МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
ЛУГАНСКОЙ НАРОДНОЙ РЕСПУБЛИКИ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
ЛУГАНСКОЙ НАРОДНОЙ РЕСПУБЛИКИ
«ЛУГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ ВЛАДИМИРА ДАЛЯ»

Стахановский инженерно-педагогический институт менеджмента
Кафедра информационных систем

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

по дисциплине

«КОМПЬЮТЕРНЫЕ И ТЕЛЕКОММУНИКАЦИОННЫЕ ТЕХНОЛОГИИ В ПРОФЕССИОНАЛЬНОЙ ДЕЯТЕЛЬНОСТИ»

для студентов направления подготовки
Профессиональное обучение (по отраслям),
магистерская программа
«Информационные технологии и системы» (в 2 х частях). Часть 1

Луганск 2023

УДК 69.032/007, 612.313

Рекомендовано к изданию Учебно-методическим советом
ГОУ ВО ЛНР «ЛГУ им. В. ДАЛЯ»
(протокол № от 2023 г.)

Методические указания к выполнению лабораторных работ по дисциплине **«Компьютерные и телекоммуникационные технологии в профессиональной деятельности»** для студентов направления подготовки **Профессиональное обучение (по отраслям),** магистерская программа «Информационные технологии и системы» в 2-х частях. Часть 1 / Сост.: Д. С. Тимошенко. – **Стаханов**: ГОУ ВО ЛНР «ЛГУ им. В. ДАЛЯ», 2023. – 69 с.

Методические указания к выполнению лабораторных работ содержит двенадцать работ по дисциплине «Компьютерные и телекоммуникационные технологии в профессиональной деятельности», которые включают в себя теоретические сведения, примеры решения, задачи и варианты с данными для их выполнения. К каждой лабораторной работе приведены вопросы для самопроверки. В методических рекомендациях представлен список использованных источников.

Предназначены для студентов магистерской программы «Информационные технологии и системы».

Составители: ст. преп. Тимошенко Д.С.

Ответственный за выпуск: доц. Карчевский В.П.

Рецензент: доц. Черникова С.А.

СОДЕРЖАНИЕ

Аннотация

Для закрепления теоретических знаний и приобретения необходимых умений, программой учебной дисциплины предусмотрено проведение лабораторных занятий. Для выполнения лабораторных занятий составлены методические указания, которые содержат краткий теоретический материал и практические примеры создания веб-страниц.

Целями освоения дисциплины Компьютерные сети и телекоммуникации является обучение теоретическим и практическим основам в организации и функционировании компьютерных сетей; формирование у студентов понимания важности применения и развития вычислительных систем, сетей и телекоммуникаций в современных технологиях, а также обучение студентов общим принципам построения вычислительных систем различных архитектур, принципам организации и характеристикам составных элементов компьютерных сетей, принципам и технологиям организации систем передачи данных.

Лабораторная работа №1 (Часть 1)

Тема: Установка программного комплекса Денвер

Цель: Научиться встановить программный комплекс Денвер

Задание для выполнения

1. Ознакомиться с теоретическим материалом.
2. Выполнить установку программного комплекса Денвер на компьютер.
3. Запустить Start servers.
4. В каталоге Z:\home\localhost\www создать текстовый файл index1.php следующего содержимого

```
<?=phpinfo() ?>
```

5. Далее загрузите в браузер созданный файл по адресуhttp://localhost/index1.php. Вы должны получить информацию о текущей версии PHP:

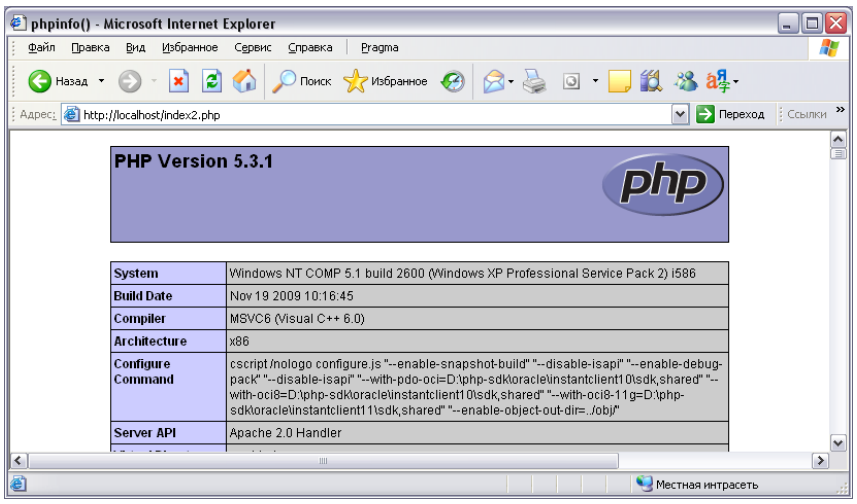

Рисунок 1.1 – Информация о текущей версии PHP

6. В каталоге Z:\home\localhost\www создать текстовый файл index2.php следующего содержимого

```
<html>
<head>
<title> Моя перша сторінка на PHP </title>
</head>
<body>
<b> Моя перша сторінка </b>
<br>
<?
   echo "HTML+ ";
   echo "PHP <br>";
?>
<i> Дякую за увагу </i>
</body>
</html>
```

7. Далее загрузите в браузер созданный файл по адресуhttp://localhost/index2.php. Вы должны получить следующую страницу

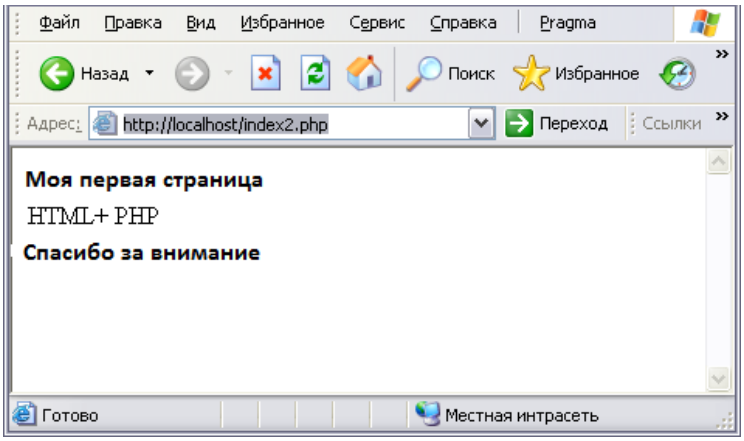

Рисунок 1.2 – Окно браузера с загруженной страницей

8. Добавьте на Вашу страницу информацию о разработчике – фамилии, имени, группе.

9. Просмотрите страницу с изменениями.

10. Завершите работу с Денвером.

Отчет должен содержать:

1. Тему, цель работы
2. Описание процесса установки комплекса Денвер.
3. Коды созданных файлов и копию окна обозревателя при их просмотре.

Методические указания

Сегодня PHP используется сотнями тысяч разработчиков. Несколько миллионов сайтов написано на PHP, что составляет более 20% доменов Internet.

"PHP может все", - заявляют его разработчики. В первую очередь PHP используется для создания скриптов, работающих на стороне сервера, для этого его собственно и придумали. PHP способен решать те же задачи, что и любые другие CGI-скрипты, в том числе обрабатывать данные HTML-форм, динамически генерировать HTML страницы и т.п. Но есть и другие области, где может использоваться PHP. Всего выделяют три главные области внедрения PHP.

- Первая область – это создание приложений (скриптов), выполняемых на стороне сервера.
- Вторая область – это создание скриптов, выполняемых в командной строке. То есть с помощью PHP можно создавать такие скрипты, которые будут производиться, независимо от web-сервера и браузера, на конкретной машине. Этот спо-

соб работы подходит, например, для скриптов, которые должны выполняться регулярно с помощью различных планировщиков задач или решения задач простой обработки текста.
- Третья область – это создание GUI-приложений (графических интерфейсов), выполняемых на стороне клиента.

Создать документ с кодом на PHP можно аналогично документу с кодомHTMLс помощью текстового редактора или специальногоWeb-редактора А вот прежде чем протестировать созданный код, Вы должны установить виртуальныйWeb-сервер с поддержкойHTML, PHPи многих других функций.

Все требуемое программное обеспечение можно установить с помощью специального программного комплекса Денвер. Джентльменский набор Web-разработчика ("Д.н.w.p", читается "Денвер" – почти как название города) – самый известный проект Лаборатории dk (http://dklab.ru) набор дистрибутивов (Apache+SSL, PHP5 в виде модуля, MySQL5, phpMyAdmin и т.д.) и программная оболочка, используемые Web-разработчиками (программистами и дизайнерами) для отладки сайтов на "домашней" (локальной) Windows-машине без необходимости выхода в Интернет.

Ключевая особенность Денвера – поддержка работы сразу с несколькими проектами, каждый из которых размещается на отдельном виртуальном хосте. Виртуальные хосты для проектов создаются автоматически: например, вам достаточно скопировать файлы проекта в /home/ИмяПроекта/www, и он тут же станет доступен по адресу http://ИмяПроекта. Это особенно удобно в работе веб-студий, разрабатывающих параллельно несколько сайтов, а также "в связке" с системами контроля версий CVS или Subversion. Схема именования директорий может быть легко настроена персонально на ваш хостинг в шаблоне виртуальных хостов.

Все компоненты Денвера уже настроены и готовы для работы (в частности, корректно настроена русскоязычная кодировка MYSQL, SSL и т.д.). Кроме того, вы можете обновлять любой из сервисов Денвера (Apache, PHP, MYSQL и т.д.) вручную, просто копируя новые версии дистрибутивов поверх старых.

Денвер автономен: он может располагаться в любой директории на диске (или даже на флеш-накопителе). Он также не изменяет системные файлы Windows так что может деинсталлироваться путем простого удаления своей папки.

СОСТАВ КОМПЛЕКСА

Состав базового пакета Денвера:

1. Apache 2 с поддержкой SSL и mod_rewrite.

2. PHP5: исполняемые файлы, модуль для веб-сервера Apache, дистрибутивный и адаптированный конфигурационный файл, библиотека GD, модули поддержки MySQL и sqLite.

3. MySQL5 с поддержкой INNODB, транзакций и русских кодировок (windows-1251).

4. phpMyAdmin – панель управления базой данных MySQL, а также скрипт, упрощающий добавление нового пользователя MySQL.

5. Настраиваемый эмулятор sendmail (/usr/sbin/sendmail), не отправляющий письма, а записывающий их в директорию /tmp/!sendmail.

6. Система автоматического поиска виртуальных хостов и обновления системного файла hosts, а также конфигурации Apache. Благодаря ей добавление нового виртуального хоста (или домена третьего уровня) заключается в простом создании каталога в /home (см. по аналогии с уже существующими хостами) и перезапуске комплекса. Все изменения вносятся в конфигурационные и системные файлы автоматически, но можно управлять этим процессом с помощью механизма шаблонов хостов (см. /usr/local/apache/conf/httpd.conf по подробным разъяснениям).

На официальном сайте Денвера доступны дополнения ("пакеты расширения"), расширяющие возможности базового комплекта:

– PHP версии 3 в виде CGI-программы;
– PHP версии 4 в виде CGI-программы;
– дополнительные модули для Apache;
– дополнительные модули для PHP;
– полная версия ActivePerl;
– интерпретатор ActivePython.
– сервер MySQL версии 4;
– другие популярные модули.

УСТАНОВКА И НАСТРОЙКА

Все дистрибутивы поставляются в виде самораспаковывающихся инсталляторов. После запуска программа установки задаст ряд вопросов о параметрах настройки Денвера.

Загрузите exe файл пакета Денвер.

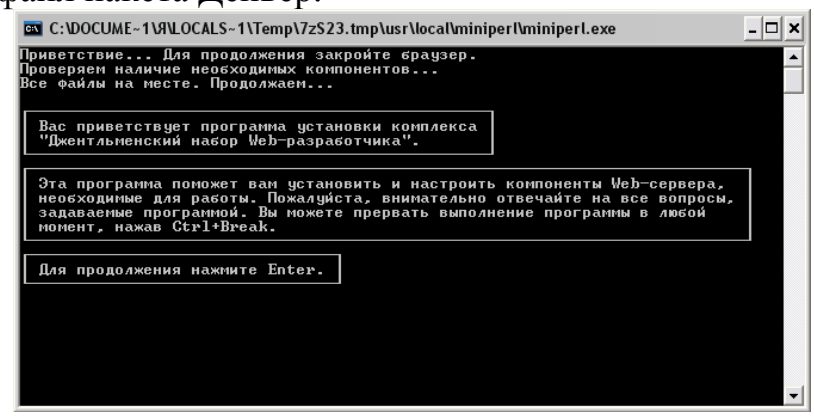

Рисунок 1.3 – Начало установки комплекса Денвер

По умолчанию программа устанавливается в C:\Webservers. При необходимости укажите имя другого каталога.

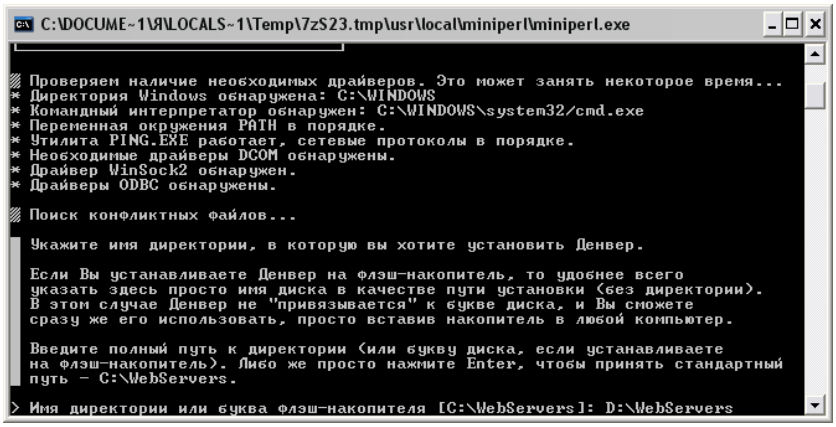

Рисунок 1.4 – Выбор каталога установки

Далее предлагается ввести букву виртуального диска, что будет связано с указанным каталогом. По умолчанию Z: Важно, чтобы диск с таким именем не был в системе.

