Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
ВЛАДИМИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени Александра Григорьевича и Николая Григорьевича Столетовых
(ВлГУ)

КАФЕДРА ИНФОРМАЦИОННЫХ СИСТЕМ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

Методические указания к лабораторным работам

Составитель:
к.т.н., доцент Салех Х.М.

Владимир 2015

ЛАБОРАТОРНАЯ РАБОТА № 1
Основы технологии создания гипертекстовых документов

1. Цель работы

Рассмотреть области применения Web-сайтов, изучить и на практике освоить основные элементы языка разметки гипертекстовых документов HTML, разработать проект и практически реализовать Web-сайт.

2. Основные сведения

2.1 Web-сайты и Web-страницы

Публикации в World Wide Web (Всемирной паутине) реализуются в форме Web-сайтов. Web-сайт состоит из компьютерных Web-страниц. Сайт является *интерактивным* и *мультимедийным* средством представления информации. Обычно сайт имеет титульную страницу (страницу с оглавлением), на которой имеются *гиперссылки* на его основные разделы (Web-страницы). Гиперссылки также имеются на других Web-страницах сайта, что обеспечивает возможность пользователю свободно перемещаться по сайту. Web-страницы сайта могут содержать *динамические объекты* (исполнимые модули), созданные с использованием сценариев на языках JavaScript и VBScript или элементов управления ActiveX. Расположенные на сайте управляющие элементы (например, кнопки) позволяют пользователю запускать те или иные динамические объекты.

Создание Web-сайтов реализуется с использованием языка разметки гипертекстовых документов HTML (Hyper Text Markup Language). Язык HTML используется для задания логической структуры документа (заголовки, абзацы, графические изображения и прочие объекты) и является той средой, в которой размещаются остальные компоненты Web-страницы. Технология HTML состоит в том, что в обычный текстовый документ вставляются управляющие символы – *тэги*, и в результате мы получаем Web-страницу. Для создания Web-страниц используются простейшие текстовые редакторы, которые не включают в создаваемый документ управляющие символы форматирования текста. В качестве такого редактора в Windows можно использовать стандартное приложение Блокнот.

2.2. Основные тэги

Вид Web-страницы задается тэгами, которые заключаются в угловые скобки. Тэги могут быть одиночными или парными, для которых обязательно наличие открывающего и закрывающего тегов (такая пара тэгов называется *контейнером).* Закрывающий тэг содержит прямой слэш (/) перед обозначением. Тэги могут записываться как прописными, так и строчными буквами.

HTML-код страницы помещается внутрь контейнера <HTML> </HTML>. Без этих тэгов браузер не в состоянии определить формат доку-

мента и правильно его интерпретировать. Web-страница разделяется на две логические части: заголовок и содержание.

Заголовок Web-страницы заключается в контейнер <HEAD> </HEAD> и содержит название документа и справочную информацию о странице, которая используется браузером для ее правильного отображения (например, тип кодировки, ключевые слова документа, используемые поисковыми роботами). Ключевые слова и кодировка, используемая на странице, задаются соответственно следующими тегами:

<META name="keywords" content="слово1, слово2, слово3">

<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-1251">.

Название Web-страницы содержится в контейнере <TITLE> </TITLE> и отображается в строке заголовка браузера при просмотре страницы.

Основное содержание страницы помещается в контейнер <BODY></BODY> и может включать текст, таблицы, бегущие строки, ссылки на графические изображения и звуковые файлы и так далее.

Созданную Web-страницу необходимо сохранить в виде файла. Принято сохранять титульный файл сайта, то есть тот, который первый загружается в браузер, под именем index.htm. В качестве расширения файла Web-страницы можно также использовать html. Необходимо различать имя файла index.htm, то есть имя, под которым Web-страница хранится в файловой системе, и собственно имя Web-страницы, которая высвечивается в верхней строке окна браузера и в первую очередь анализируется поисковыми системами. Имя Web-страницы должно в максимальной степени соответствовать ее содержанию. Рекомендуется создать для размещения сайта специальную папку и сохранять все файлы разрабатываемого сайта в этой папке.

Тег <BODY> может содержать атрибуты:

bgcolor="цвет" – для заливки заднего фона документа цветом. Цвет можно указывать явно (black, maroon, green, olive, navy, purple, teal, gray, silver, red, lime, yellow, blue, fuchsia, aqua, white) или через код цвета (например, #45BAB1);

background="местоположение и название рисунка" – для задания в качестве фона рисунка. Если рисунок находится в одной директории с документом, тогда достаточно указать лишь его название. Если рисунков много, то их удобнее выделить в отдельные директории и указывать их название с полным описанием пути к ним (при использовании HTML-документа только на локальном диске), например: <BODY background= "C:\html\pic\fon.gif">. Если же сайт опубликован в Интернете, то и путь надо указывать с адреса Web-страницы – URL (Universal Resource Locator

– универсальный указатель ресурсов), например: <BODY background= "http://www.abcd.ru/fon.gif">.

Таким образом, простая Web-страница будет содержать:

```
<HTML>
<HEAD>
<TITLE>Заголовок</TITLE>
<META name="keywords" content="слово1, слово2, слово3">
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; char-set=windows-1251">
</HEAD>
<BODY background=" pic\fon.gif">
Тело документа (основной HTML-текст)
</BODY>
</HTML>
```

Загрузить этот файл в окно браузера для просмотра в виде Web-страницы и в виде HTML-кода (через меню Вид).

2.3. Форматирование текста

Основным элементом текста в HTML является абзац (параграф). Абзац заключается в контейнер <P></P>. При просмотре в браузере абзацы отделяются друг от друга интервалами. Каждый абзац можно выровнять по левому краю, по правому краю или по центру. Для этого служит указываемое в начальном теге параграфа свойство align со значениями left, center, right. Например: .

Принудительный переход на новую строку выполняется с помощью тэга
. Например:

Строка1
Строка2
Строка3 .

Текст можно выделять жирным и/или курсивом, как показано в табл. 1.

Таблица 1

Запись в HTML	Отображение в браузере
Простой текст	Простой текст
<B>Выделение жирным</B>	**Выделение жирным**
<I>Выделение курсивом</I>	*Выделение курсивом*
<B><I>Выделение жирным и курсивом</I></B>	***Выделение жирным и курсивом***

С помощью контейнера и его атрибутов можно задать параметры форматирования шрифта любого фрагмента текста. Атрибут

FACE позволяет задать гарнитуру шрифта (например, FACE="Arial"), атрибут SIZE — размер шрифта (например, SIZE=”4”), атрибут COLOR — цвет шрифта (например, COLOR="blue"). Все эти атрибуты и тэги можно совмещать в любом порядке, например:

```
<p align="center">
<font color="green" size="3" face="Arial">
<b>Пример отформатированного текста</b>
</font>
```

Ещё одним элементом текста является заголовок. Заголовок заключается в тэги <H></H>. После буквы H в тэге заголовка следует указать размер заголовка – цифру от 1 до 6. Заголовок можно выровнять по левому, правому краю или по центру точно так же, как и абзац, т.е. при помощи атрибута align="", например:

```
<h1 align="center">Заголовок </h1>
```

Нередко для разделения текста используется горизонтальная линия. Её можно задать с помощью одиночного тэга <hr>. На самом деле, тэг <hr> рисует не линию, а прямоугольник, просто по умолчанию этот прямоугольник имеет нулевую высоту и длину на весь экран (если использовать его в таблице, то длина будет равна ячейке таблицы). Увидеть, что это прямоугольник можно, если изменить его высоту с помощью атрибута size="размер", где "размер" – это высота прямоугольника в пикселях. Чтобы сделать его непрозрачным, следует указать атрибут noshade. Для указания ширины прямоугольника используется атрибут width="размер", который позволяет задать ширину прямоугольника либо в пикселях (как и высоту), либо в процентах от экрана браузера (width="10%"). Помимо этого, прямоугольник тоже можно выровнять по любому краю или по центру. Например:

```
<hr size="20" width="100" align="center" noshade >
```

2.4. Таблицы

Таблицы являются важнейшим элементом HTML-документов. В основном таблицы используются не для представления табличных данных, а для формирования структуры HTML-документов. Практически всегда для построения страницы используются таблицы с невидимыми рамками. Внутри ячеек таблицы может располагаться что угодно – текст, картинки, гиперссылки, другие таблицы. Причём каждой ячейке таблицы можно задать свои свойства – заливка, выравнивание и т.п.

Таблица располагается между тэгами <table> и </table>. Каждая новая строка таблицы задаётся тэгом <tr>, а каждая новая ячейка в строке – тэгом <td>. Вот как задается простейшая таблица:

```
<table border="1">
<tr><td>Ячейка 1-1<td>Ячейка 1-2<td>Ячейка 1-3<td>
```

<tr><td>Ячейка 2-1<td>Ячейка 2-2<td>Ячейка 2-3<td>
<tr><td>Ячейка 3-1<td>Ячейка 3-2<td>Ячейка 3-3<td>
</table>

В результате в браузере получится:

Ячейка 1-1	Ячейка 1-2	Ячейка 1-3
Ячейка 2-1	Ячейка 2-2	Ячейка 2-3
Ячейка 3-1	Ячейка 3-2	Ячейка 3-3

В этом примере обратите внимание на тэг <table border="1">, в котором указан атрибут таблицы border="1". Этот атрибут задаёт толщину рамки для таблицы. Если указать нулевую толщину, то рамка будет невидимой.

Для выравнивания таблицы служит атрибут align="", который работает точно так же, как при выравнивании абзаца. Для этого используется конструкция <table align="...">. Подобным образом можно выровнять содержимое каждой ячейки <td align="...">. Данные в ячейках можно выравнивать и по вертикали с помощью атрибута valign="", в кавычках указывается, как это сделать (bottom – по нижнему краю ячейки, middle – по центру, top – по верхнему).

2.5. Списки

В HTML есть возможность создания списков. Простейший список – ненумерованный. Он заключается между тэгами <UL> и </UL>, причём каждый пункт списка задаётся тэгом <LI>:

<UL>
<LI>Пункт 1
<LI>Пункт 2
<LI>Пункт 3
</UL>

Существует возможность создания любого числа списков, вложенных в другой список. Для этого в HTML-коде стоит лишь написать внутри одного списка другой, например, так:

<UL>
<LI>Пункт 1
<LI>Пункт 2
<UL>
<LI>Пункт 2-1
<LI>Пункт 2-2

<LI>Пункт 2-3
</UL>
<LI>Пункт 3
</UL>

Также в HTML есть возможность создания списков с различной нумерацией. Нумерованные списки заключаются в тэги <OL> и </OL>. Пункты списка обозначаются <LI>. Для указания типа нумерации к тэгу <OL> добавляется атрибут Type="", где в кавычках может указываться любое из следующих значений: 1, i, I, a, A, которое и указывает на тип нумерации. Тип "1" используется по умолчанию и указывать его необязательно, т.е. тэги <OL> и <OL Type="1"> идентичны. В табл. 2 приводятся примеры всех типов нумерации.

Таблица 2

Написание в HTML	Отображение в браузере
<OL **Type="1"**> <LI>Пункт 1 <LI>Пункт 2 <LI>Пункт 3 </OL>	1. Пункт 1 2. Пункт 2 3. Пункт 3
<OL **Type="i"**> <LI>Пункт 1 <LI>Пункт 2 <LI>Пункт 3 </OL>	i. Пункт 1 ii. Пункт 2 iii. Пункт 3
<OL **Type="I"**> <LI>Пункт 1 <LI>Пункт 2 <LI>Пункт 3 </OL>	I. Пункт 1 II. Пункт 2 III. Пункт 3
<OL **Type="A"**> <LI>Пункт 1 <LI>Пункт 2 <LI>Пункт 3 </OL>	A. Пункт 1 B. Пункт 2 C. Пункт 3
<OL **Type="a"**> <LI>Пункт 1 <LI>Пункт 2 <LI>Пункт 3 </OL>	a. Пункт 1 b. Пункт 2 c. Пункт 3

2.6. Вставка изображений.

На Web-страницах могут размещаться графические файлы трех форматов – GIF, JPG и PNG. Если рисунок сохранен в другом формате, то его необходимо предварительно преобразовать в один из вышеуказанных форматов с помощью графического редактора. Для этих целей можно использовать редактор Photo Editor, входящий в пакет Microsoft Office.

Для вставки изображения используется тэг <IMG> с атрибутом SRC=”name”, значение которого указывает на место хранения файла на локальном компьютере или в Интернете. Если графический файл находится на локальном компьютере в том же каталоге, что и файл Web-страницы, то в качестве значения атрибута SRC достаточно указать только имя файла. Если файл находится в другом каталоге на данном локальном компьютере, то значением атрибута должно быть полное имя файла. Если файл находится на удаленном сервере в Интернете, то должен быть указан URL-адрес этого файла.

Расположить рисунок относительно текста различными способами позволяет атрибут ALIGN тэга <IMG>, который может принимать пять различных значений: TOP (верх), MIDDLE (середина), BOTTOM (низ), LEFT (слева) и RIGHT (справа).

Иллюстрации на Web-страницах стали неотъемлемым элементом дизайна. Однако иногда пользователи в целях экономии времени отключают в браузере загрузку графических изображений и читают только тексты. Поэтому, чтобы не терялись смысл и функциональность страницы, вместо рисунка с помощью атрибута ALT выводится текст, поясняющий, что должен был бы увидеть пользователь на рисунке. Например: <IMG SRC= "computer.gif" ALT="Компьютер" ALIGN=«right >.

2.7. Гиперссылки на Web-страницах

Первая титульная страница должна предоставлять посетителю Web-сайта возможность начать путешествие по сайту. Для этого на титульную страницу должны быть помещены гиперссылки на другие страницы сайта.

Гиперссылка состоит из двух частей: *указателя ссылки* и *адресной части ссылки.* Указатель ссылки — это то, что мы видим на Web-странице (текст или рисунок), обычно выделенный синим цветом и подчеркиванием. Активизация гиперссылки вызывает переход на другую страницу. Адресная часть гиперссылки представляет собой URL-адрес документа, на который указывает ссылка. URL-адрес может быть абсолютным и относительным. Абсолютный URL-адрес документа полностью определяет компьютер, каталог и файл, в котором располагается документ.

Абсолютный адрес документа, находящегося на локальном компьютере, будет включать в себя путь к файлу и имя файла, например: C:/Web-

сайт/filename.htm. Абсолютный адрес документа, находящегося на удаленном компьютере в Интернете, будет включать имя сервера Интернета, путь к файлу и имя файла, например: http://www.host.ru/Web-сайт/filename.htm.

Относительный URL-адрес указывает на местоположение документа относительно того, в котором находится указатель ссылки. При разработке сайта рекомендуется входящие в него Web-страницы связывать относительными ссылками. Это позволит не изменять адресную часть ссылок при перемещении Web-сайта в другой каталог локального компьютера или при его публикации в Интернете.

Для определения адреса перехода по каждой гиперссылке используется контейнер <A></A> с атрибутом HREF, значением которого является URL-адрес документа на локальном компьютере или в Интернете. Контейнер должен содержать указатель гиперссылки:

```
<A HREF="URL">Указатель гиперссылки</A>.
```

Создать на титульной странице *панель навигации* в виде абзаца, выровненного по центру, в котором указатели гиперссылок разделены пробелами ( ), можно с помощью следующего кода:

```
<P ALIGN="center">
<A HREF="URL 1">Указатель 1</A>  
<A HREF="URL 2">Указатель 2</A>  
<A HREF="URL 3">Указатель 3</A>
</P>
```

Полезно на титульной странице сайта создать ссылку на адрес электронной почты, по которому посетители могут связаться с администрацией сайта. Для этого необходимо атрибуту ссылки HREF присвоить URL-адрес электронной почты и вставить ее в контейнер <ADDRESS></ADDRESS>, который задает стиль абзаца, принятый для указания адреса:

```
<ADDRESS>
<A HREF="mailto:mailbox@provaider.ru">E-mail:
mailbox@provaider.ru</A>
</ADDRESS>
```

По щелчку мыши по ссылке на адрес электронной почты будет открываться почтовая программа Outlook Express (или другая используемая по умолчанию почтовая программа), где в строке «Кому» будет указан заданный в ссылке адрес.

2.8. Формы на Web-страницах

Для того чтобы посетители сайта могли не только просматривать Web-страницы, но и отправлять информацию администраторам сайта, на его страницах размещают формы. Формы включают в себя управляющие элементы различных типов: текстовые поля, раскрывающиеся списки,

флажки, переключатели и так далее. Вся форма заключается в контейнер <FORM></FORM>.

Для получения данных о посетителях разместим на форме два однострочных *текстовых поля* для ввода информации. Текстовые поля создаются с помощью тэга <INPUT> со значением атрибута TYPE="text". Атрибут NAME является обязательным и служит для идентификации полученной информации. Значением атрибута SIZE является число, задающее длину поля ввода в символах. Для того чтобы анкета «читалась», необходимо разделить строки с помощью тэга перевода строки
.

Создадим в Блокноте файл anketa.htm и добавим HTML-код, создающий текстовые поля для ввода данных:

```
<FORM>
Пожалуйста, введите ваше имя: <BR>
<INPUT TYPE="text"
NAME="name" SIZE=30> <BR>
E-mail:  <BR>
<INPUT TYPE="text"
NAME="e-mail" SIZE=30> <BR>
</FORM>
```

Для создания группы *переключателей* («*радиокнопок*») используется тэг <INPUT> со значением атрибута TYPE="radio". Все элементы в группе должны иметь одинаковые значения атрибута NAME. Например, NAME="group". Еще одним обязательным атрибутом является VALUE, которому присвоим значения "schoolboy", "student" и "teacher". Значение атрибута VALUE должно быть уникальным для каждой «радиокнопки», так как при ее выборе именно они передаются серверу. Атрибут CHECKED, который задает выбор кнопки по умолчанию, в данном случае не используется.

Добавим HTML-код, создающий группу переключателей для выбора одного варианта:

```
<BR>
<INPUT    TYPE="radio"
NAME="group"   VALUE="schoolboy">учащийся<BR>
<INPUT TYPE="radio"
NAME="group" VALUE="student">студент<BR>
<INPUT TYPE="radio"
NAME="group"  VALUE="teacher">преподаватель<BR>
```

Пусть, например, мы хотим узнать, какими сервисами Интернета посетитель сайта пользуется наиболее часто. Здесь из предложенного перечня он может выбрать одновременно несколько вариантов, пометив их *флажками*. Флажки создаются в тэге <INPUT> со значением атрибута TYPE="checkbox". Флажки, объединенные в группу, могут иметь одинаковые значения атрибута NAME. Например, NAME="group".

Еще одним обязательным атрибутом является VALUE, которому присвоим значения "www", "e-mail" и "ftp". Значение атрибута VALUE должно быть уникальным для каждого флажка, так как при его выборе именно они передаются серверу.

Добавим HTML-код, создающий флажки для выбора нескольких вариантов:

```
Какие из сервисов Интернета вы используете наиболее часто:
<BR>
<INPUT TYPE="checkbox" NAME="group" VALUE="www"> WWW <BR>
<INPUT TYPE="checkbox" NAME="group" VALUE= "e-mail"> e-mail <BR>
<INPUT TYPE="checkbox" NAME="group" VALUE="ftp"> FTP <BR>
```

Теперь выясним, какой из браузеров предпочитает посетитель сайта. Перечень браузеров представим в виде раскрывающегося *списка*, из которого можно выбрать только один вариант. Для реализации такого списка используется контейнер <SELECT></SELECT>, в котором каждый элемент списка определяется тэгом <OPTION>. Выбираемый по умолчанию элемент задается с помощью атрибута SELECTED:

```
Какой браузер вы предпочитаете:

<BR>
<SELECT NAME="browsers">
<OPTION SELECTED> Internet Explorer
<OPTION> Netscape Navigator
<OPTION> Opera
<OPTION> Neo Planet
</SELECT>
```

В заключение поинтересуемся, что хотел бы видеть посетитель на наших страницах, какую информацию следовало бы в них добавить. Так как мы не можем знать заранее, насколько обширным будет ответ читателя, отведем для него *текстовую область* с линейкой прокрутки. Создается такая область с помощью тэга <TEXTAREA> со следующими обязательными атрибутами: NAME, задающим имя области; ROWS, определя-

ющим число строк; COLS – число столбцов области. Добавим HTML-код, создающий текстовую область для ввода комментариев:

```
Какую еще информацию вы хотели бы видеть на нашем сайте?
<BR>
<TEXTAREA NAME="resume" ROWS=4 COLS=30>
</TEXTAREA>
<BR>
```

Отправка введенной в форму информации или очистка полей от уже введенной информации осуществляется с помощью кнопок. Кнопки создаются с помощью тэга <INPUT>. Для создания кнопки, которая отправляет информацию, атрибуту TYPE необходимо присвоить значение "submit", а атрибуту VALUE, который задает надпись на кнопке, — значение "Отправить". Для создания кнопки, которая производит очистку формы, атрибуту TYPE необходимо присвоить значение "reset", а атрибуту VALUE – значение "Очистить". Добавим HTML-код, создающий кнопки:

```
<INPUT TYPE="submit" VALUE="Отправить">
<INPUT TYPE="reset" VALUE="Очистить">
```

Заполненная форма отправляется на сервер, где обрабатывается специальной программой – CGI-скриптом, или по электронной почте автору сайта, где он уже самостоятельно обрабатывает полученные данные.

Для того чтобы при щелчке по кнопке «Отправить» данные из формы передавались на сервер и там обрабатывались, необходимо указать, куда их отправить и какая программа будет их обрабатывать. Эти параметры задаются с помощью атрибута ACTION контейнера <FORM>:

```
<FORM ACTION="URL"></FORM>
```

Атрибут ACTION определяет URL-адрес программы, расположенной на Web-сервере, которая обрабатывает полученные данные из формы. Пусть программой, которая заносит данные из формы в базу данных, будет программа bd.exe. Обычно такие программы хранятся в каталоге cgi-bin. Тогда атрибут ACTION будет выглядеть следующим образом:

```
ACTION="http:/www.mycompany.ru/cgi-bin/bd.exe"
```

2.9. Описание основных тэгов HTML

В табл. 3 приведено краткое описание основных тэгов HTML.

Таблица 3

Тег	Функция
Структура Web-страницы: <html> </html>	Указывает программе просмотра страниц, что это HTML-документ
<head> </head>	Определяет место, где помещается различная информация не отображаемая в теле документа. Здесь располагается тег названия

	документа и теги для поисковых машин
<title> </title>	Помещает название документа в оглавление программы просмотра страниц
<body> </body>	Определяет видимую часть документа
Атрибуты тела документа:	
<body background=”name”>	Устанавливает фоновое изображение
<body bgcolor=”*”>[1]	Устанавливает цвет фона документа с помощью названия цвета или его 16-ричного кода RRGGBB (пример: black или #000000 – черный; red или #FF0000 – красный; green или #00FF00 – зеленый; blue или #0000FF – синий; grey или #333333 – серый)
<body text=”*”>	Устанавливает цвет текста документа
<body link=”*”>	Устанавливает цвет гиперссылок
<body vlink=”*”>	Устанавливает цвет пройденной гиперссылки
<body alink=”*”>	Устанавливает цвет активной гиперссылки
Форматирование текста:	
<pre> </pre>	Обрамляет предварительно отформатированный текст
<h*> </h*>	Заголовок определенного размера (уровни от 1 до 6 по убыванию)
<h* align=”” *”> </h*>	Заголовок определенного размера с выравниванием относительно сторон документа (left – влево; center – по центру; right – вправо)
	Создает новый абзац
<p align=”” *”>	Абзац с выравниванием left, right, или center
 	Вставляет перевод строки
<blockquote> </blockquote>	Создает отступы с обеих сторон текста.
	Создает жирный текст
	Создает наклонный текст
<tt> </tt>	Создает текст, имитирующий стиль печатной машинки
<cite> </cite>	Используется для цитат, обычно наклонный текст
	Используется для выделения из текста слова (наклонный или жирный текст)

Продолжение табл. 3

Тег	Функция
	Используется для выделения наиболее важных частей текста (наклонный или жирный текст)
 	Устанавливает размер шрифта в пределах от 1 до 7
 	Устанавливает цвет шрифта
 	Устанавливает гарнитуру шрифта (Arial, Times New Roman и др.)
	Верхний индекс
	Нижний индекс
Графические элементы:	
<img src=”name”>	Добавляет изображение в HTML-документ
<img src=”name” align=”*”>	Выравнивает изображение относительно сторон документа, принимает значения: left, right, center; bottom, top, middle
<img src=”name” alt=”*”>	Вывод текста вместо изображения
<img src=”name” border=*>	Устанавливает толщину рамки вокруг изображения
<hr>	Добавляет горизонтальную линию
<hr size=*>	Устанавливает высоту (толщину) линии
<hr width=?>	Устанавливает ширину линии, можно указать ширину в пикселях или процентах
<hr noshade>	Создает линию без тени
<hr color=”*”>	Задает линии определенный цвет
Гиперссылки:	
 	Определяет закладку (отмечает часть текста как цель для гиперссылок в документе)
 указатель ссылки 	Гиперссылка на закладку в текущем документе
указатель ссылки 	Гиперссылка на закладку в другом документе
 указатель ссылки 	Гиперссылка на другую страницу
 	Создает гиперссылку вызова почтовой программы для написания письма автору документа
Списки:	

<dl></dl>	Создает список определений.
<dt> термин	Определяет каждый из терминов списка
<dd> определение	Описывает каждое определение
	Создает нумерованный список
	Определяет каждый элемент списка
<ol type=”*”>	Определяет тип нумерации (A, a, I, i, 1)
<ol start=*>	Определяет первый номер списка (1, 2, …)
	Создает ненумерованный список
<ul type=”*”>	Определяет тип метки (disk, circle, square)

Продолжение табл. 3

Тег	Функция
	Важный тег, используемый для форматирования больших блоков текста HTML-документа, также используется в таблицах стилей
<menu> </menu>	Создает меню
<dir> </dir>	Создает каталог
Таблицы:	
<table> </table>	Создает таблицу
<tr> </tr>	Определяет строку в таблице
<td> </td>	Определяет отдельную ячейку в таблице
<th> </th>	Определяет заголовок таблицы (нормальная ячейка с отцентрованным жирным текстом)
Атрибуты таблицы:	
<table border=*>	Задает толщину рамки таблицы
<table cellspacing=*>	Задает расстояние между ячейками таблицы
<table cellpadding=*>	Задает расстояние между содержимым ячейки и ее рамкой
<table width=*>	Устанавливает ширину таблицы в пикселах или процентах от ширины документа
<tr align=*> или <td align=*>	Устанавливает выравнивание ячеек в таблице, принимает значения: left, center или right
<tr valign=*> или <td valign=*>	Устанавливает вертикальное выравнивание для ячеек таблицы, принимает значения: top, middle или bottom
<td colspan=*>	Указывает количество столбцов, которое объединено в одной ячейке (по умолчанию=1)
<td rowspan=*>	Указывает количество строк, которое объединено в одной ячейке (по умолчанию=1)

<td nowrap>	Не позволяет программе просмотра делать перевод строки в ячейке таблицы
Кадры:	
<frameset> </frameset>	Предваряет тег <body> в документе, содержащем кадры
<frameset rows="value,value">	Определяет строки в таблице кадров, высота которых определена количеством пикселей или в процентном соотношении к высоте таблицы кадров
<frameset cols="value,value">	Определяет столбцы в таблице кадров, ширина которых определена количеством пикселей или в процентном соотношении к ширине таблицы кадров
<frame>	Определяет единичный кадр или область в таблице кадров
<noframes> </noframes>	Определяет, что будет показано в окне браузера, если он не поддерживает кадры

Продолжение табл. 3

Тег	Функция
Атрибуты кадров:	
<frame src="URL">	Определяет, какой из HTML-документов будет показан в кадре
<frame name="name">	Указывает имя кадра или области, что позволяет перенаправлять информацию в этот кадр или область из других кадров
<frame marginwidth=*>	Определяет величину отступов по левому и правому краям кадра; должно быть равно или больше 1
<frame marginheight=*>	Определяет величину отступов по верхнему и нижнему краям кадра; должно быть равно или больше 1
<frame scrolling=VALUE>	Указывает будет ли выводится линейка прокрутки в кадре; значение value может быть "yes", "no" или "auto". Значение по умолчанию для обычных документов – auto
<frame noresize>	Препятствует изменению размеров кадра
Формы [2]:	
<form> </form>	Создает формы
<select multiple name="*" size=*></select>	Создает прокручиваемое меню. Size устанавливает количество пунктов меню, которое будет показано на экране, остальные бу-

	дут доступны при использовании прокрутки
<option>	Указывает каждый отдельный элемент меню
<select name="*">	Создает ниспадающее меню
</select>	
<textarea name="*" cols=40 rows=8> </textarea>	Создает окно для ввода текста. Columns указывает ширину окна; rows – его высоту
<input type="checkbox" name="*">	Создает checkbox. За тегом следует текст
<input type="radio" name="*" value="x">	Создает radio-кнопку. За тегом следует текст
<input type=text name="foo" size=20>	Создает строку для ввода текста. Параметром Size указывается длина в символах
<input type="submit" value="*">	Создает кнопку "Принять"
<input type="image" border=0 name="*" src="*.gif">	Создает кнопку "Принять" – для этого используется изображение
<input type="reset">	Создает кнопку "Отмена"

[1] Здесь и далее звездочка обозначает одно из допустимых значений атрибута.

[2] Для форм, выполняющих какие-то функции должны быть запущены соответствующие CGI-скрипты на сервере. HTML только создает внешний интерфейс формы.

3. Задание к работе

По теме реферата (предприятие, для которого будет разрабатываться сайт) разработать Web-сайт, состоящий из нескольких страниц.

4. Содержание отчета

1. Цель работы.
2. Вариант индивидуального задания (Ваше предприятие, для которого будет разрабатываться сайт)

Название работы:

Задание:

1. Описать предприятие, для которого будет разрабатываться сайт:
2. Актуальность бизнеса
3. Описать структура вашего предприятия, для которого будет разрабатываться сайт:
4. Актуальность бизнеса (деятельность предприятия)

5. Описать структура вашего предприятия (описать действующий персонал, осуществляемые услуги)
   - Контакты
   - Указать способы осуществления бизнеса (например, способы продаж или предоставления услуг)
   - Рекламный материал
   - Обратная связь
   - Дополнительная информация
6. Описание структуры сайта
   - Название компании
   - Информация о вашем бизнесе
   - Предоставляемые услуги
   - Фотогалерея
   - Обратная связь
   - Контакты

3. Описание предметной области.
4. Структура сайта.
5. Тексты HTML-кода.
6. Выводы по работе.

5. Контрольные вопросы

1. Какие базовые информационные процессы реализуются в Web-технологии?
2. В каких прикладных информационных технологиях могут применяться Web-технологии?
3. Что является основными достоинствами HTML-документов?
4. Назовите инструментальные средства создания Web-страниц.
5. Какие объекты могут размещаться на Web-страницах?
6. Перечислите основные тэги.
7. Перечислите теги для форматирования текста.
8. Поясните создание таблиц, списков.
9. Как вставить графические элементы?
10. Что обеспечивает интерактивность сайтов?

Список рекомендуемой литературы

1. Браун, М. Использование HTML 4: [пер. с англ] / М. Браун, Р. Марк, Д. Хоникат [и др]. – 4-е изд., спец. – М. [и др.]: Вильямс, 1999. – 784 с.: ил. – ISBN 5-8459-0015-8.
2. http://htmlbook.ru/html

ЛАБОРАТОРНАЯ РАБОТА № 2
Каскадные таблицы стилей CSS

1. Цель работы:

- ознакомиться с базовым синтаксисом, основными элементами CSS - документа;
- изучить способы использования стилевой разметки;
- научиться создавать и применять таблицы стилей для управления представлением содержимого web-страниц.

2. Основные сведения

2.1 Каскадные таблицы стилей *или CSS*

Стиль - это набор параметров, задающий внешнее представление некоторого объекта в той или иной среде.

Каскадные таблицы стилей или CSS (от английского Cascading Style Sheets) являются следствием дальнейшего развития HTML и дают нам возможность перейти на следующий уровень представления информации. Таблицы стилей позволяют разделить смысловое содержимое странички и его оформление.

CSS – технология описания внешнего вида документа, написанного языком разметки. Преимущественно используется как средство оформления веб-страниц в формате HTML и XHTML, но может применяться с любыми видами документов в формате XML.

CSS при отображении страницы может быть взята из различных источников:

- Авторские стили (информация стилей, предоставляемая автором страницы) в виде:
- Внешних таблиц стилей, то есть отдельного файла .css, на который делается ссылка в документе.
- Встроенных стилей — блоков CSS внутри самого HTML-документа.
- Inline-стилей, когда в HTML-документе информация стиля для одного элемента указывается в его атрибуте style.
- Пользовательские стили
- Локальный CSS-файл, указанный пользователем в настройках браузера, переопределяющий авторские стили, и применяемый ко всем документам.
- Стиль браузера
- Стандартный стиль, используемый браузером по умолчанию для представления элементов.

Стандарт CSS определяет приоритеты, в порядке которых применяются правила стилей, если для какого-то элемента подходят несколько правил одновременно. Это называется «каскадом», в котором для правил рассчитываются приоритеты или «веса», что делает результаты предсказуемыми.

Таблица стилей состоит из набора правил. Каждое правило, в свою очередь, состоит из одного или нескольких селекторов, разделённых запятыми и блока определений. Блок определений же обрамляется фигурными скобками, и состоит из набора свойств и их значений.

До появления CSS оформление веб-страниц осуществлялось непосредственно внутри содержимого документа. Однако с появлением CSS стало возможным принципиальное разделение содержания и представления документа. За счёт этого нововведения стало возможным лёгкое применение единого стиля оформления для массы схожих документов, а также быстрое изменение этого оформления.

Преимущества CSS вёрстки:

- Несколько дизайнов страницы для разных устройств просмотра. Например, на экране дизайн будет рассчитан на большую ширину, во время печати меню не будет выводиться, а на КПК и сотовом телефоне меню будет следовать за содержимым.
- Уменьшение времени загрузки страниц сайта за счет переноса правил представления данных в отдельный CSS-файл. В этом случае браузер загружает только структуру документа и данные, хранимые на странице, а представление этих данных загружается браузером только один раз и кэшируется.
- Простота последующего изменения дизайна. Не нужно править каждую страницу, а лишь изменить CSS-файл.
- Дополнительные возможности оформления. Например, с помощью CSS-вёрстки можно сделать блок текста, который остальной текст будет обтекать (например для меню) или сделать так, чтобы меню было всегда видно при скроллинге страницы.

3. Синтаксис и принцип работы CSS

Как и любой другой язык программирования, CSS имеет строго определенный синтаксис, т.е. правила по которым создаются таблицы стилей. Запомните, в CSS в отличие от HTML нет ни элементов, ни атрибутов, ни тегов. Основной структурной единицей здесь является правило, которое определяет, как будет выглядеть тот или иной элемент в документе.

Рассмотрим структуру правила:

Рис-1 Синтаксис CSS

Показанное выше правило указывает на то, что все заголовки первого уровня в документе будут голубого цвета с размером шрифта 14 пикселей.

Как видно из рисунка выше, сначала записывается так называемый селектор, показывающий к какому html тегу (тегам) применяется то или иное свойство.

Далее, непосредственно за селектором, пишется блок объявления стилей, который обязательно заключается в фигурные скобки.

Каждое объявление в свою очередь состоит из свойства и его значения. После свойства ставится двоеточие. Правило может содержать в себе несколько объявлений. В таком случае они должны быть отделены друг от друга точкой с запятой(см.рисунок) причем после последнего объявления точку с запятой можно не ставить.

Описание каждого класса делается при помощи конструкции, подобной этой:

.small { font-size: 9pt; }

Сначала указывается имя класса - оно может быть произвольным, но желательно все-таки давать осмысленное название. Далее, в фигурных скобках перечисляются все необходимые параметры для данного класса. Параметры отделяются друг от друга точкой с запятой. Вот еще один пример, в котором используется более длинное описание:

Заметьте, что имя класса начинается с точки и таким образом опредяет универсальный класс, т.е. такой, который может быть применен к любому тегу. И делается это при помощи следующей конструкции:

Накладываем стиль на этот текст

Существуют универсальные классы и, так называемые, теговые классы:

p.small { font-size: 9pt; }

Класс, определенный таким образом, сработает только в том теге, для которого он предназначен, а для всех остальных будет проигнорирован.

Мы можем определять параметры не только для одного тега, но и сразу для нескольких. Для этого в определении стиля достаточно перечислить их через запятую:

```
p, td { font-size: 9pt; color:green;}
```

Такой прием называется группировкой, и в данном случае мы определили и для <p>, и для <td> одинаковый размер и цвет текста.

В случае переопределения существующих тегов, в описании стиля можно указывать не все параметры, а лишь те из них, которые мы хотим изменить. Все остальные параметры примут значения по умолчанию, которые для разных тегов различны.

Псевдоклассы

В CSS есть такое понятие как псевдокласс. В отличие от обычного класса, действие псевдокласса распространяется не на весь текст, к которому применен данный стиль, а лишь на его часть и возможно лишь в определенном состоянии. Чтобы было понятнее, давайте разберем эффект, при котором ссылки подчеркиваются лишь при наведении на них курсора. Эффект достаточно распространенный, и его можно наблюдать в том числе и на этом сайте. Вот фрагмент таблицы стилей, который позволяет достигать вышеописанного эффекта:

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

Верхняя строчка - это переопределение стандартного тега <a>, которое запрещает подчеркивать ссылки, а вот нижняя - это определение стиля для псевдокласса hover, который описывает стиль ссылки в момент, когда курсор находится над ней.

А вот и другой пример псевдокласса - определение буквицы вначале абзаца:

```
p:first-letter { font-size: 200\%; font-weight: bold; }
```

Заметьте, что и в том, и в другом случае действие стиля распространяется либо на определенное состояние (пользователь собирается щелкнуть по ссылке), либо на фрагмент текста (изменяется только первая буква абзаца). В этом и заключается смысл псевдоклассов.

Примечания

Как и в любом достаточно сложном языке, при создании таблицы стилей можно пользоваться комментариями. Их формат аналогичен классическому C:

```
/* Этот текст является комментарием */
```

Для небольших сайтов эта возможность Вам вряд ли пригодится, а вот при создании сложных, многоуровневых таблиц стилей комментарии могут пригодиться. Кстати, здесь будет уместно привести золотое правило - чем понятнее названа переменная (в данном случае имя класса), тем меньше комментариев необходимо.

Основные параметры CSS

Все параметры, используемые для определении стиля, условно можно разделить на несколько больших групп:

Управляющие видом шрифта (гарнитура, кегль, цвет, наклон, жирность,..) управляющие форматированием абзаца (выравнивание, интерлиньяж, расстояние между словами, отступ красной строки,..) управляющие свойствами блока (отступыслева-сверху-справа-снизу, местоположение блока на страничке, видимость блока,..) другие, которые не вписываются ни в одну из перечисленных выше групп (цвет ссылок странички, изменение внешнего вида курсора,..) Рассмотрим подробнее параметры, используемые для управления внешним видом текста и форматирования абзацев - как наиболее часто употребляемые.

Основные параметры шрифта

- font-weight: [bold|normal|number] - жирность шрифта
- font-style: [normal|italic|oblique] - наклон шрифта
- font-size: number - размер шрифта
- font-family: name - гарнитура шрифта
- color:number - цвет шрифта
- background-color: number - цвет подложки
- background: url - текстурная подложка

Псевдоклассы Ссылок

- A:active{} Таблица стилей для активных ссылок (при нажатии)
- A:link{} Таблица стилей для собственно ссылок
- A:visited{} Таблица стилей для посещённых ссылок
- A:hover {} Таблица стилей для ссылок при наведении указателя мыши

Основные параметры абзаца (и Элементов типа "Box")

- text-align: [left|right|center|justify] - выравнивание
- text-indent: number - отступ красной строки
- line-height: number -интерлиньяж
- letter-spacing: number - трекинг
- padding-left: number - отступ от текста слева
- padding-right: number - отступ от текста справа
- padding-top: number - отступ от текста сверху
- padding-bottom: number - отступ от текста снизу
- margin-left: number - отступ от границы слева
- margin-right: number- отступ от границы справа
- margin-top: number - отступ от границы сверху
- margin-bottom: number - отступ от границы снизу

Единицы измерения в CSS

В свойствах, которым требуется указание размеров, можно использовать несколько способов для их задания:

- относительный размер в процентах (%)
- относительный размер при помощи словесного описания (larger, smaller, xx-small, x-small, small, medium, large, x-large, xx-large)
- абсолютный размер в типографских единицах - размер может задаваться в пунктах (pt), пиках (pc), пикселях (px), средней шириной буквы "m" (em), средней шириной буквы "x" (ex)
- абсолютный размер в стандартных единицах длины - размер может задаваться в сантиметрах (cm), миллиметрах (mm), дюймах (in) абсолютный в пикселях (px)

Цвет и фон в CSS

Основными свойствами цвета и фона в CSS являются:

- color
- background-color
- background-image
- background-repeat
- background-attachment
- background-position
- background

Цвет для тех свойств, где это нужно, может быть определен одним из трех способов:

- при помощи названия цвета: yellow, red, green, grey,..
- шестнадцатеричным заданием цвета в формате #RRGGBB: #ff0000, #883490, #ffffff,..
- десятичным заданием составляющих цвета в формате rgb(red, green, blue): rgb(255,0,0), rgb(100,23,78),..

