT	Может хранить максимум 255 символов
TEXT	Может хранить не более 65 535 символов
MEDIUMTEXT	Может хранить максимум 16 777 215 символов
LONGTEXT	Может хранить 4 294 967 295 символов

Чаще всего применяется тип TEXT, но если вы не уверены, что данные не будут всегда короче 65 536 байтов, используйте LONGTEXT.

Бинарные данные

Бинарные данные — это почти то же самое, что и данные в формате TEXT, но только при поиске в них учитывается регистр символов ("abc" и "ABC" — разные строки). Вего имеется 4 типа бинарных данных (табл. 4).

Табл. 4. Типы бинарных данных, используемые в MySQL

Тип	**Описание**
TINYBLOB	Может хранить максимум 255 символов
BLOB	Может хранить не более 65 535 символов
MEDIUMBLOB	Может хранить максимум 16 777 215 символов
LONGBLOB	Может хранить 4 294 967 295 символов

BLOB-данные не перекодируются автоматически, если при работе с установленным соединением включена возможность перекодирования текста "на лету".

Дата и время

MySQL поддерживает несколько типов полей, специально приспособленных для хранения дат и времени в различных форматах (табл. 5).

Табл. 5. Представление дат и времени в базах данных MySQL

Тип	**Описание**
DATE	Дата в формате ГГГГ-ММ-ДД
TIME	Время в формате ЧЧ:ММ:СС

DATETIME	Дата и время в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС
TIMESTAMP	Время и дата в формате timestamp. Однако при получении значения поля оно отображается не в формате timestamp, а в виде ГГГГММДДЧЧММСС, что сильно умаляет преимущества его использования в PHP

Надо заметить, что в PHP будет проще самостоятельно генерировать дату и время при вставке данных в таблицу, а не задействовать встроенные в MySQL типы. Например, привлекательный с виду тип TIMESTAMP на деле оказывается довольно неудобным, потому что отображается не в том виде, который мы ожидаем.

Перечисления и множества

MySQL поддерживает еще несколько специфических типов данных, использовать которые в PHP вряд ли целесообразно. Например, тип перечисления задает, что значение соответствующего поля может быть не любой строкой или числом, а только одним из нескольких указанных при создании таблицы значений: value1, value2 и т. д. Вот как выглядит имя типа перечисления:

ENUM(value1,value2,value3,...)

В отличие от всех остальных типов, множества означают, что в соответствующем поле может содержаться не одно, а сразу несколько значений (value1, value2 и т. д., т. е. — множество значений). Формат задания данных такого типа имеет следующий вид:

SET(value1,value2,value3,...)

Удаление таблицы

drop table ИмяТаблицы

Удаляет таблицу ИмяТаблицы. Таблица не обязательно должна быть пустой, так что будьте внимательны, чтобы случайно не "аннулировать" нужную таблицу с данными.

Вставка записи

insert into ИмяТаблицы(ИмяПоля1 ИмяПоля2 ...) values('зн1','зн2',...)

Добавляет в таблицу ИмяТаблицы запись, у которой поля, обозначенные как ИмяПоляN, установлены в значения, соответственно, знN. Те поля, которые в этой команде не перечислены, получают "неопределенные" значения (неопределенное значение — это не пустая строка, а просто признак, который говорит MySQL, что у данного поля нет никакого значения). Впрочем, если для не указанного здесь поля при создании таблицы был задан not null, то данная команда закончится неуспешно.

Значения полей можно заключать и в обычные кавычки, хотя,

апострофы тут использовать удобнее. При вставке в таблицу бинарных данных (или текстовых, содержащих апострофы и слэши) некоторые символы должны быть "защищены" обратными слэшами, а именно, символы \, ' и символ с нулевым кодом.

Удаление записей

delete from ИмяТаблицы where Выражение

Удаляет из таблицы ИмяТаблицы все записи, для которых выполнено Выражение. Параметр Выражение — это просто логическое выражение, составленное почти что по правилам PHP. Вот показательный пример:

*(id<10) and (name regexp 'a*b') and (age=25)*

В выражении, помимо имен полей, констант и операторов, могут также встречаться простейшие "вычисляемые" части, например: (id<10+11*234).

Поиск записей

*select * from Таблица where Выражение [order by ИмяПоля [desc]]*

Эта команда — основная и очень мощная. Предназначена она для того, чтобы искать все записи, удовлетворяющие выражению Выражение. Если записей несколько, то при указанном предложении order by они будут отсортированы по тому полю, имя которого записывается правее этого ключевого слова (если задан описатель desc, то упорядочивание происходит в обратном порядке). В предложении order by могут также задаваться несколько полей.