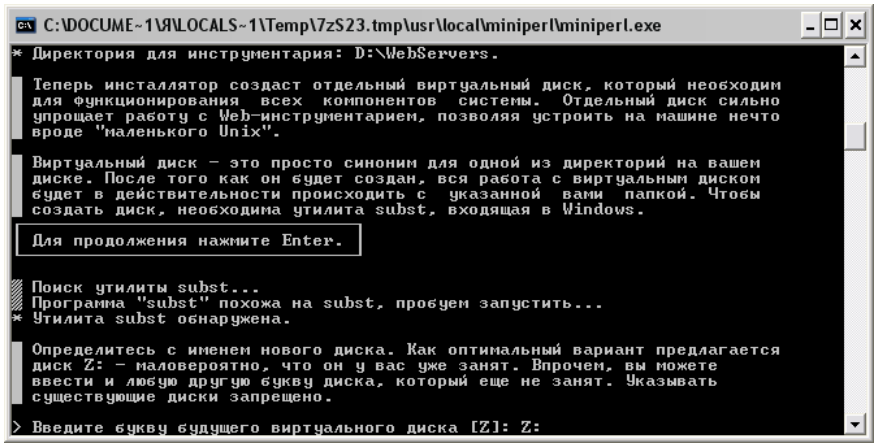

Рисунок 1.5 – Создание виртуального диска

В конце установки будет задан вопрос о выборе режимов запуска и прекращения работы комплекса. Рекомендуется выбрать режим 1:

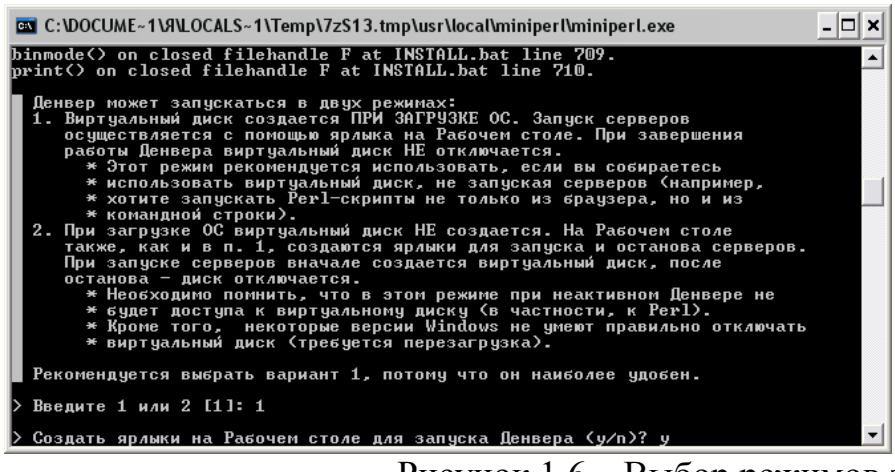

Рисунок 1.6 – Выбор режимов запуска

На Рабочем столе создаются ярлыки для запуска, перезапуска и остановки Денвера:

Рисунок 1.7 – Ярлыки для работы с Денвером

Если комплекс корректно установлен, на экране должно появиться следующее окно:

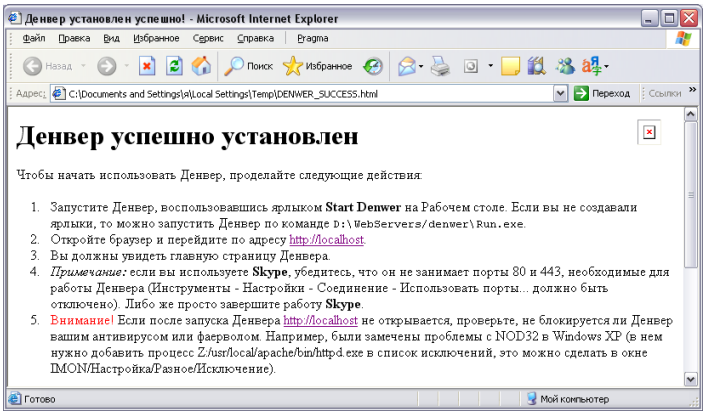

Рисунок 1.8 – Окно завершения процесса установки Денвера

После завершения установки нужно нажать на ярлыке Start servers на Рабочем столе:

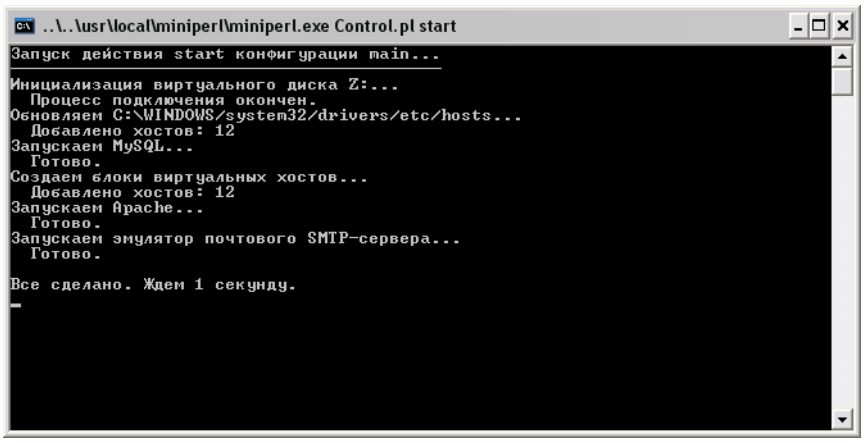

Рисунок 1.9 – Окно запуска Денвера

После закрытия консольных окон необходимо открыть браузер и набрать в нем адрес http://localhost

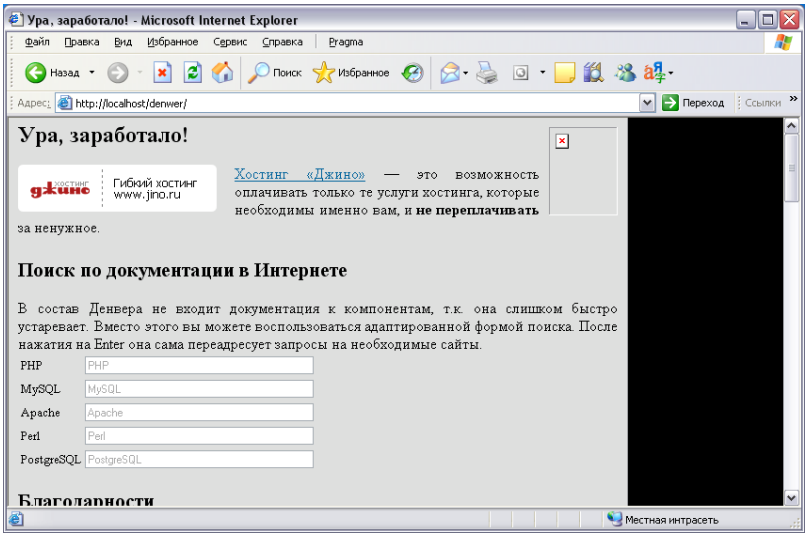

Рисунок 1.10 – Тестирование работы Денвера

Чтобы завершить работу с Денвером и остановить все работающие серверы, необходимо щелкнуть по ярлыку Stop servers на Рабочем столе. Для отключения виртуального диска нужно запустить программу SwitchOff.exe из каталога Denver.

Контрольные вопросы.

1. Что такое PHP? Его предназначение и возможности.
2. Для чего требуется программный комплекс Денвер.
3. Перечислите составляющие компоненты Денвера.
4. Опишите установку комплекса на локальный компьютер.
5. Как загрузить php-файл?
6. Как завершить работу с Денвером?

Лабораторная работа №1 (Часть 2)

Тема:Создание элементарных сценариев на языке PHP

Цель:Научиться ссоздавать элементарные сценарии на языке PHP

Задание для выполнения

1. Ознакомьтесь с теоретическим материалом.
2. Запустите Start servers.
3. В каталоге Z:\home\localhost\www создайте текстовый файл index3.php с кодом из примера, приведенного в методических указаниях (двухмерная таблица с индексами).
4. Далее загрузите в браузер созданный файл по адресу http://localhost/index3.php. Вы должны получить таблицу, приведенную на
рис. 1.11.
5. Добавьте на Вашу страницу информацию о разработчике – фамилии, имени, группе.
6. Выведите таблицу для табулирования функции y=x2 в интервале от -5 до 10 с шагом 1. Таблица должна иметь выделенное голубым цветом заголовок. Отрицательные числа выводить в ячейки желтого цвета, а положительные – серого.
7. Просмотрите страницу с изменениями.
8. Создайте функции, выполняющие вычисление суммы, произведения целых чисел от 1 до заданного n.
9. Создайте функцию, выводящую на страницу таблицу заданного размера nx m.
10*. Создайте функцию, распознающую простое число.
11. Завершите работу с Денвером.

Отчет должен содержать:

1. Тему, цель работы
2. Коды созданных файлов и копию окна обозревателя при их просмотре.

Методические указания

Синтаксис PHP

Синтаксис PHP во многом заимствован из таких языков как C, Java и Perl. Файл, обрабатываемый сервером обычно имеет расширение php.

PHP-код включаются в html-код в следующем виде:

```
<?PHP текст_кода ?>
```

или

```
<?
текст_кода;
?>
```

Комментарии

PHP поддерживает комментарии 'C', 'C++' и оболочки Unix. Например:

```
<?php echo "This is a test"; // Это однострочный комментарий в стиле c++


/* Это многострочный комментарий
это еще одна его строка */

   echo "This is yet another test"; echo "One Final Test";
   # Это комментарий в shell-стиле?>
```

echo

```
<?php echo "Эта информация будет выведена в HTML";?>
```

Присвоение значений переменным

Переменные в программах на PHP, отделяются символами $.

$city = "Tula";

city – переменная

Tula – значение

Некоторые операции

инкремента/декремента;

++$a Pre-increment Увеличивает $a на 1, затем возвращает $a.

$a++ Post-increment Возвращает $a, затем увеличивает $a на 1.

--$a Pre-decrement Уменьшает $a на 1, затем возвращает $a.

$a-- Post-decrement Возвращает $a, затем уменьшает $a на 1.

арифметические:

$a + $b Составление Сумма $a и $b.

$a - $b Вычитание Разница $a и $b.

$a * $b Умножение Произведение $a и $b.

$a / $b Деление. Доля от деления $a на $b.

$a % $b Modulus Целочисленный остаток от деления $a на $b.

срочные:

Есть две срочные операции. Первая – операция ('.'), возвращающая объединение правого и левого аргументов. Вторая – операция присвоения ('.='), присоединяющая правый аргумент в левом аргументе.

```
$a = "Hello"; $b = $a."World!"; // теперь $b содержит "Hello World!"
$a = "Hello"; $a.= "World!"; // теперь $a содержит "Hello World!"
```

Выражения сравнения

Выражения сравнения вычисляются в 0 или 1, обозначая FALSE или TRUE (соответственно).

PHP поддерживает
> (больше),
>= (больше или равно)
== (равно),
!= (не равно),
<(меньше) и <= (меньше или равно).

Эти выражения наиболее часто используются внутри условных операторов, таких как if.

сравнение:

$a == $b равно TRUE, если $a равно $b.
$a != $b не равно TRUE, если $a не равно $b.
$a <> $b не равно TRUE, если $a не равно $b.
$a < $b меньше TRUE, если $a строго меньше $b.
$a > $b больше TRUE, если $a строго больше $b.
$a <= $b меньше или равно TRUE, если $a меньше или равно $b.
$a >= $b больше или равно TRUE, если $a больше или равно $b.

Некоторые операторы

include "имя файла"

– команда для включения содержимого одного файла в другой. Содержимое файла, имя которого указывается в команде, целиком и полностью вставляется на то место, где располагается эта команда, при этом все коды PHP, содержащиеся в вставляемом файле, выполняются так же, как будто они были на месте этой команды. (Помните, что файл именно вставляется – т.е., например, пути к картинкам, которые должны присутствовать в вставляемом файле, следует указывать от местонахождения того файла, в котором находилась команда include.) Если файл, включаемый в страницу с помощью команды include, отсутствует, то вместо него размещается сообщение об этом, а программа на PHP выполняется дальше. (При необходимости завершения обработки и выдачи web-страницы в случае отсутствия включаемого файла вместо команды include следует использовать команду require.)

mail ("Кому", "Тема", "Текст сообщения", "Дополнительные заголовки")

– отправка почтового сообщения. При выполнении данной команды на сервере в соответствии с указанными параметрами формируется электронное письмо и отправляется с помощью установленной на сервере почтовой программы. В качестве параметра "Кому" может выступать набор адресов, разделенных запятыми. "Дополнительные заголовки" могут быть любые (естественно, допустимые почтовыми протоколами!), разделяться они должны комбинацией символов /n, которая в PHC означает переход на новую строку. (Если среди "Дополнительных заголовков" не указано поле From, оно заполняется по умолчанию почтовой программой web-сервера, например, именем "Unprivileged User".)

echo ("текст")

– вывод на web-страницу какого-нибудь текста. Чтобы вывести на web-страницу значение какой-либо переменной, достаточно просто написать ее имя внутри выводимой строки: команда echo "это цифра $a" выведет на web-страницу текст "это цифра

1", если раньше переменной $a было присвоено значение, равное единице. В случае необходимости использовать в выводимой строке кавычки или другие специальные символы перед этими символами следует ставить символ " .

if (условие) {...команды, которые должны выполняться, если условие верно...;} else {...команды, которые должны выполняться, если условие неверно...}

– команда, позволяющая выполнить то или иное действие в зависимости от истинности, верности или ложности того или иного условия. В фигурных скобках может размещаться несколько команд, разделенных точкой с запятой.

for (начальное значение счетчика, условие продления цикла, изменение счетчика на каждом цикле) { ...команды... ;}

– цикл, то есть повторение указанных в нем команд столько раз, сколько позволит условие смены счетчика цикла (т. с. переменной, специально выделенной для подсчета числа исполнений команд цикла).

while (условие) { ...команды... }

– цикл с условием. Команды в фигурных скобках выполняются до тех пор пока выполняется условие в заголовке цикла. Для того чтобы цикл прервался, нужно, чтобы условие выполняться перестало – поэтому внутри цикла необходимо предусмотреть возможность влиять на это условие.

Цикл do{. . .команды. . . } while (условие)

– работает так же, однако команды, указанные в фигурных скобках, будут выполнены как минимум один раз – даже если условие выполняться не будет. Прервать выполнение любого цикла можно оператором break – дальнейшее выполнение программы уйдет из команды, последующей после закрывающей фигурной скобки. Оператор же continue прерывает текущую стадию выполнения цикла, то есть после этого оператора дальнейшее выполнение программы начнется с очередной проверки условия заголовка цикла.

switch (выражение) {case значение: ... команды...; break; case другое значение: ... команды...; break;}

– оператор выбора. При его работе содержимое, помещенное в фигурные скобки, просматривается сверху вниз. Как только будет найден оператор case со значением, совпадающим со значением выражения, PHP начнет выполнять код, следующий за этим оператором case до последней фигурной скобки оператора switch или первого оператора break, в зависимости от того, что появится ранее. (Обратите внимание, что если команду break не указать в конце кода, относящегося к одному варианту значения выражения в заголовке оператора switch, PHP будет выполнять код дальше – то есть тот, который принадлежит уже следующему оператору case! Это – одно из отличий данного оператора от аналогичных в других языках программирования.) В конце оператора switch можно указать оператор default. Стоящий после него код выполнится в том случае,

foreach (переменная as массив) { . . .команды. .. ;}

– поочередное считывание всех элементов массива. Foreach считывает в указанную в его параметрах переменную поочередно все элементы указанного в них массива, выполняя каждый раз указанный в фигурных скобках код, в котором может использоваться указанная переменная. (Значение элементов массива этим оператором только считывается, их модификация с помощью команды foreach невозможна.)
Оператор foreach может использоваться только в PHP версии 4.0 и выше.

Программа на PHP может врываться кодом web-страницы – для этого достаточно вставить закрывающий тег в этот код и открывающий – после. Все, что находится между ними, будет выдаваться в браузер без какой-либо обработки, рассматриваясь как выводимый с помощью команды echo. Другими словами, код

<?php if ($a==l) { ?> Переменная а равна 1 <?php> }?>

эквивалентный коду

<?php if ($a==l) {echo "<p> Переменная а равна 1 </p>";}?>

Однако первый вариант меньше нагружает процессор компьютера, на котором расположен интерпретатор PHP. Из сказанного также следует, что все программы PHP, расположенные на одной web-странице, являются одной большой программой, несмотря на то, что они разделяются блоками обычного текста страницы. Именно поэтому переменная, объявленная в расположенном в начале страницы коде, сохраняет свое значение не только до ее конца, но и во всех подключаемых с помощью команды include файлах.