Свойство COLOR

Задает основной цвет(цвет переднего плана) того или иного элемента. Например, если мы хотим сделать цвет всех заголовков первого уровня красным, а цвет текста параграфов зеленым, то таблица стилей будет выглядеть так:

```
H1 {
color: red ;
}
P {
color: green ;
}
```

Свойство BACKGROUND-COLOR

Задает фоновый цвет элемента. В отличие от html, в котором фоновый цвет можно использовать только для страницы или ячейки таблицы(т.е. имеющих атрибут bgcolor) в CSS , фоновый цвет можно задавать для чего угодно: для таблиц, заголовков, параграфов, ссылок и др. Тут главное правильно определить что нам нужно.

```
BODY {
background-color : #FFEE8C ;
}
H1 {
color: red ;
background-color :blue ;
}
P {
color: green ;
}
```

Свойство BACKGROUND-IMAGE

Данное свойство используется для задания фонового изображения. В примере ниже, я укажу фоновое изображения для всей страницы, т.е. для элемента BODY.

```
BODY {
background-color : #FFEE8C ;
background-image : url(smile.png) ;
}
H1 {
color: red ;
background-color :blue ;
}
P {
color: green ;
}
```

Как видите, в качестве значения свойства, указывается путь к изображению, но немного не так как в html. В начале пишется URL а затем сразу**,** Без пробелов!!! **.** Если она находится в той же папке, то пишем просто название картинки, как в примере выше. Если допустим в подпапке img , то пишем так url(img/smile.png) .

Фоновое изображение в CSS можно задавать для любого элемента, не только для таблиц и всей страницы. Вверху, две картинки смайлика - большая и маленькая. Попробуйте сохранить их себе на компьютер и потренироваться. Большой смайлик поставьте на фон всей страницы, а маленький - на фон заголовков первого уровня.

Свойство BACKGROUND-REPEAT

Фоновое изображение по-умолчанию повторяется начиная с верхнего левого угла, как по вертикали , так и по горизонтали , пока не заполнит весь экран. С помощью свойства background-repeat мы можем контролировать эти повторения.

Это свойство может принимать четыре значения:

Background-repeat: repeat-x ;	повторение по горизонтали
Background-repeat: repeat-y ;	повторение по вертикали
Background-repeat: repeat ;	по вертикали и по горизонтали**(значение по-умолчанию)**
Background-repeat: no-repeat ;	не повторяется

Это очень полезное свойство, и аналогов ему в html нет.

Пример записи стиля:

```
BODY {
background-image : url(smile.png) ;
background-repeat: repeat-x;
}
```

Если вы вообще не укажете это свойство, то будет использоваться его значение по умолчанию, т.е. фоновое изображение будет повторяться как по вертикали, так и по горизонтали.

Свойство BACKGROUND-ATTACHMENT

При наличии фонового рисунка, это свойство устанавливает, будет ли фоновое изображение прокручиваться с содержимым страницы, или будет заблокировано, т.е. неподвижно. Может принимать два значения: **SCROLL** - фон прокручивается вместе с содержимым; **FIXED** - фон строго зафиксирован.

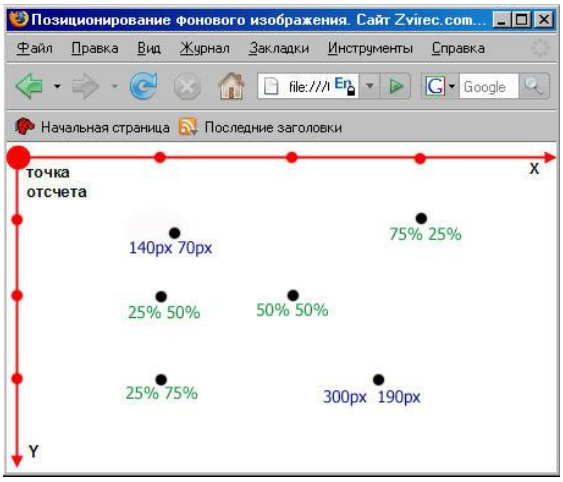
```
BODY {
background-image : url(smile.png) ;
```

```
background-repeat: no-repeat;
background-attachment: scroll ;
}
        BODY {
background-image : url(smile.png) ;
background-repeat: no-repeat;
background-attachment: fixed ;
}
```

Значение по умолчанию - **scroll** , т.е. если вообще не писать это свойство, то фон будет прокручиваться вместе с содержимым, как в первом примере.

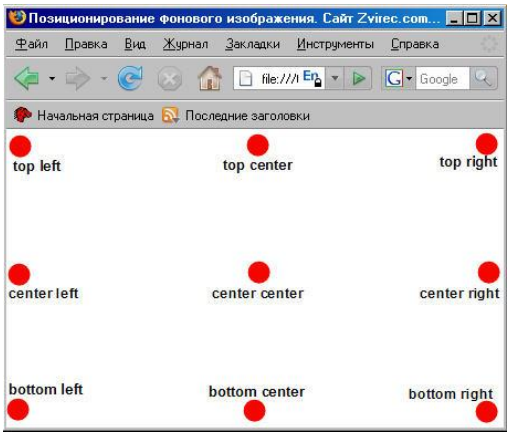
Свойство BACKGROUND-POSITION

Задает позицию фонового изображения. Значения можно задавать в процентах, в единицах длины и при помощи ключевых слов. Отсчет как обычно ведется из левого верхнего угла браузера, где и располагается фоновое изображение по умолчанию. На рисунке приведены примеры позиционирования (точка это типа изображение, а то что под ней - приблизительные координаты)



В начале указываем координату по горизонтали (по оси X), затем через пробел координату по вертикали (по оси Y). Координату можно задавать в процентах от ширины окна браузера, также в пикселях. Можно задавать и в сантиметрах, но не советую.

Также положение можно задавать специальными словами: **left** - лево, **right** - право, **center** -центр, **top** - верх, **bottom** - низ . Смотрите рисунок:



Примеры стилей:

```
BODY {
background-image : url(smile.png) ;
background-repeat: no-repeat;
background-position: top right;
}
BODY {
background-image : url(smile.png) ;
background-repeat: no-repeat;
background-position: 300px 500px ;
}
BODY {
background-image : url(smile.png) ;
background-repeat: no-repeat;
background-position: 75% 25%;
}
```

Сокращенная форма записи - BACKGROUND

Свойство BACKGROUND служит для сокращенной записи всех выше рассмотренных свойств.

Порядок свойств этого элемента таков:

background-color_background-image_background-repeat_background-attachment_background-position

Т.е. просто записывается пять значений свойств через пробел.

```
BODY {
background-color:#ffee8c ;
background-image : url(smile.png) ;
background-repeat: no-repeat;
background-attachment: fixed;
```

```
background-position: top right;
}
```

Можно записать одной строчкой:

```
BODY {
background: #ffee8c url(smile.png) no-repeat fixed top right ;
}
```

Шрифты в CSS

Посмотрим как выглядят шрифты в CSS, а именно рассмотрим основные свойства шрифтов :семейство, вес, стиль, вариант, размер, также узнаем по какому принципу браузер выбирает нужный шрифт.

Так вот за шрифты в CSS отвечают следующие свойства:

- font-family
- font-style
- font-variant
- font-weight
- font-size
- font

Свойство FONT-FAMILY

Данное свойство определяет гарнитуру шрифта. Вообще FONT-FAMILY с англ. означает семейство шрифта. Дело в том, что шрифты по тем или иным признакам, объединяются в семейства. Я рассмотрю три основных семейства:
Serif - шрифты с характерными "засечками", наиболее яркий представитель "Times New Roman"; Sans-serif - шрифты рубленые, без засечек, например Arial или Verdana ; Monospace - моноширинные шрифты (с одинаковым расстоянием между символами, наподобие печатной машинки), такие как "Courier New";

Свойство FONT-STYLE

Данное свойство задает стиль шрифта. Может принимать три значения:

- **normal** - обычный;
- **italic** - курсивный;
- **oblique** - наклонный;

У вас может возникнуть вопрос: "чем отличается курсивный от наклонного?", - дело в том, что значение **italic** означает использование встроенного в шрифт курсивного начертания. Ведь почти каждый шрифт включает в себя все символы и буквы в нормальном исполнении, в полужирном исполнении и в курсивном.

А значение **oblique** - это искусственно созданный курсив, т.е. созданный наклоном стандартных букв на определенный угол.

```
h1{
font-family: verdana, arial, sans-serif;
font-style:normal;
}
h2{
font-family: verdana, arial, sans-serif;
font-style:italic ;
}
h3{
font-family: verdana, arial, sans-serif;
font-style:oblique;
}
```

Свойство FONT-VARIANT

Это свойство используется для выбора варианта написания букв нижнего регистра. Может принимать два значения:

normal - значение по умолчанию, текст отображается обычным образом.
small-caps - буквы нижнего регистра отображаются как уменьшенные заглавные.

Вид текста умолчанию, т.е. если font-variant: normal;

ВИД ТЕКСТА ЕСЛИ FONT-VARIANT: SMALL-CAPS;

Т.е. как видите большие(заглавные буквы) остаются без изменений, а маленькие - представляют собой полную копию заглавных букв, только слегка меньшего размера.

По умолчанию (т.е. если вообще не записывать это свойство) текст будет отображаться обычным начертанием.

Пример записи стиля:

```
h1{
font-family: verdana, arial, sans-serif;
font-variant:small-caps;
}
h2{font-family: verdana, arial, sans-serif;}
```

Свойство FONT-WEIGHT

Это свойство определяет насыщенность шрифта, т.е. с его помощью можно сделать шрифт жирным. Основные значения: **normal** - обычный и **bold** - жирный. Некоторые браузеры поддерживают числовые значения: 100, 200,300,400,500,600,700,800,900. Для справки: 400 равносильно normal, а 700 - bold . Но я не советую вам пользоваться числами!

```
P {font-family: arial, verdana, sans-serif;}
DIV {
font-family: arial, verdana, sans-serif;
font-weight: bold;
}
```

Свойство FONT-SIZE

С его помощью можно регулировать размер шрифта. В качестве единиц измерения лучше всего использовать пикселы, т.к. это универсальный способ и во всех браузерах вы увидите одинаковый результат, а это самое главное.

```
h1 {
font-family: arial, verdana, sans-serif;
font-size: 18px;
}
h2{
font-family: arial, verdana, sans-serif;
font-size: 36px;
color: red;
}
```

Как видите с помощью этого свойства можно переопределять вид заголовков, что может быть полезно например при продвижении в поиско-

вых системах. Ведь тексту, заключенному в заголовок поисковые системы придают большее значение, чем обычному.

Сокращенная запись. Свойство FONT

Все, перечисленные выше, свойства можно записать в краткой форме. Это помогает экономить время и делать код стилей более легким.

При этом нужно записывать значения всех свойств через пробел, в такой последовательности:

font-style_ font-variant_font-weight_font-size_font-family

```
P{
font-style: italic;
font-variant: normal ;
font-weight: bold;
font-size: 30px;
font-family: arial, sans-serif;
}
```

А теперь то же самое в краткой форме:

```
P{ font : italic normal bold 30px arial,sans-serif; }
```

Еще раз обратите внимание, на то, что все значения указаны в нужной последовательности, и ЧЕРЕЗ ПРОБЕЛ. Если какое-либо свойство не указать, ему присвоится значение по умолчанию.

Текст в CSS

Основные свойства CSS отвечающие за форматирование текста.

- text-align
- text-decoration
- text-indent
- text-transform
- letter-spacing
- word-spacing

Свойство TEXT-ALIGN

Свойство выравнивания текста, аналогичное атрибуту ALIGN используемому в html. Может принимать четыре значения:

left - выравнивание по левому краю(значение по умолчанию);

right - выравнивание по правому краю;

center - выравнивание по центру;

justify - выравнивание по ширине(по правому и левому краям одновременно).

```
h1{ text-align:center;}
h2 {text-align:left;}
h3 {text-align:right;}
p {text-align:justify;}
```

Свойство TEXT- DECORATION

Позволяет оформлять текст определенным образом. Рассмотрим четыре основных значения данного свойства:

none - значение по умолчанию. Дополнительного оформления не происходит;

underline - текст подчеркивается снизу;

overline - текст надчеркивается сверху;

line-through - текст перечеркивается;

```
h1 { text-align:center;
text-decoration:underline;}
h2 {text-align:center;
text-decoration:overline;}
h3 {text-align:center;
text-decoration:line-through;}
```

Свойство TEXT-INDENT

Это свойство пригодится нам для создания отступов первой строки, другими словами абзацев. Значение лучше задавать в пикселях, это универсальный способ.

```
h1{text-align:center;}
p {
text-indent: 40px;
}
```

Однако можно задать и в процентах от общей длины базовой строки(строки без отступа)

```
h1{text-align:center;}
p {
text-indent: 20%;
}
```

Как видите, все очень просто, а в html это сделать гораздо сложнее.

Свойство TEXT-TRANSFORM

С помощью этого свойства, можно видоизменять текст, а именно менять большие буквы на маленькие или наоборот. Может иметь такие значения:

capitalize - меняет регистр первых букв каждого слова так, чтобы они были заглавными. **Например:"создайте сайт сейчас" станет "Создайте Сайт Сейчас".**

Частенько это свойство используется при написании рекламных текстов, это больше привлекает внимания!

uppercase - делает из всех букв заглавные. **Например: "текст в css " станет " ТЕКСТ В CSS "**

lowercase - делает из всех букв маленькие. **Например: "TRANSFORM" станет "transform".**

none - не производит смены регистра; это значение используется по умолчанию.

Свойство LETTER-SPACING

С помощью этого свойства можно изменять расстояние между буквами. Значение лучше указывать в пикселях.

```
h1 { letter-spacing: 10px;}
p{ letter-spacing :4px; }
```

В примере выше для заголовков установлен интервал между буквами в 10 px, а для параграфов в 4px;

Свойство WORD-SPACING

Позволяет изменять расстояние между словами. Значение также лучше задавать в пикселах.

```
h1 { word-spacing: 20px;}
p{ word-spacing :10px; }
```

Примеры описания таблицы стилей:

```
.epigraph {
```

```
    font-size: 12pt;
    font-style: italic;
    text-align: right;
    color: rgb(127,127,0);}
p.big {
    font-size: 16px;
    font-weight: bold;
    color: #ff0000;}
.menu {
    font-weight: bold;
    font-size: 9pt;
    font-family: arial, helvetica, sans-serif;}
a:hover {
    color: #b63a3a;
    text-decoration: none}
```

Подключение таблиц стилей

Для осуществления этой задачи мы можем воспользоваться одним из 3-х предлагаемых методов:

- внешний файл,
- inline-описание,
- описание в секции заголовка.

Inline-описание или описания, встроенное в тег:

```
<p style="color:red; text-align:center;">
Этот текст переопределен стилем
</p>
```

При помощи дополнительного атрибута style мы можем определить нужные нам стилевые параметры в любом теге. Это самый легкий способ, и действует он в пределах лишь одного тега. Но представьте, насколько вырастет размер файла, и насколько неудобно будет его исправлять, если мы будем указывать стиль у каждого тега. Этот способ не слишком отличается, к примеру, от прямого описания внешнего вида при помощи тега <font>.

Описание в секции заголовка.

Его действие распространяется на всю страничку. Определение стилей происходит при помощи классов, которые представляют собой списки с определением всех необходимых параметров оформления.

При использовании этого метода описание стилей необходимо разместить в секции заголовка:

```
<head> ....
<style type="text/css">
```

```
<!--
.header {
text-align :center;
font-size : 27pt;}
.red {color : red; }
-->
</style>
</head>
```

Теперь эти стили можно применять в любом месте html-кода. Для этого используется следующая конструкция:

```
<p class=header>Этот текст написан стилем header<p>
<p class=red>Этот текст написан красным цветом<p>
```

Как видите, все не так уж сложно. Главное понять основные принципы. Кроме определения новых классов мы также имеем возможность переопределять стандартные теги. Например, тег :

```
<style type="text/css">
<!-- p { text-align : center; font-size :12pt;}
--> </style>
```

Теперь весь текст, заключенный в теги , будет выглядеть так, как определено данным стилем. Это очень удобно и позволяет легко адаптировать уже существующие странички к использованию стилей. Кроме того, это несколько уменьшает объем файла за счет отсутствия лишних атрибутов class.

Вынесение описания стилей во внешний файл.

Диапазон его воздействия простирается на все файлы, в которые включено описание. Это способ наиболее соответствует концепции HTML 4.0. В случае, если нам потребуется изменить внешний вид сайта, то будет достаточно скорректировать лишь один этот файл. Сравните его с предыдущими способами. В одном из них придется менять описание на каждой страничке, а в другом даже более того - около каждого тега, что, разумеется, совершенно не вдохновляет.

Каким же образом производится внедрение внешнего файла? Для начала создается стилевой файл с описанием всех нужных нам классов (mystyle.css):

```
.header { text-align : center; font-size : 27pt;}
.red { color :red; }
p { text-align : center; font-size : 12pt;}
```

А потом ссылка на него внедряется в документ при помощи тега <link>:

```
<head> .... <link rel="stylesheet" type="text/css"
href="css/mystyle.css" title="MyStyleSheet"> .... </head>
```

Это самый удобный способ, и для основной таблицы стилей рекомендуется пользоваться именно им.

3. Задание к работе

1. Выполнить оформление сайта (первая лабораторная работа) при помощи CSS стилей.
2. Создать файл style.css для хранения таблицы стилей сайта.
3. Подключить файл таблицы стилей ко всем страницам сайта.
4. Просмотреть результат в браузере.
5. Сделать выводы по работе.
6. Создайте стилевой файл содержащий оформление web-сайта(ЛР1).

Обязательно, определите оформления следующих элементов:

- *Заголовки 1-4 уровней;*
- *Пункты меню 1-4 уровней;*
- *Гиперссылки.*
- *Таблицы и ячейки таблиц;*
- *Основной текст;*
- *Цвет текста;*
- *Картинки;*
- *Ссылки;*
- *Размер текста.*

Примеры описания таблицы стилей:

```
.epigraph {
   font-size: 12pt;
   font-style: italic;
   text-align: right;
   color: rgb(127,127,0);
}
 p.big {
   font-size: 16px;
   font-weight: bold;
   color: #ff0000;
}
 .menu {
   font-weight: bold;
   font-size: 9pt;
   font-family: arial, helvetica, sans-serif;
}
 a:hover {
   color: #b63a3a;
```

```
    text-decoration: none;
}
```

4. Содержание отчета

1. Цель работы.
2. Вариант индивидуального задания.
3. Описание предметной области.
4. Структура сайта.
5. Тексты CSS- документа.
6. Выводы по работе.

5. Контрольные вопросы

1. Использование внешних таблиц стилей CSS .
2. Использование внутренних таблиц стилей CSS.
3. Использование локальных таблиц стилей CSS.
4. Преимущества и недостатки внешних, внутренних, локальных таблиц стилей CSS.
5. CSS инструкции управления цветом и фоном объектов.
6. CSS инструкции управления полями и отступами объектов.
7. CSS инструкции управления шрифтами и текстом.
8. Использование классов в CSS.
9. Использование псевдоклассов в CSS.

Список рекомендуемой литературы

1. Соколов С. А. HTML и CSS в примерах, типовых решениях и задачах. Профессиональная работа. — М.: Вильямс, 2007. — С. 416. — ISBN 978-5-8459-1192-6
2. Фримен Эрик, Фримен Элизабет. Изучаем HTML, XHTML и CSS = Head First HTML with CSS & XHTML. — 1-е изд. — М.: «Питер», 2010. — С. 656. — ISBN 978-5-49807-113-8
3. Эрик А. Мейер. CSS-каскадные таблицы стилей: подробное руководство = Cascading Style Sheets: The definitive Guide. — М.: Символ, 2006. — 576 с. — ISBN 5-93286-075-8
4. http://www.w3.org/Style/CSS/
5. http://htmlbook.ru/html

Лабораторная работа № 3
JavaScript

1. Цель работы

Целью работы является приобретение навыков создания и тестирования (исследования) сценариев на языке JavaScript. Рассматриваются проблемы добавления динамики в документы, а также обеспечения правильной реакции на действия пользователя, позволяющей в удобном для восприятия виде предоставить необходимый минимум информации о возможностях взаимодействия.

2. Основные сведения

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

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

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

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

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

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

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

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

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

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

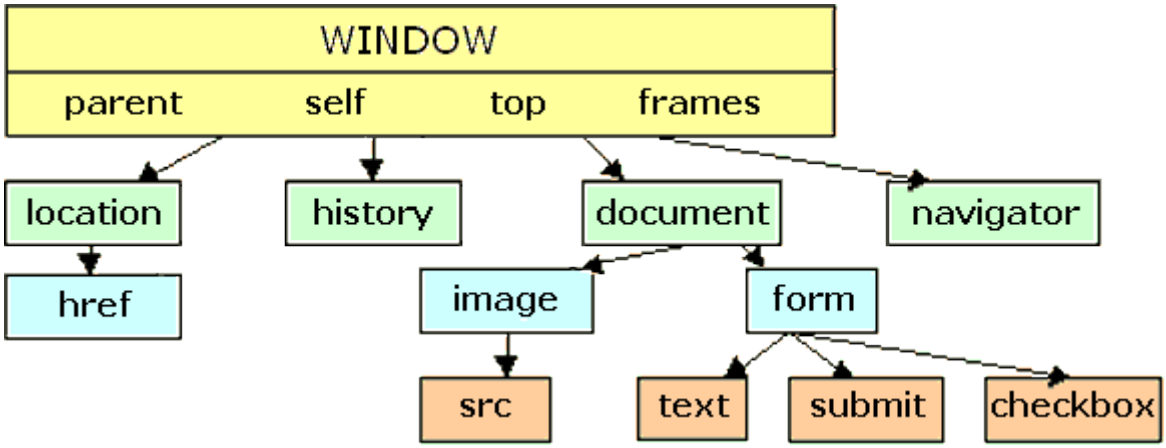
В результате при просмотре данной страницы в браузере появится текст: "Введите свое имя и нажмите кнопку".

Основные понятия:

OBJECT (объект) - это то, с чем производится действие, событие. Это может быть документ, открываемый в окне браузера или само окно браузера, или какая-то часть документа, теги. Объект должен иметь уникальное имя (ID), чтобы к нему можно было обратиться.
В нашем случае объектом является документ HTML и к нему можно просто обратиться по имени: document.

Иерархия объектов в JavaScript

В языке JavaScript определены объекты, которые называются объектами браузера. Каждый объект соответствует некоторому элементу Web- страницы: окну, документу, изображению, ссылке и т.п. Управлять частями документа можно с помощью методов браузера. Объекты браузера имеют иерархическую структуру. На самом верхнем уровне располагается объект window. Он является родителем остальных объектов. Объект window представляет окно браузера. Свойство window.status можно использовать для изменения вида строки состояния. Для отображения диалоговых сообщений объект window имеет три метода.



Иерархия объектов в JavaScript

Каждый объект обладает своими методами.
METHOD (метод объекта) - это действия, которые можно выполнять над объектом такого типа, или которые сам объект может выполнять.
Синтаксис кода: между именем объект и методом обязательно ставят разделительный оператор точка, после метода в скобках параметры метода.

Объект	.	Метод	("параметры метода")

Параметры метода относятся к типу данных - строки символов. Строки символов нужно обязательно взять в кавычки либо в одинарные, либо в двойные.
Каждый объект обладает своими свойствами.

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

Объект	.	свойство объекта	= "новое значение свойства "

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

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

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

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

СИСТЕМА СОБЫТИЙ ЯЗЫКА JAVASCRIPT

Разумеется, нас будет интересовать возможность изменения свойства при возникновении какого-либо события.
EVENT (событие) - это все, что случилось: открытие окна, загрузка в него документа, клик клавишей мышки или просто перемещение курсора по экрану, нажатие клавиши на клавиатуре - это все события, и они могут инициировать запуск больших и маленьких программ.

Использование языка JavaScript при обработке событий значительно расширило возможности языка HTML. В элементы HTML, определяющие гиперсвязи и компоненты формы, добавлены необходимые атрибуты. Чаще всего сценарии создаются для контроля реакции системы на любые действия пользователя, от движения курсора мыши по экрану до обработки информации, вводимой в поля форм. Возможности управления элементами форм обеспечиваются главным образом за счет функций обработки событий, которые задаются для всех элементов документа, в частности, для формы. Форма представляет собой контейнер, содержащий поля ввода, области текста, списки и кнопки. Для каждого из этих элементов определяются программы обработки событий, что существенно повышает степень интерактивности документа. События делятся на несколько категорий:

- события, связанные с документами в целом – загрузка и выгрузка;
- события, связанные с гиперссылками – активизация гиперссылки;
- события, связанные с формой – щелчки мыши на кнопках (button), группах кнопок выбора варианта (radiobutton), переключателях (checkbox), кнопках передачи данных и восстановления исходных значений элементов, получение и потеря фокуса ввода, а также изменение содержимого полей ввода, областей текста и списков, выделение текста в полях ввода и областях текста;
- события, связанные с рисунками – загрузка рисунка, ошибка загрузки рисунка, прерывание загрузки рисунка;
- события, связанные с клавиатурой – нажатие, давление, отпускание любой клавиши компьютера;

- события, связанные с мышью – помещение указателя мыши на гиперссылку и активизация гиперссылки, движение курсора по экрану, щелчки и двойные щелчки кнопкой мыши.

События, связанные с документами, возникают при загрузке и выгрузке документа, в то время как события гиперсвязей возникают при их активизации или при помещении на них указателя мыши. Очень популярно использование программ на языке JavaScript для обработки событий форм. События, связанные с рисунками, позволяют выполнять ответные действия как в процессе загрузки рисунка, так и при возникновении ошибок. Рисунки теперь тоже представляются как объекты с программными свойствами, которые можно описать массивами. Например, выражение document.images[0].src соответствует атрибуту *src* (адрес URI) в открывающем теге первого элемента <IMG> текущего документа.

Обработчики событий конкретного объекта задаются в HTML-теге, определяющем этот объект. Например, обработчик события, связанного с рисунком, задается в теге <IMG>, обработчик события гиперсвязи – в теге <A> и т.д. В целях «перехвата» события программируют функции-обработчики событий. Ими могут оказаться достаточно объемные коды, или только группы из одного или нескольких операторов, разделенных точкой с запятой (;). В таблице 1 перечислены имена большинства событий, используемых в языке разметки гипертекстов, и условия их возникновения. В среднем столбце таблицы сосредоточены названия атрибутов, обеспечивающих правильную реакцию на события.

Таблица 1

Имя события	Атрибут	Условие возникновения события
Blur	onBlur	Потеря фокуса ввода элементом формы
Change	onChange	Изменение содержимого поля ввода или области текста, выбор нового элемента списка
Click	onClick	Щелчок мыши на элементе HTML
Focus	onFocus	Получение фокуса ввода элементом формы
Load	onLoad	Завершение загрузки документа
MouseOver	onMouseOver	Помещение указателя мыши на элемент
MouseOut	onMouseOut	«Уход» указателя мыши из контура элемента
KeyPress	onKeyPress	Нажатие клавиши на клавиатуре компьютера
Select	onSelect	Выделение текста в поле ввода или во фрагменте текста
Submit	onSubmit	Передача данных формы
Unload	onUnLoad	Выгрузка текущего документа (начало загрузки в браузер нового документа)

Стандартные события в HTML

имя события	**происходит**
ondblclick	при двойном щелчке кнопки мыши на элементе

onmousedown	при нажатии кнопки мыши на элементе
onmouseup	при отпускании кнопки мыши на элементе
onmouseover	при попадании курсора мыши на элемент
onmousemove	при движении курсора мыши по элементу
onmouseout	при попадании курсора мыши за пределы элемента
onkeypress	при нажатии и отпускании клавиши на элементе
onkeydown	при нажатии клавиши на элементе
onkeyup	при отпускании клавиши на элементе

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

Код выглядит следующим образом:

Наведи мышь на этот текст ....

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

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

МОЖНО:

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

onmouseover='document.bgColor ="red" '

Но НЕЛЬЗЯ:

onmouseover="document.bgColor ="red" "

onmouseover='document.bgColor ='red' '

onmouseover="document.bgColor ='red" '

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

Код данного примера (ВАЖНО! Код должен быть записан в одну строчку)

 Этот абзац меняет цвет при наведении на него мышкой с синего на красный!

Разберем код.

1. style="color:blue" определяется стиль текста в данном абзаце
2. onmouseover= событие которое может произойти с этим абзацем, в кавычках надо указать что при этом делать
3. this.style.color='red' изменить стиль абзаца: цвет текста на красный:

- слово this используется для доступа к элементу (объекту), вызвавшему событие (к данному абзацу),
- через точку указывается его свойство style, дающее доступ к стилям,
- еще через точку указывается конкретное свойство, значение которого мы хотим изменить (color - цвет текста)
- затем идет знак присвоить =,
- и затем значение свойства "цвет текста" - красный ('red').

Рассмотрим еще один пример. Изменение цвета фона текста:
<p style="background-color:blue" onmouseover="this.style.backgroundColor='red' "
onmouseout="this.style.backgroundColor='blue'">Цвет фона текста меняется на красный при наведении мышкой на него!</p>

Разберем код.

1. style="background-color:blue" определяется стиль текста в данном абзаце (цвет фона)
2. onmouseover= (навести мышь) и onmouseout= (увести мышь) события которые могут произойти с этим абзацем, в кавычках надо указать что при этом делать
3. this.style.backgroundColor='red" изменить стиль абзаца: цвет фона на красный.

Допустим необходимо одинаковым образом изменять свойства нескольких однотипных объектов. Например, есть несколько ячеек таблицы, в которых необходимо поменять цвет фона при наведении мышкой.

Главная	Лекции	Лабы	Ссылки	Об авторе

Можно написать следующий код для каждой ячейки:

```
<td style="background-color:mistyrose" onmouseover="this.style.backgroundColor='white' " onmouseout="this.style.backgroundColor='mistyrose'">Главная</td>
```

Однако так как действия одинаковы можно написать подпрограмму-функцию. Функция написанная один раз может вызываться и выполняться многократно. Функции сокрашают код и упрошают процесс отладки (проверки и исправления кода).

Скрипт с функцией может находиться в разделе <head> между тегами <script> </script> и должен иметь следующий синтаксис:

```
function имя_функции (список аргументов функции)
{
тело функции
}
```

Таким образом, в разделе <head> мы размещаем функцию

```
<script type="text/javascript">
function doit(obj, zvet)
{
obj.style.backgroundColor=zvet
}
</script>
```

Имя данной функции doit, но можно выбрать и другое. В качестве аргументов в данном случае передаются объект, свойство которого нужно изменить и цвет, который нужно присвоить свойству "цвет фона" объекта. В теле функции оператор присваивания:obj.style.backgroundColor=zvet

Данная функция вызывается следующим образом:

```
onmouseover="doit(this, 'white')"
```

указывается событие и после знака равенства в кавычках имя функции, которая должна выполниться если данное событие наступит. После имени функции в скобках указывают передаваемые аргументы объект и цвет фона объекта.

Таким образом, запись кода получается следующая

```
<table class="navig">
<tr>
<td onmouseover="doit(this, 'white')" onmouseout="doit(this, 'mistyrose')">Главная</td>
<td onmouseover="doit(this, 'white')" onmouseout="doit(this, 'mistyrose')">Лекции</td>
....
</tr>
</table>
```

Где свойства таблицы определяются с помощью таблицы стилей (определяется класс navig)

```
<style type="text/css">
.navig {background-color:mistyrose; text-align:center; width:100%; height: 2em;}
</style>
```

Форма, кнопка, текстовое поле

Это пример использования формы, которая содержит такие элементы как текстовое поле и кнопку. Формы предназначены для создания интерактивных страниц, т.е. страниц, позволяющих посетителю сайта ввести некую информацию. Как, например, в данном случае свое имя.

Формы размещаются в web-документе с помощью тега <form> </form>. Ввести элементы формы и определить их свойства можно с помощью тега <input>. В общем виде код для ввода элемента формы выглядит следующим образом:

```
<form id="имя формы " name="имя формы ">
<input type="тип элемента" id="имя-индентификатор элемента " name="имя элемен-
та" value="надпись" />
</form>
```

Ключевое слово type определяет тип объекта (text - текстовое поле, button - кнопка и т.п.),
ключевое слово id задает имя-идентификатор объекта формы, также как и ключевое слово name;
value - текст, отображаемый в (на) объекте (одно из свойств объекта).

Обратите внимание, что объект форма сам обладает параметрами id и name.

Параметры id и name идентичны, но некоторые браузеры воспринимают только id, другие только name, поэтому рекомендуется оставлять оба этих параметра с одинаковыми значениями.

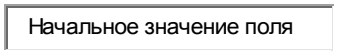
Виды элементов форм:

1) текстовое поле

Однострочное текстовое поле. Используется для ввода простейшей информации: имена, адреса и пр. Выглядит так:

<input type="text" id="Text1" size="30" value="Начальное значение поля" / >

Результат - на web-странице появляется следующий объект:



2) кнопка

Кнопка - это область окна, которая реагирует на щелчки мыши. Элемент кнопка вводится следующим образом:

```
<input type="button" id="button1" value="Нажми меня!" />
```

Данным кодом мы вводим кнопку с надписью "Нажми меня!", однако при ее нажатие ничего не произойдет, так как любую желаемую реакцию интерфейса нужно программировать. Т.е. нужно ввести событие нажатие на кнопку onmousedown и написать что должно при этом произойти. например измениться свойство кнопки value.

```
<input type="button" id="button2" value="Нажми меня!" onmousedown="this.value='ну нажми
еще раз...' "/>
```

В результате получим следующее:

Часто необходимо, чтобы при нажатии кнопки появлялось на экране диалоговое окно. Для этого можно использовать метод alert():

```
<input type="button" id="button3" value="Нажмите кнопку " onmousedown="alert('Добро пожаловать!')"/>
```

В примере 1 необходимо было изменить текст в текстовом поле, т.е. изменить свойство объекта текстовое поле. Для изменения свойства любого объекта необходимо записать следующее:

```
объект.свойство="новое значение свойства"
```

Но как обратиться к объекту текстовое поле? Для обращения к объекту формы служит id (идентификатор объекта). Т.е. если мы задали idобъекта - Text1, то следующая запись должна изменить его свойство - к старому тексту содержащемуся в объекте слева и справа будет добавлен текст "Здравствуйте" и "!"соответственно:

```
"Text1.value='Здравствуйте, '+ Text1.value + ' !' "
```

Но такое обращение к объекту можно использовать только в пределах одной формы, т.е. если объект текстовое поле и кнопка, содержатся внутри одного тега <form></form>! А если текстовое поле содержится в другой форме, то нужно соблюдать иерархию объектов. Т.е. сначала нужно обратится к объекту форма, а уже затем к объекту текстовое поле:

```
объект.вложенный объект.свойство вложенного объекта="новое значение свойства"
```

```
"Form1.Text1.value='Здравствуйте, '+ Form1.Text1.value + ' !' "
```

А если необходимо создать отдельную функцию для изменения свойства объекта, то оба вышеприведенных варианта будут работать только в Internet Explorer, так как только этот браузер поддерживает прямое обращение к объекту

```
Только для Internet Explorer!!!!

<script type="text/javascript">
function PR1()
{
Text1.value="Здравствуйте,"+Text1.value + "!"
}
</script>
```

Для остальных браузеров, для того чтобы обратиться к элементу, нужно воспользоваться методом getElementById() объекта document:

```
document.getElementById("Text1").value="Здравствуйте,"+document.getElementById("Text1").value + "!"
```

Другие объекты формы

элемент	описание
text	Однострочное поле ввода текста (принято по умолчанию). Атрибут value задает начальное значение текста, size — размер поля ввода в символах, а maxlength — максимально возможное количество символов в данном поле. Атрибут readonly запрещает изменение текста поля.
password	Поле ввода пароля. Единственное отличие от поля ввода текста состоит в том, что вводимые символы маскируются, обычно заменой на звездочки. Однако, введенная информация передается серверу как обычный текст, поэтому этот элемент не обеспечивает уровня безопасности, достаточного для передачи номеров кредитных карт или другой существенно важной информации.
checkbox	Флажок. Атрибут value задает значение этого элемента, когда он выбран, а атрибут checked — выбран ли первоначально.
radio	Переключатель. Атрибут value задает значение этого элемента, когда он выбран, а атрибутchecked — выбран ли первоначально.
reset	Кнопка сброса формы. Необязательный атрибут value перекрывает текст кнопки, принятый по умолчанию.
submit	Кнопка пересылки формы. Необязательный атрибут value перекрывает текст кнопки, принятый по умолчанию. Если форма содержит несколько таких кнопок, то атрибут name используется для передачи имени кнопки обработчику формы, что позволяет последнему осуществлять различные действия в зависимости от того, какая из кнопок пересылки формы была нажата.
image	Графическая кнопка пересылки формы. Атрибут src задает URI графического образа кнопки, а alt — альтернативный текст надписи на кнопке. При щелчке мышью по такой кнопке происходит пересылка формы, причем координаты щелчка мыши передаются серверу в виде name.x=x-value iname.y=y-value, где name — имя кнопки, а x-value и y-

	value — координаты в пикселях относительно верхнего левого угла образа. Атрибут usemap указывает на клиентскую карту ссылок и его значением должно быть имя закладки, заданное атрибутом name соответствующего элемента MAP. Атрибутismap указывает на серверную карту ссылок.
button	Кнопка общего вида. Атрибут value задает текст кнопки, а атрибут onclick должен задавать сценарий, вызываемый при нажатии этой кнопки.
file	Селектор файлов. Атрибут value задает начальное имя файла, но обозреватели обычно игнорируют его из соображений безопасности. Необязательный атрибут accept задает список типов файлов, разделенных запятыми, которые поддерживаются сервером обработчика формы. Этот список может использоваться для отфильтровки только допустимых файлов, но современные обозреватели обычно игнорируют этот атрибут. Форма, содержащая селектор файлов, должна иметь атрибуты method=post и enctype="multipart/form-data".

Типы данных

JavaScript может обрабатывать следующие типы данных:

тип	описание
строки символов	могут содержать любые буквы, цифры и другие символы. Строки символов обязательно нужно заключать в кавычки " " или в апострофы ' ' . Например: "Это строковая константа." '"Эй, вы, там наверху!", - пела Алла Пугачева.'
целые числа	допускается использовать не только десятичные числа, но и восьмеричные и шестнадцатеричные. Восьмеричные числа записываются с префиксом 0 (например - 03047), а десятичные - с префиксом 0x или 0X (например 0x63BF)
вещественные числа	дробная часть отделяется точкой
логический тип	данные могут принимать только значения true и false (истина и ложь)

Переменные

Для хранения исходных, промежуточных и результирующих данных, необходимых для работы программы, предназначены переменные.

Переменная - представляет собой зарезервированное место в оперативной памяти для временного хранения данных. Переменная характеризуется **именем.**

Имя переменной обязательно должно начинаться с латинской буквы, в именах можно использовать латинские буквы, цифры и символ подчеркивания. В качестве имен нельзя использовать служебные слова (например, function)

Следует помнить, что язык JavaScript чувствителен к регистру букв в именах: переменные с именами name и Name будут разными.

Переменная создается в момент ее декларации. JavaScript позволяет декларировать переменную двумя способами:

С помощью ключевого слова **var**, например, **var x**; или **var x = 21;**.

Просто присваиванием переменной значения, например **x = 21;**

JavaScript — *слаботипизированный язык*. Это означает, что в декларации переменной мы не указываем ее тип и в дальнейшем можем присваивать ей значения любых типов. Исполняющая система JavaScript сама выполняет автоматическое преобразование типов данных по мере необходимости. Например, в выражениях, соединяющих числовые и строковые значения операцией +, JavaScript преобразует числа в строки, например:

```
x = "Ответ равен " + 21 // возвращает "Ответ равен 21"
y = 21 + " - вот ответ" // возвращает "21 - вот ответ"
```

В остальных случаях JavaScript пытается преобразовать строку в число, например:

```
a = "21" - 1 // возвращает 20
b = "21" + 1 // возвращает 211
```

Область действия переменных

Область действия переменной определяется положением ее декларации в тексте программы. Переменная может быть **локальной и глобальной.**

Локальная переменная доступна только в той функции, в которой она объявлена. Например:

```
<script type="text/javascript">
function doit1()
{
var x=100
}
function doit2()
{
```

```
var y=x
}
</script>
```

Переменная x - локальная переменная, она определена в функции doit1, поэтому функции doit2 она неизвестна и значение переменной y будет неопределенно.