Особое значение имеет символ *. Он предписывает, что из отобранных записей следует извлечь все поля, когда будет выполнена команда получения выборки. С другой стороны, вместо звездочки можно через запятую непосредственно перечислить имена полей, которые требуют извлечения. Но чаще всего все же используется именно *.

Обновление записей

update Таблица set(ИмяПоля1='зн1', ИмяПоля1='зн2', ...) where Выражение

В таблице Таблица для всех записей, удовлетворяющих выражению Выражение, указанные поля устанавливаются в соответствующие значения. Эта команда часто отдается, если не требуется обновлять сразу все поля какой-то записи, а нужно затронуть только некоторые.

Получение результата

После того как запрос выполнен и идентификатор результирующего набора данных возвращен, возможно, необходимо будет получить этот самый результат. Результат — это просто набор данных, и количество вошедших в него записей, можно узнать через *mysql_num_rows()*.

Вопрос 3. Работа с таблицами в MySQL

Обычно в таблице содержится довольно много записей с разными значениями полей. Встает проблема выбора одной конкретной записи из этого массива. Например, если в таблице с информацией о гражданах, необходимо идентифицировать по фамилии человека. Ну а если встретятся однофамильцы? Тогда по имени. А если же и имена совпадут? Во избежание недоразумений подобного рода в таблицу вводят еще одно вспомогательное поле (колонку), назвав ее, скажем, id. Этот самый id уникален у каждой записи, поэтому мы можем, зная id нужного нам человека, тут же получить его данные. Кроме того, если понадобится, например, зафиксировать в таблице еще и родственные связи людей (кто является, чьим отцом, например), можно завести в ней еще одно поле — parent_id, в котором будет храниться id родителя конкретного человека. Таким образом, описанная техника оказывается довольно удобной.

А если необходимо добавить в таблицу сведения о еще одном человеке. Логично было бы, чтобы его id проставлялся автоматически. Возникает вопрос: как этот id вычислить? В самом деле, мы же не знаем, какие id в таблице в данный момент "свободны"... Можно использовать для этой цели текущее время в секундах. Но вдруг именно в данную секунду кто-то еще точно так же добавляет в базу данных запись? Можно, конечно, взять максимальный id, прибавить к нему единичку и занести в таблицу. Но где гарантия, что, опять же в этот момент другой администратор не проделал ту же операцию — до того, как вы добавили свою информацию, но после того, как определили максимальный id?

Как раз для решения этой проблемы и предназначена в MySQL возможность под названием *AUTO_INCREMENT*. А именно, при создании таблицы можно какое-нибудь ее поле (в нашем случае как раз id) объявить так:

ИмяПоля int auto_increment primary key

Теперь любая операция INSERT автоматически проставит у добавленной записи поле ИмяПоля так, чтобы оно было уникально во всей таблице — MySQL это гарантирует. В простейшем случае — просто увеличит на 1 некий внутренний счетчик, глобальный для всей таблицы, и занесет его новое значение в нужное поле записи. Причем гарантируется, что никакие проблемы с совместным доступом к таблице просто не могут возникнуть.

Получить только что вставленный идентификатор можно при помощи функции *mysql_insert_id().*

int mysql_insert_id([int $link_identifier])

Функция возвращает непосредственно перед ее вызовом сгенерированный идентификатор записи для автоинкрементного поля после выполнения команды insert. Вызывать ее разумно только сразу

после выполнения инструкции insert, например, в таком контексте:
mysql_query("insert into Таблица(поле1, поле2) values('aa','bb')");
$id=mysql_insert_id();

Теперь к только что вставленной записи можно обратиться, используя идентификатор $id:
$r=mysql_query("select * from Таблица where id=$id");
$Row=mysql_fetch_array($r);

Рассмотрим несколько функций, имеющих отношение к работе с таблицами в целом.
int mysql_list_fields(string $dbname, string $tblname [,int $link])

Функция *mysql_list_fields()* возвращает информацию об указанной таблице $tblname в базе данных $dbname, используя идентификатор соединения $link, если он задан (в противном случае — последнее открытое соединение). Возвращаемое значение — идентификатор результата, который может быть проанализирован обычными средствами, либо при помощи функций mysql_field_flags(), mysql_field_len(), mysql_field_name() и mysql_field_type(). В случае ошибки возвращается –1, текст сообщения ошибки может быть получен обычным способом.
int mysql_list_tables(string $database [,int $link_identifier])

Функция возвращает идентификатор результата (одна колонка), в котором содержатся имена всех таблиц, присутствующих в базе данных. Для извлечения этих имен можно использовать функцию *mysql_result()* с номером колонки, равным 0.