Пример:

Листинг скрипта, выводящего заполненную индексами таблицу

```
<?
$rows=5;
$col=5;
echo "<html><body>";
echo "<table border='5'>";
for ($i=1; $i<=$rows; $i++)
 {  echo "<tr>";

for ($j=1;$j<=$col;$j++)
  { if ((($i+$j) % 2) == 0)
      $color="#AAAAAA";
     else
       $color="#00CC00";

echo "<td bgcolor=$color> $i, $j </td>";
}
echo "</tr>";
}
echo "</table>";
echo "</body> </html>";
?>
```

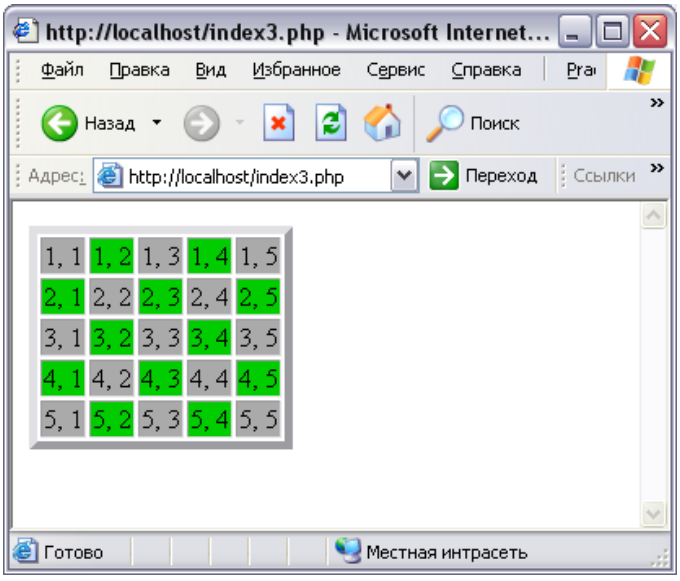

Рисунок 1.11 – Окно браузера с заданной таблицей

Массивы

Массивы в PHP – это очень мощный и гибкий механизм. Он позволяет сделать практически все, что только можно пожелать сделать с массивами. Поддерживаются как простые, так и ассоциативные массивы, причем они могут быть смешаны в любом порядке даже в пределах одного массива. Поддерживаются вложенные массивы, их вложенность никак очевидно не ограничена. В PHP существует множество функций для работы с массивами, они помогут вам выполнить большинство необходимых операций без лишних затрат времени и сил.

Кроме того, необходимо отметить еще одну особенность PHP при работе с массивами: в отличие от других языков, PHP позволяет задавать массивы практически любой сложности непосредственно в теле программы. На практике не раз придется столкнуться с необходимостью описания какой-нибудь сложной структуры данных и наполнением этой структуры данным. В других языках для этого обычно приходится писать дополнительный код, что не всегда удобно. В PHP же это возможно сделать очень просто и элегантно.

```
$data = array(1,10,100,1000, // Многочисленные данные
'Some text', 'Another text', // Срочные данные
'name'=>'John','age'=>23, //Ассоциативные связи в массиве
'date'=>array('day'=>10,'month'=>'may','year'=>2001));
//Вложенный массив
```

К этим данным можно обратиться следующим образом:

```
echo $data[1]; // Результат – 10
echo $data[5]; // Результат - 'Another text'
echo $data['age']; // Результат – 23
echo $data['date']['month']; // Результат – 'may'
```

Variable scope

Английский термин, вынесенный в название переводится как "область видимости переменной".

Под этим термином понимается то, что любая переменная, описанная в программе имеет свою область видимости, то есть если переменная описана в каком-то месте программы, то это вовсе не означает, что она автоматически становится видимой в любом другом месте этой программы.

У PHP на этот счет есть свои особенности. "Самая странная" вещь это то, что переменная, описанная вне функции невидима внутри нее. В других языках программирования мы привыкли к тому, что глобальные переменные программы видны везде, но в PHP это не так. Но посмотрев на эту "странность" внимательнее можно отметить, что она имеет больше выгоды, чем неудобств, поскольку является своего рода "защитным механизмом", который не позволит вам случайно перепутать локальную и глобальную переменную, что приводит к очень тяжелым последствиям.

Как же получить доступ к глобальным переменным из функции? Есть два способа сделать это:

1. Использовать глобальный ассоциативный массив $GLOBALS. Это единственная переменная в PHP, которая видна отовсюду и которая "содержит" в себе все глобальные переменные, имеющиеся в программе. Т.о. если вы внутри функции обращаетесь к переменной $variable – то вы обращаетесь к локальной переменной, а если $GLOBALS['variable'] – то к глобальной.

```
function myFunction() {
$variable = 5;// Присваиваем значение локальной переменной
$GLOBALS['variable'] = 10;
 // Присваиваем значение глобальной переменной
};
```

2. Используйте ключевое слово global. Оно позволит задать список глобальных переменных, которые будут видны внутри функции:

```
function myFunction() {
global $variable;
$variable = 10;// Присваиваем значение глобальной переменной
};
```

Функции в PHP

Как и любой другой алгоритмический язык, PHP имеет поддержку функций. В общем, синтаксис функций в PHP наиболее близок к тому, какие реализованные функции в C. ниже приведен пример очень простой функции:

```
function mySum($a,$b){
 $result = $a+$b;
 return($result);
};
```

Использование этой функции:

```
$result = mySum(2,3);
```

По своей сути функции есть участки кода, которые ассоциируются с определенным именем. Это как правило необходимо для того, чтобы иметь возможность выполнять какую-либо задачу несколько раз, используя один и тот же код.

Имя функции должно быть уникальным с точностью до регистра букв. Это означает, что, во-первых, имена MyFunction, myf unction и даже MyFuNcTiOn можно считать одинаковыми, и, во-вторых, мы не можем переопределить уже определенную функцию (стандартную или нет – не важно), но зато можем давать функциям такие же имена , как и переменным в программе (конечно, без знака $ в начале). Список аргументов состоит из нескольких перечисленных через запятую переменных, каждую из которых мы должны будем задать при вызове функции. Конечно, если у функции не должно быть аргументов совсем (как это сделано в функции time()), то следует оставить пустые скобки после ее имени.

Любая функция в PHP состоит из 4 основных частей:

– Имя функции

Каждая функция должна иметь свое уникальное имя, в противном случае PHP выдаст ошибку о попытке переопределения функции.

– Списка аргументов

Этот список может быть пустым (если функция не должна получать аргументы из внешней программы). Каждый аргумент должен иметь уникальное имя, под которым он будет "известен" внутри функции. В приведенном выше примере функция имеет два аргумента с именами $a и $b.

– непосредственно кода функции

Функция может содержать внутри себя практически любой код, допустимый в PHP за исключением определения других функций и объектов (это, кстати, отличает PHP например от JavaScript, где вложенное определение функций допустимо)

– Возвращаемое значение

Функция не обязана возвращать значение, но если это необходимо, то это делается с помощью оператора return().

Возврат функцией нескольких значений

Возврат нескольких значений осуществляется в виде массива.

```
function getNext(...){
$result = array();
$result['data'] = ...;
$result[1] = ...;
return($result);
};
```

Использование глобальных переменных

Для возврата нескольких значений из функции можно использовать запись требуемых значений непосредственно в глобальные переменные. Обычно это считается

очень плохим стилем программирования, поскольку делает программу очень трудной для понимания и является источником ошибок.

Использование ссылок.

У PHP4 появилась возможность работать со ссылками (references). Если вы знакомы с C/C++, то вам references не будут чем-то новым – это аналог указателей в C. Признаком использования references является наличие знака & перед именем переменной.

Использование статических переменных. В PHP возможно создание статических переменных внутри функции. Эти переменные отличаются от обычных тем, что, являясь локальными переменными внутри функции, они между тем сохраняют свое значение между вызовами функций. Посмотрим, как выглядит решение нашей задачи с использованием статической переменной как счетчик:

```
function getNext() {
global $data;
static $counter=0;
return($data[$counter++]);
};
```

Однако в более сложных случаях нельзя обойтись использованием только значений по умолчанию. Например, если неизвестно, сколько именно аргументов будет передано функции. В этом случае на помощь приходят функции работы со списком аргументов.

Контрольные вопросы

1. Можно ли выполнить php-скрипт в файле с расширением HTML?
2. Можно ли использовать дескрипторы языка HTML в php-файле?
3. По вашему мнению, транслятор языка PHP является интерпретатором или компилятором?
4. Выясните, где выполняется php-код, а где код html?
5. Назовите базовые алгоритмические структуры языка php?
6. Перечислите базовые типы данных языка php? Обязательно ли предварительное описание переменных?

Лабораторная работа №2

Тема:PHP. Обработка массивов

Цель:Научиться прорабатывать массивы с помощью сценариев на языке PHP

Задание для выполнения

1. Создать файл lab2_1.php в котором, используя функцию rand(min,max), заполнить массив двузначными случайными числами. Количество элементов массива должно быть произвольным целым числом. Вывести массив на экран в строку с пробелами между словами.
   Алгоритм исполнения: а) объявить массив; б) в цикле for заполнить массив случайными числами; в) вывести заглавие «Массив из…. элементов заполнен случайными числами»; г) в цикле вывести элементы массива.
2. Использую функцию sort() отсортировать массив по увеличению и вывести результат на экран.
3. Также вывести на экран отсортированный массив по убыванию.
4. Используя функцию array_revers(), перевернуть элементы массива в обратном порядке и результат вывести на экран.
5. Удалить последний элемент из массива (функция array_pop()), вывести на экран.
6. Подсчитать сумму частей в массиве array_sum() и количество частей в массиве count(). Найти и вывести на экран среднее арифметическое для элементов массива.
7. Добавить в массив значение "100" с индексом "maximum".
8. Используя цикл foreach, вывести на экран все элементы массива.
9. Используя функцию in_array определить, есть ли в массиве число 50.
10. Используя функцию array_unique удалить из массива повторяющиеся значения.
11. *Создать поле ввода для ввода порядкового номера месяца (в цикле), выпадающий список для выбора дня месяца (тоже с использованием циклов) и выпадающий список для указания дня недели, с которого начинается месяц. Вывести календарь на указанный месяц, с учетом количества дней в месяце и с учетом того, с какого дня недели начинается месяц.

Отчет должен содержать:

1. Тему, цель работы
2. Коды созданных файлов и копию окна обозревателя при их просмотре.

Методические указания

Массивы

Массивы в PHP – это очень мощный и гибкий механизм. Он позволяет сделать практически все, что только можно пожелать сделать с массивами. Поддерживаются как простые, так и ассоциативные массивы, причем они могут быть смешаны в любом порядке даже в пределах одного массива. Поддерживаются вложенные массивы, их вложенность никак очевидно не ограничена. В PHP существует множество функций для работы с массивами, они помогут вам выполнить большинство необходимых операций без лишних затрат времени и сил.

Создание массивов

Массивы представляют собой набор данных, объединенных под одним именем, и они занимают значительное место в программировании. Каждый массив состоит из отдельных элементов и каждый элемент массива ассоциирован с определенным индексом.

Массивы могут быть созданы с помощью оператора присвоения точно так же, как и обычные переменные. Имена массивов формируются по тем же правилам, что и имена переменных, в частности, они начинаются со знака $.

Отличительным признаком массива являются квадратные скобки после его имени, например:

```
$fruits [1] = "яблоко";
```

Данный оператор создает массив Sfruits и присваивает его элементу с индексом 1 значение "яблоко". С этого момента к элементу можно обращаться точно также, как и к обычной переменной, не забывая указывать значение индекса в квадратных скобках, например:

```
echo $fruits [l] ;
```

Этот оператор просто выведет строчку "яблоко". К массиву легко добавить и другие элементы, например:

```
$fruits [2]="груша";
$fruits [3]="абрикоса";
```

Вместе с числами в качестве индексов массивов могут быть использованы и строки, например:

```
$apple_count["Москва"] = 10000;
$apple_count["Рязань"] = 5000;
$apple_count["Казань"] =3000;
```

Следует обратить внимание на то, что в одном и том же массиве могут использоваться и числовые, и строчные индексы одновременно.

Существует сокращенная форма создания массива – после имени массива ставится пара квадратных скобок [ ], например:

```
$fruits[ ]="яблоко";
$fruits[ ]="груша";
$fruits[ ]="абрикоса";
```

PHP по умолчанию нумерует элементы массива начиная с 0, так что в этом случае $fruits[1] будет содержать строку "груша", а не "яблоко", как в первом примере.

Для обработки всех элементов массива удобно использовать циклы, например цикл for. Нумерация частей массива начинается с 0, а функция count возвращает количество частей массива. Ниже приведен пример, последовательно выводящий все элементы массива, каждый в своей строке.

```
for ($index = 0; $index < count ($fruits) ; $index+ + )
 {
 echo $fruits[$index], "\n";
 }
```

Существует еще более короткая форма для создания массива с помощью функции array:

```
$fruits = array ("яблоко", "груша", "абрикоса");
```

Этот оператор создает массив, индекс которого начинается с 0. Если нужно начать нумерацию элементов массива из другого числа, можно воспользоваться конструкцией =>:

```
$fruits = array (1 => "яблоко", "груша", "абрикоса");
```

Этот массив, в отличие от предыдущего, в элементе $fruits[1] содержит строчку "яблоко", а не "груша".

Так же может быть создан массив со строчными индексами:

```
$apple_count=array ("Москва" => 10000, "Рязань" => 5000,
"Казань" => 3000);
```

Оператор => связывает индекс и соответствующее ему значение элемента массива.

Модификация элементов массива

После создания массивов часто требуется изменение значений его элементов. Это выполняется так же просто, как изменение значения переменной. Для этого следует обратиться к элементу массива за его индексом. Например, есть следующий массив:

```
$fruits[0]="яблоко";
$fruits[1]="груша";
$fruits[2]="абрикоса";
```

Для присвоения нового значения второму элементу массива используется оператор:

```
$fruits[2] ="персик";
```

Для добавления нового элемента в конец массива используются уже знакомая конструкция:

```
$fruits[ ]="манго";
```

Наконец выведем на экран содержимое массива с помощью цикла. Все эти операторы объединены в следующем примере

Пример 1. Модификация частей массива

```
<html>
<head>
<title> Модификация элементов массива </title>
```

```
</head>
<body>
<h1> Модификация элементов массива </h1>
<?
 $fruits[0] ="яблоко";
  $fruits[1] ="груша";
  $fruits[2] ="абрикос";
  $fruits[2] ="персик";
  $fruits[ ] ="манго";
for ($index = 0; $index <count ($fruits); $index++)
{ echo $fruits[$index], "<BR>"; }
?>
</body>
</html>
```

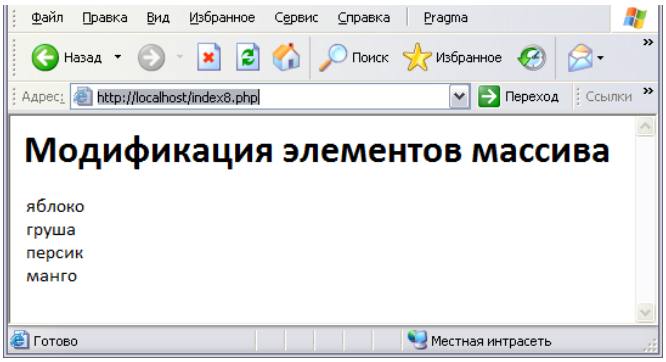

Рисунок 2.1 – Результат работы программы модификации массива

Предусмотрена также возможность скопировать массив как одну переменную:

```
<?
$fruits[0]="яблоко";
$fruits[1]="груша";
$fruits[2]="абрикоса";
$newfruits=$fruits;
echo $newfruits [2] ;
?>
```

Пример кода, приведенный выше, выведет строку абрикоса.