Глобальной переменной - называется переменная, значение которой доступно в любом месте программы. Например,

```
<script type="text/javascript">
var x=100
function doit2()
{
var y=x
}
</script>
```

Иными словами, любая переменная, декларированная вне тела всех функций, является *глобальной* и доступна всюду в тексте данной программы. Переменная, декларированная в теле функции, является *локальной* и доступна только внутри тела этой функции.

Для декларации глобальных переменных ключевое слово **var** не обязательно. Однако, оно обязательно при декларации локальных переменных.

Арифметические операции

Операции, используемые в арифметических выражениях, приведены в таблице

Операция	Название	Описание
a + b	Сложение	Возвращает сумму двух операндов.
a - b	Вычитание	Возвращает разность от вычитания правого операнда из левого.
a * b	Умножение	Возвращает произведение двух операндов.
a / b	Деление	Возвращает частное от деления левого операнда на правый.
a % b	Остаток по модулю	Возвращает целый остаток от деления левого операнда на правый. Плавающие числа перед операцией округляются до целых.
++	Инкремент	Унарная операция. Увеличивает значение переменной на 1. Если используется как префикс (++a), возвращает значение операнда после увеличения его на 1. Если используется как постфикс (a++), возвращает значение операнда перед увеличением его на 1.
--	Декремент	Унарная операция. Уменьшает значение переменной на 1. Ес-

		ли используется как префикс (--a), возвращает значение операнда после уменьшения его на 1. Если используется как постфикс (a--), возвращает значение операнда перед уменьшением его на 1.
-a	Смена знака	Унарная операция. Возвращает арифметическое отрицание операнда.

Примеры:

var i, j, k; i = 19 % 6.8; // i равно 5

k = 2; j = k++; // j равно 2, k равно 3

j = ++k; // j и k равны 4

Условная операция

Условный оператор **if…else** позволяет проверить определенное условие и, в зависимости от его истинности, выполнить ту или иную последовательность операторов. Синтаксис условного оператора следующий:

```
if (условие) {группа операторов 1} else {группа операторов 2}
```

В качестве условия можно использовать любое выражение, результат которого является логическим, т.е. true или falsh (истина или ложь). Условный оператор работает следующим образом: проверяется условие, если оно выполняется (результат true), то выполняется {группа операторов 1}; если условие не выполняется (возвращается значение falsh), то выполняется {группа операторов 2}. Например:

```
if (x>y) {window.alert('x больше y')} else {window.alert('x меньше y')}
```

Если переменная x больше переменной y, то выводится сообщение "x больше y", а если не равны, то "x меньше y".

Конструкция **else {группа операторов 2}** не является обязательной.

Если в группу операторов входит только один оператор фигурные скобки можно не ставить.

Операции сравнения

==	Равенство (равно)
!=	Не равно
!	Логическое отрицание
>=	Больше или равно

<=	Меньше или равно
>	Больше
<	Меньше (по возможности желательно воздержаться от применения этого типа)

От выражений, имеющих знак "<" следует отказываться по возможности, так, как данный символ может иметь и другое значение в HTML-документах (открытие тега)

4. Задание к работе

1. Создайте файл, соответствующий стандарту XHTML 1.1
2. Напишите скрипит, задающий свойство документа фон
3. Напишите скрипит, изменяющий свойство документа - фон при наведении курсором на какой-то текст
4. Создайте три абзаца
   - Напишите скрипит, изменяющий цвет текста первого абзаца при наведении мышкой
   - Напишите скрипит, изменяющий цвет фона текста второго абзаца при наведении мышкой
   - Напишите скрипит, изменяющий цвет фона текста третьего абзаца только при наведении мышкой
5. Создать страницу анкеты:

Создайте пустую веб-страницу. Настройте цвет фона, шрифты и другие свойства страницы, аналогичные ранее размеченным страницам сайта. Добавьте на страницу объект «форма». Поместите в форму группу кнопок выбора (radiobutton), комбинированный список с выпадающими ответами, несколько строк ввода (обязательны поля ввода адреса e-mail и номера телефона), список с возможностью выбора нескольких вариантов ответа и область ввода текста. Дайте форме и ее объектам понятные («говорящие») имена. Не следует называть форму “f”, радиокнопки “r”, “rr” или “rrr”, а область ввода текста “a”. Оформите страницу в соответствии с правилами дизайна. Пример анкеты показан на рис.1.

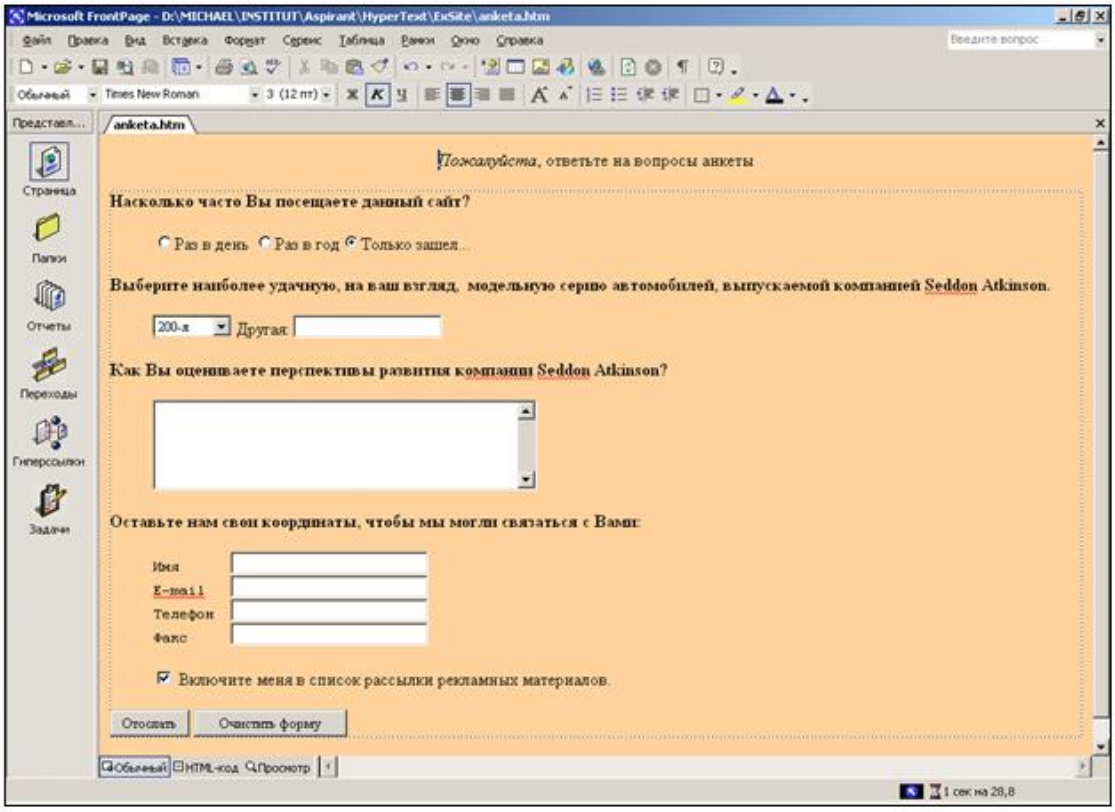


Рис. 1. Форма анкеты посетителя сайта

6. Подготовить и обосновать предложения по внесению динамики в сайт, разработанный на занятиях 6. Определить основные события, на которые должна реагировать система.
7. Сделать выводы по работе.

4. Содержание отчета

7. Цель работы.
8. Вариант индивидуального задания.
9. Тексты JavaScript кода.
10. Выводы по работе.

5. Контрольные вопросы

1. Каковы назначение и функции скриптов (сценариев) в языке разметки гипертекстов? К какой категории языка они относятся - это элементы, атрибуты, комментарии, ссылки, или они не входят ни в какие другие категории HTML?

2. В каком разделе документа HTML предпочтительно располагать элемент <SCRIPT>?

3. Каким образом прерывание от внешнего устройства или программного процесса активизирует выполнение скриптов? Постройте универсальную схему обработки прерываний.

4. Что такое событие? Основные события в JavaScript.

5. Объекты, формы, типы данных, переменные, область действия переменных.

6. Какие задачи решаются с помощью скриптов для придания динамики работе с гиперссылками? С какой целью программный код скриптов заключается в теги, воспринимаемые устаревшими браузерами как комментарии?

7. Элемент <A> (якорь) предназначен как для идентификации цели гиперссылки (атрибуты *id* и *name*), так и для указания адреса связанного ресурса (атрибут *href*). Воспримет ли система документ, в разметке которого не определены значения ни одного из этих атрибутов?

8. Могут ли программные коды сценариев составлять содержание таблиц стилей и можно ли с их помощью автоматически настраивать таблицы стилей в зависимости от конкретных характеристик компьютеров пользователей?

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Рева О. Н. JavaScript: просто как дважды два. - Москва: ЭКСМО, 2007, ISBN: 978-5-699-16032-7.
2. Прохоренок, Н. А. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера. – СПб.: БХВ-Петербург, 2008, – 640 с.

ЛАБОРАТОРНАЯ РАБОТА № 4
Основы разработки веб-приложения

1. Цель работы

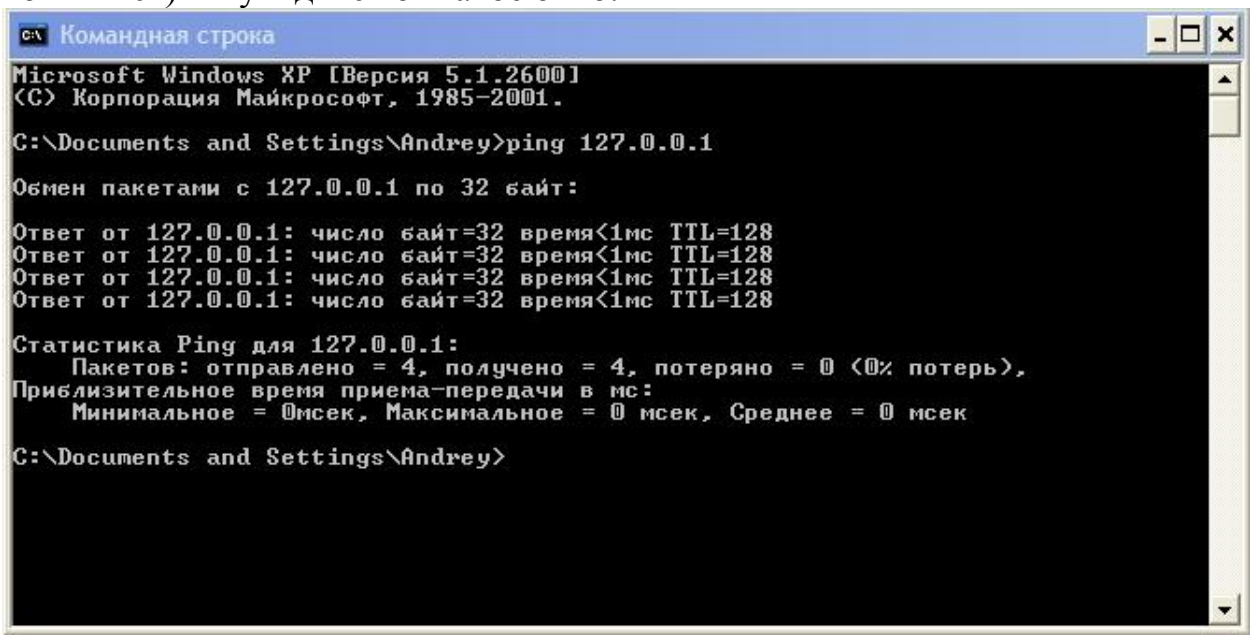
Цель работы: Изучить основы языка серверного скриптования PHP.

2. Теоретическая часть

Локальная среда разработки Denwer

Denwer – это программа, устанавливаемая на ваш локальный компьютер и реализующая на нем аналог сервера хостинг-провайдера. Т.е. вы сможете сохранить файлы сайта в определенную папку на вашем жестком диске и после этого обращаться к сайту через заданное ему доменное имя из любого браузера. При этом Денвер будет обеспечивать функционирование php-скриптов и возможность работы с базами данных MySQL. Таким образом и производится отладка сайта до заливки на сервер.

Для начала проверим готовность вашего компьютера к установке Денвера. Для этого идем в меню Пуск->Программы->Стандартные->Командная строка. В появившемся черном окне набираем команду «ping 127.0.0.1».После этого произойдет проверка локального сетевого адреса. Если все в порядке, то через несколько секунд (после того как проверка закончиться) вы увидите вот такое окно:



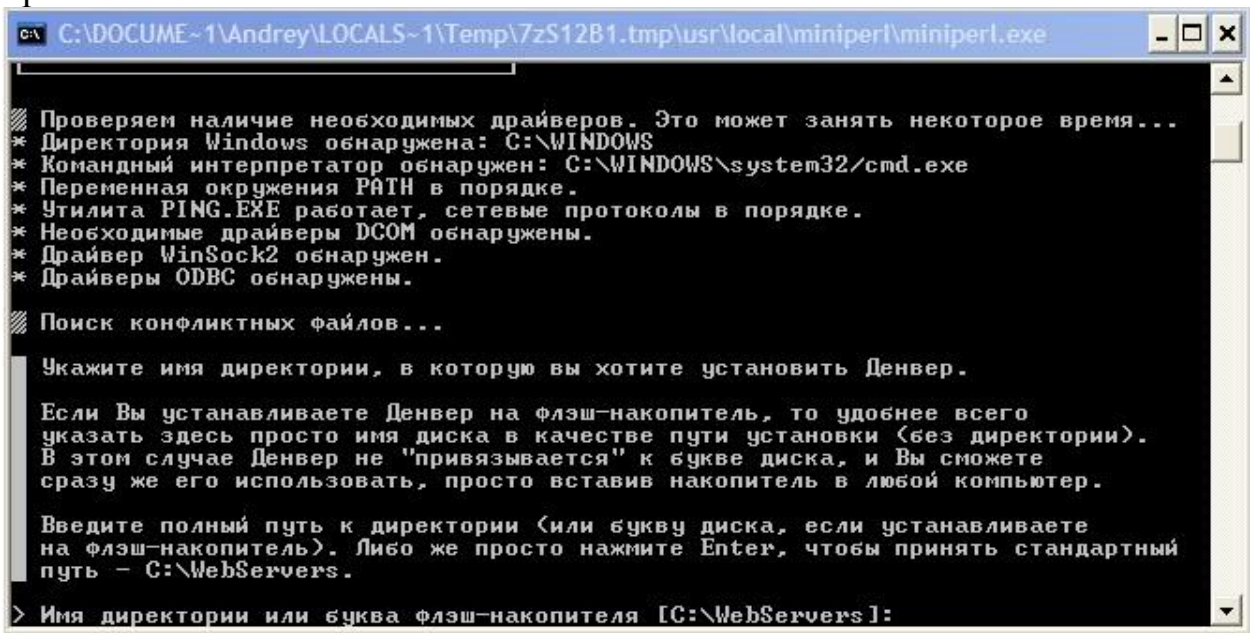
Если такого результата, как на рисунке, вы не получили, то это может означать, что ваш брандмауэр заблокировал доступ к этому IP и нужно изменить настройки брандмауэра или антивируса на разрешающие. Если вы видите тоже самое, что и на рисунке, то значит все ОК и можно переходить к установке Денвера:

1) Идем на сайт http://www.denwer.ru/ и качаем оттуда свежую версию пакета для установки:

Жмем на оранжевую кнопку, указываем e-mail, на который будет выслана ссылка для скачивания, получаем e-mail, переходим по ссылке, скачиваем Денвер.

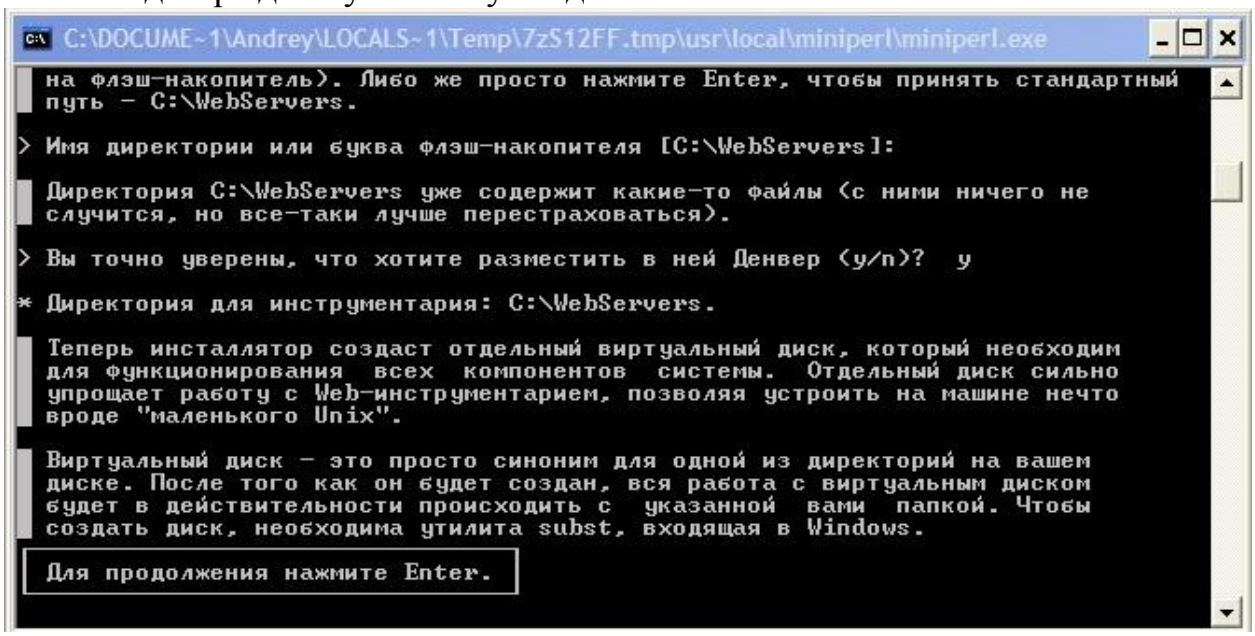
2) Запускаем скачанныйexe-файл и приступаем к установке:

3) Выполняем инструкции установщика пока не появится этот запрос:

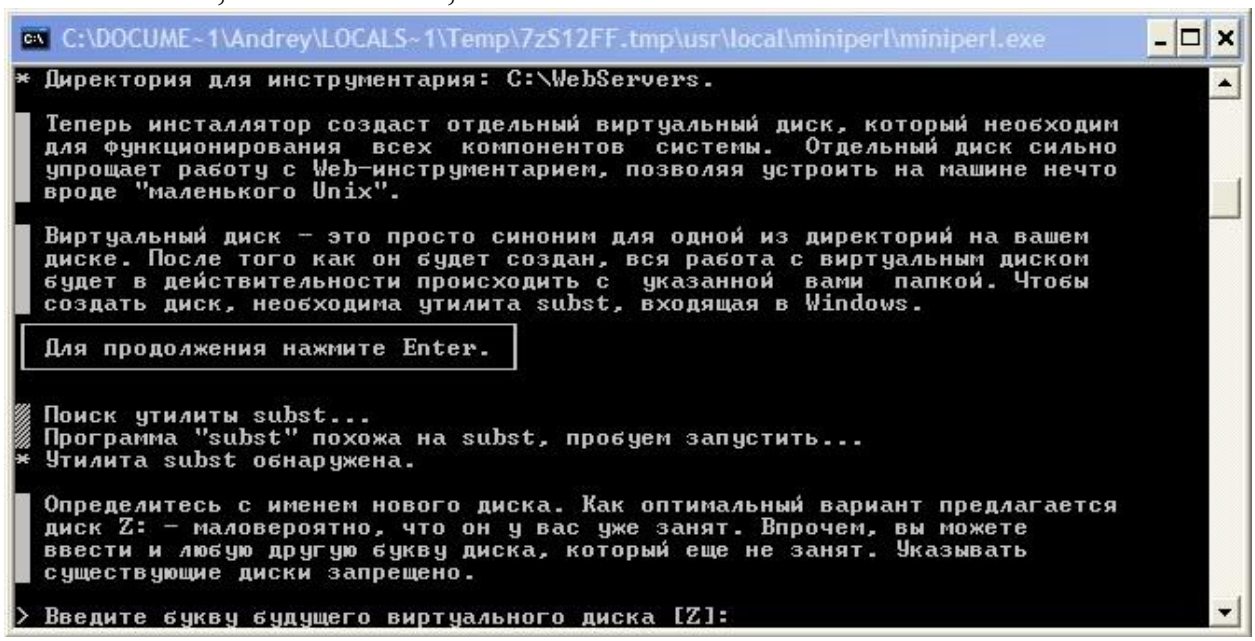


Если хотите установить Денвер в папку по умолчанию (C:\WebServers), тожмите Enter.

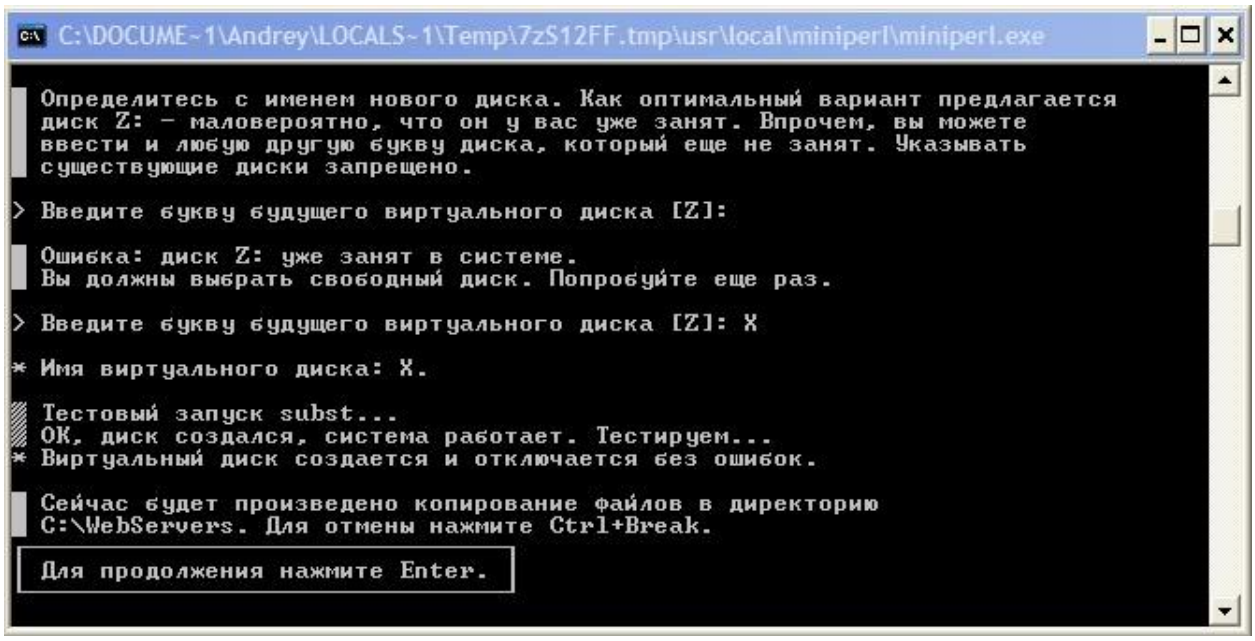
Подтверждаем установку введя Y и нажав Enter:



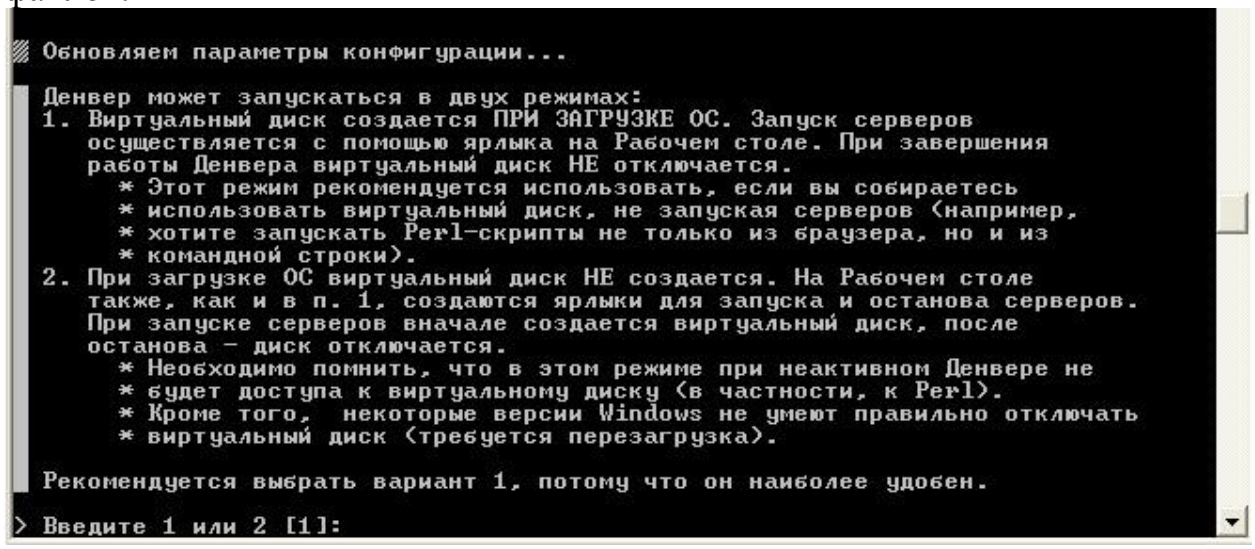
Читаем, что написано, жмем Enter:



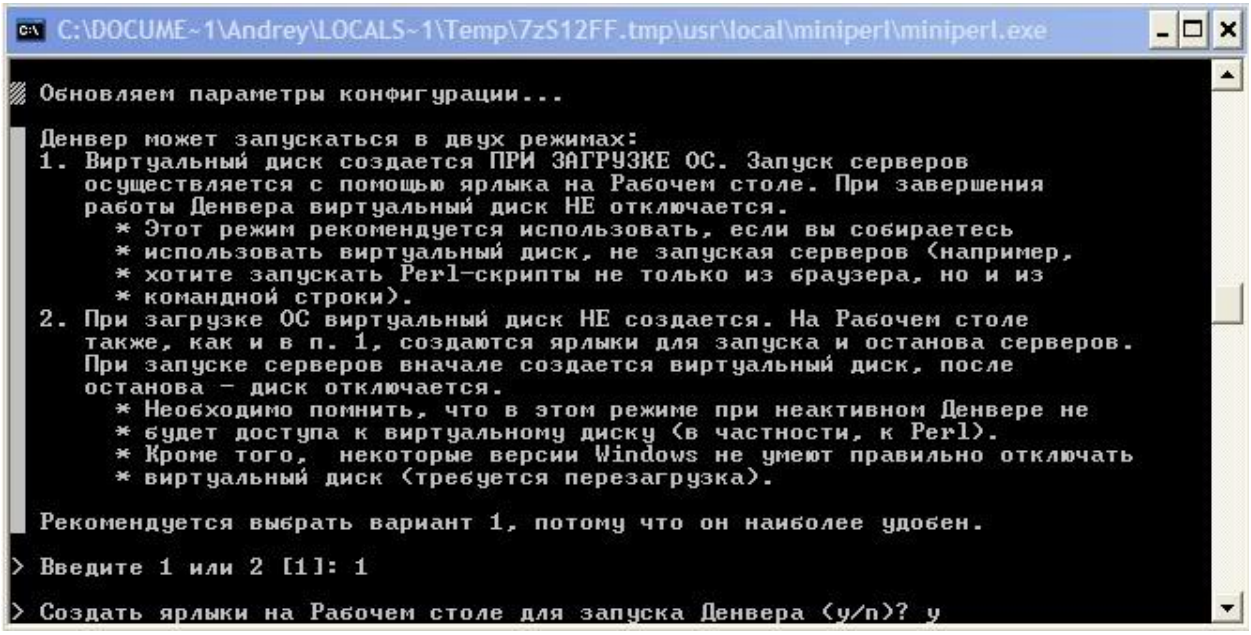
Если вдруг виртуальный диск Z присутствует в вашей системе, то введите другую букву (в примере – X). А если его нет, то продолжаем установку нажавEnter:



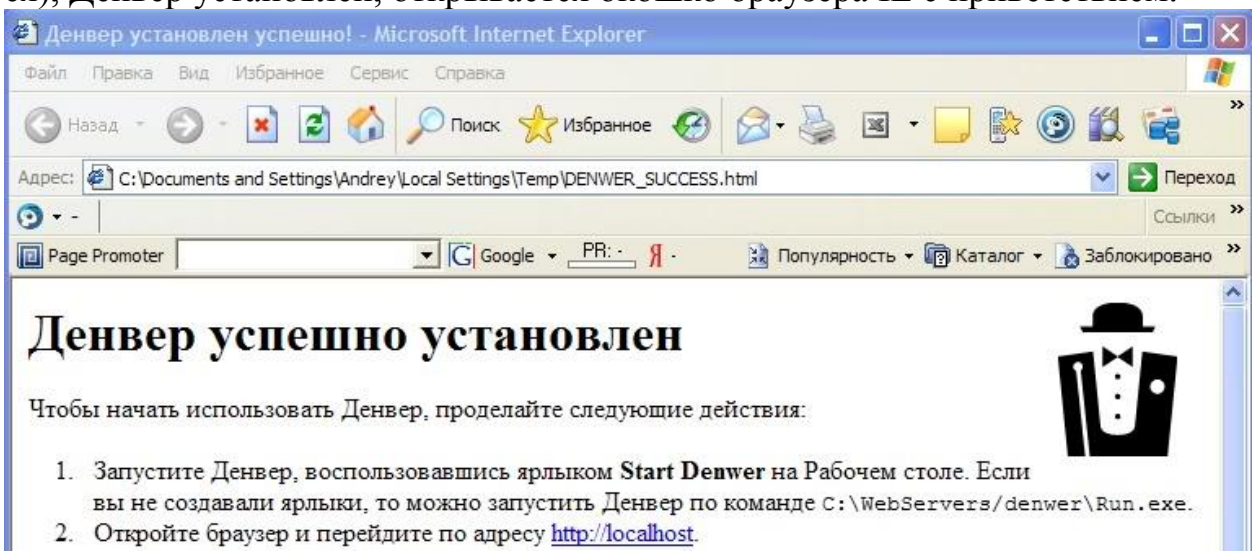
И, естественно, жмем Enter.После этого происходит копирование файлов:



Предпоследний шаг. Наиболее оптимальным является пункт 1, его-то мы и выбираем:



Создаем ярлыки на рабочем столе (чтобы удобно было пользоваться), Денвер установлен, открывается окошко браузера IE с приветствием:

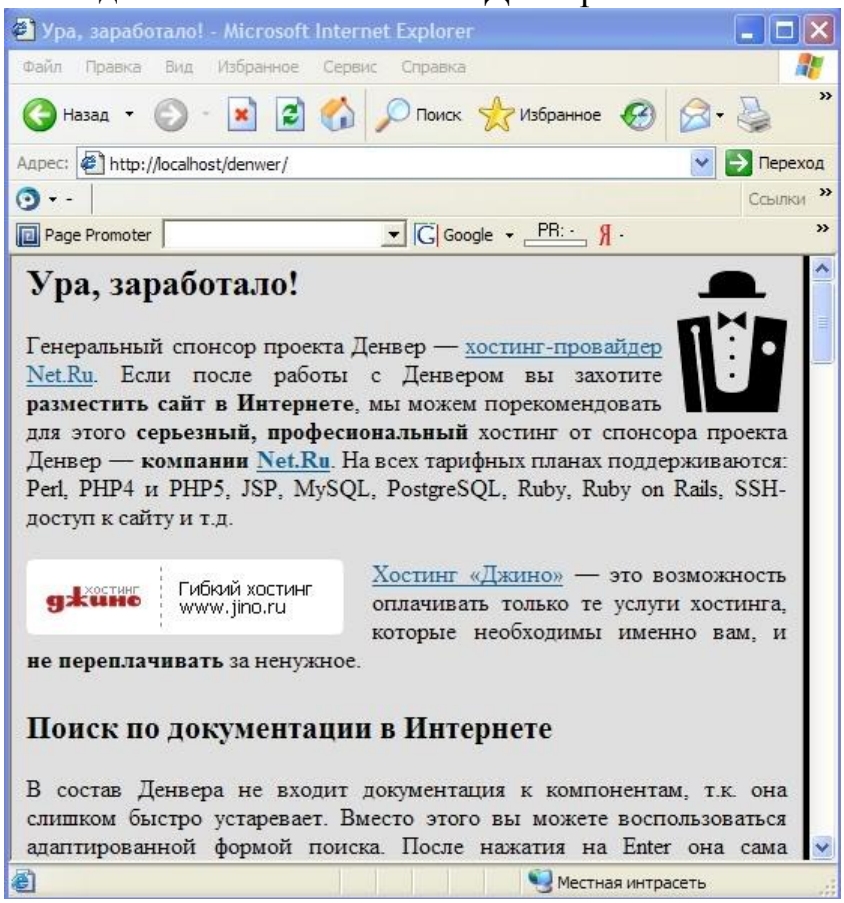


Теперь запускаем соответствующий ярлык с рабочего стола и используем Денвер:

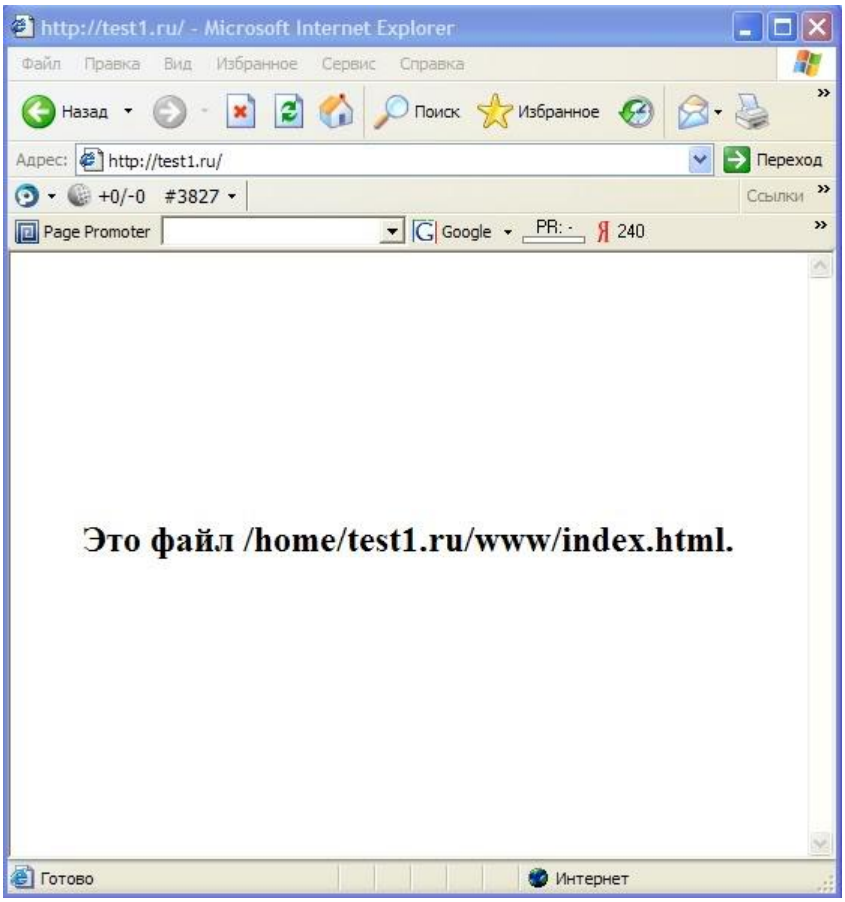
Нас интересует ярлык StartDenwer. На какое-то мгновение появится черное окошко загрузки программы и исчезнет. После этого Денвер перей-

дет в рабочее состояние. Чтобы перезапустить Денвер или отключить его воспользуйтесь двумя другими ярлыками соответственно.

4) Вводим в строку браузера http://localhost и видим запущенный с нашего жесткого диска сайт с описанием Денвера:



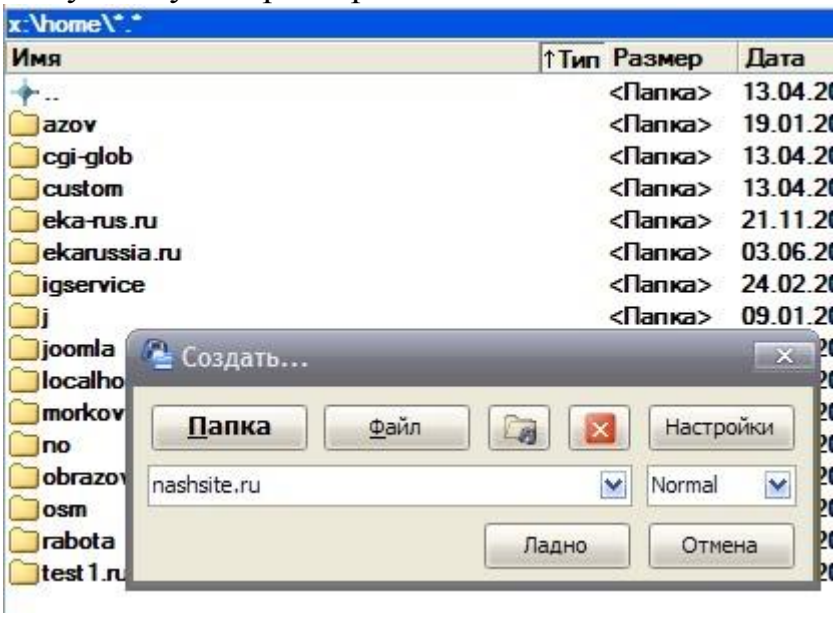
Кроме сайта http://localhost, запускаемого с нашего жесткого диска есть еще тестовый домен с адресом http://test1.ru, который тоже загружается с нашего компьютера:



Создаем папку для разработки нашего сайта (пусть он будет доступен по доменному имени nashsite.ru). Для этого нужно зайти на созданный виртуальный жесткий диск (в примере X):

Перейти в папку home и посмотреть на уже созданные при установки папки с именами, которые можно вызывать через окно браузера.У вас, скорее всего, будет только созданный автоматически test1.ru. При этом становиться понятно, что если вы хотите иметь доступ к разрабатываемо-

му сайту по доменному имени http://nashsite.ru, то, по аналогии нужно создать именно эту папку в директории home:

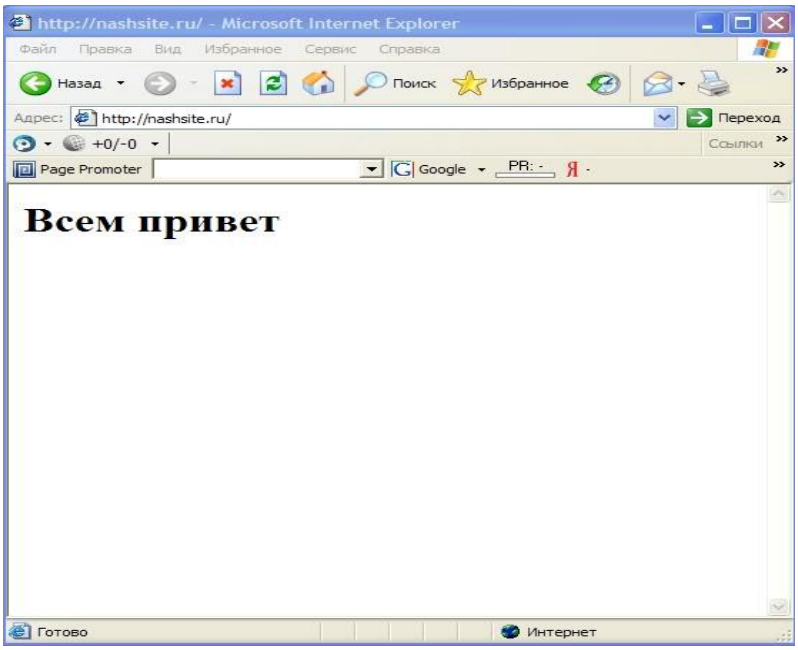


Файлы, которые будут запускаться при доступе через доменное имя http://nashsite.ru должны храниться в папке www внутри только что созданного домена.Т.е., X:\home\nashsite.ru\www. Поэтому создаем пупку www в папке nashsite.ru, и в нее будем сохранять файлы нашего проекта.После этого нужно перезапустить Денвер (при помощи иконки RestartDenwer на рабочем столе), для того, чтобы он создал новый виртуальный хост с именем http://nashsite.ru.