Тема 17. Активные серверные страницы (ASP). Особенности синтаксиса.

1. *ASP – активные серверные страницы*
2. *Создание простейшей ASP-страницы*

Вопрос 1. ASP – активные серверные страницы.

Ели при создании Web-страниц пользоваться только языком HTML, страницы получаются статическими (т.е. неизменяемые). Страницу можно нафаршировать картинками, анимацией и другими мультимедийными эффектами, но каждый посетитель, заходя на страницу, будет видеть одно и то же содержимое — пока вы вручную не измените его. Для отображения на Web-странице динамического (т.е. постоянно изменяющегося) содержимого нужно нечто большее, чем просто HTML. Например, для отображения даже такой простой вещи, как текущие дата и время, с помощью HTML придется постоянно редактировать Web-страницу на протяжении всего дня — и так каждый день! Даже если на домашней странице сайта достаточно отображать лишь текущую дату, то каждый раз при создании нового сайта в список Web-страниц, которые необходимо редактировать в полночь, будет добавляться еще одна строка.

Активные серверные страницы (Active Server Pages — ASP)

позволяют создавать динамические Web-страницы, комбинируя HTML-дескрипторы и текст с инструкциями сценария, написанными на языке, подобном VBScript, JScript (версия JavaScript от компании Microsoft), PerlScript, Python или Rexx (и это еще не все).