Удаление элементов массива

Помимо модификации, существует и возможность удаления элемента из массива. Для удаления элемента, казалось бы, можно просто присвоить элементу массива пустую строку, например:

```
<?
$fruits[0]="яблоко";
$fruits[1]="груша";
$fruits[2]="абрикоса";
$fruits[1]=" " ;
for ($index = 0; $index < count ($fruits); $index++)
```

```
{
echo $fruits[$index], "\n";
} ?>
```

Но таким образом удалить элемент массива не удастся, и в результате на месте второго элемента будет выведена пустая строка:

```
яблоко

абрикоса
```

Для удаления элемента из массива следует использовать функцию unset, действительно освобождающую занятую им память. Пример использования этой функции приведен ниже:

```
<?
$fruits[0]="яблоко";
$fruits[1]="груша";
$fruits[2]="абрикоса";
unset ($fruits [l]);
for ($index = 0; $index < count ($fruits); $index++)
{
echo $fruits [$index], "\n";
} ?>
```

При выполнении этого кода на экран будет выведено сообщение о том, что элемент массива не определен:

```
яблоко
PHP Notice: Undefined offset: 1 в 76-02.php on line 8
```

Перебор элементов массива

В предыдущем разделе был приведен пример вывода всех элементов массива посредством цикла for. Для более простого вывода всех элементов массива (в том числе со строковыми или непоследовательными числовыми индексами) предусмотрена функция print_r, пример использования которой приведен ниже:

```
<?
$fruits[0]="яблоко";
$fruits[1]="груша";
$fruits[2]="абрикоса";
print_r ($fruits);
?>
```

При выполнении этого кода будет выведено следующее

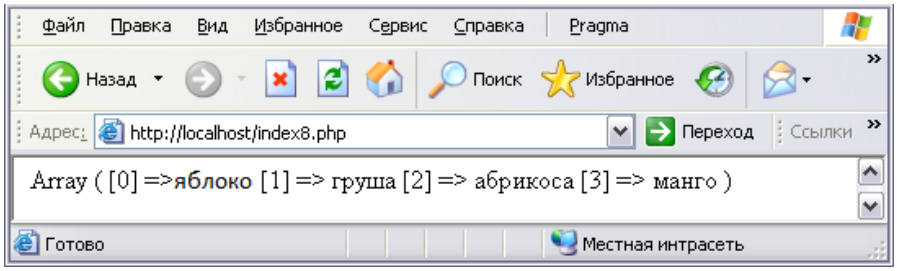

Рисунок 2.2 – Вывод значений элементов массива

Для обработки массивов предусмотрен также специальный вид цикла – цикл foreach. Синтаксис этого оператора имеет два варианта:

```
foreach (array as $value) statement
foreach (array as $key => $value) statement
```

Первый вариант оператора присваивает в цикле переменной $value очередной элемент массива. Второй вариант кроме этого присваивает переменной $key значение индекса, соответствующего текущему элементу массива. Пример использования цикла foreach приведен ниже:

```
<?
$fruits=array ("яблоко", "груша", "абрикоса");
foreach($fruits as $value)
{ echo "Значение: $value\n"; }
?>
```

В результате выполнения этого примера будут выведены строчки:

```
Значение: яблоко
Значение: груша
Значение: абрикоса
```

Для вывода индекса элемента массива вместе с его значением используется второй вариант синтаксиса оператора foreach:

```
<?
$fruits = array ("яблоко", "груша", "абрикоса");
foreach ($fruits as $key => $value)
{
echo "Индекс: $key; Значение: $value\n";
}
?>
```

В результате выполнения этого примера будут выведены строчки:

```
Индекс: 0; Значение: яблоко
Индекс: 1; Значение: груша
Индекс: 2; Значение: абрикоса
```

Для перебора элементов массива может быть использован цикл while в сочетании с функцией each. Эта функция предназначена специально для перебора элементов мас-

сива. Каждый раз, когда она вызывается, она возвращает текущий элемент массива и передвигает внутренний указатель на следующий элемент. Функция возвращает пару индекс-значения в виде массива. Для присвоения индексу и значению элемента массива отдельным переменным используется функция list. Пример использования этих двух функций приведен ниже.

```
<?
$fruits=array ("яблоко", "груша", "абрикоса");
while ($list ($key, $value) = each ($fruits))
{
echo "Индекс: $key; Значение: $value\n";
}
?>
```

Результат выполнения этого кода такой же, как и у предыдущего примера.

Функции для работы с массивами

Как и для обработки строк, для работы с массивами вPHPпредусмотрено достаточно много функций. Наиболее важные из них приведены в табл. 2.1.

Таблица 2.1 Функции обработки массивов

Функция	*Назначение*
array_chunk	Разбивка массива на несколько меньших массивов заданного размера.
array_combine	Создает массив из двух заданных массивов – массива индексов элементов и массива значений.
array_count_values	Формирует массив, индексами которого являются значения array_count_values заданного массива, а значения – число повторений соответствующего значения в заданном массиве.
array_diff	Формирует массив из тех элементов первого заданного массива, которые отсутствуют у остальных заданных в качестве аргументов функции массивов.
array_fill	Заполняет массив заданным значением.
array_intersect	Формирует массив из элементов, присутствующих во всех заданных массивах.
array_key_exists	Проверка наличия заданного индекса в массиве.
array_keys	Возвращает массив из индексов заданного массива.

Функция	*Назначение*
array_merge	Объединяет несколько массивов в один.
array_multisort	Выполнение сортировки многомерного массива или нескольких одномерных массивов.
array_pad	Дополняет массив до заданного количества элементов по заданному значению.
array_pop	Возвращает последний элемент массива, одновременно удаляя элемент из массива.
array_push	Добавляет заданные элементы в конец массива аналогично конструкции $array[] = $value;
array_rand	Выбор одного или нескольких случайно взятых элементов из массива.
array_reduce	Осуществляет последовательное применение заданной функции к элементам массива, формируя итоговое значение. К примеру, если функция выполняет составление собственных аргументов, в итоге формируется сумма всех частей массива.
array_reverse	Проводит обращение массива – первый элемент становится последним, второй – предпоследним и т и.
array_search	Ищет заданный элемент в массиве и возвращает соответствующий ему индекс
array_shift	Возвращает первый элемент массива, одновременно удаляя его из массива с перенумерованием числовых индексов.
array_slice	Вырезает из массива подмассив заданной длины, начиная с указанного элемента.
array_sum	Вычисляет сумму всех частей массива.
array_umque	Удаляет дублируемые значения из массива.
array_unshift	Добавляет один или несколько элементов в начало массива с перенумерованием числовых индексов.
array_walk	Вызов заданной функции последовательно для каждого элемента массива.
array	Создает массив из заданных значений или пар индекс-значения.
arsort	Сортирует массив по убыванию его значений, сохраняя индексы неизмен-

Функция	*Назначение*
	ными.
assort	Сортирует массив по увеличению его значений, сохраняя неизменными индексы.
count	Возвращает количество элементов в массиве.
current	Возвращает значение текущего элемента массива.
each	Возвращает текущий индекс и значение элемента массива и продвигает указатель на следующий элемент.
in_array	Проверка присутствует ли заданное значение в массиве.
key	Возвращает индекс текущего элемента массива.
krsort	Сортирует массив по убыванию его индексов.
ksort	Сортирует массив по увеличению его индексов.
list	Присваивает значение из массива списка переменных.
natcasesort	Сортирует массив естественным образом без учета символов регистра.
natsort	Сортирует массив естественным образом на основе регистра символов.
pos	Синоним функции current.
reset	Установка внутреннего указателя на первый элемент массива.
rsort	Сортирует массив по уменьшению значений его элементов с перенумерованием его индексов.
shuffle	Переставляет элементы массива случайным образом.
sizeof	Синоним функции count.
sort	Сортирует массив по увеличению значений его элементов с перенумерованием его индексов.
usort	Сортирует массив с использованием заданной функции сравнения элементов массива.

Сортировка массивов

ВPHPпредусмотрены все возможные способы сортировки данных в массивах. Простым способом является функция sort, сортирующая заданный массив по увеличению значений его элементов. Ниже приведен пример использования этой функции.

```
<?
$fruits[0]= "яблоко";
$fruits[1]="груша";
$fruits[2]="абрикоса";
print_r($fruits);
sort($fruits);
print_r($fruits);
?>
```

Ниже приведен результат выполнения этого примера. Массив $fruits отсортирован, а его элементы перенумерованы.

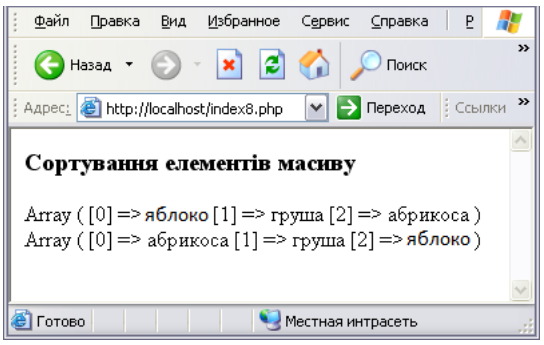

Рисунок 2.3 – Пример работы программы сортировки массивов

С помощью функции rsort массив можно отсортировать по убыванию значений его элементов:

```
<?
$fruits[0]="яблоко";
$fruits[1]="груша";
$fruits[2]="абрикоса";
print_r($fruits);
rsort ($fruits);
print_r ($fruits);
?>
```

В результате будет выведено следующее:

```
Array ([0] => яблоко [1] => груша [2] => абрикоса )
Array ([0] => яблоко [1] => груша [2] => абрикоса)
```

Но что, если в массиве используются строчные индексы? При использовании функций sort и rsort происходит перенумерование элементов массива, и информация о

строчных индексах будет потеряна. В этом случае на помощь приходит функция assort, пример использования которой приведен ниже.

```
<?
$fruits['красный']="яблоко";
$fruits['зеленый']="груша";
$fruits['оранжевый']="абрикоса";
print_r($fruits);
asort($fruits);
print_r ($fruits);
?>
```

Результат выполнения примера выглядит следующим образом:

```
Array ([красный] => яблоко [зеленый] => груша [оранжевый]
 => абрикоса)
Array ([оранжевый] => абрикоса [зеленый] => груша [крас-
 ный] => яблоко )
```

Функция arsort также сохраняет строчные индексы, но сортирует массив по убыванию. Функции ksort и krsort сортируют массив не по значениям элементов, а по их индексам (в растущем и уменьшающем порядке соответственно). Наконец, если определить собственную функцию сравнения элементов массива, то с помощью функции usort можно отсортировать массив по произвольному критерию.

Навигация по массивам

В PHPесть ряд функций для навигации по массивам. Навигация производится с помощью указателя текущего элемента массива. Например, есть следующий массив:

```
$vegetables[0]="картофель";
$vegetables[1]="морковь";
$vegetables[2]="свекла";
```

Текущий элемент массива определяется с помощью функции current.

После создания массива его первый элемент становится текущим:

```
echo "Текущий: ", current ($vegetables), "<BR>";
```

Для перемещения указателя к следующему элементу используется функция next:

```
echo "Следующий: ", next ($vegetables), "<BR>";
```

Для перемещения указателя к предыдущему элементу используется функция prev:

```
echo "Предыдущий: ", prev ($vegetables), "<BR>";
```

Функция end перемещает указатель к последнему элементу массива и возвращает его:

```
echo "Последний: ", end ($vegetables), "<BR>";
```

Для возврата указателя к началу массива используется функция reset:

```
reset ($vegetables);
```

Использование всех этих функций показано в примере.

Пример. Навигация по массиву

```
<html>
<head>
<title>Навигация по массиву</title>
</head>
<body>
<h3> Навигация по массиву </h3>
<?
   $vegetables[0] ="картофель";
   $vegetables[1] ="морковка";
   $vegetables[2] ="свекла";
   print_r($vegetables);
   echo "<BR>";
   echo "Текущий: ", current ($vegetables), "<BR>";
   echo "Следующий: ", next ($vegetables), "<BR>";
   echo "Предыдущий: ", prev ($vegetables), "<BR>";
   echo "Последний: ", end ($vegetables), "<BR>";
   echo "Сброс указателя.<BR>";
   reset ($vegetables);
   echo "Текущий: ", current ($vegetables), "<BR>";
?>
</body>
</html>
```

Превращение строк в массивы и наоборот

BPHPпредусмотрена возможность преобразования данных из строки в массив и обратно. Функция implode формирует строку из массива, а функция explode формирует массив из указанной строки.

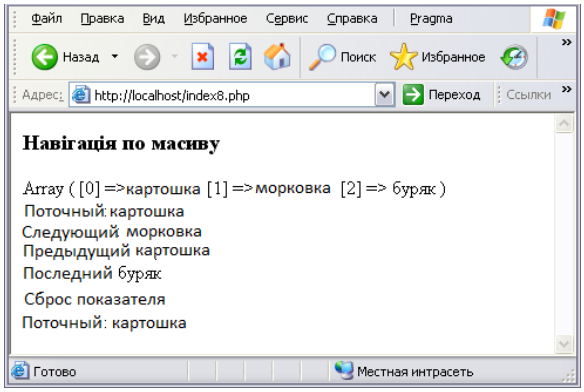

Рисунок 2.4 – Результат работы программы навигации по массиву

К примеру, нужно получить все содержание массива в виде одной строки. Для этого используется функция implode, которой передаются два аргумента – сам массив и

строка, используемая в качестве разделителя элементов массива. Пример использования implode приведен ниже, как разделитель использована запятая.

```
<? $vegetables[0]="картофель";
  $vegetables[1]="морковь";
  $vegetables[2]="свекла";
  $text = implode (",",$vegetables);
  echo $text;
?>
```

В результате выполнения этого примера будет выведена строка:

```
картофель, морковь, свекла
```

Для того чтобы добавить дополнительные пробелы после запятой, нужно просто изменить строку-раздеитель:

```
$text = implode(", ", $vegetables);
```

Результат будет следующим:

```
картофель, морковь, свекла
```

Обратная операция производится с помощью функции explode. В строке ищется заданный разделитель, и части строчки, ограниченные разделителями, становятся элементами нового массива. Пример использования функции приведен ниже.

```
<? $text = "картофель, морковь, свекла";
  $vegetables = explode(", ", $text);
  print_r($vegetables);
?>
```

Результат выполнения приведен ниже. По всей видимости, строка корректно превращена в массив.

```
Array ([0] => картофель [1] => морковь [2] => свекла )
```

Извлечение переменных из массивов

Если нужно для массива, проиндексированного строками, присвоить значение элементам переменных, одноименным с соответствующими индексами, то на помощь приходит функция extract. Например, есть следующий массив:

```
$fruits["good"]="яблоко";
$fruits["better"]="груша";
$fruits["best"]="персик";
```

После вызова функции extract будут созданы переменные $good, $better и т.д., и им будут присвоены соответствующие значения из массива.

```
extract(;fruits);
```

Это легко наблюдать в следующем примере.

```
<html>
<head>
<title>Извлечение переменных из массива </title>
```

```
</head>
<body>
<h3>Извлечение переменных из массива </h3>
<?
  $fruits["good"]="яблоко";
  $fruits["better"]="груша";
  $fruits["best"]="персик";
  extract($fruits);
  echo "\$good = $good <BR>";
  echo "\$better = $better <BR>";
  echo "\$best = $best <BR>";
?>

</body>
</html>
```

После выполнения этого примера переменной $Sgood будет присвоено значение "яблоко", переменной $better – значение "груша" и т.п., что показано ниже.