Теперь создадим в папке \nashsite.ru\wwwфайл index.php, в который поместим следующий код:

<?phpecho«Всем привет»; ?>

Пробуем вводить название хоста в браузере (если сайт не загружается, убедитесь в том, что вы не забыли перезапустить Денвер, а если не забыли, то перезапустите ваш браузер). Результат – загружается файл index.php, который лежит в папке X:\home\nashsite.ru\www. Внутри этого файла написано «Всем привет», что мы и видим в окне браузера:



3. Задание к работе

1. По теме (1 ЛР и 2 ЛР) работы разработать Web-сайт, состоящий из нескольких страниц.
2. Разместить в среде DENWER

4. Содержание отчета

1. Цель работы.
2. Вариант индивидуального задания.
3. Описание предметной области.
4. Структура сайта.
5. Тексты HTML-кода.
6. Выводы по работе.

Список рекомендуемой литературы

1. Браун, М. Использование HTML 4: [пер. с англ] / М. Браун, Р. Марк, Д. Хоникат [и др]. – 4-е изд., спец. – М. [и др.]: Вильямс, 1999. – 784 с.: ил. – ISBN 5-8459-0015-8.
2. Стив Суэринг, Тим Конверс, Джойс Парк -PHP и MySQL. Библия программиста, Изд. Диалектика, с 912, 2010
3. http://php.net/manual/ru/

ЛАБОРАТОРНАЯ РАБОТА №5
Основы PHP

1. Цель работы

Целью работы является изучение основ языка PHP.

2. Теоретический материал

Что такое PHP?

PHP – это широко используемый язык сценариев общего назначения с открытым исходным кодом.

Говоря проще, PHP это язык программирования, специально разработанный для написания web-приложений (сценариев), исполняющихся на Web-сервере.

Аббревиатура PHP означает “Hypertext Preprocessor (Препроцессор Гипертекста)". Синтаксис языка берет начало из C, Java и Perl. PHP достаточно прост для изучения. Преимуществом PHP является предоставление web-разработчикам возможности быстрого создания динамически генерируемых web-страниц.

Важным преимуществом языка PHP перед такими языками, как языков Perl и C заключается в возможности создания HTML документов с внедренными командами PHP.

Значительным отличием PHP от какого-либо кода, выполняющегося на стороне клиента, например, JavaScript, является то, что PHP-скрипты выполняются на стороне сервера. Вы даже можете сконфигурировать свой сервер таким образом, чтобы HTML-файлы обрабатывались процессором PHP, так что клиенты даже не смогут узнать, получают ли они обычный HTML-файл или результат выполнения скрипта.

PHP позволяет создавать качественные Web-приложения за очень короткие сроки, получая продукты, легко модифицируемые и поддерживаемые в будущем.

PHP прост для освоения, и вместе с тем способен удовлетворить запросы профессиональных программистов.

Даже если Вы впервые услышали о PHP, изучить этот язык не составит для Вас большого труда. Мы не сомневаемся, что изучив основы PHP в течение нескольких часов, вы уже сможете создавать простые PHP-скрипты.

Язык PHP постоянно совершенствуется, и ему наверняка обеспечено долгое доминирование в области языков web -программирования, по крайней мере, в ближайшее время.

Возможности PHP

Возможности PHP очень большие. Главным образом, область применения PHP сфокусирована на написание скриптов, работающих на стороне

сервера; таким образом, PHP способен выполнять всё то, что выполняет любая другая программа CGI. Например, обрабатывать данных форм, генерировать динамические страницы, отсылать и принимать cookies. Но PHP способен выполнять и множество других задач.

Существуют три основных области, где используется PHP:

- Создание скриптов для выполнения на стороне сервера. PHP наиболее широко используется именно таким образом. Все, что вам понадобится, это парсер PHP (в виде программы CGI или серверного модуля), вебсервер и браузер. Чтобы Вы могли просматривать результаты выполнения PHP-скриптов в браузере, вам нужен работающий вебсервер и установленный PHP. За более подробными сведениями обратитесь к подразделу установка PHP.
- Создание скриптов для выполнения в командной строке. Вы можете создать PHP-скрипт, способный запускаться вне зависимости от вебсервера и браузера. Все, что вам потребуется - парсер PHP. Такой способ использования PHP идеально подходит для скриптов, которые должны выполняться регулярно, например, с помощью cron (на платформах *nix или Linux) или с помощью планировщика задач (Task Scheduler) на платформах Windows. Эти скрипты также могут быть использованы в задачах простой обработки текстов. Дополнительная информация находится здесь.
- Создание приложений GUI, выполняющихся на стороне клиента. Возможно, PHP является не самым лучшим языком для создания подобных приложений, но, если вы очень хорошо знаете PHP и хотели бы использовать некоторые его возможности в своих клиент-приложениях, вы можете использовать PHP-GTK для создания таких приложений. Подобным образом Вы можете создавать и кросс-платформенные приложения. PHP-GTK является расширением PHP и не поставляется вместе с дистрибутивом PHP. Если вы заинтересованы, посетите сайт PHP-GTK.

PHP доступен для большинства операционных систем, включая Linux, многие модификации Unix (такие, как HP-UX, Solaris и OpenBSD), Microsoft Windows, Mac OS X, RISC OS, и многих других. (Существует даже версия PHP для OS/2. Неизвестно, правда, насколько соответствующая нынешним реалиям). Также в PHP включена поддержка большинства современных вебсерверов, таких, как Apache, Microsoft Internet Information Server, Personal Web Server, серверов Netscape и iPlanet, сервера Oreilly Website Pro, Caudium, Xitami, OmniHTTPd и многих других. Для большинства серверов PHP поставляется в качестве модуля, для других, поддерживающих стандарт CGI, PHP может функционировать в качестве процессора CGI.

Таким образом, выбирая PHP, вы получаете свободу выбора операционной системы и вебсервера. Кроме того, у вас появляется выбор между

использованием процедурного или объектно-ориентированного программирования или же их сочетания.

PHP способен не только выдавать HTML. Возможности PHP включают формирование изображений, файлов PDF и даже роликов Flash (с использованием libswf и Ming), создаваемых "на лету". PHP также способен выдавать любые текстовые данные, такие, как XHTML и другие XML-файлы. PHP способен осуществлять автоматическую генерацию таких файлов и сохранять их в файловой системе вашего сервера, вместо того, чтобы отдавать клиенту, организуя, таким образом, кеш динамического содержания, расположенный на стороне сервера.

Одним из значительных преимуществ PHP является поддержка широкого круга баз данных. Создание скрипта, использующего базы данных, - очень просто. В настоящее время PHP поддерживает следующие базы данных:

Adabas D	Ingres	Oracle (OCI7 и OCI8)
dBase	InterBase	Ovrimos
Empress	FrontBase	PostgreSQL
FilePro (только чтение)	mSQL	Solid
Hyperwave	Direct MS-SQL	Sybase
IBM DB2	MySQL	Velocis
Informix	ODBC	Unix dbm

Также в PHP включена поддержка DBX для работы на абстрактном уровне, так что вы можете работать с любой базой данных, использующих DBX. Кроме того, PHP поддерживает ODBC (Open Database Connection standard), таким образом, вы можете работать с любой базой данных, поддерживающей этот всемирно признанный стандарт. Подробнее о работе PHP с СУБД.

PHP также поддерживает "общение" с другими сервисами с использованием таких протоколов, как LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (на платформах Windows) и многих других. Кроме того, вы получаете возможность работать с сетевыми сокетами "напрямую". PHP поддерживает стандарт обмена сложными структурами данных WDDX. Обращая внимание на взаимодействие между различными языками, следует упомянуть о поддержке объектов Java и возможности их использования в качестве объектов PHP. Для доступа к удаленным объектам вы можете использовать расширение CORBA.

PHP включает средства обработки текстовой информации, начиная с регулярных выражений Perl или POSIX Extended и заканчивая парсером документов XML. Для парсинга XML используются стандарты SAX и

DOM. Для преобразования документов XML вы можете использовать расширение XSLT.

Последним по порядку, но не по значению, является поддержка многих других расширений, таких, как функции поисковой машины mnoGoSearch, функции IRC Gateway, функции для работы со сжатыми файлами (gzip, bz2), функции календарных вычислений, функции перевода и многое другое.

Преимущества PHP

Главным фактором языка PHP является практичность. PHP должен предоставить программисту средства для быстрого и эффективного решения поставленных задач. Практический характер PHP обусловлен пятью важными характеристиками:

- традиционностью;
- простотой;
- эффективностью;
- безопасностью;
- гибкостью.

Существует еще одна «характеристика», которая делает PHP особенно привлекательным: он распространяется бесплатно! Причем, с открытыми исходными кодами (Open Source).

Традиционность

Язык PHP будет казаться знакомым программистам, работающим в разных областях. Многие конструкции языка позаимствованы из Си, Perl.

Код PHP очень похож на тот, который встречается в типичных программах на C или Pascal. Это заметно снижает начальные усилия при изучении PHP. PHP — язык, сочетающий достоинства Perl и Си и специально нацеленный на работу в Интернете, язык с универсальным (правда, за некоторыми оговорками) и ясным синтаксисом.

И хотя PHP является довольно молодым языком, он обрел такую популярность среди web-программистов, что на данный момент является чуть ли не самым популярным языком для создания web-приложений (скриптов).

Простота

Сценарий PHP может состоять из 10 000 строк или из одной строки — все зависит от специфики вашей задачи. Вам не придется подгружать библиотеки, указывать специальные параметры компиляции или что-нибудь в этом роде. Механизм PHP просто начинает выполнять код после первой экранирующей последовательности (<?) и продолжает выполнение до того момента, когда он встретит парную экранирующую последовательность (?>). Если код имеет правильный синтаксис, он исполняется в точности так, как указал программист.

PHP — язык, который может быть встроен непосредственно в html - код страниц, которые, в свою очередь будут корректно обрабатываться PHP -интерпретатором. Мы можем использовать PHP для написания CGI-сценариев и избавиться от множества неудобных операторов вывода текста. Мы можем привлекать PHP для формирования HTML-документов, избавившись от множества вызовов внешних сценариев.

Большое разнообразие функций PHP избавят вас от написания многострочных пользовательских функций на C или Pascal .

Эффективность

Эффективность является исключительно важным фактором при программировании для многопользовательских сред, к числу которых относится и web .

Очень важное преимущество PHP заключается в его **«движке»**. «Движок» PHP не является ни компилятором, ни интерпретатором. Он является транслирующим интерпретатором. Такое устройство «движка» PHP позволяет обрабатывать сценарии с достаточно высокой скоростью.

По некоторым оценкам, большинство PHP-сценариев (особенно не очень больших размеров) обрабатываются быстрее аналогичных им программ, написанных на Perl. Однако, чтобы не делали разработчики PHP, откомпилированные исполняемые файлы будут работать значительно быстрее – в десятки, а иногда и в сотни раз. Но производительность PHP вполне достаточна для создания вполне серьезных web-приложений..

Безопасность

PHP предоставляет в распоряжение разработчиков и администраторов гибкие и эффективные средства безопасности, которые условно делятся на две категории: средства системного уровня и средства уровня приложения.

1. Средства безопасности системного уровня

В PHP реализованы механизмы безопасности, находящиеся под управлением администраторов; при правильной настройке PHP это обеспечивает максимальную свободу действий и безопасность. PHP может работать в так называемом безопасном режиме (safe mode), который ограничивает возможности применения PHP пользователями по ряду важных показателей. Например, можно ограничить максимальное время выполнения и использование памяти (неконтролируемый расход памяти отрицательно влияет на быстродействие сервера). По аналогии с cgi-bin администратор также может устанавливать ограничения на каталоги, в которых пользователь может просматривать и исполнять сценарии PHP, а также использовать сценарии PHP для просмотра конфиденциальной информации на сервере (например, файла passwd).

2. Средства безопасности уровня приложения

В стандартный набор функций PHP входит ряд надежных механизмов шифрования. PHP также совместим с многими приложениями независимых фирм, что позволяет легко интегрировать его с защищенными технологиями электронной коммерции (e-commerce). Другое преимущество заключается в том, что исходный текст сценариев PHP нельзя просмотреть в браузере, поскольку сценарий компилируется до его отправки по запросу пользователя. Реализация PHP на стороне сервера предотвращает похищение нетривиальных сценариев пользователями, знаний которых хватает хотя бы для выполнения команды View Source.

Гибкость

Поскольку PHP является встраиваемым (embedded) языком, он отличается исключительной гибкостью по отношению к потребностям разработчика. Хотя PHP обычно рекомендуется использовать в сочетании с HTML, он с таким же успехом интегрируется и в JavaScript, WML, XML и другие языки. Кроме того, хорошо структурированные приложения PHP легко расширяются по мере необходимости (впрочем, это относится ко всем основным языкам программирования).

Нет проблем и с зависимостью от браузеров, поскольку перед отправкой клиенту сценарии PHP полностью компилируются на стороне сервера. В сущности, сценарии PHP могут передаваться любым устройствам с браузерами, включая сотовые телефоны, электронные записные книжки, пейджеры и портативные компьютеры, не говоря уже о традиционных ПК. Программисты, занимающиеся вспомогательными утилитами, могут запускать PHP в режиме командной строки.

Поскольку PHP не содержит кода, ориентированного на конкретный web-сервер, пользователи не ограничиваются определенными серверами (возможно, незнакомыми для них). Apache, Microsoft IIS, Netscape Enterprise Server, Stronghold и Zeus — PHP работает на всех перечисленных серверах. Поскольку эти серверы работают на разных платформах, PHP в целом является платформенно-независимым языком и существует на таких платформах, как UNIX, Solaris, FreeBSD и Windows 95/98/NT/2000/XP/2003.

Наконец, средства PHP позволяют программисту работать с внешними компонентами, такими как Enterprise Java Beans или COM-объекты Win32. Благодаря этим новым возможностям PHP занимает достойное место среди современных технологий и обеспечивает масштабирование проектов до необходимых пределов.

Бесплатное распространение

Стратегия Open Source, и распространение исходных текстов программ в массах, оказало несомненно благотворное влияние на многие проекты, в первую очередь — Linux, хотя и успех проекта Apache сильно под-

крепил позиции сторонников Open Source. Сказанное относится и к истории создания PHP, поскольку поддержка пользователей со всего мира оказалась очень важным фактором в развитии проекта PHP.

Принятие стратегии Open Source и бесплатное распространение исходных текстов PHP оказало неоценимую услугу пользователям. Вдобавок, отзывчивое сообщество пользователей PHP является своего рода «коллективной службой поддержки», и в популярных электронных конференциях можно найти ответы даже на самые сложные вопросы.

Первая программа на PHP

Итак, будем считать, что для написания первой программы на PHP у вас все готово: установлен и настроен веб-сервер и интерпретатор PHP. Если нет, то мы вам поможем, чтобы процесс установки и настойки программного обеспечения не был для вас сложным.

Если все готово для работы с PHP, то начнем.

Традиционно, знакомство с языком программирования начинают с пресловутой программы "Hello, World!". Что ж, мы не будем отступать от этой традиции, и напишем нашу первую программу на PHP!

Итак, берем редактор PHP-кода (сойдет и простой текстовый редактор), и напишем следующий PHP код:

```
<?php
echo "Hello, World!";
?>
```

Прежде, чем запустить программу, ее нужно установить на сервере. Для этого сохраните написанный PHP скрипт под названием start.php. Затем скопируйте его в каталог DocumentRoot вашего сервера. По умолчанию, в Linux таким каталогом является /var/www/html (в старых версиях Linux - /home/httpd/html/). В Windows расположение каталога зависит от типа установленного web-сервера и его настроек. Теперь наберите в адресной строке вашего браузера http://localhost/start.php и, если все установлено и настроено правильно, вы увидите текст Hello, World!

Увидели? Если да, то поздравляем, вы написали первую простейшую программу (скрипт) на PHP!

А теперь разберем код нашего простейшего скрипта.

Код PHP заключается в специальные теги **<?** и **?>** Начало скрипта отмечается открывающим тегом <? , а конец - ?> После открывающего тега <? следует первый оператор echo, который осуществляет вывод информации на экран. Оператор echo можно назвать самым главным, ведь он выводит информацию в браузер, в результате его работы мы и видим наш Hello, World!

Теперь давайте несколько усложним наш скрипт, добавив в него вывод html-тегов:

```
<?php
echo "<html><body>";
echo "<h1>Hello, World!</h1>";
echo "</body></html>";
?>
```

Наш несколько модифицированный скрипт теперь будет выводить большими буквами текст Hello, World!

Настало время вывести текст на русском языке. Вы, наверняка, спросите, а почему мы сразу не могли вывести текст на русском языке? Ответ заключается в том, что текст на русском языке, без указания определенного заголовка html-документа, может быть выведен некорректно. А теперь давайте, наконец, изменим наш скрипт так, чтобы он мог приветствовать нас на русском языке:

```
<?php
echo "<html><head>";
echo '<meta http-equiv="Content-Type" content="text/html; char-
set=windows-1251">';
echo "<title>Моя первый PHP скрипт</title>";
echo "</head>";
echo "<body>";
echo "<h1>Привет! Я PHP скрипт!</h1>";
echo "</body>";
echo "</html>";
?>
```

Будьте внимательны, во второй строке кода html-код заключен в апострофы, а не в кавычки! Этим мы даем интерпретатору понять, что оператор echo должен выводить символы как есть, то есть не разбирая код в апострофах. А вот строки в кавычках позволяют выводить значения переменных.

В приведенном скрипте мы послали браузеру заголовок <head><meta....></head>, который дает понять браузеру, какую мы будем использовать кодировку для вывода текста html-документа, а также указали текст заголовка браузера: <title>Мой первый PHP скрипт</title>. А затем вывели текст на русском языке: <body><h1>Привет! Я PHP скрипт!</h1></body>.

PHP в HTML-документах

Если вы рассмотрели предыдущую страницу, то увидели, что всю информацию, включая html-код, мы выводили с помощью оператора echo. Однако PHP-скрипты можно писать и по-другому. Рассмотрим пример:

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

```
<title>Пример</title>
</head>
   <body>
      <?php
      echo "Привет, я - скрипт PHP!";
      ?>
   </body>
</html>
```

Теперь вы можете сохранить приведенный PHP скрипт на сервере в виде файла .php, например start.php и проверить результат.

Html-код будет корректно обрабатываться интерпретатором PHP, поэтому у вас не будут возникать ошибки при выполнении этого скрипта.

Когда PHP обрабатывает файл, он просто передаёт его текст, пока не встретит один из специальных тегов, который сообщает ему о необходимости начать интерпретацию текста как кода PHP. Затем он выполняет весь найденный код до закрывающего тега, говорящего интерпретатору, что далее снова идет просто текст. Этот механизм позволяет вам внедрять PHP-код в HTML - все за пределами тегов PHP остается неизменным, тогда как внутри - интерпретируется как PHP код.

Существует четыре набора тегов, которые могут быть использованы для обозначения PHP-кода. Из них только два (<?php. . .?> и <script language="php">. . .</script>) всегда доступны; другие могут быть включены или выключены в конфигурационном файле php.ini. Хотя короткие теги и теги в стиле ASP могут быть удобны, они не так переносимы, как длинные версии. Кроме того, если вы намереваетесь вставлять PHP-код в XML или XHTML, чтобы соответствовать XML, вам следует использовать форму <?php. . .?>.

Теги, поддерживаемые PHP:

1. <?php echo("если вы хотите работать с документами XHTML или XML, делайте так\n"); ?>
2. <? echo ("это простейшая инструкция обработки SGML\n"); ?>
   <?= выражение ?> Это синоним для "<? echo выражение ?>"
3. <script language="php">echo ("некоторые редакторы (например, FrontPage) не любят инструкции обработки");</script>
4. <% echo ("Вы можете по выбору использовать теги в стиле ASP"); %>
   <%= $variable; # Это синоним для "<% echo . . ." %>

Первый способ, <?php. . .?>, наиболее предпочтительный, так как он позволяет использовать PHP в коде, соответствующем правилам XML, таком как XHTML.

Второй способ не всегда доступен. Короткие теги доступны только когда они включены. Это можно сделать, используя функцию short_tags() (только в PHP 3), включив установку short_open_tag в конфигурационном файле PHP, либо скомпилировав PHP с параметром --enable-short-tags для configure. Даже если оно включено по умолчанию в php.ini-dist, использование коротких тегов не рекомендуется.

Четвертый способ доступен только если теги в стиле ASP были включены, используя конфигурационную установку asp_tags.

Замечание: Поддержка тегов в стиле ASP была добавлена в версии PHP 3.0.4.

Замечание: Следует избегать использования коротких тегов при разработке приложений или библиотек, предназначенных для распространения или размещения на PHP-серверах, не находящихся под вашим контролем, так как короткие теги могут не поддерживаться на целевом сервере. Для создания переносимого, совместимого кода, не используйте короткие теги.

Закрывающий тег блока PHP-кода включает сразу следующий за ним перевод строки, если он имеется. Кроме того, закрывающий тег автоматически подразумевает точку с запятой; вам не нужно заканчивать последнюю строку кода в блоке точкой с запятой. Закрывающий тег PHP-блока в конце файла не является обязательным.

PHP позволяет использовать такие структуры:

Профессиональная вставка:

```
<?php
if ($expression) {
   ?>
   <strong>Это истина.</strong>
   <?php
} else {
   ?>
   <strong>Это ложь.</strong>
   <?php
}
?>
```

Этот код работает так, как ожидается, потому что когда PHP встречает закрывающие теги ?>, он просто выводит все, что он находит до следующего открывающего тега. Приведенный пример более эффективен для вывода больших блоков текста, чем отправка всего текста через echo(), print() или что-либо подобное.

Комментарии в PHP-скриптах

Написание практически любой программы не обходится без комментариев.

PHP поддерживает комметарии в стиле 'C', 'C++' и оболочки Unix.

Комментарии в PHP могут быть трех типов:

```
<?php
    echo "Это тест"; // Это однострочный комментарий в стиле c++
    /* Это многострочный комментарий
      еще одна строка комментария */
    echo "Это еще один тест";
    echo "Последний тест"; # Это комментарий в стиле оболочки Uni
x
?>
```

Однострочные комментарии идут только до конца строки или текущего блока PHP-кода.

```
<h1>Это <?php # echo "простой";?> пример.</h1>
<p>Заголовок вверху выведет 'Это пример'.
```

Будьте внимательны, следите за отсутствием вложенных 'C'-комментариев, они могут появиться во время комментирования больших блоков:

```
<?php
  /*
    echo "Это тест"; /* Этот комментарий вызовет проблему */
  */
?>
```

Однострочные комментарии идут только до конца строки или текущего блока PHP-кода, в зависимости от того, что идет перед ними. Это означает, что HTML-код после // **?>** БУДЕТ напечатан: ?> выводит из режима PHP и возвращает в режим HTML, но // не позволяет этого сделать.

Переменные в PHP

Почти в любом языке программирования существует такое понятие, как переменная.

При программировании на PHP можно не скупиться на объявление новых переменных. Принципы экономии памяти, которые были актуальны несколько лет назад, сегодня в расчет не принимаются. Однако, при хранении в переменных больших объемов памяти, лучше удалять неиспользуемые переменные, используя оператор **Unset.**

Вообще, переменная - это область оперативной памяти, доступ к которой осуществляется по имени. Все данные, с которыми работает программа, хранятся в виде переменных (исключение — константа, которая, впрочем, может содержать только число или строку). Такого понятия, как

указатель (как в Си), в PHP не существует — при присваивании переменная копируется один-в-один, какую бы сложную структуру она ни имела. Тем не менее, в PHP, начиная с версии 4, существует понятие **ссылок** — жестких и символических, их мы вскоре рассмотрим.

Имена всех переменных в PHP должны начинаться со знака **$** — так интерпретатору значительно легче "понять" и отличить их, например, в строках. Имена переменных чувствительны к регистру букв: например, $var — не то же самое, что $Var или $VAR:

```
<?php
$var = "Bob";
$Var = "Joe";
echo "$var, $Var";      // выведет "Bob, Joe"

$4site = 'not yet';     // неверно; начинается с цифры
$_4site = 'not yet';    // верно; начинается с символа подчеркивания
$tдyte = 'mansikka';    // верно; 'д' это (Дополнительный) ASCII 228.
?>
```

Информацию о способах присвоения переменным PHP значений смотрите в разделе **Выражения.**

В PHP не нужно ни описывать переменные явно, ни указывать их тип. Интерпретатор все это делает сам. Однако иногда он может ошибаться (например, если в текстовой строке на самом деле задано десятичное число), поэтому изредка возникает необходимость явно указывать, какой же тип имеет то или иное выражение.

Иногда возникает потребность узнать тип переменной (например, переданной в параметрах функции) прямо во время выполнения программы.

Теперь пора узнать, какие типы данных (переменных) поддерживает PHP.

Типы данных

PHP является языком с динамической типизацией. Это значит, что тип данных переменной выводится во время выполнения, и в отличии от ряда других языков программирования в PHP не надо указывать перед переменной тип данных.

PHP поддерживает восемь простых типа данных:

- boolean (логический тип)
- integer (целые числа)
- double (дробные числа)
- string (строки)
- array (массивы)
- object (объекты)
- resource (ресурсы)

- NULL

Integer (целочисленный тип)

Представляет целое число со знаком размером в 32 бита (от -2 147 483 648 до 2 147 483 647).

```
$int = -100;
echo $int;
```

Здесь переменная $int представляет целочисленный тип, так как ей присваивается целочисленное значение.

Кроме десятичных целых чисел PHP обладает возможностью использовать также двоичные, восьмеричные и шестнадцатеричные числа. Шаблоны чисел для других систем:

- шестнадцатеричные : 0[xX][0-9a-fA-F]
- восьмеричные : 0[0-7]
- двоичные : 0b[01]

Например:

```
<?php
// Все числа в десятичной системе имеют значение 28
$int_10 = 28; // десятичное число
$int_2 = 0b11100; // двоичное число
$int_8 = 034; // восьмеричное число
$int_16 = 0x1C; // шестнадцатеричное число
echo "int_10 = $int_10 <br>";
echo "int_2 = $int_2 <br>";
echo "int_8 = $int_8 <br>";
echo "int_16 = $int_16";
?>
```

Тип double (числа с плавающей точкой)

Размер числа с плавающей точкой зависит от платформы. Максимально возможное значение, как правило, составляет ~1.8e308 с точностью около 14 десятичных цифр. Например:

```
<?php
$a1 = 1.5;
$a2 = 1.3e4; // 1.3 * 10^4
$a3 = 6E-8;
echo $a1 . " | " . $a2 . " | " . $a3;
?>
```

Тип boolean (логический тип)

Переменные логического типа могут принимать два значения: true и false или иначе говоря истина и ложь. Чаще всего логические значения используются в условных конструкциях:

```
<?php
$foo = true;
$a=10;
$b=5;
echo "foo = true <br>";
if($foo)
  echo $a+$b;
else
  echo $a-$b;
$foo = false;
echo "<br> foo = false <br>";
if($foo)
  echo $a+$b;
else
  echo $a-$b;
?>
```

Выражение if() проверяет истинность выражения. В данном случае проверяется значение переменной $foo. Или оно истинно или равно true, то выполняется следующее за оператором if выражение. А если переменная или выражение в операторе if равно false, то выполняется выражение после оператора else.

Специальное значение NULL

Значение NULL указывает, что значение переменной не определено. Использование данного значения полезно в тех случаях, когда мы хотим указать, что переменная не имеет значения. Например, если мы просто определим переменную без ее инициализации, и затем попробуем ее использовать, то нам интерпретатор выдаст диагностическое сообщение, что переменная не установлена:

```
<?php
$a;
echo $a;
?>
```

Использование значения NULL поможет избежать данной ситуации. Кроме того, мы сможем проверять наличие значения и в зависимости от результатов проверки производить те или иные действия:

```
<?php
$a=NULL;
if($a)
   echo "Переменная а определена";
else
   echo "Переменная а не определена";
?>
```

Константа NULL не чувствительна к регистру, поэтому мы можем написать и так:

```
$a=null;
```

Тип string (строки)

Для работы с текстом можно применять строки. Строки бывают двух типов: в двойных кавычках и ординарных. От типа кавычек зависит обработка строк интерпретатором. Так, переменные в двойных кавычках заменяются значениями, а переменные в ординарных кавычках остаются неизменными.

```
<?php
$a=10;
$b=5;
$result = "$a+$b <br>";
echo $result;
$result = '$a+$b';
echo $result;
?>
```

В этом случае мы получим следующий вывод:

10+5
$a+$b

Кроме обычных символов, строка может содержать специальные символы, которые могут быть неправильно интерпретированы. Например, нам надо добавить в строку кавычку:

```
$text = "Модель "Apple II"";
```

Данная запись будет ошибочна. Чтобы исправить ошибку, мы можем сочетать различные типы кавычек ('Модель "Apple II"' или "Модель 'Apple III'") или использовать слеш, чтобы ввести кавычку в строку:

```
$text = "Модель \"Apple II\"";
```

Тип resource (ресурсы)

Ресурс представляет специальную переменную, которая содержит ссылку на внешний ресурс. В качестве внешнего ресурса могут использоваться, например, файлы или подключения к базам данных. Ресурсы создаются и используются специальными функциями. Далее мы подробнее рассмотрим работу с файлами и подключения к базе данных.

Тип array (ассоциативные массивы)

Ассоциативный массив определяет набор элементов, каждый из которых представляет пару ключ=>значение. Создадим массив из 4-х элементов:

```
<?php
$phones = array('iPhone', 'Samsung Galaxy S III', 'Nokia N9', 'Samsung ACE II');
echo $phones[1];
?>
```

Массив создается с помощью конструкции array(), в котором определяются элементов. Далее выводим второй элемент массива. Поскольку отсчет элементов в массиве начинается с нуля, то чтобы обратиться ко второму элементу, нам надо использовать выражение $phones[1]

Так как в массиве только четыре элемента, мы не можем использовать в качестве ключа число, большее чем 3, например, $phones[4].

```
$A["Petrov"] = array("name"=>"Петров П.П.", "age"=>"34", "email"=>"petrov@mail.ru");
$A["Sidorov"] = array("name"=>"Сидоров С.С.", "age"=>"47", "email"=>"sidorov@mail.ru");
?>
```

Многомерные массивы похожи на записи в языке Pascal или структуры в языке C.

Подробно о массивах и операциях с массивами вы можете узнать **здесь**

Тип object (объекты)

Объект является одним из базовых понятий объектно-ориентированного программирования. Внутренняя структура объекта похожа на хэш, за исключением того, что для доступа к отдельным элементам и функциям используется оператор ->, а не квадратные скобки.

Для инициализации объекта используется выражение *new*, создающее в переменной экземпляр объекта.

```
<?php
class foo
{
    function do_foo()
```

```
    {
        echo "Doing foo.";
    }
}
$bar = new foo;
$bar->do_foo();
?>
```

Подробное рассмотрение объектов производится в подразделе **Классы и объекты PHP**

Тип resource (ресурсы)

Ресурс - это специальная переменная, содержащая ссылку на внешний ресурс. Ресурсы создаются и используются специальными функциями. Полный перечень этих функций и соответствующих типов ресурсов смотрите **здесь**.

Тип NULL (пустой тип)

Специальное значение **NULL** говорит о том, что эта переменная не имеет значения. **NULL** - это единственно возможное значение типа **NULL** (пустой тип).

Переменная считается **NULL** если:

- ей была присвоена константа **NULL**;
- ей еще не было присвоено какое-либо значение;
- она была удалена с помощью **unset()**.

```
<?php
$var = NULL;
?>
```

Смотрите также **is_null()** и **unset()**.

Псевдотип mixed (смешанный тип)

mixed говорит о том, что параметр может принимать множество (но не обязательно все) типов.

gettype(), например, принимает все типы PHP, тогда как **str_replace()** принимает строки и массивы.

Псевдотип number (числа)

number говорит о том, что параметр может быть либо **integer**, либо **float**.

Псевдотип callback (обратного вызова)

Некоторые функции, такие как **call_user_func()** или **usort()** принимают в качестве параметра определенные пользователем callback-функции. Callback-функции могут быть не только простыми функциями, но также методами объектов, включая статические методы классов.

PHP-функция передается просто как строка ее имени. Вы можете передать любую встроенную или определенную пользователем функцию за

исключением **array()**, **echo()**, **empty()**, **eval()**, **exit()**, **isset()**, **list()**, **print()** и **unset()**.

Приведем примеры **callback** функций:

```
<?php

// простой пример callback
function my_callback_function() {
    echo 'hello world!';
}
call_user_func('my_callback_function');
// примеры callback-метода
class MyClass {
    function myCallbackMethod() {
        echo 'Hello World!';
    }
}
// вызов метода статического класса без создания объекта
call_user_func(array('MyClass', 'myCallbackMethod'));
// вызов метода объекта
$obj = new MyClass();
call_user_func(array(&$obj, 'myCallbackMethod'));
?>
```

Константы в PHP

Константой называется именованная величина, которая не изменяется в процессе выполнения программы.

В отличие от переменной, вы не можете изменить значение константы, которое было ей присвоено при её объявлении. Константы удобно использовать для хранения значений, которые не должны изменяться во время работы программы. Константы могут содержать только скалярные данные (логического, целого, плавающего и строкового типов).

В PHP константы определяются функцией **define()**. Эта функция имеет следующий формат:

define (**$name**, **$value**, **$case_sen**), где:

$name - имя константы;

$value - значение константы;

$case_sen - необязательный параметр логического типа, указывающий, следует ли учитывать регистр букв (true) или нет (false).

Пример определения и использования констант в PHP:

```
<?php
define("pi",3.14,true);
echo pi;
```

```
// Выводит 3.14
?>
```

Если параметр **$case_sen** равен **true**, то интерпретатор будет учитывать регистр символов при работе с константой. Обратите внимание, что константы используются без предваряющего знака **$.**

Различия между константами и переменными:

- У констант нет приставки в виде знака доллара (**$**);
- Константы можно определить только с помощью функции **define()**, а не присваиванием значения;
- Константы могут быть определены и доступны в любом месте без учета области видимости;
- Константы не могут быть определены или аннулированы после первоначального объявления; и
- Константы могут иметь только скалярные значения.

Проверка существования констант

Для проверки существования константы можно использовать функцию **defined()**. Данная функция возвращает **true**, если константа объявлена. Приведем пример:

```
<?php
// Объявляем константу pi
define("pi",3.14,true);
if (defined("pi")==true) echo "Константа pi объявлена!";
// Скрипт выведет 'Константа pi объявлена!'
?>
```

Стандартные константы PHP

В PHP существуют следующие предопределенные константы:

PHP предоставляет большой список предопределенных констант для каждого выполняемого скрипта. Многие из этих констант определяются различными модулями и будут присутствовать только в том случае, если эти модули доступны в результате динамической загрузки или в результате статической сборки.

Есть пять предопределенных констант, которые меняют свое значение в зависимости от контекста, в котором они используются. Например, константа **__LINE__** зависит от строки в скрипте, на которой эта константа указана. Специальные константы нечувствительны к регистру и их список приведен ниже:

Имя	Описание
__LINE__	Текущая строка в файле.
__FILE__	Полный путь и имя текущего файла.

Имя	Описание
__DIR__	The directory of the file. If used inside an include, the directory of the included file is returned. This is equivalent to dirname(__FILE__). This directory name does not have a trailing slash unless it is the root directory.
__FUNCTION__	Имя функции. (Добавлена в PHP 4.3.0.)
__CLASS__	Имя класса. (Добавлена в PHP 4.3.0.)
__TRAIT__	The trait name. The trait name includes the namespace it was declared in (e.g. Foo\Bar).
__METHOD__	Имя метода класса. (Добавлена в PHP 5.0.0)
__NAMESPACE__	The name of the current namespace.

Выражения в PHP

Выражения - это краеугольный камень PHP. Почти все, что вы пишите в PHP, является выражением. Выражения являются "кирпичиками", из которых состоят PHP-программы. Под выражением в PHP понимается то, что имеет значение.

Основными формами выражений являются константы и переменные. Например, если вы записываете "$a = 100", вы присваиваете '100' переменной $a:

```
$a = 100;
```

В приведенном примере $a это переменная, = это оператор присваивания, а 100 это и есть выражения. Его значение 100.

Выражением может быть и переменная, если ей сопоставлено определенное значение:

```
$x = 7;
$y = $x;
```

В первой строке рассмотренного примера выражением является константа 7, а во второй строке - переменная **$x**, т.к. ранее ей было присвоено значение 7. **$y = $x** также является выражением.

Немного более сложными примерами выражений являются функции. Например, рассмотрим следующую функцию:

```
<?php
function funct ()
{
    return 5;
}
?>
```

Исходя из того, что вы хорошо знакомы с концепцией функций (если нет, то прочитайте раздел о пользовательских функциях),

вы полагаете, что запись *$x = funct()* абсолютно эквивалента записи *$x = 5*, и вы правы.
Функции - это выражения, значением которых является то, что возвращает функция.
Поскольку funct() возвращает 5, значением выражения 'funct()' является 5. Как правило, функции возвращают не статическое значение, а вычисленное.

PHP поддерживает три типа скалярных значений: целочисленные, с плавающей точкой и строковые значения (скалярными являются значения, которые вы не можете 'разбить' на меньшие части, в отличие, например, от массивов).
PHP поддерживает также два комбинированных (не скалярных) типа: массивы и объекты.
Каждый из этих типов значений может присваиваться переменной или возвращаться функцией.

PHP - это язык, ориентированный на выражения и рассматривающий почти все как выражение.
Вернемся к примеру, с которым мы уже имели дело: '$x = 7'.
Легко заметить, что здесь присутствуют два значения - значение целочисленной константы '7' и значение переменной $x, также принимающей значение 7.
Но на самом деле здесь присутствует и еще одно значение - значение самого присвоения.
Само присвоение вычисляется в присвоенное значение, в данном случае - в 7.
На практике это означает, что '$x = 7', независимо от того, что оно делает, является выражением со значением 7.
Таким образом, запись '$y = ($x = 7)' равносильна записи '$x = 7; $y = 7;' (точка с запятой обозначает конец выражения).
Поскольку операции присвоения анализируются справа налево, вы также можете написать '$y = $x = 7'.

Выражения в PHP связаны с арифметическими операциями, с которыми вы можете ознакомиться далее.

Операторы PHP

Оператором называется нечто, состоящее из одного или более значений (выражений, если говорить на жаргоне программирования), которое можно вычислить как новое значение (таким образом, вся конструкция может рассматриваться как выражение). Отсюда следует, что функции или любые другие конструкции, которые возвращают значение (например, **print()**) являются операторами, в отличие от всех остальных языковых конструкций (например, **echo()**), которые ничего не возвращают.

Арифметические операторы

Помните школьные основы арифметики? Описанные ниже операторы работают так же.

Пример	Название	Результат
-$a	Отрицание	Смена знака $a.
$a + $b	Сложение	Сумма $a и $b.
$a - $b	Вычитание	Разность $a и $b.
$a * $b	Умножение	Произведение $a и $b.
$a / $b	Деление	Частное от деления $a на $b.
$a % $b	Деление по модулю	Целочисленный остаток от деления $a на $b.
$a ** $b	Возведение в степень	Результат $a в степени $b. (появилось в PHP 5.6.0)

Операция деления ("/") всегда возвращает вещественный тип, даже если оба значения были целочисленными (или строками, которые преобразуются в целые числа).

Возможно также использовать скобки. Приоритет одних математических операций над другими и изменение приоритетов при использовании скобок в арифметических выражениях соответствуют обычным математическим правилам.

Операторы инкремента и декремента

PHP, аналогично C, поддерживает префиксные и постфиксные операторы инкремента и декремента.

Пример	Название	Действие
++$a	Префиксный инкремент	Увеличивает $a на единицу и возвращает значение $a.
$a++	Постфиксный инкремент	Возвращает значение $a, а затем увеличивает $a на единицу.
--$a	Префиксный декремент	Уменьшает $a на единицу и возвращает значение $a.
$a--	Постфиксный декремент	Возвращает значение $a, а затем уменьшает $a на единицу.

Булевые типы не подлежат инкрементированию и декрементированию.

Подробнее об операторах инкремента и декремента **здесь**.

Математические функции PHP

Вы можете подробно ознакомиться с математическими функциями и константами PHP **здесь**.

Операторы присвоения

Базовый оператор присвоения обозначается как =. На первый взгляд может показаться, что это оператор "равно". На самом деле это не так. В действительности, оператор присвоения означает, что левый операнд получает значение правого выражения, (т.е. устанавливается результирующим значением).