HTML-дескрипторы и текст обеспечивают статическую (неизменяемую) часть страницы, а сценарий генерирует динамическое (т.е. изменяемое) содержимое, которое должен вставить Web-сервер. Например, вместо ручного ввода (а затем изменения) в Web-странице текущей даты и времени можно сформировать ASP-страницу, в которой между ограничителями "<%" и "%>" вставлены следующие операторы VBScript:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//ENM>
<html>
<body bgcolor="#ADD8E6">
<h1><center>HTML and We b Design Tips & Techniques</center></hl>
<hr>
Добро пожаловать, посетитель!<br>
Текущее время в Киеве: <% = time %>,
дата: <% = date %>.<br>
При каждом обновлении (или новой загрузке) Web-страницы время и
(раз в сутки) дата будут меняться.
</body>
</html>
```

При запросе Web-браузера ASP-страницы (обычно это файл с расширением .asp) сервер находит эту страницу и передает ее обработчику сценариев ASP, который в среде Windows находится в файле asp.dll. Обработчик сценариев производит разбор документа и посылает операторы сценария, обнаруженные между начальным (<%) и конечным (%>) дескрипторами, на обработку соответствующему интерпретатору сценариев. Значит, в операционной системе Windows обработчик asp.dll передает VBScript-сценарий, включенный в ASP-файл, интерпретатору VBScript, операторы JScript — интерпретатору JScript и т.д.

В свою очередь, интерпретатор сценария интерпретирует оператор(ы) сценария и возвращает результат (если он есть) в виде строки HTML (дескрипторы и текст) обработчику сценариев asp.dll. Затем обработчик сценариев вставляет в ASP-документ вместо сценария результат его выполнения и передает измененную страницу Web-серверу и, наконец, сервер пересылает страницу браузеру, который запросил ее. Весь этот процесс показан на рис.1.

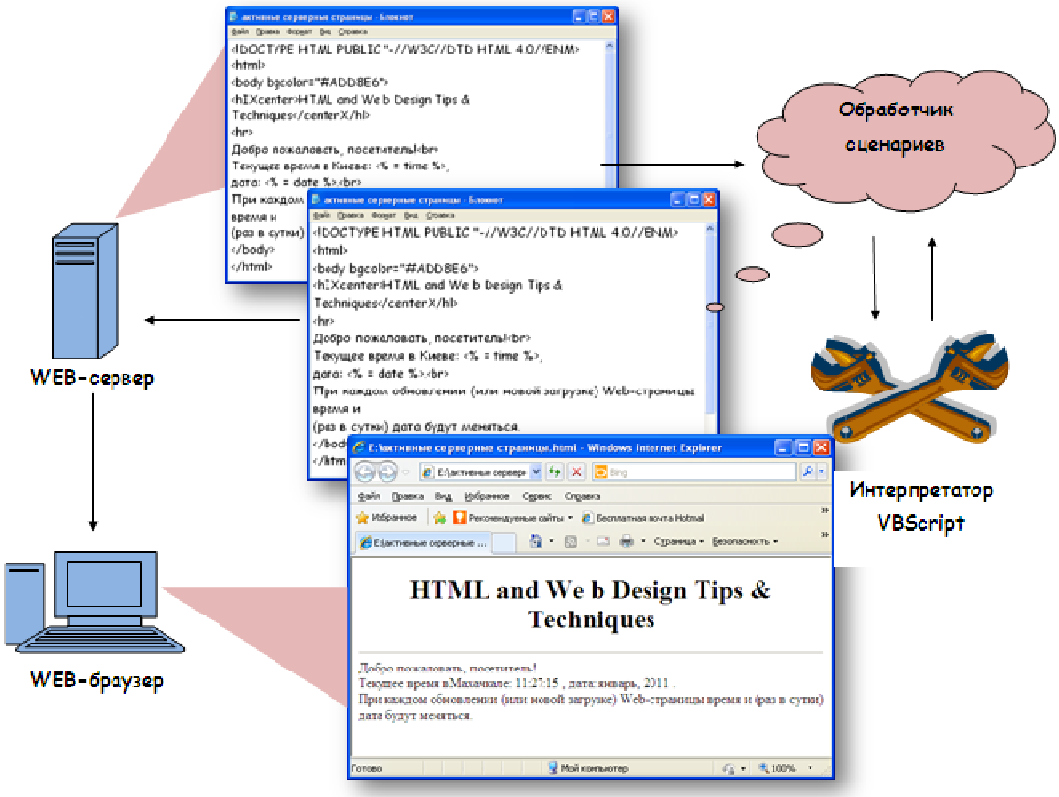


Рис. 1. Обработка asp-страницы

Если бы активные серверные страницы позволяли отображать на Web-странице только изменяющиеся дату и время, то вряд ли стоило бы создавать для этого целую технологию. К счастью, использование сценария на стороне сервера, отображающего на странице дату и время — это лишь простейший пример того, что можно сделать с помощью ASP. С помощью соответствующего сценария можно подстраивать содержимое Web-страницы в зависимости от того, кто посещает вашу страницу, какой у него браузер, и какие действия он предпринимал во время посещения вашей страницы (или во время сеанса). Например, предположим, что вы применяете технологию ASP для управления банковскими оплатами, суммами на счетах или счетами кредитных карточек. Сценарии, включенные в состав Web-страниц банка, генерируют HTML и текст для отображения информации о вашем счете. Если кто-то другой войдет на сайт вашего банка, он увидит данные по своему (а не вашему) счету — по тому же самому Web-адресу.

Кроме отображения динамического и настраиваемого содержимого Web-страницы, разработчики могут применять сценарии, вставленные в ASP-страницы, для обработки данных, принятых из форм. Например, предположим, что вы входите на сайт своего банка и заполняете форму для пересылки денег с одного счета на другой. После щелчка: на кнопке "Готово" ваш браузер пересылает введенную вами (т.е. принятую из формы) информацию по URL-адресу, указанному в атрибуте *action* дескриптора формы <form>. Если по этому адресу находится ASP-

страница, сервер передает Web-документ (заданный URL-адресом) и информацию, принятую из формы, обработчику ASP-сценариев. Обработчик сценариев передает указанные (или выбранные) вами в HTML-форме сумму и номера счетов интерпретатору сценариев, который, в свою очередь, выполняет операторы сценария, которые изменяют информацию о вашем счете, хранящуюся в таблицах базы данных.

Для создания ASP-страницы нужен только текстовый редактор наподобие Блокнота (Notepad), стандартно присутствующий во всех версиях Windows. Или же можно воспользоваться любым из множества средств разработки Web-страниц, таким как Visual InterDev или FrontPage компании Microsoft, Homesite компании Allaire, GoLive компании Adobe или Dreamweaver компании Macromedia.

ASP-страница — это обычный текстовый ASCII-файл, обычно с расширением . asp. Поэтому при использовании инструментов разработки Web-сайтов для создания ASP-страницы редактировать страницу можно либо той же инструментальной программой, либо любой другой, либо текстовым редактором (например, Блокнотом). Надо лишь помнить, что для обработки в качестве ASP-страницы этот документ должен иметь расширение, связывающее его с интерпретатором asp.dll (обычно это расширение .asp, поскольку администраторы сайтов связывают (назначают) файлы с расширением . asp с обрабатывающим его Приложением asp.dll).

Для опубликования ASP-страниц (чтобы ASP-страницы стали доступными в Internet или Intranet-сети предприятия) нужен Web-сервер, поддерживающий ASP. Поскольку компания Microsoft разработала технологию ASP, то неудивительно, что ее Web-сервер — Internet Information Server (MS) — поддерживает ASP.

Вопрос 2. Создание простейшей ASP-страницы.

Чтобы создать ASP-страницу, содержащую несколько операторов, которые должны быть обработаны интерпретатором VBScript, запустим свой любой текстовый редактор (например, Блокнот) и наберем следующий код:

```
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
<title>Начальный и конечный дескрипторы ASP-сценария на стороне сервера </title>
</head>
<body bgcolor=<"#ADD8E6">
<hl> <center>HTML and We b Design Tips & Techniques</center></hl>
<hr>
<% @language = VBScript %>
```

<%
Response.Write "<p>Датаи время на Web-сервере: "
Response.Write(now())
%>
</body>
</html>

В данном примере обработчик ASP-сценариев переносит в создаваемую им Web-страницу первые десять операторов HTML (начиная с объявления DOCTYPE и заканчивая дескриптором <hr>). А сценарий на стороне сервера в этом примере начинается со следующего оператора в строке:

<% @language = VBScript %>

Команда @language в этом операторе говорит обработчику сценариев, что сценарий (сценарии) внутри Web-страницы содержат операторы VBScript. Поэтому обработчик сценариев передаст операторы, заключенные между начальным и конечным дескрипторами сценария (<% .. . %>) на обработку интерпретатору VBScript. После того как интерпретатор VBScript вернет результат каждого из двух вызовов методов Response.write обработчику сценариев, тот добавит возвращенный текст в HTML-текст страницы.

Response.write — это метод (т.е. вызов функции), с помощью которого можно добавлять текст в HTML-код страницы. Первый вызов Response. write добавляет строку в кавычках, приведенную в рассматриваемом примере, а второй вызов возвращает текущие дату и время, которые обработчик сценариев передаст в Web-страницу.

Теперь сохраним свою страницу в файле с расширением .asp на Web-сервере и откроем этот HTML-документ в Web-браузере.

Обратите внимание, что созданную ASP-страницу необходимо получить с Web-сервера, поддерживающего ASP (наподобие IIS компании Microsoft). Если Web-сервер не поддерживает ASP-файлы, то он просто перешлет запрошенный документ Web-браузеру без изменений, и браузер отобразит операторы VBScript как обычное содержимое страницы. И наоборот, Web-сервер с поддержкой ASP передаст запрошенную вами ASP-страницу (ASP_Testi.asp) обработчику сценариев (также выполняющемуся на Web-сервере).

Список литературы

Алексеев А.П. Введение в Web-дизайн: учебное пособие. Издательство: СОЛОН-ПРЕСС, 2008 г., 185 с.

1. Горнаков С.Г. Осваиваем популярные системы управления сайтом (CMS). Издательство: ДМК Пресс, 2009 г., 326 с.
2. Дронов В. В подлиннике. Adobe Dreamweaver CS4 + Видеокурс (+CD). BHV-СПб, 2009 г., 832 с.
3. Дронов В. Профессиональное программирование. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов. BHV-СПб, 2011 г., 416 с.
4. Кирьянов Д.В., Кирьянова Е.Н. Adobe Flash CS3 – это просто! Создаем Web-анимацию. Издательство: БХВ-Петербург, 2008 г., 234 с.
5. Колисниченко Д.Н. Профессиональное программирование. PHP 5/6 и My SQL 6. Разработка Web-приложений. BHV-СПб, 2010 г., 560 с.
6. Хольцнер С. PHP в примерах. Бином, 2011 г., 352 с.
7. Шапошников И.В. PHP 5.1. Учебный курс. Учебное пособие. Издательство: Питер, 2007 г., 192 с.
8. Энж Э. SEO - искусство раскрутки сайтов. BHV-СПб, 2009 г., 464 с.

Обеспеченность учебно-методической литературой.

1. Раджабов К.Я. Учебно-методический комплекс по дисциплине "Web-программирование". Махачкала: ДГИНХ. 2006, 174с.