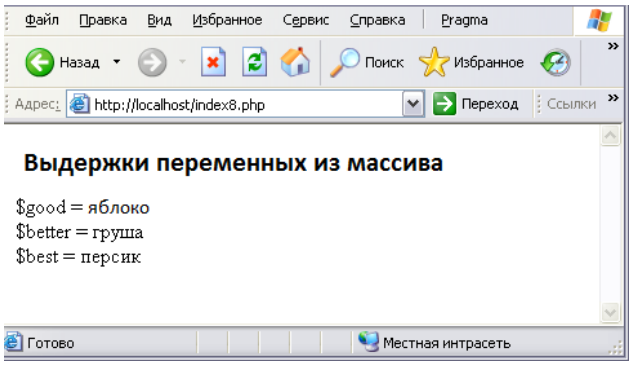

Рисунок 2.5 – Результат работы программы, выполняющей извлечение данных из массива

Для аналогичной цели может также использоваться функция list, последовательно присваивающая значение элементов массива указанным переменным. Ниже приведен пример использования этой функции:

```
<?
  $vegetables[0]="картофель";
  $vegetables[1]="морковь";
  $vegetables[2]="свекла";
  list ($first, $second)= $vegetables;
  echo $first, "\n";
  echo $second;
?>
```

В результате выполнения этого примера будут выведены строки

```
картофель
```

```
морковка
```

А что делать, если нужно выполнить обратную операцию и сформировать массив на основе списка переменных? Для этого предназначена функция compact. В качестве аргументов она принимает имена переменных или массивы имен переменных. Значения переменных с указанными именами стают еще одними элементами массива.

```
<?
  $firstname="Сергей";
  $lastname="Баранкин";
  $role="Редактор";
  $subarray = array("flrstname", "lastname");
  $resultarray = compact($subarray, "role");
  printr ($resultarray);
?>
```

В результате выполнения этого примера будет сформирован следующий массив:

```
Array [flrstname] => Сергей [lastname] => Баранкин
[role] => Редактор
```

Слияние и разделение массивов

Над массивами возможны операции слияния и разделения. Например, нужно сформировать массив, состоящий из последних двух элементов заданного трехэлементного массива. Для этого используется функция array_slice, которая имеет три аргумента: начальный массив, смещение – номер первого элемента массива (начиная с 0) и длина создаваемого массива:

```
<?
  $fruits["good"]="яблоко";
  $fruits["better"]="груша";
  $fruits["best"]="персик";
  $subarray=array_slice ($fruits, 1, 2) ;
  foreach($subarray as $value)
  {
  echo "Фрукт: $value\n";
  }
?>
```

В результате будут выведены строчки:

```
Фрукт: груша
Фрукт: персик
```

Если задан отрицательный смещение, то начальный элемент будет отсчитываться с конца, а не с начала массива. Если длина задана отрицательным числом, то выборка остановится за это число элементов к концу исходного массива. Если длина не задана

вообще, то будет вырезан массив начиная с заданного элемента и до конца массива (или до его начала в случае отрицательного смещения).

Слияние массивов производится с помощью функции array_merge:

```
<?
$fruits=array ("яблоко", "груша", "абрикоса");
$vegetables=array ("картофель", "морковь", "свекла");
$produce=array_merge ($fruits, $vegetables);
foreach($produce as $value)
{ echo "Элемент массива: $value\n"; }
?>
```

В результате будут выведены строчки:

```
Элемент массива: яблоко
Элемент массива: груша
Элемент массива: абрикос
Элемент массива: картофель
Элемент массива: морковь
Элемент массива: свекла
```

Сравнение массивов

ВPHPесть средства для сравнения массивов и нахождения различий в их элементах. Допустим, есть два массива, в которых совпадает только второй элемент. Для построения массива, в который будут входить только те элементы первого массива, которые отсутствуют во втором, используется функция array_diff.

```
<?
$local_fruits=array ("яблоко", "гранат", "абрикоса");
$tropical_fruits=array ("ананас", "гранат", "папайя");
$difference=array_diff ($local_fruits, $tropical_fruits);
foreach ($difference as $key => $value)
{
echo "Индекс: $key; Значение: $value\n";
}
?>
```

В результате выполнения этого примера будут выведены строчки:

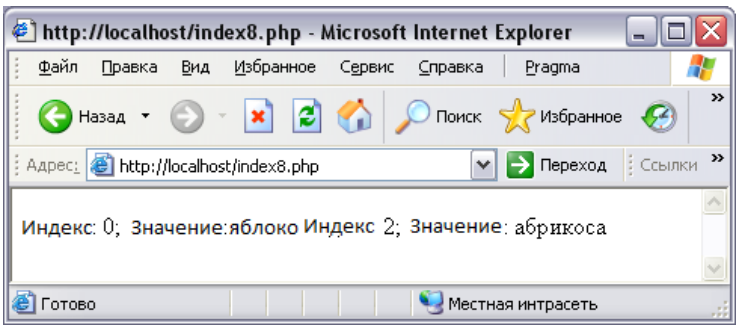

Рисунок 2.6 – Результат работы программы сравнения массивов

В данном случае сравнивались только значения элементов массива. Если же нужно анализировать еще и индексы, на помощь приходит функция array_diff_assoc (ее название объясняется тем, что массивы со строчными индексами называются также ассоциативными массивами):

```
<?
$local_fruits=array ("фрукт1" => "яблоко", "фрукт2" =>
"гранат", "фрукт3" => "абрикоса")г
$tropical_fruits = array ("фрукт_1" => "ананас", "фрукт_2"
 =>
"гранат", "фрукт_3" => "папайя");
$difference = array_diff_assoc ($local_fruits,
$tropical_fruits);
foreach ($difference as $key => $value)
{
echo "Индекс: $key; Значение: $value\n";
}
?>
```

В результате будут выведены строки (следует обратить внимание на то, что полностью совпадающих как по значению, так и по индексу элементов в двух массивах не найдено):

```
Индекс: фрукт1; Значение: яблоко
Индекс: фрукт2; Значение: гранат
Индекс: фрукт3; Значение: абрикоса
```

Встречается и обратная задача – найти элементы, общие для двух массивов. В этом случае используется функция array_intersect:

```
<?
$local_fruits=array ("яблоко", "гранат", "абрикоса");
$tropical_fruits=array ("ананас", "гранат", "папайя");
$difference=array_intersect ($local_fruits,
 $tropical_fruits);
foreach ($difference as $key => $value)
{
echo "Индекс: $key; Значение: $value\n";
}
?>
```

В результате будет выведена единая строчка:

```
Индекс-. 1; Значение: гранат
```

Если же следует учесть и индексы при определении общих элементов, используется функция array_intersect_assoc:

```
<?
$local_fruits = array("фрукт1" => "яблоко", "фрукт2" =>
 "гранат", "фрукт3" => "абрикоса");
$tropical_f ruits = array("фрукт" => "яблоко", "фрукт2" =>
 "гранат", "фрукт3" => "папайя");
$difference = array_intersect_assoc($local_fruits,
 $tropical_fruits);
foreach ($difference as $key => $value)
{ echo "Индекс: $key; Значение: $value\n"; }
?>
```

Результатом выполнения этого примера будет также одна строка (следует обратить внимание, что первые элементы массивов равны по значениям, но не по индексам):

```
Индекс:,фрукт2; Значение: гранат
```

Обработка данных в массивах

Данные в массивах могут быть обработаны самыми разными способами. Например, если нужно удалить в массиве элементы со повторяющимися значениями, на помощь приходит функция array_unique:

```
<?
$scores = array(65, 60, 70, 65, 65) ;
printer ($scores);
$scores = array_unique($scores);
printr (Sscores);
?>
```

Ниже приведен результат выполнения этого примера – следует обратить внимание на то, что дублируемые элементы удалены:

```
Array ( [0] => 65 [1] => 60 [2] => 70 [3] => 65 [4] => 65 )
Array ( [0] => 65 [1] => 60 [2] => 70 )
```

Еще одна полезная функция для обработки данных – array_sum, которая возвращает сумму всех элементов массива:

```
<?
$scores = array (65, 60, 70, 64, 66);
echo "Средний балл = ",
 array sum ($scores) / count ($scores) ;
?>
```

В данном примере вычисляется средний балл экзаменационных оценок студентов:

```
Средний балл = 65
```

И последний пример обработки данных – функция array_flip меняет местами индексы и значение элементов массива, как показано в примере

```
<html>
```

```
<head>
<title>Переворот массива</title>
</head>
<body>
<h3>Переворот массива </h3>
<?
 $local_fruits=array("фрукт1"=>"яблоко",
 "фрукт2"=>"груша", "фрукт3"=>"апельсин");
foreach ($local_fruits as $key => $value)
{ echo "Индекс: $key; Значение: $value<BR>"; }
echo "<BR>";
$local_fruits = array_flip ($local_fruits);
foreach ($local_fruits as $key => $value)
{ echo "Индекс: $key; Значение: $value<BR>"; }
?>
</body>
</html>
```

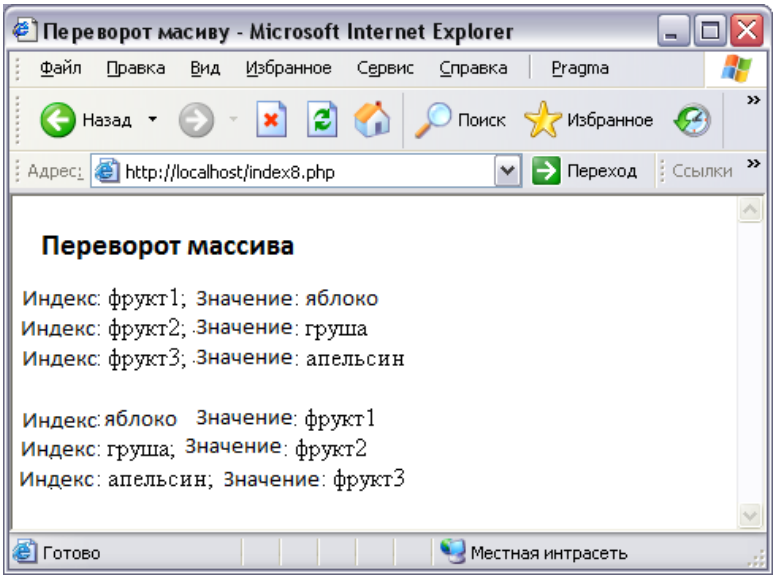

Рисунок 2.7 – Результат работы программы с использованием функции array_flip

Многомерные массивы

До сих пор рассматривались только одномерные массивы. Но вPHPесть возможность работы и с многомерными массивами. Пусть для хранения экзаменационных оценок используется одномерный массив Stest_scores:

```
$test_scores ["Иванов"] = 95;
$test_scores ["Петров"] = 87;
```

Но что если нужно хранить оценки по нескольким предметам? Для этого удобно использовать двухмерный массив, например:

```
<?
  $test_scores["Иванов"][1]=95;
  $test_scores["Иванов"][2]=85;
  $test_scores["Петров"][1] = 87;
```

```
  $test_scores ["Петров"][2] = 93;
  print_r ($test_scores);
?>
```

Элемент массива $test_scores ["Иванов"][1] содержит оценку Иванова по первому предмету, $tests_cores ["Иванов"] [2] – по второму предмету и т.д. В результате выполнения предыдущего примера будет выведен созданный многомерный массив:

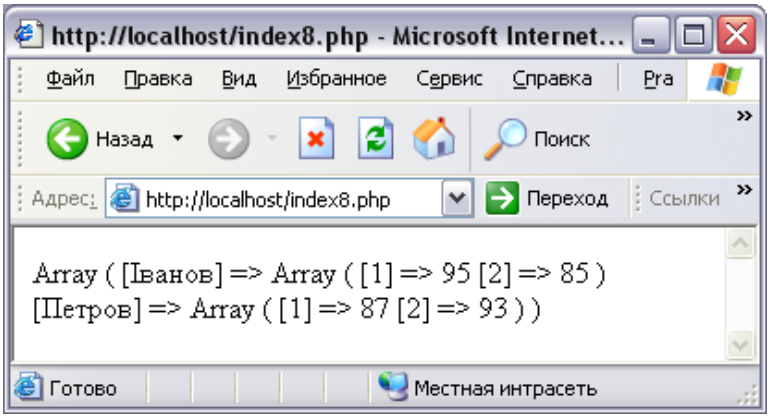

Рисунок 2.8 – Результат вывода двумерного массива

Доступ к элементу многомерного массива осуществляется путем указания всех его индексов, например:

```
echo "Оценка Иванова по первому предмету:",
$test_scores["Иванов"] [1], "\n";
```

Если нужно заменить элемент массива его значением в строковой константе, ограниченной двойными кавычками, подобно тому, как это делается с обычной переменной, следует вложить элемент массива в фигурные скобки и использовать для строчных индексов одиночные кавычки:

```
echo "Оценка Иванова по первому предмету:
{$test_scores['Иванов' ] [1]} "\n";
```

Для создания многомерных массивов можно использовать и сокращенную форму, но следует иметь в виду, что в данном случае нумерация второго индекса начинается с 0:

```
<?
  $test_scores ["Иванов"] [] =95
  $test_scores ["Иванов"] [] =85
  $test_scores ["Петров"] [] =87
  $test_scores ["Петров"] [] =93
  print_r($test_scores);
?>
```

ВPHPМногомерные массивы можно рассматривать как массивы массивов.

К примеру, двухмерный массив можно рассматривать как одномерный массив, элементами которого в свою очередь также являются одномерные массивы. Пример такого описания приведен ниже:

```
<?
 $test_scores = array("Иванов" => array(95, 85),
  "Петров" => array(87, 93)) ;
 print_r($test_scores);
?>
```

В результате будет создан следующий массив:

```
Array ( [Иванов] => Array ( [0] =>95 - [1] =>85 )
[Петров] => Array ( [0] =>87 [1] =>93 ) )
```

Если необходимо начать нумерацию второго индекса с 1, используется уже известный синтаксис:

```
<?
  $test_scores = array ("Иванов" => array A => 95, 85)
  "Петров" => array A => 87, 93)));
  print_r($test_scores);
?>
```

В результате будет создан массив следующего вида:

```
Array ( [Иванов] => Array ( [1]=>95 [2]=>85 )
[Петров] => Array ( [1]=>87 [2]=>93 ) )
```

Многомерные массивы и циклы

Часто возникает необходимость взять все элементы многомерного массива в цикле. Для этого используются вложенные циклы. К примеру, для случая двухмерного массива внешний цикл перебирает первый индекс массива, а внутренний цикл перебирает второй индекс. Пример вложенных циклов приведен ниже.

```
<?
$test_scores [0] []= 95;
$test_scores [0] []= 85;
$test_scores [1] []= 87;
$test_scores [1] []= 93;
for($outer_index = 0;
 $outer_index <count ($test_scores);
 $outer_index++)
{ for ($inner_index = 0;
 $inner_index < count ($test_scores
[$outer_index]); $inner_index++)
{
echo "\$test_scores [$outer_index] [$inner_index] = ",
$test_scores[$outer_index] [$inner_index], "\n";
} }
```

```
?>
```

В результате выполнения этого кода будет выведен начальный двухмерный массив:

```
$test scores [0] [0] = 95
$test scores [0] [1] = 85
$test scores [1] [0] = 87
$test scores [1] [1] = 93
```

Для этой же цели могут использоваться циклы foreach, а для случая строчных индексов это наилучший способ, поскольку строчные индексы нельзя увеличивать оператором ++. В следующем примере при каждом исполнении тела наружного цикла из исходного массива извлекается одномерный массив, соответствующий значению счетчика цикла.

```
<html>
<head>
<title> Цикл по всем элементам двухмерного массива
 </title>
</head>
<body>
<h3>Цикл по всем элементам двухмерного массива </h3>
<?
 $test_scores["Иванов"]["первый"] = 95;
 $test_scores["Иванов"]["второй"] = 85;
 $test_scores["Петров"] ["первый"] = 87;
 $test_scores["Петров"]["второй"] = 93;
foreach ($test_scores as $outer_key => $single_array)
{
 echo "$value <BR>" ;
 }

foreach ($single_array as $inner_key => $value)
{
 echo "$test_scores[$outer_key][$inner_key] <Br>";
}
?>
</body>
</html>
```

Цикл по всем элементам двухмерного массива

```
$tests_cores [Иванов] [первый] = 95
$tests_cores [Иванов] [второй] = 85
$tests_cores [Петров] [щзвый] = 87
$tests_cores [Петров] [второй] = 93
```

Операторы над массивами

Кроме различных функций, над массивами можно производить действия с помощью операторов.