Результатом выполнения оператора присвоения является само присвоенное значение. Таким образом, результат выполнения *$a = 3* будет равен *3*. Это позволяет использовать конструкции вида:

```
<?php
$a = ($b = 4) + 5; // результат: $a установлена значением 9, перемен
ной $b присвоено 4.
?>
```

В дополнение к базовому оператору присвоения имеются "комбинированные операторы" для всех бинарных арифметических и строковых операций, которые позволяют использовать некоторое значение в выражении, а затем установить его как результат данного выражения. Например:

```
<?php
$a = 3;
$a += 5; // устанавливает $a значением 8, аналогично записи: $a =
$a + 5;
$b = "Hello ";
$b .= "There!"; // устанавливает $b строкой "Hello There!",  как и $
b = $b . "There!";
?>
```

Обратите внимание, что присвоение копирует оригинальную переменную в новую (присвоение по значению), таким образом все последующие изменения одной из переменных на другой никак не отражаются. Начиная с PHP 4, также поддерживается присваивание по ссылке, используя синтаксис $var = &$othervar; 'Присвоение по ссылке' означает, что обе переменные указывают на одни и те же данные и никакого копирования не происходит. Подробно о ссылках вы можете узнать **здесь**.

Побитовые операторы

Данные операторы предназначены для установки или снятия групп битов целочисленной переменной. Ведь любое число - это просто последовательность бит. Целые числа в PHP - 32-битные.

Для представления одного числа используются 32 бита:

- 0000 0000 0000 0000 0000 0000 0000 0000 - это ноль;
- 0000 0000 0000 0000 0000 0000 0000 0001 - это 1;
- 0000 0000 0000 0000 0000 0000 0000 0010 - это 2;

- 0000 0000 0000 0000 0000 0000 0000 0011 - это 3;
- 0000 0000 0000 0000 0000 0000 0000 0100 - это 4;
- 0000 0000 0000 0000 0000 0000 0000 0101 - это 5;
- ...
- 0000 0000 0000 0000 0000 0000 0000 1111 - это 15;
- ...

Побитовые операторы:

Пример	Название	Результат
$a & $b	Побитовое 'и'	Устанавливаются только те биты, которые установлены и в $a, и в $b.
$a | $b	Побитовое 'или'	Устанавливаются те биты, которые установлены либо в $a, либо в $b.
$a ^ $b	Исключающее или	Устанавливаются только те биты, которые установлены либо только в $a, либо только в $b
~ $a	Отрицание	Устанавливаются те биты, которые в $a не установлены, и наоборот.
$a << $b	Сдвиг влево	Все биты переменной $a сдвигаются на $b позиций влево (каждая позиция подразумевает 'умножение на 2')
$a >> $b	Сдвиг вправо	Все биты переменной $a сдвигаются на $b позиций вправо (каждая позиция подразумевает 'деление на 2')

Операторы сравнения

Операторы сравнения, как это видно из их названия, позволяют сравнивать между собой два значения. Также вам возможно будет интересно ознакомиться с разделом сравнение типов, в котором приведено большое количество соответствующих примеров.

В PHP разрешается сравнивать только скалярные переменные. Массивы и объекты в PHP сравнивать нельзя.

Операторы сравнения:

Пример	Название	Результат
$a == $b	Равно	TRUE если $a равно $b.
$a === $b	Тождественно равно	TRUE если $a равно $b и имеет тот же тип. (Добавлено в PHP 4)
$a != $b	Не равно	TRUE если $a не равно $b.
$a <> $b	Не равно	TRUE если $a не равно $b.
$a !== $b	Тождественно не равно	TRUE если $a не равно $b или в случае, если они разных типов (Добавлено в PHP 4)
$a < $b	Меньше	TRUE если $a строго меньше $b.
$a > $b	Больше	TRUE если $a строго больше $b.

Пример	Название	Результат
$a <= $b	Меньше или равно	TRUE если $a is меньше или равно $b.
$a >= $b	Больше или равно	TRUE если $a больше или равно $b.

Логические операторы

Приведем таблицу логических операторов PHP:

Пример	Название	Результат
$a and $b	Логическое 'и'	TRUE если и $a, и $b TRUE.
$a or $b	Логическое 'или'	TRUE если или $a, или $b TRUE.
$a xor $b	Исключающее 'или'	TRUE если $a, или $b TRUE, но не оба.
! $a	Отрицание	TRUE если $a не TRUE.
$a && $b	Логическое 'и'	TRUE если и $a, и $b TRUE.
$a || $b	Логическое 'или'	TRUE если или $a, или $b TRUE.

Операторы инкремента (++) и декремента (--) не работают с логическими переменными.

Приоритеты операторов PHP

Операторы с более высоким уровнем приоритета выполняются в первую очередь:

Приоритет	Оператор	Порядок выполнения
13	(постфикс)++ (постфикс)--	слева направо
12	++(префикс) --(префикс)	справа налево
11	* / %	слева направо
10	+ -	слева направо
9	<< >>	слева направо
8	< <= > >=	слева направо
7	== !=	слева направо
6	&	слева направо
5	^	слева направо
4	|	слева направо
3	&&	слева направо
2	||	слева направо
1	= += -= *= /= %= >>= <<== &= ^= |=	справа налево

В любом случае, если вы сомневаетесь, или боитесь ошибиться, используйте скобки.

Строковые операторы

В PHP есть два оператора для работы со строками. Первый - оператор конкатенации ('.'), который возвращает объединение левого и правого аргумента. Второй - оператор присвоения с конкатенацией, который присоединяет правый аргумент к левому. Приведем конкретный пример:

```
<?php
$a = "Hello ";
$b = $a . "World!"; // $b содержит строку "Hello World!"
$a = "Hello ";
$a .= "World!";     // $a содержит строку "Hello World!"
?>
```

Конструкции языка PHP

Любой сценарий PHP сформирован из ряда конструкций. Конструкцией могут быть операторы, функции, циклы, условные конструкции, даже конструкции, которые не делают ничего (пустые конструкции). Конструкции обычно заканчиваются точкой с запятой. Кроме того, конструкции могут быть сгруппированы в группу, формируя группу конструкций с изогнутыми фигурными скобками {...}. Группа конструкций - это также отдельная конструкция. Конструкции языка PHP похожи на конструкции языка C.

Кратко рассмотрим основные конструкции языка PHP. Для более детального изучения конструкций PHP обратитесь к разделу "Конструкции PHP".

Итак, основные конструкции языка PHP:

» Условные операторы:

if
else
elseif

» Циклы:

while
do-while
for
foreach
break
continue

» Конструкции выбора:

switch

» Конструкции возврата значений:

return

» Конструкции включений:

require()
include()
require_once()
include_once()

Пользовательские функции в PHP

В любом языке программирования существуют подпрограммы. В языке С они называются функциями, в ассемблере - подпрограммами, а в Pascal существуют два вида подпрограмм: процедуры и функции.

Подпрограмма - это специальным образом оформленный фрагмент программы, к которому можно обратиться из любого места внутри программы. Подпрограммы существенно упрощают жизнь программистам, улучшая читабельность исходного кода, а также сокращая его, поскольку отдельные фрагменты кода не нужно писать несколько раз.

В PHP такими подпрограммами являются пользовательские функции.

Особенности пользовательских функций PHP

Перечислим особенности пользовательских функций в PHP:

- Доступны параметры по умолчанию. Есть возможность вызывать одну и ту же функцию с переменным числом параметров;
- Пользовательские функции могут возвращать любой тип;
- Область видимости переменных внутри функции является иерархической (древовидной);
- Есть возможность изменять переменные, переданные в качестве аргумента.

Глобальные переменные - это переменные, которые доступны всей программе, включая подпрограммы (функции).

Локальные переменные - переменные, определенные внутри подпрограммы (функции). Они доступны только внутри функции, в которой они определены.

Для PHP все объявленные и используемые в функции переменные по умолчанию локальны для функции. То есть, по умолчанию нет возможности изменить значение глобальной переменной в теле функции.

Если вы в теле пользовательской функции будете использовать переменную с именем, идентичным имени глобальной переменной (находящейся вне пользовательской функции), то никакого отношения глобальной переменной эта локальный переменная иметь не будет. В данной ситуации в пользовательской функции будет создана локальная переменная с име-

нем, идентичным имени глобальной переменной, но доступна данная локальная переменная будет только внутри этой пользовательской функции.

Поясним данный факт на конкретном примере:

```
<?php
$a = 100;

function funct() {
$a = 70;
 echo "<h4>$a</h4>";
}
funct();
echo "<h2>$a</h2>";
?>
```

Сценарий выведет сперва 70, а затем 100:

70
100

Для избавления от приведенного недостатка, в PHP существует специальная инструкция **global**, позволяющая пользовательской функции работать с глобальными переменными. Подробнее об этом можете узнать **здесь**

Помимо локальных и глобальных переменных, в PHP существует еще один тип переменных - статические переменные.

Если в теле пользовательской функции объявлена статическая переменная, то компилятор не будет ее удалять после завершения работы функции. Пример работы пользовательской функции, содержащей статические переменные:

```
<?php
  function funct()
  {
    static $a;
    $a++;
    echo "$a";
  }
  for ($i = 0; $i++<10;) funct();
?>
```

Данный сценарий выводит строку:
1 2 3 4 5 6 7 8 9 10
Если мы удалим инструкцию static, будет выведена строка:
1 1 1 1 1 1 1 1 1 1

Это связано с тем, что переменная **$a** будет удаляться при завершении работы функции и обнуляться при каждом ее вызове. Переменная **$a** инкрементируется сразу после обнуления, а только потом выводится.

Создание пользовательских функций

Пользовательская функция может быть объявлена в любой части программы (скрипта), до места ее первого использования. И не нужно никакого предварительного объявления.

Синтаксис объявления функций следующий:

function Имя (аргумент1[=значение1],...,аргумент1[=значение1])
{
тело_функции
}

Объявление функции начинается служебным словом **function**, затем следует имя функции, после имени функции - список аргументов в скобках. Тело функции заключается в фигурные скобки и может содержать любое количество операторов.

Требования, предъявляемые к именам функций:

- Имена функций могут содержать русские буквы, но давать функциям имена, состоящие из русских букв не рекомендуется;
- Имена функций не должны содержать пробелов;
- Имя каждой пользовательской функции должно быть уникальным. При этом, необходимо помнить, что регистр при объявлении функций и обращении к ним не учитывается. То есть, например, функции funct() и FUNCT() имеют одинаковые имена;
- Функциям можно давать такие же имена, как и переменным, только без знака $ в начале имен.

Типы значений, возвращаемые пользовательскими функциями, могут быть любыми. Для передачи результата работы пользовательских функций в основную программу (скрипт) используется конструкция **return**. Если функция ничего не возвращает, конструкцию **return** не указывают. Конструкция **return** может возвращать все, что угодно, в том числе и массивы.

Приведем примеры испольльзования пользовательских функций:

```
<?php
function funct()
{
$number = 777;
return $number;
}
$a = funct();
echo $a;
?>
```

В рассмотренном примере функция funct возвращает с помощью инструкции **return** число 777. Возвращенное функцией значение присваивается глобальной переменной **$a**, а затем оператор echo выводит значение переменной **$a** в браузер. В результате мы увидим в браузере число 777.

Передача аргументов пользовательским функциям

При объявлении функции можно указать список параметров, которые могут передаваться функции, например:

```
<?php
function funct($a, $b, /* ..., */ $z) { ... };
?>
```

При вызове функции **funct()** нужно указать все передаваемые параметры, поскольку они являются обязательными. В PHP пользовательские функции могут обладать необязательными параметрами или параметрами по умолчанию, но об этом позже.

Передача аргументов по ссылке

Если вы хотите, что бы аргумент передавался по ссылке, вы должны указать амперсанд (&) перед именем аргумента в описании функции:

```
<?php
function funct(&$string)
{
   $string .= 'а эта внутри.';
}
$str = 'Эта строка за пределами функции, ';
funct($str);
echo $str;    // Выведет 'Эта строка за пределами функции, а эта внутри.'
?>
```

Параметры по умолчанию

В PHP функции могут возвращать любые значения в зависимости от переданных им параметров.

```
<?php
function makecup($type = "Чая")
{
    return "Сделайте чашечку $type.\n";
}
echo makecup();
echo makecup("Кофе");
?>
```

Результат работы приведенного скрипта будет таким:

Сделайте чашечку Чая

Сделайте чашечку Кофе

Значение по умолчанию должно быть константным выражением.

Мы рассмотрели лишь часть возможностей по работе с пользовательскими функциями.

Основы ООП

В последнее время идея объектно-ориентированного программирования (ООП), кардинально новая идеология написания программ, все более занимает умы программистов.

Объектно-ориентированные программы более просты и мобильны, их легче модифицировать и сопровождать, чем их "традиционных" собратьев. Кроме того, похоже, сама идея объектной ориентированности при грамотном ее использовании позволяет программе быть даже более защищенной от различного рода ошибок, чем это задумывал программист в момент работы над ней. Однако ничего не дается даром: сами идеи ООП довольно трудны для восприятия "с нуля", поэтому до сих пор очень большое количество программ (различные системы Unix, Apache, Perl, да и сам **PHP**) все еще пишутся на старом добром "объектно-неориентированном" Си.

PHP до недавнего времени обеспечивал лишь некоторую поддержку ООП. Однако, после выхода **PHP5** поддержка ООП в PHP стала практически полной.

Стратегию ООП лучше всего описать как смещение приоритетов в процессе программирования от функциональности приложения к структурам данных. Это позволяет программисту моделировать в создаваемых приложениях реальные объекты и ситуации. Технология ООП обладает тремя главными преимуществами:

- она проста для понимания: ООП позволяет мыслить категориями повседневных объектов;
- повышенно надежна и проста для сопровождения — правильное проектирование обеспечивает простоту расширения и модификации объектно-ориентированных программ. Модульная структура позволяет вносить независимые изменения в разные части программы, сводя к минимуму риск ошибок программирования;
- ускоряет цикл разработки — модульность и здесь играет важную роль, поскольку различные компоненты объектно-ориентированных программ можно легко использовать в других программах, что уменьшает избыточность кода и снижает риск внесения ошибок при копировании.

Специфика ООП заметно повышает эффективность труда программистов и позволяет им создавать более мощные, масштабируемые и эффективные приложения.

Объектно-ориентированное программирование основано на:

- Инкапсуляции;
- Полиморфизме;

- Наследовании.

Инкапсуляция

Инкапсуляция - это механизм, объединяющий данные и обрабатывающий их код как единое целое.

Многие преимущества ООП обусловлены одним из его фундаментальных принципов — инкапсуляцией. Инкапсуляцией называется включение различных мелких элементов в более крупный объект, в результате чего программист работает непосредственно с этим объектом. Это приводит к упрощению программы, поскольку из нее исключаются второстепенные детали.

Инкапсуляцию можно сравнить с работой автомобиля с точки зрения типичного водителя. Многие водители не разбираются в подробностях внутреннего устройства машины, но при этом управляют ею именно так, как было задумано. Пусть они не знают, как устроен двигатель, тормоз или рулевое управление, — существует специальный интерфейс, который автоматизирует и упрощает эти сложные операции. Сказанное также относится к инкапсуляции и ООП — многие подробности "внутреннего устройства" скрываются от пользователя, что позволяет ему сосредоточиться на решении конкретных задач. В ООП эта возможность обеспечивается классами, объектами и различными средствами выражения иерархических связей между ними.

Полиморфизм

Полиморфизм позволяет использовать одни и те же имена для похожих, но технически разных задач. Главным в полиморфизме является то, что он позволяет манипулировать объектами путем создания стандартных интерфейсов для схожих действий. Полиморфизм значительно облегчает написание сложных программ.

Наследование

Наследование позволяет одному объекту приобретать свойства другого объекта, не путайте с копированием объектов. При копировании создается точная копия объекта, а при наследовании точная копия дополняется уникальными свойствами, которые характерны только для производного объекта.

Классы и объекты в PHP

Класс - это базовое понятие в объектно-ориентированном программировании (ООП). Если сказать проще, то класс - это своеобразный тип переменной.

Экземпляр класса - это **объект**. Объект - это совокупность данных (свойств) и функций (методов) для их обработки. Данные и методы называются членами класса. Вообще, объектом является все то, что поддерживает инкапсуляцию.

Внутри объекта данные и код (члены класса) могут быть либо открыты, либо нет. Открытые данные и члены класса являются доступными для других частей программы, которые не являются частью объекта. А вот закрытые данные и члены класса доступны только внутри этого объекта.

Описание классов в PHP начинаются служебным словом **class**:

```
class Имя_класса {
// описание членов класса - данных и методов для их обработки
}
```

Для объявления объекта необходимо использовать оператор **new**:

Объект = new Имя_класса;

Данные описываются с помощью служебного слова var. Метод описывается так же, как и обыкновенная функция. Методу также можно передавать параметры.

Пример класса на PHP:

```
<?php
// Создаем новый класс Coor:
class Coor {
// данные (свойства):
var $name;
var $addr;
// методы:
 function Name() {
 echo "<h3>John</h3>";
 }
}
// Создаем объект класса Coor:
$object = new Coor;
?>
```

Доступ к классам и объектам в PHP

Мы рассмотрели, каким образом описываются классы и создаются объекты. Теперь нам необходимо получить доступ к членам класса, для этого в PHP предназначен оператор ->. Приведем пример:

```
<?php
// Создаем новый класс Coor:
class Coor {
// данные (свойства):
var $name;
// методы:
 function Getname() {
 echo "<h3>John</h3>";
 }
}
```

```
// Создаем объект класса Coor:
$object = new Coor;
// Получаем доступ к членам класса:
$object->name = "Alex";
echo $object->name;
// Выводит 'Alex'
// А теперь получим доступ к методу класса (фактически, к функции внутри класса):
$object->Getname();
// Выводит 'John' заглавными буквами
?>
```

Чтобы получить доступ к членам класса внутри класса, необходимо использовать указатель **$this**, который всегда относится к текущему объекту. Модифицированный метод **Getname()**:

```
function Getname() {
echo $this->name;
}
```

Таким же образом, можно написать метод **Setname()**:

```
function Setname($name) {
$this->name = $name;
}
```

Теперь для изменения имени можно использовать метод **Setname()**:

```
$object->Setname("Peter");
$object->Getname();
```

А вот и полный листинг кода:

```
<?php
// Создаем новый класс Coor:
class Coor {
// данные (свойства):
var $name;
// методы:
 function Getname() {
 echo $this->name;
 }
 function Setname($name) {
 $this->name = $name;
 }
}
// Создаем объект класса Coor:
$object = new Coor;
// Теперь для изменения имени используем метод Setname():
$object->Setname("Nick");
```

```
// А для доступа, как и прежде, Getname():
$object->Getname();
// Сценарий выводит 'Nick'
?>
```

Указатель **$this** можно также использовать для доступа к методам, а не только для доступа к данным:

```
function Setname($name) {
$this->name = $name;
$this->Getname();
}
```

Инициализация объектов

Иногда возникает необходимость выполнить инициализацию объекта - присвоить его свойствам первоначальные значения. Предположим, имя класса Coor и он содержит два свойства:имя человека и город его проживания. Можно написать метод (функцию), который будет выполнять инициализацию объекта, например **Init()**:

```
<?php
// Создаем новый класс Coor:
class Coor {
// данные (свойства):
var $name;
var $city;

// Инициализирующий метод:
 function Init($name) {
 $this->name = $name;
 $this->city = "London";
 }
}
// Создаем объект класса Coor:
$object = new Coor;
// Для инициализации объекта сразу вызываем метод:
$object->Init();
?>
```

Главное не забыть вызвать функцию сразу после создания объекта, либо вызвать какой-нибудь метод между созданием (оператор **new**) объекта и его инициализацией (вызовом **Init**).

Для того, чтобы PHP знал, что определенный метод нужно вызывать автоматически при создании объекта, ему нужно дать имя такое же, как и у класса (**Coor**):

```
function Coor ($name)
```

```
$this->name = $name;
$this->city = "London";
}
```

Метод, инициализирующий объект, называется конструктором. Однако, PHP не имеет деструкторов, поскольку ресурсы освобождаются автоматически при завершении работы скриптов.

Наследование и полиморфизм классов в PHP

Наследование классов в PHP

Наследование - это не просто создание точной копии класса, а расширение уже существующего класса, чтобы потомок мог выполнять какие-нибудь новые, характерные только ему функции. Рассмотрим конкретный пример на PHP:

```
<?php
    class Parent {
    function parent_funct() { echo "<h1>Это родительская функция</h1
    >"; }
    function test () { echo "<h1>Это родительский класс</h1>"; }
    }

    class Child extends Parent {
    function child_funct() { echo "<h2>Это дочерняя функция</h2>"; }
    function test () { echo "<h2>Это дочерний класс</h2>"; }
    }

    $object = new Parent;
    $object = new Child;

    $object->parent_funct(); // Выводит 'Это родительская функция'
    $object->child_funct(); // Выводит 'Это дочерняя функция'
    $object->test(); // Выводит 'Это дочерний класс'
    ?>
```

Ключевое слово **extends** (см. пример) говорит о том, что дочерний класс **Child** наследует все методы и свойства класса **Parent**. Родительский класс обычно называют базовым классом или суперклассом, а дочерний класс **Child** - производным или подклассом.

Полиморфизм классов в PHP

Полиморфизм - это свойство базового класса использовать функции производных классов. Практический пример, показывающий свойство класса - полиморфизм:

```
<?php
class Base {
```

```
 function funct() {
 echo "<h2>Функция базового класса</h2>";
 }
 function base_funct() {
 $this->funct();
 }
}
class Derivative extends Base {
 function funct() {
 echo "<h3>Функция производного класса</h3>";
 }
}
$b = new Base();
$d = new Derivative();

$b->base_funct();
$d->funct();
$d->base_funct();
// Скрипт выводит:
// Функция базового класса
// Функция производного класса
// Функция производного класса
?>
```

В рассмотренном примере функция **base_funct()** класса **Base** была перезаписана одноименной функцией класса **Derivative**. Функция, переопределенная таким образом, называется виртуальной.

Математические функции

- abs -- Модуль числа
- acos -- Арккосинус
- acosh -- Инверсный гиперполический косинус
- asin -- Арксинус
- asinh -- Инверсный гиперболический синус
- atan2 -- Арктангенс двух переменных
- atan -- Арктангенс
- atanh -- Инверсный гиперболический тангенс
- base_convert -- конвертирует число между различными базами
- bindec -- Конвертирует двоичные числа в десятичные
- ceil -- Округляет дробь в большую сторону
- cos -- Косинус
- cosh -- Гиперболический косинус
- decbin -- Конвертирует десятичные числа в двоичные
- dechex -- Конвертирует десятичные числа в шестнадцатиричные

- decoct -- Конвертирует десятичные числа в восьмеричные
- deg2rad -- Конвертирует число в градусах к эквиваленту в радианах
- exp -- Вычисляет экспоненту e (основание натурального логарифма)
- expm1 -- Возвращает exp(число)-1, вычисленный точно, даже когда значение числа близко к нулю
- floor -- Округляет дробь в меньшую сторону
- fmod -- Возвращает дробный остаток от деления
- getrandmax -- Вовзращает максимально возможное случайное число
- hexdec -- Конвертирует шестнадцатиричное значение в десятичное
- hypot -- Вычисляет длину гипотенузы треугольника прямого угла
- is_finite -- Определяет, является ли значение допустимым конечным числом
- is_infinite -- Определяет, бесконечно ли значение
- is_nan -- Определяет, не является ли значение числом
- lcg_value -- Объединенный линейный congruential генератор
- log10 -- Логарифм с основанием 10
- log1p -- Возвращает log(1 + число), вычисленный точно, даже когда значение числа близко к нулю
- log -- Вычисляет натуральный логарифм
- max -- Находит наибольшее значение
- min -- Находит наименьшее значение
- mt_getrandmax -- Показывает наибольшее возможное случайное значение числа
- mt_rand -- Генерирует наилучшее случайное число
- mt_srand -- подготавливает наилучший генератор случайных чисел
- octdec -- Конвертирует восьмеричное число в десятичное
- pi -- Возвращает число Пи
- pow -- Экспоненциальное выражение
- rad2deg -- Конвертирует значение в радианах к эквивалентному значению в градусах
- rand -- Генерирует случайное число
- round -- Округляет число типа float
- sin -- Синус
- sinh -- Гиперболический синус
- sqrt -- Вычисляет квадратный корень числа
- srand -- Изменяет начальное число генератора псевдослучайных чисел
- tan -- Тангенс
- tanh -- Гиперболический тангенс

Задание: Выполнить задания № 1, 2, 3,4, 6 по вариантам (см. документ «Задания»)

Список рекомендуемой литературы

1. Веллинг, Люк. Разработка Web-приложений с помощью PHP и MySQL. /Люк Веллинг, Лаура Томсон ; пер. с англ. – 3-е изд. – М. : Издательский дом «Вильямс», 2008. – 880 с.; ил.
2. «Вильямс», 2008. – 848 с.; ил.
3. http://www.php.ru/manual/introduction.html, русскоязычный официальный сайт языка PHP.
4. http://www.apache.ru/docs/rabota.html, русскоязычный официальный сайт web-сервера Apache.
5. http://php-myadmin.ru/learning/instrument-intro.html, инструментарий веб- разработчика, статья, Виктор Волков, Иван Шумилов.
6. http://dev.mysql.com/doc/ , официальный англоязычный сайт СУБД MySQL.
7. http://www.mysql.ru/docs/tkachenko/ , Вступление в PHP и MySQL, Вадим Ткаченко
8. http://www.mysql.ru/docs/pautov/ , Описание СУБД MySQL, А.Паутов
9. http://php-myadmin.ru/doc/ability.html , русскоязычный сайт PhpMyAdmin
10. http://php-myadmin.ru/learning/instrument-intro.html, инструментарий веб- разработчика, статья, Виктор Волков, Иван Шумилов.

ЛАБОРАТОРНАЯ РАБОТА №6
Работа с циклами и массивами в PHP

1. Цель работы.

Целью работы является изучение основ языка PHP. Работа с массивами. Создание и обработка простых (индексированных) массивов и ассоциированных массивов. Использование циклов для работы с массивами.

Синтаксис PHP

http://writecodeonline.com/php/
http://www.tutorialspoint.com/execute_php_online.php

Общие понятия

Язык PHP специально предназначен для веб-программирования. PHP сочетает достоинства языков C и Perl и при этом весьма прост в изучении и обладает значительными преимуществами перед традиционными языками программирования.

Синтаксис PHP очень напоминает синтаксис языка C и во многом заимствован из таких языков как Java и Perl.

Программист C очень быстро освоит язык PHP и сможет использовать его с максимальной эффективностью.

В принципе, в PHP есть практически все операторы и функции, имеющиеся в стандартном GNU C (или их аналоги), например есть циклы (while, for), операторы выбора (if, switch), функции работы с файловой системой и процессами (fopen, *dir, stat, unlink, popen, exec), функции ввода-вывода (fgets,fputs,printf) и множество других...

Цель данного раздела - краткое ознакомление с основами синтаксиса языка PHP. Более подробную информацию по конкретным составляющим синтаксиса PHP вы найдете в соответствующих разделах.

PHP и HTML

Синтаксис любого языка программирования гораздо легче "почувствовать" на примерах, нежели используя какие-то диаграммы и схемы. Поэтому приведем пример простейшего скрипта на PHP:

```
<html>
   <head>
      <title>Пример</title>
   </head>
   <body>

      <?
```

```
    echo "Привет, я - скрипт PHP!";
    ?>
  </body>
</html>
```

Обратите внимание, что HTML-код корректно обрабатывается интерпретатором PHP.

Начало сценария вас может озадачить: разве это сценарий? Откуда HTML-тэги <html> и <body>? Вот тут-то и кроется главная особенность (кстати, чрезвычайно удобная) языка PHP: PHP-скрипт может вообще не отличаться от обычного HTML-документа.

Идем дальше. Вы, наверное, догадались, что сам код сценария начинается после открывающего тэга <? и заканчивается закрывающим ?>. Итак, между этими двумя тэгами текст интерпретируется как программа, и в HTML-документ не попадает. Если же программе нужно что-то вывести, она должна воспользоваться оператором echo.

Итак, PHP устроен так, что любой текст, который расположен вне программных блоков, ограниченных <? и ?>, выводится в браузер непосредственно. В этом и заключается главная особенность PHP, в отличие от Perl и C, где вывод осуществляется с помощью стандартных операторов.

Разделение инструкций

Инструкции разделяются также как и в C или Perl - каждое выражение заканчивается точкой с запятой.

Закрывающий тег (?>) также подразумевает конец инструкции, поэтому два следующих фрагмента кода эквиваленты:

```
<?php
  echo "Это тест";
?>
<?php echo "Это тест" ?>
```

Комментарии в PHP скриптах

Написание практически любого скрипта не обходится без комментариев.

PHP поддерживает комметарии в стиле 'C', 'C++' и оболочки Unix. Например,

```
    <?php
  echo "Это тест"; // Это однострочный комментарий в стиле c++
                                    /* Это многострочный комментарий
                                       еще одна строка комментария */
  echo "Это еще один тест";
```

```
        echo "Последний тест"; # Это комментарий в стиле оболочки Unix
?>
```

Однострочные комментарии идут только до конца строки или текущего блока PHP-кода, в зависимости от того, что идет перед ними.

```
    <h1>Это <?php # echo "простой";?> пример.</h1>
<p>Заголовок вверху выведет 'Это пример'.
```

Будьте внимательны, следите за отсутствием вложенных 'C'-комментариев, они могут появиться во время комментирования больших блоков:

```
    <?php
 /*
                echo "Это тест"; /* Этот комментарий вызовет проблему */
 */
?>
```

Однострочные комментарии идут только до конца строки или текущего блока PHP-кода, в зависимости от того, что идет перед ними. Это означает, что HTML-код после // ?> БУДЕТ напечатан: ?> выводит из режима PHP и возвращает в режим HTML, но // не позволяет этого сделать.

Переменные в PHP

Имена переменных обозначаются знаком $. То же самое "Привет, я - скрипт PHP! " можно получить следующим образом:

```
    <?php
$message = "Привет, я - скрипт PHP!";
echo $message;
?>
```

Типы данных языка PHP

Скалярные типы данных

- Двоичные данные (boolean)
- Целые числа (Integer)
- Числа с плавающей точкой (Float)
- Строки (String)

Смешанные типы данных

- Массивы (Array)
- Объекты (Object)

Специальные типы данных

- Ресурсы (Resource)
- Пустой тип (NULL)

Псевдотипы данных

- Смешанный (Mixed)
- Числа (Number)
- Обратного вызова (Callback)

Дополнительно

- Манипуляции с типами данных

Синтаксис PHP

Общие понятия

Язык PHP специально предназначен для веб-программирования. PHP сочетает достоинства языков C и Perl и при этом весьма прост в изучении и обладает значительными преимуществами перед традиционными языками программирования.

Синтаксис PHP очень напоминает синтаксис языка C и во многом заимствован из таких языков как Java и Perl.

Программист C очень быстро освоит язык PHP и сможет использовать его с максимальной эффективностью. В принципе, в PHP есть практически все операторы и функции, имеющиеся в стандартном GNU C (или их аналоги), например есть циклы (while, for), операторы выбора (if, switch), функции работы с файловой системой и процессами (fopen, *dir, stat, unlink, popen, exec), функции ввода-вывода (fgets,fputs,printf) и множество других...

Цель данного раздела - краткое ознакомление с основами синтаксиса языка PHP. Более подробную информацию по конкретным составляющим синтаксиса PHP вы найдете в соответствующих разделах.

PHP и HTML

Синтаксис любого языка программирования гораздо легче "почувствовать" на примерах, нежели используя какие-то диаграммы и схемы. Поэтому приведем пример простейшего скрипта на PHP:

```
<html>
   <head>
      <title>Пример</title>
   </head>
   <body>
      <?
      echo "Привет, я - скрипт PHP!";
      ?>
   </body>
</html>
```

Вы уже наверняка заметили, что это классический скрипт, с которого начинают изучение языка программирования.

Обратите внимание, что HTML-код корректно обрабатывается интерпретатором PHP.

Начало сценария вас может озадачить: разве это сценарий? Откуда HTML-тэги <html> и <body>? Вот тут-то и кроется главная особенность (кстати, чрезвычайно удобная) языка PHP: PHP-скрипт может вообще не отличаться от обычного HTML-документа.

Идем дальше. Вы, наверное, догадались, что сам код сценария начинается после открывающего тэга <? и заканчивается закрывающим ?>. Итак, между этими двумя тэгами текст интерпретируется как программа, и в HTML-документ не попадает. Если же программе нужно что-то вывести, она должна воспользоваться оператором echo.

Итак, PHP устроен так, что любой текст, который расположен вне программных блоков, ограниченных <? и ?>, выводится в браузер непосредственно. В этом и заключается главная особенность PHP, в отличие от Perl и C, где вывод осуществляется только с помощью стандартных операторов.

Разделение инструкций

Инструкции разделяются также как и в C или Perl - каждое выражение заканчивается точкой с запятой.

Закрывающий тег (?>) также подразумевает конец инструкции, поэтому два следующих фрагмента кода эквиваленты:

```
<?php
    echo "Это тест";
?>
<?php echo "Это тест" ?>
```

Комментарии в PHP скриптах

Написание практически любого скрипта не обходится без комментариев.

PHP поддерживает комметарии в стиле 'C', 'C++' и оболочки Unix. Например:

```
<?php
    echo "Это тест"; // Это однострочный комментарий в стиле c++
    /* Это многострочный комментарий
       еще одна строка комментария */
    echo "Это еще один тест";
    echo "Последний тест"; # Это комментарий в стиле оболочки Unix
?>
```

Однострочные комментарии идут только до конца строки или текущего блока PHP-кода, в зависимости от того, что идет перед ними.

```
<h1>Это <?php # echo "простой";?> пример.</h1>
<p>Заголовок вверху выведет 'Это пример'.
```

Будьте внимательны, следите за отсутствием вложенных 'C'-комментариев, они могут появиться во время комментирования больших блоков:

```
<?php
  /*
```

```
    echo "Это тест"; /* Этот комментарий вызовет проблему */
 */
?>
```

Однострочные комментарии идут только до конца строки или текущего блока PHP-кода, в зависимости от того, что идет перед ними. Это означает, что HTML-код после // ?> БУДЕТ напечатан: ?> выводит из режима PHP и возвращает в режим HTML, но // не позволяет этого сделать.

Переменные в PHP

Имена переменных обозначаются знаком $. То же самое "Привет, я - скрипт PHP! " можно получить следующим образом:

```
<?php
$message = "Привет, я - скрипт PHP!";
echo $message;
?>
```

Типы данных в PHP

PHP поддерживает восемь простых типов данных:
Четыре скалярных типа:

- boolean (двоичные данные)
- integer (целые числа)
- float (числа с плавающей точкой или 'double')
- string (строки)

Два смешанных типа:

- array (массивы)
- object (объекты)

И два специальных типа:

- resource (ресурсы)
- NULL ("пустые")

Существуют также несколько псевдотипов:

- mixed (смешанные)
- number (числа)
- allback (обратного вызова)

Выражения в PHP

Основными формами выражений являются константы и переменные. Например, если вы записываете "$a = 100", вы присваиваете '100' переменной $a:

```
$a = 100;
```

В приведенном примере $a - это переменная, = - это оператор присваивания, а 100 - это и есть выражения. Его значение 100.

Выражением может быть и переменная, если ей сопоставлено определенное значение:

```
$x = 7;
$y = $x;
```

В первой строке рассмотренного примера выражением является константа 7, а во второй строке - переменная $x, т.к. ранее ей было присвоено значение 7. $y = $x также является выражением.

Подробно о выражениях в PHP вы найдете

Операторы PHP

Оператором называется нечто, состоящее из одного или более значений (выражений, если говорить на жаргоне программирования), которое можно вычислить как новое значение (таким образом, вся конструкция может рассматриваться как выражение).

Примеры операторов PHP:

Операторы присвоения:

```
<?php
$a = ($b = 4) + 5; // результат: $a установлена значением 9, переменной $b п
рисвоено 4.
?>
```

Комбинированные операторы:

```
<?php
$a = 3;
$a += 5; // устанавливает $a значением 8, аналогично записи: $a = $a + 5;
$b = "Hello ";
$b .= "There!"; // устанавливает $b строкой "Hello There!",  как и $b = $b . "T
here!";
?>
```

Строковые операторы:

```
<?php
$a = "Hello ";
$b = $a . "World!"; // $b содержит строку "Hello World!"

$a = "Hello ";
```

```
$a .= "World!";     // $a содержит строку "Hello World!"
?>
```

Существуют также логические операторы и операторы сравнения, однако их принято рассматривать в контексте управляющих конструкций языка.

Подробную информацию по операторам PHP вы найдете .

Управляющие конструкции языка PHP

Основными конструкциями языка PHP являются:

1. Условные операторы (if, else);
2. Циклы (while, do-while, for, foreach, break, continue);
3. Конструкции выбора (switch);
4. Конструкции объявления (declare);
5. Конструкции возврата значений (return);
6. Конструкции включений (require, include).

Примеры конструкций языка PHP:

```
<?php
if ($a > $b) echo "значение a больше, чем b";
?>
```

Приведенный пример наглядно показывает использование конструкции if совместно с оператором сравнения ($a > $b).

В следующем примере если переменная $a не равна нулю, будет выведена строка "значение a истинно (true), то есть показано взаимодействие условного оператора (конструкции) if с логическим оператором:

```
<?php
if ($a) echo "значение a истинно (true) ";
?>
```

А вот пример цикла while:

```
<?php
$x=0;
while ($x++<10) echo $x;
// Выводит 12345678910
?>
```

Информацию по всем управляющим конструкциям PHP вы можете получить

Пользовательские функции в PHP

В любом языке программирования существуют подпрограммы. В языке С они называются функциями, в ассемблере - подпрограммами, а в Pascal существуют два вида подпрограмм: процедуры и функции.

В PHP такими подпрограммами являются пользовательские функции.

Подпрограмма - это специальным образом оформленный фрагмент программы, к которому можно обратиться из любого места внутри программы. Подпрограммы существенно упрощают жизнь программистам, улучшая читабельность исходного кода, а также сокращая его, поскольку отдельные фрагменты кода не нужно писать несколько раз.

Приведем пример пользовательской функции на PHP:

```
<?php

function funct() {
$a = 100;
 echo "<h4>$a</h4>";
}
funct();
?>
```

Сценарий выводит 100:

```
100
```

Пользовательским функциям в PHP можно передавать аргументы и получать возвращаемые функциями значения.

Подробную информацию по пользовательским функциям PHP вы найдете

Встроенные (стандартные) функции PHP

PHP содержит огромное колличество встроенных функций, способных выполнять задачи различного уровня сложности.

Портал PHP.SU содержит полный справочник по стандартным функциям PHP.

ООП и PHP

PHP имеет достаточно хорошую поддержку объектно-ориентированного программирования (ООП).

В PHP можно создавать классы различных уровней, объекты и достаточно гибко ими оперировать.

Вот пример PHP класса и его использования:

```
<?php
// Создаем новый класс Coor:
class Coor {
// данные (свойства):
```

```
var $name;
// методы:
 function Getname() {
 echo "<h3>John</h3>";
 }
}
// Создаем объект класса Coor:
$object = new Coor;
// Получаем доступ к членам класса:
$object->name = "Alex";
echo $object->name;
// Выводит 'Alex'
// А теперь получим доступ к методу класса (фактически, к функции внутр
и класса):
$object->Getname();
// Выводит 'John' крупными буквами
?>
```

Теоретическая часть

Циклы в PHP

На втором месте по частоте использования, после конструкций условий (условных операторов), находятся циклы.

Циклы позволяют повторять определенное (и даже неопределенное - когда работа цикла зависит от условия) количество раз различные операторы. Данные операторы называются телом цикла. Проход цикла называется итерацией.

PHP поддерживает три вида циклов:

- Цикл с предусловием (**while**);
- Цикл с постусловием (**do-while**);
- Цикл со счетчиком (**for**);
- Специальный цикл перебора массивов (**foreach**).

Рассмотрим циклы PHP:

Цикл с предусловием while

Цикл с предусловием while работает по следующим принципам:

1. Вычисляется значение логического выражения.
2. Если значение истинно, выполняется тело цикла, в противном случае - переходим на следующий за циклом оператор.

Синтаксис цикла с предусловием:

```
while (логическое_выражение)
инструкция;
```

В данном случае телом цикла является инструкция. Обычно тело цикла состоит из большого числа операторов. Приведем пример цикла с предусловием while:

```
<?php
$x=0;
while ($x++<10) echo $x;
// Выводит 12345678910
?>
```

Обратите внимание на последовательность выполнения операций условия *$x++<10*. Сначала проверяется условие, а только потом увеличивается значение переменной. Если мы поставим операцию инкремента перед переменной (*++$x<10*), то сначала будет выполнено увеличение переменной, а только затем - сравнение. В результате мы получим строку *123456789*. Этот же цикл можно было бы записать по-другому:

```
<?php
$x=0;
while ($x<10)
{
$x++; // Увеличение счетчика
echo $x;
}
// Выводит 12345678910
?>
```

Если мы увеличим счетчик после выполнения оператора echo, мы получим строку 0123456789. В любом случае, мы имеем 10 итераций. Итерация - это выполнение операторов внутри тела цикла.

Подобно конструкции условного оператора if, можно группировать операторы внутри тела цикла while, используя следующий альтернативный синтаксис:

```
while (логическое_выражение):
инструкция;
...
endwhile;
```

Пример использования альтернативного синтаксиса:

```
<?php
$x = 1;
while ($x <= 10):
    echo $x;
    $x++;
```

```
endwhile;
?>
```

Цикл с постусловием do while

В отличие от цикла while, этот цикл проверяет значение выражения не до, а после каждого прохода (итерации). Таким образом, тело цикла выполняется хотя бы один раз. Синтаксис цикла с постусловием такой:

```
do
{
тело_цикла;
}
while (логическое_выражение);
```

После очередной итерации проверяется, истинно ли логическое выражение, и если это так, управление передается вновь на начало цикла, в противном случае цикл обрывается. Альтернативного синтаксиса для do-while разработчики PHP не предусмотрели (видимо, из-за того, что, в отличие от прикладного программирования, этот цикл довольно редко используется при программировании web-приложений).

Пример скрипта, показывающего работу цикла с постусловием do-while:

```
<?php
$x = 1;
do {
    echo $x;
} while ($x++<10);
?>
```

Рассмотренный сценарий выводит: 12345678910

Цикл со счетчиком for

Цикл со счетчиком используется для выполнения тела цикла определенное число раз. С помощью цикла for можно (и нужно) создавать конструкции, которые будут выполнять действия совсем не такие тривиальные, как простая переборка значения счетчика.

Синтаксис цикла for такой:

```
for (инициализирующие_команды; условие_цикла; команды_после_итерации) { тело_цикла; }
```

Цикл for начинает свою работу с выполнения инициализирующих_команд. Данные команды выполняются только один раз. После этого проверяется условие_цикла, если оно истинно (true), то выполняется те-

ло_цикла. После того, как будет выполнен последний оператор тела, выполняются команды_после_итерации. Затем снова проверяется условие_цикла. Если оно истинно (true), выполняется тело_цикла и команды_после_итерации, и.т.д.

```
<?php
for ($x=0; $x<10; $x++) echo $x;
?>
```

Данный сценарий выводит: 0123456789

Есть вариант вывода строки 12345678910:

```
<?php
for ($x=0; $x++<10;) echo $x;
// Выводит 12345678910
?>
```

В данном примере мы обеспечили увеличение счетчика при проверке логического выражения. В таком случае нам не нужны были команды, выполняющиеся после итерации.

Если необходимо указать несколько команд, их можно разделить запятыми, пример:

```
<?php
for ($x=0, $y=0; $x<10; $x++, $y++) echo $x;
// Выводит 0123456789
?>
```

Приведем еще один, более практичный пример использования нескольких команд в цикле for:

```
<?php
for($i=0,$j=0,$k="Точки"; $i<10; $j++,$i+=$j) { $k=$k."."; echo $k; }
// Выводит Точки.Точки..Точки...Точки....
?>
```

Рассмотренный пример (да и вообще любой цикл for) можно реализовать и через while, только это будет выглядеть не так изящно и лаконично.

Для цикла for имеется и альтернативный синтаксис:

```
for(инициализирующие_команды; условие_цикла; команды_после_итерации):
операторы;
endfor;
```

Цикл перебора массивов foreach

В PHP4 появился еще один специальный тип цикла - foreach. Данный цикл предназначен специально для перебора массивов.

Синтаксис цикла foreach выглядит следующим образом:

```
foreach (массив as $ключ=>$значение)
команды;
```

Здесь команды циклически выполняются для каждого элемента массива, при этом очередная пара ключ=>значение оказывается в переменных $ключ и $значение. Приведем пример работы цикла foreach:

```
<?php
$names["Иванов"] = "Андрей";
$names["Петров"] = "Борис";
$names["Волков"] = "Сергей";
$names["Макаров"] = "Федор";
foreach ($names as $key => $value) {
echo "<b>$value $key</b><br>";
}
?>
```

Рассмотренный сценарий выводит:

```
Андрей Иванов
Борис Петров
Сергей Волков
Федор Макаров
```

У цикла foreach имеется и другая форма записи, которую следует применять, когда нас не интересует значение ключа очередного элемента. Выглядит она так:

```
foreach (массив as $значение)
команды;
```

В этом случае доступно лишь значение очередного элемента массива, но не его ключ. Это может быть полезно, например, для работы с массивами-списками:

```
    <?php
$names[] = "Андрей";
$names[] = "Борис";
$names[] = "Сергей";
$names[] = "Федор";
foreach ($names as $value) {
echo "<b>$value</b><br>";
```

```
}
?>
```

Внимание: Цикл foreach оперирует не исходным массивом, а его копией. Это означает, что любые изменения, которые вносятся в массив, не могут быть "видны" из тела цикла. Что позволяет, например, в качестве массива использовать не только переменную, но и результат работы какой-нибудь функции, возвращающей массив (в этом случае функция будет вызвана всего один раз - до начала цикла, а затем работа будет производиться с копией возвращенного значения).

Конструкция break

Очень часто для того, чтобы упростить логику какого-нибудь сложного цикла, удобно иметь возможность его прервать в ходе очередной итерации (к примеру, при выполнении какого-нибудь особенного условия). Для этого и существует конструкция break, которая осуществляет немедленный выход из цикла. Она может задаваться с одним необязательным параметром - числом, которое указывает, из какого вложенного цикла должен быть произведен выход. По умолчанию используется 1, т. е. выход из текущего цикла, но иногда применяются и другие значения. Синтаксис конструкции break:

```
break; // По умолчанию
break(номер_цикла); // Для вложенных циклов (указывается номер прерываемого цикла)
```

Приведем примеры:

```
<?php
$x=0;
while ($x++<10) {
if ($x==3) break;
echo "<b>Итерация $x</b><br>";
}
// Когда $x равен 3, цикл прерывается
?>
```

Рассмотренный сценарий выводит:

```
Итерация 1
Итерация 2
```

Если нам нужно прервать работу определенного (вложенного) цикла, то нужно передать конструкции break параметр - номер_цикла, например, break(1). Нумерация циклов выглядит следующим образом:

```
for (...) // Третий цикл
{
  for (...) // Второй цикл
  {
    for (...) // Первый цикл
    {
    }
  }
}
```

Конструкция continue

Конструкция continue так же, как и break, работает только "в паре" с циклическими конструкциями. Она немедленно завершает текущую итерацию цикла и переходит к новой (конечно, если выполняется условие цикла для цикла с предусловием). Точно так же, как и для break, для continue можно указать уровень вложенности цикла, который будет продолжен по возврату управления.

В основном continue позволяет вам сэкономить количество фигурных скобок в коде и увеличить его удобочитаемость. Это чаще всего бывает нужно в циклах-фильтрах, когда требуется перебрать некоторое количество объектов и выбрать из них только те, которые удовлетворяют определенным условиям. Приведем пример использования конструкции continue:

```
<?php
$x=0;
while ($x++<5) {
if ($x==3) continue;
echo "<b>Итерация $x</b><br>";
}
// Цикл прервется только на третьей итерации
?>
```

Рассмотренный скрипт выводит:

```
Итерация 1
Итерация 2
Итерация 4
Итерация 5
```

Грамотное использование break и continue позволяет заметно улучшить "читабельность" кода и количество блоков else.

Массивы

Массивы (arrays) - это упорядоченные наборы данных, представляющие собой список однотипных элементов.

Существует два типа массивов, различающиеся по способу идентификации элементов.

1. В массивах первого типа элемент определяется индексом в последовательности. Такие массивы называются простыми массивами.

2. Массивы второго типа имеют ассоциативную природу, и для обращения к элементам используются ключи, логически связанные со значениями. Такие массивы называют ассоциативными массивами.

Важной особенностью PHP является то, что PHP, в отличие от других языков, позволяет создавать массивы любой сложности непосредственно в теле программы (скрипта).

Массивы могут быть как одномерными, так и многомерными.

Простые массивы и списки в PHP

При обращении к элементам простых индексируемых массивов используется целочисленный индекс, определяющий позицию заданного элемента.

Простые одномерные массивы:

Обобщенный синтаксис элементов простого одномерного массива:

```
$имя[индекс];
```

Массивы, индексами которых являются числа, начинающиеся с нуля - это **списки:**

```
<?php
// Простой способ инициализации массива
$names[0]="Апельсин";
$names[1]="Банан";
$names[2]="Груша";
$names[3]="Помидор";
// Здесь: names - имя массива, а 0, 1, 2, 3 - индексы массива
?>
```

Доступ к элементам простых массивов (списков) осуществляется следующим образом:

```
<?php
// Простой способ инициализации массива
$names[0]="Апельсин";
$names[1]="Банан";
$names[2]="Груша";
```

```
$names[3]="Помидор";
// Здесь: names - имя массива, а 0, 1, 2, 3 - индексы массива

// Выводим элементы массивов в браузер:
echo $names[0]; // Вывод элемента массива names с индексом 0
echo "<br>";
echo $names[3]; // Вывод элемента массива names с индексом 3
// Выводит:
// Апельсин
// Помидор
?>
```

С технической точки зрения разницы между простыми массивами и списками нет.

Простые массивы можно создавать, не указывая индекс нового элемента массива, это за вас сделает PHP. Вот пример:

```
<?php
// Простой способ инициализации массива, без указания индексов
$names[]="Апельсин";
$names[]="Банан";
$names[]="Груша";
$names[]="Помидор";
// PHP автоматически присвоит индексы элементам массива, начиная с 0

// Выводим элементы массивов в браузер:
echo $names[0]; // Вывод элемента массива names с индексом 0
echo "<br>";
echo $names[3]; // Вывод элемента массива names с индексом 3
// Выводит:
// Апельсин
// Помидор
?>
```

В рассмотренном примере вы можете добавлять элементы массива names простым способом, то есть не указывая индекс элемента массива:

```
$names[]="Яблоко";
```

Новый элемент простого массива (списка) будет добавлен в конец массива. В дальнейшем, с каждым новым элементом массива, индекс будет увеличиваться на единицу.

Простые многомерные массивы:

Обобщенный синтаксис элементов многомерного простого массива:

```
$имя[индекс1][индекс2]..[индексN];
```

Пример простого многомерного массива:

```
<?php
// Многомерный простой массив:
$arr[0][0]="Овощи";
$arr[0][1]="Фрукты";
$arr[1][0]="Абрикос";
$arr[1][1]="Апельсин";
$arr[1][2]="Банан";
$arr[2][0]="Огурец";
$arr[2][1]="Помидор";
$arr[2][2]="Тыква";

// Выводим элементы массива:
echo "<h3>".$arr[0][0].":</h3>";
for ($q=0; $q<=2; $q++) {
echo $arr[2][$q]."<br>";
}
echo "<h3>".$arr[0][1].":</h3>";
for ($w=0; $w<=2; $w++) {
echo $arr[1][$w]."<br>";
}
?>
```

Ассоциативные массивы в PHP

В PHP индексом массива может быть не только число, но и строка. Причем на такую строку не накладываются никакие ограничения: она может содержать пробелы, длина такой строки может быть любой.

Ассоциативные массивы особенно удобны в ситуациях, когда элементы массива удобнее связывать со словами, а не с числами.

Итак, массивы, индексами которых являются строки, называются ассоциативными массивами.

Одномерные ассоциативные массивы:

Одномерные ассоциативные массивы содержат только один ключ (элемент), соответствующий конкретному индексу ассоциативного массива. Приведем пример:

```
<?php
// Ассоциативный массив
$names["Иванов"]="Иван";
$names["Сидоров"]="Николай";
$names["Петров"]="Петр";
// В данном примере: фамилии - ключи ассоциативного массива
```

```
// , а имена - элементы массива names
?>
```

Доступ к элементам одномерных ассоциативных массивов осуществляется так же, как и к элементам обыкновенных массивов, и называется доступом по ключу:

```
echo $names["Иванов"];
```

Многомерные ассоциативные массивы:

Многомерные ассоциативные массивы могут содержать несколько ключей, соответствующих конкретному индексу ассоциативного массива. Рассмотрим пример многомерного ассоциативного массива:

```
<?php
// Многомерный массив
$A["Ivanov"] = array("name"=>"Иванов И.И.", "age"=>"25", "email"=>"ivanov@mail.ru");
$A["Petrov"] = array("name"=>"Петров П.П.", "age"=>"34", "email"=>"petrov@mail.ru");
$A["Sidorov"] = array("name"=>"Сидоров С.С.", "age"=>"47", "email"=>"sidorov@mail.ru");
?>
```

Многомерные массивы похожи на записи в языке Pascal или структуры в языке C.

Доступ к элементам многомерного ассоциативного массива осуществляется следующим образом:

```
echo $A["Ivanov"]["name"]; // Выводит Иванов И.И.
echo $A["Petrov"]["email"]; // Выводит petrov@mail.ru
```

Как вы уже заметили, для создания многомерного ассоциативного массива мы использовали специальную функцию **array**, мы ее рассмотрим позже, когда будем рассматривать операции над массивами.

Ассоциативные многомерные массивы можно создавать и классическим способом, хотя это не так удобно:

```
<?php
// Многомерный ассоциативный массив
$A["Ivanov"]["name"]="Иванов И.И.";
$A["Ivanov"]["age"]="25";
$A["Ivanov"]["email"]="ivanov@mail.ru";

$A["Petrov"]["name"]="Петров П.П.";
$A["Petrov"]["age"]="34";
$A["Petrov"]["email"]="petrov@mail.ru";
```

```
$A["Sidorov"]["name"]="Сидоров С.С.";
$A["Sidorov"]["age"]="47";
$A["Sidorov"]["email"]="sidorov@mail.ru";

// Получаем доступ к ключам многомерного ассоциативного массива
echo $A["Ivanov"]["name"]."<br>"; // Выводит Иванов И.И.
echo $A["Sidorov"]["age"]."<br>"; // Выводит 47
echo $A["Petrov"]["email"]."<br>"; // Выводит petrov@mail.ru
?>
```

Функции для работы с массивами и операции над массивами

Рассмотрим некоторые часто используемые функции для работы с массивами.

Функция list()

Предположим, у нас есть массив, состоящий из трех элементов:

```
$names[0]="Александр";
$names[1]="Николай";
$names[2]="Яков";
```

Допустим, в какой-то момент нам нужно передать значения всех трех элементов массива, соответственно трем переменным: **$alex, $nick, $yakov**. Это можно сделать так:

```
$alex = $names[0];
$nick = $names[1];
$yakov = $names[2];
```

Если массив большой, то такой способ присвоения элементов массива переменным не очень удобен.

Есть более рациональный подход - использование функции **list():**

```
list ($alex, $nick, $yakov) = $names;
```

Если нам нужны только "Николай" и "Яков", то мы можем сделать так:

```
list (, $nick, $yakov) = $names;
```

Функция array()

Функция **Array()** используется специально для создания массивов. При этом она позволяет создавать пустые массивы. Вот методы использования функции **Array():**

```
<?php
// Создает пустой массив:
$arr = array();
```

```
// Создает список с тремя элементами. Индексы начинаются с нуля:
$arr2 = array("Иванов","Петров","Сидоров");
// Создает ассоциативный массив с тремя элементами:
$arr3 = array("Иванов"=>"Иван", "Петров"=>"Петр", "Сидоров"=>"Сидор");
// Создает многомерный ассоциативный массив:
$arr4 = array("name"=>"Иванов", "age"=>"24", "email"=>"ivanov@mail.ru");
$arr4 = array("name"=>"Петров", "age"=>"34", "email"=>"petrov@mail.ru");
$arr4 = array("name"=>"Сидоров", "age"=>"47", "email"=>"sidorov@mail.ru"
);
?>
```

Операции над массивами

Сортировка массивов

Начнем с самого простого — сортировки массивов. В **PHP** для этого существует очень много функций. С их помощью можно сортировать ассоциативные массивы и списки в порядке возрастания или убывания, а также в том порядке, в каком вам необходимо — посредством пользовательской функции сортировки.

Сортировка массива по значениям с помощью функций asort() и arsort():

Функция **asort()** сортирует массив, указанный в ее параметре, так, чтобы его значения шли в алфавитном (если это строки) или в возрастающем (для чисел) порядке.

При этом сохраняются связи между ключами и соответствующими им значениями, т. е. некоторые пары ключ=>значение просто "всплывают" наверх, а некоторые — наоборот, "опускаются". Например:

```
$A=array("a"=>"Zero","b"=>"Weapon","c"=>"Alpha","d"=>"Processor");
asort($A);
foreach($A as $k=>$v) echo "$k=>$v ";
// выводит "c=>Alpha d=>Processor b=>Weapon a=>Zero"
// как видим, поменялся только порядок пар ключ=>значение
```

Функция **arsort()** выполняет то же самое, за одним исключением: она упорядочивает массив не по возрастанию, а по убыванию.

Сортировка по ключам с помощью функций ksort() и krsort():

Функция **ksort()** практически идентична функции **asort()**, с тем различием, что сортировка осуществляется не по значениями, а по ключам (в порядке возрастания).
Например:

```
$A=array("d"=>"Zero", "c"=>"Weapon", "b"=>"Alpha", "a"=>"Processor");
ksort($A);
for(Reset($A); list($k,$v)=each($A);) echo "$k=>$v ";
// выводит "a=>Processor b=>Alpha c=>Weapon d=>Zero"
```

Функция для сортировки по ключам в обратном порядке называется **krsort()** и применяется точно в таком же контексте, что и **ksort()**.

Сортировка по ключам при помощи функции uksort():

Довольно часто нам приходится сортировать что-то по более сложному критерию, чем просто по алфавиту. Например, пусть в **$Files** хранится список имен файлов и подкаталогов в текущем каталоге. Возможно, мы захотим вывести этот список не только в лексикографическом порядке, но также и чтобы все каталоги предшествовали файлам. В этом случае нам стоит воспользоваться функцией **uksort()**, написав предварительно функцию сравнения с двумя параметрами, как того требует **uksort()**.

```
<?php
// Эта функция должна сравнивать значения $f1 и $f2 и возвращать:
// -1, если $f1<$f2,
// 0, если $f1==$f2
// 1, если $f1>$f2
// Под < и > понимается следование этих имен в выводимом списке
function FCmp($f1,$f2)
{ // Каталог всегда предшествует файлу
if(is_dir($f1) && !is_dir($f2)) return -1;
// Файл всегда идет после каталога
if(!is_dir($f1) && is_dir($f2)) return 1;
// Иначе сравниваем лексикографически
if($f1<$f2) return -1; elseif($f1>$f2) return 1; else return 0;
}
// Пусть $Files содержит массив с ключами — именами файлов
// в текущем каталоге. Отсортируем его.
uksort($Files,"FCmp"); // передаем функцию сортировки "по ссылке"
?>
```

Конечно, связи между ключами и значениями функцией **uksort()** сохраняются, т. е., опять же, некоторые пары просто "всплывают" наверх, а другие — "оседают".

Сортировка по значениям при помощи функции uasort()

Функция **uasort()** очень похожа на **uksort(),** с той разницей, что сменной (пользовательской) функции сортировки "подсовываются" не

ключи, а очередные значения из массива. При этом также сохраняются связи в парах ключ=>значение.

Переворачивание массива с помощью функции array_reverce()

Функция **array_reverse()** возвращает массив, элементы которого следуют в обратном порядке относительно массива, переданного в параметре. При этом связи между ключами и значениями, конечно, не теряются. Например, вместо того, чтобы ранжировать массив в обратном порядке при помощи **arsort()**, мы можем отсортировать его в прямом порядке, а затем перевернуть:

```
$A=array("a"=>"Zero","b"=>"Weapon","c"=>"Alpha","d"=>"Processor");
asort($A);
$A=array_reverse($A);
```

Конечно, указанная последовательность работает дольше, чем один-единственный вызов **arsort()**.

Сортировка списка при помощи функций sort() и rsort()

Эти две функции предназначены в первую очередь для сортировки списков.

Функция **sort()** сортирует список (разумеется, по значениям) в порядке возрастания, а **rsort()** — в порядке убывания. Пример для функции **sort()**:

```
<?php
$A=array("40", "20", "10", "30");
sort($A);
for($i=0; $i<count($A); $i++) echo "$A[$i]"."<br>";
// выводит 10 20 30 40
?>
```

Перемешивание списка с помощью функции shuffle()

Функция **shuffle()** "перемешивает" список, переданный ей первым параметром, так, чтобы его значения распределялись случайным образом. Обратите внимание, что, во-первых, изменяется сам массив, а во вторых, ассоциативные массивы воспринимаются как списки. Пример:

```
$A=array(10,20,30,40,50);
shuffle($A);
foreach($A as $v) echo "$v ";
```

Приведенный фрагмент кода выводит числа **10, 20, 30, 40 и 50** в случайном порядке.

Выполнив этот фрагмент несколько раз, вы можете обнаружить, что от запуска к запуску очередность следования чисел не изменяется. Это свойство обусловлено тем, что функция **shuffle()** использует стандартный

генератор случайных чисел, который перед работой необходимо инициализировать при помощи вызова **srand().**

Операции с ключами и значениями массива

array_flip(array $arr)

Функция **array_flip()** "пробегает" по массиву и меняет местами его ключи и значения. Исходный массив **$arr** не изменяется, а результирующий массив просто возвращается.

Конечно, если в массиве присутствовали несколько элементов с одинаковыми значениями, учитываться будет только последний из них:

```
$A=array("a"=>"aaa", "b"=>"aaa", "c"=>"ccc");
$A=array_flip($A);
// теперь $A===array("aaa"=>"b", "ccc"=>"c");
```

array_keys(array $arr [,mixed $SearchVal])

Функция **array_keys()** возвращает список, содержащий все ключи массива **$arr**. Если задан необязательный параметр **$SearchVal**, то она вернет только те ключи, которым соответствуют значения **$SearchVal**.

Фактически, эта функция с заданным вторым параметром является обратной по отношению к оператору **[]** — извлечению значения по его ключу.

array_values(array $arr)

Функция **array_values()** возвращает список всех значений в ассоциативном массиве **$arr**. Очевидно, такое действие бесполезно для списков, но иногда оправдано для хэшей.

in_array(mixed $val, array $arr)

Функция **in_array()** возвращает **true**, если элемент со значением **$val** присутствует в массиве **$arr**. Впрочем, если вам часто приходится проделывать эту операцию, подумайте: не лучше ли будет воспользоваться ассоциативным массивом и хранить данные в его ключах, а не в значениях? На этом вы можете сильно выиграть в быстродействии.

array_count_values(list $List)

Функция **array_count_values()** подсчитывает, сколько раз каждое значение встречается в списке **$List**, и возвращает ассоциативный массив с ключами — элементами списка и значениями — количеством повторов этих элементов.

Иными словами, функция **array_count_values()** подсчитывает частоту появления значений в списке**$List**. Пример:

```
$List=array(1, "hello", 1, "world", "hello");
array_count_values($array);
// возвращает array(1=>2, "hello"=>2, "world"=>1)
```

Слияние массивов

Слияние (конкатенация) массивов - это операция создания массива, состоящего из элементов нескольких других массивов. Слияние массивов - это очень опасная операция, поскольку результат слияния подчиняется своей логике, забыв о которой можно потерять данные. Слияние массивов реализуется при помощи оператора "+" или с помощью функции **array_merge()**. Слияние списков может осуществляться только с помощью функции **array_merge()**.

Предположим, мы имеем два массива:

```
$A = array("1"=>"первый", "2"=>"Второй");
$B = array("1"=>"первый", "2"=>"Второй");
```

Теперь сольем данные два массива в один массив **$C**:

```
$C = $A + $B;
```

Оператор "+" для массивов не коммутативен. Это означает, что **$A + $B** не равно **$B + $A.**

В результате рассмотренного примера мы получим массив **$C** следующего вида:

```
"1"=>"Первый", "2"=>"Второй", "3"=>"Третий", "4"=>"Четвертый"
```

А в результате **$B + $A** мы получим такой массив:

```
"3"=>"Третий", "4"=>"Четвертый", "1"=>"Первый", "2"=>"Второй"
```

При слиянии списков такой метод не работает. Поясним данный факт на примере:

Предположим, у нас есть два массива:

```
$A = array(10,11,12);
$B = array(13,14,15);
```

В результате слияния списков **$A** и **$B ($A + $B)** мы получим: 10,11,12. А это совсем не тот результат, который мы хотели получить... Связано это с тем, что при слиянии списков с одинаковыми индексами в результирующем массиве остается элемент первого массива, причем на том же месте. В таком случае нам необходимо использовать функцию **array_merge()**

Функция array_merge()

Функция **array_merge()** призвана устранить все недостатки, присущие оператору "+" для слияния массивов. А именно, она сливает массивы, перечисленные в ее аргументах, в один большой массив и возвращает результат. Если в массивах встречаются одинаковые ключи, в результат помещается пара ключ=>значение из того массива, который расположен правее в списке аргументов. Однако это не затрагивает числовые ключи: эле-

менты с такими ключами помещаются в конец результирующего массива в любом случае.
Таким образом, с помощью **array_merge()** мы можем избавиться от всех недостатков оператора "+" для массивов. Вот пример, сливающий два списка в один:

```
$L1=array(100,200,300);
$L2=array(400,500,600);
$L=array_merge($L1,$L2);
// теперь $L===array(100,200,300,400,500,600);
```

Всегда используйте эту функцию, если вам нужно работать именно со списками, а не с обычными ассоциативными массивами.

Получение части массива

Для получения части массива можно использовать функцию **array_slice()**

array_slice(array $Arr, int $offset [, int $len])

Эта функция возвращает часть ассоциативного массива, начиная с пары **ключ=>значения** со смещением (номером)**$offset** от начала и длиной **$len** (если последний параметр не задан - до конца массива). Параметры **$offset** и **$len** задаются по точно таким же правилам, как и аналогичные параметры в функции **substr()**. А именно, они могут быть отрицательными (в этом случае отсчет осуществляется от конца массива), и т. д. Вот несколько примеров:

```
$input = array ("a", "b", "c", "d", "e");
$output = array_slice ($input, 2); // "c", "d", "e"
$output = array_slice ($input, 2, -1); // "c", "d"
$output = array_slice ($input, -2, 1); // "d"
$output = array_slice ($input, 0, 3); // "a", "b", "c"
```

Вставка и удаление элементов массивов

Мы уже знаем несколько операторов, которые отвечают за вставку и удаление элементов. Например, оператор **[]**(пустые квадратные скобки) добавляет элемент в конец массива, присваивая ему числовой ключ, а оператор **Unset()** вместе с извлечением по ключу удаляет нужный элемент. Язык PHP поддерживает и многие другие функции, которые иногда бывает удобно использовать.

array_push(alist &$Arr, mixed $var1 [, mixed $var2, …])

Эта функция добавляет к списку **$Arr** элементы **$var1**, **$var2** и т. д. Она присваивает им числовые индексы — точно так же, как это происходит для стандарных **[]**. Если вам нужно добавить всего один элемент, наверное, проще и будет воспользоваться этим оператором:

```
array_push($Arr,1000); // вызываем функцию…
$Arr[]=100; // то же самое, но короче
```

Обратите внимание, что функция **array_push()** воспринимает массив, как стек, и добавляет элементы всегда в его конец. Она возвращает новое число элементов в массиве.

array_pop(list &$Arr)

Функция **array_pop()**, является противоположностью **array_push()**, снимает элемент с "вершины" стека (то есть берет последний элемент списка) и возвращает его, удалив после этого его из **$Arr**. С помощью этой функции мы можем строить конструкции, напоминающие стек. Если список **$Arr** был пуст, функция возвращает пустую строку.

array_unshift(list &$Arr, mixed $var1 [, mixed $var2, …])

Функция **array_unshift** очень похожа на **array_push()**, но добавляет перечисленные элементы не в конец, а в начало массива. При этом порядок следования **$var1, $var2** и т. д. остается тем же, т. е. элементы как бы "вдвигаются" в список слева. Новым элементам списка, как обычно, назначаются числовые индексы, начиная с **0**; при этом все ключи старых элементов массива, которые также были числовыми, изменяются (чаще всего они увеличиваются на число вставляемых значений). Функция возвращает новый размер массива. Вот пример ее применения:

```
$A=array(10,"a"=>20,30);
array_unshift($A,"!","?");
// теперь $A===array(0=>"!", 1=>"?", 2=>10, a=>20, 3=>30)
```

mixed array_shift(list &$Arr)

Функция **mixed array_shift** извлекает первый элемент массива **$Arr** и возвращает его. Она сильно напоминает **array_pop(),** но только получает начальный, а не конечный элемент, а также производит довольно сильную "встряску" всего массива: ведь при извлечении первого элемента приходится корректировать все числовые индексы у всех оставшихся элементов…

array_unique(array $Arr)

Функция array_unique**()** возвращает массив, составленный из всех уникальных значений массива $Arr вместе с их ключами. В результирующий массив помещаются первые встретившиеся пары ключ=>значение:

```
$input=array("a" => "green", "red", "b" => "green", "blue", "red");
$result=array_unique($input);
// теперь $result===array("a"=>"green", "red", "blue");
```

array_splice(array &$Arr, int $offset [, int $len] [, int $Repl])

Функция array_splice, также как и array_slice(), возвращает подмассив $Arr, начиная с индекса $offset максимальной длины $len, но, вместе с тем, она делает и другое полезное действие. А именно, она заменяет только

что указанные элементы на то, что находится в массиве $Repl (или просто удаляет, если $Repl не указан). Параметры $offset и $len задаются так же, как и в функции substr() — а именно, они могут быть и отрицательными, в этом случае отсчет начинается от конца массива. Вот некоторые примеры:

```
<?php
$input=array("red", "green", "blue", "yellow");
array_splice($input,2);
// Теперь $input===array("red", "green")
array_splice($input,1,-1);
// Теперь $input===array("red", "yellow")
array_splice($input, -1, 1, array("black", "maroon"));
// Теперь $input===array("red", "green", "blue", "black", "maroon")
array_splice($input, 1, count($input), "orange");
// Теперь $input===array("red", "orange")
?>
```

Последний пример показывает, что в качестве параметра **$Repl** мы можем указать и обычное, строковое значение, а не массив из одного элемента.

Переменные и массивы

compact(mixed $vn1 [, mixed $vn2, …])

Функция **compact()** упаковывает в массив переменные из текущего контекста (глобального или контекста функции), заданные своими именами в **$vn1**, **$vn2** и т. д. При этом в массиве образуются пары с ключами, равными содержимому **$vnN**, и значениями соответствующих переменных. Вот пример использования этой функции:

```
$a="Test string";
$b="Some text";
$A=compact("a","b");
// теперь $A===array("a"=>"Test string", "b"=>"Some text")
```

Почему же тогда параметры функции обозначены как **mixed**? Дело в том, что они могут быть не только строками, но и списками строк. В этом случае функция последовательно перебирает все элементы этого списка, и упаковывает те переменные из текущего контекста, имена которых она встретила. Более того — эти списки могут, в свою очередь, также содержать списки строк, и т. д. Правда, последнее используется сравнительно редко, но все же вот пример:

```
$a="Test";
$b="Text";
$c="CCC";
$d="DDD";
$Lst=array("b",array("c","d"));
```

```
$A=compact("a",$Lst);
// теперь $A===array("a"=>"Test", "b"=>"Text", "c"=>"CCC", "d"=>"DDD")
```

extract(array $Arr [, int $type] [, string $prefix])

Функция **extract()** производит действия, прямо противоположные **compact()**. А именно, она получает в параметрах массив **$Arr** и превращает каждую его пару ключ=>значение в переменную текущего контекста.

Создание списка – диапазона чисел

range(int $low, int $high)

Эта функция очень простая. Она создает список, заполненный целыми числами от **$low** до **$high** включительно.

Счетчик элементов массива

Для подсчета элементов массива предназначена функция **count()**.

Пример использования функции **count()**:

```
<?php
$arr[]=5;
$arr[]=4;
$arr[]=8;
$arr[]=3;
$arr[]=8;
echo "<h2>Число элементов массива: ".count($arr)."</h2>";
// Выводит: Число элементов массива: 5
?>
```

Удаление массива и его элементов

Если вы хотите удалить массив целиком, воспользуйтесь функцией **unset().**

Если вы хотите удалить пару ключ/значение, вы также можете использовать функцию **unset().** Приведем конкретные примеры:

```
<?php
$arr = array(5 => 1, 12 => 2);
$arr[] = 56;    // В этом месте скрипта это эквивалентно $arr[13] = 56;
$arr["x"] = 42; // Это добавляет к массиву новый элемент с ключом "x"
unset($arr[5]); // Это удаляет элемент из массива
unset($arr);    // Это удаляет массив полностью
?>
```

Некоторые особенности работы с массивами

Преобразование в массив (mun array)

Для любого из типов: **integer, float, string, boolean** и **resource**, если вы преобразуете значение в массив, вы получите массив с одним элементом (с индексом 0), являющимся скалярным значением, с которого вы начали.

Если вы преобразуете в массив объект (**object**), вы получите в качестве элементов массива свойства (переменные-члены) этого объекта. Ключами будут имена переменных-членов.

Если вы преобразуете в массив значение **NULL**, вы получите пустой массив.

Сравнение массивов

Массивы можно сравнивать при помощи функции **array_diff()** и **операторов массивов:**

Операторы, работающие с массивами:

Пример	**Название**	**Результат**
$a + $b	Объединение	Объединение массива $a и массива $b.
$a == $b	Равно	**TRUE**в случае, если $a и $b содержат одни и те же элементы.
$a === $b	Тождественно равно	**TRUE** в случае, если $a и $b содержат одни и те же элементы в том же самом порядке.
$a != $b	Не равно	**TRUE** если массив $a не равен массиву $b.
$a <> $b	Не равно	**TRUE** если массив $a не равен массиву $b.
$a !== $b	Тождественно не равно	**TRUE** если массив $a не равен тождественно массиву $b.

Примеры

Пример сравнения массивов:

```
<?php
$a = array("apple", "banana");
$b = array(1 => "banana", "0" => "apple");
var_dump($a == $b); // bool(true)
var_dump($a === $b); // bool(false)
?>
```

Некоторые полезные практические примеры по работе с массивами

```
<?php
// это
$a = array( 'color' => 'red',
            'taste' => 'sweet',
            'shape' => 'round',
            'name'  => 'apple',
                       4        // ключом будет 0 );
// полностью соответствует
$a['color'] = 'red';
$a['taste'] = 'sweet';
$a['shape'] = 'round';
$a['name']  = 'apple';
$a[]        = 4;        // ключом будет 0

$b[] = 'a';
$b[] = 'b';
$b[] = 'c';
// создаст массив array(0 => 'a' , 1 => 'b' , 2 => 'c'),
// или просто array('a', 'b', 'c')
?>
```

Еще один практический пример:

```
        <?php
// Массив как карта (свойств)
$map = array( 'version'    => 4,
              'OS'         => 'Linux',
              'lang'       => 'english',
              'short_tags' => true
            );

// исключительно числовые ключи
$array = array( 7,
                8,
                0,
                156,
                -10
              );
// это то же самое, что и array(0 => 7, 1 => 8, ...)

$switching = array(         10, // ключ = 0
                    5    =>  6,
                    3    =>  7,
```

```
            'a'  =>  4,
                  11, // ключ = 6 (максимальным числовым индексом был 5)
            '8'  =>  2, // ключ = 8 (число!)
            '02' => 77, // ключ = '02'
            0    => 12  // значение 10 будет перезаписано на 12
           );

// пустой массив
$empty = array();
?>
```

Коллекция:

```
<?php
$colors = array('красный', 'синий', 'зеленый', 'желтый');

foreach ($colors as $color) {
    echo "Вам нравится $color?\n";
}
?>
```

Результат работы рассмотренного скрипта:

```
Вам нравится красный?
Вам нравится синий?
Вам нравится зеленый?
Вам нравится желтый?
```

Следующий пример создает начинающийся с единицы массив:

```
<?php
$firstquarter  = array(1 => 'Январь', 'Февраль', 'Март');
print_r($firstquarter);

?>
```

Результат работы приведенного скрипта будет следующий:

```
Array
(
    [1] => 'Январь'
    [2] => 'Февраль'
    [3] => 'Март'
)
```

Пример заполнения массива:

```
<?php
// заполняет массив всеми элементами директории
$handle = opendir('.');
while (false !== ($file = readdir($handle))) {
    $files[] = $file;
}
closedir($handle);
?>
```

Массивы упорядочены. Вы можете изменять порядок элементов, используя различные функции сортировки. Для дополнительной информации смотрите раздел функции для работы с массивами. Вы можете подсчитать количество элементов в массиве, используя функцию **count()**.

Рекурсивные и многомерные массивы:

```
<?php
$fruits = array ( "фрукты" => array ( "a" => "апельсин",
                                      "b" => "банан",
                                      "c" => "яблоко"
                                    ),
                  "числа"  => array ( 1,
                                      2,
                                      3,
                                      4,
                                      5,
                                      6
                                    ),
                  "дырки"  => array (      "первая",
                                      5 => "вторая",
                                           "третья"
                                    )
                                                                          );

// Несколько примеров доступа к значениям предыдущего массива
echo $fruits["дырки"][5];    // напечатает "вторая"
echo $fruits["фрукты"]["a"]; // напечатает "апельсин"
unset($fruits["дырки"][0]);  // удалит "первая"

// Создаст новый многомерный массив
$juices["яблоко"]["зеленое"] = "хорошее";
?>
```

Обратите внимание, что при присваивании массива всегда происходит копирование значения. Чтобы копировать массив по ссылке, вам нужно использовать оператор ссылки:

```
<?php
$arr1 = array(2, 3);
$arr2 = $arr1;
$arr2[] = 4; // $arr2 изменился,
          // $arr1 по прежнему array(2,3)

$arr3 = &$arr1;
$arr3[] = 4; // теперь $arr1 и $arr3 эквивалентны
?>
```

Порядок выполнения работы

1. Создание массива

Элементам массива могут быть присвоены значения двумя способами – непосредственно или с помощью функции array(). Рассмотрим оба способа.

Определение массива с помощью функции array()

Пример 1 Определение массива с помощью функции array()

```
<?php
$students = array("Ira","Vadim","Alex","Ann");
print $students[0];
?>
```

Данный пример состоит всего из двух строчек. В первой строке мы создаем массив с помощью функции array(), в скобках которой содержатся элементы нашего массива. Имя массива задается перед знаком равенства ($students), правила создания имени массива схожи с правилами создания имени переменной. Во второй строчке мы просим функцию print() вывести элемент массива с индексом 0 (это означает, что мы просим вывести первый элемент массива).

Индекс элемента массива, к которому происходит обращение, указывается в квадратных скобках после имени массива. Таким способом можно обращаться к элементу массива, как для получения его значения, так и для присвоения ему значения. В нашем примере, Ira имеет индекс 0, Vadim –1, Alex – 2, Ann – 3. Очевидно, запросив вызов на экран значения массива имеющего индекс 0, мы увидим на экране браузера имя Ira.

Создание элементов массива с помощью идентификатора

Существует возможность создать новый массив или добавить элемент к тому, который уже есть, с помощью идентификатора (имени) массива. Для этого достаточно указать имя массива и пару пустых квадратных скобок. Давайте создадим массив $students таким способом.

Пример 2 Создание новых элементов с помощью идентификатора

```
<?php
$students[] = "Ira";
$students[] = "Vadim";
$students[] = "Alex";
$students[] = "Ann";
print $students[0];
?>
```

Обратите внимание на то, что мы не указываем номер элемента в квадратных скобках. PHP автоматически вычисляет его, освобождая нас от необходимости помнить о том, какой следующий элемент свободен.

Результат работы программы из примера 2 будет полностью аналогичен результату работы программы из примера 1.

После того как массив создан, можно добавлять к нему новые элементы. В примере 3 мы создаем массив с помощью функции array() и добавляем к нему новые элементы.

Пример 3 Добавление новых элементов

```
<?php
$students = array("Ira","Vadim","Alex","Ann");
$students[] = "Max";
print $students[4];
?>
```

Во второй строчке программы мы добавили в массив новый элемент, имеющий индекс 4 и значение Max. После этого, обратившись к нему по индексу, мы вывели его на печать.

Ассоциированные массивы

К любому элементу массива можно обратиться по индексу. Раньше в качестве индексов мы использовали целые числа. Это довольно удобно, но имеется один существенный недостаток этого метода. Представьте себе массив, элементы которого хранят телефоны ваших друзей. Допустим, вы хотите посмотреть телефон Макса. Для этого вам надо вспомнить, под каким индексом хранится этот телефон у вас в массиве. Когда телефонов всего 5 – 6 это еще реально, но когда их 50. В этом случае нам очень сильно могут помочь *ассоциированные массивы*. Дело в том, что к каждому элементу такого массива можно обратиться по имени. Вы просто просите показать вам элемент массива под именем Макс и сразу получаете интересующий вас телефон. Удобно и просто.