$a + $b	Объединение	Объединение массивов ab
$a = $b	Равняется	TRUE, если $a и $b содержат одни и те же элементы
$a == $b	Тождественно равно	TRUE, если $a и $b содержат одни и те же элементы в том же 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 тождественно не равен массиву $b

Контрольные вопросы

1. Что такое массив?
2. Что такое ассоциативный массив?
3. В чем особенность использования массивов в PHP?
4. Как создать массив?
5. Как добавить или удалить элемент массива?
6. Какие функции для сортировки массивов вам известны? В чем их особенность?
7. Как вычислить сумму элементов массива?
8. Как осуществлять навигацию по элементам массива?
9. Какие есть операторы работы с массивами?

Лабораторная работа №3

Тема:PHP. Работа с формами

Цель:Научиться прорабатывать формы на Web-страницах с помощью сценариев на языкеPHP

Задание для выполнения

1. Реализовать на практике пример обработчика формы для ввода логина и пароля.
2. Разработать сценарий для вывода приветствия пользователю, который вводит свое имя с помощью формы. Для передачи данных использовать метод GET.
3. Разработать сценарий вычисления суммы, разницы, произведения, доли и остатка для введенных двух чисел. Проверить его исполнение. Для передачи данных можно использовать метод POST.
4. Разработать сценарий для проведения тестирования (4-5 вопросов) и вывода результатов анкетирования пользователя.
5*. Разработать сценарий для форматирования введенного текста (предусмотреть определение следующих свойств текста: цвета, гарнитуры, кегля).
6*. Разработать сценарий для вычисления количества щелчков на каждую из двух кнопок в течение одного сеанса.

Отчет должен содержать:

1. Тему, цель работы
2. Коды созданных файлов и копию окна обозревателя при их просмотре.

Методические указания

Протокол HTTP (HyperText Transfer Protocol, протокол передачи гипертекста) реализует принцип запрос/ответ. Запрашивающая программа – клиент инициирует взаимодействие с отвечающей программой – сервером и отправляет запрос, содержащий:

- метод доступа;
- адрес URL;
- версию протокола;
- сообщения (похоже по форме на MIME) с информацией о типе передаваемых данных, информацией о посланном запросе клиенте и, возможно, с содержательной частью (телом) сообщения.

Ответ сервера содержит:

- срока состояния, в которую входит версия протокола и код возврата (успех или ошибка);
- сообщение (в форме, похожей на MIME), в которое входит информация сервера, метаинформация (т.е. информация о содержании сообщения) и тело сообщения.

В протоколе не указывается, кто должен открывать и закрывать соединение между клиентом и сервером. На практике соединение, как правило, открывает клиент, а сервер после отправки ответа инициирует его разрыв.

Рассмотрим более подробно, в какой форме отправляются запросы на сервер.

Клиент отправляет серверу запрос в одной из двух форм: в полной или сокращенной. Запрос в первой форме называется соответственно полным запросом, а во второй форме – простым запросом.

Простой запрос содержит метод доступа и адрес ресурса. Формально это можно записать так:

<Простой-Запрос>:=<Метод><символ пробел><Спрашиваемый-URL><символ новой строки>

В качестве метода может быть указан GET, POST, HEAD, PUT, DELETE и другие. В качестве запрашиваемого URL чаще всего используется URL-адрес ресурса.

Пример простого запроса:

GET http://phpbook.info/

Здесь GET – это метод доступа, то есть метод, который должен быть применен к запрашиваемому ресурсу, а http://phpbook.info/ – это URL-адрес запрашиваемого ресурса.

Полный запрос содержит строку состояния, несколько заголовков (заголовок запроса, общее заглавие или заглавие содержания) и, возможно, тело запроса. Формально общий вид полного запроса можно записать так:

<Полный запрос> := <Строка Состояния>
(<Общий заголовок>|<Заголовок запроса>|<Заголовок содержания>)
<символ новой строки>
[<содержание запроса>]

Квадратные скобки здесь обозначают необязательные элементы заголовка, из-за вертикального риска перечислены альтернативные варианты. Элемент <Страница состояния> содержит метод запроса и URL ресурса (как и простой запрос) и, кроме того, используемую версию протокола HTTP. К примеру, для вызова внешней программы можно задействовать следующую строку состояния:

POST http://phpbook.info/cgi-bin/test HTTP/1.0

В этом случае используется метод POST и протокол HTTP версии 1.0.

В обеих формах запроса важное место занимает URL запрашиваемого ресурса. Чаще всего URL используется в виде URL ресурса. При обращении к серверу можно использовать как полную форму URL, так и упрощенную.

Полная форма содержит тип протокола доступа, адрес сервера и адрес ресурса на сервере (рис. 3.1).

В сокращенной форме опускают протокол и адрес сервера, указывая только местоположение ресурса от корня сервера. Полную форму используют, если возможна пересылка запроса другому серверу. Если же работа производится только с одним сервером, то чаще применяют сокращенную форму.

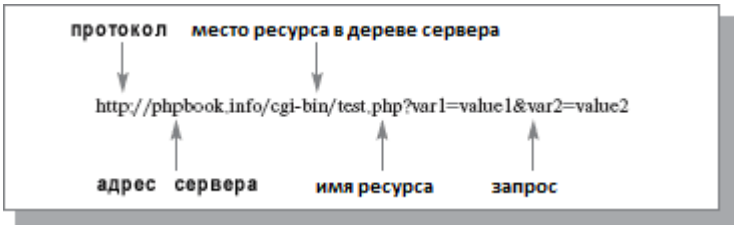

Рисунок 3.1 – Полная форма URL

Метод уведомляет о цели запроса клиента. Протокол HTTP поддерживает достаточно много методов, но реально используются только три:

GET
HEAD
POST

Метод GET позволяет получить любые данные, идентифицированные с помощью URL в запросе ресурса. Если URL указывает на программу, то возвращается результат работы программы, а не ее текст (если, конечно, текст не результат ее работы). Дополнительная информация, необходимая для обработки запроса, встраивается в сам запрос (в строку статуса). При использовании метода GET в поле тела ресурса возвращается вызванная собственно информация (текст HTML-документа, например).

Существует разновидность метода GET – условный GET. Этот метод сообщает серверу о том, что на запрос нужно ответить, только если выполнено условие, содержащееся в поле if-Modified-Since заголовка запроса. Если говорить более точно, то тело ресурса передается в ответ на запрос, если этот ресурс изменялся после даты указанной в if-Modified-Since.

Метод HEAD аналогичен методу GET, только не возвращает тело ресурса и не имеет условного аналога. Метод HEAD используется для получения информации о ресурсе. Это может пригодиться, например, при решении задачи тестирования гипертекстовых ссылок.

Метод POST разработан для передачи на сервер такой информации как аннотации ресурсов, новости и почтовые сообщения, данные для добавления в базу данных, то есть для передачи информации большого объема и достаточно важной. В отличие от методов GET и HEAD, в POST передается тело ресурса, что и является информацией, получаемой с полей форм или других источников ввода.

Использование HTML-форм для передачи данных на сервер

Для метода GET

При отправке данных формы с помощью метода GET содержимое формы добавляется к URL после вопросительного знака в виде пар имя=значения, объединенных с помощью амперсанда &:

```
action?name1=value1&name2=value2&name3=value3
```

Здесь action – это URL-адрес программы, который должен обрабатывать форму (это либо программа, заданная в атрибуте action тега form, либо сама текущая программа, если этот атрибут опущен). Имена name1, name2, name3 соответствуют именам элементов формы, а value1, value2, value3 – значению этих элементов. Все специальные символы, включая = &, в именах или значениях этих параметров будут опущены.

Поэтому не следует использовать в названиях или значениях элементов формы эти символы и символы кириллицы в идентификаторах.

Если в поле для ввода ввести какой-нибудь служебный символ, то он будет передан в его шестнадцатилетнем коде, например символ $ заменится на %24. Так же передаются и русские буквы.

Для полей ввода текста и пароля(это элементы input с атрибутом type=text и type=password), значением будет то, что пользователь введет. Если пользователь ничего не вводит в поле, то в строке запроса будет присутствовать элемент name=, где name соответствует имени этого элемента формы.

Для кнопок типа checkbox и radioзначение value определяется атрибутом VALUE в том случае, когда кнопка отмечена. Не отмеченные кнопки при составлении строки запроса игнорируются полностью. Несколько кнопок типа checkbox могут иметь один атрибут NAME (и разные VALUE), если это необходимо. Кнопки типа radio предназначены для одного из всех предложенных вариантов, и поэтому должны иметь одинаковый атрибут NAME и различные атрибуты VALUE.

В принципе создавать HTML-форму для передачи данных по методу GET не обязательно. Можно просто добавить в строчку URL нужные переменные и их значения.

```
http://www.ru/test.php?id=10&user=pit
```

В этой связи в передаче данных методом GET есть один существенный недостаток – любой может подделывать значение параметров. Поэтому не советуют использовать этот метод для доступа к защищенным паролем страницам, для передачи информации, влияющей на безопасность работы программы или сервера. Кроме того, не следует применять метод GET для передачи информации, которую не разрешено изменять пользователю.

Несмотря на все эти недостатки, использовать метод GET достаточно удобно при отладке скриптов (тогда можно видеть значения и передаваемые имена переменных) и для передачи параметров, не влияющих на безопасность.

Для метода POST

Содержимое формы кодируется точно так же, как и для метода GET, но вместо добавления строки в URL содержимое запроса ссылается блоком данных как часть операции POST. Если присутствует атрибут ACTION, то значение URL, находящееся там, определяет, куда посылать этот блок данных. Этот метод, как отмечалось, рекомендуется для передачи больших по объему блоков данных.

Информация, введенная пользователем и отправленная серверу с помощью метода POST, подается на стандартный ввод программе, указанной в атрибуте action, или текущем скрипте, если этот атрибут опущен. Длина ссылаемого файла передается в переменной CONTENT_LENGTH, а тип данных – в переменной CONTENT_TYPE.

Передать данные методом POST можно только с помощью HTML-формы, поскольку данные передаются в теле запроса, а не в заглавии, как в GET. Соответственно и изменить значение параметров можно только изменив значение, введенное в форму. При использовании POST пользователь не видит передаваемые серверу данные.

Основное преимущество запросов POST – это их большая безопасность и функциональность по сравнению с GET-запросами. Поэтому метод POST чаще используется для передачи важной информации, а также информации большого объема. Однако не

стоит полностью полагаться на безопасность этого механизма, поскольку данные запроса POST также можно подделывать, например создав html-файл на своей машине и заполнив его нужными данными. Кроме того, не все клиенты могут использовать метод POST, ограничивающий варианты его использования.

При отправке данных на сервер любым методом передаются не только сами данные, введенные пользователем, но и ряд переменных, которые называются переменными окружающими, характеризуют клиента, историю его работы, пути к файлам и т.п. Вот некоторые из переменных окружения:

· REMOTE_ADDR – IP-адрес хоста, отправляющий запрос;

·REMOTE_HOST – имя хоста, из которого отправлен запрос;

·HTTP_REFERER – адрес страницы, ссылающийся на текущий скрипт;

·REQUEST_METHOD – метод, который был использован при отправке запроса;

·QUERY_STRING – информация, находящаяся в URL после знака вопроса;

·SCRIPT_NAME – виртуальный путь к программе, которая должна выполняться;

·HTTP_USER_AGENT – информация о браузере, который использует клиент.

Обработка запросов с помощью PHP

Внутри PHP-скрипта существует несколько способов получения доступа к данным, передаваемым клиентом по протоколу HTTP. До версии PHP 4.1.0 доступ к таким данным осуществлялся по именам переданных переменных. Таким образом, если, например, передано first_name=Nina, то внутри скрипта появлялась переменная $first_name со значением Nina. Если нужно было различать, каким методом были переданы данные, то использовались ассоциативные массивы \$HTTP_POST_VARS и \$HTTP_GET_VARS, ключами которых были имена переданных переменных, а значениями соответственно значения этих переменных. Таким образом, если пара first_name=Nina передана методом GET, то

```
$HTTP_GET_VARS["first_name"]="Nina"".
```

Использовать в программе имена переданных переменных напрямую опасно. Поэтому было решено начиная с PHP 4.1.0 задействовать для обращения к переменным, передаваемым с помощью HTTP-запросов, специальный массив – $_REQUEST. Этот массив содержит данные, передаваемые методами POST и GET, а также посредством HTTP cookies. Это суперглобальный ассоциативный массив, то есть его значения можно приобрести в любом месте программы, используя как ключ имя соответствующей переменной (элемента формы).

Пример

Создана форма для регистрации участников заочной школы программирования. Тогда в файле action.php, обрабатывающем эту форму, можно написать следующее:

```
<?php $str = "Здравствуйте
 ".$_REQUEST["first_name"]."
 ".$_REQUEST["last_name"]."! <br>";
$str .="Вы выбрали для изучения курс по
 ".$_REQUEST["kurs"];
```

```
echo $str; ?>
```

Тогда, если в форму мы ввели имя "Вася", фамилию "Петров" и выбрали среди всех курсов курс по PHP, на экране браузера получим следующее сообщение:

Здравствуйте, Вася Петров!
Вы выбрали для изучения курс по PHP

После ввода массива $_REQUEST массивы $HTTP_POST_VARS и $HTTP_GET_VARS для однородности были переименованы в $_POST и $_GET соответственно, но сами они из употребления не исчезли из соображений совместимости с предыдущими версиями PHP. В отличие от своих предшественников, массивы $_POST и $_GET стали суперглобальными, то есть доступными непосредственно внутри функций и методов.

Рисунок 3.2 – Пример внешнего вида формы

Приведем пример использования этих массивов. Допустим, нам нужно обработать форму, содержащую элементы ввода с именами first_name, last_name, kurs (например, форму, приведенную выше). Данные были переданы методом POST и данные, переданные другими методами, мы обрабатывать не хотим. Это можно сделать следующим образом:

```
<?
php $str = "Здравствуйте
 ".$_POST ["first_name"]."
 ".$_POST ["last_name"] ."! <br>";
$str .= "Вы выбрали для изучения курс по".
 $_POST["kurs"];
echo $str;
?>
```

Тогда на экране браузера, если мы ввели имя «Вася», фамилию «Петров» и выбрали среди всех курсов курс по PHP, увидим сообщение как в предыдущем примере:

Здравствуйте, Вася Петров!
Вы выбрали для изучения курс по PHP

Чтобы сохранить возможность обработки скриптов предыдущих версий, чем PHP 4.1.0, была введена директива register_globals, решающая или запрещающая доступ к переменным непосредственно по их именам. Если в файле настроек PHP параметр register_globals=On, то к переменным, переданным серверу методами GET и POST, можно обращаться прямо по их именам (можно писать $first_name). Если register_globals=Off, то нужно писать

```
$_REQUEST["first_name"]
или
$_POST["first_name"],
$_GET["first_name"],
$HTTP_POST_VARS["first_name"],
$HTTP_GET_VARS["first_name"].
```

С точки зрения безопасности, эту директиву лучше отключать (register_globals=Off). При включенной директиве register_globals вышеперечисленные массивы также будут содержать данные, передаваемые клиентом.

Иногда возникает необходимость узнать значение какой-либо переменной окружения, например метод, использовавшийся при передаче запроса или IP-адрес отправившего запрос компьютеру. Получить эту информацию можно с помощью функции getenv(). Она возвращает значение переменной окружения, имя которой передано ей в качестве параметра.

```
<? getenv("REQUEST_METHOD");
 // вернет использованный метод
echo getenv ("REMOTE_ADDR");
 // выведет IP-адрес пользователя
 // пославший запрос
?>
```

Если используется метод GET, данные передаются добавлением строки запроса в виде пар «имя_переменной=значение к URL-адресу ресурса». Все, что записано в URL после вопросительного знака, можно получить с помощью команды

```
getenv("QUERY_STRING");
```

Благодаря этому можно по методу GET передавать данные в каком-нибудь другом виде. Например, указывать только значение нескольких параметров через знак плюс, а в скрипте разбирать строку запроса на части или можно передавать значение всего одного параметра. В этом случае в массиве $_GET появится пустой элемент с ключом, равным этому значению (всей строке запроса), причем символ «+», встретившийся в строке запроса, будет заменен на подчеркивание «_».

Методом POST данные передаются только посредством форм, и пользователь (клиент) не видит, какие именно данные отправляются серверу. Чтобы их увидеть, хакер должен подменить нашу форму своей. Тогда сервер отправит результаты обработки неправильной формы не туда, куда следует. Чтобы этого избежать, можно проверять адрес страницы, с которой были посланы данные. Это можно сделать опять же с помощью функции getenv():

```
getenv("HTTP_REFERER");
```

Пример обработки запроса с помощью PHP

Следует написать обработчик формы для ввода логина и пароля и вывода их на экран.

Форма нужна иметь следующий вид:

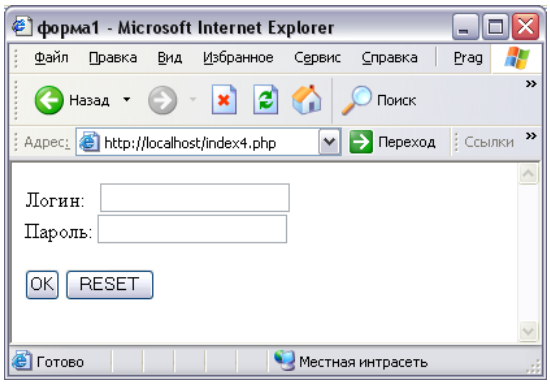

Рисунок 3.3 – Форма для ввода логина и пароля

После ввода данных и нажатия кнопки «ОК» на странице должна появиться следующая информация:

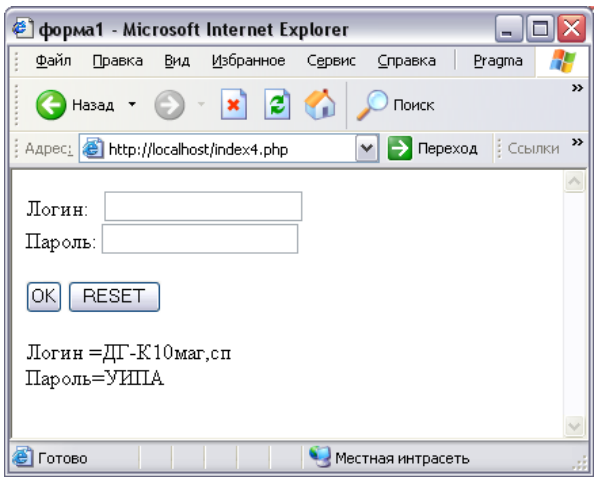

Рисунок 3.4. Результат работы обработчика формы

Если форма и обработчик находятся в одном файле, код страницы будет следующим:

Листинг файла index.php (обработчик и форма в одном файле)

```
<html>
<head>
<title> форма1</title>
</head>
<body>
<form method="post">
Логин:   <input name="user"><br>
Пароль: <input name="pass" type="password"><br><br>

<input type="submit"  value="OK">
<input type="reset"   value="RESET">
</form>

<?
$user=$_POST['user'];
$pass=$_POST['pass'];
if (isset($user)) echo "Логин  =$user<br>";
if (isset($pass)) echo "Пароль=$pass<br>";
?>
</body>
</html>
```

В случае, если форма и обработчик находятся в разных файлах, страницы и их код будут выглядеть следующим образом:

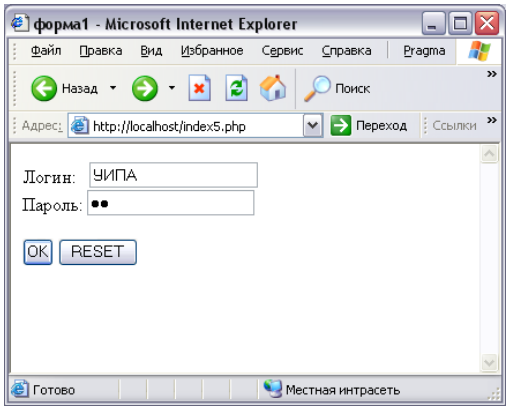

Рисунок 3.5 – Форма для ввода логина и пароля

Листинг файла index5.php (форма)

```
<html>
<head>
<title> форма1</title>
</head>
<body>
<form method="post" action="act.php">
Логин:   <input name="user"><br>
Пароль: <input name="pass" type="password"><br><br>
<input type="submit"  value="OK">
<input type="reset"   value="RESET">
</form>
</body>
</html>
```

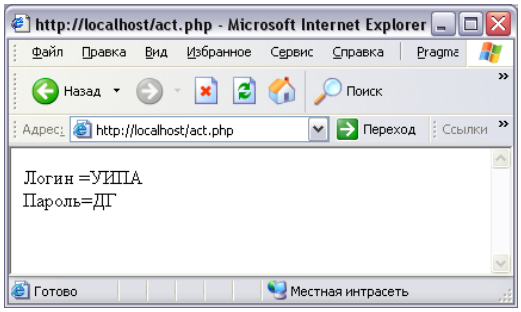

Рисунок 3.6 – Результат работы обработчика формы

Листинг файла Act.php (обработчик)

```
<?
$user=$_POST['user'];
$pass=$_POST['pass'];
if (isset($user)) echo "Логин  =$user<br>";
if (isset($pass)) echo "Пароль=$pass<br>";
?>
```

Контрольные вопросы

1. Что такое форма? Зачем она предназначена?
2. Какие элементы могут содержаться в форме? Как создать их?
3. Для чего предназначен протокол HTTP?
4. Что такое способ доступа?
5. Какие массивы используются для передачи данных?
6. Охарактеризуйте способ GET.
7. Охарактеризуйте способ POST.
8. Для чего необходима функция getenv()?

Лабораторная работа №4

Тема:PHP. Работа с файлами

Цель:Достояние практических навыков и умений работы с файлами с помощью сценариев на языкеPHP

Задание для выполнения

1. Ознакомиться с методическими указаниями по данной теме.
2. Создать в текущем каталоге произвольный текстовый файл, содержащий несколько строк.
   Средствами PHP
3. Отобразить содержимое этого файла.
4. Определите параметры этого файла.
5. Вставить после первой строки фразу «Мы изучаем PHP» и добавить к концу файла фразу «Все готово!».
6. Отобразить содержимое измененного файла.
7. Создать новый файл, в который можно записать первую и третью строку из предыдущего файла.
8. Отобразить содержимое нового файла.
9. Отобразить содержимое текущего каталога.

Отчет должен содержать:

1. Тему, цель работы
2. Коды созданных файлов и копию окна обозревателя при их просмотре.

Методические указания

Файл– именуемая область носителя информации, предназначенная для хранения логически связанной информации. В сценариях на языке PHP, как и в других языках программирования, есть возможность обработки информации, хранящейся в файлах, а также работы с файлами и каталогами. Наиболее используются следующие функции:

Открытие файла

fopen

Открывает файл и привязывает его к дескрипторе.

Синтаксис:

```
int fopen(string $filename, string $mode, bool
$use_include_path=false)
```

Открывает файл с именем $filename в режиме $mode и возвращает дескриптор открытого файла. Если операция провалилась, то функция возвращает false. Необязательный параметр use_include_path говорит о том, что если задано относительное имя файла,

его следует искать также и в списке путей, устанавливающих инструкции include и require. Обычно этот параметр не используется.

Параметр $mode может принимать следующие значения:

r− Файл открывается только для чтения. Если файл не существует, вызов регистрирует ошибку. После удачного открытия указатель файла устанавливается на его первый байт, то есть до начала.

r+− Файл открывается одновременно для чтения и записи. Указатель текущей позиции устанавливается на первый байт. Если файл не существует возвращает false. Если в момент записи указатель файла установлен где-то внутри файла, то данные запишутся прямо поверх уже имеющегося, а не раздвинут их, при необходимости увеличив размер файла.

w− Создает новый пустой файл. Если к моменту вызова уже был файл с таким именем, то он заранее уничтожается. В случае неверно заданного имени файла вызов "проваливается".

w+− Аналогичный r+, но если файл изначально не существовало, создает его. После этого с файлом можно работать как в режиме чтения, так и в записи. Если файл существовал до момента вызова, его содержимое удаляется.

a− Открывает существующий файл в режиме записи, и при этом перемещает указатель текущей позиции за последний байт файла. Вызов неуспешен при отсутствии файла.

a+− Открывает файл в режиме чтения и записи, указатель файла устанавливается на конец файла, при этом содержимое файла не уничтожается. Отличается от того, что если файла первоначально не существовало, то он создается. Этот режим полезен, если вам нужно что-то дописать в файл, но вы не знаете, создан ли уже такой файл.

Но это еще не полное описание параметра $mode. Дело в том, что в конце любой из строк r, w, a, r+, w+ и a+ может находиться еще один необязательный символ – b или t. Если указанный b (или не указан вообще никакой), то файл открывается в режиме бинарного чтения/записи. Если это t, то для файла устанавливается режим трансляции символа перевода строки, то есть он воспринимается как текстовый.

tmpfile

Создает новый временный файл с уникальным именем и открывает его для чтения и записи.

Синтаксис:`int tmpfile()`

В дальнейшем вся работа должна вестись с возвращенным файловым дескриптором, потому что имя файла недоступно. Пространство, занимаемое временным файлом, автоматически освобождается при его закрытии и при завершении работы программы.

Закрытие файла

fclose

Закрывает файл, заранее открытый функцией fopen().

Синтаксис:`int fclose(int $fp)`

Возвращает false, если файл закрыть не удалось (например, что-то с ним случилось или разорвалась связь с удаленным хостом). Иначе возвращает значение "истина". Всегда нужно закрывать FTP- и HTTP-соединение, потому что иначе"беспризорный

файл приведет к неоправданному простою канала и лишней загрузке сервера. Кроме того, успешно закрыв соединение, вы будете уверены в том, что все данные были доставлены без ошибок.

Чтение и запись

fread
Читает определенное количество символов из открытого файла.

Синтаксис:`string fread(int $f, int $numbytes)`

Читает из файла $f $numbytes символы и возвращает строку этих символов. После прочтения указатель файла продвигается к следующей после прочитанного блока позиции. Если $numbytes больше, чем можно прочитать из файла, возвращается то, что удалось считать. Этот прием можно использовать, если вам нужно считать в строку файл целиком. Для этого просто задается $numbytes очень большим числом.

fwrite
Запись в файл.

Синтаксис:`int fwrite(int $f, string $str)`

Записывает в файл $f все содержимое строки $str. Эта функция составляет пару для fread(), действуя "в обратном направлении". При работе с текстовыми файлами (т.е. когда указан символ t в режиме открытия файла) все «\n» автоматически преобразуются в разделитель строк, который принят в вашей операционной системе.

fgets
Читает из файла одну строку, заканчивающуюся символом новой строки \n.

Синтаксис:`string fgets(int $f, int $length)`

Этот символ также считывается и включается в результат. Если строка в файле занимает больше $length-1 байтов, то возвращаются только его $length-1 символы. Функция полезна, если вы открыли файл и хотите "пройтись" по всем его строчкам. Однако даже в этом случае (и быстрее) будет пользоваться функцией file(). Следует также отметить, что эта функция (как и функция fread()) в случае текстового режима в Windows будет преобразовывать пары \r\n в один символ \n.

fgetss
Читает строку файла с удалением из него меток HTML.

Синтаксис: `string fgetss (int$file, int $length [, string allowable_tags])`

Необязательный третий параметр allowable_tags может содержать строку со списком тегов, которые не должны быть отброшены, при этом теги в строке записываются через запятую.

fputsПолный аналог fwrite().

Синтаксис:`int fputs(int $f, string $str)`

Отображение содержимого файла

fpassthru
Отображение содержимого открытого файла в окне обозревателя.
Синтаксис: `int fpassthru (int$file)`

***Пример*:**
```
<? $file = fopen("ris.jpg", "rb");
 if(!file)
 { echo("Ошибка открытия файла"); }
 else
 { fpassthru($file); }
?>
```

readfile
Отображение содержимого текстового файла.

Синтаксис:readfile (string filename)

Параметр filename – имя файла, а не его дескриптор.

Пример:
```
<?
 readfile("file.txt");
?>
```

Положение указателя текущей позиции

feof
Указатель конца файла.

Синтаксис:`int feof(int $f)`

Возвращает true, если достигнут конец файла (т.е. если указатель файла установлен по концу файла).

Пример:
```
$f=fopen("myfile.txt","r");
while(!feof($f))
{ $str=fgets($f);
// Обрабатываем очередную строчку $str
}
fclose($f);
```

fseekУстановка указателя файла на определенную позицию.

Синтаксис:`int fseek(int $f, int $offset,`
`int $whence=SEEK_SET)`

Устанавливает указатель файла на байт со смещением $offset (от начала файла, от его конца или от текущей позиции в зависимости от параметра $whence). Это может не

сработать, если дескриптор $f ассоциирован не с обычным локальным файлом, а с соединением HTTP или FTP. Параметр $whence задает с какого места отсчитывается смещение $offset. В PHP для этого существуют три константы, равные, соответственно, 0, 1 и 2:

SEEK_SET – устанавливает позицию начиная с начала файла;
SEEK_CUR – отсчитывает позицию относительно текущей позиции;
SEEK_END − отсчитывает позицию относительно конца файла;

При использовании последних двух констант параметр $offset вполне может быть отрицательным (а при применении SEEK_END он будет отрицательным наверняка). При успешном завершении эта функция возвращает 0, а в случае неудачи –1.

ftell
Возвращает положение указателя файла.

Синтаксис:`int ftell(int $f)`

Функции для определения типов файлов

file_exists
Проверка существования вызываемого файла.

Синтаксис:`bool file_exists(string filename)`

Возвращает true, если файл с именем filename существует на момент вызова. Следует использовать эту функцию с осторожностью. К примеру, следующий код никуда не годится с точки зрения безопасности:

```
$fname="passwd";
if(!file_exists($fname))
$f=fopen($fname, "w");
 else
$f=fopen($fname,"r");
```

Дело в том, что между вызовом file_exists() и открытием файла в режиме w проходит некоторое время, за которое другой процесс может вклиниться и подменить используемый нами файл. Функция не работает с удаленными файлами, файл должен находиться в доступной для сервера файловой системе.

filetype
Возвращает тип файла.

Синтаксис:`string filetype(string filename)`

Возвращает строку, описывающую тип файла с именем filename. Если файл не существует, возвращает false. После вызова строка будет содержать одно из следующих значений:

file −обычный файл;
dir− каталог;
unknown− неизвестный тип файла;

is_file
Проверка обычного файла.