Ассоциированный массив – это массив, к элементу которого можно обратиться по имени. Ассоциированный массив можно создать непосредственно или с помощью функции array().

Создание ассоциированного массива с помощью функции array()

Пример 4 Создание ассоциированного массива с помощью функции array()

```
<?php
$tel = array(
                "Max Koshelev" => "580-46-82",
"Ann Reish" => "589-90-34",
"Pashsa Golikov" => "480-57-58"
  );
print $tel["Max Koshelev"];
?>
```

Для того чтобы создать ассоциированный массив с помощью функции array(), нужно задать как имя **(MaxKoshelev, AnnReish, PashsaGolikov), так и значение (580-46-82, 589-90-34, 480-57-58)** для каждого элемента. В нашем примере мы создаем массив $tel из трех элементов. К любому элементу массива можно обратиться по имени. Имена элементов массива – это строки, которые необходимо брать в кавычки. Имя элемента может состоять из нескольких слов (как у нас, фамилия и имя). Результат работы программы – появление на экране браузера телефона Кошелева Максима - 580-46-82.

Непосредственное создание ассоциированного массива

Создать новый массив или добавить к существующему пару – имя/значение можно, просто присвоив значение элементу массива, указав этот элемент по имени.

Пример 5 Непосредственное задание ассоциированного массива

```
<?php
$tel["Max Koshelev"] = "580-46-82";
$tel["Ann Reish”] = "589-90-34";
$tel["Pashsa Golikov"] = "480-57-58";
print $tel["Max Koshelev"];
?>
```

В данном примере мы опять создаем массив $tel.

2. Работа с массивами

Получение размера массива

К любому элементу массива можно обратиться по его номеру. Однако механизм работы с массивами настолько гибок, что иногда вы можете не знать, сколько элементов содержится в массиве. В этом случае на помощь приходит функция **count()**. Эта функция сообщает нам количество элементов массива.

Пример 6 Получение размера массива

```
<?php
$student["name"] = "Ann";
```

```
$student["surname"] = "Petrova";
$student["age"] = 20;
printcount($student);
?>
```

В данном примере мы сначала создаем массив $student, а затем просим вывести на экран браузера результат работы функции count(). В качестве аргумента функции используем массив $student, количество элементов которого необходимо подсчитать. Результатом работы программы является появление на экране браузера числа 3.

Просмотр массива с помощью цикла foreach

Очень часто требуется вывести на экран браузера все элементы массива. Для выполнения этой операции лучше всего использовать инструкцию foreach.

Пример 7 Просмотр массива с помощью инструкции foreach

```
<?php
$students = array("Ira","Vadim","Alex","AnnM);
foreach($students as $temp)
    {
print "$temp<br>";
}
?>
```

В данном примере мы сначала создаем массив $students, состоящий из четырех элементов. Затем мы используем инструкцию foreach. В данном случае $students – это имя массива, который нужно просмотреть, а $temp – переменная, где будет временно храниться значение каждого элемента. Инструкция foreach работает следующим образом: сначала значение каждого элемента массива временно помещается в переменную $temp, а потом выводится на печать с помощью функции print, содержащейся в теле (между фигурных скобок) инструкции foreach. Элементы массива перебираются последовательно, один за другим. Результатом работы данной программы будет вывод на экран браузера следующего:

Ira
Vadim
Alex
An4n

Просмотр в цикле ассоциированного массива

Для того, чтобы просмотреть в цикле ассоциированный массив, нужно написать инструкцию foreach несколько по-другому. Дело в том, что нам придется временно сохранять не только значение каждого элемента, но и его имя.

Пример 8 Просмотр ассоциированного массива

```
<?php
```

```
$tel = array(
        "Max Koshelev" => "580-46-82",
"Ann Reish"=> "589-90-34",
"Pashsa Golikov" => "480-57-58"
            );
foreach ($tel as $key=>$temp);
{
        print"$key - $temp<br>";
}
?>
```

Здесь $tel – это имя массива, $key – переменная (ключ), в которой сохраняется имя каждого элемента массива, а $temp – переменная, где временно сохраняются значения каждого элемента.

Вывод этой программы выглядит следующим образом:

Max Koshelev - 580-46-82
Ann Reish - 589-90-34
Pashsa Golikov - 480-57-58

3. Сортировка массива

Сортировка простого массива с помощью функции sort()

Сортировка данных (т.е. расположение данных в некотором специальном порядке, например, по возрастанию или убыванию) является одним из наиболее важных применений компьютера.

Функция sort() принимает всего один аргумент – массив – и сортирует его в алфавитном порядке, если хотя бы один, из числа его элементов является строкой, и в числовом порядке, если все его элементы - числа. Эта функция преобразует переданный массив.

Пример 9 Сортировка простого массива с помощью функции sort()

```
<?php
$abc = array("c", "a", "b");
sort($abc);
foreach($abc as $temp)
{
print"$temp<br>";
}
?>
```

Результатом работы программы будет вывод на экран элементов отсортированного массива $abc.

a
b
c

Функция sort() изменяет положение элементов внутри нашего массива. Если до начала сортировки при запросе элемента массива с нулевым индексом вы получили бы значение **c**, то после применения функции sort() результатом на аналогичный запрос будет значение **a**. Сортировка массива в обратном порядке выполняется функцией rsort().

Сортировка ассоциированного массива

Сортировка ассоциированного массива производится аналогично, только используются функции asort() (сортировка по значениям в порядке возрастания) и ksort() (сортировка по ключам). Для сортировки в обратном порядке используются функции arsort() и krsort() соответственно.

Варианта заданий

Вариант 1

1. Подготовить текст программы, выполняющей следующие действия.

- Создать список (индексированный массив), состоящий из 5-ти наименований товаров с помощью функции **array()** (см. пример 1).
- Добавить еще не менее двух элементов массива с помощью идентификатора массива (см. пример 2).
- Определить количество элементов массив, используя функцию **count()**, и вывести названия товаров в цикле **for.**

2. Протестировать работу программы с различным количеством элементов массива.

3. Модифицировать программу, добавив сортировку массива в алфавитном порядке наименований товаров (использовать функцию **sort**). Вывести на экран исходный массив и результат сортировки.

Вариант 2

1. Подготовить программу для обработки ассоциативного массива.

Программа должна обеспечивать следующее.

- Создать ассоциативный массив: ТОВАР => ЦЕНА. Где название товара – это ключ (индекс) массива, а цена – значения элементов массива.
- Массив должен содержать не мене 5-ти элементов, три из них задать с помощью функции array(), а остальные задать непосредственно в операторе присваивания (см. примеры 4 и 5).
- Вывести товары и цены, используя оператор цикла **foreach().**

2. Протестировать работу программы с различным количеством элементов массива, добавив их любым способом.

3. Модифицировать программу для решения следующих задач.

- Подсчитать количество товаров и их суммарную стоимость.

4. Отсортировать массив:

- В порядке убывания (возрастания) цены товара и вывести на экран (использовать функции asort()/arsort()).

Вариант 3

1. Подготовить программу для обработки ассоциативного массива.

Программа должна обеспечивать следующее.

- Создать ассоциативный массив: ТОВАР => ЦЕНА. Где название товара – это ключ (индекс) массива, а цена – значения элементов массива.
- Массив должен содержать не мене 5-ти элементов, три из них задать с помощью функции array(), а остальные задать непосредственно в операторе присваивания (см. примеры 4 и 5).
- Вывести товары и цены, используя оператор ла **foreach()** (см. пример 8).

2. Протестировать работу программы с различным количеством элементов массива, добавив их любым способом.

3. Модифицировать программу для решения следующих задач.

- Подсчитать количество товаров и их суммарную стоимость.

4. Отсортировать массив:

- Выполнить сортировку массива так, чтобы товары расположились в алфавитном порядке для чего использовать функции или ksort()/krsort()

Варианты заданий

Распечатать на экране монитора таблицу символов, используя только циклы (без использования массивов)

Варинт 1
A D G J M
M D G J M
M J G J M
M J G D M
M J G D A

Вариант 2
A B C D E F G H I
B C D E F G H
C D E F G
D E F
E

Вариант 3
I
I G
I G E
I G E C
I G E C A

Вариант 4
A
B C
D E F
G H I J
K L M N O

Вариант 5

A B C D E
Z B C D E
Y Z C D E
X Y Z D E
W X Y Z E

Вариант 6
A B C D E
B C D E
C D E
D E
E

Вариант 7
A B C D E
E A C D E
E D A D E
E D C A E
E D C B A

Вариант 8
A E D C B
B E D C
C E D
D E
E

Вариант 9
A
B A B
C B A B C
D C B A B C A
C B A B C
B A B
A

Вариант 10
F
E F
D E F
C D E F
B C D E F
A B C D E F

Вариант 11
F E D C B A
E F E D C B
D E F E D C
C D E F E D
B C D E F E
A B C D E F

Вариант 12
F
F E
F E D
F E D C
F E D C B
F E D C B A

Вариант 13
A B C D E
E D C B A
B A C D E
E D C B A
A B C D E

Вариант 14
A B C D E
A B C D
A B C
A B
A

Вариант 15

A
A B A
A B C B A
A B C D C B A

A B C B A
A B A
A

Содержание отчета

Отчет должен содержать следующие пункты:

1. Титульный лист;
2. Цель работы;
3 Вариант задания;
4. Ход выполнения работы;
5. Результаты работы;
6. Вывод.

ЛАБОРАТОРНАЯ РАБОТА №7
Файлы на PHP

1. Цель работы

Целью работы является изучение основ языка PHP. Рассмотреть основные принципы работы с файлами на PHP.

2. Теоретический материал

Проверка существования файла и определение его размера.

Прежде чем пытаться работать с файлом, необходимо убедиться в том, что он существует. Для решения этой задачи обычно используются две функции:

file_exists( ) uis_file( ).
file_exists( )

Функция file_exists ( ) проверяет, существует ли заданный файл. Если файл существует, функция возвращает TRUE, в противном случае возвращается FALSE. Синтаксис функции file_exists( ):

boolfile_exists(string файл)

Пример проверки существования файла:

```
if (! file_exists ($filename)) :
print "File $filename does not exist!";
endif:
```

is_file( )

Функция is_file( ) проверяет существование заданного файла и возможность выполнения с ним операций чтения/записи. В сущности, is_file( ) представляет собой более надежную версию file_exists( ), которая проверяет не только факт существования файла, но и то, поддерживает ли он чтение и запись данных:

boolis_file(string файл)

Следующий пример показывает, как убедиться в существовании файла и возможности выполнения операций с ним:

```
$file = "somefile.txt";
if (is_file($file)) :
print "The file $file is valid and exists!";
else :
```

print "The file $file does not exist or it is not a valid file!";
endif:

Убедившись в том, что нужный файл существует и с ним можно выполнять различные операции чтения/записи, можно переходить к следующему шагу -- открытию файла.

filesize( )

Функция filesize( ) возвращает размер (в байтах) файла с заданным именем или FALSE в случае ошибки. Синтаксис функции filesize( ):

intfilesize(stringимя_файла)

Предположим, вы хотите определить размер файла pastry.txt. Для получения нужной информации можно воспользоваться функцией filesize( ):

$fs = filesize("pastry.txt"); print "Pastry.txt is $fs bytes.";

Выводится следующий результат:
Pastry.txt is 179 bytes.

Прежде чем выполнять операции с файлом, необходимо открыть его и связать с файловым манипулятором, а после завершения работы с файлом его следует закрыть.

Открытие и закрытие файла

Прежде чем выполнять операции ввода/вывода с файлом, необходимо открыть его функцией fopen( ).

fopen( )

Функция fopen( ) открывает файл (если он существует) и возвращает целое число -- так называемый *файловый манипулятор* (filehandle). Синтаксис функции fopen( ):

intfopen (string файл, string режим [, intвключение_пути])

Открываемый файл может находиться в локальной файловой системе, существовать в виде стандартного потока ввода/вывода или представлять файл в удаленной системе, принимаемой средствами HTTP или FTP.

Параметр файл может задаваться в нескольких формах, перечисленных ниже:

- Если параметр содержит имя локального файла, функция fopen( ) открывает этот файл и возвращает манипулятор.
- Если параметр задан в виде php://stdin, php://stdout или php://stderr, открывается соответствующий стандартный поток ввода/вывода.
- Если параметр начинается с префикса http://, функция открывает подключение HTTP к серверу и возвращает манипулятор для указанного файла.
- Если параметр начинается с префикса ftp://, функция открывает подключение FTP к серверу и возвращает манипулятор для указанного файла. В этом случае следует обратить особое внимание на два обстоятельства: если сервер не поддерживает пассивный режим FTP, вызов fopen() завершается неудачей. Более того, FTP-файлы открываются либо для чтения, либо для записи.

При работе в пассивном режиме сервер ожидает подключения со стороны клиентов. При работе в активном режиме сервер сам устанавливает соединение с клиентом. По умолчанию обычно используется активный режим.

Параметр режим определяет возможность выполнения чтения и записи в файл. В табл. 7.1 перечислены некоторые значения, определяющие режим открытия файла.

Таблица 7.1. *Режимы открытия файла*

Режим	Описание
r	Только чтение. Указатель текущей позиции устанавливается в начало файла
r+	Чтение и запись. Указатель текущей позиции устанавливается в начало файла
w	Только запись. Указатель текущей позиции устанавливается в начало файла, а все содержимое файла уничтожается. Если файл не существует, функция пытается создать его
w+	Чтение и запись. Указатель текущей позиции устанавливается в начало файла, а все содержимое файла уничтожается. Если файл не существует, функция пытается создать его
a	Только запись. Указатель текущей позиции устанавливается в конец файла. Если файл не существует, функция пытается создать его
a+	Чтение и запись. Указатель текущей позиции устанавливается в конец файла. Если файл не существует, функция пытается создать его

Если необязательный третий параметр включение_пути равен 1, то путь к файлу определяется по отношению к каталогу включаемых файлов, указанному в файле php.ini.

Ниже приведен пример открытия файла функцией fopen(). Вызов die(), используемый в сочетании с fopen(), обеспечивает вывод сообщения об ошибке в том случае, если открыть файл не удастся:

```
$file = "userdata.txt"; // Некоторыйфайл
$fh = fopen($file, "a+") or die("File ($file) does not exist!");
```

Следующий фрагмент открывает подключение к сайту PHP (http://www.php.net):

```
$site = "http://www.php.net": // Сервер, доступный через HTTP
$sh = fopen($site., "r");     //Связать манипулятор с индексной страницей Php.net
```

После завершения работы файл всегда следует закрывать функцией fclose( ).

```
fclose ( )
```

Функция fclose( ) закрывает файл с заданным манипулятором. При успешном закрытии возвращается TRUE, при неудаче -- FALSE. Синтаксис функции fclose( ):

```
intfclose(int манипулятор)
```

Функция fclose( ) успешно закрывает только те файлы, которые были ранее открыты функциями fopen( ) или fsockopen( ). Примерзакрытияфайла:

```
$file = "userdata.txt";
if (file_exists($file)) :
$fh = fopen($file, "r");
// Выполнить операции с файлом
fclose($fh);
else :
print "File Sfile does not exist!";
endif;
```

Запись в файл

С открытыми файлами выполняются две основные операции -- чтение и запись.

is_writeable( )

Функция is_writeable( ) позволяет убедиться в том, что файл существует и для него разрешена операция записи. Возможность записи проверяется как для файла, так и для каталога. Синтаксис функции is_writeable( ):

boolis_writeable (string файл)

Одно важное обстоятельство: скорее всего, PHP будет работать под идентификатором пользователя, используемым web-сервером (как правило, «nobody»). Пример использования is_writeable( ) приведен в описании функции fwrite( ).

fwrite ( )

Функция fwrite( ) записывает содержимое строковой переменной в файл, заданный файловым манипулятором. Синтаксис функции fwrite( ):

intfwrite(int манипулятор, string переменная [, int длина])

Если при вызове функции передается необязательный параметр длина, запись останавливается либо после записи указанного количества символов, либо при достижении конца строки. Проверка возможности записи в файл продемонстрирована в следующем примере:

```
<?
// Информация о трафике на пользовательском сайте
$data = "08:13:00|12:37:12|208.247.106.187|Win98";
$filename = "somefile.txt";
// Если файл существует и в него возможна запись
if( is_writeable($filename) ) :
// Открыть файл и установить указатель текущей позиции в конец файла
$fh = fopen($filename, "a+");
// Записать содержимое $data в файл
$ success - fwrite($fh, $data);
// Закрытьфайл
fclose($fh); else :
print "Could not open Sfilename for writing";
endif;
?>
```

Функция fputs( ) является псевдонимом fwrite( ) и может использоваться всюду, где используется fwrite( ).

fputs( )

Функция fputs( ) является псевдонимом fwrite( ) и имеет точно такой же синтаксис. Синтаксис функции fputs( ):

intfputs(int манипулятор, string переменная [, int длина])

Чтение из файла

Несомненно, чтение является самой главной операцией, выполняемой с файлами. Ниже описаны некоторые функции, повышающие эффективность чтения из файла. Синтаксис этих функций практически точно копирует синтаксис аналогичных функций записи.

is_readable( )

Функция i s_readable( ) позволяет убедиться в том, что файл существует и для него разрешена операция чтения. Возможность чтения проверяется как для файла, так и для каталога. Синтаксис функции is_readable( ):

boolis_readable (stringфайл]

Скорее всего, PHP будет работать под идентификатором пользователя, используемым web-сервером (как правило, «nobody»), поэтому для того чтобы функция is_readable( ) возвращала TRUE, чтение из файла должно быть разрешено всем желающим. Следующий пример показывает, как убедиться в том, что файл существует и доступен для чтения:

```
if( is_readable($filename) ) :
// Открыть файл и установить указатель текущей позиции в конец файла
$fh = fopen($filename, "r");
else :
print "$filename is not readable!";
endif;
```

Функция fread( ) читает из файла, заданного файловым манипулятором, заданное количество байт. Синтаксис функции fwrite( ):

intfread(intманипулятор, intдлина)

Манипулятор должен ссылаться на открытый файл, доступный для чтения (см. описание функции is_readable( )). Чтение прекращается после прочтения заданного количества байт или при достижении конца файла. Чтение и вывод этого файла в браузере осуществляется следующим фрагментом:

```
$fh = fopen('pastry.txt', "r") or die("Can't open file!");
$file = fread($fh, filesize($fh));
print $file;
fclose($fh);
```

Используя функцию fllesize( ) для определения размера pastry.txt в байтах, вы гарантируете, что функция fread( ) прочитает все содержимое файла.

Листинг 7.1. Текстовыйфайл pastry.txt

```
Recipe: Pastry Dough
1 1/4 cups all-purpose flour
3/4 stick (6 tablespoons) unsalted butter, chopped
2 tablespoons vegetable shortening 1/4 teaspoon salt
3 tablespoons water
```

Функция fgetc( ) возвращает строку, содержащую один символ из файла в текущей позиции указателя, или FALSE при достижении конца файла. Синтаксис функции fgetc( ):

stringfgetc (int манипулятор)

Манипулятор должен ссылаться на открытый файл, доступный для чтения (см. описание функции is_readable( ) ранее в этой главе). В следующем примере продемонстрированы посимвольное чтение и вывод файла с использованием функции fgetc( ):

```
$fh = fopen("pastry.txt", "r"); while (! feof($fh)) :
$char = fgetc($fh):
print $char; endwhile;
fclose($fh);
```

Функция fgets( ) возвращает строку, прочитанную от текущей позиции указателя в файле, определяемом файловым манипулятором. Файловый указатель должен ссылаться на открытый файл, доступный для чтения

(см. описание функции is_readable( ) ранее в этой главе). Синтаксис функции fgets( ):

stringfgets (int манипулятор, int длина)

Чтение прекращается при выполнении одного из следующих условий:

- из файла прочитано длина -- 1 байт;
- из файла прочитан символ новой строки (включается в возвращаемую строку);
- из файла прочитан признак конца файла (EOF).

Если вы хотите организовать построчное чтение файла, передайте во втором параметре значение, заведомо превышающее количество байт в строке. Пример построчного чтения и вывода файла:

```
$fh = fopen("pastry.txt", "r");
while (! feof($fh));
$line = fgets($fh, 4096);
print $line. "<br>";
endwhile;
fclose($fh):
```

Функция fgetss( ) полностью аналогична fgets( ) за одним исключением -- она пытается удалять из прочитанного текста все теги HTML и PHP:

stringfgetss (Int манипулятор, int длина [, stringразрешенные_теги])

Прежде чем переходить к примерам, ознакомьтесь с содержимым листинга 7.2 -- этот файл используется в листингах 7.3 и 7.4.

Листинг 7.2. *Файл science.html*

```
<html>
<head>
<title>Breaking News - Science</title>
<body>
<h1>Alien lifeform discovered</h1><br>
<b>August 20. 2000</b><br>
Early this morning, a strange new form of fungus was found growing in the closet of W. J. Gilmore's old apartment refrigerator. It is not known if powerful radiation emanating from the tenant's computer monitor aided in this evolution.
</body>
</html>
```

Листинг 7.3. *Удаление тегов из файла HTML перед отображением в браузере*

```
<?
$fh = fopen("science.html", "r");
while (! feof($fh)) :
printfgetss($fh, 2048);
endwhile;
fclose($fh);
?>
```

Результат приведен ниже. Как видите, из файла science.html были удалены все теги HTML, что привело к потере форматирования:

Breaking News - Science Alien lifeform discovered August 20. 2000 Early this morning, a strange new form of fungus was found growing in the closet of W. J. Gilmore's old apartment refrigerator. It is not known if powerful radiation emanating from the tenant's computer monitor aided in this evolution.

В некоторых ситуациях из файла удаляются все теги, кроме некоторых -- например, тегов разрыва строк
. Листинг 7.4 показывает, как это делается.

Листинг 7.4. *Выборочное удаление тегов из файла HTML*

```
<?
$fh = fopenC'science.html", "r");
$allowable = "<br>";
while (! feof($fh)) :
printfgetss($fh. 2048, $allowable);
endwhile;
fclose($fh);
?>
```

Результат:

Breaking News - Science Alien lifeform discovered August 20. 2000 Early this morning, a strange new form of fungus was found growing in the closet of W. J. Gilmore's old apartment refrigerator. It is not known if powerful radiation emanating from the tenant's computer monitor aided in this evolution.

Как видите, функция fgetss( ) упрощает преобразование файлов, особенно при наличии большого количества файлов HTML, отформатированных сходным образом.

3. Порядок выполнения работы

Отображение файла

Для того, чтобы отобразить информацию, которая содержится в файле на страницу, необходимо выполнить следующие действия:

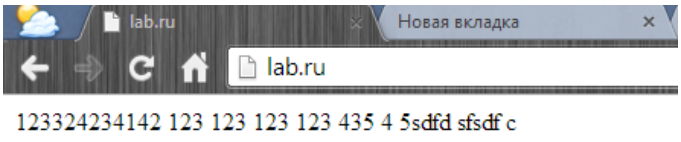
1. Создаться .txt файл в корневом каталоге (в котором лежит ваша страница). К примеру:"C:\WebServers\home\tsa.ru\www".

2. Отредактировать файл, добавив туда текст.

3. Открыть вашу страничку index.html и добавить в нее следующий код:

```
if(!file)
   {
echo("Ошибка открытия файла");
   }
else
   {
readfile("file.txt");
   }
```

4. Запустить вашу страницу и убедиться, что на ней отображен текст из файла.



Чтение из файла

Для того, чтобы считать информацию из файла нам необходимо воспользоваться функцией, которая считывает файл с именем **filename** и возвращает массив, каждый элемент которого соответствует строке в прочитанном файле. Для этого необходимо на нашей странице index.php прописать следующий код:

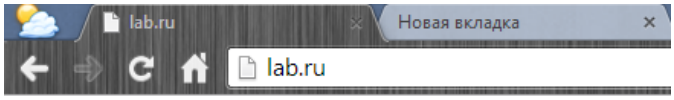
```
<?
 $file_array = file("file.txt");
if(!$file_array)
 {
echo("Ошибкаоткрытияфайла");
 }
else
 {
for($i=0; $i< count($file_array); $i++)
   {
printf("%s<br>", $file_array[$i]);
}
 }
?>
```

Также воспользуемся функцией count. Эта функция удобна также тем, что с ее помощью можно легко подсчитать количество строк в файле:

```
<?
 $file_array=  file("file.txt");
if(!$file_array)
 {
echo("Ошибкаоткрытияфайла");
}
else
 {
   $num_str=  count($file_array);
echo('Количествострок = ');
echo($num_str);
 }
?>
```

Заметим, что функцию file следует применять лишь для чтения небольших файлов.

Результат выполнения работы должен выглядеть следующим образом:



Унылая пора! Очей очарованье!
Приятна мне твоя прощальная краса —
Люблю я пышное природы увяданье,
В багрец и в золото одетые леса,
В их сенях ветра шум и свежее дыханье,
И мглой волнистою покрыты небеса,
И редкий солнца луч, и первые морозы,
И отдаленные седой зимы угрозы.
8

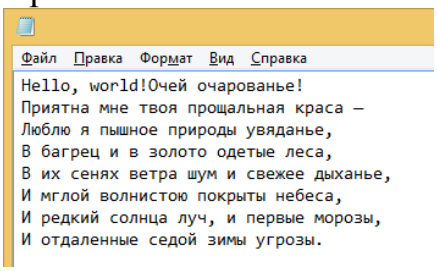
Запись в файл

Для того, что бы добавить информацию в файл нам необходимо воспользоваться функцией, которая записывает сроку в файл с именем **filename** . Для этого необходимо на нашей странице index.php прописать следующий код:

```
$file = fopen ("file.txt","r+");
$str = "Hello, world!";
if ( !$file )
 {
echo("Ошибкаоткрытияфайла");
 }
else
 {
```

```
fputs( $file, $str);
  }
fclose($file);
```

Результат выполнения работы:



5. Задание

Для выполнения лабораторной работы необходимо выполнить следующие действия над файлом:

1. Выполнить задания из теоретической части.
2. Определить существует ли файл?
3. Определить доступен ли файл для чтения?
4. Определить доступен ли фал для записи?
5. Определить размер файла в байтах?
6. Вывести данную информацию на страницу.
7. Прочитать из файла информацию. Вывести на экран
8. Записать в файл информацию.
9. Удалить одну строку из файла (на выбор преподавателя).

Варианты заданий.

Вариант 1

Примета осени - живое упорно тянется к теплу
Коричным плюшкам в сладкой корке, такой коричневой на вид
Молочным плошкам, мягким кошкам, шарфам и шалям шерстяным,
Верблюжьим пледам, берегущим полупустынную жару.

Вариант 2

Мы — облака, закрывшие луну…
Мы кружимся, и вертимся, и вьёмся…
Но, поблистав минуту лишь одну,
Уйдём во мрак и больше не вернёмся…

Мы — флейты, отзвучавшие давно,
Забытые… И в неверном строе
Мы отвечаем ветру то одно,
То, миг спустя, совсем уже другое…

Вариант 3

Идёт, сутулится старуха,
Волочит ноги по земле.
Вдруг дед навстречу: «Ты, Веруха?
Ты лучик солнца мой во мгле!
Да что ж глаза твои в печали?
Ведь ты любовь моя одна,
Меня-то годы потрепали,
Вон, видишь, — лезет седина.

Вариант 4

Про́за (лат. *prōsa*) — устная или письменная речь без деления на соизмеримые отрезки — стихи; в противоположность поэзии её ритм опирается на приблизительную соотнесенностьсинтаксических конструкций (периодов, предложений, колонов). Иногда термин употребляется в качестве противопоставления художественной литературы вообще (поэзия) литературе научной или публицистической, то есть не относящейся к искусству.

Содержание отчета

Отчет должен содержать следующие пункты:

1. Титульный лист;
2. Цель работы;
3 Вариант задания;
4. Ход выполнения работы;
5. Результаты работы;
6. Вывод.

Контрольные вопросы

1. Какой операцией можно проверить существует ли данный файл по указанному пути?
2. Какие существуют режимы открытия файла?
3. Прежде чем приступить к чтению данных из файла в чем надо убедиться? С помощью каких операций это можно сделать?
4. Прежде чем приступить к записи в файл в чем надо убедиться и с помощью каких операций это можно сделать?

Лабораторная работа № 8
PHP and MySQL

1.Цель работы

Изучить основы работы PHP с базами данных MySQL.

2.Базовые сведения

Что такое PHP?

PHP - язык создания сценариев, который давно перерос свое название. Дело в том, что PHP - это аббревиатура от слов Personal Home Page. Первая версия PHP была создана Расмусом Лердорфом в 1994 г. и представляла собой набор инструментов для отслеживания посетителей Web-страницы. Со временем PHP из набора инструментов превратился в полноценный язык программирования, а его название было изменено как рекурсивное образование PHP HyperText Preprocessor (препроцессор гипертекста PHP).

PHP - это серверный язык создания сценариев. Конструкции PHP, вставленные в HTML-текст, выполняются сервером при каждом посещении страницы. Результат их обработки вместе с обычным HTML-текстом передается браузеру.

В настоящее время основной версией PHP является пятая.

Существуют два основных конкурента PHP: Active Server Pages (ASP) компании Microsoft и ColdFusion компании Allaire. По сравнению с ними PHP обладает рядом преимуществ, в числе которых:

- **Высокая производительность.** PHP-программы работают быстрее, чем ASP.
- **Функциональность.** Разработку PHP-программы можно отделить от собственно разработки Web-страницы, что упростит жизнь и программисту, и дизайнеру.
- **Цена.** PHP абсолютно бесплатен.
- **Простота в использовании.** Имеющие опыт программирования на распространенных языках найдут синтаксис PHP хорошо знакомым.
- **Переносимость.** Один и тот же PHP-код можно использовать как в среде NT, так и на платформах UNIX.

Описание MySQL

MySQL — свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU General Public License, так и под собственной

коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей, именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типаMyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

3. Выполнение работы

3.1. Рассмотрение теоретического примера

Подключение к серверу MySQL

Для начала работы с mysql необходимо подключиться к этому самому серверу БД MySQL. Делается это следующей командой:

```
$link = mysql_connect('host','user','pass');
```

где соответственно host - имя хоста (по умолчанию localhost), user - имя пользователя (по умолчанию root) и pass - пароль (по умолчанию пустая строка - '').

Выбор базы данных для работы

Далее необходимо выбрать собственно саму базу, с которой мы будем работать, это делается функцией

```
mysql_select_db('bd_name');
```

Где мы указываем нашу базу соответственно.

Установка кодировки

Далее, для правильной кодировки при работе с данными - чтобы не выводились "кракозябры" вместо русских букв нужно установить кодировку для соединения и работы:

```
mysql_set_charset('utf8');
```

Внимание, важный момент! Все скрипты, таблицы, и сама база должны быть в одной кодировке!!!
Иначе вы получите головную боль при использовании русских символов!

Выполнение запросов

Идем дальше. Мы соединились с сервером, выбрали базу данных для работы, установили нужную нам кодировку.
Теперь нам необходимо произвести необходимые нам в данном случае операции с данными, которые у нас хранятся в нашей базе данных.
Запросы выполняются с помощью функции mysql_query():

mysql_query("Текст запроса");

//выбор записей

mysql_query(" SELECT * FROM `fruits` ",$link);

//вставка данных

mysql_query(" INSERT INTO `fruits` (`category`) VALUES ('киви') ",$link);

//обновление записей

mysql_query(" UPDATE `fruits` SET `category`='груши' WHERE `id`='10' ",$link);

проверить способов есть вполне достаточно, самый распространенный - дописать вывод ошибки и прерывание дальнейшей работы скрипта:

mysql_query("запрос",$link) or die("ERROR: ".mysql_error());

Разберем чуть подробнее саму функцию mysql_query() и что она возвращает.
Вы спросите, зачем я писал $link после запроса. Это указатель на нужное соединение. Он не обязательный, если его не указать, то функция будет пытаться использовать последнее открытое соединение.

Теперь о возвращаемых значениях. Для запросов SELECT, SHOW, EXPLAIN и DESCRIBE функция возвращает указатель на результат запроса при успехе, при неудаче - возвращает FALSE. Для всех остальных запросов (INSERT,UPDATE, и т.д.) функция возвращает TRUE в случае успеха и FALSE в случае ошибки.

Для вывода полученных данных при запросе SELECT, SHOW, EXPLAIN или DESCRIBE необходимо обработать этот самый указатель для этого используется fetch-функции. Две самые распространенные из них - это mysql_fetch_assoc() и mysql_fetch_array(). Первая возвращает ассоциатив-

ный массив, вторая несколько массивов сразу (если не нужны числовые индексы и т. п., то рекомендую использовать первую).

3.2 Рассмотрение практического примера

Итак, например в доступной для Вас базе данных MySQL «hotels» есть некая таблица с перечнем названий тур фирм, их даты создания (Рисунок 1), и Вы хотите вывести эти данные на страницу сайта. Решить эту задачу довольно просто. Для начала предположим, что в нужной нам таблице MySQL определены следующие поля:

```
mysql> SELECT * from firms;
+----+---------+------------+
| id | name    | datafuond  |
+----+---------+------------+
|  1 | SunRise | 2010-04-12 |
|  2 | Exotic  | 2012-05-24 |
+----+---------+------------+
```

Рис 1. Содержание таблицы «firms».

```
// Соединение с базой данных.
<?php
$link = mysqli_connect('localhost','root','root','hotels');

//Проверка соединения.
if(mysqli_connect_errno()) die('Ошибка соединения:
'.mysqli_connect_error());
else {
print ('Успешно установлено!!!');

//Создаем таблицу на PHP, в которую будем выводить данные
   echo '<table border="1">';
   echo '<thead>';
   echo '<tr>';
   echo '<th>№</th>';
   echo '<th>Название</th>';
   echo '<th>Дата создания</th>';
   echo '</tr>';
   echo '</thead>';
   echo '<tbody>';

//Делаем запрос к нашей БД
$res = mysqli_query($link,"SELECT * FROM `firms`");

//Заполняем таблицу, созданную выше
if($res) {
while($row = mysqli_fetch_assoc($res)) {
      echo '<tr>';
      echo '<td>' . $row['id'] . '</td>';
      echo '<td>' . $row['name'] . '</td>';
      echo '<td>' . $row['datafuond'] . '</td>';
      echo '</tr>';
```

```
}
echo '</tbody>';
echo '</table>';

mysqli_free_result($res); //очищаем занятую память - она уже
не нужна
}

//Закрываем соединение
mysqli_close($link);
}
?>
```

В результате выполнения приведенного в пример PHP скрипта мы получим на веб-странице примерно следующее:

Соединение установлено успешно!!!

№	Название	Дата создания
1	SunRise	2010-04-12
2	Exotic	2012-05-24

Рис 2. Вывод на страницу результата запроса

Таким образом можно выводить на страницу HTML любые данные из любой таблицы MySQL, главное, чтобы у Вас был доступ к базе данных.

4. Контрольные вопросы

1. К кому виду языков программирования относится PHP?
2. С какими видами таблиц работает MySQL?
3. Производитель СУБД MySQL?
4. Основатель PHP?

5. Варианты индивидуальных заданий

1. Выполнить подключение и вывод данных из БД «Гостиница».
2. Выполнить подключение и вывод данных из БД «Автосалон».
3. Выполнить подключение и вывод данных из БД «Деканат».
4. Выполнить подключение и вывод данных из БД «Больница».
5. Выполнить подключение и вывод данных из БД «Салон красоты».

6. Рекомендуемая литература

1. http://www.cyberforum.ru/php-database/thread639162.html#a_mysql начало
2. http://htmlweb.ru/php/mysql.php

ЛАБОРАТОРНАЯ РАБОТА №9
Установка Joomla

http://домен.ru/

Когда все файлы будут закачаныв домен.ru, можно продолжать.

Установка Joomla происходит посредством интернет-браузера (рекомендуется использовать браузер Firefox)

Теперь нужно набрать в адресной строке адрес ресурса (доменное имя Вашего сайта куда были закачаны файлы Joomla) и нажать клавишу «Enter» на клавиатуре, это запустит веб-инсталлятор системы и начнется установка.

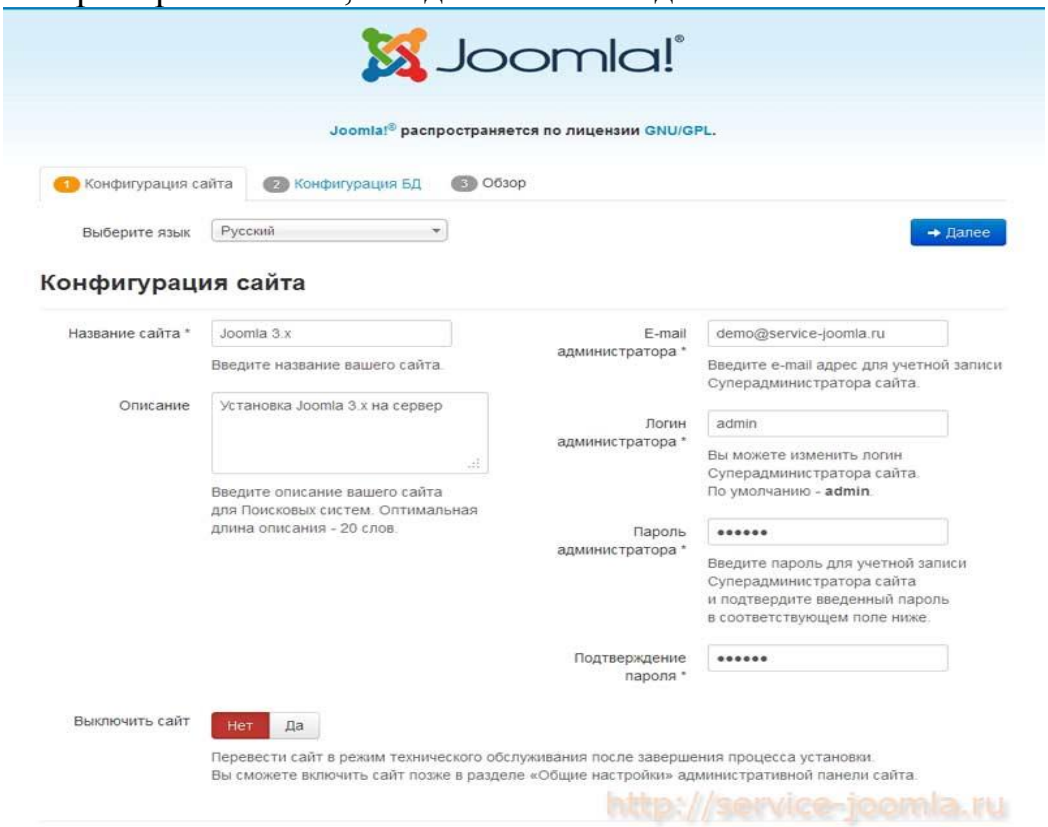
Минимальные требования к серверу:

-1 база данных MySQL
-PHP 5.3.1 или выше
- MySQL 5.1 или более поздней версии
- Дисковое пространство < 50MB

Шаг 1: Конфигурация сайта

Все следующие поля ввода должны быть правильно заполнены.

На примере показано, как должны выглядеть заполненные поля.



Название вашего сайта – обязательное поле.

Описание сайта – необязательное поле (заполняется для поисковых систем).