Синтаксис:`bool is_file(string filename)`

Возвращает true, если filename – обычный файл.

is_dir
Проверка существования каталога.

Синтаксис:`bool is_dir(string filename)`

Возвращает true, если каталог filename существует.

is_readable
Проверка файла, доступного для чтения.

Синтаксис:`bool is_readable(string filename)`

Возвращает true, если файл может быть открыт для чтения. Обычно PHP осуществляет доступ к файлу с привилегиями пользователя, запускающим web-сервер (часто "nobody"). Соображения безопасности должны учитываться.

is_writeable
Проверка файла, доступного для записи.

Синтаксис:`bool is_writeable(string filename)`

Возвращает true, если файл можно писать. Обычно PHP осуществляет доступ к файлу с привилегиями пользователя, запускающим web-сервер (часто "nobody").

is_executable
Проверка существования запускаемого файла.

Синтаксис:`bool is_executable(string filename)`

Возвращает true, если файл filename – исполняемый.

is_uploaded_file
Проверка существования файла, загруженного методом HTTP POST.

Синтаксис:`bool is_uploaded_file(string filename)`

Возвращает true, если файл с именем filename был загружен на сервер с помощью HTTP POST. Часто это полезно, чтобы убедиться, что пользователи по злому умыслу не пытались заставить сценарий работать с файлами, с которыми им работать не следует.

Определение параметров файла

stat

Функция собирает вместе всю информацию, выдаваемую операционной системой для указанного файла, и возвращает ее в виде массива.

Синтаксис:`array stat(string $filename)`

Этот массив всегда содержит следующие элементы с указанными ключами:

0 – устройство;
1 − Номер узла inode;
2 − атрибуты защиты файла;
3 − число синонимов ("жестких" ссылок) файла;
4 – идентификатор uid владельца;
5 – идентификатор gid группы;
6 − тип устройства;
7 − размер файла в байтах;
8 – время последнего доступа в секундах, прошедших с 1 января 1970 года;
9 − время последней модификации содержимого файла;
10 − время последнего изменения атрибутов файла;
11 – размер блока;
12 – число занятых блоков;

Этот массив содержит информацию, которая доступна в системах Unix. Во Windows многие поля могут быть пусты. Если $filename задает не имя файла, а имя символической ссылки, то все же будет возвращена информация о том файле, на который ссылается эта ссылка (а не о ссылке).

fileatime

Возвращает время последнего доступа к файлу.

Синтаксис:`int fileatime(string filename)`

Время выражается в количестве секунд, прошедших с 1 января 1970 г. (Unix timestamp). Если файл не обнаружен, возвращается false. Атрибут времени последнего доступа к файлу меняется каждый раз, когда данные файла читаются. Поскольку это сильно снижает производительность при интенсивной работе с файлами и каталогами, часто изменение этого атрибута в операционных системах блокируется, и тогда функция не работает.

filemtime

Возвращает время последнего изменения файла или false при отсутствии файла.

Синтаксис:`int filemtime(string $filename)`

filectime

Возвращает время создания файла.

Синтаксис:`int filectime(string $filename)`

filesize

Возвращает размер файла в байтах или false, если файл не существует.

Синтаксис:`int filesize(string $filename)`

touch
Установка времени модификации.

Синтаксис:`int touch(string $filename [, int $timestamp])`

Установка времени модификации указанного файла $filename равным $timestamp (в секундах, прошедших с 1 января 1970 года). Если второй параметр не указан, то подразумевается текущее время. В случае ошибки возвращает false. Если файл с указанным именем не существует, он создается пустым.

Функции работы с именами файлов

basename
Выделяет имя файла из пути.

Синтаксис:`string basename(string $path)`

Выделяет основное имя из пути $path

Примеры:
```
echo basename("/home/somebody/somefile.txt");
 // выводит "somefile.txt"
echo basename("/"); // ничего не выводит
echo basename("/."); // выводит "."
echo basename("/./"); // также выводит "."
```

Функция basename() не проверяет наличие файла. Она просто берет часть строки после самого правого слеша и возвращает ее. Эта функция правильно обрабатывает как прямые, так и обратные слеши под Windows.

dirname
Выделение имени каталога.

Синтаксис:`string dirname(string $path)`

Возвращает имя каталога, выделенное из пути $path. Функция достаточно "умна" и умеет выделять нетривиальные ситуации, описанные в примерах:

```
    echo dirname("/home/file.txt"); // выводит "/home"
    echo dirname("../file.txt"); // выводит ".."
    echo dirname("/file.txt"); // выводит "/" под Unix // "\"
под Windows
    echo dirname("/"); // то же самое
    echo dirname("file.txt"); // выводит "."
```

Если функции dirname() передать просто имя файла, она вернет ".", что означает "текущий каталог".

realpath
Превращает относительный путь в абсолютный.

Синтаксис:`string realpath(string $path)`

Превращает относительный путь $path в абсолютный, то есть начинающийся от корня. Пример:

```
echo realpath("../t.php"); // например, /home/t.php
echo realpath("."); // выводит имя текущего каталога
```

Файл, указанный в параметре $path, должен существовать, иначе функция вернет false.

Функции манипулирования целыми файлами

cpam
Копирует файл.

Синтаксис:`bool copy(string $src, string $dst)`

Копирует файл с именем $src в файл с именем $dst. При этом, если файл $dst на момент вызова существовал, производится его перезапись. Функция возвращает true, если копирование прошло успешно, а в случае неудачи – false. Функция не выполняет переименование файла, если его новое имя находится в другой файловой системе (на другой смонтированной системе в Unix или на другом диске в Windows).

unlink
Удаление файла.

Синтаксис:`bool unlink(string $filename)`

Удаляет файл с именем $filename. В случае неудачи возвращает false, иначе – true.

file
Прочитывает файл и разбивает его по строчкам.

Синтаксис:`List file(string $filename)`

Считывает файл с именем $filename целиком и возвращает массив-список, каждый элемент которого соответствует строке в прочитанном файле. Неудобство этой функции состоит в том, что символы конца строки (обычно \n) не вырезаются из строк файла, а также не транслируются, как это делается для текстовых файлов.

Другие функции

ftruncate
Усекает файл.

Синтаксис:`bool ftruncate(int $f, int $newsize)`

Эта функция удаляет открытый файл $f до $newsize. Разумеется, файл должен быть открыт в режиме, разрешающем запись. К примеру, следующий код очищает весь файл: ftruncate($f,0);

fflush
Запись всех конфигураций в файле.

Синтаксис:`void fflush(int $f)`

Заставляет PHP немедленно записать на диск все изменения, которые производились до этого с открытым файлом $f. Что это за изменения? Дело в том, что для повышения производительности все операции записи в файл буферизируются: к примеру, вызов fputs($f, "Это строчка!") не приводит к непосредственной записи данных на диск − поначалу они попадают во внутренний буфер (традиционно размером 8К). Как только буфер заполняется, содержимое отправляется на диск, а сам он очищается, и все повторяется снова. Особый выигрыш от буферизации ощущается в сетевых операциях, когда просто глупо отправлять данные маленькими порциями.

set_file_buffer
Установка размера буфера.

Синтаксис:`int set_file_buffer(int $f, int $size)`

Эта функция устанавливает размер буфера, о котором говорилось выше, для указанного открытого файла $f. Чаще всего она используется следующим образом: set_file_buffer($f,0); Приведенный код отключает буферизацию для указанного файла, так что теперь все данные, записываемые в файл, немедленно отправляются на диск или сеть.

flock
Блокировка файла.

Синтаксис:
```
bool flock(int $f, int $operation
 [, int $wouldblock])
```

Функция устанавливает для указанного открытого дескриптора файла $f режим блокировки, который хотел бы получить текущий процесс. Этот режим задается аргументом $operation и может быть одной из следующих констант:

• LOCK_SH (или 1) – разделяемая блокировка;
• LOCK_EX (или 2) – исключительная блокировка;
• LOCK_UN (или 3) – снять блокировку;
• LOCK_NB (или 4) – эту константу нужно добавить в одну из предыдущих, если вы не хотите, чтобы программа «подвисала» на flock() в ожидании своей очереди, а сразу возвращала управление.

В случае, если установлен режим без ожидания, и блокировка не была успешно установлена, в необязательный параметр-переменную $wouldblock будет записано значение истина true. В случае ошибки функция возвращает false, а в случае успешного завершения – true.

Манипулирование каталогами

mkdir
Создание каталога.

Синтаксис:`bool mkdir(string $name, int $perms)`

Создает каталог с именем $name и правами доступа perms. Права доступа для каталогов указываются точно так же, как для файлов. Чаще значение $perms устанавливают

равным 0770 (предшествующий ноль обязателен – он указывает PHP на то, что это – восьмеричная константа, а не десятичное число).

***Пример*:**

```
mkdir("my_directory",0755);
 // создает подкаталог в текущем каталоге
mkdir("/data");
 // создает подкаталог data в корневом каталоге
```

В случае успеха функция возвращает true, в противном случае – false.

rmdir
Удаление каталога.

Синтаксис:`bool rmdir(string $name)`

Удаляет каталог с именем $name. Каталог должен быть пустым, а его атрибуты должны разрешать это. В случае успеха функция возвращает true, в противном случае – false.

chdir
Изменение текущего каталога.

Синтаксис:`int chdir(string $directory)`

Изменение текущего каталога PHP на directory. Возвращает FALSE если не может изменить, TRUE если изменение произошло. Параметр $directory может определять и относительный путь, задаваемый от текущего каталога.

***Примеры*:**

```
chdir("/tmp/data"); // переход по абсолютному пути
chdir("./js"); // переход в подкаталог текущего каталога
chdir(".."); // переход в родительский каталог
```

getcwd
Полный путь.

Синтаксис:`string getcwd()`

Данная функция возвращает текущую директорию, в отношении которой производятся файловые операции, то есть возвращает полный путь к текущему каталогу, начиная от "корня" (/). Если такой путь не может быть отслежен, вызов проваливается и возвращается false.

diskfreespace
Определение свободного пространства в каталоге

Синтаксис:`float diskfreespace (string directory)`

Данная функция возвращает в байтах свободное пространство в каталоге directory, то есть в соответствующей ей файловой системе или разделе диска.

***Пример*:**

```
$diskspace=diskfreespace("/");
// Тем самым определили свободное место в корневой директории
```

Работа с записями

dir

Класс каталога (псевдо-объектно нацеленный механизм).

Синтаксис:`new dir(string directory)`

Псевдо-объектно-ориентированный механизм для получения списка файлов каталога. Открывает каталог из directory. После этого становятся доступны два свойства объекта: дескриптор каталога handle и строка path, указывающая, какой каталог сейчас используется. Эти свойства доступны, если только каталог был открыт. Свойство handle может быть использовано вместе с другими функциями работы с каталогом типareaddir(),rewinddir()iclosedir(). Для класса доступны три метода: чтение, возврат к началу и закрытию (read, rewind и close соответственно).

Пример:

```
$d = dir("/etc");
echo "Handle: ".$d->handle."<br>\n";
echo "Path: ".$d->path."<br>\n";
while($entry=$d->read()){ // Последовательно выводить
echo $entry."<br>\n"; // имя каждого файла
} // имеющегося в каталоге
$d->close();
```

closedir

Закрыть дескриптор(handle) каталога.

Синтаксис:`void closedir(int dir_handle)`

Закрывает поток каталога, обозначенный как dir_handle. Поток заранее должен быть открыт функцией opendir().

opendir

Откройте дескриптор каталога.

Синтаксис:`int opendir(string path)`

Возвращает дескриптор открытого каталога path, который в дальнейшем используется в функцияхclosedir(),readdir(), иrewinddir().

readdir

Получение имени следующего файла в списке каталога.

Синтаксис:`string readdir(int dir_handle)`

Возвращает имя следующего файла из каталога. Имена файлов возвращаются в виде неупорядоченной последовательности.

Пример:

```
<?
$handle=opendir(".");
echo "Directory handle: $handle\n";
echo "Files:\n";
```

```
while ($file = readdir($handle)) {
echo "$file\n";
} closedir($handle);
?>
```

Следует отметить, что функция возвращает значение "." и "..". Если эти значения не нужны, то их можно исключить следующим образом:

```
<?
$handle=opendir(".");
while($file=readdir($handle)){
if($file !="." && $file != ".."){
echo "Имя файла: $file<br>";
}; };
closedir($handle);
?>
```

rewinddir

Реинициализация дескриптора каталога.

Синтаксис:`void rewinddir(int dir_handle)`
После вызова этой функции функция readdir() с аргументом dir_handle будет возвращать имена файлов из начала в списке каталог

Контрольные вопросы

1. Что такое файл?
2. Как выполнить открытие файла? Какие параметры можно использовать?
3. Как осуществляется закрытие файлов? Обязательно ли выполнять эту процедуру?
4. Как выполнить чтение содержимого файла?
5. Как можно записать данные в файл?
6. Как просмотреть содержимое файла?
7. Какие функции существуют для работы с целыми файлами?
8. Какие функции и для чего используются при работе с каталогами?
9. Какие функции предназначены для определения параметров файла?

```
return True;
}
exit;
}
?>
```

Контрольные вопросы

1. Для чего предназначена функция date()?
2. Какие параметры имеет функция date()?
3. Как часто используется эта функция?
4. Как можно генерировать пароли?
5. Как отправить письмо с вложением? Какие функции при этом используются?

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кузнецов М. В., Симдянов И. В. Самоучитель PHP 5.– СПб.: БХВ-Петербург, 2006.– 608 с.
2. Дамашке Г. PHP и MySQL. – М.: НТ Пресс, 2008. – 314 с.
3. Стеймец В., Вард Б. PHP: 75 готовых решений для вашего Web-сайта. – СПб.: Наука и техника, 2009. – 256 с.
4. Колисниченко, Денис PHP и MySQL. Разработка Web-приложений / Денис Колисниченко. - М.: БХВ-Петербург, 2013. - 560 с.
5. Колисниченко, Денис Профессиональное программирование на PHP (+CD-ROM) / Денис Колисниченко. - М.: БХВ-Петербург, 2015. - 416 с.
6. Кузнецов, М. PHP. Практика создания Web-сайтов / М. Кузнецов, И. Симдянов. - М.: БХВ-Петербург, 2012. - 577 с.
7. Кузнецов, М. Объектно-ориентированное программирование на PHP / М. Кузнецов, И.

Учебное издание

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ
по дисциплине
«КОМПЬЮТЕРНЫЕ И ТЕЛЕКОММУНИКАЦИОННЫЕ ТЕХНОЛОГИИ В ПРОФЕССИОНАЛЬНОЙ ДЕЯТЕЛЬНОСТИ»
для студентов направления подготовки
Профессиональное обучение (по отраслям),
магистерская программа
«Информационные технологии и системы» (в 2 х частях). Часть 1

С о с т а в и т е л ь:
Тимошенко Дарья Сергеевна

Печатается в авторской редакции.
Компьютерная верстка и оригинал-макет автора.

Подписано в печать _____
Формат 60х841/16. Бумага типограф. Гарнитура Times
Печать офсетная. Усл. печ. л. _____. Уч.-изд. л. _____
Тираж 100 экз. Изд. №_____. Заказ №_____. Цена договорная.

Издательство Луганского государственного
университета имени Владимира Даля

Свидетельство о государственной регистрации издательства
МИ-СРГ ИД 000003 от 20 ноября 2015г.

Адрес издательства: 91034, г. Луганск, кв. Молодежный, 20а
Телефон: 8 (0642) 41-34-12, факс: 8 (0642) 41-31-60
E-mail: izdat.lguv.dal@gmail.com **http:** //izdat.dahluniver.ru/