Электронная почта администратора – обязательно поле (введите в это поле свой e-mail)

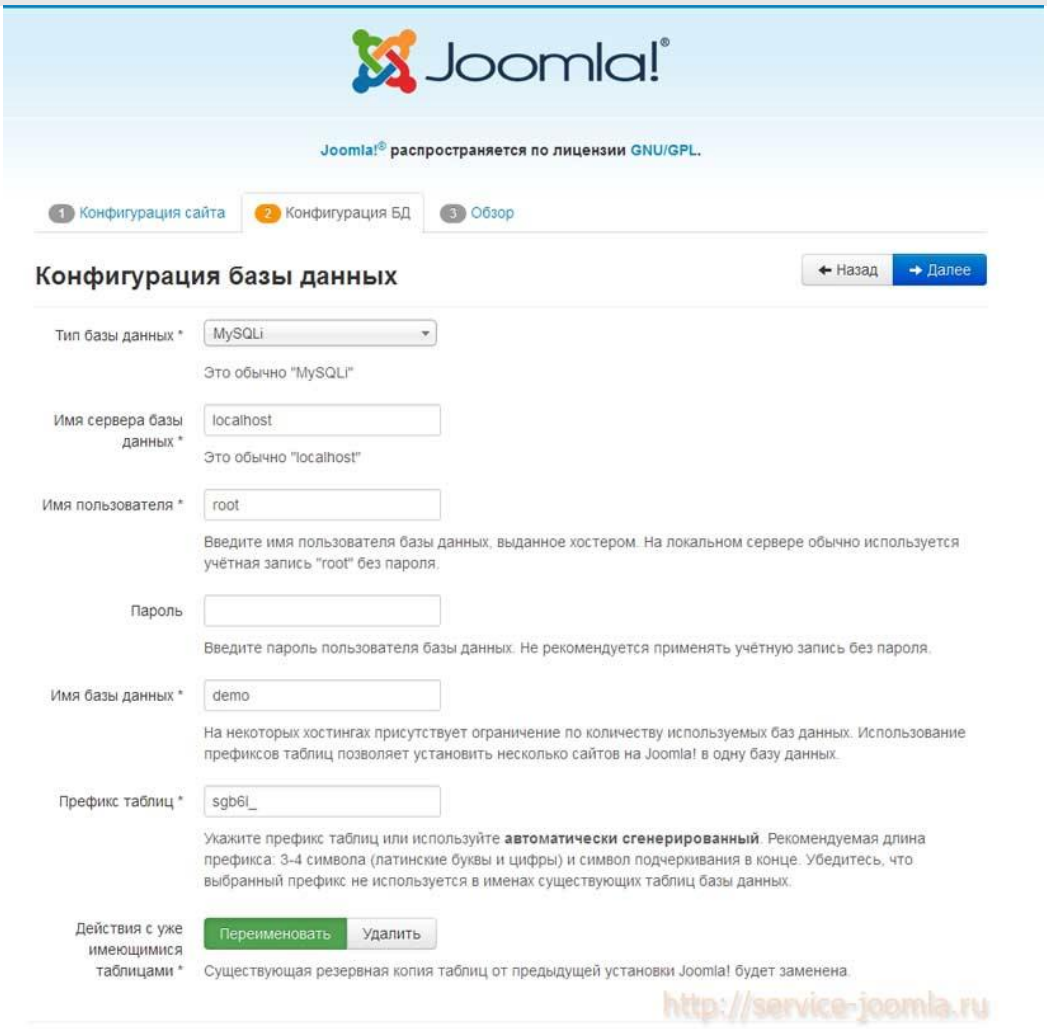
Логин администратора – обязательное поле, по умолчанию admin, (если Jooma устанавливается на хостинг, рекомендуется изменить)

Пароль администратора – обязательное поле (используется для входа в админку сайта)

Повторите пароль еще раз – обязательное поле

Нажмите «Далее» в верхней части страницы для перехода к следующему шагу.

Шаг 2: Конфигурация базы данных.



На этом шаге Вам нужно настроить подключение к базе данных для Joomla.

Тип базы данных - Здесь нужно выбрать какую базу данных использовать для подключения (обычно это значение используется по умолчанию и не изменяется).

Имя сервера базы данных - Если сервер базы данных находится вместе с веб-сервером, то название хоста будет «localhost», если нет, то эти параметры уточняются у службы поддержки Вашего хостинга, (обычно

хостинг предоставляет все эти данные когда вы приобретаете у него службу). Вводить параметры нужно точно, так как Joomla будет подключаться к базе данных, установить Joomla без них не получится.

Имя Пользователя - Здесь указывается имя пользователя базы данных (выдается хостинг-провайдером или самостоятельно устанавливается в панели хостинга.Вы всегда все эти данные можете узнать у службы поддержки Вашего хостинга).

Для установки на Денвер в поле имя пользователя вводится имя пользователя: root

Пароль - Введите надежный пароль

Для установки на Денвер ввод пароля не требуется (оставьте пустым)

Имя базы данных - Полное название базы данных на хостинге (выдается хостинг-провайдером или самостоятельно устанавливается в панели хостинга при создании базы данных.Вы всегда можете уточнить эти данные у службы поддержки Вашего хостинга).

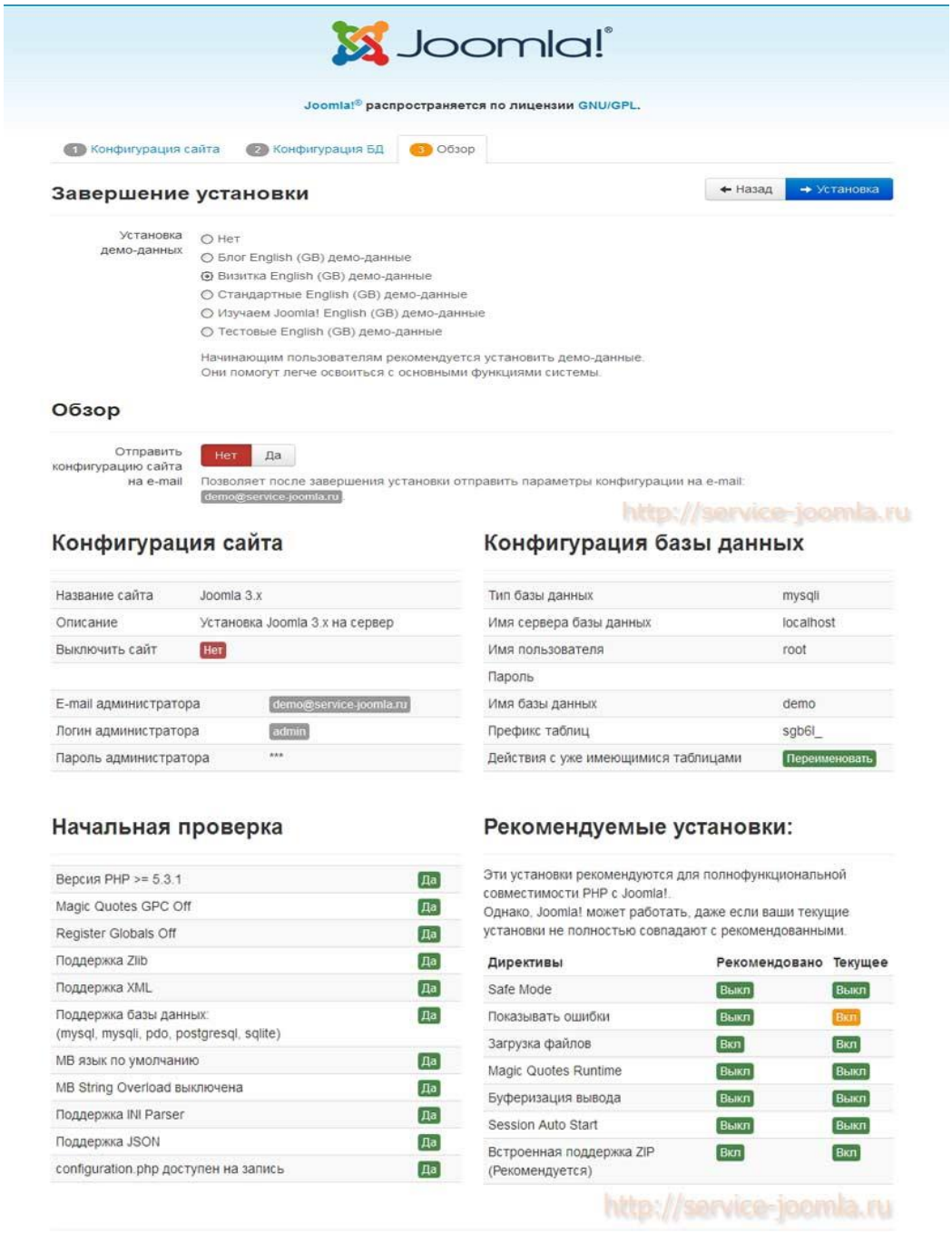
Для установки Joomla на Денвер создавать базу данных не требуется. Вводите в это поле любое название, например, joomla3 и база данных автоматически будет создана.

Префикс для таблиц – изменять префикс таблиц необязательно (если у Вас уже используется эта база данных для другого сайта, тогда Вы можете задать свой префикс для таблиц Joomla 3, например, j30_)

Действие с уже имеющимися таблицами - используется если у Вас уже используется или использовалась ранее эта база данных – переименовать или удалить все существующие таблицы из базы данных.

Нажмите «Далее» в верхней части страницы для перехода к 3 шагу.

Шаг 3: Обзор — Завершение установки.



Завершающий шаг установки Joomla 3.x

Установка Демо-данных – Если Вы ранее не работали с Joomla прежних версий, тогда рекомендуется установить демо данные. В примере выбраны демо данные сайта визитки. Если выбрать нет, тогда сайт будет пустой (без статей и т.д.).

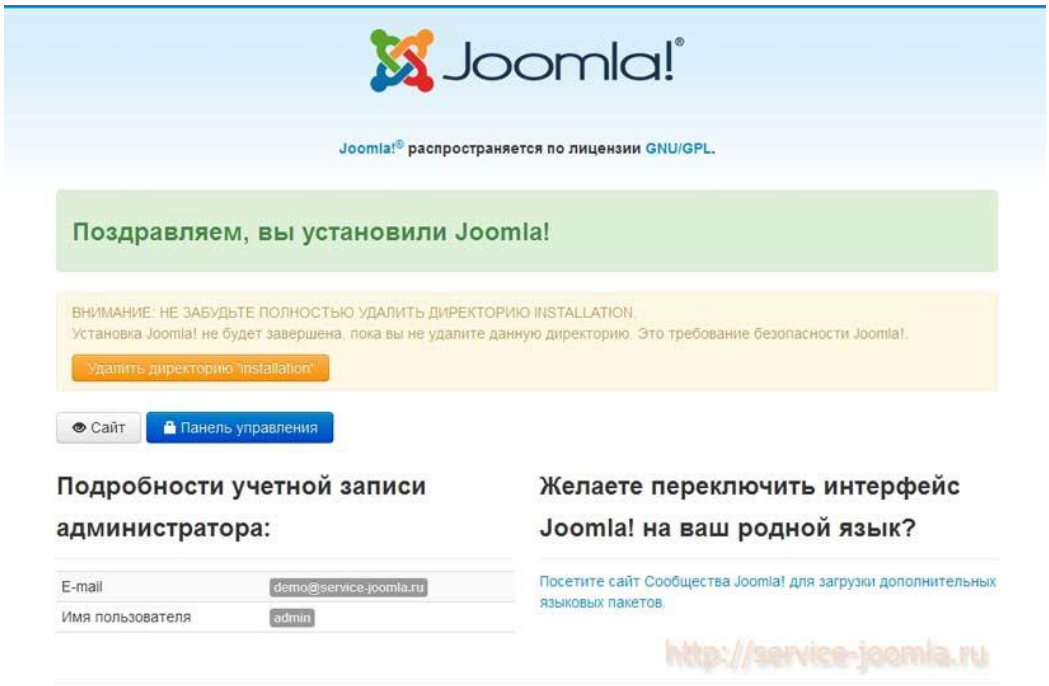
Примечание: Демо данные в основном используется для новичков и при выборе каких либо демо данных на их примере можно смотреть как организован контент на сайте.

Обзор

Отправить конфигурацию сайта на email - выберите «Да». И на ваш email будет отправлена конфигурация сайта. (эта функция не работает на Денвере)

Далее на этой странице инсталлятор Joomla покажет Вам конфигурационные данные и результаты проверки на совместимость с Вашим сервером и рекомендуемые установки настроек сервера.

Нажмите «Установка» в верхней части страницы для завершения установки Joomla 3.x



Поздравляем, Вы установили Joomla!

Если Вы видите это сообщение, значит установка прошла успешно и теперь Вы должны удалить директорию установки, это папка installation - которая находится в корне сайта и она должна быть удалена из соображений безопасности, чтобы помешать кому-либо еще раз запустить установку. Нажмите на кнопку: Удалить директорию 'installation' и папка будет удалена, об этом Вы узнаете, когда текст кнопки изменится на: Директория 'installation' успешно удалена

Далее нажмите на кнопку Панель управления

Нажав кнопку Панель управления, откроется страница входа в административный раздел. Для входа в админку введите: логин и пароль администратора которые Вы указали на первом шаге.

Далее вход в Панель управления будет происходить по ссылке:

http://ваш домен/administrator

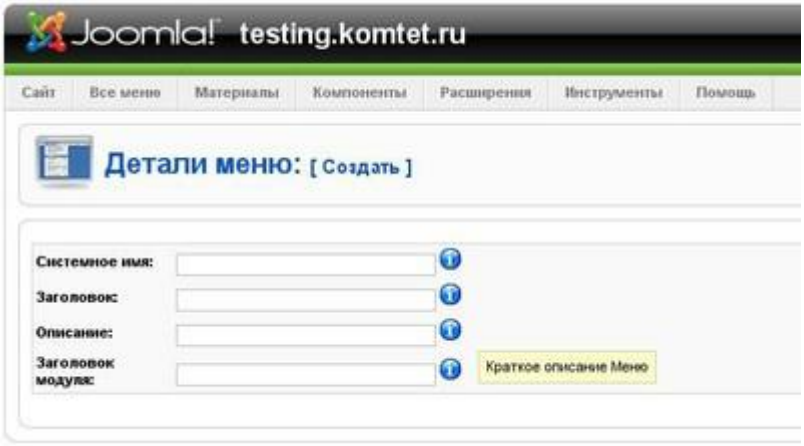
ЛАБОРАТОРНАЯ РАБОТА №8
Создание меню

Цель работы: нац=учится создавать меню.

MainMenu, UserMenu и TopMenu используются большинством шаблонов для Joomla по умолчанию. Поэтому дабы избежать дополнительной работы по изменению кода шаблонов, просто переименуем эти три меню на Главное меню, Пользовательское меню и Верхнее меню.

Теперь приступим непосредственно к созданию меню. Например, нам надо помимо трех существующих, добавить еще одно - Скачать на сайте. Создается меню так:

- Заходим в Все меню -> Менеджер меню
- Нажимаем кнопку  Создать в правой части.
- В появившемся окне задаем параметры нового меню:



Системное имя (обязательно на английском): *filemenu*

Заголовок (название меню на сайте): *Скачать на сайте*

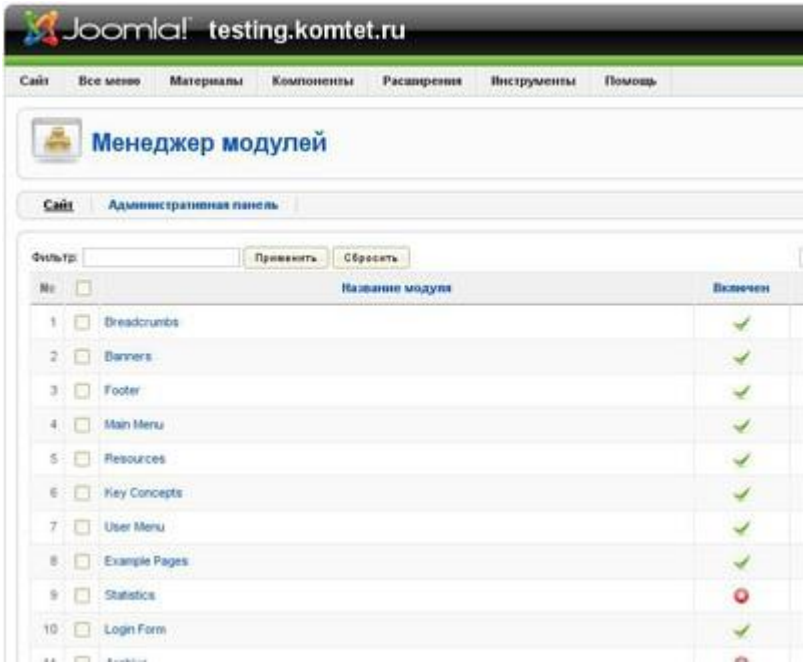
Описание (необязательный параметр): *здесь качаем файлы*

Заголовок модуля (название модуля для данного меню): *mod_mainmenu*

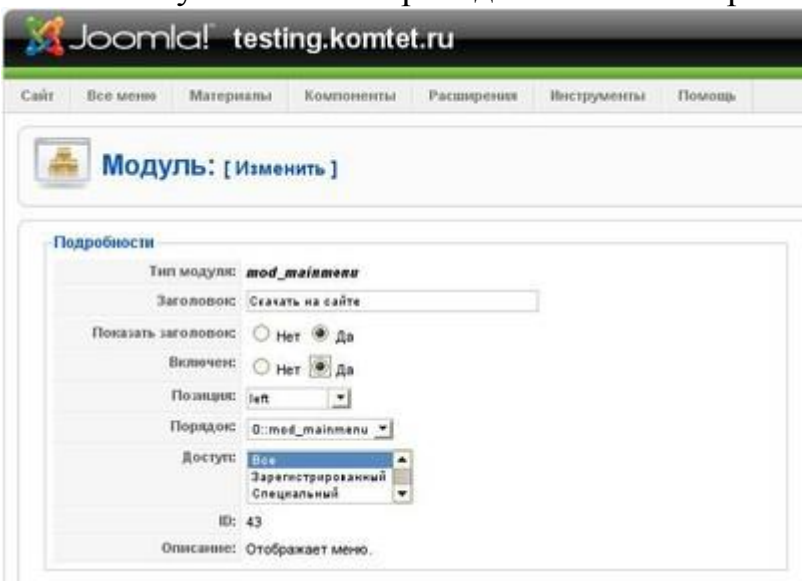
Очень важно заметить что в некоторых версиях Joomla если заголовок модуля меню назвать как то иначе, он не отобразиться на сайте, даже если его опубликовать. Поэтому присваиваем имямod_mainmenu, а потом переименуем его в менеджере модулей.

Итак, меню и модуль данного меню созданы. Чтобы меню появилось на сайте, необходимо опубликовать его. Делается это следующим образом:

- Заходим в Расширения -> Менеджер модулей



- Выбираем модуль с названием mod_mainmenu
- Щелкаем по нему мышью и переходим в окно настройки модуля



- Теперь изменяем имя mod_mainmenu на Скачать на сайте
- В полях Показать заголовок: и Включен: ставим параметр Да
- Нажимаем кнопку  Сохранить

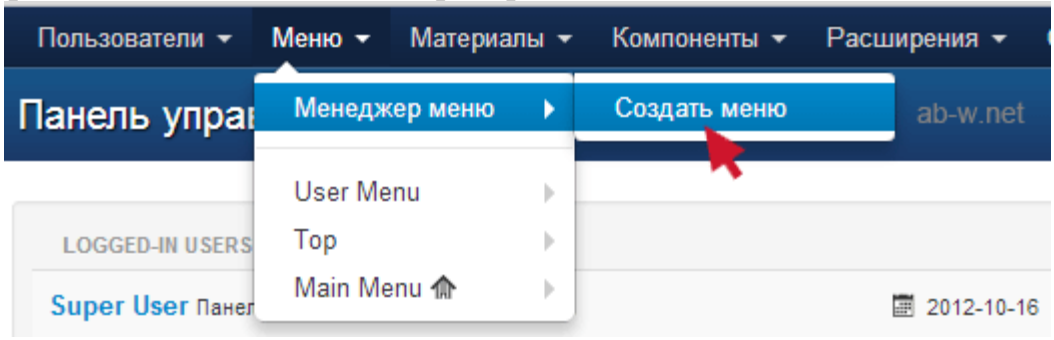
В принципе меню Скачать на сайте уже активно на странице сайта и ждет наполнения, но будет отображаться в списке всех меню на последнем месте. Если Вы хотите поставить его на какой либо другой позиции делаем следующее:

- Заходим снова в Расширения -> Менеджер модулей
- В столбце Порядок рядом с каждым модулем мы видим различные числовые значения.вообщем то это и есть приоритет каждого модуля на сайте (от 1 и ниже, причем 1 является наивысшим приоритетом). Так как

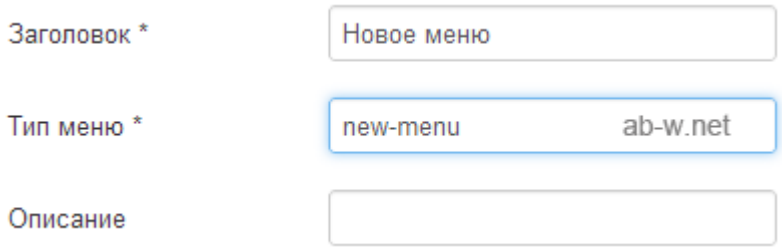
нам необходимо поставить наше меню Скачать на сайте после менюГлавное меню (приоритет 1), мы изменяем значение Порядок меню Скачать на сайте на 2. Теперь это меню будет следовать на панели всех меню ровно после Главного меню.

Инструкция по управлению сайтом CMS Joomla

1. Открываем в панели администратора сайта Меню → Создать меню:



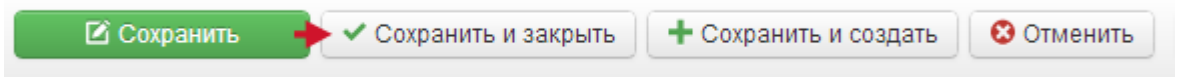
2. На открывшейся странице менеджера админки прописываем заголовок и тип:



Заголовок – понятными буквами прописываем произвольный заголовок.

Тип – латинскими буквами прописываем произвольное имя, системный псевдоним.

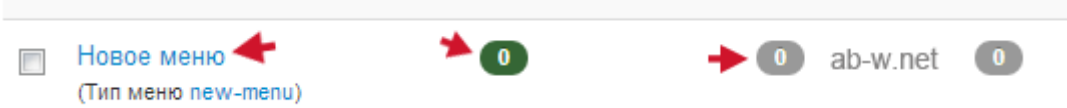
3. Вверху панели управления жмем Сохранить и закрыть:



В результате нам откроется список со всеми существующими на данный момент менюшками.

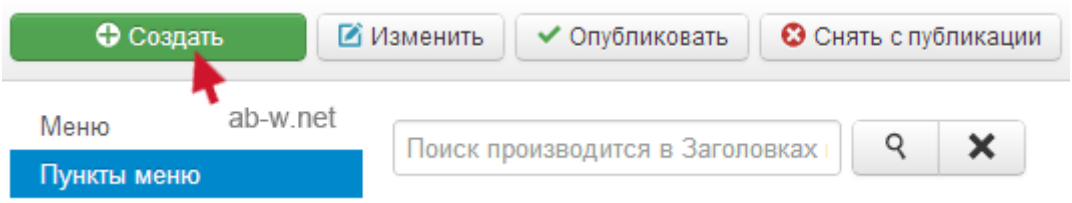
Так как в одном из уроков мы удалили демо-данные, вновь созданный объект вы найдете быстро.

4. Находим его. Нули – это опубликованные и нет пункты, мы их пока не создавали:



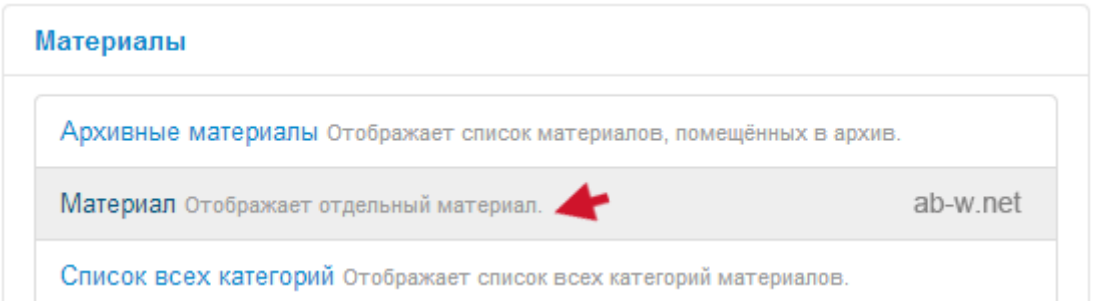
5. Нажатием кнопки мыши открываем наше Новое меню.

6. В панели, в менеджере менюшек, на странице пунктов жмем Создать:



7. Напротив области Тип пункта меню жмем Выбрать:

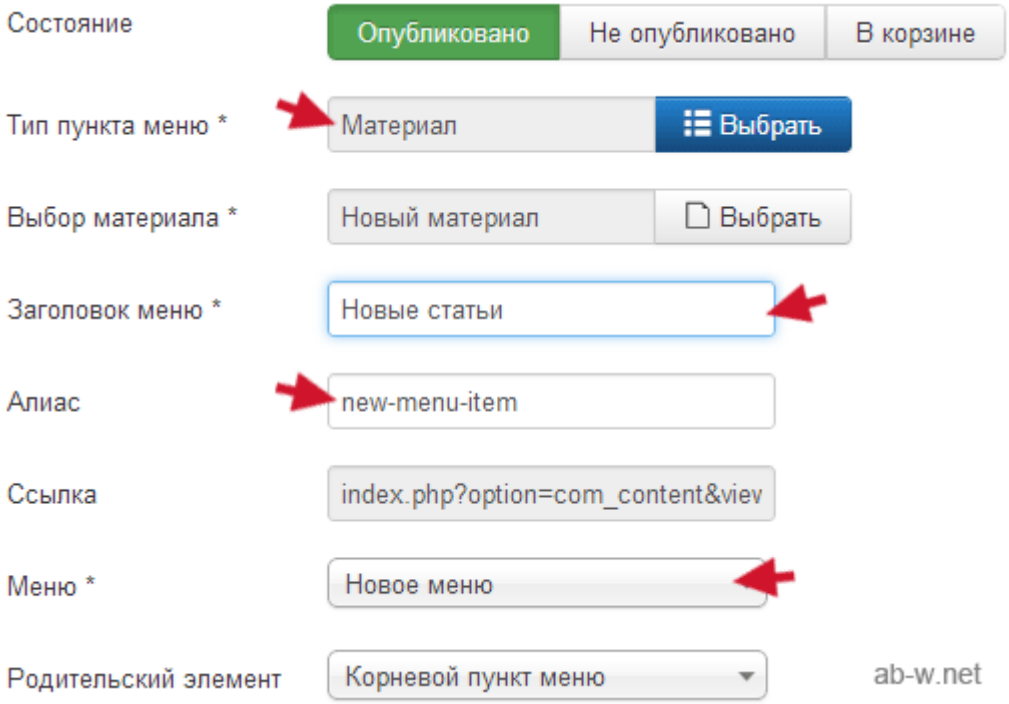
8. Нажатием кнопки мыши производим наш выбор в пользу Материал:



Сейчас, наверное, вы вспомнили один из прошлых уроков, где мы создавали материал (статью).

Теперь мы видим, что в поле ввода появился выбранный нами Тип.

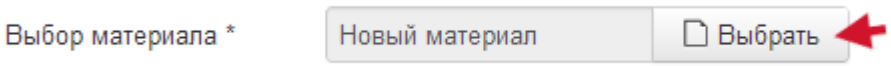
9. Прописываем заголовок пункта меню и Алиас:



Алиас – участок URL, произвольное имя на латинице.

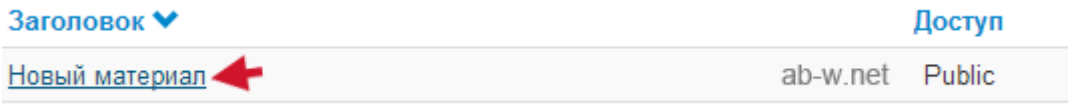
10. Пункт привязан к созданному во 2-ом и 3-ем шаге Новому меню (смотри выше).

11. Обратим внимание на поле Выбор материала:

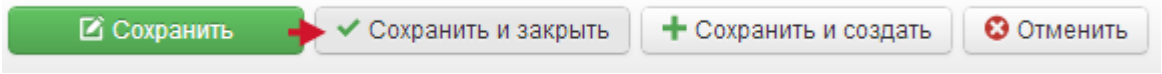


Сейчас вы уже точно вспомнили один из прошлых уроков, где мы создавали материал (статью).

12. Щелчком кнопки мыши выбираем наш материал:



13. Проверьте все еще один раз и нажмите Сохранить и закрыть:

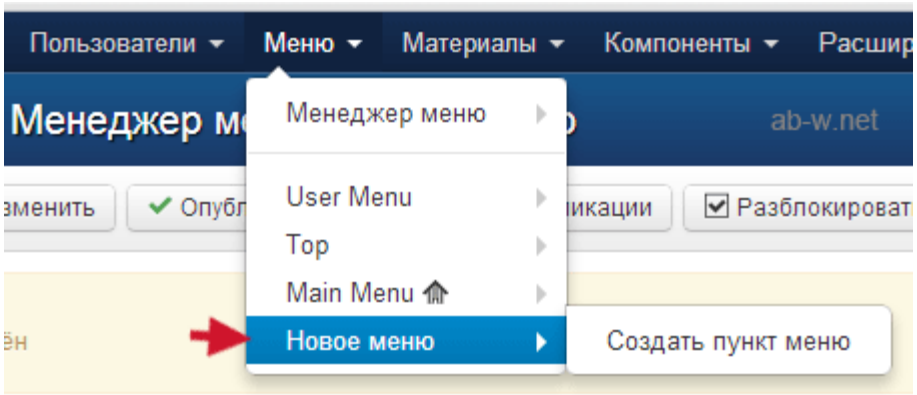


В принципе, урок завершен.

Ну, а где-же обещанный в прошлой главе материал (статья) и категория – скажете вы.

На страницах сайта изменений мы по-прежнему не увидим. Отгадайте почему?

Вот почему. Новуюменюшку то мы создали, вот она на иллюстрации ниже:



Открыв его, вы найдете пункт Новые статьи, который мы создали здесь в 9-ом шаге и, который содержит статью Новый материал, созданную в прошлом уроке, а сама статья под названием Новый материал находится в категории Происшествия, которая была создана еще раньше. Другими словами, система хранит все эти данные и в следующем уроке мы их все-таки выведем на страницы нашего сайта.

ЛАБОРАТОРНАЯ РАБОТА №10
Создание шаблона для сайта

Цель работы: научиться создавать шаблон.

Для начала необходима обычная XHTML страничка. Можно воспользоваться каким-нибудь сервисом для генератора HTML шаблонов. Или создать шаблон самостоятельно. Рассмотрим, в качестве примера, часто используемый шаблон с шапкой, футером и двумя колонками по бокам.

Разметка HTML будет выглядеть так:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
   <title></title>
   <meta name="title" content="" />
   <meta name="keywords" content="" />
   <meta name="description" content="" />
   <link rel="stylesheet" href="/style.css" type="text/css" media="screen, projection" />
   <!--[if lte IE 6]><link rel="stylesheet" href="/style_ie.css" type="text/css" media="screen, projection" /><![endif]-->
</head>
<body>
<div id="wrapper">
   <!-- #header-->
   <div id="header">
   </div>
   <div id="middle">
      <div id="container">
         <div id="content">
         </div><!-- #content-->
      </div><!-- #container-->
```

```
        <div class="sidebar" id="sideLeft">
        </div><!-- .sidebar#sideLeft -->
        <div class="sidebar" id="sideRight">
        </div><!-- .sidebar#sideRight -->
    </div><!-- #middle-->
    <div id="footer">
    </div><!-- #footer -->
</div><!-- #wrapper -->
</body>
</html>
```

И файл стилей style.css, для данной разметки:

```
* {
    margin: 0;
    padding: 0;
}
body {
    font: 12px/18px Arial, Tahoma, Verdana, sans-serif;
}
a {
    color: blue;
    outline: none;
    text-decoration: underline;
}
a:hover {
    text-decoration: none;
}
/* Begin of styles for the demonstration (you can remove them) */
a.expand {
    width: 150px;
    display: block;
    margin: 10px 0 0;
}
a.expand:hover {
    height: 500px;
}
/* End of of styles for the demonstration */
p {
    margin: 0 0 18px
}
img {
    border: none;
}
input {
    vertical-align: middle;
}
#wrapper {
    width: 100%;
    min-width: 1000px;
```

```
}
/* Header
----------------------------------------*/
#header {
   height: 150px;
   background: #FFE680;
}
/* Middle
----------------------------------------*/
#middle {
   border-left: 250px solid #B5E3FF;
   border-right: 250px solid #FFACAA;
   height: 1%;
   position: relative;
}
#middle:after {
   content: '.';
   display: block;
   clear: both;
   visibility: hidden;
   height: 0;
}
#container {
   width: 100%;
   float: left;
   overflow: hidden;
   margin-right: -100%;
}
#content {
   padding: 0 20px;
}
/* Sidebar Left
----------------------------------------*/
#sideLeft {
   float: left;
   width: 250px;
   position: relative;
   background: #B5E3FF;
   left: -250px;
}
/* Sidebar Right
----------------------------------------*/
#sideRight {
   float: right;
   margin-right: -250px;
   width: 250px;
   position: relative;
   background: #FFACAA;
}
```

```
/* Footer
---------------------------------------*/
#footer {
   height: 100px;
   background: #BFF08E;
}
```

Обзор макета для Joomla

С точки зрения Joomla этот макет разбивается на области, где будет выводиться основное содержимое (компонент) и дополнительное (модули).

При создании шаблонов Joomla используются следующие конструкции для вывода содержимого:

Содержимое HEAD

Здесь выводятся содержимое между тегами <head>…</head>, мета описание, заголовок страницы, подключаемые JavaScript и т. д. Для этого используется конструкция:

<jdoc:include type="head" />

Основное содержимое (компонент)

Для вывода основного содержимого, как правило это является содержимое компонента, используется следующая конструкция:

<jdoc:include type="component" />

Содержимое сообщения

Для вывода системных сообщений, например, сообщение при неудачной авторизации, используется конструкция:

<jdoc:include type="message" />

Содержимое модулей

Для вывода содержимого модулей используется конструкция:

<jdoc:include type="modules" name="position" style="xhtml" />

- name — позиция, в которой опубликованы модули
- style — стиль для вывода позиции модулей

Для контроля и подсчета модулей в позициях предусмотрен метод countModules.

Содержимое модуля

Можно также вывести содержимое одного модуля, для этого используется конструкция:

<jdoc:include type="module" name="custom" title="Title for module" />

- name — это название модуля, в данном примере это будет mod_custom
- title — заголовок модуля, должен совпадать с настройками модуля

В эту конструкцию можно добавлять дополнительные атрибуты, для контроля вывода содержимого модуля, например style=«xhtml».

При добавлении позиций или модуля в шаблон, не забывайте проверять настройки модулей — публикацию модуля и доступность модуля для текущего пользователя.

Использование параметров в шаблоне

Параметры для шаблона устанавливаются в XML файле описания шаблона. Их можно устанавливать в административной панели Joomla для нужного шаблона («Расширения» -> «Менеджер шаблонов»). С помощью этих параметров можно контролировать поведение шаблона, например, задать какой-то цвет для фона, вывести в качестве логотипа нужную картинку и т. п. Для получения значения параметра в шаблоне используется:

<?php $this->params->get('Имя параметра'); ?>

Соответственно для вывода значение параметра используется:

<?php echo $this->params->get('Имя параметра'); ?>

Стандартно в Joomla используется несколько типов для параметров, которые описываются в XML файле.

Шаблон для Joomla

С учетом этих данных нужно переписать HTML шаблон, сделать его шаблоном для Joomla. Расставить позиции модулей по своим местам, создать и подключить нужные CSS стили, задать область для вывода компонента и сообщений.

```
<?php
// защита от прямого доступа к файлу
defined('_JEXEC') or die;
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="<?php echo $this->language; ?>" lang="<?php echo $this->language; ?>" dir="<?php echo $this->direction; ?>" >
<head>
  <jdoc:include type="head" />
```

```
        <link rel="stylesheet" href="/templates/<?php echo $this->template; ?>/css/style.css" type="text/css" />
        <!--[if lte IE 6]>
        <link rel="stylesheet" href="/templates/<?php echo $this->template; ?>/css/style_ie.css" type="text/css" />
        <![endif]-->
    </head>
    <body>
    <div id="wrapper">
        <!-- #header-->
        <div id="header">
            <?php if ($this->params->get('logo')) : ?>
            <img src="/<?php echo $this->baseurl ?>/<?php echo htmlspecialchars($this->params->get('logo')); ?>" alt="MyLogo" />
            <?php endif;?>
            <jdoc:include type="modules" name="top" style="xhtml" />
        </div>
        <div id="middle">
            <div id="container">
                <div id="content">
                    <jdoc:include type="message" />
                    <jdoc:include type="component" />
                </div><!-- #content-->
            </div><!-- #container-->
            <div class="sidebar" id="sideLeft">
                <jdoc:include type="modules" name="left" style="xhtml" />
            </div><!-- .sidebar#sideLeft -->
            <div class="sidebar" id="sideRight">
                <jdoc:include type="modules" name="right" style="xhtml" />
            </div><!-- .sidebar#sideRight -->
        </div><!-- #middle-->
        <div id="footer">
            <jdoc:include type="modules" name="footer" style="xhtml" />
        </div><!-- #footer -->
    </div><!-- #wrapper -->
    </body>
    </html>
```

Нужно сохранить этот файл как index.PHP, это будет основной файл шаблона. В этом примере используется один параметр с именем «logo», в зависимости от которого будет выводиться соответствующая картинка в шапке шаблона. Хотя этих параметров может быть сколько угодно много и разного типа. Свойства типа $this->baseurl и т. п., это стандартные свойства Joomla, которые берутся из классов JDocumentHTML и JDocument соответственно.

XML файл инструкций для шаблона

Теперь нужно создать XML файл инструкций для шаблона templateDetails.XML

```
<?xml version="1.0" encoding="utf-8"?>
<extension version="1.6" type="template" client="site">
   <!-- Название шаблона -->
   <name>Mytemplate</name>
   <!-- Это не обязательные элементы, отображают информацию о авторе, лицензии и прочее -->
   <creationDate>21 May 2010</creationDate>
   <author>SmokerMan</author>
   <authorEmail>j-wiki@bk.ru</authorEmail>
   <authorUrl>http://www.j-wiki.ru</authorUrl>
   <copyright>Copyright (C) 2005 - 2011 Open Source Matters, Inc. All rights reserved.</copyright>
   <license>GNU General Public License version 2 or later</license>
   <!-- Версия шаблона -->
   <version>1.6.0</version>
   <!-- Описание шаблона -->
   <description>TPL_MYTEMPLATE_XML_DESCRIPTION</description>

   <!-- Файлы из которых состоит шаблон -->
   <files>
      <!-- Можно указывать директории -->
      <folder>css</folder>
      <folder>language</folder>
      <filename>index.html</filename>
      <filename>index.php</filename>
      <filename>templateDetails.xml</filename>
   </files>
   <!-- Позиции модулей, используемые в шаблоне -->
   <positions>
      <position>top</position>
      <position>left</position>
      <position>right</position>
      <position>footer</position>
   </positions>
   <!-- Установка/Удаление файлов локализации -->
   <languages folder="language">
      <language tag="ru-RU">ru-RU/ru-RU.tpl_mytemplate.ini</language>
      <language tag="ru-RU">ru-RU/ru-RU.tpl_mytemplate.sys.ini</language>
   </languages>
   <!-- Параметры для шаблона -->
   <config>
      <fields name="params">
         <fieldset name="advanced">
            <field name="logo" type="media"
                  label="TPL_MYTEMPLATE_LOGO_LABEL"
                  description="TPL_MYTEMPLATE_FIELD_LOGO_DESC" />
```

```
        </fieldset>
      </fields>
    </config>
</extension>
```

Локализация шаблона

Для перевода значений используются файлы локализации. Нужно создать их в директории language. Например, файл для перевода описаний и позиций для русской локализации будет следующий:

language/ru-RU/ru-RU.tpl_mytemplate.sys.ini.

```
TPL_MYTEMPLATE_XML_DESCRIPTION="Это мой первый шаблон"
TPL_MYTEMPLATE_POSITION_TOP="Модули вверху"
TPL_MYTEMPLATE_POSITION_LEFT="Модули слева"
TPL_MYTEMPLATE_POSITION_RIGHT="Модули справа"
TPL_MYTEMPLATE_POSITION_FOOTER="Модули внизу"
```

В этой директории нужно создать второй файл, для перевода значений параметров, ru-RU.tpl_mytemplate.ini.

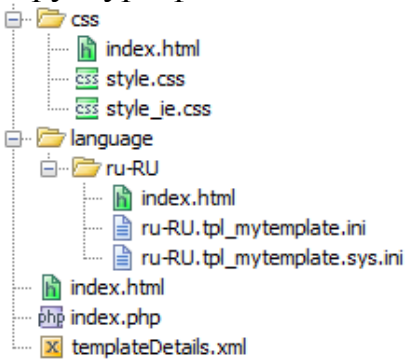
```
TPL_MYTEMPLATE_LOGO_LABEL="Логотип"
TPL_MYTEMPLATE_FIELD_LOGO_DESC="Выберите желаемый логотип"
```

Конечная структура шаблона

Также в каждой директории желательно создать пустые файлы index.html, для запрета листинга директорий.

```
<html><body bgcolor="#FFFFFF"></body></html>
```

Структура файлов обычного шаблона для Joomla будет следующей:



Теперь можно архивировать файлы в ZIP архив и устанавливать в Joomla через «Менеджер расширений».